2011

Anomaly-based Correlation of IDS Alarms

Tjhai, Gina C.

ANOMALY-BASED CORRELATION OF UDS ALARMS

GINA C. TJHAI

Ph.D. 2011

ANOMALY-BASED CORRELATION OF IDS ALARMS

G.C. Tjhai

Ph.D.
January 2011

ANOMALY-BASED CORRELATION OF IDS ALARMS

A thesis submitted to the University of Plymouth
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Gina C. Tjhai
January 2011

School of Computing and Mathematics
Faculty of Science and Technology
University of Plymouth, UK

Anomaly-Based Correlation of IDS Alarms

Gina C. Tjhai

Abstract

An Intrusion Detection System (IDS) is one of the major techniques for securing information systems and keeping pace with current and potential threats and vulnerabilities in computing systems. It is an indisputable fact that the art of detecting intrusions is still far from perfect, and IDSs tend to generate a large number of false IDS alarms. Hence human has to inevitably validate those alarms before any action can be taken. As IT infrastructure become larger and more complicated, the number of alarms that need to be reviewed can escalate rapidly, making this task very difficult to manage. The need for an automated correlation and reduction system is therefore very much evident. In addition, alarm correlation is valuable in providing the operators with a more condensed view of potential security issues within the network infrastructure.

The thesis embraces a comprehensive evaluation of the problem of false alarms and a proposal for an automated alarm correlation system. A critical analysis of existing alarm correlation systems is presented along with a description of the need for an enhanced correlation system. The study concludes that whilst a large number of works were carried out in improving correlation techniques, none of them were perfect. They either required an extensive level of domain knowledge from the human experts to effectively run the system or were unable to provide high level information of the false alerts for future tuning. The overall objective of the research has therefore been to establish an alarm correlation framework and system which enables the administrator to effectively group alerts from the same attack instance and subsequently reduce the volume of false alarms without the need of domain knowledge.

The achievement of this aim has comprised the proposal of an attribute-based approach, which is used as a foundation to systematically develop an unsupervised-based two-stage correlation technique. From this formation, a novel SOM K-Means Alarm Reduction Tool (SMART) architecture has been modelled as the framework from which time and attribute-based aggregation technique is offered. The thesis describes the design and features of the proposed architecture, focusing on the key components forming the underlying architecture, the alert attributes and the way they are processed and applied to correlate alerts. The architecture is strengthened by the development of a statistical tool, which offers a mean to perform results or alert analysis and comparison.

The main concepts of the novel architecture are validated through the implementation of a prototype system. A series of experiments were conducted to assess the effectiveness of SMART in reducing false alarms. This aimed to prove the viability of implementing the system in a practical environment and that the study has provided appropriate contribution to knowledge in this field.

Table of Contents

List of Tables

List of Figures

Acknowledgements

This thesis presents the results of three and half year research into numerous aspects of IDS alarm correlation, in the field of information security. All works were carried out within the Centre for Security, Communications and Network Research (CSCAN), at University of Plymouth, Plymouth, UK.

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

In the first place I would like to show my gratitude to Prof. Steven M. Furnell, my Director of Studies and Research Director of the CSCAN group, for his supervision and advice from the very early stage of this research until the completion of this thesis. His wide knowledge, understanding and support have been invaluable for me.

Secondly, I would like to thank Dr. Maria Papadaki, my second supervisor, without whose knowledge and assistance this study would not have been successful. The good supervision, personal guidance and support of hers have been of great value on both an academic and a personal level, for which I am extremely grateful.

I also wish to express my sincere thanks to my third supervisor, Dr. Nathan L. Clarke for his excellent advices and constructive comments that have been very valuable to this study. Without his kind support, it would be impossible to complete this research. I also thank Dr Bogdan Ghita for his help in capturing the network traffic and for his support until the completion of this thesis.

I am indebted to many of my fellow researchers for their support, and I would like to extend my warmest thanks to those who have helped me with my work in CSCAN Group at the University of Plymouth. Special thanks go to my brother; Dr. Cen Jung Tjhai, without whose personal advice and assistance, I would not have finished this study. Of course, thanks must also go to all my friends in Plymouth and Indonesia for their endless support.

Lastly, and most importantly, I am grateful to my parents, brothers and sister for their unlimited support and patience. Without them, this work would never come into existence.

For those who have contributed but not mentioned, please accept my thanks.

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Graduate Committee.

This study was financed with the aid of scholarship from Faculty of Science and Technology, University of Plymouth, Plymouth, United Kingdom.

Relevant seminars and conferences were regularly attended at which work was often presented. Several papers were published in the course of this research project, details of which are listed in the appendices.

Word count of main body of thesis: 46,951 words

Signed : .....

Date : 09-02-2011

1 Introduction

The information security threat landscape is changing rapidly; many organisations are struggling to keep up with the changes of nature, complexity and scale of attacks or intrusions. In its 2010 security threat report, Sophos has highlighted six major IT security threats, including social networking, data loss and encryption, web threats, email threats, spam and malware (Sophos, 2010). One prominent example is the rise of social networking threats, with a 70% rise in the proportion of firms that report encountering spam and malware via social networks during 2009 (Sophos, 2010). It is also predicted that social networking sites will face more sophisticated threats as the number of users grows (McAfee, 2010). In general, intrusion is defined as

> any actions or attempts made to compromise the integrity, the confidentiality or availability of a resource (Heady et al., 1990).

According to the report presented by Symantec (2010), there are a number of recent and growing trends in the threat activity landscape that were observed by Symantec (2010) in 2009. The trends include attacks targeted on enterprise are increasing, with web application continuing to be the favoured vector of attacks over the Internet.

In discovering the security threat landscape, there are two fundamental issues that have been affected by the information security domain over the years (Kark, 2010). Firstly, the threat landscape keeps evolving and gains sophistication and secondly, the attackers will always be a step ahead of the defenders in exploiting the vulnerabilities in the domain of people, processes and technologies. Given these issues, attention should now be directed to focus on the changing nature of the threat landscape that includes the motivation of the attackers, the attacking methods and tools. The attacking techniques have also evolved from a simple and visible attack to a more sophisticated or stealth attack. In terms of the tools, the process has moved from manual to more automated techniques that are easy enough to implement that even people with minimal technical knowledge can use them effectively (Symantec, 2010).

Changes in the current threat landscape (in other words, the complexity, the evolution of attackers and attack patterns) show not only the attack sophistication (the trend of threats) but also the need for enhanced security mechanisms to combat these changes. In fact, the use of authentication, cryptography, firewalls and antivirus systems do not provide a complete solution to secure information system. In spite of those security tools, one of the most apparent network tools being developed, and which has continuously grown in popularity, is the Intrusion Detection System (IDS) (Bace, 2000).

This chapter presents a brief introduction to the context of this research by presenting an overview of the main problems related to the subject of study. Next, the goals and objectives of the research are determined, followed by a brief summary of each chapter.

1.1 The Intrusion Detection System

An Intrusion Detection System (IDS) is essentially a burglar alarm or a security tool, which is aimed at detecting attacks against computer systems and network. IDS, much like the security industry itself, has grown rapidly over the past two decades (Goeldenitz, 2002). It has become one of the most vital components of a defensive tool protecting computer system and networks from abuse.

The Intrusion Detection System is required in today's computing because it is unfeasible to keep pace with current and potential threats and vulnerabilities in computing systems. The environment is continuously evolving and changing motivated by new technology and the Internet. Intrusion detection products help in protecting a company from intrusion and securing its information. This security appliance is used to detect an intruder, identify and block the intruder, support investigations to find out how the incident occurs and stop the possibility of future exploits. Such a measure should be applied across the enterprise and could serve as a very powerful tool in the information security practitioner's tool kit.

Owing to this fact, the existence of intrusion detection has acted as a second line of defense by monitoring, detecting or even responding to the unauthorised activities which could bypass the firewall system. It is worth remembering that IDS is not a silver bullet when it comes to protecting systems or network infrastructure. Instead, it is only one aspect of multi-layered protective mechanism, an approach referred to as 'defense in depth' (McHugh et al., 2000).

Although intrusion detection technology has been well established and been an active research for more than two decades, the art of detecting intrusion is still far from perfect. In fact, the system still suffers from the problem of false alarms, which is considered the major limiting factor for the performance of IDS (Axelsson, 2000). The following subsection briefly discusses the issue and its impact on the overall security implementation.

1.2 The Problem of False Alarms

While IDSs have been used for years and shown to be an invaluable improvement towards organisation's security, they endure the problem of high false alarm (alert)[†] rate (Khosravifar et al., 2009). The false alarm rate, which is also known as a false positive, is the frequency with which the IDS reports the malicious activities in error. This issue is aggravated by the fact that some commercial IDSs may generate thousands of alarms per day (El-Hajj et al., 2010). Recognising the real alarms from the significant volume of alarms is a frustrating task for security officers. False alerts always cause an additional workload for IT personnel, who must handle and verify every single alert generated to inhibit or block possible loss of data confidentiality, integrity and availability. The manual verification of these true and false alarms among the flood of alerts is not only deemed to be labour intensive but also error prone (Bolzoni et al., 2007).

The number of alerts generated by an IDS on a Local Area Network (LAN) could be very large, for example 15,000 alerts per day per sensor (Cuff, 2006). Reducing the false alarm rate is not an easy task. Indeed, it often worsens the situation by causing poorer IDS reliability or accuracy.

[†]The terms *alarm* and *alert* are used interchangeably throughout the thesis

Accordingly, further research is needed to devise a better approach to reducing false alarms while improving the quality of alerts generated.

Traditional IDSs raise alerts independently, though there may be logical links between them. A successful intrusion could trigger a sequence of alerts that correspond to different stages of the attack (Ning et al., 2002). Thus, identifying alerts related to an intrusion could help construct the attack scenarios. In fact, knowing the real attack patterns and the tactics used by the criminals to launch the attacks enable the network administrators to take appropriate actions to block or prevent them from escalating. Therefore, in order to support the security administrators in the analysis of the security incidents and to provide them with a comprehensive view of the events, an alert correlation technique is introduced. Such a technique has become well-liked and commonly studied in current IDS research (the literature of the existing techniques is presented in Chapter 3. It provides a mean to find the causal relationships in data by associating alerts, which are parts of linked chains of events. Thus, it is anticipated that the method will enable the administrators to discover the general attacks pattern from raw intrusion alerts, manage large volume of intrusion alerts and help reducing false alarms.

1.3 Aims and Objectives of the Research

This study focuses on the issue of an automated alarm correlation system and specifically the selection of the features used in the alert mapping processes, enabling the design and evaluation of a novel prototype system for an automated alert correlation and filtering method.

The main objective of this research is to evaluate and design an improved alarm reduction and correlation method. This encompasses the following stages:

1. Investigation of the extent of the false alarm issue and its impact on IDS detection performance.

2. Critically review existing alarm correlation approaches. This was achieved by investigating the use of AI for the improvement of false alarm handling.

3. Identify factors that influence the alert correlation process. It is expected that the appropriate alarm correlation method to particular IDSs will vary depending on the features of alarms generated. The suitability of such a method on a particular detection system relies on its ability to properly and effectively interpret the features of alarms generated by the IDS. Not only is this approach anticipated to aggregate alerts from the same event, but also to distinguish between true and false alerts.

4. Propose a novel architectural framework and a new approach to an unsupervised automated alarm correlation system. The framework will be designed focusing on its alert mapping processes.

5. Evaluate the effectiveness of the architecture by designing and implementing a prototype system. The main objective of developing a prototype system is to facilitate the practical evaluation of the proposed unsupervised automated alarm correlation method.

The objectives presented above relate to the general sequence of the material presented in this thesis, the structure of which will be discussed in the next session.

1.4 Thesis Structure

Chapter 2 presents the evolution and basic concepts of intrusion detection, by introducing the IDS architecture and responsibilities of each IDS component, followed by the fundamental principles of the technology. This is followed by an introduction to the taxonomy of IDS and quality parameters of the technology. It also highlights five key measurement criteria that determine the proficiency of the intrusion detection technologies. The chapter then provides an overview of the IDS alarm generation and the conventional alarm reduction method, aiming to underline the main concepts of traditional tuning methods.

Chapter 3 focuses on the existing studies on alarm clustering and reduction and begins with an introduction of the basic concepts of alarm correlation. It then continues with a review of several existing correlation approaches, highlighting their strengths and weaknesses in tackling false alarms. The chapter concludes with a discussion of the Security Information and Event Management (SIEM) tool, a significant example of a correlation system that collects all security-related information generated by software running on the network to provide a more condensed view of intrusive activities. This chapter highlights the advantages of the product and also indicating the requirement of further improvements on alarm correlation system based on the concept SIEM.

Chapter 4 assesses the extent of the problem of false alarms based on experiments involving the popular open source network IDS, Snort (Caswell and Roesch, 1998). A number of potential issues are presented along with the IDS performance analysis on both the synthesised 1999 DARPA evaluation data set and real network traffic (University of Plymouth private data set). This chapter then continues to explore the issue of false alarm generation and critically scrutinise the impact of false alarms on the IDS detection rate.

Chapter 5 presents the main contribution of this research, an automatic unsupervised alarm correlation system. The aim of developing this system is to process alerts generated by signature-based IDS, including aggregating and correlating alerts associated with the same attack instance and clustering the alerts into groups of true and false alarms. The new correlation approach should help the IDS filter the unnecessary alerts and provide a more concise and high level view of occurring or attempted intrusions. Hence, the automatic alarm correlation and filtering system is developed using unsupervised techniques, namely Self Organising Map (SOM) and K–Means algorithm. The chapter begins by introducing the concepts of the applied methodologies and the rationale behind their implementation. The effectiveness of the proposed system is tested in preliminary experiments on both 1999 DARPA data set and University of Plymouth private data.

Chapter 6 extends the research by presenting a novel architectural framework of an automated unsupervised alarm correlation system. The principal focus of the research is then presented, briefly describing the main components of the proposed architecture, followed by the underlying concepts of the system. Initially, the operational characteristics of the SMART (SOM K–Means Alarm Reduction Tool) system, as a novel approach to existing research in the domain of automated alarm correlation is described, followed by a detailed presentation of its main modules.

Chapter 7 defines the implementation of a prototype system, which embodies a subset of the key elements of the proposed architecture, describing the interactions or relationships among them. Initially, the chapter begins with an overview of the system development process, the software applications used to develop the system as well as its system/software requirements. In addition, example scenarios are provided, in order to demonstrate how the correlation is run, how a graphical chart can be created from the correlation results and a false signature rule can be further scrutinised. Subsequently, a user-friendly I/O interface is defined, highlighting its features and its role in processing and presenting IDS alerts. The chapter continues to discuss the implications of the practical evaluation, underlining the prospective changes to the system architecture. Finally, the chapter concludes with a comparison on the overall correlations results (false alarms reduction) from DARPA 1999 and private data sets against the results of false alarm evaluation discussed previously in Chapter 4.

Finally, Chapter 8 presents the main conclusions from this research, highlighting the principle achievements and limitations of the work, along with suggestions for potential further enhancement.

The thesis also includes a number of appendices, which contain a variety of additional information in support of the main discussion, including a number of published papers from the research project.

2 Rationale of IDS Technology

Although Intrusion Detection technology has existed for three decades, it still remains the subject of active research area. So far, a large number of research studies have been conducted to improve the effectiveness of IDS technology as an essential part of organisational security infrastructure. In spite of its improvement, there is always significant scope for further enhancement. This chapter will begin by exploring the fundamental principle and evolution of IDS technology in its research domain; followed by the underlying problem of IDS alerts. Having presented a brief description of the false alarms, this chapter will continue to describe a traditional alarm reduction method.

2.1 Principles of Intrusion Detection

In terms of its components, IDS technology consists of three main aspects; namely a sensor (analysis engine), an event generator and a response module. In order to have a better understanding about the IDS' components, Figure 2.1 depicts a simple example an of IDS architecture (Lundin and Jonsson, 2002).

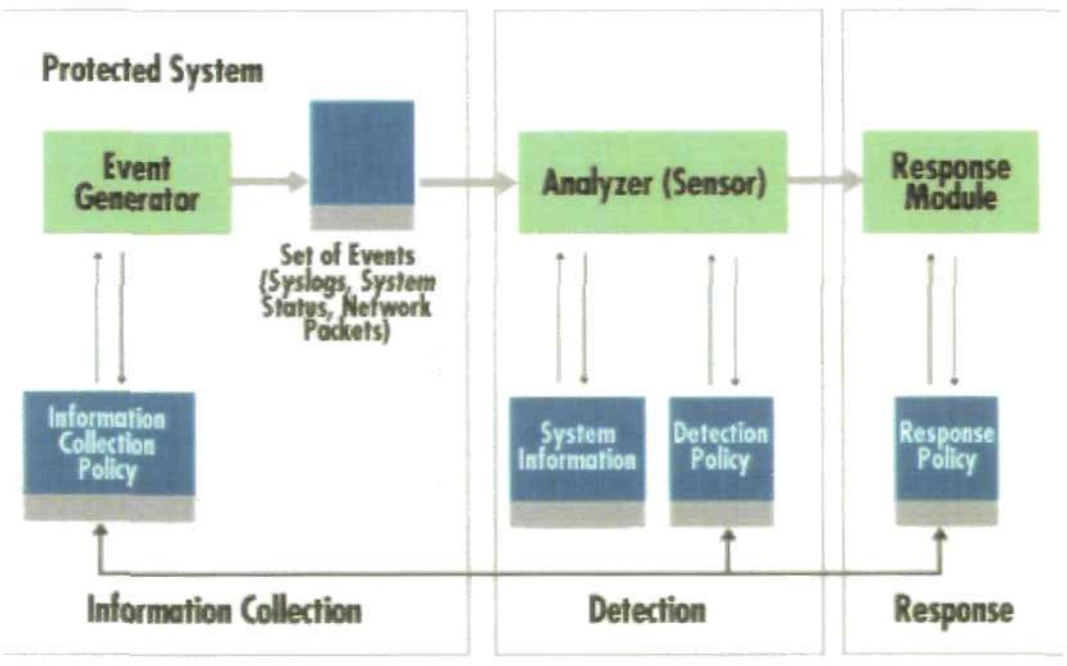


Figure 2.1: IDS architecture (Lundin and Jonsson, 2002)

The sensor, the core element of IDS, is responsible for filtering information and discarding any irrelevant data obtained from the event collector, thus detecting suspicious activities. This component, which comprises a decision-making mechanism, obtains raw data from 3 major informa-

tion sources namely IDS detection policy database (knowledge base), system information and audit trails. A host-based IDS will rely on system information and audit trails as its major information sources to detect potential intrusions. By receiving those 3 types of information, therefore, the sensor (analyser) is expected to utilise the data received as the basis for further decision-making process.

As a main information source of the IDS, the policy database does not only comprise the attack signatures or user behaviour profile, but it also holds the IDS configuration parameters, including the communication method with the response modules. The database will determine how the detection result should be presented to the end users (passive IDS) or whether further action is required to stop the attack (active IDS). Moreover, in order to effectively detect the intrusions, the sensor has its own database containing the details of potential complex intrusions (multistep attacks). The information enables the sensor not only to discover malicious activity but also to link two or more activities into a single attack (complex attack).

In order to enable the system to collect data from the network packets, the sensor is integrated with one component responsible for data collection, known as event generator. The role of the event generator is to produce a policy-consistent set of events that may be an audit (log) of system activities or network packets. Owing to its position as the event collector, this component could be an operating system, network or application depending on the type of IDS. In order to properly perform its task as data collector, the event generator will receive information from the information collection policy; containing the rules of filtering event notification information.

On the other hand, the response module, the last component of IDS, is responsible for taking an appropriate response action regarding the detection result. This component is known as the extension of the traditional IDS. More specifically, an IDS that comes with a response module is recognised as an Intrusion Prevention System (IPS). The actions taken by the IPS involve for example alerting the administrator about the intrusion via email or even a more active response; reconfiguring the firewall settings to inhibit the attacks. In most cases, the response module is known as the user interface. In this instance, the response module facilitates the user to view the output from the system or even to monitor the behaviour of the system (Grandison and Terzi, 2007).

IDSs are classified in many different ways, including sensor-based and detection model.

2.1.1 Sensor-based

Based on the location of its sensor, the IDS can be divided into 3 types, namely network, host and hybrid IDS.

Network Intrusion Detection Systems (NIDS), such as Snort, CISCO Secure IDS and Clog, are IDSs that aim to detect malevolent activities such as denial of service attack, port scanning or even any suspicious activity or attempt made to crack into the protected system, by monitoring the network traffic (Herberlain et al., 1990). Due to its operational characteristic, the NIDS has become one of the most popular types of IDS in the current market (Scarfone and Mell, 2007). The cost-efficient (only one IDS required to monitor all devices or hosts in a network) and easy deployment of the system (Operating System independent, in other words, the implementation of IDS does not affect existing systems or network infrastructure) are few of its main advantages (Carter, 2002). Unlike the network-based sensor which performs a packet level analysis, a host-based IDS (HIDS), such as

OSSEC, Tripwire and Symantec Critical System Protection (Timberline Technologies, 2009), monitors system level activities and event log consolidation. With the ability to access all system calls and application logs, the HIDS sensor can detect any improper change of event activities as soon as it is executed, thus determining the occurrence of an attack with greater accuracy and lesser false alerts than network-based IDS.

Given the advantages of both network and host-based IDS, it is more beneficial that the IDS is developed by combining multiple different IDS technologies, for example, network-based and host-based IDSs, into a single system or also known as Hybrid IDS. The examples of which are RealSecure, Cisco Security Monitoring Analysis and Response System (MARS), NetIQ's Security Manager and OSSIM (Open Source Security Information Management) (CISCO, 2010). The hybrid IDS is commonly created based on a model which brings multiple agents of multiple types such network-based IDS, host-based IDS, network packet capture and filtering, for example, TCPdump, and other multi-vendor IDS systems; all of which are integrated and analysed by the centralised management console (Bashah et al., 2005). The advanced hybrid system is now known as Security Information and Event Management (SIEM) (Zoho, 2007). The aim of developing the hybrid technology is to bring more flexibility, expandability and greater accuracy of alerts by cross-checking anomalies against other systems; hence reducing the rate of false alerts.

2.1.2 Detection Model

Based on its detection method (triggering mechanism), IDS can be classified into 2 common types, namely misuse detection and anomaly detection.

Misuse detection is also referred to as signature-based detection. The idea behind the concept of misuse detection is the application of a signature pattern that represents an attack's behaviour or even the variations of the attack. Misuse detection IDS, such as Snort IDS, offers various benefits including the ability to detect known intrusive activity and to provide detailed information about the attack the IDS is programmed to alert on. Despite the benefits offered, it has a problem in maintaining the state of information for signatures. In order to maintain the IDS detection performance, the signature database must define all possible attacks that an attacker may launch against the network. This, therefore, requires a frequent signature updates to keep the knowledge base up-to-date (Bon, 2005).

On the contrary, an anomaly detection system (also known as statistical-based IDS) detects an intrusive activity by monitoring the traffic or system activities and classifying it as either anomalous or normal. One example of this IDS is SPADE (Statistical Packet of Anomaly Detection Engine) (SecurityFocus, 2010). An anomaly detection system can effectively detect insider attacks and unknown (novel) attacks since it merely relies on the normal behaviour profile and no prior knowledge of attacks is required. Such method is opposed to the signature-based IDS which can only identify known attacks for which signatures have been created previously. Having said that, the biggest drawback of anomaly detection is the issue of false alarms. The unusual behaviour of a legitimate user, which is not defined in the user profile, could possibly raise the issue of false alarm. Therefore, selecting an appropriate threshold level to achieve a balance between the rates of false alerts (false positives) and missed attacks (false negatives) is a challenge (Garcia-Teodoro et al., 2009).

2.2 The Evolution of IDS

Intrusion detection technology has evolved significantly over the past few decades. Modern IDS has grown into a mature and feature rich technology that provides advanced features to detect intrusion, provide responses and also the management system that allows the security analyst to monitor, configure and analyse the intrusion data. In order to gain a better understanding of how intrusion detection has evolved, the following description presents several faces of intrusion detection technology.

- **Primordial Intrusion Detection Technology - Simple Pattern Matching**

  The very first Intrusion Detection System (IDS) relied on operating system log files as a main data source and run on a critical server to detect an intrusion (Kumar and Spafford, 1994). The systems applied a simple pattern matching approach on the incoming logs. In order to perform the pattern matching, a table of patterns (signatures), representing known intrusive tactics was used to match with the analysed patterns (usually presented as ASCII strings or string fragments).

  The earliest Network Intrusion Detection System (NIDS) applying this simple approach was then introduced (Herberlain et al., 1990). The system performed a comparison on every packet passing over a network with a list known attack strings. Each string would be compared byte-by-byte with all of the traffic monitored by the sensor. Although this system was easy to implement, it did not scale well. A rise in the number of patterns and data sources requires an exponential increase in processing power.

- **Protocol Awareness**

  The next level of the technology was the application of the knowledge of packets to the network traffic. The effectiveness of the IDS heavily hinged on the knowledge of packets and protocol standards in identifying malicious behaviours. The ability to decode the protocol header certainly allowed improvements in the pattern matching technique. Not only could the pattern matching be directed to focus on the appropriate part of a packet, for example, packet header or payload, other enhancements could be done in what traffic was monitored.

  Over time, attackers became more sophisticated. Packet fragmentation was one of the cunning techniques used to evade the IDS (Carlo, 2003). The attackers could break each packet into smaller pieces to avoid the detection since the IDS was designed to merely detect a complete pattern. In this case, no attack was seen by the IDS. The target network would then reassembly the packets and be successfully compromised. In order to address this issue, fragmentation reassembly was added to the IDS (Song et al., 1999). Every monitored network packet would be retained, reassembled and then evaluated to look for the suspicious patterns. This allowed the sensor to search for the potential fragmented packets effectively.

- **Understanding Network Sessions**

  Shifting beyond a single packet analysis, IDS technology developed a better approach to counter session-based attacks that occurred in a form of a dialog between two systems and would not be held in a single packet. In order to effectively uncover this type of attacks,

stream reassembly was introduced to IDS (Necker et al., 2002). This method enabled the IDS to observe the complete exchange between a source and destination instead of a small slice of the exchange and fully review it for malicious activity.

- **Full Protocol Analysis**

  With the improvements made on the IDS detection approaches (as described above), the IDS technology moved to the next level. The next development of the IDS technology was the implementation of the full protocol analysis (in other words, the application of specific knowledge of protocols) (Baba and Matsuda, 2004). This technique is proved effective to not only detect known bad behaviours but also to flag anomalous behaviour as suspicious, discovering new attack tactics even before they are announced.

  Intrusion detection based on protocol analysis was the earlier version of an anomaly-based IDS. Unlike the basic pattern matching technique, which detects an intrusion based on a matching string, the protocol-based evaluation identifies the intrusive behaviours using the pre-defined policy, thus enabling the system to detect variant of the attacks. As the full protocol analysis uses knowledge of protocol to identify how the packets would be interpreted by the destinations, most variations of the attacks can be identified via one mechanism. In addition, the attackers can easily evade the detection by simply creating a variation of the same string if only the pattern matching technique is applied in the detection system. Another benefit of advanced protocol analysis is that it can be applied to anticipate attack patterns. Any attempt made to send an attack against the pre-defined protocol can be detected as an anomaly.

- **Intrusion Prevention System (IPS)**

  The primary achievement of IDS development is the successful migration of IDS from a passive monitoring system to an active prevention system, which is known as Intrusion Prevention System (IPS). IPS or IDPS (Intrusion Detection and Prevention System) is considered the extension of IDS, since both are designed to monitor network traffic or system activity for malevolent activity (Enterasys, 2010). Unlike IDS, IPS is able to actively prevent or block detected event by taking appropriate actions, such as dropping malicious packets, sending an alarm to the administrator or resetting the connection to stop/block the ongoing attack.

- **Security Information and Event Management**

  Despite the success of the established IPS, the latest realisation of IDS development is the introduction of Security Information and Event Management (SIEM) system, an analysis centre that will combine all IDS outputs, network traffic analysis and other information to provide a more condensed picture of adversarial activity (Smith, 2006). It is actually a hybrid solution from two distinct security-related products, Security Information Management (SIM) and Security Event Management (SEM) systems. Unlike IDS/IPS, which relies on a single source of information to flag potential breaches, SIEM is capable of assessing log data and correlating information coming from various sources. The system can provide a mean to detect security-related events in two distinct ways: by offering a real-time detection or evaluation of security-related information directed to it (inherited from SEM) and by supporting non-real-time forensic analysis of consolidated log record collected (archive) from various disparate security measures (such as packet filter, IDS/IPS, servers) (inherited from SIM).

2.3 Efficiency of IDS

As to the success or failure of IDS technology, the effectiveness of the IDS event analysis largely hinges on the qualities of its detection. This involves three main metrics or parameters that determine the proficiency of intrusion detection technology (Porras and Valdes, 1998).

1. **Accuracy**. This parameter can be quantitatively measured by looking at the rate of false alarms generated by the system. Inaccuracy is shown in the number of legitimate transactions being flagged as intrusive activities (false alarms).

2. **Performance**. The rate at which transaction or audit events are processed determines the value of IDS performance. The lower number of packets dropped by IDS, the better the performance it has.

3. **Completeness**. Completeness refers to the ability of an IDS to detect all attacks. Incompleteness is reflected when IDS fails to raise an alert when malicious activities actually occur (false negative). Unlike the "accuracy" parameter, which can be measured based on the false positive rate; the "completeness" is assessed based on IDS detection rate.

Aside from these main characteristics determining overall IDS' detection performance, there are two other properties that have also reflected the additional values of IDS consistency (Debar, 2000). Those parameters are fault tolerance and timeliness. Fault tolerance indicates IDS' resistance to attacks or its immunity to any attempted damage made to break down the detection system itself. This is particularly important because this attribute determines the reliability of IDS technology as a security defense tool. The second attribute, timeliness, refers to the response time required by the IDS to disseminate and react to the information received. A good IDS should not only provide a fast processing speed of the information but also enable the security-conscious administrator to promptly act in response to the detected intrusion before further damage has been made (Porras and Valdes, 1998).

2.4 IDS Alarm Generation

Today, there exist a number of deployment issues faced by current IDS. One of the most significant problems facing this technology nowadays is the level of false alarms. The issue of false positive has become the major limiting factor for the performance of an IDS (Axelsson, 2000). The sheer number of alarms triggered by an IDS can be overwhelming. It is believed that a high rate of false alarms is a significant challenge for current network intrusion detection systems, which could possibly trigger 80-90% of fake alarms from the total alerts generated (Julisch, 2001; Laskov, 2007). In order to gain a better perception about the issue of IDS alarms, the following subsections discuss two critical problems of IDS alert generation.

2.4.1 False Negative

False negative is a real or genuine attack that is undetected by IDS. Ultimately this is the most dangerous type of error could possibly faced by IDS. Principally, false negative is difficult to compute since no evidence can be found from the IDS when the error occurs. Nevertheless, a proper

implementation of network defense in depth strategy can help to keep false negatives at a minimum. Significantly, the following are several prominent causes of false negatives (missed alerts) defined by Cox and Gerg (2004).

- **Traffic Encryption**

  Encrypted traffic is often used to perform a secure web communication and it is commonly applied to deliver confidential information. Encrypted data does not raise alerts because the signature rules do not match. As explained in the previous section, the limitation of NIDS in interpreting encrypted traffic becomes one of the major causes of false negative. Having said that, modern IDSs are now able to decode the encrypted traffic as long as the encryption keys are given. One of the research studies has been conducted to work on a NIDS that makes use of IPsec ("Two-Key IPsec") to decrypt the traffic before being processed by the IDS (McLain et al., 2007).

- **IDS Evasion Techniques**

  The ability of the attackers to evade the detection could be one of the main causes of the false negatives. Blackhats, security researchers and security developers are now competing with each other when it comes to network-based IDS. Blackhats community continually develops techniques to evade IDS sensors, whilst security vendors defeat these methods by releasing new patches. Examples of common IDS evasion techniques are fragmentation attacks, session splicing and basic string matching weakness (Ptacek and Newsham, 1998).

- **Badly Written Signatures**

  Too specific signatures could also be the factor of false negatives. This signature might not watch for the correct attack since a variant of an attack might not be detected by the same rule as its ancestor (Koziol, 2003).

- ***Poor Change Management***

  Poor management within an organisation could possibly result in a poor organisational security posture. Without proper security management, a malicious alteration made on the organisational security infrastructure (environment) may go unnoticed by network security defense (IDS).

- **IDS Sensor Administration Problem**

  The issue of false negative could possibly be engendered by the problem fine tuning. A thorough knowledge of the protected system is required by a qualified IT staff. Indeed, a detailed examination of the environment, in-depth knowledge of attacking techniques and awareness of new vulnerabilities or threats are required before tuning can be carried out. A careless tuning of IDS signatures to control the number of false positives might indeed render the system to miss a real security event (Amoroso, 1999).

2.4.2 False Positive

A false positive is an alert raised by the IDS because the system has reported malicious activity in error. False positive errors will lead users of the intrusion detection system to ignore its output,

as it flags legitimate actions as intrusions; thus leading to an actual intrusion being detected yet ignored by the users. Indeed, this error is the major bane of a security administrator's assistance. The occurrences of this type of error should be diminished (it may not be possible to completely eliminate them) in order to provide useful information to the system administrators.

Although IDS has been used for more than a decade, current detection systems still suffer from high false alarms and low detection rate (Bolzoni, 2009). An ideal detection system is the one that has 0% false positive rate with 100% attack detection rate. In actual fact, both signature-based and anomaly-based systems can be duped to raising thousands of alerts. Usually, with anomaly-based detectors the abnormality is determined by calculating the distinction between the monitored behaviour and the model. Indeed, a selected threshold is used as the benchmark for behaviour classification. The value of threshold has a direct impact on both false positives and false negatives. Increasing the threshold does tighten the system security but it is likely to provoke more false alarms. Conversely, the lower the value of threshold, the lesser the number of alarms generated, but this might result in a high number of false negatives. Similar to anomaly-based IDS, the more specific the rule set of signature-based IDS, the stronger the security that can be achieved, but this will lead to a higher false alarm rate.

In order to explore more about the issue of false positives, there are several common causes of false alarms discussed here (Pietraszek and Tanner, 2005).

- **Runtime Limitation**

  In practice, it is reasonably hard to distinguish between a real intrusion and normal activity. A detailed knowledge of the protected system and a thorough investigation are required to accurately isolate malicious events from normal traffic. It is worth remembering that IDS cannot analyse the contexts of all activities in details (Ptacek and Newsham, 1998). Owing to this limitation, detection system is relatively prone to the false positive issue.

- **Specificity of attack signatures**

  Writing a good signature rule for IDS is not a straightforward task, it is a challenging job instead (Paxson, 1999). In many cases, a right balance between a very specific signature and an overly general one is very difficult to determine. An overly specific signature is very prone to cause the system to miss a real attack (false negative). In contrast, an overly general signature is likely to induce a large number of false positive alarms.

- **Dependency on environment**

  An activity that is normal in one environment might be harmful in certain situations. For example, performing a network scanning is a malicious act unless it is carried out by someone authorised such as a network administrator. Hence, a thoughtless investigation of this instance might render the system to produce a large volume of false positives.

The accuracy of intrusion detection largely depends on its detection rate and false alarm rate. More to the point, the inaccuracy of the alerts generated can be classified into two forms, namely (Debar and Wespi, 2001):

1. *Intrinsic* inaccuracy

This inaccuracy occurred as a result of poorly written rule. As such it does not differentiate well between normal and harmful activity for a particular attack.

2. *Relative* inaccuracy

   This inaccuracy happened owing to the resemblance of normal activity (from the monitored system) to those of malicious activities.

Both aspects need to be taken into account in calculating the best value of detection system. In fact, based on the analysis of Axelsson (2000), it is believed that the substantial values of Bayesian detection rate (Intrusion/Alarm) will be achieved if the detection system has attained a very low false alarm rate. Unfortunately, according to the 1998 DARPA off-line Intrusion Detection Evaluation, the false alarm rate of the best IDS is not satisfactory (Lippmann et al., 2000).

2.5 Conventional Alarm Reduction Methods

In order to achieve the best performance of an intrusion detection system, the issue of false alarms has to be tackled. One of the best ways to reduce the false alarm rate is by performing a tuning. Fine tuning is a process of adapting the signature policy to the specific environment and modifying or disabling the signatures to reduce false alarms (Chapple, 2003). This is also driven by the fact that some vulnerabilities exist in a particular OS (Operating System) platform only.

In practice, there exist several common techniques applied to the tuning process, namely:

2.5.1 Disabling Signature

Disabling signature is carried out by deactivating the signatures relative to the vulnerabilities that are not present in a given environment. This method is usually applied to signature-based detectors. Some signatures can be switched off as the monitored systems are not exposed to certain vulnerabilities or the vulnerability itself only affects certain Operating System platforms. These irrelevant signatures are likely to trigger false alarms if the monitored packets happen to match one of the signature rules.

2.5.2 Pass Rule

This type of tuning technique can only be applied to a signature-based IDS, which largely hinges on the pattern matching for detecting an intrusion. In spite of alert rule, pass rule can be used to ignore alerts from certain hosts, networks and rules (Roesch, 1999). A poorly written pass rule can cause all the signatures to be missed, making the IDS sensor futile. Ignoring alerts from port 21 for example, might render the actual attack go unnoticed.

2.5.3 Thresholding and Suppression

Thresholding and suppression are the most effective tuning techniques, which enable the administrator to handle the number of logged alerts for noisy rule, generated from or to a given host or for a

particular signature. Instead of controlling the alerts per signature, this technique also introduces global thresholds; enabling the administrator to control the amount of alerts for all rules.

Principally, thresholding commands regulate the number of times a particular event is logged during a pre-defined time interval. Significantly, threshold rules come in three categories (Beale and Caswell, 2004), namely:

- *Limit*

  Alerts on the first n events during the time interval, and then ignores events for the rest of the time interval.

- *Threshold*

  Alerts every n times once this event is seen during the time interval.

- *Both*

  Alerts once per time interval after seeing n occurrences of the event and then ignores any additional events during the time interval.

Additionally, thresholding can be included as part of IDS rule signature (rule key format) or even written as a standalone command (standalone format). Although there is no functional difference between the rule key and the standalone format, there is a logical difference between them. Some rules can only work with the thresholds. For example, a rule for detecting brute force attack requires a threshold of 5 attempts before the alert can be triggered.

In order to maximise the performance of an IDS, there are 3 methods of tuning IDS sensors (Raikar and Ramarao, 2007). Those are:

1. *Tuning based on the deployment of the sensor*

   If the sensors are deployed to monitor the external traffic, then to reduce the number of false positives, the sensors must be finely tuned based on the perimeter defense firewall. This aims to alert only those attacks that have the potential to penetrate the firewall.

2. *Tuning based on the information about the protected system*

   In order to effectively protect the monitored system, the IDS rules must be tuned based on the system's vulnerabilities, such as what application or operating systems are the attacks intended to affect. This must be accurately mapped with a superset of all the information from the protected system.

3. *Tuning based on observing production environment monitoring*

   Apart from tuning the sensors based on its deployment objectives and the information of the protected system, alerts from the sensors need to be monitored in a production environment for a certain period. Additionally, an extra tuning of the attack signature should also be carried out. If the alerts are found to be generated from the normal traffic, then further tuning should be conducted for example by disabling the signatures or lowering the severity of the alerts.

Although tuning does offer a good solution in reducing a large number false alarms, this procedure could possibly exacerbate the situation by degrading the security level and increasing the

risk of missing noteworthy incidents. Therefore, the tuning problem is actually a trade-off between reducing false alarms and maintaining the security level. Furthermore, tuning requires a thorough examination of the environment by qualified IT personnel and requires a frequent updating to keep up with the flow of new vulnerabilities or threats discovered.

2.6 Conclusions

This chapter initially presents the concept and evolution of IDS technology in its research domain, followed by the fundamental principles of the technology. The components of the IDS technology are briefly described in this stage; by introducing the IDS architecture as well as the responsibility of each IDS component. Lastly, the issue of false alarms was briefly highlighted by including a discussion about two critical problems of IDS alert generations and the facts relative to these issues. And to address such problems, a conventional alarm reduction method (that is fine tuning) was introduced.

Despite the fact that IDS technology has grown into a mature product, the problem of false alarms is still far from being solved. Although tuning does offer a good solution in reducing a large number false alarms, this procedure increases the risk of missing noteworthy incidents. In fact, it requires a thorough examination of the environment by qualified IT personnel and requires a frequent updating to keep up with the flow of new vulnerabilities or threats discovered. With this issue in mind, it is of paramount importance to focus on an intelligent alarm reduction tool that could improve the quality of alerts generated by effectively filtering the false alarms without the need of human intervention. The following chapter will present a review of current literature or studies on IDS alarm correlation methods as well as the drawbacks associated with them.

3 Investigation into alert reduction methods using alarm clustering and correlation techniques

Apart from tuning, research has recently focused on alternative alarm reduction techniques. This chapter covers the state of the art specifically in the area of IDS alarm correlation using Artificial Intelligence (AI) techniques. It begins with an introduction of the concepts of alarm correlation approaches, highlighting the need for correlation systems. Existing works on false alarm reduction approaches are then presented, followed by the characteristics of correlation systems proposed in current IDS research.

3.1 The Need for an Automated Alarm Correlation

It is very common that an attack is performed by sending a significant number of malicious packets to the targeted network. As a signature-based IDS triggers an alarm for each detected malicious packet, alarm flooding may occur. Additionally, many attacks are launched as a sequence of steps (that is multi-stage attacks), which depict the logical relationship (in other words, cause and effect) between each stage of the attacks. Whilst each attack step can be identified by the IDS, it is worthwhile for the security administrators to obtain information about the detected attack based on the aggregation of alarms related to different steps rather than on each single alarm. Moreover, a single alert analysis provides only partial information on the attack, which is deemed not valuable enough to uncover the real patterns or scenario of the attack.

In order to allow the administrators to perform a complete alert analysis on the aggregated alerts and to cope with the issue of false alarms, an alarm correlation system is now a necessity. In general, alarm correlation is a process that analyses the intrusion alerts generated by IDS, filters the false alarms and then provides a more concise and high level view of occurring or attempted intrusions.

3.2 Concept of Alarm Correlation

In terms of their underlying concepts, alarm correlation methods can be classified into several prominent classes. Those are:

- **Correlating alerts based on the prerequisites of intrusions**

  This approach is based on the assumption that most intrusions are not isolated, but are related to the different stages of attack sequences, with the early one prepared for the later one.

It is believed that most of the traditional IDSs only focus on low level attacks and raise alerts independently, without considering the possible logical connection between them or the potential attack strategies behind them. Another problem is that they cannot fully detect unknown attacks, or the variation of known attacks, without generating a large volume of alerts.

Several studies, including works from Cuppens and Miege (2002) and Ning et al. (2002) were based on this concept. The authors proposed a novel system that correlated alarms by using the prerequisites and consequences of corresponding attacks, for example, the existence of a vulnerable service can serve as the prerequisite for the remote buffer overflow attacks. Furthermore, such an approach provides an intuitive mechanism to represent potential attack scenarios, known as hyper alert correlation. Even though this technique helps removing insignificant alerts and discovering a sequence of attack plans, it cannot correlate unknown attacks (without attack patterns).

- **Alert correlation based on the similarity between alert features**

  This approach correlates alerts based on the similarities of selected features, for example source IP address, destination IP address or port number (Debar and Wespi, 2001). Alerts with a higher value of overall feature similarity will be correlated. Another research study were also conducted in evaluating the use of a feature similarity function to fuse alerts that match closely but not perfectly (Valdes and Skinner, 2001). The similarity function was used to calculate the likeness of the features that match at least the minimum similarity specification, regardless of the match on the feature set as a whole. Several works from Julisch (2001); Al-Mamory and Zhang (2009); Maggi et al. (2009) were also based on this approach. Although such method seems to effectively reduce false alarms, it cannot fully discover the causal relationship between related alerts.

- **Alert correlation based on known attack scenarios**

  The last approach correlates alerts based on the known attack scenario. One of the methods to fuse alerts into a scenario is by using a data mining technique (Dain and Cunningham, 2001). This technique can produce a real time algorithm to combine the alerts produced by heterogeneous IDSs into a scenario. The main purpose of this work is to simply group alerts which share a common cause, thus providing a better view of the security issue to the system administrator. Such approach works well in reducing the false alarms, since either individual alerts or the whole scenario could be labelled as false alarm. Significantly, it could also effectively uncover the causal relationship between alerts; however, it could not be applied to correlate alerts generated by unknown attack scenarios.

It is worth remembering that the process of correlating alerts does not only involve a single or few components of procedure, instead it is a complete process involving various or a comprehensive set of components.

A study was carried out to propose a general correlation model that identified a comprehensive set of components and a framework that analysed how each component contributes to the overall goal of the correlation (Valeur et al., 2004). Figure 3.1 shows a graphical representation of the integrated correlation process. The main purpose of this process is to gain a better understanding of the features of the intrusions, for example, the alerts generated, the target and source host of

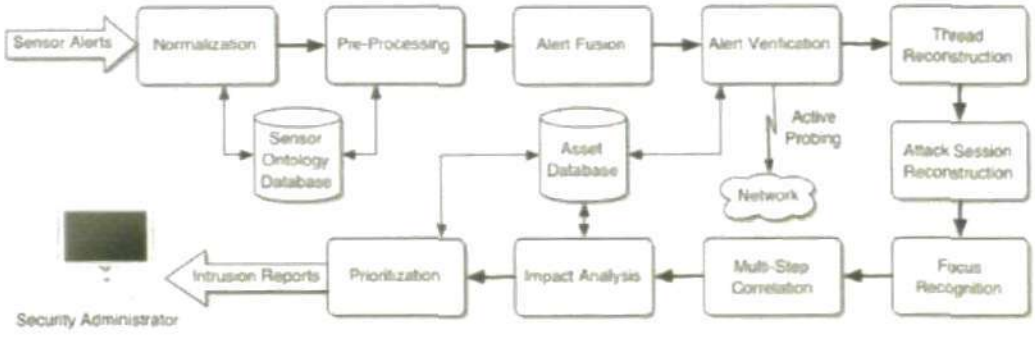


Figure 3.1: Correlation process overview (Valeur et al., 2004)

the attacks, the impact of attacks as well as the priority of alerts. Although numerous correlation methods were proposed in recent studies, not all of them provided a detailed account of how these components should be evaluated and implemented in a real life environment. For example, the identification of alarm patterns into scenarios using prerequisites and consequences features do not provide enough detail on how the incoming alerts will be pre-processed before being correlated into scenarios.

3.3 Alarm Reduction Methods

In order to better understand the roles of correlation systems in IDS technology, this section presents the development or techniques of existing false alarm reduction models.

3.3.1 Categories of Alarm Reduction Approaches

One of the main objectives of performing alarm correlation is to reduce false alarms. Alarm reduction is not a trivial task as numerous aspects, for example, attack features, need to be considered (Kruegel and Robertson, 2004). In addition to the conventional tuning method, two reduction approaches have been proposed so far, namely alarm classification and alarm correlation.

3.3.1.1 Alarm classification

Alarm classification is a process of identifying true and false alerts based on a pre-defined alarm model or pattern. This technique is commonly used to detect abnormal pattern from an alarm sequence using a pre-defined normal alarm model (Law and Kwok, 2004; Alharby and Imai, 2005).

3.3.1.2 Alarm correlation

Alert correlation is known to be the first step of the alert management processes. The main objective of the alert correlation technique is to reduce redundant alerts while keeping the important information. Besides, it could also provide a more succinct or high level view of security issues occurring in the protected network. This technique is designed to discover the logical connections (causal relationships) between alerts and also structural relationship in data by grouping/aggregating alerts

with common features (their similarity) (Ning et al., 2004; Julisch, 2001). Alarm correlation is also used to manage large volumes of alerts generated by heterogeneous IDSs (Zurutuza and Uribeetxeberria, 2004).

Moreover, knowing the sequence and outcome of an attack does not only help filtering false alarms but also enable more appropriate responses to stop and prevent attacks from escalating. This is another primary goal of alert correlation.

Having identified two common alarm reduction approaches, the following subsections describe how both techniques are applied in current IDS research.

3.3.2 Alarm Classification Techniques

One of the reasons that causes IDS technology to generate a high false positive rate is the lack of correlation between input and output traffic, which can be essentially used to look for abnormal output traffic (Bolzoni and Etalle, 2006). The main concept behind this study was the idea that a successful intrusion to a system usually generates an anomaly in the outgoing traffic; otherwise it is a normal activity. The proposed system, which was known as APHRODITE, consisted of two main components, namely Output Anomaly Detector (OAD) and correlation engine (Bolzoni and Etalle, 2006). OAD had a responsibility for monitoring the output of the system and by referring to a statistical model describing the normal output, flagging any behaviour that deviates from the pre-defined model as a possible attack. On the other hand, the correlation engine was responsible for correlating the input to the output of the system belonging to a same connection.

APHRODITE had various advantages in terms of operational factors. It worked effectively without optimal training (without using attack-free traffic) and could successfully detect an unknown attack without the need for a new signature. In addition, it was also proved to effectively reduce up to 99% of false positives generated by Snort. Despite the benefits offered, the system was unable to reduce the number of redundant alerts produced by the same event, and not able to conduct a real-time inspection, since the output of the event was required as the prerequisite of the detection process.

A post-processing filter based on the concept of neighbouring alerts and high alert frequency was presented by Spathoulas and Katsikas (2010). The proposed filtering scheme was developed according to two major assumptions; first, the distribution of the number of the neighbouring alerts varies significantly from false to true positives. Second, it is more probable for an alert to be a true positive if it occurs in higher frequency compared to the mean frequency of alerts from the same signature. There were three main components implemented in this proposed system, namely the Neighbouring Related Alerts (NRA) component, the High Alert Frequency (HAF) component and the Usual False Positives (UFP) component. For each alert received, each component should give a score (belief) which represented the probability that such alert was a true positive. The total scores from the three components were then calculated and a final verdict was produced to determine whether the alert is a false or true positive. Evaluation carried out using the DARPA 1999 data set pointed out that such approach can significantly reduce the false positives by 75%. However, one major weakness is that there is still no clear explanation been given to justify how an alert caused by a stealth attack can be detected as a true positive as such intrusion does not fire a large number of alerts, that is low frequency, in the network.

Data mining technique has been commonly applied by numerous research studies; one of them was proposed by (Law and Kwok, 2004). The authors suggested a novel system that modelled normal alarm patterns and detected anomaly from incoming alarm streams using a K-Nearest-Neighbour (KNN) classifier. The system monitored and detected abnormal patterns (in other words, suspicious events), from tones of alert generated by an IDS. It was believed that when an attack occurs, the alerts will have different patterns from the one generated in an attack-free environment. The main idea of the study was to let the false alarms be generated as they are, and then to determine whether the incoming alarm sequence generated are deviated from normal situations.

Although this model successfully reduced up to 93% of false alarms while maintaining its detection rate, it was not applied on live data and implemented in the real life environment. For that reason, there is still much more work to be undertaken in order to assert that this idea is applicable to existing IDSs under real life environment.

Another research based on the similar concept was proposed by Alharby and Imai (2005). By observing the frequent behaviours within an extended period of time, it was believed that a normal alarm pattern could be accurately formed. Based on the created model, the system could flag a sudden burst of a sequence of alarms that has never been seen before as a suspicious activity.

Given that the historical alarms pattern was used to learn the future alarms by using the extraction of the sequential pattern, this approach overcame some limitations of existing systems by constructing a more systematic model. The proposed system matched the extracted newly arrived sequence pattern with the extracted sequential pattern that represented the normal behaviours. The more matches found in this process, the more likely it is a normal behaviour.

Like other proposed methods, the system has a major drawback. The classification accuracy of both approaches from Law and Kwok and Alharby and Imai relied on the length of the time window for each alarm set. Since the alarm patterns are varied depending on the allocated time frame, the anomalous alarm pattern will share similarity with the normal pattern if the time allotment is amiss.

Unlike other works that focused on off-line analysis, Jan et al. (2009) proposed a decision support system for constructing alert classification behaviour patterns for on-line network behaviour monitoring through a large volume of alerts. The authors proposed three kinds of alert classification rule classes including normal behaviour, intrusion behaviour, and suspicious behaviour classification rule classes. Each class consisted of a fixed number of classification rules.

The experiment conducted showed the effectiveness of the proposed decision support system. Besides being able to run an on-line monitoring, the system also enabled domain experts to quickly and accurately discover suspicious behaviour patterns, ease the workload of on-line alert analysis for the administrators and effectively reduced up to 80% of false alerts. Having said that, the system also suffers from several common limitations. Firstly, in order to ensure that the alert sequences are properly flagged, the classification rules need to be frequently refined. This requires high computational overhead and sufficient domain knowledge from the experts. In fact, labelled data (rules) are not readily available in most cases. With a very large volume of network data encountered, it is undoubtedly expensive to classify them manually. Secondly, since the system can only support a limited number of rules (up to 200 rules) in each classification class, they need to be wisely selected to provide adequate coverage for whole attack variants. Lastly, the system appears to be not cost efficient enough since the rule classes are created for each target host (each sensor). A significant

number of rules will be required for IDS implemented on a multi hosts network environment.

Viinikka et al. (2009) presented a novel system that aggregated alerts into an alert sequence. There were two basic assumptions applied in this study: first, normal system behaviours in alert flow can be identified by the regularities and smooth changes in the alert intensity. Second, the normal behaviour is not observable at an individual alert but at alert sequence (flow). To follow these hypotheses, the system modelled the regularities of the alert flows from the normal behaviours and used the created model to filter out the irrelevant or low impact alarms from the alarm log. Although the system revealed a great performance, still, there is a risk of modelling abnormal behaviours into a normal behaviour model if no abrupt changes in the alert intensity caused by true alarms have been detected.

Aside from the data mining techniques, a novel system utilising machine learning technique, known as Adaptive Learner for Alert Classification (ALAC), was proposed by Pietraszek (2004). By building an alert classifier using such technique, the system would classify the alerts and send the classification outcome to the security analyst for further feedback. Once the feedback was received, the system would initially build and subsequently update the classifier, which was then used to classify new alerts in the future (as shown in Figure 3.2).

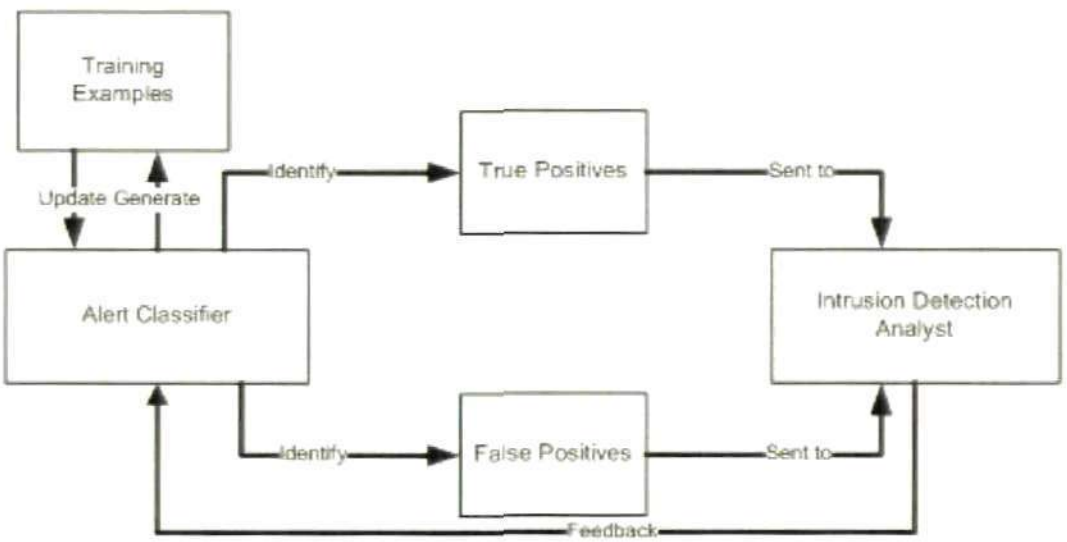


Figure 3.2: The framework model of ALAC classifier

The system offered a greater efficiency in terms of its operation. ALAC could be set to process autonomously alerts that were classified previously. For example, it could remove any alerts that have been classified as false positive in high confidence. Indeed, the experiment proved that the system could reduce the false alerts by more than 50%.

Along with the strengths, the system also has its share of drawbacks. One of the biggest problems is the dependency of the system on human intervention for a decision-making process. The ability of the analyst to correctly classify the alerts determines the accuracy or performance of the method. Another drawback is that the system must be able to adapt to the new changes as a new data arrives; in order to perform a real-time analysis. Besides, applying additional background knowledge, for example, network topology, alert database, to build an accurate alert classifier will

increase the complexity of learning tasks.

Another similar piece of study developing an adaptive learner detection system was proposed by Yu et al. (2008). The authors proposed an adaptive and automatically tuning intrusion detection system (ADAT) that controlled the number of alarms presented to the administrators and tuned the detection model according to the feedback provided by the administrators when false predictions were identified. The proposed system built a detection model in the training stage and controlled a prediction filter to push the most suspicious alarms (predictions) to be verified and labelled by the administrators. The volume of alerts presented must be restricted according to system operators' ability to respond to the predictions.

In terms of its performance, the system appeared to be very effective; with the total misclassification cost dropped at least 25.5% from the cost of systems with a fixed detection model. Unfortunately, one major common limitation of the system is that the system greatly relies on human intervention, in other words, security operators, in creating the detection model. The administrators are urged to verify intrusive predictions made by the prediction engine and to ensure any possible false predictions are identified. Additionally, it is very difficult to work out the optimal tuning strength (threshold) required in order to reduce the number of false predictions on future unknown data. This, hence, requires a thorough work or analysis to prevent the issue of false negatives.

3.3.3 Alarm Correlation Techniques

Julisch (2001) suggested a technique to efficiently handle large groups of redundant alerts by correlating alerts into clusters based on their root causes. The main purpose is to identify and remove the root causes of the false alarms. For each alarm cluster, a model of alarms, which is known as generalised alarm, is derived. Clearly, a generalised alarm is a pattern that an alarm must match in order to fit into a respective cluster. The knowledge of these generalised alarms vastly simplifies the identification of alarms' similarities. The author observed that over 90% of all alarms corresponded to a small number of root causes. By knowing the root causes, the IDS can be regularly adjusted and root causes can be removed, reducing the false alarms by 82%. Unfortunately, as such method focused merely on a large group of superfluous alarms, it was considered not effective enough in identifying false alarms in a small cluster. Julisch and Dacier (2002) also developed a technique to mine historical IDS alarms for episode rules. The rules are created to predict a prospective alert when a specific set of alarms was generated. Whilst this approach is deemed outstanding enough to give an insight into the pattern of false alarms and the potential future attacks, it could only offer 1% reduction in alarm rate, whilst 99% of alarms were still left for manual processing.

Al-Mamory and Zhang (2009) also proposed a novel system using a root cause analysis and clustering, which is extended from previous work done by Julisch (2001). The main contribution of this work was a development of a new data mining technique, Attribute-Oriented Induction (AOI), which has a new generalisation technique to avoid over-generalisation and has a good distance measure between the values of alarms' features. Although the experiments conducted with DARPA data sets and real network traffic clearly demonstrated the efficiency of the system, with a reduction ratio of 74%, it does suffer from the same limitation as Julisch's work. Extensive knowledge and experiences from the administrators are required to help creating a safe filtering rule in order to avoid the issue of false negative.

Another significant study conducted was the application of alert fusion to correlate alerts from multiple sensors in a distributed environment (Siraj and Vaughn, 2005). Alert fusion is a process of interpretation, combination and analysis of alerts to determine and provide a quantitative view of the status of the system being monitored. Importantly, the infrastructure consisted of three essential components; namely alert prioritisation, alert clustering and alert correlation. Thus, in order to fuse the alerts, a causal knowledge-based inference technique with Fuzzy Cognitive Modelling was implemented to find out the causal relationship in sensor data.

Overall, Fuzzy Cognitive modelling offers a good representation of data that enables the human operator to learn and interpret the data much easier. The technique also had an advantage in describing an attack scenario for Distributed Denial of Service Attacks (DDoS) by using the concept of "cause and effect". Besides, it also had a capability in discovering the causal relationship of alerts; which then could lead to the identification of a series of attacks. However, in spite of the advantages offered, it has one major limitation; namely its inability to deal with unknown alerts.

Another similar piece of work was also proposed by Maggi et al. (2009) to aggregate IDS alerts based on the concept of an alert fusion. The core components of the alert fusion process were aggregation and correlation modules. The first module was responsible for clustering alerts sharing common features, whilst the second one identified the logical relationships between alerts. The authors explored the concept of fuzzy set theory and fuzzy measure to semantically define the notion of "closeness" in time. The proposed technique was claimed to be simple yet robust approach for computing the time distance between alerts as it took into account major uncertainties on timestamps, in other words, the choice of window size is less sensitive due the concept of the applied fuzzy theory. Furthermore, it was able to decrease the false positives rate at the price of small reduction of the detection rate.

Similar to the previous study conducted by Siraj and Vaughn (2005); Perdisci et al. (2006) suggested a novel nearest neighbour-based on-line alarm clustering or fusion algorithm, which produced a unified description of attacks from alarms produced by multiple IDSs. The system comprised three architectural components, namely alert pre-processing module, classification module and clustering/fusion module. In the initial phase, an alert pre-processing module translated every alert into a standardised format that was understood by all correlation processes. On the other hand, a classification module labelled an alarm message as belonging to one or more attack classes. Finally, a clustering/fusion module determined whether the received alarm could be clustered, thus fused to the pre-defined clusters or to initialise a new meta alarm. Initially, the proposed system processed a sequence of alarms produced by IDSs and then produced meta-alarms, that is a summary description of events obtained by aggregating correlated alarms generated by various IDS sensors. The main objective of this new strategy was to provide a concise high-level description of the attacks and to reduce the volume of alarms presented to the administrators. Although the system was demonstrated to effectively reduce up to 80% of false alarms, it had a difficulty in obtaining sufficient attack data for the classifier to model and build the attack classes and the data acquisition is always time-consuming and greatly relied on the domain experts.

While several research studies have been focused on finding the relationships between alerts automatically, not much attention has been given to the issue of real-time correlation. Sadoddin and Ghorbani (2009) proposed a new framework for real-time alert correlation, which applied a novel technique for aggregating alerts into meaningful patterns and incremental mining of frequent

structured patterns. The authors presented several generalisation rules to improve the constructed pattern and generate an abstract signature for the patterns. A new algorithm, known as Frequent Structure Mining (FSP_Growth) was introduced to mine frequent pattern by considering the structure of the alerts. Such method not only offered more accurate frequency analysis of patterns but also gave the exact structure of the extracted pattern within the network. The core strength of the proposed framework lies on its ability to maintain time-sensitive statistical relationship between alerts in an efficient data structure and update the relationship incrementally to reflect the latest trend of patterns. The result of the experiments conducted with DARPA 2000 clearly demonstrated the effectiveness of the proposed technique. Approximately 96% of total alerts can be reduced effectively. Having said that, as the system included time lapse between alerts in the correlation, selecting a right (optimal) value to balance a security threshold is still a challenge.

3.4 Underlying Mechanisms of Alarm Correlation System

Unlike the comprehensive correlation framework proposed by Valeur et al. (2004), the majority of alert correlation studies deployed only a few major correlation components, involving only alert fusion and alert verification approaches, as described in Figure 3.1 (Siraj and Vaughn, 2005; Dondo et al., 2006). Alert merging (alert fusion) has a task of grouping alerts that represent independent detections of the same attack instance into a cluster. Each cluster is then passed onto the alert verification module (alert filtering), which is responsible for determining the success of an attack from the corresponding alert and filtering the insignificant alerts.

The correlation methods can be categorised into two different approaches, namely (Cuppens and Miege, 2002):

1. *Explicit Correlation*. Explicit correlation is a type of correlation that relies solely on intrusion knowledge of the security administrator to correlate alerts. The relationship of alerts can be discovered from its logical link based on the knowledge of the alerts, instead of the outcome of data mapping (Zhu and Ghorbani, 2006). Such correlation can express explicitly the known logical links between attacks. In addition, it forms and utilises correlation rules to define alerts' condition and the potential events generated from the corresponding intrusion.

2. *Implicit Correlation*. Implicit correlation is a correlation that depends on the analysis or computation of the alert data instead of the domain knowledge of the experts. By utilising the data mapping (either statistical or graphical data) produced by the generated events, the relations between alerts or events can be identified. The key objective of this approach is to investigate the behaviour of the alerts and extract the implicit connection between them. Many research studies were conducted to prove that IDS sensors can properly produce alerts based on the feature, traffic or the topology of information systems. And the correlation can be achieved by implementing learning techniques, such as machine learning, data mining and neural network.

In order to gain a better perception of the alert correlation model, following subsections present the evolving studies of both explicit and implicit methods in more details.

3.4.1 Explicit Alarm Correlation

A correlation method using consequence mechanism was proposed by Debar and Wespi (2001) and Ning et al. (2002). The proposed algorithm used the knowledge of prerequisite and consequence to group all possibly related alerts.

In this context, the prerequisite of an attack is the necessary condition for the attack to be successful. For example, the existence of vulnerable system is the main prerequisite for the attack to succeed. On the other hand, the consequence is the outcome of an attack. In a series of attacks where the attackers launch earlier attacks to set up the following attack, there exists strong connection between the consequences of the earlier attacks and the prerequisites of the later one. Moreover, the notion of hyper alert type is built to represent the prerequisite and consequence of each type of alert. Despite its benefits, this technique is deemed not effective enough to prevent the occurrence of false negative. For example, if a particular alert is generated by a specific attack that does not correspond to any other subsequent attack, it might not be correlated into a hyper alert. This alarm might be considered as a low priority alert that does not require full attention from security analyst. This situation might thus generate a false negative.

Apart from the consequences mechanisms, another novel system based on the concept of similarity between the alert features was proposed by Valdes and Skinner (2001). The system was created using a probabilistic method, which heavily relied on the parameters selected by human experts, for example, alert features. Owing to this fact, it is not suitable for fully discovering the causal relationship between alerts.

3.4.2 Implicit Alarm Correlation

Following subsections describe how data mining, machine learning and neural networks are applied in current IDS research.

3.4.2.1 Data Mining

Data mining, which is known as knowledge discovery, is a process of analysing data from different perspectives and summarising the valuable information from a large data set, for example, relational database (Zaiane, 1999).

Various different data mining techniques exist for cluster analysis and the suitability of the different approaches heavily depend on the area of their applications and features. One of the examples is to use data mining to look for alert clusters corresponding to root causes (Julisch, 2001; Julisch and Dacier, 2002); as described previously in subsection 3.3.3.

To achieve an effective data mining mechanism, a proposed system should satisfy several functional requirements, as described below (Julisch and Dacier, 2002):

- Scalability

  As the main task of data mining technique is to deal with a large data set, scalability has become its necessity. Scalability is a desirable property which indicates its ability to either handle growing amounts of data or to be readily enlarged.

- Noise Tolerance

Intrusion detection alarms can be very noisy (Paxson, 1999); thus the capability of filtering the noise from the real data is desirable.

- Multiple attribute types

  Alarms can be made of various types of data attributes such as numerical, categorical, time and free-text attributes (Julisch, 2001). An ideal data mining technique should support all attribute types.

- Ease of use

  The usability of data mining is of importance. Setting the parameters, for example, should not require an extensive and profound knowledge of data mining and statistics from the users.

- Interpretability and relevance of patterns

  Since the process of analysing the outcomes of data mining is iterative or it has to be repeated to keep up with changes of IDS alarm patterns, this feature become highly important. Otherwise, the human cost of learning from these patterns would become excessively expensive.

One of the fundamental techniques of data mining is associated with finding association rule. The concept of association (episode) rule has become well-accepted in the area of IDS research. Typically, episode rules are a data mining technique that was created to find patterns in event sequences (Shin et al., 2003). This method refers to a set of inference rules that predict the occurrence of an alarm based on the occurrence of other alarms. Indeed, it allows one to extract useful information from an unknown attack. Knowing the episodes of a legitimate activity makes the filtering of fake alarms effective, thus preventing false positives in the future. Similarly, if a number of redundant alarms have been discovered by episode rules, then overall alarm load can be reduced by fusing those duplicate alarms into a single, better meta-alarm. Ultimately, episodes that are generated from a real attack can be reliably applied to effectively detect future attacks.

Apart from using the theory of relationship to mine the historical alerts in a more inferential manner, a data mining technique can also be deployed for a virtual data mapping. It can be applied to map alerts data into a set of data points in order to provide a more descriptive view of the analysis. One prominent example of such application is the usage of KNN classifier to classify normal and abnormal IDS alerts (Law and Kwok, 2004). The KNN technique models the normal alerts patterns into an N-dimensional space. In fact, it was also commonly used in anomaly detection to observe the behaviours and to detect the intrusion from audit data (Li et al., 2007).

3.4.2.2 Machine Learning

Machine learning is a broad subfield of artificial intelligence, which is concerned with the design and development of algorithms and techniques that enable computers to learn (Bishop, 2007). Its major focus is similar to the data mining technique, to extract useful information automatically either by computational or statistical methods.

In the context of intrusion detection technology, machine learning can be used to store user-input stream of commands in a vectorial form and is implemented as model of normal user behaviour profile (Nilsson, 1996). Having developed the profile of normal behaviour patterns, they are then clustered in a group containing user commands with similar characteristics. Another significant

example of machine learning is the Adaptive Learner for Alert Classification (ALAC) (Pietraszek, 2004). It utilises the idea of labelled alerts to create the patterns or training examples for the input of machine learning.

3.4.2.3 Neural Networks

An artificial neural network is a broad subfield of artificial intelligence technique. It consists of collection of processing nodes that are highly interconnected and convert a set of input into a set of required output. The outcome of the alteration is determined by the nodes' attributes and the weights associated with the relationships or connections between them (Stergiou and Siganos, 1996). By adjusting the characteristics and connection between the elements, the network is able to adapt to the final outputs.

An increasing number of research studies so far investigated the application of neural networks to intrusion detection. If well designed and implemented, it has the potential to alleviate a number of problems encountered by other current approaches. Neural network is specifically created to learn the typical behaviours of actors in the system and to statistically recognise the significant variations from the established patterns (Bishop, 1995). The main advantage of using this approach is that it gives a simple method to express nonlinear relationships between parameters and learns the relationship automatically.

There are several typical neural network approaches applied in the area of intrusion detection. Further descriptions of those techniques are presented in the subsections below.

3.4.2.3.1 Self Organising Map (SOM)

The Self Organising Map (SOM), developed by Kohonen, is one of the most popular neural network models. It is a fully connected, single layer neural network (Kohonen, 1995). The SOM algorithm performs a smooth and linear mapping of a high-dimensional data set into 1-or 2-dimensional space. More to the point, it transforms non-linear statistical variables in a multi-dimensional map into geometrical connections between data points in a 2 dimensional space.

To date, the implementation of SOM algorithm to intrusion detection technology is prevalent. Ramadas et al. (2003) proposed an anomalous network traffic detector using the SOM algorithm. In this context, the SOM was trained by using the normal network. If the minimum distance between a network connection and the trained neurons exceeded the pre-defined threshold, the connection was flagged as an intrusion. This technique was also applied to perform the clustering of network traffic (Labib and Vemuri, 2002). It was implemented to plot network connections onto 2-dimensional maps, which were then presented to the network analyst. With the visual representation of data, such approach can effectively facilitate the detection of malicious network activities.

A work proposed by Rhodes et al. (2003) suggested the implementation of SOM as a network monitor stack, which utilised a protocol analyser to profile the network and to shrink or isolate the traffic before it was subjected to map analysis. The monitor stack was constructed at various layers of TCP/IP protocol stack. The proposed system monitored the activities at every layer of the monitoring stack since malicious attacks could target any protocol layer. Moreover, it was particularly tested to investigate one of the most well-known attacks, buffer overflow attack. Vectorisation scheme was adapted, consisting of a simple six-category histogram specifying the percentage of

bytes each packet fits a particular character class such as alphabetic, numeric, control and non ASCII.

An anomaly-based intrusion detection using self organising map approach was proposed by Lichodzijewski et al. (2002) and Vokorokos et al. (2006). Lichodzijewski et al. (2002) built an anomaly-based detection system by firstly identifying the characteristics of the normal connection to the target host using DARPA 1998 Intrusion Detection Evaluation data set. The proposed architecture comprised two levels, in which the first level is responsible for feature specification or detector for six basic TCP features, whereas the second level aims to combine the features identified by the six first level features into a single map. Vokorokos et al. (2006) suggested an anomaly IDS, which heavily relied on user behavioural patterns to distinguish between normal and abnormal behaviour. In order to properly model the user behaviours, the system log information was used as the main sources for the SOM networks.

Kayacik et al. (2007) proposed a novel NIDS based on a hierarchy self-organising feature maps (SOMs), which was evaluated on KDD-99 data set. An extensive analysis was conducted to assess the significance of features employed, partitioning of training data and the complexity of the architecture. The study also performed an evaluation to select the most significant basic features for IDS detection patterns. A two-layer SOM hierarchy system based on all 41 basic features from KDD data set was selected as the best detector.

The system offered excellent detection rate and false positive rate of 90.4% and 1.38% under test conditions. Having said that, SOM methodology was deployed on a new IDS solely to improve IDS detection rate, it was not implemented to enhance the quality of the alerts. Therefore, there was no alert correlation being proposed and none of the works were conducted to interpret the alerts.

Another novel SOM-based system using artificial immune system was presented by Powers and He (2008). The authors proposed a hybrid system, which combined both an anomaly detection component and a misuse component. The key concept was to deploy separate components for anomaly detection and attack classification. The idea behind this was to detect abnormal activity using anomaly-based detector and analyse examples of known attacks for common statistical patterns or features, thereby allowing the attacks with similar properties to be grouped into a cluster. A cluster centre should share common properties of many attacks by providing a higher level abstraction of attack patterns. After the cluster centres were created, an anomalous activity could be matched to the cluster that it was most similar to. By combining both anomaly and misuse detection, the proposed system could ensure that the known attacks were recognised and the novel attack patterns were detected. However, as the attack classification component classified the detected intrusions solely based on the known signature patterns, it is unclear how the detected novel attacks are grouped into the pre-defined clusters.

Apart from attack identification, SOM has also been widely applied to alarm clustering systems. A new clustering method, Improved Evolving Self Organising Maps (IESOM), was proposed by Xiao and Han (2006) to aggregate multiple alerts into attack scenarios. The proposed correlation system consisted of four modules, namely filtering, aggregation, condensing and combination modules. In the initial phase, the filtering component deleted alerts belonging to invalid value set, for example an alert with an invalid timestamp. Next, an aggregation component would group similar alerts together. The condensing component was responsible for discarding redundant alerts in each cluster while reserving only one alert per cluster. On the other hand, the combination component was in

charge of discovering an attack scenario based on the intrusive events. Finally, a visual attack scenario is given as the output of the system.

Notably, SOM has excellent capabilities for visualising high dimensional data onto one or two dimensional space. However, it has a major weakness in the context of alarm clustering. The number of neurons used in data mapping could affect the performance of the clustering. Therefore, it should be carefully selected in order to gain a better clustering result. Increasing the number of nodes implemented in the data mapping might increase the resolution of map.

3.4.2.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a set of unsupervised learning techniques used for pattern classification and nonlinear regression (Cristianini and Shawe-Taylor, 2000). It is deemed to be one of the most successful classification algorithms in neural networks and it is commonly applied in the area of intrusion detection. Zhu and Ghorbani (2006) described the implementation of alert correlation system using SVM. In this approach, the SVM was trained with a small number of patterns, which were manually generated and labeled. Indeed, it did not require probability correlation to be assigned to the training patterns, but only the class labels were assigned (in this case, 1 and -1). This appears to be one of the major advantages of using SVM. Moreover, its fast training speed has made it possible to incrementally update such approach in a real time environment.

Khan et al. (2007) presented a study for enhancing the training time of SVM, specifically when dealing with large data sets, using hierarchical clustering analysis. Hierarchical clustering is a method of cluster analysis that develops a hierarchy (tree like structure) of clusters to see the relationship among entities (Sambamoorthi, 2003). The clustering analysis was proposed to help find the boundary points, which were the most qualified data points to train SVM, between 2 groups. It was also used to generate support vectors to improve the accuracy and effectiveness of SVM classifier. Hence, by introducing a reduction technique to reduce the training data set using the clustering analysis, it was expected that the training process could be expedited.

3.4.2.3.3 Multilayer Perceptron (MLP)

One of the most widely used neural classifier today is the Multilayer Perceptron Network, a type of supervised neural network, which has been extensively investigated and for which various learning algorithm have been created (Seung, 2003). The MLP network is a flexible and nonlinear model comprising of a number of nodes arranged into multiple layers (Kanellopoulos et al., 1997).

Zhu and Ghorbani (2006) proposed an alert correlation engine using both MLP and SVM techniques, which has been discussed in subsection 3.4.2.3.2. In this work, the training data similar to that given to SVM approach was adapted to MLP network. The main objective of the proposed work was to verify the suitability of both correlation methods in correlating alerts and extracting attack strategies. The final outcome revealed that both approaches have their own strengths and weaknesses. MLP seems to have the potential to generate more precise correlation probabilities than SVM if the knowledge for assigning accurate probabilities to training data is available. On the other hand, labeling training examples for SVM is much easier since only two variables, 1 and -1, are used for the class labels.

Cannady (1998) proposed a hybrid model of SOM and MLP. The model was created to detect complicated and possibly collaborative attacks. Aside from alert correlation engine, MLP network

was also been deployed in keyword selection approach. Lippmann and Cunningham (2000) utilised back propagation technique as the learning algorithm for a classification function. The method was implemented to adapt the weights of the neural networks, and revealed a detection rate of 80% when it was tested against the DARPA 1998 evaluation data set. The study also described the implementation of MLP in anomaly detection system with a remarkable result of 77% detection rate and 2.2% false alarms.

3.5 Security Information and Event Management

Security Information and Event Management (SIEM) is a piece of software used on enterprise data networks to collect input logs and alerts from a variety of security systems such as Firewalls, Routers and Servers and attempt to interpret the collected events as well as inform the security operators of unusual occurrences (Miller et al., 2010). SIEM is relatively a new idea, pioneered a decade ago and still evolving rapidly as yet. More importantly, it has now grown into a very powerful security tool that obtains information from many systems at both network and application level, having a perception of security events and ability to access vulnerability databases, for example, system known weaknesses and their exploitation. SIEM may have also feature a reporting tool to assist the security analysts with an event investigation and a report production.

Many studies have focused on optimising security event interpretation via event correlation to improve the process of security investigation. Libeau (2008) defined three options of event analysis, namely log management (collection and storage of events in a repository), security information management (historical analysis of security events) and security event management (real-time analysis of security events). The process of event analysis can be summarised into a life cycle, which entails 7 sub-processes. Those are generation, collection, transport, real-time analysis, storage, reporting and forensics. These processes can be further simplified into 5 phases to reflect details of the investigation procedure. Those are data collection, normalisation, enrichment, correlation and report (360IS, 2010). Data collection refers to a process of receiving event logs and alerts from various security tools, whilst normalisation is a process of converting numerous data formats into a standardised version; for example common date format and address notation. In order to perform a comprehensive analysis, it is necessary to involve extra security information such as publicly known exploit data, inventory of enterprise and vulnerability scan results. The data are then grouped, prioritised using an event correlation method to flag the events and subsequently eliminate the false positives. Finally, a real-time event display and activity report can be performed for example by dispatching email notification, SMS, or other means.

The development of SIEM technology is driven by the need for improved security monitoring capabilities (Nicolett and Kavanagh, 2008). The main goals of its deployment are to primarily consolidate logs from different security systems into single events and to reduce false positives. The technology has become an established security product in the field of security management and is deemed to be one of the fastest growing security markets in 2007 (Nicolett and Kavanagh, 2008). Its market is motivated by the demand for a real-time security event analysis (for threat management on network events) and log analysis as well as report (for security compliance monitoring on host and application events). Numerous security organisations have funded the development of

such technology and commonly integrated it with various related security products. Examples of which are IDS, system management functions, event management and IT governance or risk and compliance management.

3.6 Conclusions

This chapter focused on the existing studies on false alarm reduction system using alarm classification and correlation methods. Numerous correlation methods are reviewed; highlighting their strengths and weaknesses in tackling false alarms.

The main purposes of deploying an alert correlation engine are to improve the quality of the generated alerts and to help automatically extracting attack strategies from a large volume of intrusion alerts. Based on the reviewed works, the key objectives of alarm correlation systems can be concluded as follows:

- to construct attack scenario by aggregating alerts related to the same attack (Ning et al., 2002; Debar and Wespi, 2001; Cuppens and Miege, 2002)
- to classify alerts into two classes (in other words, true and false alarms) (Maggi et al., 2009; Spathoulas and Katsikas, 2010)

In general, two main correlation methods have been applied in existing works. The first category is called explicit approach (that is knowledge-based correlation), in which the correlation relies solely on intrusion knowledge of the systems to correlate alerts. These methods are limited to the experience of the system itself and cannot correlate alerts of new attack. The second category is known as a learning-based correlation (that is implicit approach). Instead of relying on the domain knowledge of the experts or systems, this method depends on the analysis or the computation of the alert data. By utilising the data mapping (either statistical or graphical data) produced by the generated events, the relations between alerts or events can be identified. The key objective of this approach is to investigate the behaviour of the alerts and extract the implicit connection between them.

Although most proposed methods were proved to effectively reduce false alarms, none of them were perfect. They either required an extensive level of domain knowledge from the human experts to effectively run the system or were unable to provide high level information of the false alerts for future tuning. On the other hand, those, which were able to discover attack patterns from the aggregated alerts, did not have an ability to determine the validity of the alerts. The administrator was, therefore, left with a task of identifying the real and the false alarms.

In fact, the rule-based or explicit approaches (Julisch and Dacier, 2002; Cuppens and Miege, 2002; Ning et al., 2002; Sadoddin and Ghorbani, 2009; Jan et al., 2009) and the supervised algorithms (Law and Kwok, 2004; Alharby and Imai, 2005; Pietraszek, 2004; Zhu and Ghorbani, 2006; Viinikka et al., 2009) are not ideal approaches for alert correlation owing to the dynamic growth of network attacks. Such techniques require extensive knowledge of the administrators to create rules for defining any potential relationships between alerts. On the contrary, implicit correlation method using unsupervised approach (Kayacik et al., 2007; Powers and He, 2008; Lichodzijewski

et al., 2002; Xiao and Han, 2006) is considered a better technique than the rule-based method in creating an automated correlation engine. Such an approach applies a competitive learning algorithm, which does not require any human intervention to classify the alerts.

Given that the concepts of data mining, unsupervised neural network and machine learning techniques are so powerful and it is believed that they are holding the future of IDSs, it is worthwhile that future research study should be devoted to investigating these approaches in alarm correlation engine. The key objective of the research is to establish an alarm correlation framework and system which enables the administrator to effectively group alerts from the same attack instance and subsequently reduce the volume of false alarms without the need of domain knowledge (that is based on implicit approach). In fact, the purpose of SIEM technology has become another main objective of the proposed system. Unlike SIEM, which focuses on various security tools, the proposed engine collects alerts from a single tool, network intrusion detection system. Moreover, an ideal system should not only classify alerts as true or false alarms but also provide a mean to facilitate alert analysis for future tuning.

Finally, in order to better understand the fundamental issue of current IDS, the following chapter presents a series of experiments conducted to explore the extent of the problem of false alarms and to assess the effectiveness of tuning method and the impact of it on the IDS detection rate.

4 An Experimental Study of the Problem of False Alarms

After evaluating or reviewing the existing research on IDS alarm correlation and false alarm reduction methods, it is now essential to look at the main issue that has significantly highlighted the need for an automated false alarm reduction system. And in order to investigate the extent of the problem of false alarms faced by current IDS technology, a series of experiments were conducted in Snort (Caswell and Roesch, 1998), using the DARPA'99 data set as well as a private data set, before and after fine tuning Snort's signature set.

This chapter discusses the design of the experiments, and the rationale behind it. This is then followed by an analysis of the experimental results, the aim of which is to assess the impact of false alarms on the IDS detection rate.

4.1 Experiment Description

Prior to presenting the experimental results, this section provides a brief description of the experiments data set as well as the tools used to carry out the evaluation.

4.1.1 Experiment Data Set

A number of research efforts have been conducted to evaluate the performance of IDS in terms of its detection rate and false positive rate. One of the most well-known and determined IDS assessments to date was undertaken by Defense Advanced Research Projects Agency (DARPA) IDS evaluation (Lincoln Lab, 2010). This quantitative evaluation was performed by building a small network (test bed), which aimed to generate live background traffic similar to that on a government site connected to the Internet. The generated data set, which included a number of injected attacks at well defined points, were presented as tcpdump data, Basic Security Model (BSM), Windows NT audit data, process and File system information. The data were then used to evaluate the detection performance of signature-based as well as anomaly-based IDSs (Lippmann et al., 2000).

Although this data set appears to be one of the most preferred evaluation data sets used in IDS research and addressed some of the concerns raised in the IDS research community, it received in-depth criticisms on how this data was collected. The degree to which the stimulated background traffic is representative of real traffic is questionable, especially when it deals with the reservation about the value of the assessment made to explore the problem of the false alarm rate in real network traffic (McHugh, 2000). Significantly, Mahoney and Chan (2003) also critically discuss how this data can be further used to evaluate the performance of network anomaly detector. Although the DARPA dataset can help to evaluate the detection (true positive) performance on a network, it

is doubtful whether it can be used to evaluate false positive performance. In fact, the time span between the dataset creation and its application to the current research has resulted in another reservation about the degree to which the data is representative of modern traffic. However, despite all of these criticisms, the dataset still remains of interest and appears to be the largest publicly available benchmark for IDS researchers (McHugh, 2000). Moreover, it is also significant that an assessment of the DARPA dataset is carried out to further investigate the potential false alarms generated from this synthetic network traffic. It is expected that the result of this analysis could describe or provide a general picture of the false alert issue faced by the existing IDSs.

Given that DARPA dataset is deemed to be the largest publicly available benchmark and the baseline of many research (Thomas et al., 2008), the first experiment was designed to utilise such data as the source of the investigation. The primary data source of this evaluation was collected from 1999 DARPA IDS evaluation dataset. Without training the Snort IDS with the three weeks training data provided for DARPA off-line evaluation beforehand, two weeks testing data (fourth and fifth week of test data) were downloaded and tested against Snort IDS.

Although DARPA allows comparison with other research studies, it is still just synthesised traffic that was collected ten years ago. Owing to this issue, the second experiment involved the evaluation of Snort on a private data set, based on the collection of network traffic (100-150 MB/s network) to and from the University of Plymouth's web server over a period of 40 days, starting from May 17th to June 25th 2007. Technically, the data was collected by port mirroring the external interface of the UoP Internet connection. The capture was performed using a conventional network capturing tool, tcpdump, filtering for external requests to port 80 and the UoP extranet server IP address. The purpose of conducting an experiment on the University's private data set is to test IDS performance on a more recent and real life data set than DARPA. In fact, an evaluation, which is conducted merely on a synthetic data such as DARPA data set is not adequate enough to provide an insight into the issue of IDS implementation on a real life environment.

The experiments involved a process of identifying real and false alarms; before and after tuning was performed. The main purpose of comparing results with or without tuning is to assess the effectiveness of tuning IDS signatures in reducing false alarms and the impact of it on the overall IDS performance.

4.1.2 Experimental Tools

To carry out the experiments, there are three software applications are required, namely Snort (an open source network intrusion detection), Wireshark (a network protocol analyser) and BASE (a web front-end application that is used to query and analyse Snort IDS alerts). The following subsections briefly define these applications.

4.1.2.1 Snort

Snort is a lightweight Network-based IDS created by Caswell and Roesch (1998). It is a primarily a signature-based IDS that monitors network traffic in real time, examining each packet carefully to detect harmful payload or suspicious behaviours. Snort uses rules written in text files to capture suspicious data. It comes with a full set of pre-defined rules to detect intrusive activities and the

administrators are free to edit, disable the built-in rules or even to create new rules in an attempt to improve the Snort detection rate.

The reason for utilising Snort was due to its openness and public availability. Moreover, an investigation involving such a commonly used IDS can give an insight into the extent of the false alarm problem in other IDS systems as well.

Snort version 2.6 was selected as the main detector, whilst the Snort ruleset deployed in this evaluation is VRT Certified Rules for Snort v2.6 registered user release (released on 14 May 2007). And since the objective of this research is to explore the extent of false alarm problem on IDS detection rate, only the Snort's default configuration could be deployed in this evaluation; with all signature rules enabled.

4.1.2.2 Wireshark

Wireshark is a free open source GUI-based packet analyser for Windows and Unix (Wireshark, 2010). It uses pcap to capture data and is commonly used for network troubleshooting and packet analysis. The reason for using Wireshark as the analysis tool is its user-friendliness. Unlike some of the more complicated command-line driven tools like Tcpdump, Wireshark has a graphical front-end and features various sorting or filtering options that enables the users to interactively analyse and filter the content based on the different protocols, ports, and other data.

4.1.2.3 Basic Analysis and Security Engine (BASE)

BASE is a front-end tool for Snort IDS system and has been created based on previous Analysis Console for Intrusion Database (ACID) project (BASE, 2009). It provides a web-front end to perform an analysis of alerts coming from the Snort IDS system. This application processes database containing security incidents logged by IDS programs and presents the information to the users in a user-friendly web interface. In addition, the product also features a graph creation tool that allows the users to present the data in a graphical report. The reason of choosing BASE is because it is one of the well-known alert analysis tools that has been specifically developed for Snort IDS.

In this experiment, the Snort alert output was stored in a MySQL database and the front-end tool BASE was utilised as the intrusion analyst console. The investigation was accomplished by exhaustively examining every single alert that was generated by Snort. And to help analyse the alerts, the Wireshark was run to read the packet capture dump file and identify the network packets associated with the triggered alerts.

4.2 An Experiment using the 1999 DARPA Data Set

The first stage of the experiment was to run Snort in NIDS mode against the DARPA dataset. The manual validation and analysis of alerts produced by Snort were undertaken by matching against the Detection and Identification Scoring Truth. The Detection Scoring Truth is comprised of a list of all attack instances in the 1999 test data, while Identification Scoring Truth consists of alert entries of all attack instances in those test data (Lincoln Lab, 2010). A match is identified as same source or destination IP address, port numbers and their protocol type. In this case, timestamp does not really help identifying the true alerts since the attacks were labelled by the time the malicious

activities set off while Snort spotted them when malevolent packets occurred. This might render the system missing numerous matches. Hence, by recognising the matches for those attack instances, the number of false positives alarms will then be identified.

Once the alerts were manually verified and the false positives were isolated, the results were presented in several diagrams to give a clear picture on the issue of false alarms. In this experiment, the main purpose is to depict the severity of the false alert issue based on the type of signature raised. Given that Snort IDS enables the user to freely access the ruleset, this facilitates an investigation of the causes of false alarms generation. Hence, by examining each signature rule associated with the false alarms, this will provide an insight into the extent of false alarms issue and the impact of false alarms on the IDS detection rate.

This section presents the findings of the experiment. There were a total of 91,671 alerts, made up of 115 signature rules, generated by Snort in this experiment. In order to visualise the number of alerts, a pie chart is presented in Figure 4.1. Of the roughly 90,000 alerts generated from this dataset, 69% are false positives.

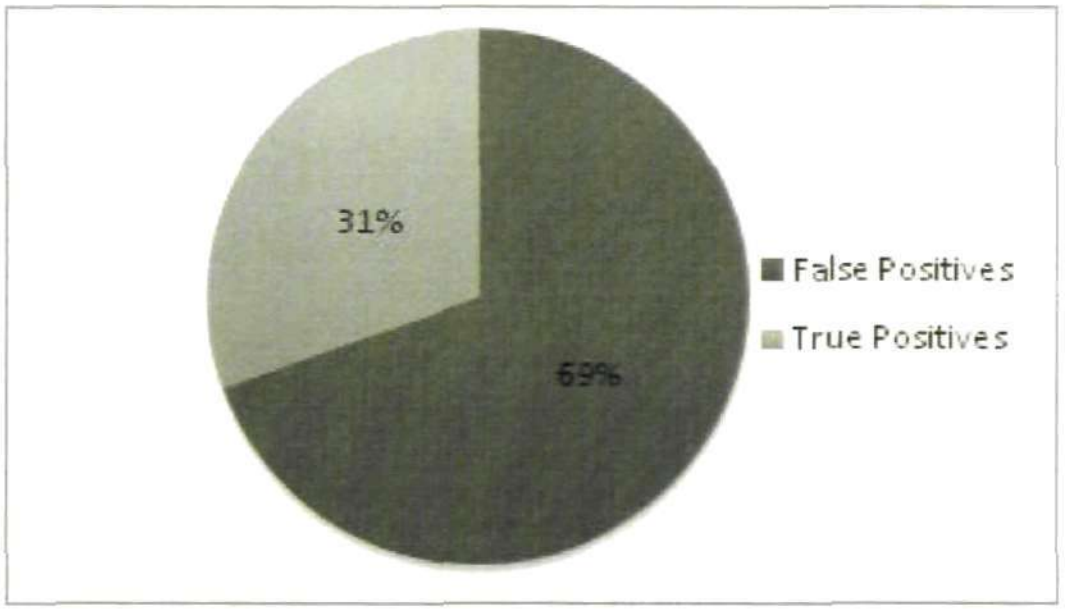


Figure 4.1: Percentage of true and false positive alerts on DARPA dataset

To gain a more in-depth understanding of the nature of Snort's alert generation, Figure 4.2 portrays a ROC (Receiver Operating Characteristic) plot for the overall result, which illustrates the overall alert generation of Snort's signature rule. The number of false positives is presented per signature for the X-axis, while true positive is portrayed on the Y-axis. This diagram also describes the random guess line (non-discriminatory line), which gives a point along a diagonal line from the left bottom (0, 0) to the top right corner (10, 10). This diagonal line divides the space into two domains; namely good and bad zones. Ideally, a good detection system should yield a point above the line, meaning the number of real alerts (true positives) triggered should not be exceeded by the number of false positives generated. The area below the line represents a higher number of false positives than true positives. Thus, the more plots are scattered on this area, the poorer the IDS is.

As the plot diagram can only give an overview of IDS alert generation, Figure 4.3 provides the

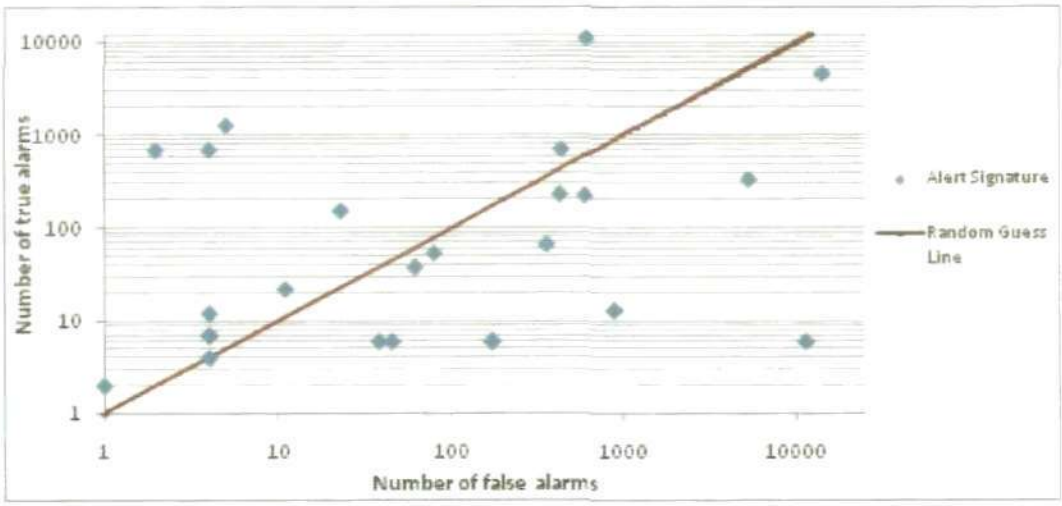


Figure 4.2: Overall alert generation per signature

exact figures of Snort's signatures generating the false and true positive alerts in a Venn diagram. Seventy three signatures raised the false positive alarms; of which 26 of them triggered both true and false positives. It is also worth noticing that of those 26 rules, 14 signatures had false positives outnumbering the true positives. This seems to be a very critical issue faced by contemporary IDSs. The following subsections discuss this issue in greater detail.

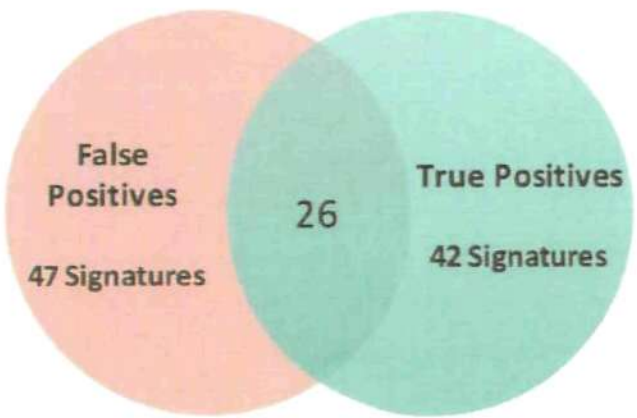


Figure 4.3: Snort IDS alarm - True and false positive Venn diagram

A complete list of true and false alarms as well as the attack types detected on this data set is presented in Appendix A.

4.2.1 True Positives

Given that the objective of this experiment is to investigate the issue of IDS false alarms, evaluating Snort's detection performance on DARPA dataset is beyond the scope of this study. Therefore, the extent of Snort's detection performance on a particular attack will not be further evaluated in a greater detail. However, this subsection presents a brief overview of the generation of Snort's true

alarms on 4 attack categories, namely probe, Denial of Services (DoS), Remote to Local (R2L) and User to Root (U2R).

Interestingly, about 72% of true positives were generated due to the probing activities. Generally, Snort fares well in detecting probe attacks, which largely generate noisy connections. In this study, it was found that Snort has a very low threshold for detecting probing activity; for example in detecting ICMP packets. This has made up of 40% (37,322 alerts) of the total alerts. In spite of its sensitivity, Snort generated a low level of true ICMP alarms in this experiment, which accounted for only 13% of those 37,322 alerts. This significantly highlights the underlying flaw of Snort IDS alarms.

In terms of the DoS attacks, Snort does not perform well. Only one attack, "Back" (a denial of service attack against the Apache web server) (Lincoln Lab, 2010), could be perfectly detected without generating any false positives. This has contributed to 20% of total true alarms. As for remote to local attacks, about 16 out of 20 types of attacks were detected. This, however, only made up of 2% of true alarms. Although Snort seems to fare well in this category, it critically missed numerous attack instances.

The last attack category, user to root (U2R), is the most challenging attack for Snort IDS. Since U2R attack typically occurs on a local machine, which attempts to elevate administrator's privileges, it relies solely on a system log or computer's file system. As such, Snort, a network-based IDS which merely depends on network connections, does not work well in detecting such attacks. Such attack could only be detected if it is launched by a remote machine to a local host across the network. In this case, only a small proportion of true alerts (less than 1%) were generated owing to this category.

4.2.2 False Positives

A large volume of alerts, largely comprised of false positives, were generated by Snort IDS. Approximately, 69% of total alarms are false positives. Figure 4.4 shows the top five false alarms raised by Snort. Interestingly, 48% of the total false alarms were made up of ICMP alerts. This explains one of the flaws of Snort IDS. As Snort has a very low threshold for ICMP traffic, logging every connection associated with probing, for example all ping activities, will only tend to generate a significant number of false positives. In fact, all detected ICMP traffic did not imply the occurrence of probing actions, but it was merely an informational event; indicating the occurrence of network outage. Thus, this concern drives the need to verify every single alert generated or even to improve the performance of IDS alarm reporting system.

In terms of the category of alerts generated, 39% (24,835 alerts) of the total false alerts were triggered due to policy violations. Significantly, this type of alerts is more related to irrelevant positives than false positives. Irrelevant positives refer to the alerts generated from unsuccessful attempts or unrelated vulnerability. However, as those informational alerts were not related to any suspicious activity from DARPA attack database and in order to make it simpler, they were flagged as false positives.

The highest number of false alarms in this experiment was triggered by INFO web bug 1x1 gif attempt signature. This signature rule was raised when the privacy policy violation was detected (Snort, 2010b). Theoretically, the web bug is a graphic on the web page, which is used to

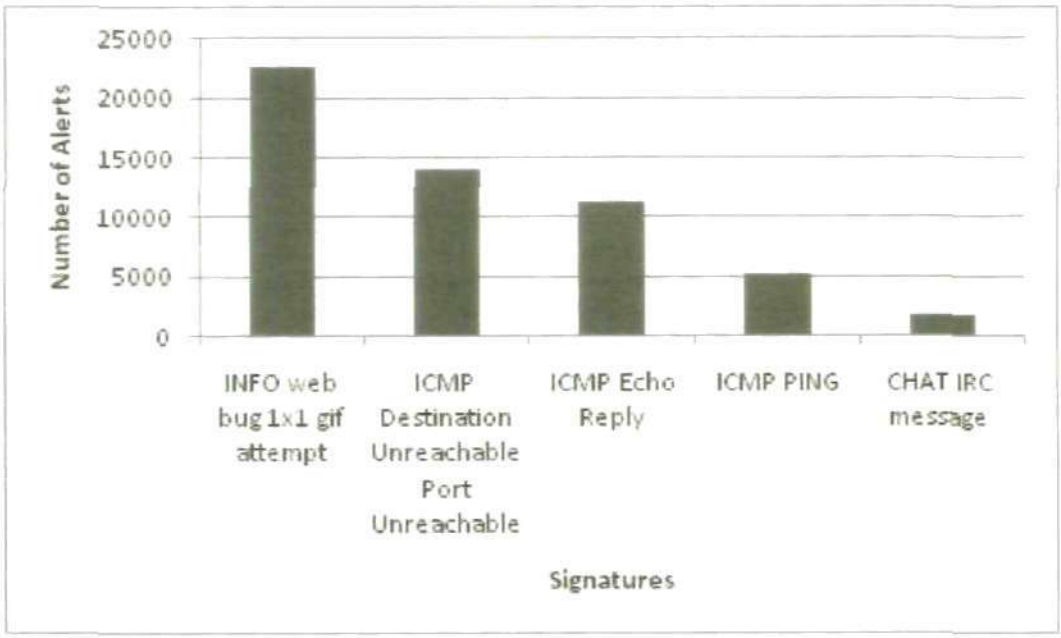


Figure 4.4: Top 5 DARPA false alarms

monitor users' behaviours. This is often invisible (typically only 1x1 pixel in size) and hidden to hide the fact that the surveillance is taking place (Smith, 1999). In fact, it is also possible to place web bug in a Word document as it allows html in a document or images to be downloaded from the external server. This is particularly useful if the document is supposed to be kept private, and web bug provides the information if the the document had leaked by finding out how many IP addresses had looked at it. Owing to its legitimate use and since none of these web bug alerts fitted in any attack instances described in DARPA attack database, the study reveals that no true alarms associated with this signature was generated. Therefore, total 22,559 alerts from this signature were entirely asserted as false positives. This contributed to 35% of total false alarms raised by the system.

Another similar policy-related alarms logged in this experiment is CHAT IRC alerts. These alerts accounted for 3.6% (2,276 alerts) of total false alarms. Snort generates these IRC alerts because the network chat clients have been detected. In common with the previous "web bug" signature, IRC alerts were not truly false positives. Principally, Snort, given the correct rule, fares well in detecting policy violation. Indeed, through the investigation of the DARPA packet payload, it was noticeable that the chat activity did take place on a certain time. However, since these alerts did not contribute to any attack instances in the attack list, they were considered as false positives. These CHAT IRC alerts were triggered by three signature rules; namely CHAT IRC message, CHAT IRC nick change and CHAT IRC channel join.

Interestingly enough, 25 web-related signatures triggered pure false positives. A signature is considered to generate pure false positive if no true alarm associated with this signature is generated. Although port 80 was one of the most vulnerable ports for DARPA attacks, these signatures did not correspond to any attack instances listed in the attack database. Aside from the web-related alerts, other 22 signatures, involving ICMP informational rule, policy, preprocessors, exploit at-

tempt and SQL rules, also generated a significant number of pure false positives in this evaluation. In view of that, all these alerts made up of 44% (28,340 alerts) of total false alarms raised by the system.

As described in the previous section, 14 signatures produced more false positives than true positives. This certainly becomes a good example, which highlights the critical issue of false alarms in the real world. If the false positives per signature highly outnumbered the true positives, this could undermine the process of identifying real attacks. In addition, this often renders the administrator less concerned about the alerts; thus tending to conclude them as false positives. This problem could seriously inhibit IDS detection performance in a real environment.

While Snort's performance looks sufficiently impressive by detecting 32 types of attacks, it produced a large volume of unnecessary alerts; for example the alerts triggered due to the detection of a DoS attack (that is "Back" attack), by WEB-MISC apache directory disclosure attempt signature. Only seven instances from this attack were included into the DARPA dataset, but surprisingly Snort detected all seven instances by triggering 5,628 alerts from single signature. Obviously, Snort has generated a significant number of redundant alerts in this case. Indeed, this often leaves the administrator with the difficulty of verifying every single alert logged by the system.

4.3 An Experiment using the University of Plymouth Data Set

The next phase of the experiments is to assess the problem of false alarm using University of Plymouth data set.

Although storing the full packet information significantly increased the storage requirements for the experiment, it was important to maintain this information for the validation and analysis of IDS alarms. It should also be noted that traffic containing web pages with the potential of having sensitive / confidential information was excluded from the packet capture, in order to preserve the privacy of web users. This was accomplished by parsing the packet trace using ngrep to ensure that no personal data (such as personal details on contact/feedback web pages) existed in the trace.

The first stage of the experiment was to run Snort in NIDS mode, in its default configuration. This means that no tuning whatsoever was conducted. The idea behind this is to investigate the extent of the problem of false alarms and to compare the effect that tuning can have on false alarm reduction. The next phase of the experiment involved the analysis of the same traffic, after tuning was performed on Snort. A number of techniques were applied for the tuning, including setting up the event thresholds and adjusting Snort's rules (Beale and Caswell, 2004). A necessary requirement for this was the manual validation and analysis of alerts produced by Snort in the first phase, and identification of signatures that are prone to false alarms. The analysis of IDS alerts was supervised by a certified intrusion analyst, and BASE was utilised to assist the intrusion analysis process

Once the alerts were manually verified, the result was presented in a ROC diagram; a graphical plot of Snort alarm generation. In order to reveal a clear picture of the false alarm problem, a ROC plot is preferable. Unfortunately, there were no true negative (number of benevolent activities passed) and false negative (number of real attacks missed) value known in this analysis since no

alarms were triggered due to these events. In fact, the only way to obtain these figures is via packet analysis. Unfortunately, no traffic analysis was conducted on the captured traffic to identify true and false negatives. As an alternative, the plot diagram is drawn to represent the actual number of false and true alarms instead of their alarms rate. This diagram provides a simple graphical representation of the false alarm problem, thus enabling the analyser to easily comprehend the performance of Snort IDS in a real network environment. By demonstrating the graphical plot of false positive versus true positive, this approach visibly explains the criticality of the false alarm issue. As the values of false and true negatives are unknown in this case, the alarm rate is calculated per total generated alarms instead of its total negative and positive values. The formula is presented as follows:

$$\text{False Alarm Rate} = \frac{\text{False Alarms}}{\text{Total Alarms}} \times 100$$

$$\text{True Alarm Rate} = \frac{\text{True Alarms}}{\text{Total Alarms}} \times 100$$

The lack of knowledge or awareness about the complexity of network by IDS technology has led to the generation of excessive amount of false alarms. Generally, there are three possible alert types raised by the system, namely true positives (alerts from real attacks), false positives (legitimate activities thought to be malicious) and irrelevant positives (alerts from unsuccessful attacks or attempts (Kruegel and Robertson, 2004). The last two alerts are the main concerns in this study.

This section presents the results of the experiment. Figure 4.5 depicts the overall result, which represents the general detection performance of Snort IDS using a similar ROC plot diagram as illustrated in Figure 4.2. In order to create a simpler illustrative graph, which facilitates the comprehension of Snort's detection ability, the false and true positives values are presented in a proportion of thousands. The number of false positives generated is presented per unit time (per day) for the X-axis, while true positives are portrayed for the Y-axis.

Significantly, the research has also produced a similar result to that yielded in Brugger and Chow's evaluation (Brugger and Chow, 2007). Their study reported that the number of false positives generated was massive. This indicates that Snort's false positive performance on real network could be much worse than described in their evaluation.

This experiment focused on the analysis of false positive alarms, as opposed to other studies (Mahoney and Chan, 2003; Brugger and Chow, 2007), which were directed to explore the issue of false negatives. The main objective of this analysis is to merely provide a general view of the scale of false positives that may be generated by current IDS. The following subsections discuss this case in greater detail.

4.3.1 False Positives

A large volume of alerts, largely comprised of false alarms and irrelevant positives, drives the need to verify the validity of the alerts generated. Interestingly, apart from the false positives, the study reveals that some alerts were raised due to informational events, which merely occurred as a result of a network problem, not owing to the detection of real attacks. These types of alerts are known as

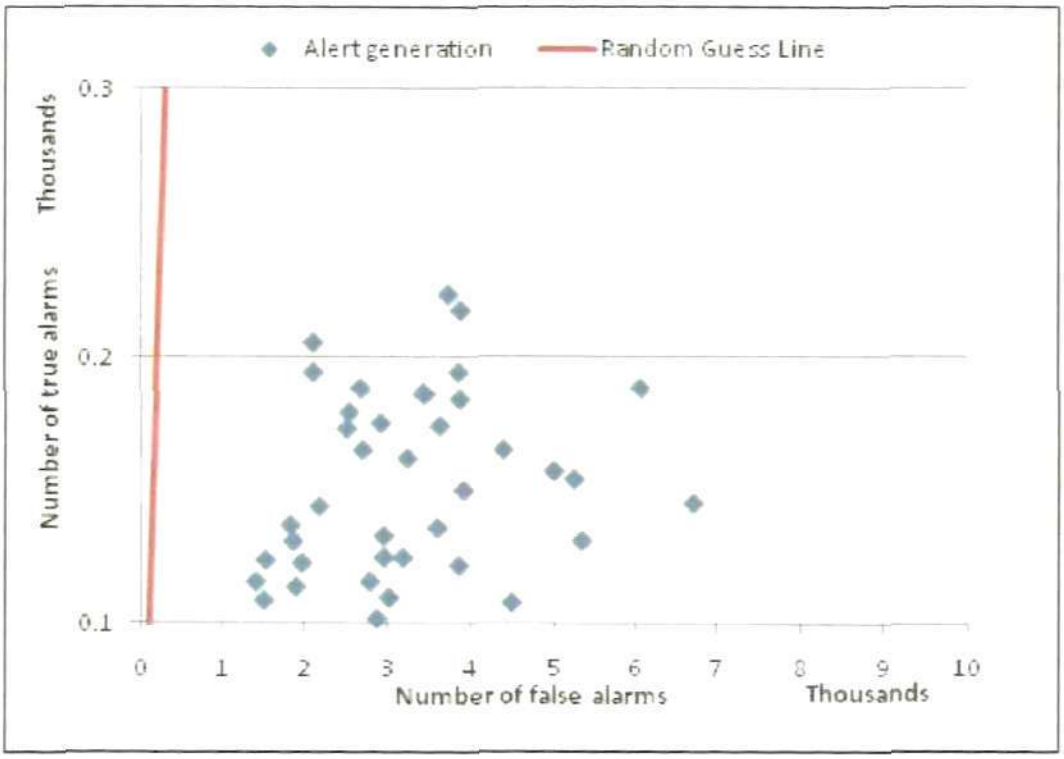


Figure 4.5: Generation of alerts on University of Plymouth data before tuning

irrelevant positives. Indeed, the unsuccessful attacks, or attempts that aim at an invincible target, might cause the system to generate such alarms.

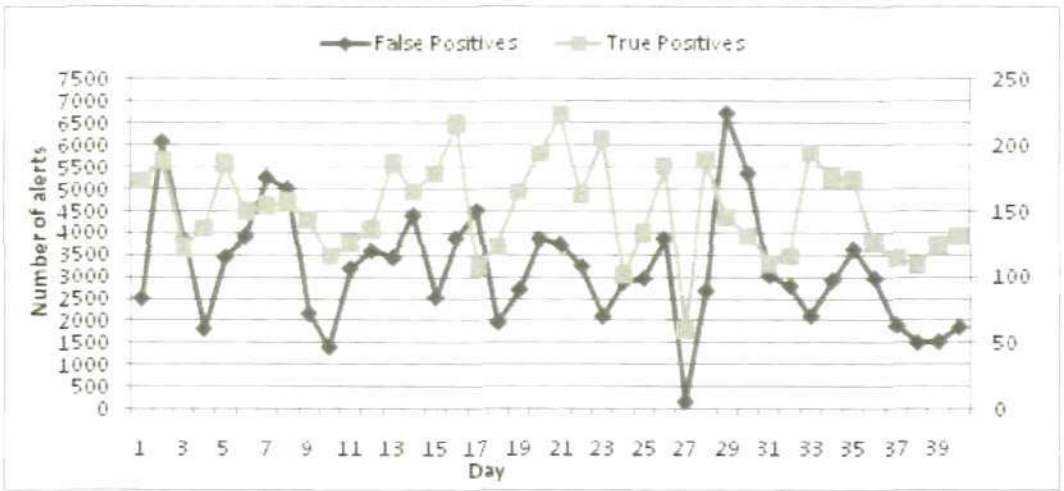


Figure 4.6: Comparison between false positive and true positive alarms on University of Plymouth data

Figure 4.6 provides a clear picture of the number of true and false alarms generated per day. To clearly map the figures of true alarms, the plot diagram uses secondary axis to plot the values of true positives. In this case, the right hand side y-axis represents true positives, whilst the left hand

side is for the false positives. In this context, it is obvious that the false alarms highly outnumbered the true alarms. Significantly, this experiment has revealed a similar result to that yielded in the DARPA evaluation (Section 4.2). The result is similar to the previous findings, which further confirms the severity of the problem of false alarms. Approximately 96% of alerts generated are false positives, while less than 1% of the total alerts are affirmed to be irrelevant positives. In order to make it simpler, irrelevant alarms are regarded as false positives alerts in this case since no immediate and crucial responses required from these events. By looking at the Snort alerts generated from the University's web server, most of the false positive alarms came from the category of web application activity. Table 4.1 shows a complete list of the Snort alerts triggered by the data. The first three alerts are the false positives alerts, which will be further investigated later on. The reason for focusing on these alerts is due to the quantity generated, which accounts for more than 80% of total alerts raised by the system.

Table 4.1: Total Alerts per Signature

No	Signatures	Total Alerts
1.	WEB-IIS view source via translate header	78865
2.	WEB-MISC robots.txt access	30011
3.	ICMP L3retriever Ping	10254
4.	BARE BYTE UNICODE ENCODING	6392
5.	POLICY Google Desktop activity	3258
6.	SPYWARE-PUT Trackware funwebproducts mywebsearchtoolbar-funtools runtime detection	1873
7.	ATTACK-RESPONSE 403 Forbidden	720
8.	ICMP PING Cyberkit 2.2 Windows	651
9.	DOUBLE DECODING ATTACK	504
10.	ICMP Destination Unreachable Communication Administratively Prohibited	151
11.	TCP Portsweep	124
12.	SPYWARE-PUT Hijacker searchmiracle-elitebar runtime detection	80
13.	WEB-MISC .DS_Store access	60
14.	IIS UNICODE CODEPOINT ENCODING	49
15.	WEBROOT DIRECTORY TRAVERSAL	35
16.	SPYWARE-PUT Adware hotbar runtime detection hotbar user-agent	27
17.	WEB-IIS asp-dot attempt	26
18.	TCP Portscan	19
19.	SPYWARE-PUT Trackware alexa runtime detection	19
20.	WEB-PHP IGeneric Free Shopping Cart page.php access	17
21.	ICMP PING NMAP	17
22.	ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited	13

Continued on next page

Table 4.1 – continued from previous page

No	Signatures	False alarms
23.	WEB-CGI calendar access	11
24.	MULTIMEDIA Quicktime User Agent Access	10
25.	WEB-MISC intranet access	8
26.	ICMP redirect host	8
27.	ICMP PING speedera	7
28.	SPYWARE-PUT Hijacker marketscore runtime detection	7
29.	WARNING: ICMP Original IP Fragmented and Offset Not 0!	6
30.	WEB-MISC WebDAV search access	5
31.	WEB-FRONTPAGE /_vti_bin/access	5
32.	Open Port	5
33.	WEB-PHP remote include path	4
34.	WEB-CGI formmail access	3
35.	WEB-FRONTPAGE_vti_inf.html access	3
36.	SPYWARE-PUT Trickler teomasearchbar runtime detection	2
37.	WEB-PHP xmlrpc.php post attempt	2
38.	WEB-CLIENT Microsoft wmf metafile access	2
39.	WEB-MISC Domino webadmin.nsf access	2
40.	OVERSIZE CHUNK ENCODING	2
41.	ICMP Source Quench	2
42.	WEB-PHP test.php access	2
43.	WEB-PHP calendar.php access	1
44.	WEB-PHP admin.php access	1

4.3.1.1 WEB-IIS view source via translate header

This event is categorised as web application activity, which targets the Microsoft IIS 5.0 source disclosure vulnerability (Snort, 2010c). Since Microsoft IIS has the capability of handling various advanced scriptable files such as ASP, ASA and HTR, the use of specialised header "Translate f" on HTTP GET request is likely to force the web server to present the complete source code of the requested file to the client without being executed first by the scripting engine. In addition, this attack only works well if the trailing slash "/" is appended to the end of requested URL (Bugtraq, 2010a,b).

Surprisingly, this signature accounted for 59% of the total alerts. Therefore, approximately 1970 alerts were generated per day by this event. Although the signature was solely created to detect an attack targeting the Microsoft IIS source disclosure vulnerability, there is a certainty that this signature will generate a false alarms in a certain case. Some applications, for example Web-based Distributed Authoring and Versioning (WebDAV) that make use of "Translate f" as a legitimate header, might cause this rule to produce an excessive amount of false alarms (WebDAV,

2001). Moreover, WebDAV PROPFIND and OPTION methods also make use of this "Translate f" as a legitimate header to retrieve the information or properties of the resources identified by the Uniform Resource Identifier (URI) (nearly 96% of alerts generated by this signature were not HTTP GET requests). Significantly, in this experiment, none of the alerts generated by this signature were triggered as a result of a real attack, so no further acts were required to handle these alerts.

4.3.1.2 WEB-MISC robots.txt access

This event is raised when an attempt has been made to access robots.txt file directly (Snort, 2010d). Basically, robots.txt file is a file that is created to keep the web pages from being indexed by search engines. More to the point, this file provides a specific instruction and determines which part of a website a spider robot may visit. Interestingly, the problem is that the webmaster may detail sensitive and hidden directories or even the location of the secret files within the robots.txt file. This is considered extremely unsafe since this file is located in web server's document root directory, which can be freely retrieved by anyone.

Although this event is raised as the indicator of vulnerable information attack, there exists high possibility that all these alerts were raised due to legitimate activities from web robots or spiders. A spider is software that gathers information for search engines by crawling around the web indexing web pages and links in those pages. Robots.txt file is basically created to restrict the web spider from indexing pages that should not be indexed, for example, submission pages or enquiry pages. As web indexing is regular and structurally repetitive, this activity tends to cause the IDS to trigger a superfluous amount of alerts. In this study, approximately 23% of total alerts (approximately 750 alarms per day) were accounted for by this web-misc activity. Given that all alerts generated from this event are owing to the activities of web spider, they are considered to be false positives. Significantly, this issue has apparently disclosed the drawback of Snort IDS in distinguishing legitimate activity from the malicious one; especially when it deals with the authorisation or file permission.

4.3.1.3 L3Retriever Ping

ICMP L3retriever Ping is an event that occurs when ICMP echo request is made from a host running L3Retriever scanner (Snort, 2010a). This type of ICMP echo request has a unique payload in the message, which significantly designates its distinctive characteristic. This traffic is considered to be an attempted reconnaissance since the attackers may use the ping command to obtain ICMP echo reply from a listening host. Surprisingly, in this analysis, quite a few alerts were generated from this event; contributing to 8% of the total alerts generated. This figure indicates that approximately 250 alerts were generated by this signature rule every day.

Considering the source IP address associated with these alerts, it is obviously clear that all ICMP requests were sent from the external hosts. Further investigation was conducted to critically analyse and discover if possible malicious events happened subsequent to the ICMP echo request. Surprisingly, there were no malevolent activities detected following the ICMP traffic. In addition, normal ICMP requests generated by Windows 2000 and Windows XP are also known to have similar payloads to the one generated by L3Retriever scanner (Greenwood, 2007). Generally, this traffic is routine activities run by computer systems (notably Windows 2000 and XP systems) to communicate with their domain controllers or to perform network discovery. In view of this issue and given

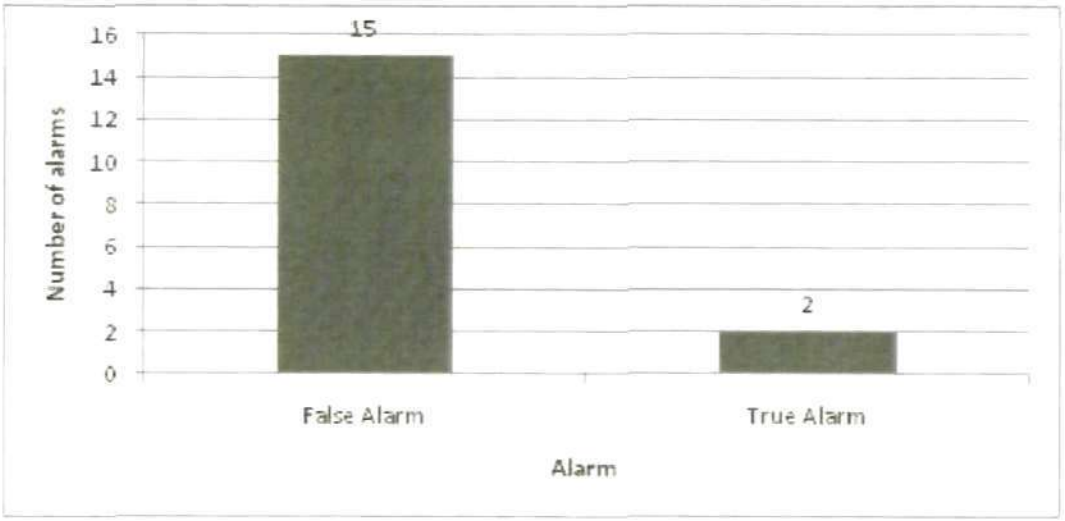


Figure 4.7: "ICMP PING NMAP" event

that no suspicious output detected following these ICMP requests; these alerts were labelled as false positives.

4.3.2 Fine Tuning

False alarms for one system might not be an erroneous alert for other systems. For example, port scanning might be a malicious activity for normal users, but it is a legitimate activity if it is performed by a system administrator. Figure 4.7 shows an example of an event which triggered both false alarms and true alarms from the experiment. From the IDS's perspective, as long the activity's pattern match to the signature defined in the rule database, it is considered to be a malicious event. In view of this, fine tuning is exceptionally required to maintain the IDS performance and enable the administrator to adapt the signature rule to the protected environment.

In order to optimize Snort performance, fine tuning is necessary to reduce the number of alerts raised. Since only three signatures were tuned in this experiment, the false alarm rate accounted for 86.8% of total alarms after tuning was performed. Figure 4.8 depicts the ROC plots for the overall result after tuning was performed. Obviously, only less than two thousands alerts per alert type have been generated by Snort. In order to understand the effectiveness of fine tuning, the alarm rate between default and tuned Snort is presented in Figure 4.9. This figure does not seem particularly impressive, but fine tuning did fare well on those signatures; reducing up to 90% of false alarms per signature, excluding WEB-MISC robots.txt access. The following subsections discuss the tuning processes in more details.

4.3.2.1 WEB-IIS view source via translate header

Regarding the information disclosure vulnerability attack, Snort does not seem proficient enough to detect this type of event. The signature rule appears to be very loosely written, by searching for a particular string in the packet payload (in this case, "Translate: f"). Since the "Translate: f" is a valid header used in WebDAV application, as discussed previously, this rule tends to trigger a vast

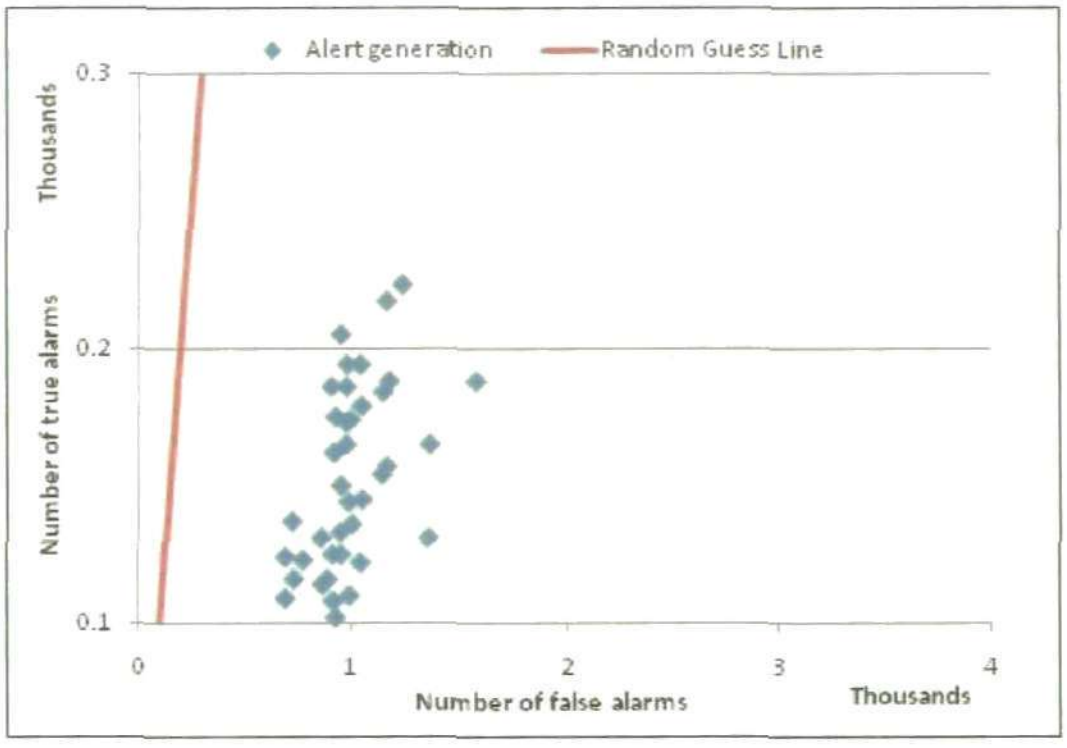


Figure 4.8: Generation of alerts on University of Plymouth data after tuning

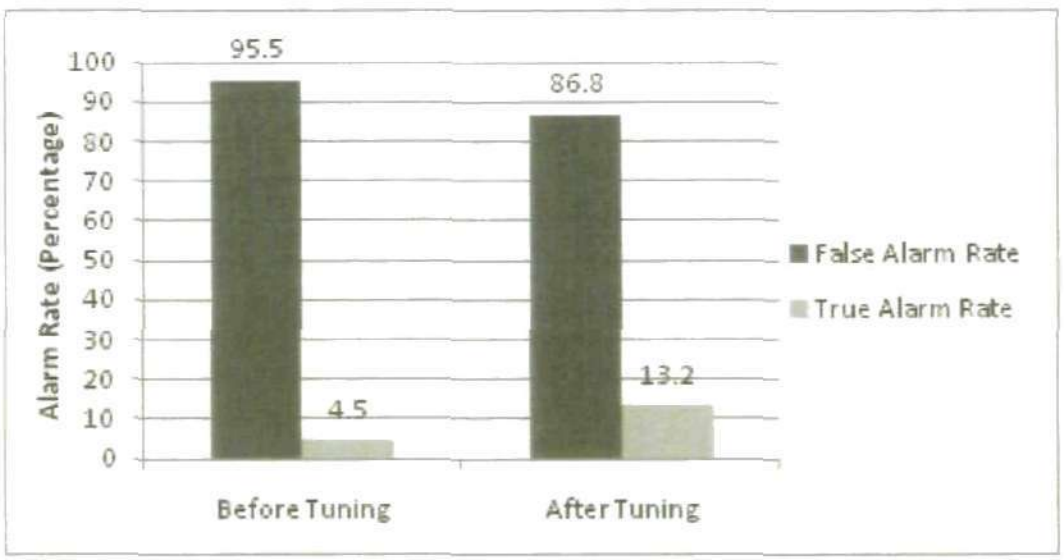


Figure 4.9: Alarm rate before and after tuning

volume of alerts from the legitimate activities. Hence, tuning is needed to search for a more specific pattern of the attack signature.

As this attack is basically launched through HTTP GET request, searching for "GET" command in the content of analysed packet can be a good start. The attack is launched by requesting a specific resource using HTTP GET command, followed by "Translate: f" as the header of HTTP request. In

this case, a tuning can be performed by modifying the signature rule to:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS view source via translate header";
flow:to_server,established; content:"GET|20|";
content:"Translate|3A| F"; distance:0; nocase;
reference:arachnids,305; reference:bugtraq,14764;
reference:bugtraq,1578; reference:cve,2000-0778;
reference:nessus,10491; classtype:web-application-activity;
sid:1042; rev:13;)
```

The tuning process significantly reduced the number of alerts, with only 3,463 generated by this rule as against 78,865 alerts in the first case (that is without tuning). Significantly, this tuned rule was proved to effectively eliminate 95% of the initial false alarms from this event.

Although the tuning process decreased the volume of alerts, there is still a possibility that those 5% alerts were false positives. Searching for GET command and the Translate f header is not effective enough to detect such attack. Putting trailing slash "/" at the end of requested URL to HTTP request for example could lead in the security bug (Bugtraq, 2010a). Thus, matching the "/" pattern against the packet payload will be helpful. Unfortunately, this idea seems hardly possible to achieve. Snort does not have a specific rule option that can be used to match a specific pattern at a particular location.

As to the signatures of Snort, looking for an overly specific pattern of a particular attack may effectively reduce the false alarms; however, this method can highly increase the risk of missing its range. A skilful attacker can easily alter and abuse the vulnerability in various ways as an attempt to evade the IDS. This might lead to false negatives as a consequence.

4.3.2.2 WEB-MISC robots.txt access

Since accessing the robots.txt file is a legitimate request for Internet bots (web spiders), a subjective rule, which mainly focuses on the source IP addresses, is necessary to verify user authorisation in accessing a certain file. This approach, however, seems to be hardly feasible to deploy. Of course, identifying all authorised hosts from their source IP addresses is impractical. There is an infinite number of IP addresses need to be discovered before the rule can be written. Indeed, lawfully allowing specific hosts to access certain file might increase the risk of having false negatives.

In this case, the only solution to suppress the number of false alarms generated is by using event thresholding (Beale and Caswell, 2004). As robots.txt access requests generate regular and repetitive traffic, a "limit" type of threshold command is the most suitable tuning in this case. Such a threshold configuration would be as follows:

```
threshold gen_id 1, sig_id 1852, type limit, track by_src, count
1, seconds 60
```

This rule logs the first event every 60 seconds, and ignores events for the rest of the time interval. The result showed that approximately 10% of false alarms were effectively reduced. This indicates that only an insignificant number of false alarms can be reduced in this scenario. The frequency of fetching robots.txt files greatly depends on the web spider's policy. Hence, deploying event suppression and thresholding cannot effectively trim down the number of false alarms logged by the system. Having said that, suppressing the number of alerts generated can also create a possibility of ignoring or missing real alerts. For example, a malicious user can hide his/her action within the excessive number of alerts generated by using a spoofed address from web spider agent.

4.3.2.3 ICMP L3Retriever Ping

The only method that can be deployed to suppress the number of false positive triggered from this event is by applying event suppressing or thresholding command. Similar to the one applied to "WEB-MISC robots.txt access" signature, a threshold command is written to limit the number of alarms logged. Instead of using "limit" type of threshold command as previous signature, this rule utilised "both" type of command to log alerts once per time interval and ignore additional alerts generated during that period:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP
L3retriever Ping"; icode:0; itype:8;
content:"ABCDEFGHIJKLMNOPQRSTUVWABCDEFGHI"; depth:32;
reference:arachnids,311; classtype:attempted-recon; threshold:
type both, track by_src, count 3, seconds 60; sid:466; rev:5;)
```

Similar to the previous signature (robots.txt access), the threshold applied will not prevent the generation of false positives, but it will significantly reduce the number of redundant false positives triggered. Importantly, the threshold is written to detect brisk ICMP echo requests by logging alerts once per 60 seconds after seeing 3 occurrences of this event.

The result showed that only 1,143 alerts were generated from this event in 40 days experiment data. This experiment has also proved that the event thresholding can successfully reduce up to 89% of the false alarms generated by this activity. Despite its ability in suppressing redundant alarms, the system is prone to missing stealthy ICMP requests, for example, requests sent once every 60 seconds can be missed by the system.

Although such technique effectively suppresses the alerts, it potentially increases the risk of false negatives. In a case where an attacker sends the request only once, the IDS might miss this event if the thresholding is enabled. The malicious request might have assimilated into the superfluous number of genuine requests.

In consideration of this issue, it is highly advised that tuning and thresholding should only be carried out by a security expert who has broad knowledge of the network security and also the protected system. It is often a case of trading off between sensitivity (that is false negative) and specificity (that is false positive).

4.4 Discussion

Similar to this experiment, an evaluation was carried out by Brugger and Chow (2007) to assess the performance of traditional IDS, Snort. This evaluation was conducted using the baseline Defense Advanced Research Projects Agency (DARPA) dataset 1999 against a contemporary version of Snort. This assessment was performed to appraise the usefulness of DARPA as an IDS evaluation dataset and the effectiveness of the Snort ruleset against the dataset. In order to analyse Snort's alarms, a perl matcher script was used to report the false negative and positive rates; thus generating the ROC curve for a given set of attacks. Given the six year time span between the ruleset and the creation of the dataset, it was expected that Snort could have effectively identified all attacks contained in the dataset. Conversely, what they found instead was the detection performance was very low and the system produced an unacceptably high rate of false positives, which rose above the 50% ROC's guess line rate. This might be due to the fact that Snort has a problem detecting attacks modelled by the DARPA dataset, which focused on denial of service and probing activities (Lippmann et al., 2000). In particular, the false alarm rate reported in this evaluation was not creditable enough to prove Snort false positive performance in a real network, which might be much worse or much better. In view of that, the research decided to utilise more realistic data to critically evaluate the false positive issue of the system.

The experiment presented here has revealed a similar result to the work of Brugger and Chow (2007). Over a span of two years since their research was published, the issue of false positives remains a critical challenge for the Snort IDS. Obviously, Snort performance does not look particularly remarkable as illustrated in Figure 4.5. The bottom right scattered plots demonstrate that the number of false positives largely overwhelms the number of true positives. Approximately 3,000 alerts were generated per day, requiring manual verification to validate their legitimacy. Although the administrator can effectively distinguish the false and true positives from the list of alerts, the massive amount of false alarms triggered by one signature rule might cause the administrator to miss a malicious attack.

The overall effectiveness of Snort greatly hinges on the effectiveness of keyword spotting (in other words, matching the packet content to the signature rule). This has rendered the system prone to generating a superfluous number of false alerts. Interestingly, most of the rules looking for web traffic related attacks are loosely written and merely check for the presence of a particular string in the packet payload. This could trigger a large number of false alerts if a particular string is included in the content distributed by the web server. Hence, from this perspective, Snort is deemed not to be ideal enough to detect more complex attacks, which are not detectable by a pre-defined signature.

As for the DARPA experiment, it was initially thought that Snort could fare well in detecting DARPA attacks. However, the fact is Snort detection performance is low, only 32 attacks were detected, and Snort produced a large volume of false positives. Indeed, Snort also missed numerous attack instances from those 32 attacks. From this experiment, it is obvious that the issue of false alarm has become very much critical. The false positives generated greatly outnumbered the true positive alarms, as depicted in Figure 4.1. In fact, more than half of the signatures producing both true and false positives in this evaluation triggered more false positive and true positive alarms. This issue would critically decline IDS detection performance; not only in this simulated network

environment but also bound to happen in a real environment.

Regarding the quality of alerts, Snort generated a significant number of redundant alerts, which critically highlighted the performance issue of its alert reporting system. This often leaves the administrator with the overwhelming alerts, which then render the alert validation job hard to manage. Importantly, this issue has also driven the need to have an improved or better alarm reporting system through the implementation of alarm suppression and correlation methods.

Snort raised 28340 pure false positive alarms that accounted for 31% of total alarms generated. Such issue is also likely to happen in a real-network environment. However, in this experiment, the cause of these alerts was not individually tracked. Having said that, it is believed that this might be caused by the nature of Snort IDS, which relies on keyword spotting (in other words, matching the packet content to signature rule) to detect malicious activity. Significantly, this finding underlines another weakness of Snort IDS, which render the system prone to produce excessive alerts.

4.5 Conclusions

This chapter discussed the results of false alarm evaluation on both the DARPA 1999 data set and the University of Plymouth private data set. It then continued to highlight the issue of false alarms and critically examine the impact of false alarms on the IDS detection rate.

In general, the study has confirmed the criticality of IDS false alarm issue. Given the findings in the DARPA evaluation, endorsed by the experiment results on University private data set, it is clear that false alarm is a never-ending issue faced by current IDS. This motivates the need to enhance the system performance or even to improve the quality of alerts generated.

Having investigated the problem of false alarms, the next chapter presents the architectural framework of a proposed alarm correlation and reduction system. This is followed by the descriptions of the main modules involved in such an approach.

5 A Novel Approach to Alarm Correlation

5.1 Introduction

Having identified current literature or studies on IDS alarm correlation methods and the drawbacks associated with them, this chapter introduces a new alarm correlation system, which aims to aggregate alerts from the same attack instances and classify alerts into two classes, the true and the false alarm. This chapter begins by introducing the concepts of the applied methodologies and the rationale behind the implementation. The proposed model is then presented, followed by preliminary experiment results on both 1999 DARPA data set and University of Plymouth private data.

5.2 Methodology

To answer the issues of current correlation systems (as described in Chapter 3), an automatic alarm correlation and filtering system for signature-based IDS is proposed using unsupervised clustering techniques, namely Self Organising Map (SOM) (Kohonen, 1995) and $K-$means algorithm (MacQueen, 1967). The data mining techniques are commonly used in data reduction and data clustering. The reason of choosing these algorithms is because they are easy to implement and able to show or clarify the relationship between the classified data.

The proposed system is developed to process alerts generated by signature-based IDS, including aggregating or correlating alerts associated with the same attack instance and clustering the alerts into groups of true and false alarms. Existing alarm correlation studies (as presented in Chapter 3) were conducted to achieve either of these purposes; to construct an attack scenario by aggregating alerts related to the same attack (Ning et al., 2002; Debar and Wespi, 2001; Cuppens and Miege, 2002) or to identify false alerts (Maggi et al., 2009; Spathoulas and Katsikas, 2010). Unlike the previous works, the proposed system aims to achieve both objectives by introducing a two-stage correlation. The rationale behind this is to propose an automated tool that not only can discover the relationship between alerts but can also identify false alarms without the need of domain knowledge.

The following subsections introduce the basic concepts and the operational features of the applied algorithms (SOM and $K-$means).

5.2.1 Self Organising Map (SOM)

A Self Organising Map (SOM) is an unsupervised neural network which produces a feature map that maintains the topology of the input data according to their similarity. Unlike typical neural networks that need to be trained with their desired outputs, SOM can automatically categorise the varieties of input presented during training without any external supervision whatsoever and assess the accuracy of its classification. In other words, SOM is a type of learning process in which the neighbouring cells in a neural network are competing in their activities through mutual lateral interactions and are specifically tuned to adapt to various input signal patterns or classes of patterns (Kohonen, 1995). It relies on the absence or presence of an active response given by the cell or local cell group, not so much on the input-output signal transformation itself, to provide an interpretation of the input information.

Some SOM applications require a proficient construction of large maps. Searching the best matching unit from a large map is usually the computationally heaviest operation in the SOM. However, using a tree-structured SOM, it is possible to use hierarchical search for the best unit (Koikkalainen, 1994). The idea behind this method is to build a hierarchy of SOMs, in other words, multiple maps, training the SOM on each layer before proceeding to the next level. In this work, a standard version of SOM was applied instead of the tree-structured SOM, which could be applied as an alternative method for the future work.

The SOMs have been commonly used in a wide range of fields; involving data mining, pattern recognition, image processing, robotic, process controls and visualization methods for complex data sets. In fact, it is one of the most popular unsupervised learning algorithms applied in IDS research (Albayrak et al., 2005). The initial applications of a SOM architecture to the IDS issue were already proposed (as discussed in Chapter 3) (Ramadas et al., 2003; Kayacik et al., 2007; Xiao and Han, 2006; Powers and He, 2008). Moreover, the appropriateness of such a method in the study of large data sets was also established (Kohonen et al., 2000).

Unsupervised learning using SOM offers a simple yet efficient way of clustering data sets. It is empirically proven that SOM is best suited to data classification due to their high speed and fast conversion rates as compared with other learning techniques (Labib and Vemuri, 2002). Also, in terms of its data representation, this method is deemed to outperform other algorithms,for example, ART (Adaptive Resonance Theory), owing to its ability to preserve topological mappings between the input data. This represents a significant feature, which is desired when introducing the relationship between the generated alerts. In other words, the organisation of the data using the SOM-based approach enables the system to learn the relationship between alerts based on the defined attributes.

The idea of the SOM algorithm is to perform a data compression technique (vector quantisation) where a high dimensional data is represented or mapped into something that is better understood visually such as a 2-dimensional array. The approach is considered as being highly effective as a complex visualisation tool for picturing extensive, multidimensional space with the intrinsic relationship among the various attributes comprising the data. Given these benefits, SOM is selected as the main correlation method in the proposed architecture. The basic concept, architecture and implementation technique of SOM can be found in (Kohonen, 1995).

5.2.2 $K-$Means

$K-$means (MacQueen, 1967) is a simple unsupervised learning algorithm that answers the well-known clustering problem by grouping n objects based on attributes into k partitions, where $k < n$. The implementation of $K-$means assumes all attributes to be independent and normally dispersed. The main concept of this approach is to define k appropriate centroids, one for each cluster and then group all data into the pre-defined k subsets. The grouping is done by calculating the sum of distances or sum of squared Euclidean distances from the mean of each cluster, as shown below.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{5.1}$$

, where p and q are the cluster points and n is the number of attributes.

Hence, the objective of this clustering is to minimise a measure of dispersion within the clusters and to maximise the distance between clusters (Kanungo et al., 2002).

In this work, the $K-$means procedure will be started by assigning the data to k initial clusters at random. It is also worth noting that the cluster solutions can be influenced by the order of the input data. The randomised trials therefore involve randomising both the initial clusters and the data order. To get the best clustering solution, the proposed system looks for the top solution by exploring a range of cluster solutions produced by the procedure and examining their criterion value; involving the minimum sum of squared error and the highest frequency rate.

The popularity of the $K-$means algorithm is mainly attributed to its simplicity, scalability and fast convergence. The rationale behind the application of the $K-$means algorithm is to overcome the limitation of the Self Organizing Map, the lack of automatic cluster detection in U-Matrix representation. The detail of this issue is discussed in Section 5.3. The idea of combining Self Organizing Map and $K-$Means algorithm was already demonstrated in data clustering and visualisation (Ong and Abidi, 1999).

Similar to other algorithms, $K-$means clustering also has weaknesses. $K-$means is considered to be unstable; running the procedure several times will give several different cluster solutions (van der Heijden et al., 2004). Depending on its initial condition, the algorithm may converge or be trapped in the local optimum (minima). In addition, when the number of pre-specified classes is high, it often happens that some clusters are ignored during the classification as no sufficient support is given. In that case, the number of effective clusters will turn out to be much less than k. *With the issues of classification in mind, the proposed approach is directed to focus only on an interaction* between the intrinsic structures (alerts' attributes) in the instances and the representation of the data in the Kohonen map.

5.3 A Proposed Alarm Reduction and Correlation System

The system comprises two main components; the first module is responsible for aggregating alerts raised by the same attack instances into clusters whilst the second component classifies the formed clusters into meta-clusters of true and false alarms. Meta-cluster is a cluster that is made of other

smaller clusters. The main objectives of the proposed correlation system are to filter the false alarms, enable the security operators to scrutinise or learn the trends of false alarms through statistical figures or charts provided; thus assisting them in tuning the IDS signatures for future detection.

5.3.1 A Two-Tier Architecture

The main benefit of using the unsupervised algorithms is the automated clustering of IDS alerts based on their feature similarity. To perform a correlation, features (attributes) are extracted from the alerts and fed into the correlation engine. Building accurate and efficient classifiers largely depends on the accuracy of the attributes, which are used as the input data for the classification.

Figure 5.1 depicts the proposed classification model.

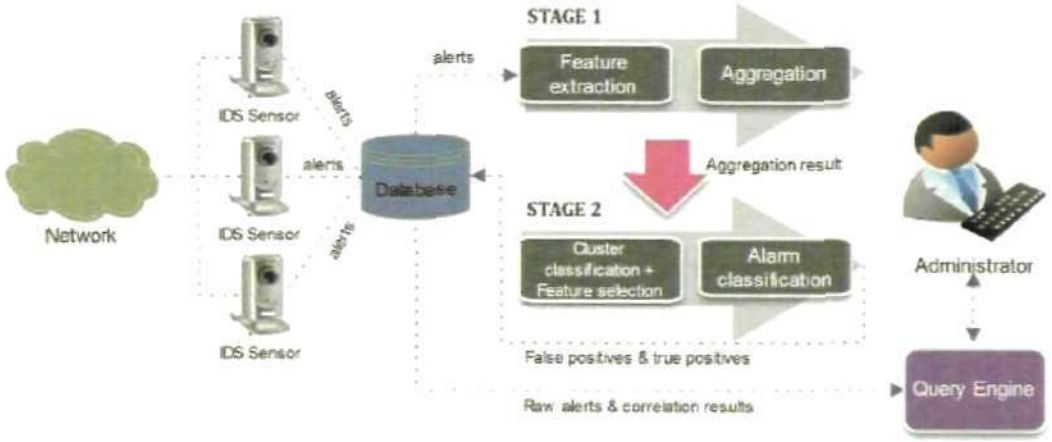


Figure 5.1: Framework of alarm correlation system

In order to achieve both objectives (as explained in Section 5.2), the whole correlation procedure consists of four phases: *feature extraction, aggregation, cluster classification + feature selection* and *alarm classification*. The *feature extraction* and *aggregation* phases have been commonly implemented in alert classification and correlation model (Jan et al., 2009; Al-Mamory and Zhang, 2009; Julisch, 2001; Julisch and Dacier, 2002; Sadoddin and Ghorbani, 2009; Smith et al., 2008).

As described in Figure 5.1, raw alerts from IDS sensors are collected and stored in a database. In the *feature extraction* phase, the alerts are retrieved from the database and several attributes, which are considered effective to correlate alerts coming from a single activity, are extracted from each alert. The alert attributes are the characteristics of alerts that are commonly used to identify an intrusion, for example the IP addresses or the protocols. The extracted data are then normalised since the value of the data are varied depending on the type of attributes used. A significant variance between attributes value will produce an uneven or biased result.

Given a set of n-dimensional input vectors, where n is the number attributes for each alert, from the first phase, the system is trained unsupervised using SOM algorithm in the second phase to map the inputs so that similar vectors are reflected in their arrangement. The map created by SOM consists of a number of nodes, where each node contains a vector of weights of the same dimension as the input vectors. The distance between two input vectors is presented on the map, not by their

absolute dissimilarity (which can be calculated), but the relative differences of the data properties. The created map would be especially useful as a visual feedback to the user (network administrator), which is one of the main reasons why this approach is used. Whilst many other techniques offer a better or more accurate result for clustering and multidimensional scaling (Flexer, 1997), they were deemed not as suitable for online, real-time data processing as SOM (Kohonen, 1990) (a future enhancement for the system).

The SOM training process, through which the relationships are built, is fairly simple. During the training, SOM is expected to randomise the map's prototype (node) vector elements within the range of the input value. The iteration is carried out to obtain a node which is most similar to the input vector. Once it is found, the node and its neighbours on the map are incrementally adjusted to more closely resemble the data.

As soon as the final Kohonen map is produced, the trained SOM can be automatically visualised using the U-Matrix method. Having said that, SOM clustering alone is not good enough to describe the boundaries between the data items since there are no clear walls to separate them from the other items. Classifying the data without any prior knowledge, is therefore inconsistent and difficult. The result of this U-Matrix is merely used for visualisation purposes and the interpretation of the U-Matrix values is considered subjective. To avoid this issue, therefore, the system applies a traditional clustering method, $K-$means clustering. Based on the map produced by the trained SOM, $K-$means clustering is implemented to further define the boundaries between the data and concurrently classify the input vectors into a number of pre-defined clusters. At the end of the second phase, the system is expected to form clusters by correlating all alerts generated by a single activity, meaning one cluster for each event/activity.

In the third phase, *cluster classification + feature selection*, the clusters formed in the previous phase are further evaluated and the attributes of each cluster are extracted. Seven alert attributes (features) were chosen from each cluster to represent the value of each input vector in the final clustering. Whilst five attributes can be automatically extracted from the clusters, the other two attributes, namely the number of occurrences of an event triggering a particular alarm signature and the average time interval between the events in a pre-defined time frame will need to be computed individually. These features are considered to be the most crucial attributes influencing the magnitude of the alert signatures.

A high generation of alerts from a single signature provides a good indication of noisy false alerts, such as ones triggered by the ICMP traffic. However, it can also indicate the occurrence of a denial of service attack if the alerts are raised within a short time interval. To answer this problem, the recurrence rate of an event triggering a particular signature and the average time interval between the events are selected to determine the authenticity of the alerts. The combination of both features will ultimately decide the validity of the alerts, whether they are true or false alarms.

To achieve an optimal result, the number of events and time interval features are emphasised and it is necessary to examine how the attributes' weights from the two features can greatly affect the outcome of the classification. In which case, fine-tuning is performed before the classification to determine the most appropriate attribute weights and to ensure that such attributes contribute more to the grouping processes.

Using the similar clustering concepts applied in the *aggregation* phase, the SOM and $K-$means algorithm are re-applied in the final phase to classify the input data produced in the third phase. In

this scenario, a set of seven-dimensional input vectors are fed into the clustering engine. And the final outcome of this classification is two meta-clusters, namely true positives and false positives.

The final clustering reveals that a cluster containing a signature with higher event frequency rate and a shorter time interval between events are prone to represent a false alarm class and vice versa for true alarms. The diagram of the correlation process and the relationships among the components appear in Figure 5.2. In addition to this, the full algorithms of the correlation are presented in Section 5.4.

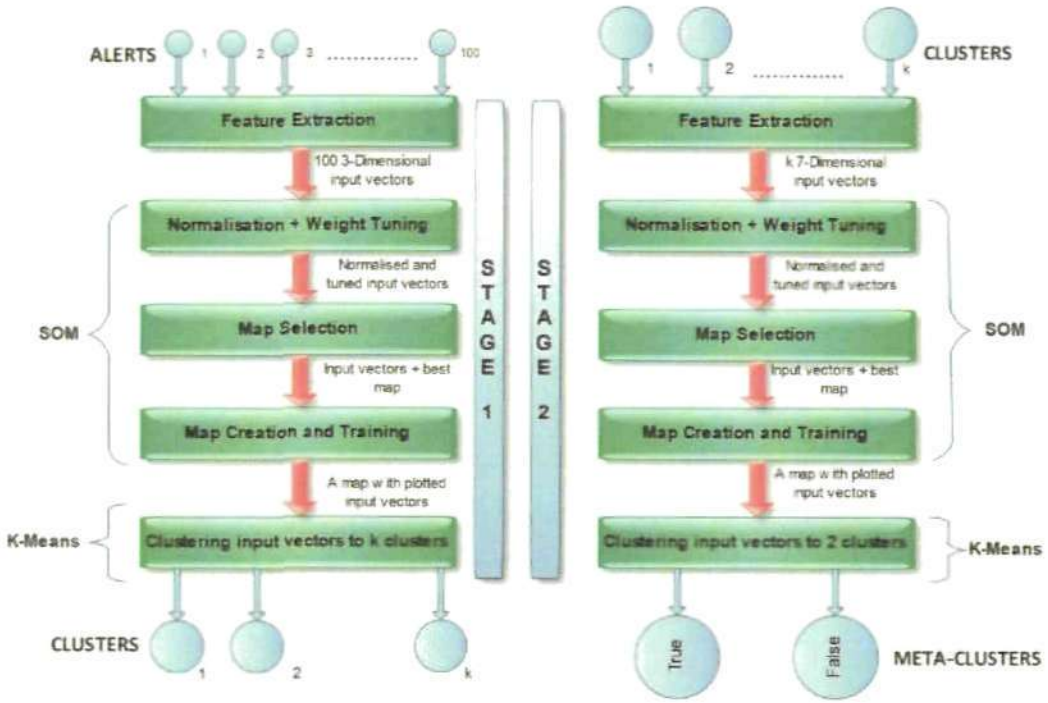


Figure 5.2: Architecture of alarm correlation system

5.3.2 Stage 1 - Alarm Aggregation

Following subsections present alarm attributes and fundamental concepts of the alarm aggregation phase:

5.3.2.1 Alarm attributes

In order to correlate related alarms, it is necessary to remove the inapt attributes and select only the most appropriate ones. After evaluating a number of potential features, three significant attributes have been chosen to represent the relationships between alerts. Those are the timestamp, the source and destination IP addresses. IP address is deemed to be the most critical feature determining the subject of the occurrence. Conversely, the timestamp determines the time of the event and whether a particular alert within a specific time period should be aggregated. By using the combination of these features, alerts triggered by particular IP addresses were correlated within a particular period of time.

In order to correctly spot the events triggered by particular hosts, the combination of both source and destination IP addresses was used. So, instead of using the original IP addresses, the system is designed to compute the addition and the subtraction between the source and destination IP addresses. Before the computation, the IP addresses were converted into their decimal value from the common dotted decimal notation, for example, 123.7.1.10 becomes 2064056586. The idea of translating dotted decimal IP address to its decimal equivalent is commonly used by MySQL database.

The main objective of this pre-processing step is to obtain a distinctive pair of IP addresses from an alert without the need of identifying the source and destination addresses. Such approach enables the system to connect all alerts which involve the two IP addresses within a particular time frame. For example, alerts generated by ICMP Ping and ICMP Echo Reply signatures can be correlated since they commonly associate to a same pair of IP addresses. In order to obtain a same pair of addition and subtraction values of two IP addresses in any order, only the absolute value of the subtraction is taken. For example, if the subtraction between 2886758706 (source) and 2886759119 (dest) is -413, then the absolute value 413 is selected. So, although the source and destination IP addresses are reversed, the subtraction will still yield the same value. As this technique uses the characteristics of both difference and addition of IP addresses, which are taken in time context, the likelihood of having collisions is low (different pair of IP addresses are mapped into the same cluster) (Chyssler et al., 2004). A unique combination of the value, hence, indicates a unique event triggered by the corresponding IP addresses.

Apart from the IP addresses, the third attribute, timestamp, also requires a slight conversion. As the timestamp is represented as date string format rather than a number, an alteration is necessary. The timestamp is normally presented as date vector, consisting of 6 elements specifying year, month, day, hour, minute and second. So, in order to perform the conversion whilst keeping the value of the attribute, "datenum" function from MATLAB was utilised to convert the string or date vector into a serial date number. For example, "2002-08-15 18:16:00" is converted to 731443.7611111112. In addition, in order to prevent the issue of over-fitting caused by the over-specified timestamp, this is very important to modify the last element of the attribute (seconds). And to make it even, this element is set to "00".

5.3.2.2 Concepts of alarm aggregation

The main idea of this stage is to aggregate alerts that belong to the same attack instance (activity) within a particular time window. Prior to the mapping process, a data conversion is carried out, as presented in previous subsection. Since three alarm attributes are selected, a set of three-dimensional input vectors is fed into the classification function.

Using nodes with three-dimensional vectors to build the SOM map directly is likely to be biased to a certain dimension, as different attributes values tend to be in different units. If some vector components have variance which is considerably higher than other components, they will certainly dominate the map formation. Therefore, normalisation is performed to control the variance of the vector components. The experiments utilise variance normalisation method, which is known as *var* (CIS, 2005). This is a linear transformation which scales the values such that their variance is equal to 1.

In the $K-$means algorithm, each attribute is assumed to have the same weight; which then

makes it impossible to know which feature contributes more to the grouping process. Having said that, the value of the attributes' weights can be completely adjusted if the fine-tuning is desired. In this stage, the key attributes of the classification are the IP addresses. To achieve the best outcome, it is necessary to ensure two IP address features, namely addition and subtraction, contribute more to the clustering process than the timestamp feature. In order to do so, a weight adjustment is a necessity.

In order to achieve an ideal weight, a number of classifications were run using three-dimensional input vectors and different weights of IP address were used for each classification. This experiment was conducted to search for the best classification outcome by setting the weights of the IP address attributes to be higher than the timestamp feature. The default weight value of an attribute is 1. If the weight of a particular attribute is set to 2; then the final attribute value itself can be computed by multiplying the weight value to the original attribute value.

Although the main objective of this weight adjustment method is to prioritise the attributes on the classification processes, the alert features should not be over-weighted, which could lead to a biased clustering result. From the observation, it was found that the data started producing inequitable classification outcome once the weights of the source and destination IP address were set to more than 3 times higher than the weight of timestamp attribute. To avoid this problem, the weight values of the IP addresses (w) should only be set to 1 to 3 times higher than the timestamp's ($1 < w <= 3$).

The classification was run several times with the weights of IP addresses range between 1 and 3. The starting weight value is 1.1 and it gradually increases by 0.1 in each iteration (classification). The ideal weight is determined by the best classification outcome. In this system, the finest weight values for the IP address attributes are set to be 1.8.

The number of neurons or the size of the map itself greatly influences the performance of the SOM system. In the classical SOM, the number of neurons should usually be selected as big as possible, with the neighbourhood function maintaining the efficiency and generalisation of the mapping. The increase of the map size, however, could cause the training phase to become computationally and impractically heavy for most applicants. With the aim of gaining the best map result, the system needs to select the number of neurons based on the smallest quantisation and topographic errors, in which case the error values of less than 0.1 are selected. In order to do so, the system runs a loop programme creating maps with different number of units and the programme will be terminated once the map has the quantisation and topographic errors less than 0.1. The quantisation and topographic errors are computed after training to measure the quality of the generated map. However, it is worth noticing that a low quantisation error does not necessarily mean a good result; it might lead to the issue of overfitting. This may happen when the numbers of units are larger than the number of training data (CIS, 2005). Having said that, overfitting is not a real problem since $K-$means is applied as a second classifier. In fact, the implementation of multi-stage classifiers can actually avoid the issue of overfitting (Weijters et al., 1997).

One of the most significant weaknesses of $K-$means clustering is the need to determine the number of clusters prior to classification. The default setting for $K-$means initialisation value $kmax$ (maximum number of clusters), set by SOM Toolbox, is the square root of the length of data. As in the first stage, the real number of the clusters is not known, and it is believed that the data can be classified into more clusters than specified by the default setting above. To affirm this idea,

four chunks of sample data were taken and manually analysed to estimate the expected number of clusters and three of them had clusters reached up to two fifth of the number of input data (alerts). Hence, in order to avoid possible misclassification, the system determines to increase the $kmax$ value (the maximum number of clusters) for $K-$means to half of the length of data (alerts). For example, if there are 3000 alerts fed into the system, a maximum of 1500 clusters will be formed. In this stage, a number of classifications are run using different k (number of clusters), starting from 2 to $kmax$ value. The best number of clusters (k value) is selected based on the lowest SSE value of the corresponding classification.

Again, the problem of overfitting is very common in the subject of data mining and neural network. Such issue occurs when the number of nodes (clusters) is as large as or larger than the number of training cases (CIS, 2005). Since the number of training data used in the experiment is two times more than the clusters ($k = data$), the network is unlikely to suffer from overfitting.

Appendix Cand D provides the pseudocode and full codings for stage one and two correlations.

5.3.3 Stage 2 - False Alarm Classification

Having correlated the related alerts into a number of clusters, the second stage of classification is carried out to label the alerts into true and false alarms. The main objective of this false alarm classification stage is to obtain a better alarm management by reducing the number of false alarms generated before being presented to the administrator.

5.3.3.1 Alarm attributes

Similar to the first mapping, the alert attributes are selected and pre-processed prior to the classification. However, in this case, the outcome of the first classification will be fed into the second alarm classifier; meaning that the clusters generated in stage 1 correlation will act as inputs for the stage 2 classification. In this stage, seven alert features are selected from each cluster. Those are the number of alerts, number of signatures, port number, protocol, priority, time interval and the number of events. The attributes are carefully selected to represent the dimensions of the input data and to describe the inherent relationship between alerts. Table 5.1 presents a brief description of the selected alarms' attributes and their data collection methods.

5.3.3.2 Concept of true and false alarm classification

In this false alarm classification module, the final result of the classification largely hinges on the dimensions or the attributes applied in the SOM mappings. Typically $K-$means algorithm treats all features fairly and distributes the weights on all attributes equally. The features' weights can be derived based on the importance of the feature to the clustering quality. The higher the attribute's weight, the more the contribution it has on the clustering process. Amongst the seven features, two attributes, namely the number of events and the time intervals between events, are deemed to be the most influential ones. So, to ensure such features contribute more to the clustering processes than the remaining five, the fine-tuning was carried out by properly adjusting the attributes' weights. Using the similar method as carried out in the previous alarm aggregation stage, the ideal

Table 5.1: The interpretation and data collection methods of the alarm attributes for second stage classification

ALERT FEATURES	DESCRIPTION	COLLECTION METHODS
No of alerts	Total number of alerts grouped in one cluster	N/A
No of signatures	Total number of signature type in a cluster	N/A
Protocol	Type of traffic from event triggering the alerts	There are only three values that can be assigned to this feature. Alert with the protocol number below 255 is assigned to a value of 1 and 3 for protocol number 255. If there are two types of protocol number found in a cluster, the value is set to 2.
Port number	Only the service port number is applied in the classification.	If the alert contains a well-known port number (< 1024), the value will be set to 1; if not (≥ 1024) value of 3 will be given. If the cluster has two types of port numbers, then the value will be set to 2.
Alert priority	Criticality of the alerts. There are 3 types of alert priority, namely 1st, 2nd, and 3rd.	Based on the type of signature, alert with the 1st priority is assigned to a value of 300, 2nd to 200 and 3rd to 100. If multiple signatures are found in a cluster, the priority value for each signature could be added together.
Time interval	Time interval between events[‡] from a particular signature	Should an alert signature occur in 3 different events in a particular time frame, the mean of the time interval between each event is calculated. This attribute is computed in seconds. However, if there are multiple signature types in one cluster, the highest time interval will be selected.
No of events	The number of events[‡] in which a particular alert signature is triggered within a time frame, for example, every two hours, one hour or half an hour	If there are multiple signature types in a cluster, the lowest no of events is selected.

[‡] One event may contain one or multiple types of alert signature, which are triggered by a particular activity or attack.

weights were then computed to be 2.5 for the 6th attribute, namely time interval and 2.8 for 7th attribute, namely number of events.

To overcome the weakness of $K-$means clustering, the system generates 500 randomised trials or iterative classifications, involving randomising both the initial k clusters and the data order. The number of clusters (k value) of the stage 2 correlation is set to 2 as the classification aims to generate two meta-clusters, namely the true and the false alarms. In each trial, the input data order is randomised and the first two input data are selected as the initial centroids for the $K-$Means algorithm. The objective of randomising the data order is to simply have distinct classification

results with different initial clusters for 500 interations.

In $K-$means, the most essential approach to determining the best classification result is by looking into its SSE value (MacQueen, 1967). Hence, this feature is taken as one of the selection criteria to select the finest cluster solution. The sum of squared error refers to the least distance between the data and the corresponding cluster centroid. A map is considered equal to other maps if they have the same SSE value.

For $K-$means algorithm, the lower the sum of squared error, the more accurate the classification should be. This theory, however, in some cases, might not apply to the system. In the map with the lowest SSE value, the algorithm tends to assign the centroids to the data points with the farthest distance; generating two clusters with highly unbalanced cluster sizes. This definitely indicates two poor outcomes; either a tighter security level with a lower reduction rate or a loose security level with the risk of false negatives. Such an issue clearly demonstrates the trade-off between maintaining the security level and the need for reducing the false alarms.

In view of this trade-off issue, thresholding is required to balance the security issue and the alarm reduction. Since the randomised experiments were used to select the best cluster solution, evaluating the frequency rate of each solution (map) is necessary. So, instead of merely focusing on the lowest SSE value, the best map is also selected based on its frequency rate (frequency distribution). The concepts of the SSE values and frequency rate are explained as follows:

1. Each classification yields a SSE (sum of squared error) value, which can be computed using Euclidean distance of data inputs to the mean of their corresponding clusters. The details have been described in subsection 5.2.2.

2. The frequency rate refers to the count of the occurrences of a SSE value within the 500 classifications.

To use both elements to assess the quality of the generated maps, an experiment, consisting of five classifications, was carried out using five set of sample data to examine the frequency rate of the SSE value. From the observation, a cluster solution (map) that had a SSE value with a frequency rate above 0.6 (300 out of 500) had the best classification result compared to other solutions. From the study, it is evident that a solution with a high occurrence rate (reassuringly occurs in at least the third fifth of the random trials) generates a better grouping compared to those with low frequency rates. The higher the frequency rate of a SSE value from a map, the more stable the map is. The best classification solution is, therefore, selected based on the highest frequency rate and set the thresholding value to 0.6. Any map with a frequency rate exceeding the thresholding value (0.6), will be automatically selected as the finest choice without any further evaluation.

Conversely, if the highest frequency rate falls below the value (it does not dominate other solutions), further evaluation will be required in this case. To find which of the maps are worth evaluated, it is necessary to set another threshold to select the dominant solutions. The second thresholding (s), which is derived from a standard deviation of the SSE or map's probability distribution, will determine which of the cluster solutions need further investigation. The standard deviation represents the average variation of the frequency rates from the mean distribution. Again, by observing the results of five sample classifications, it is concluded that the maps with frequencies range from t to $t - s$ are likely to produce better clustering results compared to those with low

frequency rates. So, for this reason, only those solutions with frequency rates that fall between t (highest frequency) and $(t - s)$ are evaluated.

Hence, to conclude this, the procedure of the map selection is described as follows:

```
Step 1   :   Check if any map or SSE has a frequency rate above
             0.6. If yes, go to Step 4.1.
Step 2   :   Check if the highest frequency rate exceeds other
             solutions (t-s > second highest frequency rate). If yes, go to
             Step 4.1.
Step 3   :   If the number of maps whose frequency rate are between
             t and t-s is equal to 2, go to Step 4.2; else calculate the
             average SSE value of the maps whose frequency rates are between t
             and t-s (as shown in equation 5.2). Go to Step 4.3
Step 4.1:    Take the map with the highest frequency rate as the
             best cluster solution.
Step 4.2:    Take the map with the lowest SSE value as the best
             map choice
Step 4.3:    Select the cluster solution with the SSE value
             closest to the average SSE as the best map choice.
```

$$\frac{1}{n}\sum_{i=1}^{n} SSE_i, \tag{5.2}$$

where n is the number of map solutions whose frequency rates range from t to $t - s$.

5.4 Algorithms

The complete algorithms of the whole correlation process can be defined as follows:

5.4.1 Stage 1 Correlation

len = the length of input data (alerts)
$munit = len + 100 - munit$ is the initial size of map (number of units in a map)

1. Extracting alert attributes - source, destination IP addresses and timestamp are extracted from each input alert
   (a) The output of this process is a set of 3-dimensional input vectors
2. Normalising the input vectors using var method from MATLAB
3. Increasing the weights of source and destination IP addresses to 1.8
4. Determining the best size of map units based on the smallest topographic and quantisation errors

(a) Creating a SOM map with the number of units is equal to munit

(b) Calculating topographic ($terr$) and quantisation errors ($qerr$)

(c) While $terr > 0.1 \parallel qerr > 0.1$

- Increase $munit$ by 10
- Creating a SOM map with the number of units is equal to $munit$
- Calculating topographic and quantisation errors

(d) End

5. Training the created SOM map with the input vectors using SOM algorithms

   (a) The output of this process is a U-Matrix map with the plotted input vectors.

6. Clustering the vectors plotted in the SOM map into several pre-defined clusters using $K-$means algorithm

   (a) $kmax$ is set to $1/2$ of len

   (b) Set SSE to the largest number of the floating point number

   (c) Set itr to $kmax$

   (d) Run the classifications several times using different k values ranging between 2 and $kmax$
   While $itr > 1$

   - Randomising input data order
   - Assigning the first itr input data as the centroids
   - Clustering the data using $K-$Means algorithm
   - Calculating SSE value
   - If the $SSE < err$,
     Set err to the SSE
     Set k to itr
   - Decrease the itr by 1

7. The final output of this stage is the input vectors being grouped into k clusters

5.4.2 Stage 2 Correlation

$len = k$ (number of clusters formed in the previous stage)
$munit = len + 100$ – munit is the initial size of map (number of units in a map)

1. The clusters formed in stage 1 correlation will become the input of stage 2 correlation.

2. Extracting alert attributes- no of alerts, no of signatures, protocol, port number, priority, no of events and time interval are extracted from each cluster

   (a) The output of this process is a set of 7-dimensional input vectors

3. Normalising the input vectors using var method from MATLAB

4. Tuning the weights of no of event to 2.8 and time interval to 2.5
5. Determining the best size of map units based on the smallest topographic ($terr$) and quantisation errors ($qerr$)
   (a) Creating a SOM map with the number of units is equal to $munit$
   (b) Calculating topographic ($terr$) and quantisation errors ($qerr$)
   (c) While $terr > 0.1 \parallel qerr > 0.1$
      - Increase $munit$ by 10
      - Creating a SOM map with the number of units is equal to $munit$
      - Calculating topographic and quantisation errors

   (d) End
6. Training the created SOM map with the input vectors using SOM algorithms
   (a) The output of this process is a U-Matrix map with the plotted input vectors.
7. Clustering the vectors plotted in the SOM map into several pre-defined clusters using $K-$means algorithm
   (a) Set k to 2
   (b) Set itr to 500
   (c) Run the classifications 500 times using different k values ranging between 2 and $kmax$
   While $itr > 0$
      - Randomising input data order
      - Assigning the first k input data as the centroids
      - Clustering the data using $K-$Means algorithm
      - Calculating and storing the SSE
      - Decrease the itr by 1

   End
8. Selecting the best map from the 500 maps generated based on the SSE value and the frequency rate
9. The final output of this stage is the input vectors (clusters) being grouped into 2 meta-clusters

5.5 Experimental Results

In order to evaluate the performance of this proposed system, two experiments were carried out. For the experiments, two types of datasets were used; the public (DARPA 1999 data set) and the private (University of Plymouth data set). Table 5.2 presents the properties of data set selected for the experiments. Following subsections present the experimental results of the false alarm classifier.

Table 5.2: Properties of DARPA and Plymouth Private Data Sets

		DARPA		PLYMOUTH	
		PART 1	**PART 2**	**PART 1**	**PART 2**
	No of Alerts	1224	1838	330	2226
STAGE 1	Map Units	1333	1924	437	2385
	Errors	$Q = 0.001$ $T = 0.019$	$Q = 0.009$ $T = 0.041$	$Q = 0.050$ $T = 0.048$	$Q = 0.003$ $T = 0.011$
	k-value	612	919	165	1113
STAGE 2	Map Units	190	299	290	260
	Errors	$Q = 0.048$ $T = 0.012$	$Q = 0.097$ $T = 0.054$	$Q = 0.097$ $T = 0.043$	$Q = 0.044$ $T = 0.077$
	k-value	2	2	2	2
RESULT		$FA = 1131$ $TA = 93$	$FA = 1297$ $TA = 541$	$FA = 260$ $TA = 70$	$FA = 2139$ $TA = 87$

Q is Quantisation Error, T is Topographic Error, FA is False Alarm and TA is True Alarm.

5.5.1 DARPA 1999 Data Set

Due to the criticisms that were raised over the DARPA data set, questioning the use of synthetic data to picture a real world network (as explained in Chapter 4), the experiments used not only a DARPA data set, but also a private data set, which will be discussed later in subsection 5.5.2. As the main objective is to facilitate alarm management for the administrator, the proposed technique is designed to process the generated IDS alerts in maximal two-hour alerts. With the aim of providing a quick overview of the system performance, the experiments randomly selected and evaluated only a chunk of DARPA 1999 data set as the input of the IDS system. In this case, only 4 hours data from week 4 DARPA testing data set is fed into the system.

To obtain a set of network alarm data for the classification system, Snort was run (Caswell and Roesch, 1998) under Linux Fedora 7 against the DARPA data set. In order to facilitate the analysis of IDS alerts, a front-end tool Basic Analysis and Security Engine (BASE, 2009) was then utilised as the intrusion analyst console. Regarding the neural networks, the SOM-based and $K-$means system is implemented on the SOM Toolbox 2.0 (CIS, 2005) which is run on MATLAB 7.8.0.

For DARPA data set, 4 hours of data (total 3,062 alerts) was extracted from the first day of the 4th week testing data and was evaluated as two separate inputs. Figure 5.3 presents the result of the stage 1 DARPA classification. The maps as shown in Figure 5.3, 5.4, 5.5and 5.6 are examples of U-Matrix maps generated by SOM algorithms. With the pre-defined map units and input vectors presented, the maps are then created and trained by the mapping algorithms.

A total of 790 clusters were generated in the first part of classification (first 2 hours); shown on the left map. Interestingly, only 203 of them were active whilst the rest were considered dead centres. Active cluster is a cluster that contains data whilst the dead centre is a centroid that has no members or associated data. Similarly, from 605 clusters generated in the second part (shown on the right map), only 86 classes were active. This seems obvious that $K-$means clustering tends to generate a significant number of dead centres. Enhancing the $K-$means performance, however, is out of the scope of the study and is not discussed in this thesis.

In general, the classification demonstrated a reasonable outcome. Approximately 93% of data from the first part of the classification was mapped and classified into the correct clusters, that is

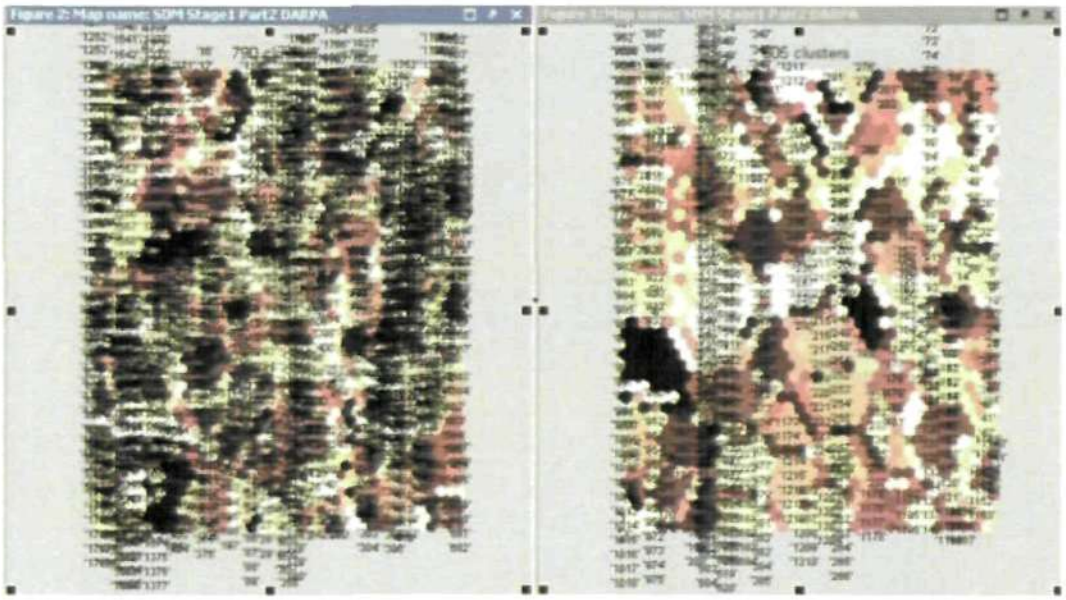


Figure 5.3: Stage 1 classification using DARPA 1999 data set

Table 5.3: SSE and Frequency Rate from DARPA Data Set Part 1

MAP	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
SSE	6.3276	6.3350	6.5053	6.5056	6.8716	6.8721	6.8731	6.8767	6.9091	7.6474	7.6807
Frequency	0.3620	0.3320	0.0620	0.0640	0.0040	0.0040	0.0040	0.0440	0.0020	0.0560	0.0660

Table 5.4: SSE and Frequency Rate from DARPA Data Set Part 2

MAP	**1**	**2**	**3**
SSE	2.4721	2.9135	4.0591
Frequency	0.7860	0.1540	0.0600

accuracy = 0.93. Conversely, 0.9 was revealed from the second classification. In terms of clustering accuracy (the number of clusters with the correct data), the first classification showed 0.86 accuracy, whilst second classification revealed 0.81.

In the second stage, there were eleven maps (cluster solutions) that were produced in the first part of DARPA classification, as shown in Table 5.3. Only two clusters, the true and the false alarm classes, were desired in this stage. The result shows on the left map from Figure 5.4 is corresponding to criterion value in Table 5.3.

Based on the map selection procedure, it is obvious that the solution with the highest frequency rate in Table 5.3 does not conform to the first and second criterion rules (MAP 2 has the frequency rate higher than $t - s$; $t = MAP1's frequency rate$; $t = 0.362$; $s = 0.129$). Since only two maps (MAP 1 and MAP 2) have frequency rates higher than $t - s(0.362 - 0.129 = 0.233)$, the one with the lowest SSE value is selected. In this scenario, MAP 1 is selected as the best map choice (presented on the left side of Figure 5.4). On the other hand, the second part of DARPA classification, which is shown on Table 5.4, presents 3 possible cluster solutions.

The solution with the highest frequency (MAP 1) is automatically chosen as the best map since

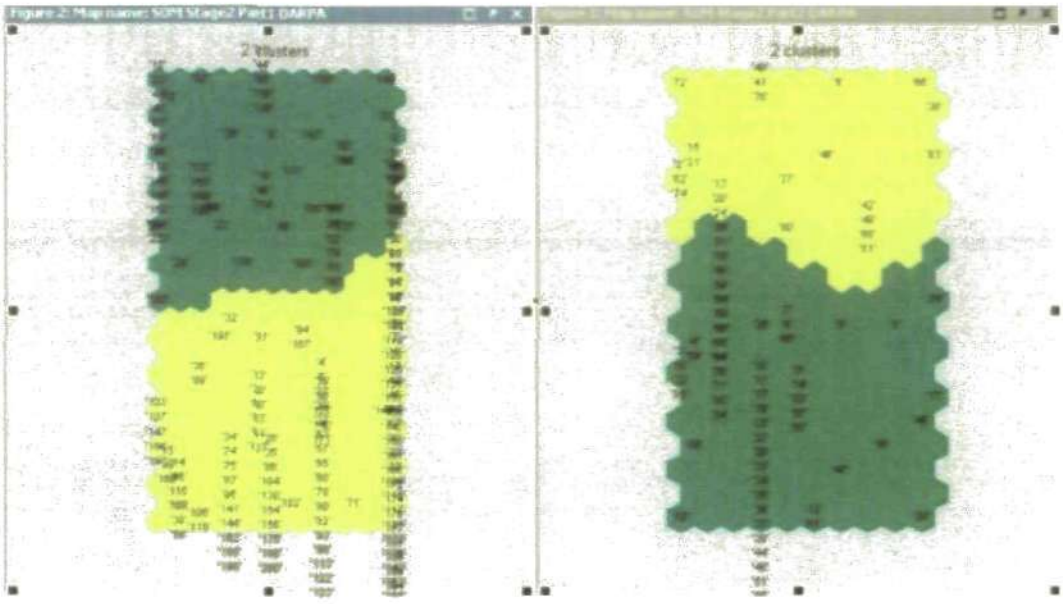


Figure 5.4: Stage 2 alarm classifier using DARPA data set

Table 5.5: Result comparisons

No	Proposed by	Method	Data Set	False Alarm Reduction Rate
1.	Julisch, 2001	Root cause clustering	Real network	82%
2.	Al-Mamory and Zhang, 2009	Root cause clustering	Real network, DARPA 1998 and 1999	74%
3.	Perdisci et al., 2006	Clustering: meta alarms	DARPA 1999	64.6%
4.	Sadoddin and Ghorbani, 2009	Incremental mining of frequent structured patterns	DARPA 2000	96%
5.	Khanchi and Adibnia, 2009	Alert feature frequencies	DARPA 2000	71%
6.	Spathoulas and Katsikas, 2010	Neighbouring alerts and alert frequencies	DARPA 1999	75%

the frequency rate (0.786) has exceeded the first thresholding value. The mapping result is presented on the right map in Figure 5.4.

In this context, the proposed system is considered effective in reducing the number of false alarms; with 95% being correctly labelled in the first classification, whilst the second categorisation reduced approximately 99% of the total false alarms. Those alarms located in the upper portion were labelled as true alarms, whilst the lower portion was for the false alarms. The system also appears effective in detecting false alarms generated by noisy traffic such as the ICMP traffic (ICMP Ping and echo reply) and the web-bug alerts, which formed the highest number of false alarms triggered in the experiment (as discussed in Chapter 4).

Table 5.5 shows the performance results of other existing false alarm classification or corre-

lation methods. From this preliminary experiment, it is obvious that the proposed system has outperformed other existing methods. However, due to the small data set used in this experiment, the comparison is considered biased at this stage. Therefore, a further evaluation should only be conducted to assess the system performance at the final stage, that is using a complete data set. This will be presented in Chapter 7, Section 7.8.

5.5.2 University of Plymouth Private Data set

The UoP private data used in this experiment is the same as the one used for fine-tuning, as discussed in Chapter 4. Similar to the DARPA data set, 4 hours data from University's network data (2556 alerts) was analysed using two separate inputs. Figure 5.5 presents the result of the classifications.

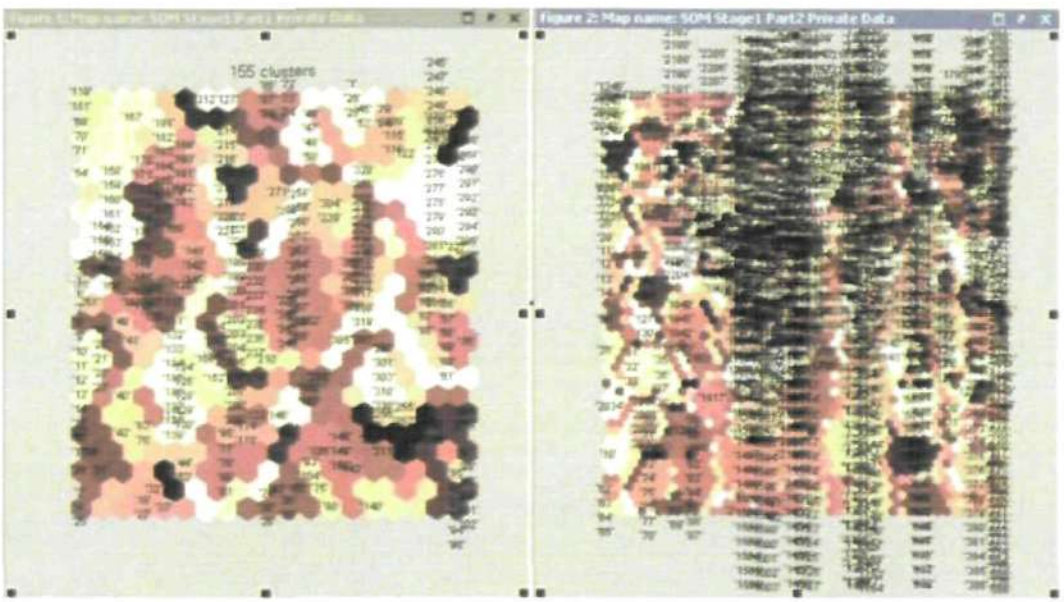


Figure 5.5: Stage 1 classification using University of Plymouth data set

The classification from this network data shows a slightly better result compared to those from DARPA data set. Approximately 0.92 and 0.94 are computed for the first and second classification, whilst the cluster accuracy accounts for 0.89 and 0.93 respectively.

Unlike the stage 2 classifications on DARPA data set, the classifications on private data set reveal quite a straightforward result. The computation of the average SSE value is not required in this scenario as the highest frequency rates from both classifications shown on Tables 5.6 and 5.7 conform to the first criterion. In view of this, the solutions with the highest frequency rate are determined to be the best maps. In addition, the selected maps have the lowest SSE value among all cluster solutions. The final results of both classifications are presented in Figure 5.6.

As for the private data, the classification reveals that about 78.8% of false alarms have been identified in the first map, whereas 96% of them have been detected in the second mappings. It is notable that the system has shown promising result in filtering all hectic and unnecessary alerts triggered by the IDS. For example, the alerts from WEB-IIS view source via translate header and

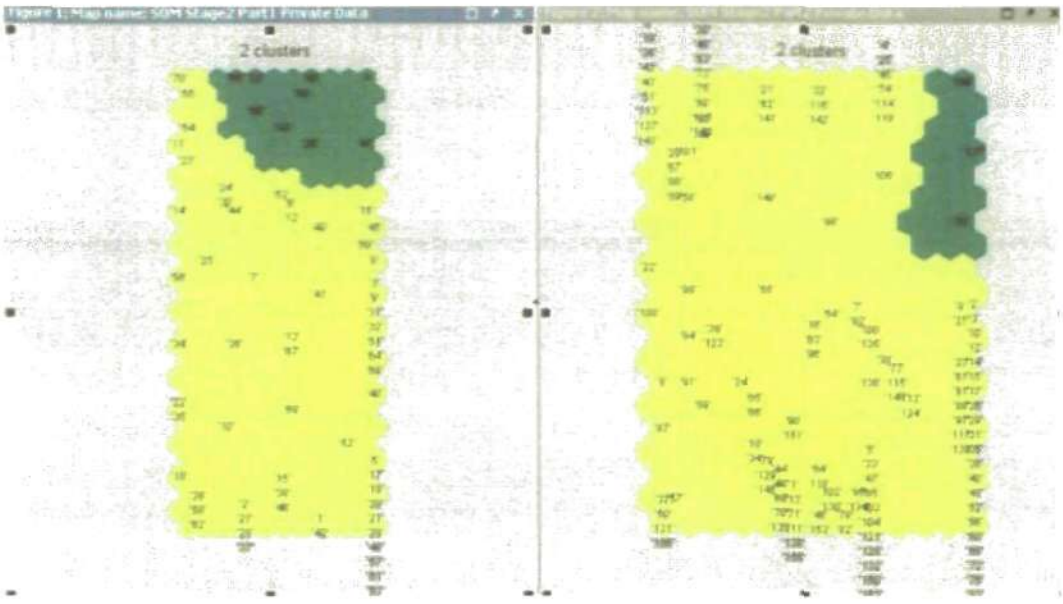


Figure 5.6: Stage 2 alarm classifier using private data

Table 5.6: SSE and Frequency Rate from PLYMOUTH Data Set Part 1

MAP	**1**	**2**	**3**
SSE	2.7161	2.8643	2.8645
Frequency	0.6360	0.1820	0.1820

Table 5.7: SSE and Frequency Rate from PLYMOUTH Data Set Part 2

MAP	**1**	**2**	**3**	**4**
SSE	3.0918	5.3372	5.6675	5.6784
Frequency	0.8180	0.1220	0.0320	0.0280

WEB-MISC robots.txt access signatures, which caused 82% of false alarms from the entire private data (as discussed in Chapter 4).

The suggested alarm filtering system is believed to significantly outperform other existing methods. Unlike many proposed systems that need to be trained with a considerable volume (gigabytes) of attack-free data, this system applies unsupervised training to train the classifier; hence no attack-free data is necessary. In terms of its configuration, this approach is considered efficient enough as it is easy to set up and no knowledge of the attacks required to filter the alarms. Moreover, the system's filtering processes are independent from the intrusion detection process. As to its performance, the system does not only provide a better alarm management, but also shows the relationship between the generated alerts, thus enabling administrator to discover the potential attack scenarios.

5.6 Conclusions

This chapter has discussed the proposed data mining techniques used in identifying and subsequently reducing the number of IDS false alarms. Unlike other existing alarm correlation methods, which focused on either discovering attack patterns or identifying true and false alarms, the proposed system aims to achieve both objectives. To do so, a two-stage classification system using the combination of two data mining techniques, namely SOM and $K-$means clustering were proposed. The first stage classification was developed to properly correlate alerts related to a particular activity. All alerts, regardless the signature type, triggered by a single event are mapped and grouped into one cluster. In addition, the main objective of the second stage is to subsequently label all clusters produced in the first classification into groups of true and false alarms.

To verify the idea, preliminary experiments were carried out with two different datasets; the 1999 DARPA IDS evaluation data set and the private network data. The result shows that more than 90% of false alarms from DARPA data set were filtered without ignoring the true alarms whilst approximately 87% of false alarms from private data set can be correctly identified. Despite the lower false alarm detection rate in private data set than the DARPA data set, this system has demonstrated its effectiveness in filtering all noisy and unnecessary IDS alerts, which have usually contributed to more than 50% of false alarms from the common IDSs.

To further investigate the characteristics of the proposed system, the following chapter presents a novel architectural framework along with the components of the proposed system.

6 A Conceptual Architecture for an Automatic Alarm Correlation System

6.1 Introduction

Having developed a preliminary version of the alarm reduction system, it is of importance to design an automated correlation system to facilitate a review of IDS false alarm trends and to tune IDS signatures for future detection. The basis for developing a standalone IDS alarm correlation system is the fact that current alert management tools do not have the ability to verify the validity of the generated alarms, that is whether it is a true or false alarm.

Based on the findings from the alarm reduction system, this chapter contemplates the issue of designing a generic alert correlation system that can facilitate IDS alarm validation as well as tuning or management of IDS ruleset. This chapter presents the fundamental components and the architectural framework of the proposed correlation system. Section 6.2 discusses the underlying concept of the system, whilst its operational characteristics are presented in Section 6.3. The main modules of the system are introduced in Section 6.4; followed by conclusions in Section 6.5.

6.2 SOM K–Means Alarm Reduction Tool (SMART)

An alert management tool is created based on the supervision of the collected logs or alerts and the representation of the data in a more comprehensible manner, for example, charts and statistical figures. The tool aims to offer various features, including an in-depth query or analysis on the alerts and also the generation of a final security report for the security administrators. Conventional management tool cannot verify the accuracy of alerts, the SOM K–Means Alarm Reduction Tool (SMART) presents entirely new work in this context.

SMART is based on the concept of the Security Information and Event Manager (SIEM). The rationale behind the application of SIEM as the underlying model relates to its ability to perform both a correlation (event investigation) and a report production. In addition, the complex analysis and the outstanding alerts management or presentation of SIEM do not only assist in identifying anomalous events but also help reducing the information overload. In a sense, this is considered the main benefit of SIEM and also the key objective of the proposed SMART system.

In terms of the processing ability, the SIEM may also benefit from having available to it information from various systems at both network and application level, information of the event severity, and also the knowledge of state of the protected network vulnerabilities. In a way, this can be regarded as a limitation of this research; in the sense that the SMART system merely focuses on the

alert correlation from a single IDS. However, proving the feasibility of the SMART system and its concepts is the main focus of this study, and adapting these approaches to allow more input data from more than one security tool highlights a scope in which this research can be extended in the future.

SMART enables the security administrator to correlate and filter false alarms generated by Snort IDS. The result of the correlation is then fed into a database, thus providing an efficient way for the administrator to further query and analyse the correlated alarms. The elements of SMART are illustrated in Figure 6.1 and discussed below.

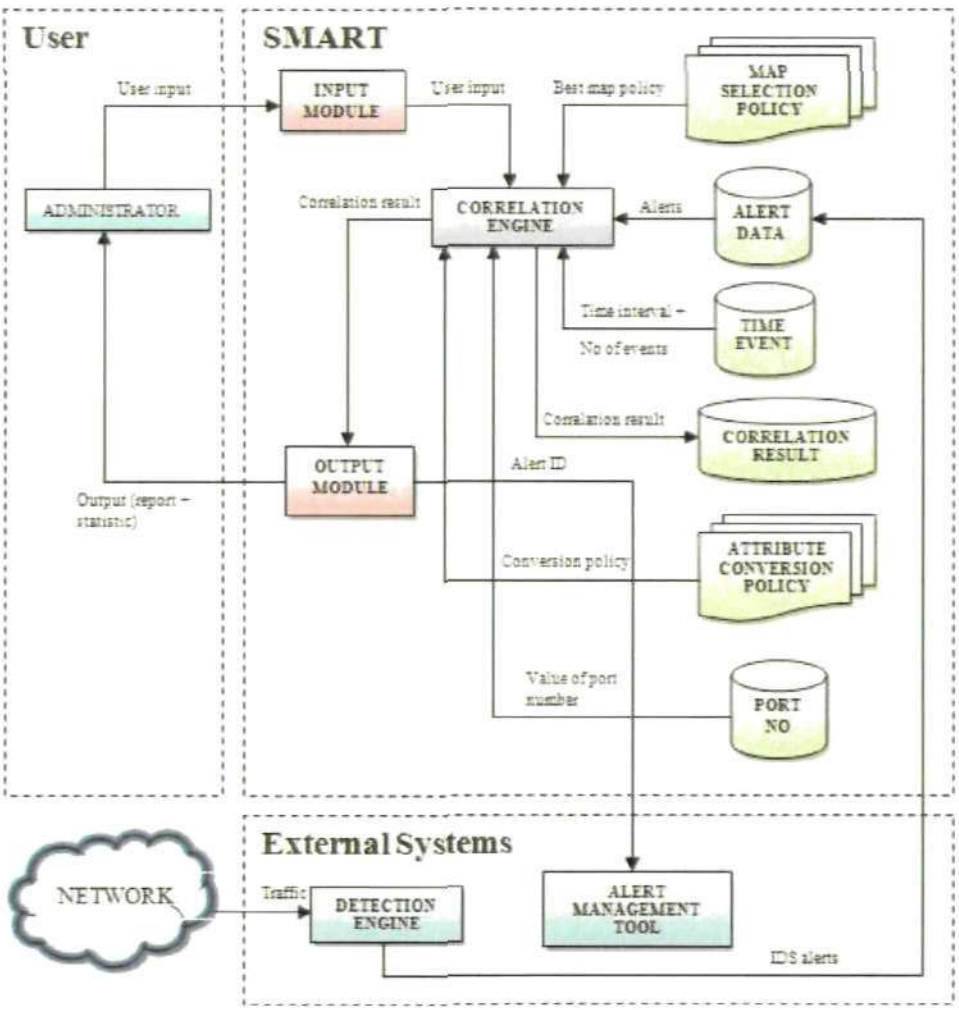


Figure 6.1: SMART architecture

SMART comprises the following elements:

1. **Correlation Engine:** This is the main engine of the proposed system. The *Correlation Engine* performs two stages of clustering processes; including aggregating as well as filtering the false alarms. With the information obtained from the alert and time event database, the correlations are carried out by applying the best map policy.

2. **Input Module:** The *Input Module* is a front-end interface (GUI) of the system that enables

the user to provide input to the correlation as well as to instigate the clustering processes. In addition, it also allows the user to monitor the progress of the correlation.

3. **Output Module:** Similar to the *Input Module*, the *Output Module* is a front-end interface that allows the user to view the classification results. Apart from presenting the final outcome, it generates statistical charts or graphs comparing both true and false alarms over a particular period of time. Moreover, such module enables the user to further view the alerts details, for example, alert payload, by redirecting the user to the *Alert Management Tool*.

4. **Map Selection Policy:** In order to select the best map from the randomised trials, the rules that describe the characteristics of the best map are then specified in the *Map Selection Policy*.

5. **Alert data:** The *Alert data*, which contains complete information about the alert attributes, such as source/destination IP addresses, signature name, timestamp, port number, protocol and etc, is a database generated by the Detection Engine (IDS). The information stored in this database will be extracted by the *Correlation Engine* to perform the alert classification.

6. **Time Event:** Before performing the second stage correlation, the sixth and the seventh attributes, namely time interval and number of events for each signature, are computed for every two hours, one hour and half an hour and stored in the *Time Event*. Once the correlation is carried out, the values of the sixth and the seventh alert attributes are retrieved from this file.

7. **Correlation Results:** The *Correlation Results* hold the information of all classification results; including those from stage 1 and stage 2 correlations. In particular, the final results consist of the figures from the correlations, the cluster index the alerts belong to and the alert status, namely true or false alarm.

8. **Attribute Conversion Policy:** *Attribute Conversion Policy* holds all information needed to convert the alert attributes into appropriate values (format) used for the correlation. The details about the policy applied to each attribute are described in Table 5.3.

9. **Port No:** *Port No* lists all official port numbers (in other words, those that have been registered with IANA). The file is used to assist the system in determining which of the ports, namely source and destination port, from the captured traffic is a service port number. Significantly, only the service port is used as the attribute of the correlation.

In addition to the components of SMART, the external systems, which are adopted from the open source market, consist of the following tools.

1. **Detection Engine:** The *Detection Engine* refers to the IDS used to monitor the network traffic. In this case, the Snort IDS was used.

2. **Alert Management Tool:** The *Alert Management Tool* is a front-end tool for IDS, which allows the administrators to organise as well as investigate the IDS alerts. In this study, BASE was used as the alert management tool.

Table 6.1: Alert Attributes of Stage 2 Correlation

No	Attributes	Selected YES/NO
1.	Number of alerts	✓
2.	Number of signatures	✓
3.	Alert priority	✓
4.	Timestamp	✗
5.	Source IP address	✗
6.	Destination IP address	✗
7.	Protocol	✓
8.	Service port number	✓
	Additional attributes	**Selected YES/NO**
9.	Time interval	✓
10.	Number of events	✓

6.3 Operational Characteristics of SMART

There are several distinct attributes of the system that apparently distinguish it from other technologies. These are

- an ability to perform an attribute-based alarm correlation approach
- its strength in aggregating and classifying alerts based on time windows
- an ability to classify alerts into true and false alarms
- lastly, an ability that allows the administrators to exhaustively evaluate and examine the IDS alerts in a more comprehensible statistical view

The following sub-sections discuss these operational characteristics in more detail.

6.3.1 Offer an Attribute-based Alarm Correlation approach

The main goal of performing attribute-based correlation is to classify IDS alerts based on the most relevant features that highlight their behaviours. Most importantly, it is worth knowing that such an approach has been highly accepted or commonly used in the study of implicit correlation.

Since the IDS alerts have numerous attributes ranging from the IP addresses to the length of the packet, selecting the best attributes that represent the characteristic of the alert is a challenging task. Therefore, a feature selection was carried out to choose the most appropriate attributes to symbolise an alert or even a group of alerts. As previously mentioned, seven attributes have been selected in the second stage to represent a cluster of alerts. The seven attributes are made of five main attributes and two additional attributes (as listed in Table 6.1) The rationale behind this selection is to choose the most relevant and basic features that properly represent the alert clusters. Following table summarise the selected attributes in the second stage correlation.

It is worth noticing that the timestamp and the source/destination IP addresses are not selected in this phase since these have been applied in the first correlation.

The underlying idea behind the application of the alert attributes in the IDS alert correlation is to measure the similarity between the generated alerts. The selected attributes need to be converted into correct values before any correlation is carried out. The closer the attributes values between two alerts, the more likely they are grouped into the same cluster.

6.3.2 Evaluate and Aggregate Alarms based on Time windows

Another important feature of the correlation system is the flexibility of the time windows applied in the correlation process. To alleviate the workload of the correlation, the system is designed to process or correlate the input data based on the predefined time frame. Generally, the number of the alert data presented to the clustering system could be massive; thus rendering the correlation unfeasible in the prevailing circumstances. In view of this constraint, it is necessary to limit the amount of data fed into the system. So, instead of running one correlation for the entire data set, it would be much more practical to run a correlation over a particular time period, for example every one or two hours.

In order to alleviate the correlation processes whilst maintaining the value of it, the system offers three interval options, namely every half an hour, every hour and every two hours. Such choices are given to help the administrators determine the most suitable time frame. In theory, the bigger time frame the better the correlation should be. This is because more information or alerts are processed in the correlation. However, it is worth remembering that selecting a big time frame (in other words, higher number of alerts being processed) is not necessarily the best choice; in fact it might cause several operational constraints including high processing time and high system memory consumption. By contrast, the selection of a small time window could reduce or even obliterate the value of the correlation itself. For instance, alerts that belong to the same attack instance could be regarded as alerts from two separate activities due to the extreme data splitting. Such issue clearly indicates the trade-off between performance and accuracy of the correlation.

Due to the high memory usage in the unsupervised-based correlation process (will be described in Chapter 8), a maximum of two hours data is run per correlation. The two hours correlation is the highest time frame available. Although three options are available, the highest interval is the most recommended one in order to achieve an optimal result.

As discussed in Chapter 5, the additional features, namely the time interval and number of events, are implemented as the most influential and significant inputs. Thereby, the administrators are given the flexibility to select the most suitable interval options available for their alert correlations. The correlation file (database) is retrieved by the system according to the decision made by the administrators. For example, if "every 2 hours" interval is selected for the correlation, then the sixth and the seventh alert attributes from "every 2 hours" Time Event file will be retrieved and applied in the correlation.

6.3.3 Classify Alerts into True and False alarms

One of the most significant features and also the main goal of the system is the ability to classify alerts into groups of true and false alarms. Having computed two leading alert attributes, time interval and number of events, and five other attributes, the second stage correlation is carried out to group the stage 1 clusters based on their 7 attributes into classes of true and false alarms. All

alerts from clusters (from stage 1 correlation) that are grouped into false alarm class are flagged as false alarms, otherwise they are true alarms.

6.3.4 Offer a Flexible and High Level of Alarm Comparison using IDS signatures

Unlike other conventional alert management tools, the proposed system offers a statistical tool that allows the administrator to further evaluate the trend of IDS alarms and to do a comparison between the true and false alarms. This distinctive tool introduces an extended feature, which offers numerous statistical charts, to assist the administrator with the alert analysis.

Apart from the chart creation tool, the system also enables the administrators to perform an alarm comparison using IDS signatures. The generation of true and false alarms based on the IDS signature rules is presented; this facilitates the administrator in identifying which of the signature rules can be disabled and which of them need to be further tuned. The key objective of this feature is solely to give an insight into the pattern of IDS signatures and also guidance for the administrators to properly tune signature rules for a future detection. If the signature has raised only the false alarms, then it is reasonable to remove that signature from the detection rules. Conversely, if it has triggered both true and false alarms, then a fine-tuning is required.

As for the comparison charts, there are five types of statistical charts proposed by the SMART system. The charts are described as follow:

- Time Vs False alarms

  In this category, the charts present the number of generated false alarms in every hour, day and month.

- Time Vs True alarms

  Similar to the previous category, the number of true alarms is computed and charted in every hour, day and month

- True alarms Vs False alarms

  This group does a comparison between the number of true and false alarms in every hour, day or month. In other words, this is the combination of the first and second category.

- Time Vs False signatures

  The charts represent the number of false alarms triggered in every hour, day and month by each signature rule.

- Time Vs True signatures

  This category is similar to the previous one except that it presents the number of true alarms instead of the false alarms.

In addition, the SMART system also features a tool that allows the administrators to specially evaluate the signature rules that have triggered both true and false alarms. A chart is also created to plot the alerts according to their source IP addresses and the hour of occurrence. On top of the graphical view, the system will also provide two tables containing the cluster index, time interval

and the number of events from a particular signature rule. Therefore, with the information provided, the administrators are now able to learn the signature patterns and perform a signature revision if a fine tuning is needed.

6.4 SMART Modules

As previously discussed, there are four key components that have essentially formed the architecture of the SMART system, namely the input module, the output module, the correlation engine and the storage. Two of these modules are related to the front-end interfaces while the others are the back-end system. In order to obtain a better understanding about the structure of the system, the following sub-sections describe the elements involved in each of the four components.

6.4.1 User Input - User Interface

To effectively run the system, there are two user inputs required to initiate the correlation engine. The first input will be the starting and the ending timestamp of the alerts processed whilst the second input will be the time frame (interval) for each correlation. In the main application interface, the administrators are prompted to specify which alerts to be processed by entering the alerts timestamp and also prompted to select one from three interval options provided before a correlation can be executed.

6.4.2 Correlation Engine

As the main clustering algorithms reside within the correlation engine, this module is considered to be the most significant as well as the core component of the SMART system. In order to provide an overview of the key elements relevant to the main back-end system, the attributes of the correlation are presented in Table 6.2.

Table 6.2: Correlation Attributes

No	Attributes	Description
1	Alarm ID	Alarm ID is a unique identifier of alerts that differentiate one alert from others. In most correlation tables, the attribute is used as the primary key that distinctively identifies each alert. Based on the occurrence time of the alerts, each alarm is assigned a unique ID in ascending order, starting from 1.
2	Source and destination IP addresses	As described previously in Chapter 5, these attributes are applied in the first stage of correlation. The values of both attributes are converted into their decimal notations before being applied in the correlation.

Continued on next page

Table 6.2 – continued from previous page

No	Attributes	Description
3	Timestamp	The timestamp indicates the occurrence time of the alerts. Similar to the source and destination IP addresses, this attribute is used the feature of the first correlation.
4	Number of alarms per cluster	This attribute is the first feature of the second stage correlation. Having run the first correlation, the number of alerts grouped in each cluster is computed.
5	Number of signatures per cluster	Instead of counting the number of alerts per cluster as previous attribute, this feature identifies the signatures, which have triggered the alerts in each cluster. The number of signature type in each cluster is determined and used as the second attribute.
6	Protocol	The protocol type of the packet triggering the alert is retrieved as one of the correlations attributes. The detail about how the protocol is used in the correlation as well as the value assigned to this attribute is described in Table 5-3.
7	Port number	Similar to the protocol, the application of port number in the correlation is mainly aimed to inspect the type of traffic or service triggering the alerts. Again, the information about the value assigned to this attribute is presented in Table 5-3.
8	Alarm priority	Alarm priority is a unique feature that ranks the alerts in order of severity. Such attribute is given by the Snort IDS. The higher the rank is given, the more critical the alert is or the higher the priority it has.
9	Time interval	Time interval is one of the leading features of the second stage correlation. It is the time span between the occurrences of events from a specific signature. Prior to running the correlation, the time span of all signatures within a particular time frame is computed. As there are three choices of time frames, namely half an hour, an hour and two hours, available, the time interval between events from all signatures within each time frame is gauged.

Continued on next page

Table 6.2 – continued from previous page

No	Attributes	Description
10	Number of events	Similar to the time interval, this attribute is another leading feature of the second correlation. The number of events is calculated per signature within a particular time frame, for example, half an hour, an hour, two hours.
11	Cluster number	Cluster number is the indexes assigned to the clusters generated in the first stage correlation. It is also used as the input references for the second correlation.
12	Alert status	The alert status, which is the final outcome of the second correlation, indicates the validity of the generated alert (that is whether it is a true or false alarm). Zero (0) indicates false alarm whilst 1 indicates the true alarm.
13	Signature	The last important feature of the alerts that considerably influences the trend of IDS alert generation is the IDS signature. This attribute indicates the type of IDS signature rules that have raised the alarms.

In order to provide a clearer picture of how the two leading attributes, namely the time interval and the number of events are calculated, the following description provides the complete pseudocode of the calculations of both attributes.

Algorithm: Counting the Number of Events and Time Interval

$A \leftarrow$ starting timestamp
$B \leftarrow$ endingtimestamp
$dayAlerts \leftarrow$ database table
{Calculating the number of events and the time interval}
$A \leftarrow curTime$
while $curTime \neq B$ **do**
  $curVec \leftarrow$ datevec($curTime$) {$curVec$ is [year, month, day, hour, minute, second]}
  $NextVec \leftarrow curVec$[hour] + 1
  $NextTime \leftarrow$ datestr($NextVec$) {$NextTime$ is in "yyyy-mm-dd HH:MM:SS" format}
  $sig \leftarrow$ signatures in range $curTime$ to $NextTime$ {Retrieve signatures that have triggered alerts from $curTime$ to $NextTime$ from table $dayAlerts$}
  STORE the values into sig
  $pj \leftarrow |sig|$ {pj is the size of sig}
  $Timetb \leftarrow pj$x3 zero matrix
  CREATE a sql table named $TimeEventOne$

for $co = 1$ to pj **do**
  $Timetb(co,1)$ = $sig(co)$ {Insert distinct signatures into matrix $Timetb$}
end for
for $x = 1$ to pj **do**
  $src \leftarrow$ source IP addresses
  $dst \leftarrow$ destination IP addresses
  $ti \leftarrow$ timestamp {RETRIEVE the values of timestamp, source and destination IP addresses from table $dayAlerts$, where the signature is equal to the $sig(x)$ and the timestamp starts from $curTime$ to $NextTime$}
  $noALerts \leftarrow |src|$ {$noAlerts$ is the size of src}
  for $v = 1$ to $noAlert$ **do**
    vec = datevec$(ti(v))$
    $vec(6) \leftarrow 0$
    $time(v) \leftarrow vec$
  end for
  for $col = 1$ to 2 **do**
    for $row = 1$ to $noAlert$ **do**
      if col is equal to 1 **then**
        $data(row, col) \leftarrow src(row)$
      else
        $data(row, col) \leftarrow dst(row)$
      end if
    end for
  end for
  $t \leftarrow 0$
  $i \leftarrow 1$
  $T \leftarrow$ 1x$noAlert$ zero matrix
  $event \leftarrow 0$
  $index \leftarrow 0$
  for $n = 1$ to $noAlert$ **do**
    {Check if the alarm has been covered by previous alarm}
    if $n \in T$ **then**
      $event \leftarrow event$
    else
      $event + 1$
      if $event = 1$ **then**
        $t \leftarrow 0$
      else
        $difTime \leftarrow |time(n) - time(index)|$
        $t \leftarrow t + difTime$
      end if
      $index \leftarrow n$
      $T(1, i) \leftarrow index$

```
        i + 1
        j ← n + 1
        {Finding the alarms covered by the current alert}
        if n ≠ noAlert then
          while |time(j) − time(n)| ≤ 180 do
            {If the elapsed time is less than equal to 3 minutes}
            if data(j, 2) = data(n, 2) || data(j, 1) = data(n, 1) then
              {If either the destination or the source IP addresses are match}
              T(1, i) ← j
              i + 1
            end if
            INCREMENT j
            if j > noAlert then
              Break out of the loop
            end if
          end while
        end if
      end if
    end for
    Timetb(x, 2) ← event
    if t ÷ event = 0 then
      Timetb(x, 3) ← 7200
      {No of seconds in two hours}
    else
      Timetb(x, 3) ← t ÷ (event − 1)
    end if
    INSERT the values Timetb(x, :) to the sql table TimeEventOne
  end for
end while
```

An event is considered the same as others from the same signature if it is triggered within a time frame of three minutes. In other words, all alerts that have been triggered by the same signature within a time frame of three minutes will be counted as one event. Several observations were carried out beforehand to determine the average time span between the occurrence of events. As a result, the three-minute time frame is considered the most sensible threshold value; thus being selected and applied in the computation.

Once the values are computed, the results are stored in a table named $TimeEventOnetimestamp$; only if the values are calculated for a time frame of 1 hour, $TimeEventTwotimestamp$ for two hours time frame and $TimeEventHalftimestamp$ for half an hour time frame.

6.4.3 Data Storage

Having looked into the components or the attributes of the correlation system, it is now essential to discover the type of data storage used in the correlation as well as the information stored in each

of the files or databases.

It is worth remembering that the SMART system is deploying a relational database as the main information sources. Following sub-sections provide a brief description of the database files created during the correlation processes.

6.4.3.1 BASE database

The BASE database is the main information resource of the proposed correlation system. In fact, the system is currently developed to work with the BASE system only. This database, created by the Snort IDS and BASE, holds all the information required to perform an alert correlation and analysis.

6.4.3.2 *Time_Event* table

This table stores the values of the time interval and number of events pre-computed by the system for each signature within a particular time window. The table consists of three fields, namely the signature index, the time interval and the number of events. The values of the time interval and the number of events are calculated for all interval (time frame) options and the results of the computation are stored in separate tables.

6.4.3.3 *TimeEventRecord* table

This table holds the values of the sixth and seventh alert attributes, namely the time interval and the number of events, per cluster index. In other words, the values of time interval and the number of events for the inputs of the second correlation are recorded in the *TimeEventRecord* table.

6.4.3.4 *Stage1* table

This table is fundamentally created to store the result of the first stage classification. It holds the information about the alert ID as well as the cluster the alert belongs to. The table consists of two columns, namely the alert ID and the cluster index.

6.4.3.5 *Stage2* table

Similar to the previous table, the Stage2 table keeps the final outcome of the classification (second stage). There are primarily two fields created in this table, namely the cluster index and the alert status. The alert status is a grade assigned by the system to a cluster that indicates the validity of the alerts, that is whether they are true or false alarms, inside the cluster. For example, if the status of a particular cluster is set to 0, then all alerts inside the cluster are considered false alarms.

6.4.3.6 *PortNo* table

PortNo table contains a list of all official port numbers. The idea of creating this table is to match the port numbers of a packet triggering the alert with the list of the official port numbers. If a match is found, the port is regarded as a service port and will be used as an attribute for the classification.

6.4.4 System Output

In addition to the statistical figures and charts presented in the front-end interface, the system also produces several output files, which are saved in the local drive. Owing to the proposed two stages of correlation, there are two parts of classification results generated in this context. Therefore, in order to gain more understanding about the outcome or the end product of each stage, brief descriptions about the final outputs of two stages are presented as follows.

6.4.4.1 Stage1 classification result

In this level, there are ultimately two files generated during the process. Those are:

- Stage1 input file

  Prior to executing the main correlation engine, the system is programmed to automatically collect and process all necessary alert attributes for the correlation. Once they have been processed, then the attributes are written into a simple text file and serves as the input data for the classification. The filename format for this text file is set to "stg1tbtimestamp.txt", for example stg1tb19990330120000.txt. The timestamp starts from year, month, day, hour, minute and second.

- A figure file from stage 1 correlation

  The map, which is generated by the SOM and K−mean correlation and presented in the shape of U-Matrix, is saved as a figure file.

6.4.4.2 Stage2 classification result

Similar to the stage 1 classification, there are two files produced in this stage, namely:

- Stage2 input file

  The file contains all selected and pre-processed attributes for the alerts. The information is then used as the input data for the second classification. Moreover, the filename format is set to "stg2tbtimestamp.txt", for example, stg2tb19990408220000, with stg2 refers to stage 2.

- A figure file from stage 2 correlation

  This file is similar to the stage 1 figure except that it is a map figure generated by the stage 2 correlation.

6.5 Conclusions

This chapter has focused on the conceptual architecture for a SOM K−Means Alarm Reduction Tool. The descriptions include an introduction of the main concepts or the characteristics of the architecture, and the modules within it. Key focus was given to the role of each module, and especially its contribution in the alert correlation process.

Although identifying the underlying features of the SMART system is a significant process, it is necessary to evaluate the practical viability of the system and demonstrate how the main features

of SMART would work in a practical scenario. Having established this, it is also important to describe the implementation or the deployment view of such a system. As such, the next chapter presents the implementation of the SMART prototype system, which aims to prove the feasibility of implementing the system in a practical environment.

7 A Prototype Alarm Correlation System

7.1 Introduction

This chapter describes the development process, in other words, the implementation stage of the prototype system, which represents a subset of the key elements of the proposed architecture, namely the ability to aggregate and filter the false alarms, and to provide a flexible and high level IDS alert comparison. Several modelling languages, including a behavioural diagram, an interaction diagram, an activity diagram and a static structural diagram are then presented, offering a standard way to visualise the system architectural model. Finally, the interface of the alarm correlation tool is described, highlighting its features and its role in processing and presenting IDS alerts.

7.2 Implementation Overview

The elements of the SMART architecture that have been implemented in the prototype are depicted in Figure 7.1. The figure also shows the elements that have been incorporated into architecture rather than developed, such as Snort IDS and BASE (refer to subsection 4.1.2.1 and 4.1.2.3. Significantly, focus has been given to the features that correlate and cluster alerts into groups of true and false alarms, and organise the correlated alarms in a statistical view. It should be noted that the implementation of a complex and fully functional interface of the SMART system would require a further study, in order to develop or enhance the performance of elements, such as the Alert Management Tool.

In general, the prototype system consists of three modules, namely the I/O Interface, the Correlation Engine and the Knowledge Database. The I/O Interface enables user interaction on the system. The Correlation Engine encompasses the functionality of feature extraction, attributes conversion and the correlation itself. Finally, the Knowledge Database serves as the main information source and incorporates the feature of Map Selection Policy, Attribute Conversion Policy, Alert Database, Time Event and Port No. The three modules of the prototype system are illustrated in Figure 7.2. As depicted in the figure, the users are able to provide inputs to the system via the interactive I/O interfaces. The Correlation Engine then communicates with the Database to either retrieve essential information for the correlation or store the outcome of the classification. Once the correlation process is completed, the final results are presented to the users using the I/O interfaces.

As described in Chapter 5, SMART uses a collaboration of SOM Neural Network and $K-$means clustering algorithm, which serves as the main correlation engine. The reason of choosing these technologies is due to their ability to preserve topological mappings between the input data, which

is very important for the representation of complex data (Kohonen, 1995). The proposed correlation engine (back-end) was developed in the MATLAB environment, using SOM Toolbox. SOM Toolbox is a software library for MATLAB 5 or above implementing various clustering algorithms; including Self Organising Map and K−means (CIS, 2005). On the other hand, the front-end that features the input and output of the application interface was written in Java. Although other well-known programming languages, such as C#, are commonly used to develop front-end tools, they are not platform independent as Java program. In fact, Java support is becoming ubiquitous; it is integrated into practically all major operating systems (Flanagan, 2005). Also, in order to integrate both back-end (that is Correlation Engine) and front-end systems (that is I/O Interface), the MATLAB Builder JA (MathWorks, 2010) is utilised. It enables the system to deploy MATLAB code as Java classes. The builder creates deployable components that allow the MATLAB based computation and visualisations to be accessible to the end users of the Java programs.

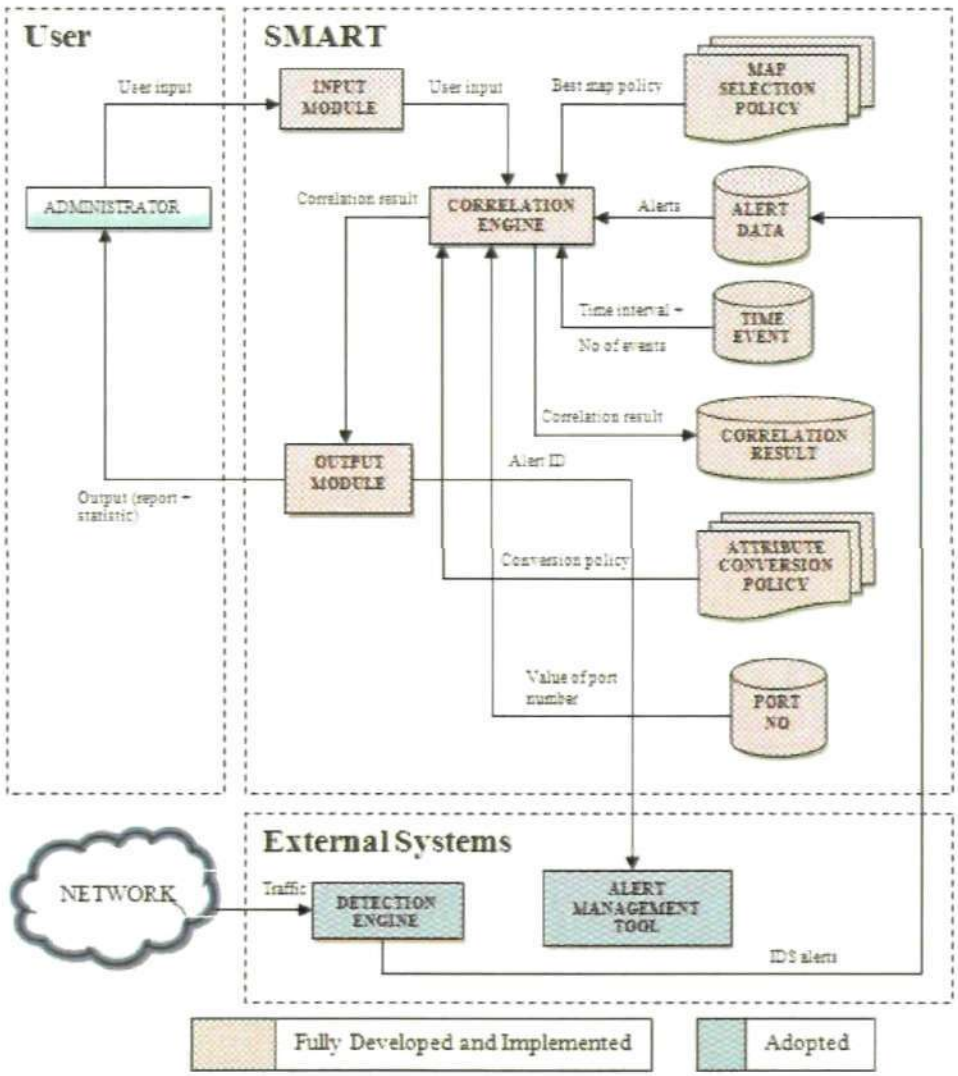


Figure 7.1: Prototype implementation

Considering that the MATLAB code has been deployed as Java classes, the prototype system is

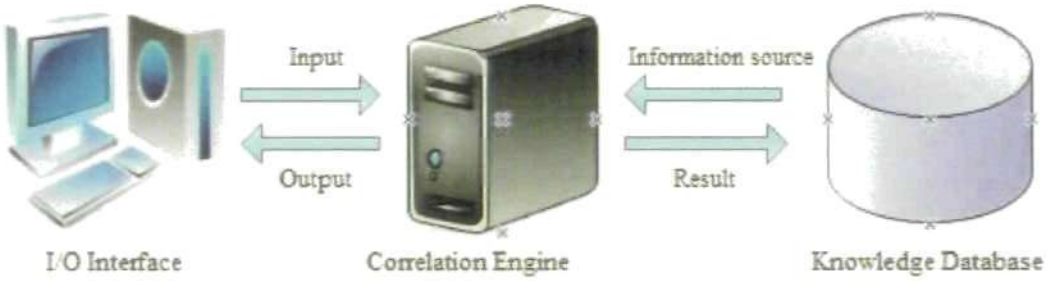


Figure 7.2: Prototype modules

now regarded as a stand-alone application, which can be run independently of any other applications. Indeed, the system is compiled into an executable JAR (Java Archive) file, which aggregates many files into one. The JAR file format is commonly used to distribute the Java applications or libraries, in the form of classes and associated resources.

The prototype system has the following dependencies, which are not included in the distributed application package and need to be installed separately before the prototype system is executed:

1. Java Runtime Environment (JRE) (version 6 or above)

   The JRE provides a main platform for the deployment of the prototype system.

2. MySQL Database

   A relational database is required to store all necessary information for the correlation process and also to keep the final result of the correlation.

3. BASE

   Prior to executing the prototype system, it is crucial to set up an alert management tool, which is connected to the Detection Engine (that is Snort IDS). More importantly, it enables the alerts to be studied to a much greater degree, for example, by examining the packet payload. Bear in mind that before installing BASE, there are several prerequisite configurations or software requirements to be met. These requirements are external to the BASE system and are listed below.

   - Snort IDS system
   - MySQL
   - Apache Server
   - PHP
   - Pear, which includes Image_Graph, Image_Canvas, Image_Color, Numbers_Roman and Mail_Mime
   - ADODB

   The details of BASE installation can be found in (BASE, 2009).

For more details of system functional requirements, please refer to Appendix E.

7.3 Input

Ultimately there are three main user inputs required to initiate a correlation, namely the starting timestamp, the ending timestamp and the correlation time frame. The input interface, which is created as the starting interface (that is the main page) of the prototype system, is primarily responsible for providing a media for those inputs. The following subsections demonstrate how the input interface has been practically designed to enable basic user interactions.

7.3.1 Starting and Ending Timestamp

In the main page of the prototype system, the users are prompted to specify the range of alerts to be processed in the correlation by providing the starting and the ending timestamps of the alerts; as shown in Figure 7.3.

Figure 7.3: SMART - user input

Two text boxes, as shown in Figure 7.3, are provided to allow the users to enter the aforementioned starting and ending timestamps. The date and time format used in this input module is of the complete date plus hours, minutes and seconds (that is yyyy-mm-dd hh:mm:ss). In order to ensure the correctness of the date and time format entered by the users, an input validation is performed once the "Submit" button is pressed. The failure to provide the correct format may result in a correlation error. Additionally, an error dialog box will be raised to notify the users of the invalid inputs and subsequently, the users are prompted to re-enter the timestamps.

7.3.2 Time Frame

The third user input required in the correlation process is the time frame. The time frame refers to the time interval between each correlation. There are three options of interval available in the combo box namely every two hours, every one hour and every hour; as described in Figure 7.3. By choosing one of these options, the alerts are then divided into separate groups based on the chosen interval and fed into different correlations. Bear in mind that the correlations are not run in parallel, instead they are executed in temporal order. If there is no action has been made to select the time interval, the two hours time interval is set by default.

7.4 Output

The final outcomes of the correlations are presented in several methods, including the presentation of the statistic figures, the alert and signature tables as well as a feature, which facilitates the analysis of the signature and the packet payloads. An output interface, which is responsible for presenting the final correlation results, consists of three tabbed panels. The following subsections will describe the features as well as the layout of each panel in greater detail.

7.4.1 Alarm Statistics

The first panel, which is named "Alert Table" tab, holds an alert table containing all alerts features, the cluster number the alerts are belong to and also the status of the alerts (either true or false alarms). Apart from the table described, there is also a statistic pie chart that depicts the proportion of true to false alarms in the total alerts. The layout of the "Alert Table" tab is shown in Figure 7.4.

The 16 columns table, shown in Figure 7.4, is composed of 12 feature columns, one counter column, one id column and two result columns, namely the results of the first and second stage correlations. The Cluster_No field, which holds the outcome of the stage1 classification, is the index number of the cluster the alert is grouped into. Conversely, the last column of the table (Alert Status), which contains the final result of stage2 correlation, shows the class the alert belongs to, whether it is a true or false alarm.

7.4.2 Chart Report

In terms of a graphical report, SMART prototype enables the administrators to produce statistical charts based on the results of the correlations. The second tab, "Chart Report" is therefore created to allow the users to generate a statistical chart by simply filling in the chart attribute form provided; as shown in Figure 7.5.

The key objective of facilitating a chart feature is to primarily provide a better view of the false alarm issue to the administrators and also to allow the users to observe the trend of the false alarms generation over a particular period. A total of 15 charts can be generated from this feature, namely:

1. Time (hours) vs No of False Alarms
2. Time (days) vs No of False Alarms

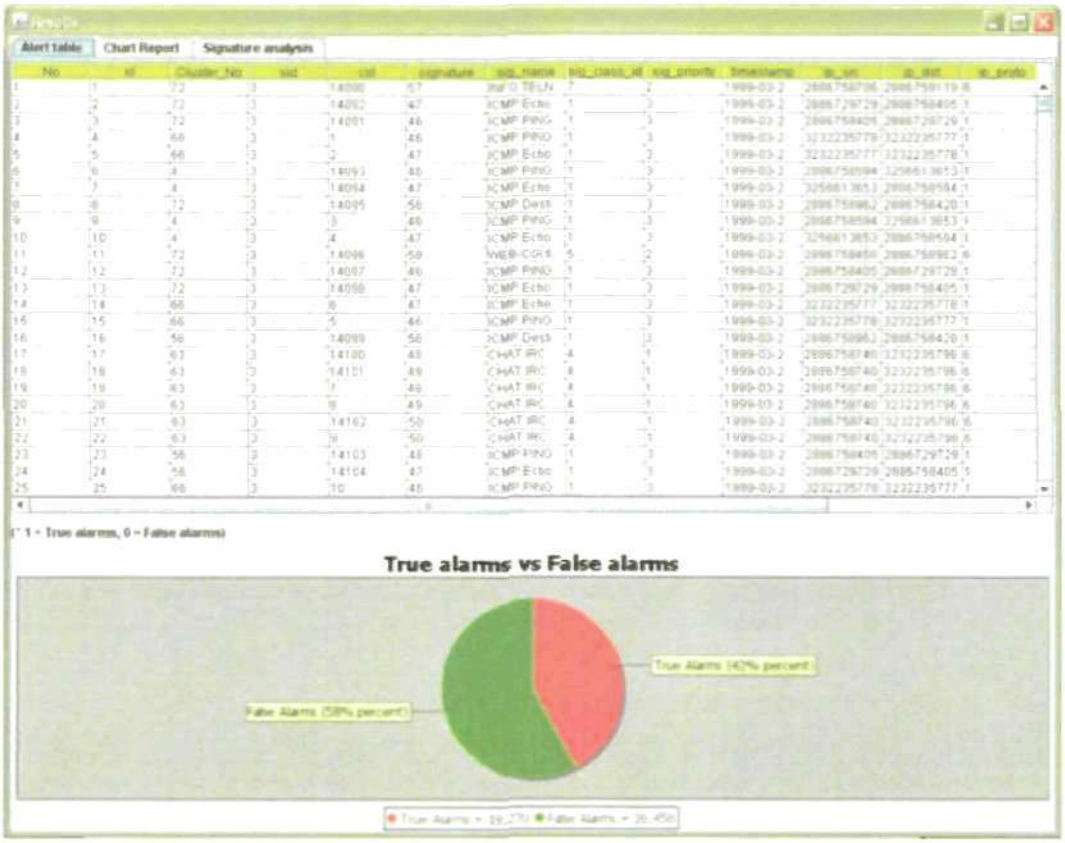


Figure 7.4: SMART - Alert table tab

3. Time (months) vs No of False Alarms
4. Time (hours) vs No of True Alarms
5. Time (days) vs No of True Alarms
6. Time (months) vs No of True Alarms
7. True Alarms vs False Alarms (hours)
8. True Alarms vs False Alarms (days)
9. True Alarm vs False Alarms (months)
10. Time (hours) vs False Signatures
11. Time (days) vs False Signatures
12. Time (months) vs False Signatures
13. Time (hours) vs True Signatures
14. Time (days) vs True Signatures
15. Time (months) vs True Signatures

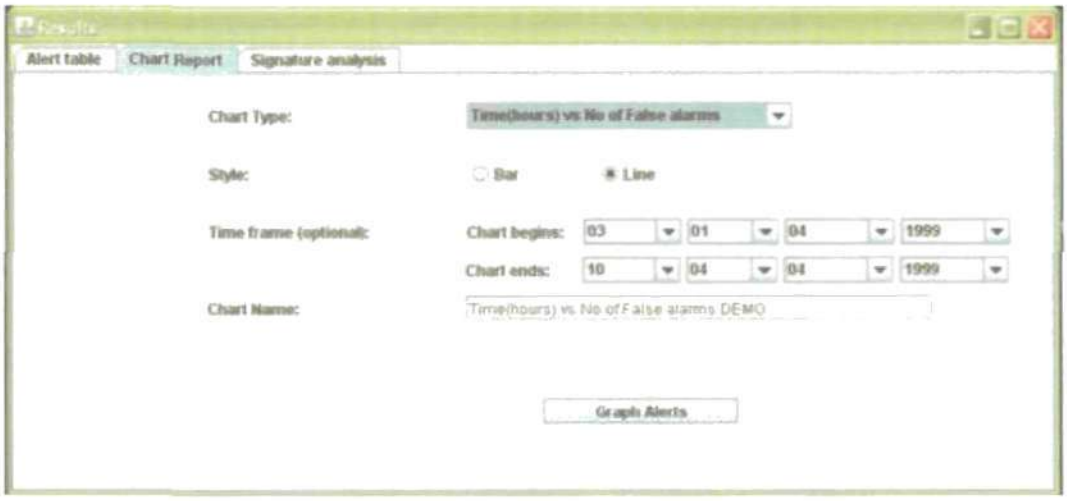


Figure 7.5: SMART - Chart report tab

The idea of using the timing information in the the context of false alarms is to simply provide an overview of the trend of the false alarms for the administrators. In fact, such a method has been commonly used by most well-know alert analysis tool, such as BASE.

By investigating the false alarms over time, the administrators could identify the behaviours of the false signatures triggering the alarms. In this way, the signatures could be analysed by critically examining the payload of the packets triggering the false alarms. The final act of this analysis would be either disabling or tuning the signature rules. Moreover, if the signatures generate both true and false alarms, the administrators need to further evaluate the clusters (in other words, the result of stage 1 correlation), to which the alerts belong to. Subsequently, the cause of the alerts could be identified based on the time interval and number of events of the corresponding clusters. The process of signature analysis will be detailed in subsection 7.4.3.

Each chart is created in a separate frame and presented in the form of either bar (histogram) or line chart. Apart from choosing the style and type of the chart created, the users are also allowed to set the time window for each diagram by determining the starting and ending time of the chart. Thus, instead of reviewing alerts in general, such option enables the administrators to specifically focus on alerts from a particular time frame. Figures 7.6 and 7.7 illustrate the samples of bar and line diagrams generated by the chart report tool.

7.4.3 Tables of Signatures

The last tab, "Signature Analysis", is responsible for listing all signatures rules triggering the alerts and also identifying which of the signatures from the list that have solely raised the false alarms, true alarms or both true and false alarms. Figure 7.8 shows a sample of the "Signature Analysis" tab. This feature aims to facilitate the administrator in performing a signature analysis by presenting lists of true and false alarms generated by the particular signature.

The signatures that have purely triggered true alarms are shown in red fonts, whilst those that have generated only false alarms are displayed in green fonts. On the contrary, the blue wording represents the rules that have raised both true and false alarms. The classification of the signatures

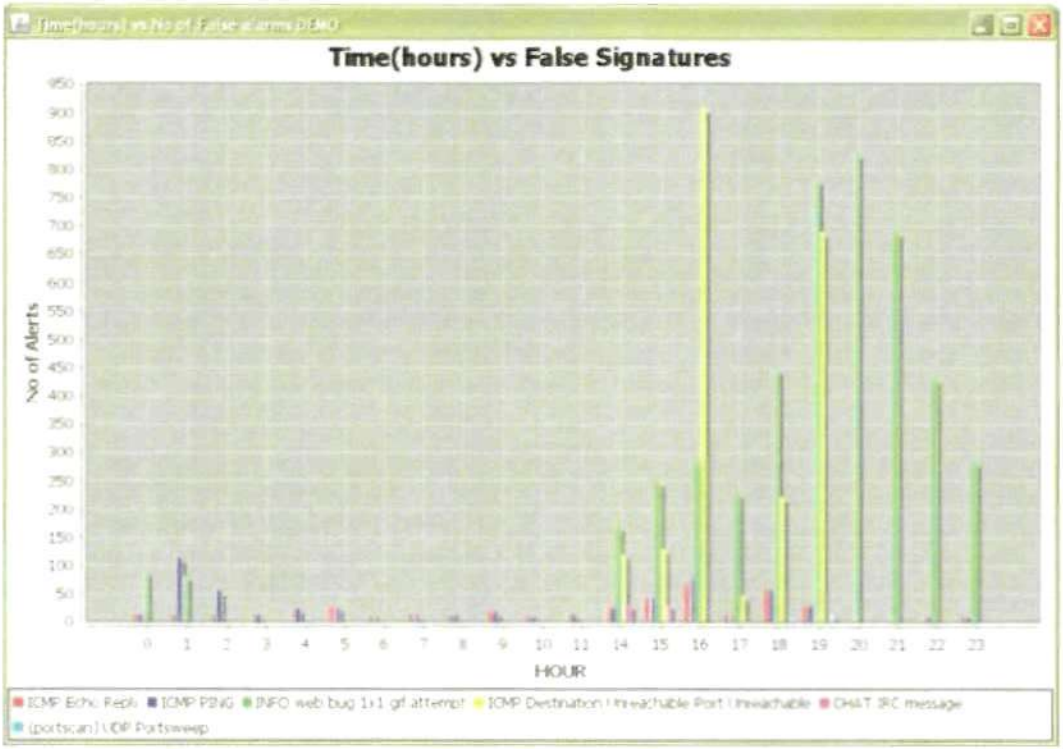


Figure 7.6: SMART bar diagram - Time(hours) vs False Signatures

according to the type of generated alerts not only allows the administrators to focus on the false signatures but also enables them to review or revise the signature pattern.

Having been presented with a list of signatures, the users are then prompted to select one signature from the signature list for a further analysis. The details of the selected signature is then described in a plot diagram depicting the distribution of the source IP addresses (from both true and false alarms) related to a particular signature in every hour (as shown in Figure 7.9).

Besides, two tables, which contain the attributes, for example, Cluster index number, Number of events, time interval and signature rules, of the true and false alerts triggered by the corresponding signature, are also presented. The values of the attributes described in the tables are fundamentally retrieved from the features of the clusters resulted from the stage1 correlation. The idea behind these tables is to let the administrators observe the variance between alerts from the same signature as well as to compare the characteristics of the alerts from both categories, namely true and false alarms. Figure 7.10 displays a sample of the true and the false alarms tables.

In the process of reducing the false alarm rate in the future detection, the administrators are required to properly inspect the signature pattern and if necessary to tune the signature rules that could potentially generate the false alarms. In order to perform such task, it is essential for an alert management tool to provide the administrator with access to the payload of the packet triggering the alerts. Although the presentation of the payload is only intended to highlight the cause of the alert generation, it certainly could help the users understand the inherent behaviours or patterns of the applied signature rules. So, with this benefit in mind, the prototype system allows the users to gain access to the packet payload via BASE (Basic Analysis and Security Engine). The ID number

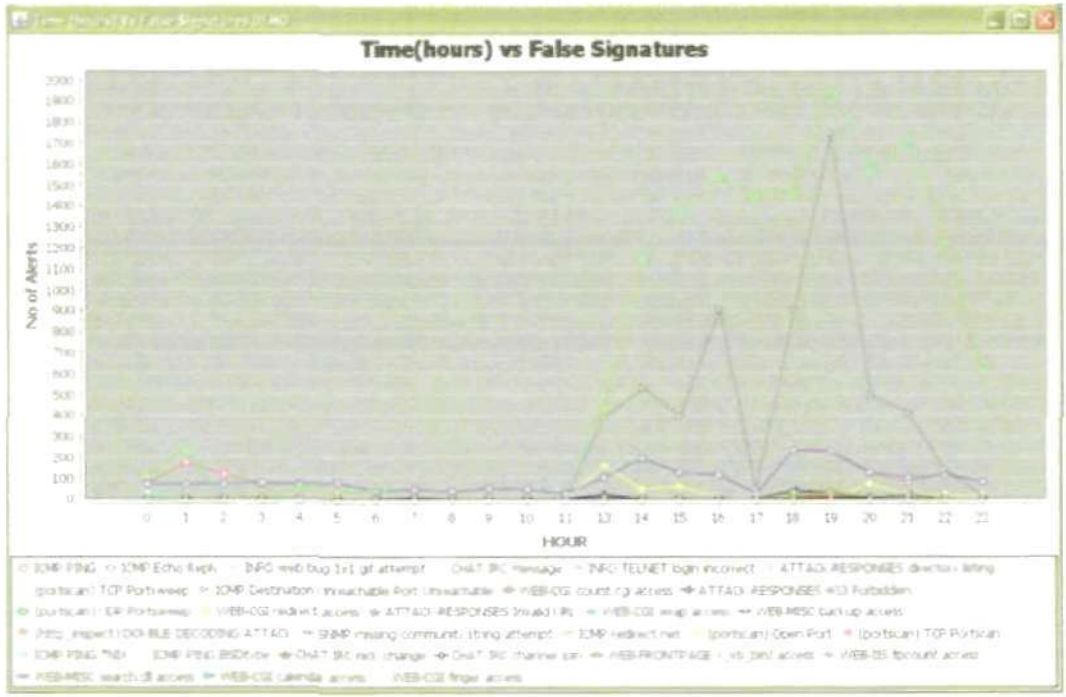


Figure 7.7: SMART line diagram - Time(hours) vs False Signatures

from each alert shown in the signature table is linked to the BASE page containing the details of the packet header and payload. Figure 7.11 shows a sample of the packet payload taken from BASE.

7.5 Demonstrating the SMART Prototype System

Having described the role and main features of the prototype system, this section will provide examples of how alert attributes can influence the correlation decision process and how the classification system can utilise the features of the IDS alerts to cluster as well as to determine the validity of the alerts presented. Specifically, the examples presented demonstrate how the occurrence of alerts from the same signature can be classified into different categories (either true or false alarms) in different context. The alerts, which are correlated in examples, are generated by Snort IDS, which run on both real data (University of Plymouth data set) and DARPA data set 1999. The processes of running the correlation and examining the correlation results are demonstrated in the examples and the context in which the correlation outcome are evaluated reflect two parts of output assessment. The first section of the assessment exhibits the overall result of the alarm reduction method, for instance by presenting a comparison chart between the true and the false alarms. On the other hand, the second segment enables the users to individually analyse the false signatures and subsequently allows access to the packet payload via BASE page. Finally, it should be noted that, the correlation time frame has been configured in all cases to the highest time windows available (that is two hours time frame). All of the examples will be demonstrated in practice on the prototype system.

Figure 7.8: SMART - Signature analysis tab

7.5.1 Data Description

Similar to the previous experiments conducted on the false alarm issue (Chapter 4) and the preliminary study (Chapter 5; two types of data sets, namely the University of Plymouth and the 1999 DARPA data sets are used in the demonstration. As for the DARPA data set, the 4th and 5th weeks of testing data is used, whilst for the University of Plymouth data set, only the first two weeks of data is selected.

Total of 91,671 alerts, which were recorded from the two weeks DARPA testing data, will be fed into the prototype system. Unfortunately, due to the memory issue suffered by the MATLAB application, the system is unable to process more than 3,000 alerts per correlation (this issue is further discussed in Chapter 8, Conclusions). As a result, the correlations, which process more than 3,000 alerts in each two-hour time frame, are omitted; thus resulting in only 56,119 alerts handled by the correlation system. As for the University private dataset, fortunately, none of the correlation processes are required to run more than 3,000 alerts; thus the complete alerts (54,893 alerts) from the two weeks data set can be successfully correlated by the system.

Since the section aims to simply demonstrate how to run the prototype system, only the DARPA data set is described in these examples. A detailed discussion of the correlation result from both DARPA and University data sets will be presented later in Section 7.8.

7.5.2 Example 1 - Running the Correlation

The manual for running the correlation can be found in Appendix F.

During the correlation process, several files and database tables are created and saved in the local hard drive. Following lists the files created throughout the process of correlation:

Text Files:

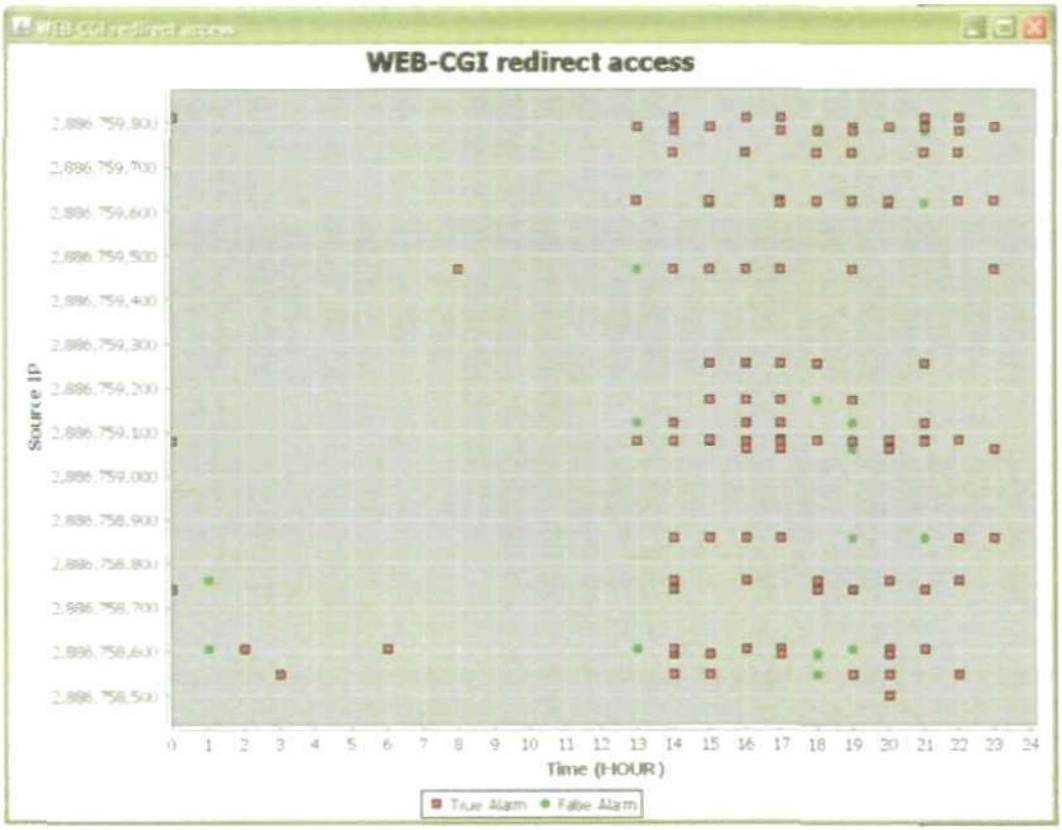


Figure 7.9: SMART - Signature plot diagram

1. Stage1 Input data

   Prior to the correlation, the alert features for stage1 correlation, namely timestamp and IP addresses, are extracted, pre-processed and written into an input text file.

2. Stage2 Input data

   Once the first correlation is completed, the cluster attributes for the second stage correlation are extracted from the first outcome. Total five attributes are retrieved from the stage1 result, whilst the rest 2 attributes, namely time interval and number of events are obtained from the *Time_Event* table (see subsection 6.4.3.2). Similar to the stage1 input data; the collected attributes are pre-processed and written into a text file.

3. Stage1 Result

   This text file contains the output of the stage1 correlation. It lists the index numbers of alerts per cluster in row; one row for each cluster.

4. Stage2 Result

   This file, which holds the final outcome of the stage2 correlation, contains only two rows of data since only two classes, namely true and false alarm classes are formed in the final classification. The index numbers of the clusters created in the first correlation are listed in either row depending on the category the clusters are grouped into.

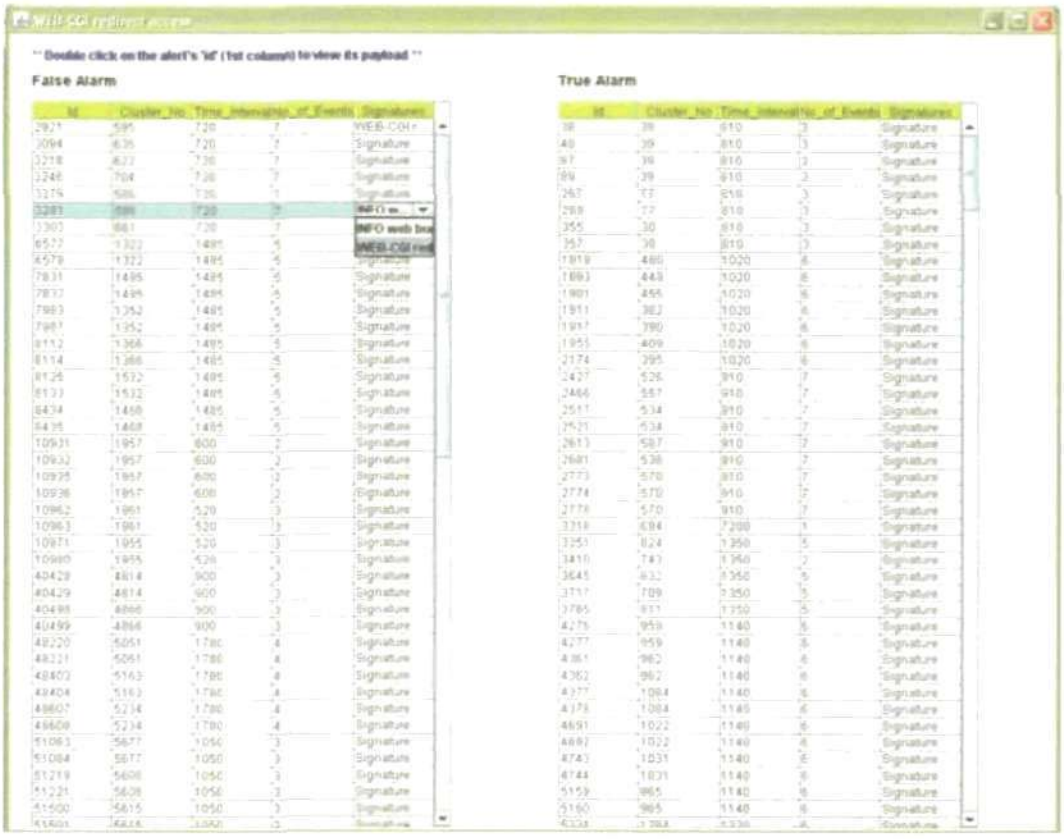


Figure 7.10: SMART - Signature tables

MATLAB Figures:

1. Stage1 Result

   The figure is generated by the MATLAB application and contains the final map of the first classification. Figure 7.12 displays an example of the map produced by the stage1 correlation.

2. Stage2 Result

   Stage2 map is similar to the stage1 map except that the former is divided into two clusters only (see Figure 7.13).

Database tables:

1. *Stage*1 result table (Subsection 6.4.3.4)

2. *Stage*2 result table (Subsection 6.4.3.5).

3. Table of selected time interval and number of events (*TimeEventRecord* table; Subsection 6.4.3.3).

4. Table of time interval and number of events per time window (*Time_Event* table; see Subsection 6.4.3.2).

Figure 7.11: BASE - Payload page

7.5.3 Example 2 - Viewing Overall Correlation Results

As soon as the correlation is completed, the result is ready for analysis. In order to view the outcomes, the user is required to press the "View Result" button as shown in Figure 7.14. Should the user need to return to the main page without viewing the result, the "Reset" button should be pressed.

As discussed previously in Section 7.4, the result is presented in a new tabbed window, which is the same as shown in Figure 7.4. There are total three tabs included in the window, namely the "Alert Table", the "Chart Report" and the "Signature Analysis". In order for the user to evaluate and compare the values of the true and false alarms classes resulted from the correlation, a chart tool is provided in the "Chart Report" tab to assist the user with the creation of a graphical report. And in order to use this feature, the user needs to click on the "Chart Report" tab, as illustrated previously in Figure 7.5.

A chart attribute form is presented and the user is prompted to create a graphical chart by filling in the form provided. There are a number of chart types and style available in the "Chart Report" tab for the user to choose from. Should the user need to produce a chart from a particular period, an optional attribute called the time frame is available for the user to configure. Once the form is completed, the user can subsequently press the "Graph Alerts" button, as shown in Figure 7.5, to initiate the chart creation.

As described previously in subsection 7.4.2, a total of 15 charts from five categories can be produced by the chart engine. Following description explains the objectives and provides examples of those 15 charts. The figures are presented in either bar or line chart.

1. Time vs No of False Alarms

   Following charts present the trend of false alarm generation over time. The reason of presenting this category of chart is to simply provide an overview of the false alarm distribution for

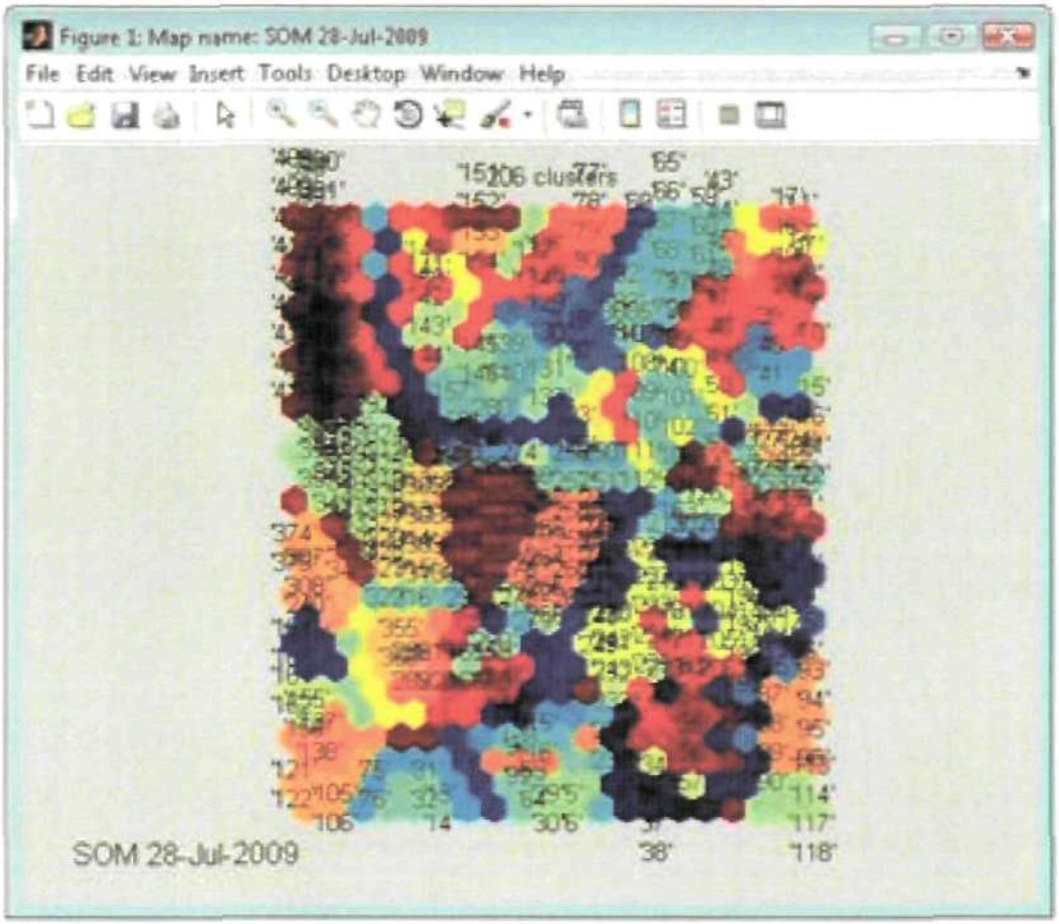


Figure 7.12: An example of stage 1 map

each hour, day and month and also enable the administrators to assess the IDS performance in terms of its false alarm rate.

(a) Hours

The chart depicts the number of false alarms detected from the Snort alerts for each hour of the day. Figure 7.15 shows an example of the chart.

(b) Days

This type of chart is similar to the previous one except that the alerts are charted per day. An example can be seen from Figure 7.16.

(c) Months

The graph shown in Figure 7.17 portrays the generation of the false alarms per month.

2. Time vs No of True Alarms

A similar objective is aimed in this type of diagram. The generation of a graph representing the figure of the true alarms aims to evaluate and grasp the trend of the real alerts (real warnings); thus measuring the effectiveness or the capability of the Snort IDS in detecting real attacks. It is worth noticing that the time of occurrence of a real alert can be used to

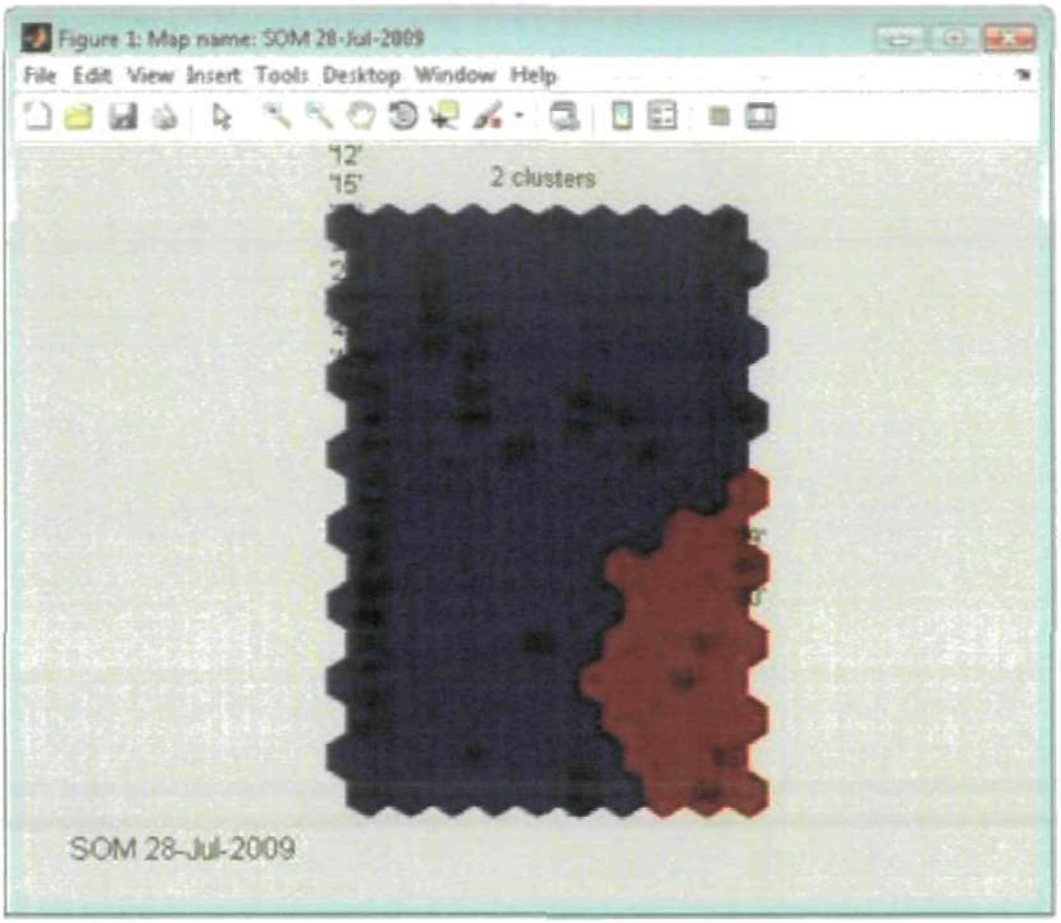


Figure 7.13: An example of stage 2 map

uncover the behaviours of the attack. In fact, the graphs also highlight the security status of the monitored network based on the rate of the detected true alarms. Figure 7.18, 7.19 and 7.20 show examples of the true alarms figures graphed on different periods, namely hours, days and months respectively.

(a) Hours

(b) Days

(c) Months

3. True Alarms vs False Alarms

Figure 7.21, 7.22 and 7.23 are examples of charts that compare or evaluate the figures of true and false alarms on different periods, namely hours, days and months. The key idea of presenting a comparison chart between the generation of the true and the false alarms over time is to evaluate the trend of the false alarms against the true alarms for a specific period. Moreover, the charts also aim to highlight the severity of the false alarm issue and also to assess the performance or the detection rate of the applied IDS.

(a) Hours

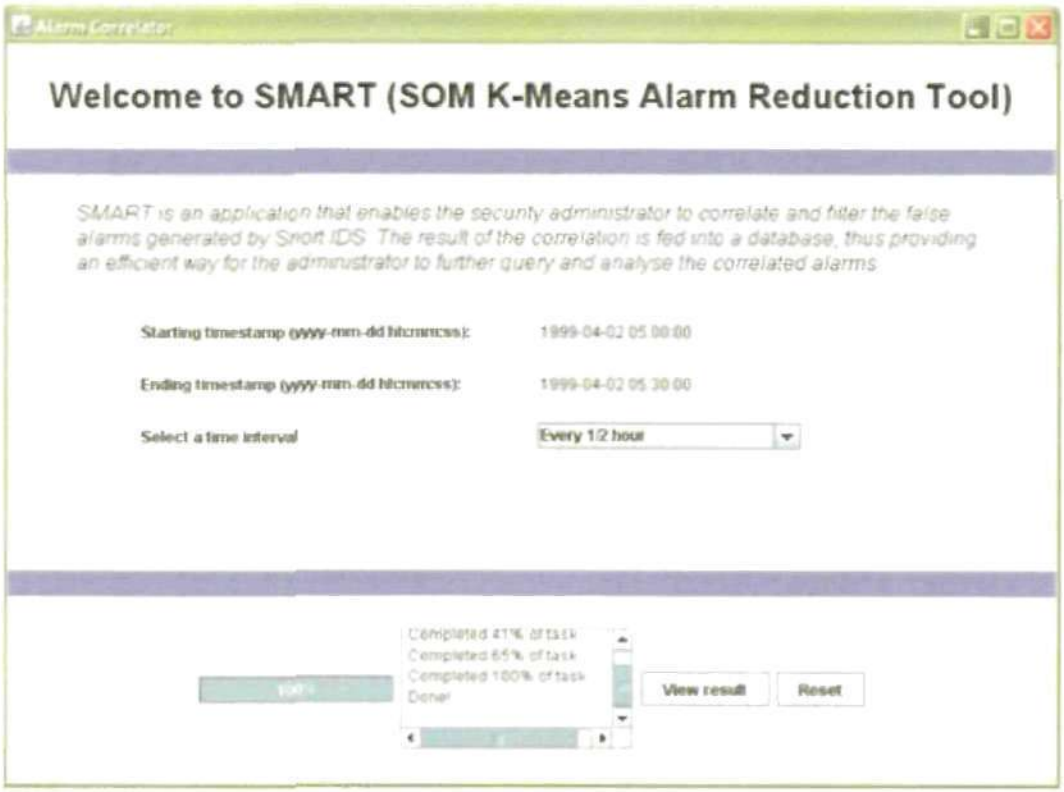


Figure 7.14: SMART - Correlation completed

(b) Days

(c) Months

4. Time vs False Signatures

This category of charts lists the number of false alarms triggered by each signature in a particular period, for instance hours, days and months. The benefit of having this chart is that it provides an overview of the problem of false alarms for each signature rule and also it helps the users identify the "noisy alerts" and the associated signatures for a future tuning. By observing the distribution of false alarms each period per signature rule, the user could discover behaviours of that particular signature. The charts surely help the user determining which of the signature rules need to be further reviewed. Figures 7.24, 7.25 and 7.26 are the examples of the false signatures charts.

(a) Hours

(b) Days

(c) Months

5. Time vs True Signatures

These charts are the same as those shown in the previous category except that the figures depict the true alarms. The idea behind this true signature chart is to help the user discover

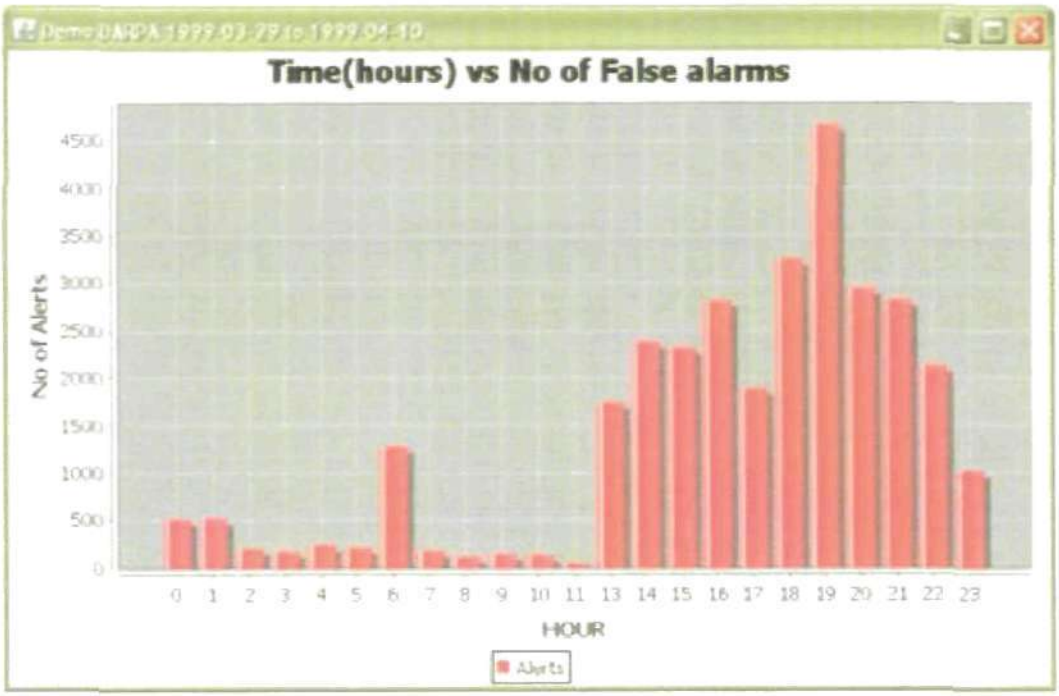


Figure 7.15: SMART - Time (hours) vs No of False alarms

the potential threats detected by the IDS and also assist the user in evaluating the performance of a particular signature rule in detecting real attacks. By looking at the signatures triggering the true alarms for each period, the user could gain an insight into the criticality of the monitored network and also to learn the type intrusive activities detected by the IDS. Figures 7.27, 7.28 and 7.29 represent the samples of the charts.

(a) Hours

(b) Days

(c) Months

7.5.4 Example 3 - Analysing Signature Rules

Aside from viewing and interpreting the results of the correlation through a graphical report, the prototype system also allows the user to explore the signature rules in more details. Figure 7-18 has portrayed a sample of the third tab, "Signature Analysis", which presents a list of signature rules from the DARPA data set.

As mentioned earlier in subsection 7.4.3, the status of the signatures, whether it is a false or true signature, is distinguished by the font colour of the signature name shown in the signature table. With this significant colour difference, the user can decide which of the signatures presented require an analysis or yet a future tuning. As shown in Figure 7.8, a selection column is available for the user to pick one signature for a further investigation. And once the signature is selected, then the user is expected to press the "Chart" button available below the signature table to initiate the analysis.

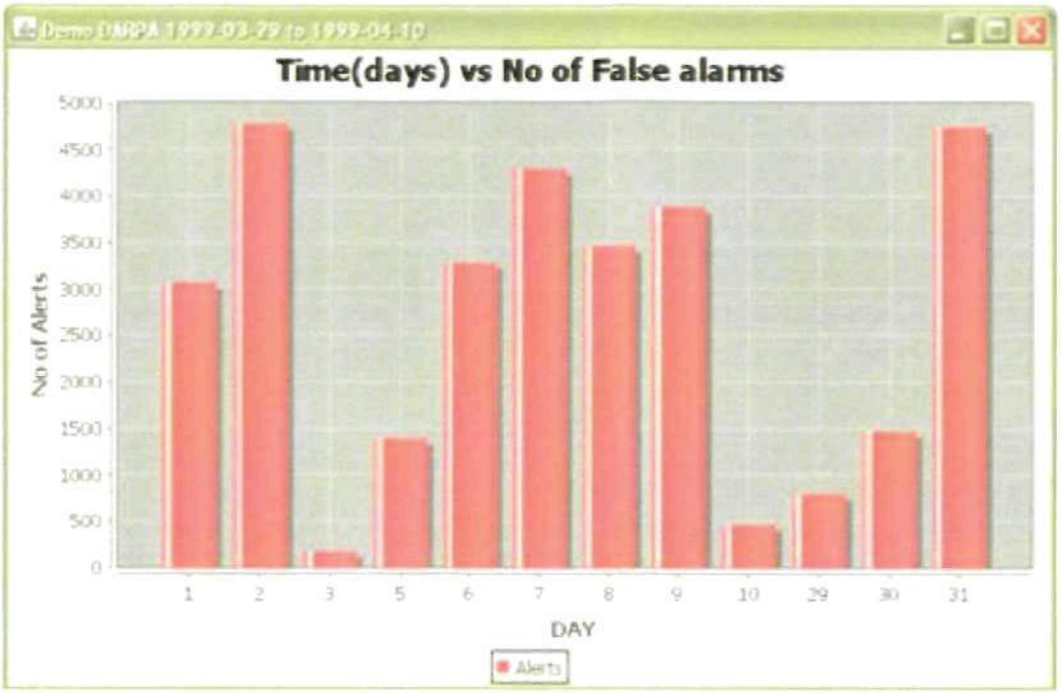


Figure 7.16: SMART - Time (days) vs No of False alarms

To evaluate the selected signature, two new windows showing a plot diagram and tables of true and false alarms are presented. In this scenario, the ATTACK-RESPONSES 403 forbidden signature is selected for the analysis. Figure 7.30 and 7.31 display the plot diagram and the true and false alarms tables associated to the signature respectively.

The plot diagram describes the distribution of the true and false alarms triggered by the ATTACK-RESPONSES 403 Forbidden signature based on its source IP addresses and the time of its occurrence. The key idea behind this diagram is mainly to explore the relationships among the true/false alerts, the source IP addresses of packets generating the alerts and also the occurrence time of the alerts. In this example, the diagram demonstrates that traffic from a single IP address could have triggered both true and false alarms from the same signature. This significantly indicates the issue of ambiguous alerts, which commonly causes trouble for the administrators in identifying real threats.

On the other hand, the second new window presents tables of true and false alarms containing the information of alerts from the ATTACK-RESPONSES 403 Forbidden signature. From Figure 7.31, it is clearly shown that each table contains five columns, namely id, Cluster_No, Time_Interval, No_of_Event, and Signatures. The id refers to a unique number the alert is assigned to, whilst the Cluster_No is a unique number of the cluster the alert is grouped into in the first stage correlation. The Time_Interval and No_of_Event are actually the dominant features used in the stage2 correlation. These attributes are computed during the correlation process; as described previously in subsection 6.4.2. Since both features are the leading attributes which significantly influence the final outcome of the correlation, the Time_Interval and No_of_Event are presented in the alert tables to help distinguish the characteristic between the true and the false alarms. Lastly, the

Figure 7.17: SMART - Time (months) vs No of False alarms

Signatures column holds the names of the signature rules that are grouped into the same cluster in the stage1 classification. A click on the signature row could reveal the list of signatures names in a combo box (see Figure 7.31).

Apart from dividing the alerts into two tables according to their status (either true or false alarms), it would be worth enabling the administrator to inspect the pattern or the payload of the packets triggering the alerts. This will surely aid the operator in his investigation on the signature pattern and the packet payload; thus help identify the cause of the false alarms. The prototype system allows the users to view the packet payload by double clicking on the alert's id number in the first column. Figure 7.11 has presented a sample of the payload page.

7.6 The Implications of the Practical Evaluation

The SMART prototype system is developed and compiled as a standalone application, which is designed to run on any Windows or Linux system. The viability of the approach implemented on the proposed system has been measured through the practical demonstrations run on the 1999 DARPA data set and the University of Plymouth private data set.

Despite its practicality, there are several problems encountered with the system during the evaluation. The issues range from a less critical problem, for instance the interface design, to a more serious subject, such as the issue of the memory consumption. To gain a better understanding of these concerns, following explains the issues in more details and how they are implicated as the most influential factors that affect the system performance.

1. Database change

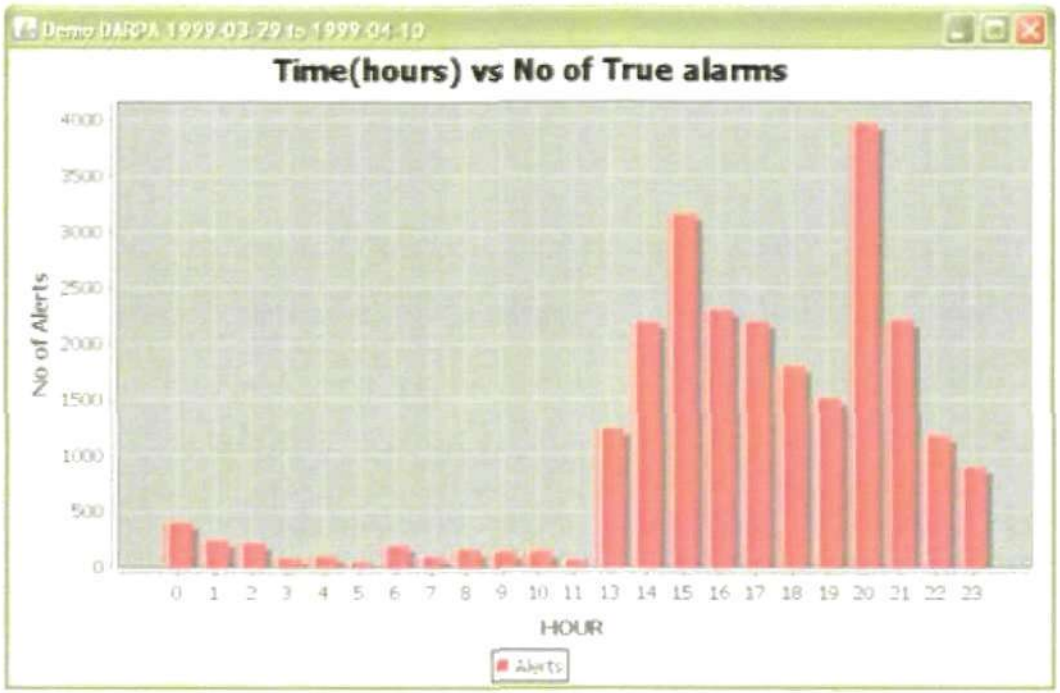


Figure 7.18: SMART - Time (hours) vs No of True alarms

In terms of its data storage, the prototype system has been programmed to use only one pre-defined database in whole correlations. In fact, there is no feature or option provided for the user to change or add a new database. In the practical demonstration, "darpa2" is a database that was generated by BASE tool and has been applied as the main database of the correlation. It contains all details of alerts information from the 1999 DARPA testing data. Since the system is not yet fully designed to optimise the system flexibility, the value for the database is set in the coding itself. In the interest of efficiency, the system could be enhanced in the future to enable the use of multiple databases and also allow the user to select a database as a primary source of the correlation.

2. Logarithmic scale charts

   All diagrams generated in the output module are categorised as a linear chart type (standard type). However, due to a high number of data (that is alerts) being presented, the linear chart type is not an optimal choice for the data illustration. The distance of which always represents the same absolute changes in price. Owing to this issue, it might be worth describing the data in a logarithmic chart type, which is useful to present data with large differences in scale on the same chart. Unlike the linear chart that represents the same absolute change, a given distance of the logarithmic chart represents the same percentage change in price. For instance, the distance from 10 to 100 on a logarithmic chart is the same as the distance from 1 to 10 on a linear chart, but the former distance is ten times greater on a linear chart.

3. Improved signature plot diagram

   The signature plot diagram presented on the "Signature Analysis" tab undoubtedly provides

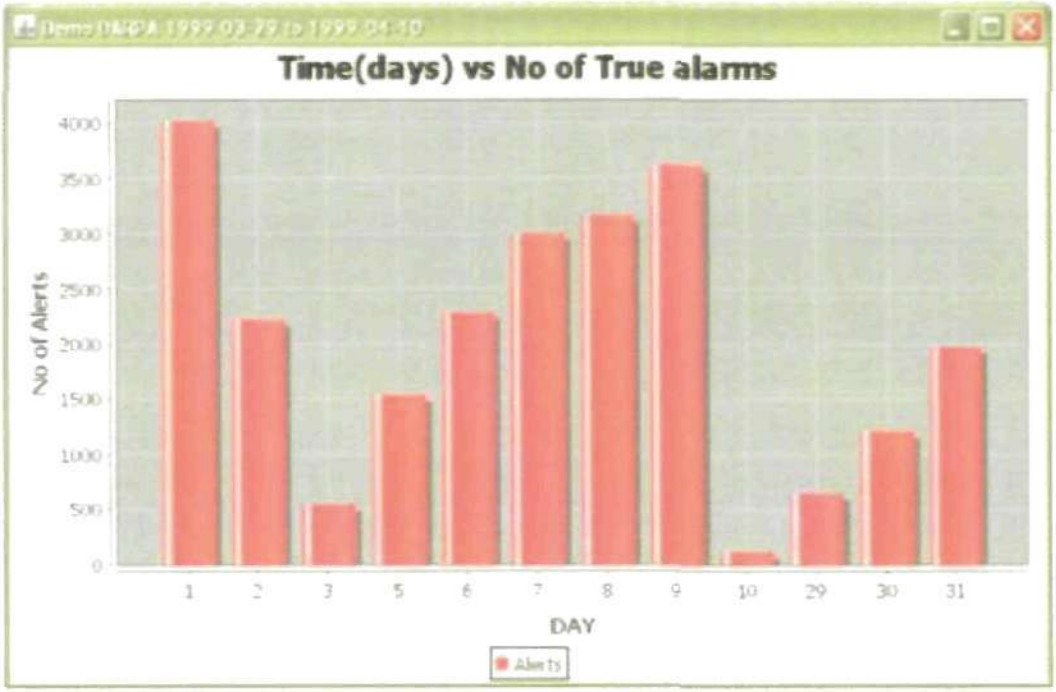


Figure 7.19: SMART - Time (days) vs No of True alarms

a general overview about the distribution of IDS alerts from a particular signature. However, the diagram seems to be lack of informative details to trace the elements illustrated in the figure, such as the IP addresses and the exact time of occurrence. It would be better if an additional table/description or link is given in the future to connect the alerts described in the plot diagram to the main alert table. This way the diagram will be more useful for the analysis process.

4. Memory usage

   If a Java application involves a large amount of data processing (consuming large amounts of memory) or is long lived, there is a possibility that out of memory exception can be thrown. With a significant (infinite) number of alerts processed by the application, the system on which the program is running may have run out of physical and virtual memory. Besides, the creation of a large number of individual charts, that is chart window (without closing the unwanted charts) may just be another cause of this error. Therefore, several ways to address this problem is by modifying the maximum heap size of the virtual machine, creating multiple charts on a window, or even redesigning the application. Designing applications for minimal memory consumption is not an easy job. Since it is more of a design problem, several programming techniques, such as the use of more efficient algorithms or subdividing tasks into smaller pieces are recommended to solve this error. In this scenario, the easiest way to answer this problem to change the maximum memory heap size, for example by using the Xms and Xmx JVM options: $-Xmx$ for maximum heap size and $-Xms$ for initial heap size, for example,

$$-Xms256m - Xmx512m$$

Figure 7.20: SMART - Time (months) vs No of True alarms

In the future, an analysis on the Java memory usage should be conducted to ensure minimal memory consumption and if necessary, the system could be redesign to achieve efficient algorithms.

5. Processing time

Since the system is implementing an unsupervised approach of correlation using SOM algorithm, the processing time may vary significantly and it is not uncommon that the processing time may take so long. This issue has certainly rendered the system inefficient. Bear in mind that the higher the number of clusters required, the longer the correlation will take. In the first stage correlation, there is no exact value of k (that is number of clusters) variable in each correlation. In other words, the number of clusters to be formed in this stage is unknown; therefore the value is then set as half of the length of data. On the other hand, the k value of the stage2 correlation has been defined as 2 as the correlation aims to group the alerts into two classes, namely clusters of true and false alarms. Therefore, by looking at the k values from both stages, there is no doubt that the first correlation will take time longer than the second classification due to its large k value. The only solution to address this issue in the future is perhaps revising the applied method, for instance by reducing the size of the map or the number units (neurons) in each correlation. Although determining an ideal map size is not a straightforward task but surely this will speed up the correlation process.

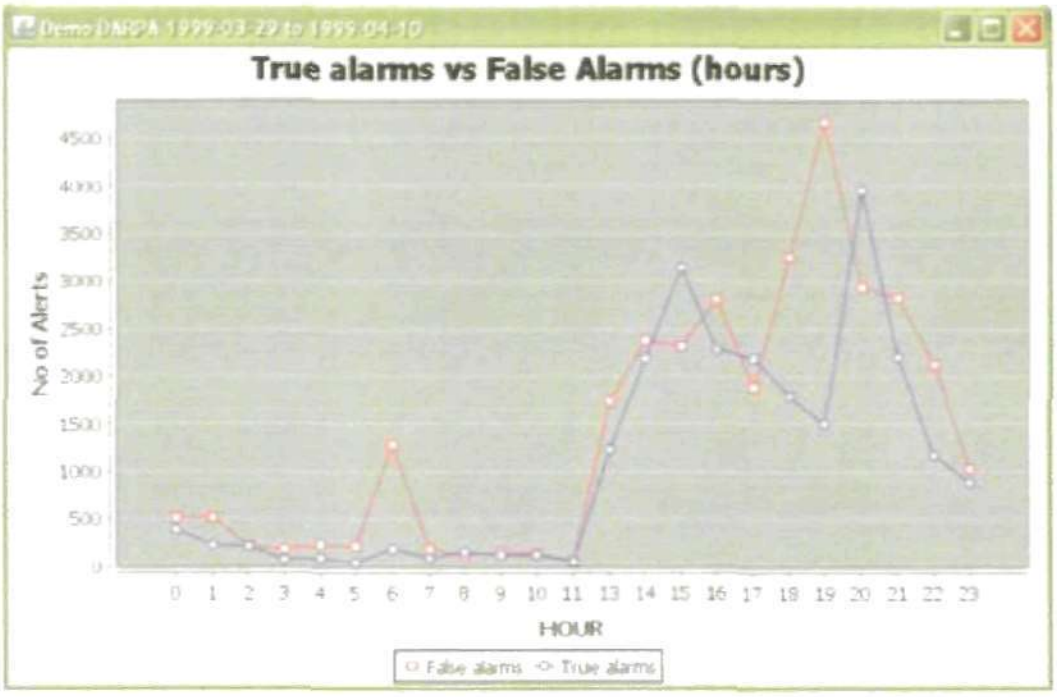


Figure 7.21: SMART - True Alarms vs False Alarms (hours)

7.7 Experiment Results

Having demonstrated the prototype system, it is now essential to evaluate the performance of the system itself on the complete DARPA data set and the real network traffic (University data set). The results of the correlations run by the system on both data sets are compared with the outcomes of the previous experiments described in Chapter 4. In this case, an evaluation is conducted to find out whether the false alerts identified in the earlier experiments can be correctly classified by the prototype system. The results of the correlations on both data sets are presented in the following subsections.

7.7.1 DARPA Data Set 1999

As previously mentioned in subsection 7.5.1, due to the memory issue, only 56,119 alerts from the two weeks DARPA 1999 testing data set were fed into the correlation system. A total of 13 hours of alerts were skipped during the correlations.

The evaluation revealed that 57% of the total alerts processed by the correlation system were classified as false alarms. And more importantly, at least 50% of the total false alarms identified in the earlier experiments were correctly detected as false alerts by the correlation engine. Table 7.1 below presents top five false alarms and its reduction rate using the correlation system.

Overall, the experiment yielded a quite promising result; with most of the noisy alerts such as the ICMP and web bug alerts were effectively eliminated up to 78% in the best scenario. Indeed, the superfluous alerts such as the web bug alerts (from "INFO web bug 1x1 gif attempt" signature) contributed to 35% of the total false alarms. Such alerts were regarded as pure false positives

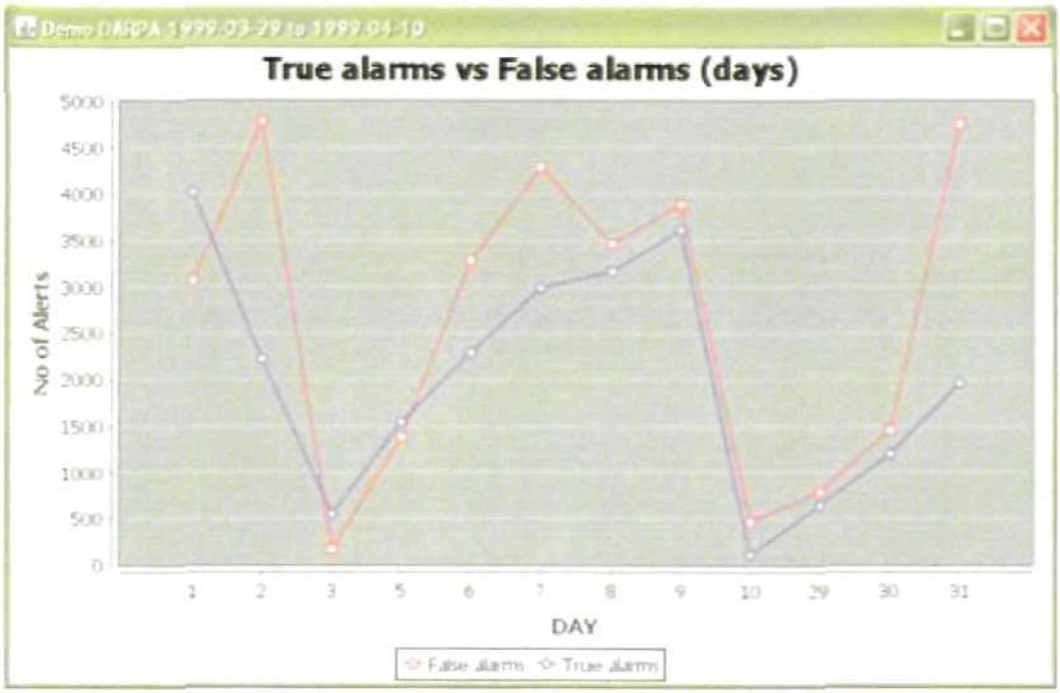


Figure 7.22: SMART - True Alarms vs False Alarms (days)

Table 7.1: DARPA – Reduction Rate of Top 5 False Alarms

No	Signatures	No of False Alarms		Reduction Rate (%)
		Before	After	
1.	INFO web bug 1x1 gif attempt	22,559	17,696	78.44
2.	ICMP Destination Unreachable Port Unreachable	14,017	6,465	46.12
3.	ICMP Echo Reply	11,275	2,508	22.24
4.	ICMP Ping	5,259	2,639	50.18
5.	CHAT IRC message	1,829	456	24.93

since they were not in common with any true alarms. A complete list of signatures and their corresponding reduction rate is presented in details in Appendix B.

Although the preliminary experiment presented in Chapter 5 yielded a remarkable result; up to 99% reduction of false alarms, the experiment on the complete data set produced a far more realistic result, ranging between 20-70%. The cause of the difference is the amount of alerts used in the experiments and the variation of the corresponding signatures. The preliminary experiment used only a small of chunk of DARPA data set, which inadvertently contains a high number of "noisy" false alerts that can be effortlessly filtered by the correlation engine. On the other hand, the complete data set contains a variety of false signatures, including uncommon false signatures, which are not easily spotted by the proposed system.

Like other correlation engines, the proposed system does suffer from one major drawback, namely its inability to filter the false alarms from infrequent signatures since it relies on the frequency and the time interval of alert occurrences to distinguish between the true and the false positives. Infrequent signature refers to a signature that rarely triggers any alert. Only six percent of the total

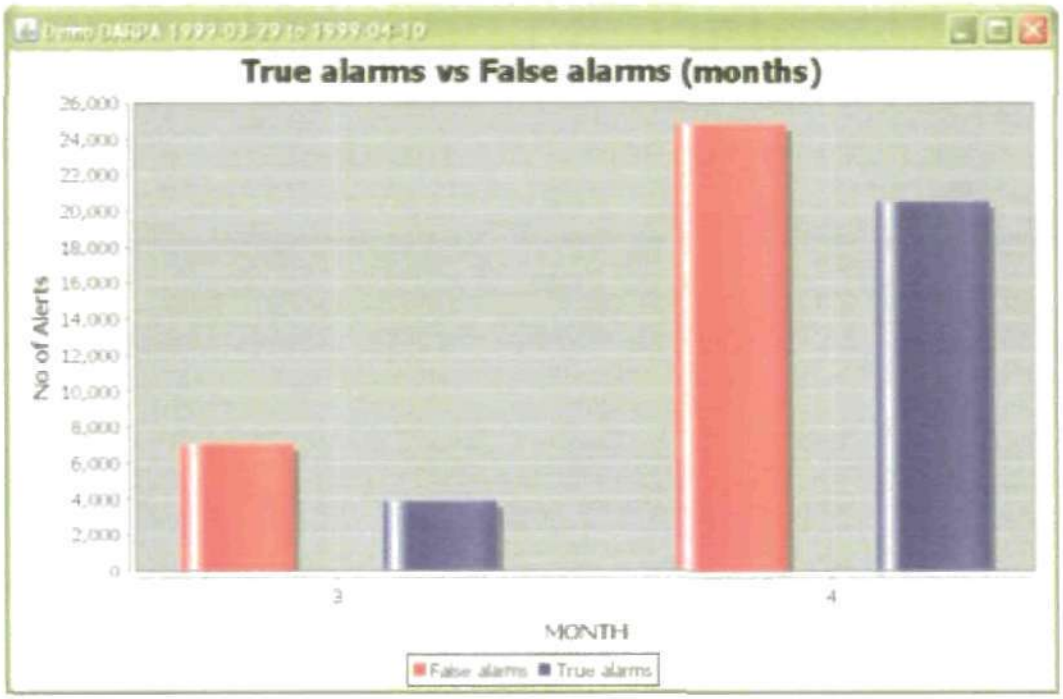


Figure 7.23: SMART - True Alarms vs False Alarms (months)

Table 7.2: DARPA – Misclassified Alerts

No	Signatures	Misclassified Alerts		Reduction Rate (%)
		Before	After	
1.	(portscan) TCP Portscan	5	123	118
2.	WEB-CGI phf access	4	72	68
3.	WEB-MISC handler access	2	72	70
4.	WEB-CGI test-cgi access	0	73	73
5.	RPC Portmap listing TCP 111	0	40	40
6.	SNMP missing community string attempt	0	2	2

non-noisy false alerts (that is excluding the top four false alarms) were perfectly detected. The lists of the unfiltered false alarms and correctly identified true alarms from the DARPA data set are presented in Appendix B.

In terms of the true positives, the system fared well in detecting the true alarms. It did perfectly identify and group the real alarms into the category of true positives. Having said that, there is a possibility that the correlation engine might have misclassified a relatively small percentage of the true alarms as false alarms. For example, the system incorrectly classified two out of five "SNMP missing community string attempt" alerts as false alarms. The cause of this error was that the misclassified alerts appeared to have higher frequency rate and lower time interval than the other three alerts. This occurs when the true alerts are classified into wrong clusters (that is cluster of false alerts) in the stage1 correlation. Although such an issue can seriously affect the system performance, it is deemed to be a common limitation in the field of unsupervised classification. Consequently, future works could be carried out to enhance the correlation algorithm by using

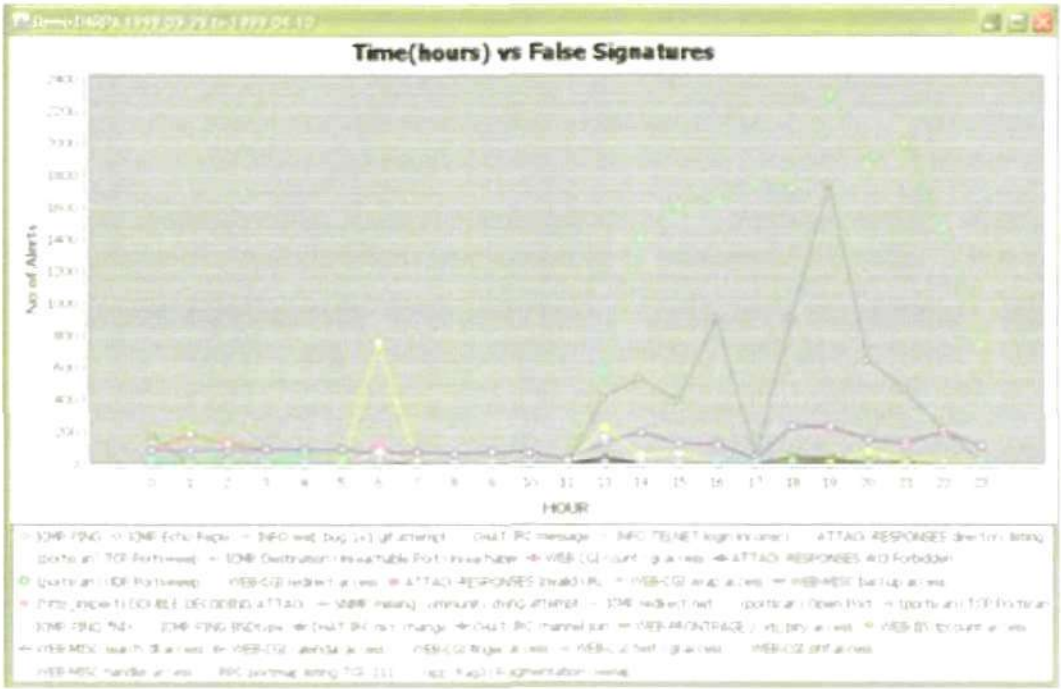


Figure 7.24: SMART - Time (hours) vs False Signatures

more enhanced SOM methods such as Time Adaptive Self Organizing Map (TASOM). TASOM is an extension of basic SOM, which provides flexible correlation parameters, such as learning rate and neighbourhood size (Shah-Hosseini and Safabakhsh, 2000). Table 7.2 presents a list of misclassified alerts.

7.7.2 University of Plymouth Data Set

As for the University private data set, the first 15 days (starting from 17th May 2007 to 31st May 2007) of data that contributed to total 54,893 alerts were fed into the correlation engine. The alerts were triggered by 37 signatures and 31 of which generated hundred per cent false positives.

In this context, the false alarms highly outnumbered the true alarms. Approximately 99% of the total alerts were asserted as false positives in the previous experiments. Significantly, the correlation system yielded a remarkable result by detecting up to 72.5% of the total false alerts identified previously. Table 7.3 shows a list of top five signature rules, the number of alerts relative to the signatures and also their corresponding reduction rate.

From the figures shown in Table 7.3, it is obvious that the prototype system is effective in filtering noisy or excessive alerts, which accounted for 90% of the total false alerts, such as those from "WEB-IIS view source via translate header" and "WEB-MISC robots.txt access" signatures. Both signatures were are affirmed to raise only false positives and to have high frequency rate as well as short time lapse between alert occurrences. In this example, 93.15% of false alarms from WEB-MISC robots.txt access signature were effectively filtered, whilst the WEB-IIS view source via translate header alerts were reduced by 74.24%.

More importantly, none of the true alarms in this example were misclassified as false alarms. A

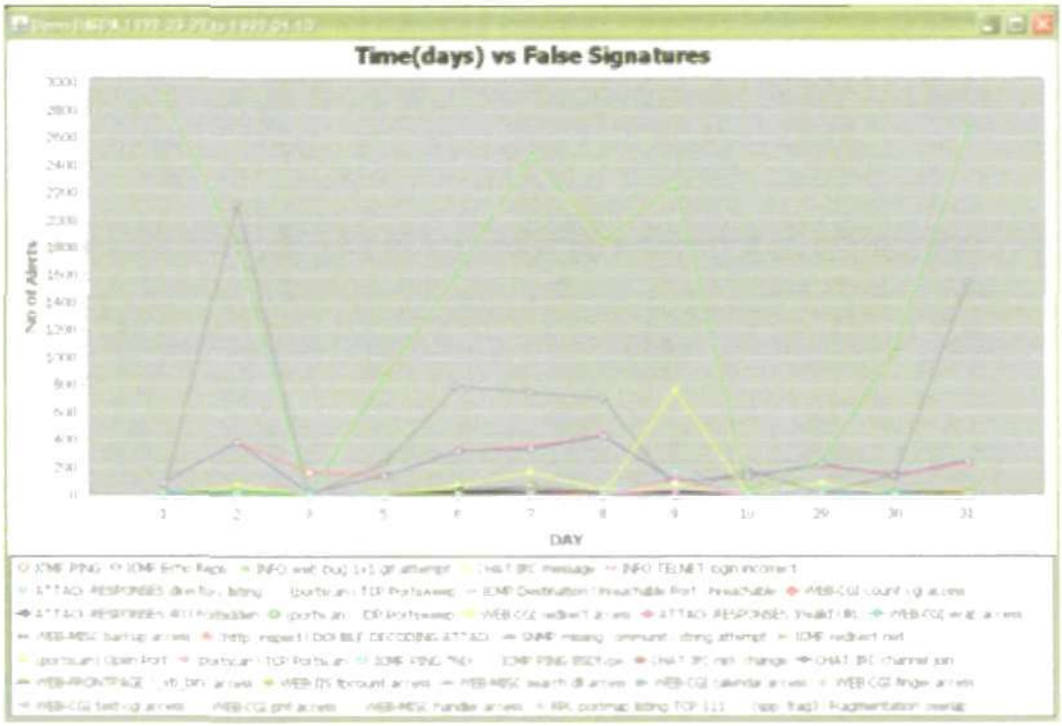


Figure 7.25: SMART - Time (days) vs False Signatures

Table 7.3: University of Plymouth Private Data – Reduction Rate of Top 5 False Alarms

No	Signatures	No of False Alarms		Reduction Rate (%)
		Before	After	
1.	WEB-IIS view source via translate header	33,902	25,170	74.24
2.	WEB-MISC robots.txt access	11,073	10,315	93.15
3.	ICMP L3retriever Ping	4,355	2,097	48.15
4.	(http_inspect) BARE BYTE UNICODE ENCODING	2,489	1,385	55.64
5.	POLICY Google Desktop activity	1,272	456	35.85

complete list of correctly identified true signatures as well as a list unfiltered false alarms from the University of Plymouth data set are presented in Appendix B.

Despite its ability to effectively detect various noisy alerts, the system was unable to filter false positives from uncommon signatures. This means that a rare signature with a low frequency rate is more likely to be classified as a true alarm. In fact, only 37.6% of the total non-noisy false alerts (that is excluding the top three false alarms) were correctly flagged as false positives.

Compared to tuning method, which was discussed in Chapter 4, SMART reveals a better average reduction rate, as shown in Figure 7.32.

As depicted in Figure 7.32, SMART performed less well in both WEB-IIS view source via translate header and ICMP L3Retriever Ping cases. As discussed in Chapter 5, the proposed system relies on the time interval and frequency rate of each signature to filter the false alarms. The occurrence rates of WEB-IIS view source via translate header event are varied, depending on the allocated time windows. Therefore, alerts with lower frequency rate (refers to Chapter 6 to see

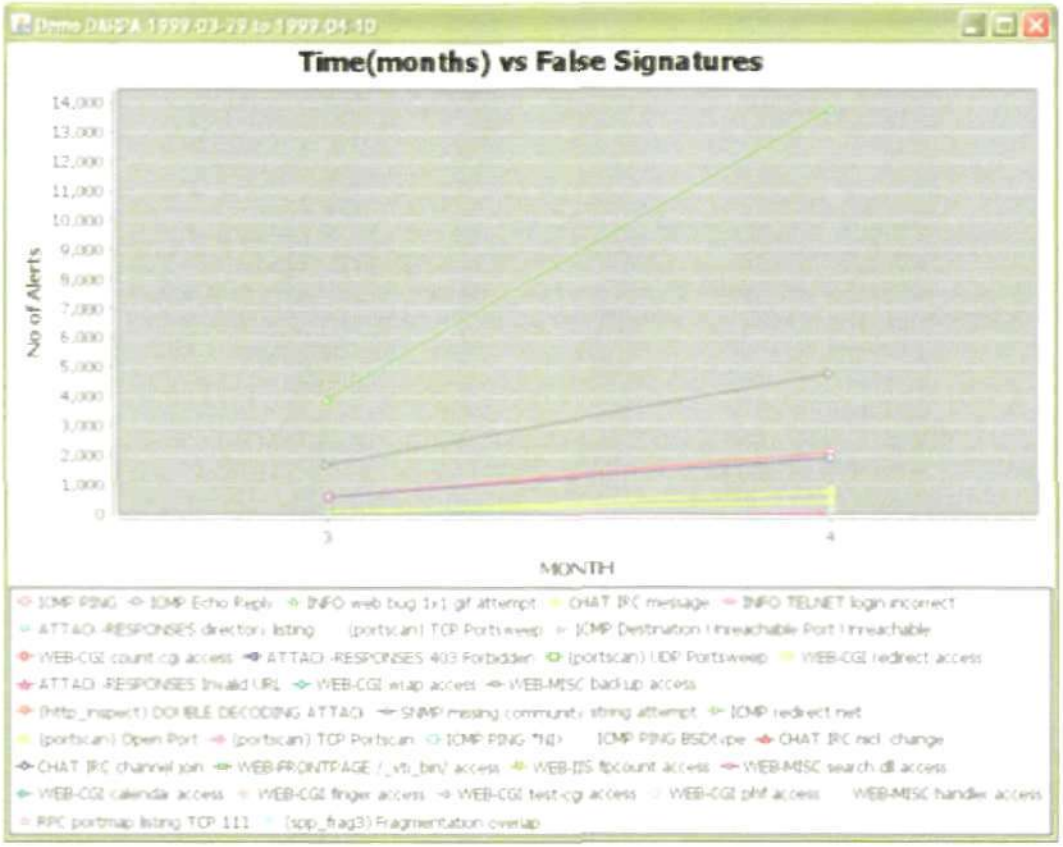


Figure 7.26: SMART - Time (months) vs False Signatures

how frequency rate and time interval are calculated) are more likely to be flagged as true than false alarms. Conversely, tuning looks for overly specific pattern to reduce false alarms. Whilst the technique effectively eliminates false alarms by 90%, it is more inclined to produce false negatives.

In terms of ICMP L3Retriever Ping, the thresholding method (refers to Chapter 4 to see how this signature is tuned) appeared to outperform the proposed system. This signature triggered a significant number of constant alerts, which could be easily suppressed using the thresholding technique. It is worth remembering that the only purpose of performing thresholding is to limit the occurrence of the alerts within a particular period. This, therefore, creates a possibility of missing real alerts.

On the other hand, SMART could only detect 48% of false alerts from this signature. As the alerts contain the same destination IP address, SMART assumed the alerts were triggered due to the same event if they occurred within a three-minute time frame (as explained in Chapter 6). As such, the occurrence rate of events related to this alert was much lower than the frequency rate of the alert itself. With the low event frequency rate, the alerts are therefore more likely flagged as true alarms.

Although SMART appears less effective in both cases, it efficiently addresses the issue of subjective rule suffered by the tuning method (as explained in Chapter 4). This is demonstrated by its ability to filter WEB-MISC robots.txt access up to 93%, whilst only 10% can be reduced by tuning

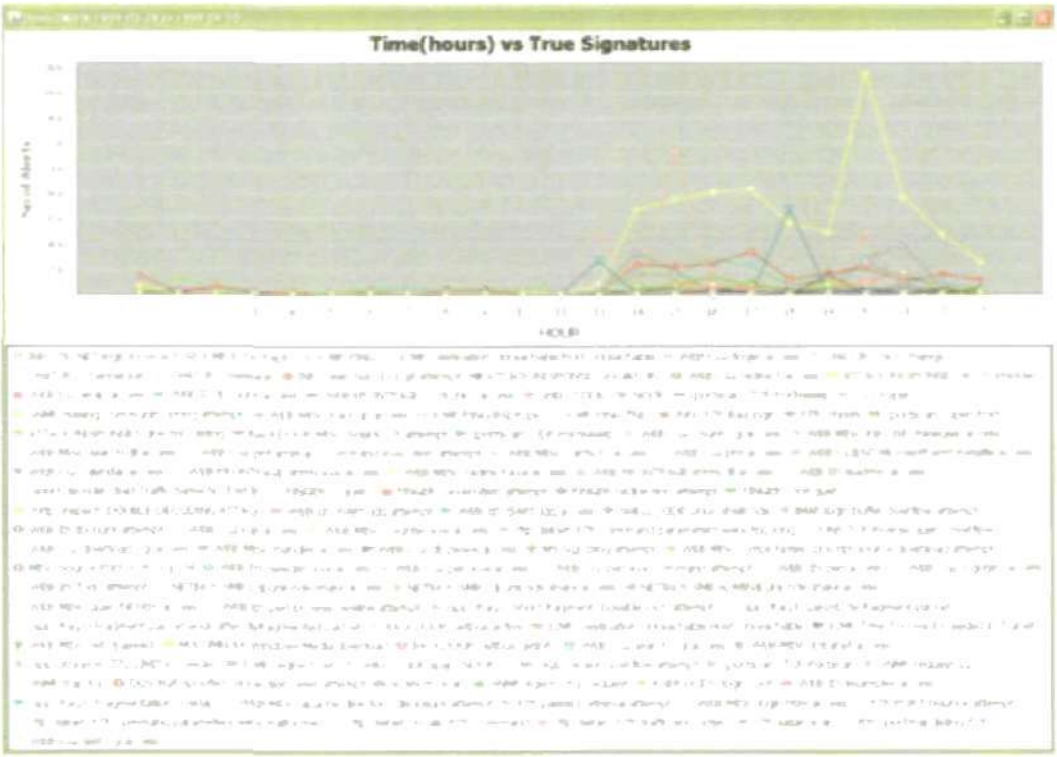


Figure 7.27: SMART - Time (hours) vs True Signatures

(as shown in Figure 7.32).

In order to achieve optimal results, SMART could be used to validate IDS alerts and tuning should be performed on signatures of the validated false alerts. The combination of both techniques could surely enhance the performance of IDS in filtering false alarms. Indeed, SMART presents a more condensed view of the false alarm issue that can significantly help the administrator in performing future tuning.

Overall the system yielded a promising result, with an average reduction ratio of 65% was achieved on both DARPA and private data set. Although the result proved the efficiency of the proposed model in detecting noisy (superfluous) alerts, not all of them could be perfectly filtered. Alerts from the same signature could have different frequency rates and time intervals, depending on the time windows the alerts belonged to. Therefore, an alert with a lower frequency than others from the same signature could be detected as a true alarm.

Compared to other existing machine learning techniques, such as data-mining and supervised learning, as listed in Table 5.5 in Chapter 5, the proposed system appears less effective in reducing the false alarm. The result, however, is positively biased given its reliance on the validation data set. On the other hand, the system demonstrates a reasonable outcome when it is compared to other unsupervised clustering algorithm, called autoassociator proposed by Smith et al. (2008). Their system obtained an accuracy of 67.4%, which is 5.1% lower than SMART (comparing with the SMART's best performance on the University of Plymouth data set - 72.5%). Moreover, this experiment also highlighted a common built-in shortage suffered by the unsupervised methods, namely the sensitivity of the detection results to the parameters, which are difficult to be deter-

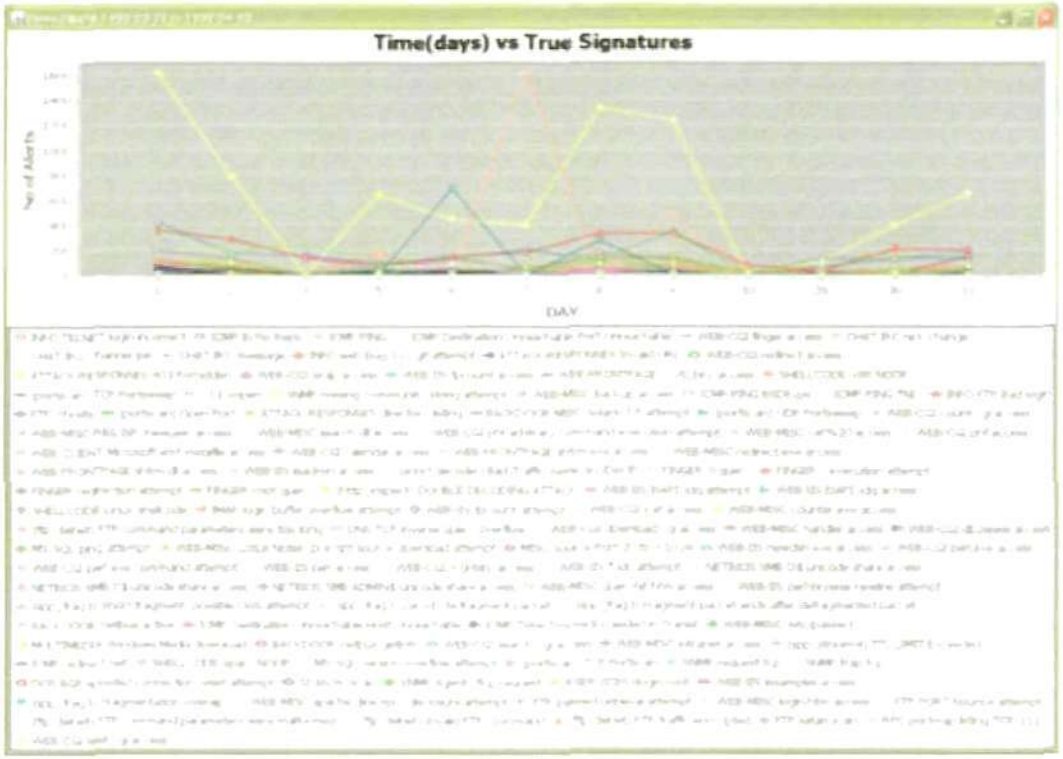


Figure 7.28: SMART - Time (days) vs True Signatures

mined. In this context, the parameters are the number of clusters and the size of time windows for the correlation.

7.8 Conclusions

This chapter described the SMART prototype system, demonstrating the features and interfaces of the SMART prototype system, detailing the system functional requirements via modelling languages and evaluating the system performance based upon the experiments on DARPA and University private data sets. The system interface consists of two main components; input and output interfaces. The user-friendly input interface enables the user to enter inputs and initiate a correlation, whilst the output interface presents the results of the correlation to the user.

The key objective of describing the system functional requirements is to gain better understanding of the inherent behaviours of the system. The functional requirements analysis specifies the interaction between the proposed system and the external agent – the administrator and explains the structure as well as the processes involved within the correlation. The analysis presents several modelling languages; including a behavioural diagram, an interaction diagram, an activity diagram and a static structural diagram.

Significantly, the result of the experiments proved the effectiveness of the SMART in filtering noisy alerts, with an average reduction rate of 65%. It is worth remembering that the accuracy of the unsupervised-based SMART system significantly depends on the initial parameters (in other words, the number clusters applied in the correlation and the size of time windows). This indicates

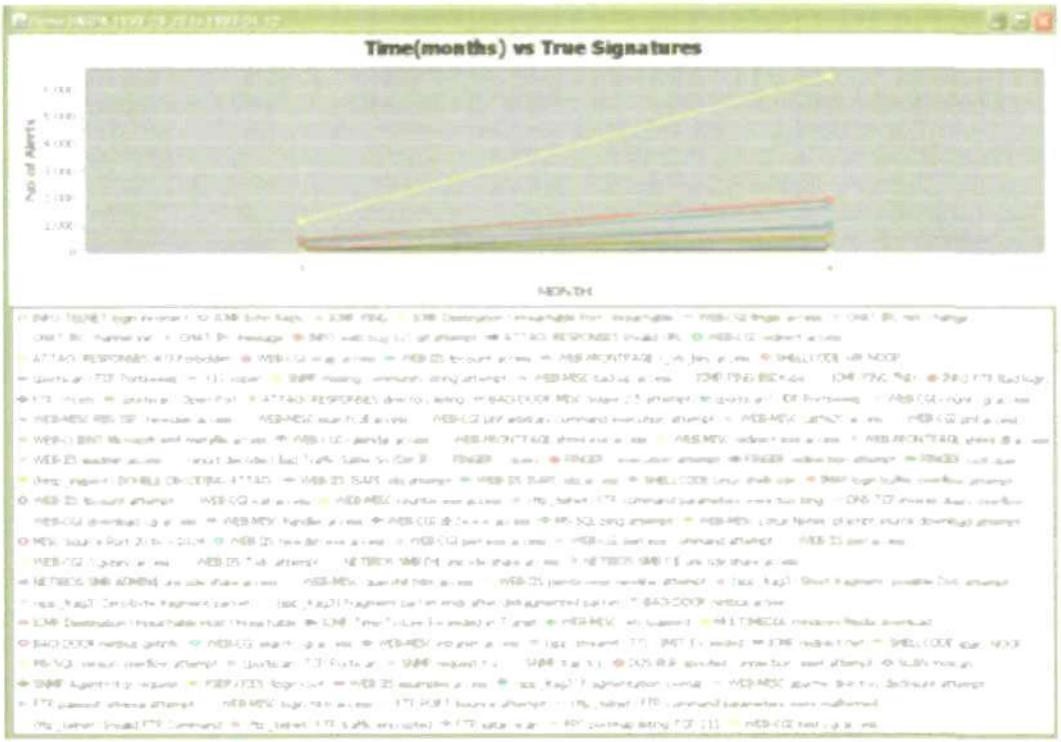


Figure 7.29: SMART - Time (months) vs True Signatures

the major limitation of the unsupervised methods, which makes them less favourable than the supervised learning methods. Having said that, unsupervised techniques involve less computational overhead than supervised learning since no domain knowledge and maintenance required once all the initial parameters are determined.

Overall, the contributions of this study can be summarised as follows:

1. Proposing a novel framework for off-line correlation of alerts based on their attributes. The proposed framework classifies IDS alerts based on the most relevant features that highlight their behaviours.

2. Developing a two-stage correlation approach based on time windows using unsupervised algorithm. The key objectives of this method are to identify or aggregate alerts from the same attack instance and to classify alerts into two categories, true and false alarms

3. Proposing a front-end interface that offers a statistical tool, allowing the administrators to evaluate the trend of IDS alarms and to do a comparison between the true and false alarms.

The development of the prototype has also helped to enlighten the SMART architecture, especially in areas relating to how alert features are selected and used in the correlation. For instance, the two leading attributes; the time interval and number of events were chosen to reflect unique characteristics of the alerts. Overall, the prototype has aided to prove the viability of the SMART architecture. It has provided a practical validation of the ability to achieve an automated alarm reduction tool. In a sense, the system in its current form is deemed to represent an enhancement on existing false alarm reduction approaches.

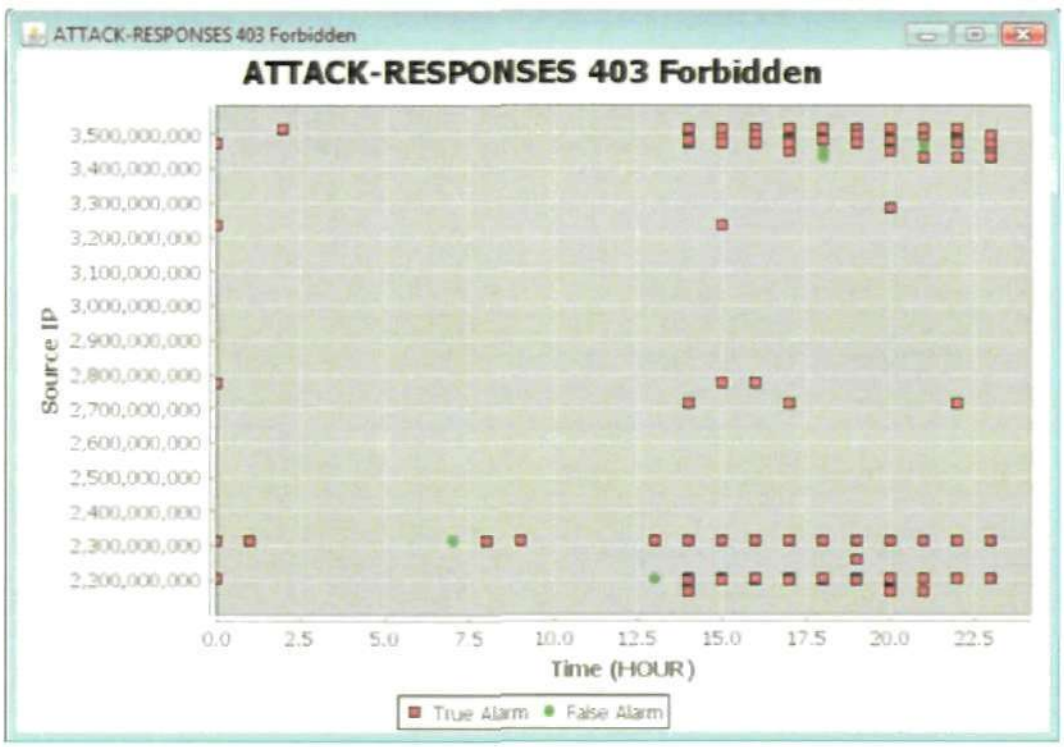


Figure 7.30: SMART - ATTACK-RESPONSES 403 Forbidden (Plot Diagram)

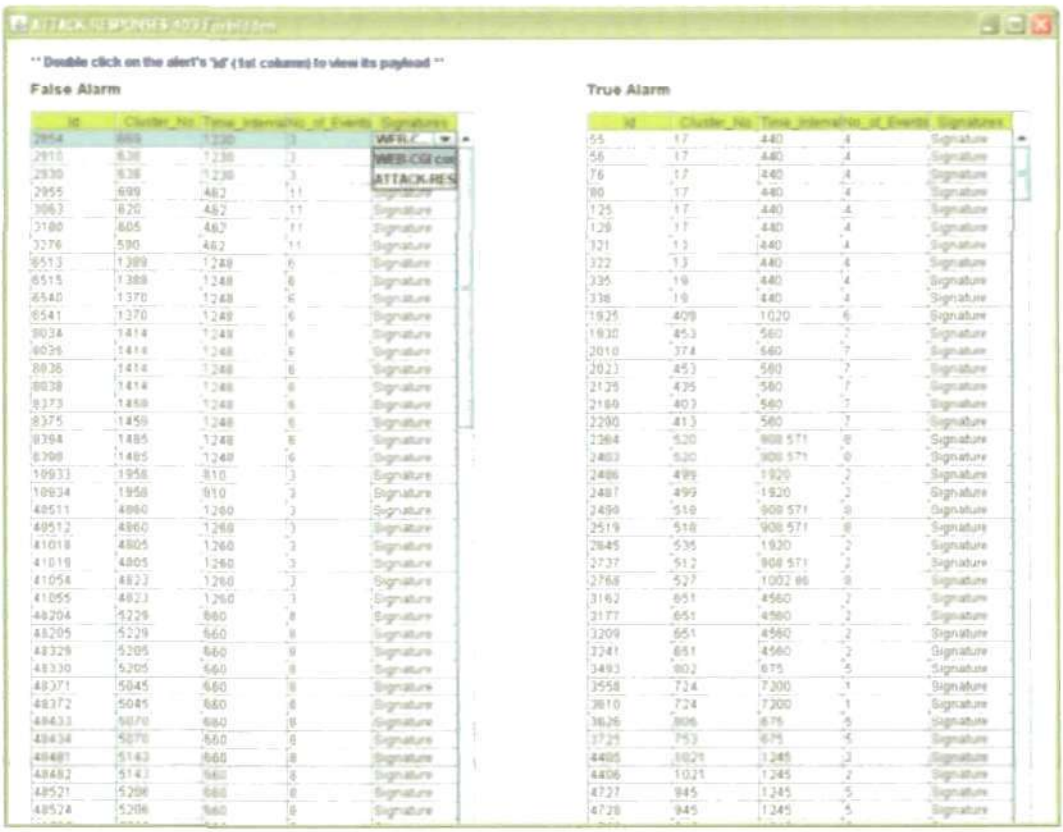


Figure 7.31: SMART - ATTACK-RESPONSES 403 Forbidden (Alert Table)

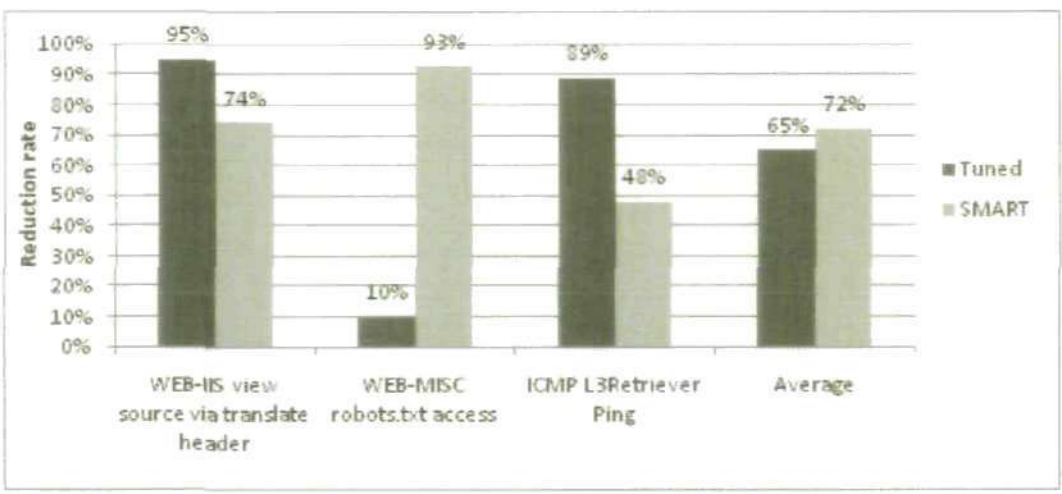


Figure 7.32: Tuning vs SMART

8 Conclusions

This chapter summarises the thesis by reviewing the project's achievements and underlining the main limitations of the research. It then continues to detail potential new research fields within which the work proposed could be improved in the future.

8.1 Achievements of the Research Programme

On the whole, the research has achieved all of the objectives initially identified in Chapter 1, with a number of experiments and works carried out for the development of a new alarm correlation system. The detailed achievements are:

1. A practical exploration of the problem of false alarms. The experiments conducted on both synthesized network data (DARPA) and real network traffic (University of Plymouth private data set) enabled first-hand assessment of the IDS performance and the quality of alerts presented.

2. A practical investigation and assessment of the feasibility of fine tuning (as described in Chapter 4). This revealed that the conventional alarm reduction method does help reducing false alarms but it requires a thorough examination of the protected environment by the qualified IT personnel before the tuning can be performed. In fact, the tuning requires a frequent update to keep up with the flow of new vulnerabilities or threats discovered; otherwise it might increase the risk of missing real attacks.

3. Investigation of the drawbacks of existing alarm correlation techniques and the problems that lead to excessive false alarms (Chapter 2 and 3). Contributions of previous alarm correlation studies have also been reviewed, determining the scope and necessity for further enhancement (Chapter 3).

4. The proposal and realisation of a novel alarm correlation concept using unsupervised algorithms (Self Organising Map and K-Means). The proposed technique consists of two stages of correlation, aggregating alerts from the same attack instance and classifying alerts into groups of true and false alarms, and has been assessed using alerts from both synthesised and real networks (Chapter 5). A number of alert attributes were selected and used to define the relationships between alerts by clustering them based on their similarity.

5. The design and development of the SOM K-Means Alarm Reduction Tool (SMART) architecture, in which the unsupervised techniques are implemented to improve alarm management (Chapter 6). The main concepts of the proposed architecture, aside from minimising the false alarm rate, are the ability to correlate all alerts triggered by a single event and to help discover the potential attack scenarios.

6. The implementation of a working prototype of the SMART system in order to validate the viability of the proposed architecture and the ability to achieve an automated false alarm reduction tool (Chapter 7). Apart from adopting the aforementioned correlation scheme within the system, the prototype system also feature a user-friendly interface and a graphical reporting tool that enables the administrator to fully analyse the correlated alerts and also to create a graphical report.

Several papers concerning the research topic have been presented and published at refereed journal and conferences (the papers are attached in Appendix G) and have received positive feedback from the associated reviewers. Therefore, it is considered that the research has made appropriate and useful contributions to IT security field, and particularly in the domains of intrusion detection and alarm correlation.

8.2 Limitations of the Research

Despite having met all objectives defined in Chapter 1, a number of limitations associated with the work can be identified. The main points in this respect are listed below.

1. Since in certain cases, a large amount of data (alerts) will need to be processed in a single correlation, memory has always been an issue. The problem is derived from the MATLAB application that serves as the main language of the proposed correlation engine. It is not uncommon that MATLAB raises "Out of Memory" error when the system has in fact run out of heap space to hold all variables. In this context, it is believed that the engine could only process less than three thousand alerts per correlation (as mentioned previously in Chapter 7). As such, this memory limitation could render the system impractical in a wider context. In terms of its processing time, one correlation may take longer than other depending on the size of data correlated and the number of units (neurons) applied. The bigger the data is (consequently, a higher number of units applied), the longer the processing time will be. This issue has become one of the significant drawbacks suffered by the proposed system. The longest processing time ever measured is 7-8 hours, when the number of alerts processed close to three thousands.

2. The correlation engine has been specifically designed and developed to process alerts from the signature-based IDS only; particularly Snort IDS. The available time did not permit the development of such system to focus on other IDSs such as anomaly-based IDS or other signature-based examples. Although the proposed system is not widely applicable and is limited to Snort only, it is considered valuable since Snort is the world's most widely used IDS and has been named one of the greatest open source programs of all time (Dineley and Mobley, 2009).

3. The SMART system would not be applicable to on-line correlation since it heavily relies on two key features, the frequency and time interval between alerts to classify the alerts. Besides, both attributes could only be computed from off-line alerts (in other words, alerts that have been raised and logged into a database), meaning that the correlation could only be performed after the intrusion detection is carried out. In spite of this limitation, however, it is believed

that the system would be more effective to run as an off-line filtering system as it allows the administrator to evaluate the original alerts before being correlated.

8.3 Suggestions and scope for future work

As for potential future work, there are a number of areas, in which activities could be carried out to build on what was done in this project. The details of the prospective works are summarised below.

1. Although overfitting has been compensated with the $K-$Means algorithm (as explained in Chapter 5), there is still an issue of dead centres, caused by a large number of map units being generated compared to the data set. Due to the strong topology relations of the SOM, these dead centres are located between data clusters and thereby introduce data for the $K-$Means algorithm in areas where no real data is. Thus, future work can be conducted to boost the system performance by optimally minimising the number of neurons applied whilst preventing the issue of under-fitting.

2. The correlation process involves several model selection procedures, for example choosing the best number of units for the correlation map and selecting a good model based on the frequency and the SSE value. Although current selection method is proved feasible in practice, however, it is deemed necessary to improve the quality of the selection process by adopting a better measurement theory. This aspect, nevertheless, represents an issue for further research in its own right. One of the potential methods to be applied in this context is Minimum Description Length (MDL) principle. The MDL is a technique for inductive inference that provides a basic solution to the model selection problem (Rissanen, 1978). In fact, it is commonly used in model selection process to determine the model complexity and give a better indication on the model quality. Apart from providing an approach to determine the finest model, MDL also provides a natural safeguard against overfitting (Grunwald, 2005). In the future work, it is also essential to review or take a close look at the applied algorithm in order to enhance the correlation system; for example by replacing $K-$Means with a more robust algorithm such as Robust Growing Neural Gas. This, therefore, represents another research area, in which the work presented in the thesis can be enhanced.

3. The proposed system aggregates alerts related to the same attack into a cluster. A future work should be carried out to construct a potential attack scenario by identifying the logical connections between alerts for each cluster.

4. As mentioned earlier, the proposed system has been specifically designed for Snort IDS only. The input application, thereby, is limited to Snort-based alerts. Given this limitation, it is beneficial that the future works should be directed to further improve the correlation engine by focusing on more types of IDS such as other signature-based IDS and anomaly-based IDS.

5. Improving the front-end of the system or creating a more interactive user interface, is another area that can significantly boost the presentation of the system. Therefore, revising the chart scale, the signature plot diagram (as described in Section 7.6) and providing a comparison of the results from different correlation time frames could represent other opportunities for development, in which the work presented can be expanded.

8.4 The Future for Automated Alarm Correlation Systems

With the widespread use of computer networks, the number of attacks has grown extensively. In fact, the significant increase of everyday life dependency on Information and Communication Technologies has intensified the importance of survivability of networks. Intrusion Detection System (IDS) has been an essential component of a complete defense-in-depth architecture for computer network security. Although signature-based IDS is believed to produce fewer false alarms than the anomaly-based system, the packet inspection method or the fine grain analysis applied by signature-based IDS causes the system to produce a high number of false alarms. The source of such a large amount of alarms is induced by the nature of some categories of attacks which send a large number of malevolent packets. Additionally, many attacks are launched in a sequence of steps. The valuable information for the network administrator relies on the aggregation alarms related to the different steps, rather than on each single alarm. More importantly, the alarms generated are often vague and could report the details of the detected event that are either too generic or too specific. As a result, the ability to automatically correlate the alarms and filter the false alarms is becoming increasingly important.

The development of an automated alarm correlation system now represents an active research field in the intrusion detection domain, with a large number of research studies have been focused on improving the correlation methods. In fact, current studies have significantly contributed to the enhancement of the IDS false alarm filtering system. Nonetheless, the problem has still been far from solved and there is still a significant scope to improve its performance. This research project has contributed to the domain at several degrees. It has highlighted the importance of an alarm correlation system, contributed in understanding of Snort-based alerts by proposing the Snort-based alarm reduction system (SMART). More importantly, it has focused on enhancing the filtering mechanism, by basing the correlation decision on the alarm frequency rate and time interval between events. As a consequence, the proposed alarm reduction system is able to aggregate alerts from the same event or attack instance and significantly reduce the false alarms. From a wider perspective, the correlated alerts produced by the proposed system has significantly improved the quality of IDS alerts, thus providing a better or condensed view of security issues to the administrator.

The implementation of such approach and the proposal of the user-friendly alarm analysis interface will enable the IDS alarms correlation technologies to mature. This will eventually enhance the IDS performance without reducing the values of the alarms generated, instead offer a better alarm presentation (quality) that allows the administrator to properly analyse the detected threats.

A Results of the Experiment on 1999 DARPA Data Set and Snort IDS

A.1 True and False Alarms per Signature

Table A.1: False alarms per signature

No	Signatures	False alarms
1.	INFO web bug 1x1 gif attempt	22559
2.	ICMP Destination Unreachable Port Unreachable	14017
3.	ICMP Echo Reply	11275
4.	ICMP PING	5259
5.	CHAT IRC message	1829
6.	ICMP PING BSDtype	883
7.	ICMP PING *NIX	883
8.	ATTACK-RESPONSES 403 Forbidden	792
9.	WEB-CGI redirect access	613
10.	(portscan) Open Port	601
11.	INFO TELNET login incorrect	594
12.	(spp_frag3) Fragmentation overlap	431
13.	ATTACK-RESPONSES directory listing	423
14.	UDP Portsweep	353
15.	WEB-CGI count.cgi access	297
16.	ICMP redirect net	281
17.	ATTACK-RESPONSES Invalid URL	265
18.	CHAT IRC nick change	228
19.	CHAT IRC channel join	219
20.	WEB-FRONTPAGE /_vti_bin/ access	174
21.	WEB-IIS fpcount access	171
22.	(snort_decoder) WARNING: ICMP Original IP Fragmented and Offset Not 0!	168
23.	WEB-CGI calendar access	167
24.	ICMP Time-To-Live Exceeded in Transit	157
25.	WEB-MISC search.dll access	103
26.	WEB-MISC backup access	99

Continued on next page

Table A.1 – continued from previous page

No	Signatures	False alarms
27.	WEB-CGI finger access	83
28.	SNMP trap tcp	79
29.	ICMP Destination Unreachable Host Unreachable	78
30.	(portscan) TCP Portsweep	61
31.	WEB-CGI db2www access	60
32.	WEB-MISC RBS ISP /newuser access	58
33.	(ftp_telnet) Invalid FTP Command	45
34.	WEB-MISC intranet access	42
35.	WEB-IIS fpcount attempt	38
36.	WEB-CGI wrap access	37
37.	WEB-MISC counter.exe access	32
38.	(http_inspect) DOUBLE DECODING ATTACK	30
39.	INFO FTP Bad login	23
40.	(spp_stream4) TTL LIMIT Exceeded	15
41.	WEB-CGI download.cgi access	14
42.	(ftp_telnet) FTP traffic encrypted	12
43.	WEB-IIS iissamples access	12
44.	WEB-FRONTPAGE shtml.dll access	11
45.	X11 xopen	11
46.	WEB-IIS iisadmin access	11
47.	WEB-MISC redirect.exe access	9
48.	WEB-CGI icat access	8
49.	BACKDOOR MISC Solaris 2.5 attempt	8
50.	(portscan) TCP Portscan	5
51.	WEB-CLIENT Microsoft emf metafile access	5
52.	ICMP Fragment Reassembly Time Exceeded	4
53.	WEB-MISC Lotus Notes .pl script source download attempt	4
54.	WEB-CGI phf access	4
55.	MULTIMEDIA Windows Media download	4
56.	WEB-IIS ISAPI .idq attempt	4
57.	WEB-MISC cat%20 access	4
58.	FTP passwd retrieval attempt	4
59.	WEB-CGI phf arbitrary command execution attempt	4
60.	WEB-CGI perl.exe command attempt	4
61.	WEB-IIS ISAPI .idq access	4
62.	WEB-MISC login.htm access	4
63.	WEB-CGI search.cgi access	2
64.	(ftp_telnet) Telnet traffic encrypted	2

Continued on next page

Table A.1 – continued from previous page

No	Signatures	False alarms
65.	(http_inspect) BARE BYTE UNICODE ENCODING	2
66.	WEB-MISC oracle web application server access	2
67.	SNMP trap udp	2
68.	WEB-MISC handler access	2
69.	RSERVICES rlogin root	1
70.	MS-SQL version overflow attempt	1
71.	MS-SQL ping attempt	1
72.	WEB-FRONTPAGE shtml.exe access	1
73.	Bad Traffic Same Src/Dst IP	1

Table A.2: True alarms per signature

No	Signatures	True alarms
1.	Open port	11130
2.	Web-misc apache directory disclosure attempt	5628
3.	ICMP Destination unreachable port unreachable	4634
4.	TCP Portscan	1283
5.	(spp_frag3) Fragmentation overlap	713
6.	Web-cgi phf access	700
7.	Web-cgi test-cgi access	696
8.	Web-misc handler access	696
9.	RPC Portmap listing TCP 111	398
10.	ICMP PING	339
11.	Attack response directory listing	233
12.	INFO Telnet login incorrect	225
13.	INFO FTP bad login	155
14.	SNMP Agent X/TCP request	147
15.	SNMP Request TCP	133
16.	(spp_frag3) Short fragment, possible DoS attempt	123
17.	(spp_frag3) zero byte fragment packet	123
18.	(spp_frag3) fragment packet ends after defragmented packet	118
19.	UDP Portsweep	67
20.	SNMP Trap TCP	54
21.	TCP Portsweep	38
22.	Misc source port 20<1024	26
23.	Finger/execution attempt	24

Continued on next page

Table A.2 – continued from previous page

No	Signatures	True alarms
24.	X11 Xopen	22
25.	Web-cgi perl.exe access	16
26.	ICMP PING *NIX	13
27.	ICMP PING BSD Type	13
28.	SHELLCODE X86 NOOP	13
29.	FTP command parameters were malformed	12
30.	FTP Port bounce attempt	12
31.	Web-cgi perl.exe command attempt	12
32.	Web-cgi cgi-bin /access	8
33.	Web-iis *.idc attempt	8
34.	Web-iis newdsn.exe access	8
35.	Web-iis perl access	8
36.	Web-cgi phf arbitrary command execution attempt	7
37.	Web-misc cat %20 access	7
38.	(ftp_telnet) FTP command parameters were too long	6
39.	(ftp_telnet) invalid TCP command	6
40.	Backdoor netbus active	6
41.	DNS TCP inverse query overflow	6
42.	ICMP echo reply	6
43.	NETBIOS SMB ADMIN$ unicode share access	6
44.	NETBIOS SMB C$ unicode share access	6
45.	NETBIOS SMB D$ unicode share access	6
46.	Web-frontpage /vti_bin/access	6
47.	Web-iis fpcount access	6
48.	Web-iis fpcount attempt	6
49.	Web-iis perl browse newline attempt	6
50.	Web-misc /etc/passw	6
51.	Web-misc queryhit.htm access	6
52.	SNMP missing community string attempt	5
53.	Backdoor netbus getinfo	4
54.	Finger 0 query	4
55.	Finger redirection attempt	4
56.	Finger root query	4
57.	ICMP Fragment Reassembly Time Exceeded	4
58.	Scan myscan	4
59.	SHELLCODE Sparc NOOP	4
60.	UDP Portscan	4
61.	FTP .rhosts	3

Continued on next page

Table A.2 – continued from previous page

No	Signatures	True alarms
62.	IMAP login buffer overflow attempt	3
63.	SHELLCODE Linux shellcode	3
64.	Bad Traffic Same Src/Dst IP	2
65.	DOS BGP Spoofed connection reset attempt	2
66.	FTP Satan scan	2
67.	MISC source port 53 to < 1024	2
68.	Warning: ICMP Original IP Payload > 576 bytes	2

A.2 Tables of Attack Types Detected per Day

Table A.3: Day 1 - 29th March 1999

Attack Type	Name	Alert	Quantity
U2R	yaga	Attack response directory listing	5
R2L	sendmail	SHELLCODE X86 NOOP	2
	Xsnoop	x11 xopen	2
	snmpget	SNMP missing community string attempt	3
	guesstelnet	INFO Telnet login incorrect	14
	guessftp	INFO FTP bad login	5
	ftpwrite	FTP .Rhosts	2
Probe	portsweep	ICMP destination unreachable port unreachable	6

Table A.4: Day 2 - 30th March 1999

Attack Type	Name	Alert	Quantity
DOS	land	Bad traffic Same src/dst IP	1
U2R	sechole	Attack response directory listing	23
R2L	phf	Web-cgi phf arbitrary command execution attempt	1
		Web-cgi phf access	1
		Web-misc cat%20 access	1

Table A.5: Day 3 - 31th March 1999

Attack Type	Name	Alert	Quantity
R2L	netcat_setup	ICMP Destination unreachable port unreachable	6
	imap	SHELLCODE X86 NOOP	2
		SHELLCODE Linux shellcode	2
		IMAP login buffer overflow attempt	2
	netcat_breakin	Attack response directory listing	4
	ncftp	FTP Commands parameter were too long	1
	named	DNS TCP inverse query overflow	2
		SHELLCODE X86 NOOP	2
		X11 Xopen	2
	guessftp	INFO FTP Bad login	150
	guest	INFO Telnet login incorrect	16
	guess telnet	INFO Telnet login incorrect	40
	snmpget	SNMP missing community string attempt	2
Probe	satan	ICMP Destination unreachable port unreachable	8
		FINGER/EXECUTION attempt	12
		Finger 0 query	2
		Finger redirection attempt	2
		Finger root query	2

Table A.6: Day 4 - 1st April 1999

Attack Type	Name	Alert	Quantity
DOS	teardrop	(spp_frag3) short fragment, possible DoS attempt	13
		(spp_frag3) zero byte fragment packet	13
		(spp_frag3) fragment packet ends after defragmented packet	12
R2L	netbus	Backdoor netbus active	2
	ncftp	(ftp_telnet) FTP command parameters were too long	1
	guest	INFO Telnet login incorrect	16
	xlock	X11 Xopen	4
	phf	Web-cgi phf arbitrary command execution attempt	2
		Web-misc cat %20 access	2
		Web-cgi phf access	2
		Web-misc /etc/passw	2
Probe	ntis	Web-frontpage /vti_bin/access	2
		Web-iis fpcount access	2
		Web-iis fpcount attempt	2

Continued on next page

Table A.6 – continued from previous page

Attack Type	Name	Alert	Quantity
		Misc source port 20¡1024	2
		Web-iis *.idc attempt	2
		Web-cgi cgi-bin /access	2
		Web-cgi perl.exe access	4
		Web-iis newdsn.exe access	2
		Web-iis perl browse newline attempt	1
		Web-iis perl access	4
		Web-misc queryhit.htm access	2
		NETBIOS SMB ADMIN$ unicode share access	2
		NETBIOS SMB C$ unicode share access	2
		NETBIOS SMB D$ unicode share access	2
	ipsweep	ICMP PING	12

Table A.7: Day 5 - 2nd April 1999

Attack Type	Name	Alert	Quantity
U2R	loadmodule	INFO TELNET login incorrect	4
	sechole	Attack response directory listing	34
R2L	xlock	X11 Xopen	4
	named	DNS TCP inverse query overflow	2
		X11 Xopen	2
		SHELLCODE X86 NOOP	2
	ncftp	FTP Command parameters were too long	2
	netbus	Backdoor netbus active	2
		Backdoor netbus getinfo	2
	named	SHELLCODE X86 NOOP	2
		DNS TCP inverse query overflow	2
		X11 Xopen	2
Probe	ipsweep	ICMP PING	12
	portsweep	ICMP Destination unreachable port unreachable	12
	ipsweep	204.233.47.21 → 172.16.114.50	
		ICMP PING	2
		ICMP PING BSD Type	2
		ICMP PING *NIX	2
		128.223.199.68 → 172.16.112.3	
		ICMP PING	2
		ICMP PING BSD Type	2
		ICMP PING *NIX	2

Continued on next page

Table A.7 – continued from previous page

Attack Type	Name	Alert	Quantity
		204.71.51.16 → 172.16.114.5	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
		207.114.237.57 → 172.16.114.4	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
		209.1.12.46 → 172.16.114.1	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
	ipsweep	194.7.248.153 → 172.16.112.1-254	
		ICMP PING	262

Table A.8: Day 6 - 5th April 1999

Attack Type	Name	Alert	Quantity
DOS	pod	ICMP PING	6
		(spp_frag3) Fragmentation overlap	28
		ICMP Fragment Reassembly Time Exceeded	2
	pod	ICMP Fragment Reassembly Time Exceeded	2
		Warning: ICMP Original IP Payload ¿ 576 bytes	2
		ICMP PING	4
	pod	ICMP PING	6
	neptune	TCP Portscan	7
		Open port	36
		SNMP request TCP	40
		SNMP Trap TCP	40
		Scan myscan	2
		SNMP Agent X/TCP request	40
		DOS BGP Spoofed connection reset attempt	1
U2R	loadmodule	INFO TELNET login incorrect	1
	ffbconfig	SHELLCODE Sparc NOOP	2
R2L	guesstelnet	INFO TELNET login incorrect	40
	imap	SHELLCODE Linux shellcode	1
		SHELLCODE X86 NOOP	1
		IMAP login buffer overflow attempt	1

Continued on next page

Table A.8 – continued from previous page

Attack Type	Name	Alert	Quantity
	dict	INFO TELNET login incorrect	86
	ncftp	(ftp_telnet) FTP command parameters were too long	1
Probe	portsweep	TCP Portscan	8
		Open port	1
	ipsweep	128.223.199.68 → 172.16.113.3	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
		204.71.51.16 → 172.16.113.5	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
		204.233.47.21 → 172.16.113.50	
		ICMP PING	2
		ICMP PING BSD Type	2
		ICMP PING *NIX	2
		207.114.237.57 → 172.16.113.4	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1
		209.1.12.46 → 172.16.113.1	
		ICMP PING	1
		ICMP PING BSD Type	1
		ICMP PING *NIX	1

Table A.9: Day 7 - 6th April 1999

Attack Type	Name	Alert	Quantity
DOS	teardrop	(spp_frag3) Short Fragment, possible DoS attempt	90
		(spp_frag3) zero-byte fragment packet	90
		(spp_frag3) Fragment packet ends after defragmented packet	88
	back	Web-misc apache directory disclosure attempt	3530
		Open port	5
	neptune	TCP Portscan	14
		Open port	52
		SNMP request TCP	81
		Scan myscan	2

Continued on next page

Table A.9 – continued from previous page

Attack Type	Name	Alert	Quantity
		SNMP agent X/TCP request	80
		MISC source port 53 to ¡ 1024	2
		DOS BGP spoofed connection reset attempt	1
	pod	ICMP PING	10
		(spp_frag3) Fragmentation overlap	685
	neptune	SNMP Request TCP	10
		SNMP Trap TCP	10
		SNMP agent X/TCP request	10
U2R	casesen	Attack response directory listing	8
	yaga	Attack response directory listing	12
R2L	xsnoop	X11 Xopen	2
	ftpwrite	FTP .rhosts	1
	ncftp	(ftp_telnet) FTP Commands parameters were too long	1

Table A.10: Day 8 - 7th April 1999

Attack Type	Name	Alert	Quantity
DOS	back	Web-misc apache directory disclosure attempt	1138
	back	Web-misc apache directory disclosure attempt	480
U2R	ffbconfig	Shellcode Sparc NOOP	2
R2L	xlock	X11 Xopen	2
	phf	Web-cgi phf arbitrary command execution attempt	2
		Web-misc cat %20 access	2
		Web-cgi phf access	2
		Web-misc /etc/passwd	2
	netbus	BACKDOOR netbus active	2
		BACKDOOR netbus getinfo	2
Probe	portsweep	TCP Portscan	4
		Open port	16

Table A.11: Day 9 - 8th April 1999

Attack Type	Name	Alert	Quantity
DOS	teardrop	(spp_frag3) Short Fragment, possible DoS attempt	20
		(spp_frag3) zero byte fragment packet	20

Continued on next page

Table A.11 – continued from previous page

Attack Type	Name	Alert	Quantity
		(spp_frag3) Fragment packet ends after defragmented packet	18
U2R	casesen	Attack response directory listing	47
	yaga	Attack response directory listing	26
	sechole	Attack response directory listing	42
R2L	phf	Web-misc /etc/passwd	2
		Web-cgi phf arbitrary command execution attempt	2
		Web-misc cat%20 access	2
Probe	portsweep	ICMP Destination unreachable port unreachable	6
	ntinfoscan	*Web-frontpage /vti_bin/access*	*4*
		Web-iis fpcount access	4
		Web-iis fpcount attempt	4
		Misc source port 20¡1024	24
		Web-iis *.idc attempt	6
		Web-cgi cgi-bin /access	6
		Web-cgi perl.exe access	12
		Web-iis newdsn.exe access	6
		Web-iis perl browse newline attempt	5
		Web-iis perl access	4
		Web-misc queryhit.htm access	4
		NETBIOS SMB ADMIN$ unicode share access	4
		NETBIOS SMB C$ unicode share access	4
		NETBIOS SMB D$ unicode share access	4
		Web-cgi perl.exe command attempt	12
		FTP Port bounce attempt	12
		FTP command parameters were malformed	12
	satan	TCP Portscan	2
		Open port	9
		ICMP PING	2
		ICMP echo reply	2
		ICMP Destination unreachable port unreachable	2682
		Finger/execution attempt	12
		Finger 0 query	2
		Finger redirection attempt	2
		Finger root query	2
		SNMP request TCP	2
		SNMP Trap TCP	2
		SNMP agentX/TCP request	1
		UDP Portscan	4

Continued on next page

Table A.11 – continued from previous page

Attack Type	Name	Alert	Quantity
		SNMP trap UDP	2
		FTP Satan scan	2
	ipsweep	ICMP PING	12
		ICMP echo reply	4
	mscan	TCP Portscan	1248
		Open port	11011
		TCP Portsweep	38
		ICMP Destination unreachable port unreachable	1908
		UDP Portsweep	67
		Web-cgi phf access	695
		Web-misc handler access	696
		SNMP agentX/TCP request	16
		RPC Portmap listing TCP 111	398
		Web-cgi test-cgi access	696

Table A.12: Day 10 - 9th April 1999

Attack Type	Name	Alert	Quantity
DOS	back	Web-misc apache directory disclosure attempt	480
	land	Bad Traffic Same Src/Dst IP	1
U2R	yaga	Attack response directory listing	16
	eject	(ftp_telnet) invalid TCP command	6
	casesen	Attack response directory listing	16
R2L	xsnoop	X11 Xopen	2
	guest	INFO Telnet Login incorrect	8
	sendmail	SHELLCODE X86 NOOP	2
Probe	Portsweep	ICMP Destination unreachable port unreachable	6

B Results of the Experiments on Snort vs SMART

Table B.1: 1999 DARPA Data Set – Reduction Rate

No	Signatures	No of False Alarms		Reduction Rate (%)
		Before	After	
1.	INFO web bug 1x1 gif attempt	22,559	17,696	78.44
2.	ICMP Destination Unreachable Port Unreachable	14,017	6,465	46.12
3.	ICMP Echo Reply	11,275	2,508	22.24
4.	ICMP Ping	5,259	2,639	50.18
5.	CHAT IRC message	1,829	456	24.93
6.	ICMP PING *NIX	883	17	1.93
7.	ICMP PING BSD Type	883	17	1.93
8.	ATTACK Responses 403 forbidden	792	109	13.76
9.	WEB-CGI redirect access	613	95	15.5
10.	(portscan) Open Port	601	230	38.27
11.	INFO Telnet login incorrect	594	80	13.47
12.	(spp_frag3) Fragmentation overlap	431	160	37.12
13.	Attack response directory listing	423	61	14.42
14.	UDP Portsweep	353	67	18.98
15.	WEB-CGI count.cgi access	297	29	9.76
16.	ICMP Redirect net	281	30	10.68
17.	ATTACK Responses invalid URL	265	41	15.47
18.	CHAT IRC Nick change	228	43	18.86
19.	CHAT IRC Channel join	219	43	19.63
20.	Web-iis fpcount access	174	10	5.75
21.	WEB-CGI calendar access	167	12	7.19
22.	WEB-MISC search.dll access	103	4	3.88
23.	WEB-MISC backup access	99	8	8.08
24.	WEB-CGI finger access	83	9	10.84
25.	TCP Portsweep	61	7	11.48
26.	Web-frontpage /vti_bin/access	38	10	26.32
27.	WEB-CGI wrap access	37	2	5.41
28.	(http_inspect) DOUBLE DECODING ATTACK	30	8	26.67

Table B.2: 1999 DARPA Data Set – Unfiltered False Alarms

No	Signatures
1	WEB-IIS fpcount attempt
2	ICMP Time-To-Live Exceeded in Transit
3	ICMP Destination Unreachable Host Unreachable
4	WEB-MISC RBS ISP /newuser access
5	(ftp_telnet) invalid FTP command
6	WEB-MISC intranet access
7	WEB-MISC counter.exe access
8	INFO FTP bad login
9	(spp_stream4) TTL LIMIT Exceeded
10	WEB-CGI download.cgi access
11	(ftp_telnet) FTP traffic encrypted
12	WEB-IIS iissamples access
13	WEB-FRONTPAGE shtml.dll access
14	x11 xopen
15	WEB-IIS iisadmin access
16	WEB-MISC redirect.exe access
17	WEB-CGI icat access
18	BACKDOOR MISC Solaris 2.5 attempt
19	WEB-CLIENT Microsoft emf metafile access
20	WEB-MISC Lotus Notes .pl script source download attempt
21	MULTIMEDIA Windows Media download
22	WEB-IIS ISAPI .idq attempt
23	WEB-CGI phf arbitrary command execution attempt
24	FTP passwd retrieval attempt
25	WEB-MISC cat %20 access
26	WEB-CGI perl.exe command attempt
27	WEB-IIS ISAPI .idq access
28	WEB-MISC login.htm access
29	WEB-CGI search.cgi access
30	RSERVICES rlogin root
31	MS-SQL version overflow attempt
32	MS-SQL ping attempt
33	WEB-FRONTPAGE shtml.exe access
34	(snort decoder) Bad Traffic Same Src/Dst IP
35	WEB-CGI db2www access

Table B.3: 1999 DARPA Data Set – Correctly Identified True Alarms

No	Signatures
1	SNMP Trap TCP
2	(ftp_telnet) FTP command parameters were too long
3	FTP Port bounce attempt
4	WEB-MISC queryhit.htm access
5	WEB-IIS perl access
6	Misc source port 20¡1024
7	FTP .rhosts
8	WEB-MISC apache disclosure attempt
9	WEB-IIS newdsn.exe access
10	WEB-IIS *.idc attempt
11	IMAP login buffer overflow attempt
12	SHELLCODE Linux shellcode
13	Backdoor netbus active
14	WEB-IIS perl browse newline attempt
15	Finger redirection attempt
16	Finger 0 query
17	Finger root query
18	DNS TCP inverse query overflow
19	FTP Satan scan
20	Finger/execution attempt
21	SNMP Agent X/TCP request
22	WEB-MISC /etc/passw
23	Scan myscan
24	SHELLCODE Sparc NOOP
25	SNMP Request TCP
26	(spp_frag3) fragment packet ends after defragmented packet
27	NETBIOS SMB D$ unicode share access
28	DOS BGP Spoofed connection reset attempt
29	SHELLCODE X86 NOOP
30	(spp_frag3) zero byte fragment packet
31	NETBIOS SMB ADMIN$ unicode share access
32	NETBIOS SMB C$ unicode share access
33	(ftp_telnet) FTP command parameters were malformed
34	(spp_frag3) Short fragment, possible DoS attempt
35	Backdoor netbus getinfo
36	WEB-CGI perl.exe access
37	WEB-CGI cgi-bin /access

Table B.4: Plymouth Data Set – Reduction Rate

No	Signatures	No of False Alarms		Reduction
		Before	After	Rate (%)
1.	(http_inspect) BARE BYTE UNICODE ENCODING	2489	1385	55.64
2.	(http_inspect) DOUBLE DECODING ATTACK	207	3	1.45
3.	(portscan) TCP Portsweep	56	1	1.79
4.	ATTACK-RESPONSES 403 Forbidden	313	21	6.71
5.	ICMP L3retriever Ping	4355	2097	48.15
6.	POLICY Google Desktop activity	1272	456	35.85
7.	SPYWARE-PUT Trackware funwebproducts mywebsearchtoolbar-funtools runtime detection	763	123	16.12
8.	WEB-IIS view source via translate header	33902	25170	74.24
9.	WEB-MISC robots.txt access	11073	10315	93.15

Table B.5: Plymouth Data Set – Unfiltered False Alarms

No	Signatures
1	(http_inspect) IIS UNICODE CODEPOINT ENCODING
2	(http_inspect) WEBROOT DIRECTORY TRAVERSAL
3	(portscan) Open Port
4	(portscan) TCP Portscan
5	(snort_decoder) WARNING: ICMP Original IP Fragmented and Offset Not 0!
6	ICMP Destination Unreachable Communication Administratively Prohibited
7	ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited
8	ICMP PING NMAP
9	ICMP redirect host
10	ICMP Source Quench
11	MULTIMEDIA Quicktime User Agent access
12	SPYWARE-PUT Trickler teomasearchbar runtime detection
13	WEB-CGI calendar access
14	WEB-FRONTPAGE /_vti_bin/ access
15	WEB-IIS asp-dot attempt
16	WEB-MISC .DS_Store access
17	WEB-MISC WebDAV search access
18	WEB-PHP calendar.php access
19	WEB-PHP remote include path
20	WEB-PHP test.php access

Continued on next page

Table B.5 – continued from previous page

No	Signatures
21	WEB-PHP xmlrpc.php post attempt

Table B.6: Plymouth Data Set – Correctly Identified True Alarms

No	Signatures
1	ICMP PING CyberKit 2.2 Windows
2	SPYWARE-PUT Adware hotbar runtime detection - hotbar user-agent
3	SPYWARE-PUT Hijacker Marketscore runtime detection
4	SPYWARE-PUT Hijacker searchmiracle – elitebar runtime detection
5	SPYWARE-PUT Trackware alexa runtime detection
6	WEB-CGI formmail access
7	WEB-MISC Domino webadmin.nsf access

C The Pseudocode

C.1 Main Alarm Aggregation Pseudocode

*dataText*1 as the data containing alarm attributes
*figureName*1 as the name of final mapping figure
*stage*1*Res* as the name of text file that will contain the classification result
minID as the smallest alert ID from the processed alerts
{Data Mapping}
READ *dataText*1
OBTAIN normalised data via var method
COMPUTE IP address attributes' weights as 1.8 times current values
GET the best size of map based on the smallest quantisation and topographic errors
{This called function will be expanded in subsection C.2.1}
CREATE the map via som_make method
CLASSIFY data on the map using K-means algorithm and STORE the result into *data_ind*
{This called function will be expanded in subsection C.2.2}
{*data_ind* is a cell array with size of (maximum number of clusters x 1)}
COMPUTE y as the length *data_ind*
{y = number of clusters resulted from the classification}
COMPUTE x as length of input data
{Output Writing(MySQL database)}
SET *No_alerts* to x
SET *No_clusters* to y
INIT *result* as a cell array size (2, 1)
{A cell array with size of 2 rows and 1 column}
{DETERMINE *num* as the highest cluster number from table *stage*1}
if *num* is not a number **then**
  SET h to 0
else
  SET h to *num*
end if
for i = 1 to *No_alerts* **do**
  for j = 1 to *No_clusters* **do**
    COMPUTE *clust* as sum of h and j
    COMPUTE *len* as the length of *data_ind* with array position (j)
    for $x = 1$ to *len* **do**
      if *minID* is equal to *data_ind* with array position (j) and element number (x) **then**

APPEND $minID$ to result with array position (1)
APPEND $clust$ to result with array position (2)
Break out of loop
end if
end for
if length of $result$ with array position (1) is not equal to 0 and the last element of $result$ with array position (1) is equal to i **then**
Break out of loop
end if
end for
Increment $minID$
end for
for $c = 1$ to No_alerts **do**
INSERT into table $Stage1$ (MySQL database) the values of $result$ array position (1), element number (c) and $result$ array position (2), element number (c).
end for
{Writing data index into a file}
OPEN a file named $stage1Res$ for writing; discard existing contents
SET pj to $No\_clusters$
for $z = 1$ to pj **do**
SET len to length of $data\_ind$ with array position (z)
if $len = 0$ **then**
WRITE "NA"
INSERT carriage return
else
WRITE the values of $data\_ind$ with array position (z) and element number 1 to ($len - 1$)
INSERT " "
WRITE the values of $data\_ind$ with array position (z) and element number len
INSERT carriage return
end if
end for

C.2 Called Functions Pseudocode

C.2.1 GET the best size of map based on the smallest quantisation and topographic errors

SET sA as data struct
OBTAIN $data$ field from sA and STORE it into D
DETERMINE the size of D and STORE into variables $dlen$ and dim
{dlen (row) is the number of alerts being processed, whilst dim ($column$) is the number of attributes per alert}
SET $munit$ to the sum of $dlen$ and 100

CREATE a map via som_make function with the number of units set to $munit$
CALCULATE its quantitative and topographic errors via som_quality method and STORE the values into an array $[mqe, tge]$
{mqe is a quantitative error, whilst tge is a topographic error}
ROUND mqe to not more than three decimal points
ROUND tge to not more than three decimal points
while $mqe > 0.1$ or $tge > 0.1$ **do**
  COMPUTE $munit$ as $munit + 10$
  CREATE a map via som_make function with the number of units set to $munit$
  CALCULATE its quantitative and topographic errors via som_quality method and STORE the values into an array $[mqe, tge]$
  ROUND mqe to not more than three decimal points
  ROUND tge to not more than three decimal points
end while
RETURN $munit$

C.2.2 CLASSIFY data on the map using K-means algorithm and STORE the result into $data\_ind$

SET sA as data struct
$sMap$ as trained map struct
$figureName1$ is a string containing the name of final mapping figure
$minID$ as the smallest alert ID from the processed alerts
n_max as maxium number of clusters
c_max as maximum number of k-means runs
$verbose$ as verbose level, 0 by devault
OBTAIN $data$ field from sA and STORE it into D
DETERMINE the size of D and STORE into variable $dlen$ and dim
{$dlen$ is the number of alerts being processed (row), whilst dim is the number of attributes per alert ($column$)}
CALCULATE cl as $dlen$ divided by 2
if number of input argumens < 5 or n_max is not defined or n_max is not a number **then**
  SET n_max to cl
end if
if number of input arguments < 6 or c_max is not defined or c_max is not a number **then**
  SET c_max to 5
end if
if number of input arguments < 7 or $verbose$ is not defined or $verbose$ is not a number **then**
  SET $verbose$ = 0
end if
SET t_max to 1
{t_max is the number of randomised trials run by K-means algorithm}
INIT e as a zero t_max-by-1 matrix

{the matrix has t_max row(s) and 1 column}
INIT data_post as a zero 1-by n_max matrix
{the matrix has 1 row and n_max column(s)}
{data_post holds the number of data per cluster}
INIT *data_ind* as a cell array size (n_max, 1)
{A cell array with size of n_max row(s) and 1 column}
{*data_ind* contains the data's ID number in each cluster}
SET *errcomp* to the largest double precision floating point number
{Choosing the best map}
for $w = 1$ to t_max **do**
  {CLUSTER $sMap$ using K-means algorithm via kmeans_clusters method and STORE the result into c, p, err and ind}
  {This called function will be explained in subsection C.2.3}
  {c (cell array) contains cluster centroids, p (cell array) contains cluster indexes, err (row vector) contains squared sum of errors, and ind (row vector) contains Davies-Bouldin index value for each clustering}
  SELECT the minimum values from err and STORE the value as well as its index number into *dummy* and i respectively
  SET $(w, 1)$ entry of matrix e to (i) entry of row vector err
  if (i) entry of $err <$ *errcomp* **then**
    SET *errcomp* to (i) entry of err
    SET $bestMap$ to p
    SET *index* to i
    SET *iter* to w
  end if
end for
SHOW the selected map using som_show method
{som_show is a default method from SOM Toolbox}
ADD label automatically to trained struct map ($sMap$) via som_autolabel method
{som_autolabel is a default method from SOM Toolbox}
SHOW the label on the map using som_show_add method
{som_show_add is a default method from SOM Toolbox}
SAVE the selected map as a figure named $figureName1$
DETERMINE the index for each unit in $sMap$ that best matched the vectors in sA using som_bmus method and STORE the indexes into *bmus*
{bmus is a column vector}
{som_bmus is a default method from SOM Toolbox}
SET ze to $bestMap$ with array position *index*
{Identifying cluster members}
for $s = 1$ to *dlen* **do**
  INCREMENT $ze(bmus(s))$ entry of matrix *data_post*
  ADD $minID$ to *data_ind* with cell position $ze(bmus(s))$
  INCREMENT $minID$

end for
RETURN *data_ind*

C.2.3 CLUSTER *sMap* using K-means algorithm via kmeans_clusters method and STORE the result into *c*, *p*, *err* and *ind*

SET *sMap* as trained map struct
SET *n_max* as maxium number of clusters
SET *c_max* as maximum number of k-means runs for each k (number of centroids)
SET w as an index number of the randomised trial run by K-means
SET *verbose* as verbose level, 0 by devault
SET OBTAIN "codebook" field from *sMap* and STORE it into D
SET DETERMINE the size of D and STORE into variable *dlen* and *dim*
{*dlen* is the number of vectors in the map (row), whilst dim is the number of attributes per vector (column)}
if number of input argumens < 2 or *n_max* is not defined or *n_max* is not a number **then**
  SET *n_max* to the square root of *dlen* and ROUND it to the nearest integer towards infinity
end if
if number of input arguments < 3 or *c_max* is not defined or *c_max* is not a number **then**
  SET *c_max* to 5
end if
if number of input arguments < 7 or *verbose* is not defined or *verbose* is not a number **then**
  SET *verbose* $= 0$
end if
INIT *centers* as a cell array size (*n_max*, 1)
{A cell array with size of *n_max* row(s) and 1 column}
{*centers* contains the cluster centroids}
INIT *clusters* as a cell array size (*n_max*, 1)
{A cell array with size of *n_max* row(s) and 1 column}
{*clusters* contains cluster indexes}
INIT *ind* as a zero 1-by-n_max row vector
{the matrix has 1 row and *n_max* column(s)}
{*ind* contains Davies-Bouldin index value for each clustering}
INIT *errors* as a zero 1-by-*n_max* row vector
{the matrix has 1 row and *n_max* column(s)}
{Classification process}
{For k (centroid) = 1 classification}
SET INIT m as a zero1-by-*dim* matrix
for $i = 1$ to *dim* **do**
  CALCULATE the average value of the i^{th} attributes from D
end for
SET the first array of *centers* to m
SET the first array of *clusters* to a one *dlen*-by-1 matrix

DETERMINE the requested best matching unit for each vector in $sMap$ and their corresponding quantisation errors using som_bmus method and STORE the indexes and errors into $dummy$ and $qerr$ respectively
{$dummy$ and $qerr$ are column vectors}
{som_bmus is a default method from SOM Toolbox}
SET the first entry of $errors$ to the sum of the square $qerr$ from each vector
{For k = 2 to $k = n_max$ classification}
for $i = 2$ to n_max **do**
  SET $best$ to the largest double precision floating point number
  for $j = 1$ to c_max **do**
    CLASSIFY the vectors on the sMap using som_kmeans function and STORE the cluster centroids, cluster indexes and the sum of squared error for the classification into c, k and err respectively
    {som_kmeans is a default method from SOM Toolbox}
    if $err < best$ **then**
      SET k_best to k
      SET c_best to c
      SET $best$ to err
    end if
  end for
end for
{Storing the results}
SET the i^{th} array of $centers$ to c_best
SET the i^{th} array of $clusters$ to k_best
SET the i^{th} entry of $errors$ to $best$
CALCULATE the Davies-Bouldin index using db_index method and STORE the value into the i^{th} entry of ind

C.3 Main False Alarm Classification Pseudocode

$dataText2$ as the data containing alarm attributes
$figureName2$ as the name of final mapping figure
$finalIndex$ as the name of text file that will contain final indexes of true and false alarms
{Data Mapping}
READ $dataText2$
COMPUTE len as the length of input data
if all values of the 4^{th} attribute (column) of the input data is not a number **then**
  SET all values of the 4^{th} column to -1
end if
OBTAIN normalised data via var method
COMPUTE time intervals attribute's weight as 2.5 times current values
COMPUTE number of events attribute's weight as 2.8 times current values

GET the best size of map based on the smallest quantisation and topographic errors
{This called function has been expanded in subsection C.2.1}
CREATE the map via som_make method
CLASSIFY data on the map using K-means algorithm and STORE the result into $data\_indFinal$ and rec_post
{This function will be further expanded in subsection C.4.1}
{$data\_ind$ is a cell array with size of (maximum number of clusters x 1)}
{rec_post is a cell array with size of (2,1)}
{Writing output into the database}
SET $a1$ to the value of the first element and the first cell of cell array $data\_indFinal$
SET $a2$ to the value of the first element and the second cell of cell array $data\_indFinal$
for $t = 1$ to len **do**
  if $a1$ is equal to the t element of the first cell of cell array rec_post **then**
    SET $cmp1$ to the t element of the second cell of cell array rec_post
    Break out of the loop
  end if
end for
for $t = 1$ to len **do**
  if $a2$ is equal to the t element of the first cell of cell array rec_post **then**
    SET $cmp2$ to the t element of the second cell of cell array rec_post
    Break out of the loop
  end if
end for
if the 7^{th} attribute of the $cmp1$ entry of the input data > the 7^{th} attribute of the $cmp2$ entry of the input data **then**
  SET x to 0
  SET y to 1
else if the 7^{th} attribute of the $cmp1$ entry of the input data = the 7^{th} attribute of the $cmp2$ entry of the input data **then**
  if the 1^{st} attribute of the $cmp1$ entry of the data > the 1^{st} attribute of the $cmp2$ entry of the data **then**
    SET x to 0
    SET y to 1
  else if the 1^{st} attribute of the $cmp1$ entry of the data ¡ the 1^{st} attribute of the $cmp2$ entry of the data **then**
    SET x to 1
    SET y to 0
  else
    CREATE array $array$ and ASSIGN the values 2, 5 and 6 to the array respectively
    for $d = 1$ to length of array **do**
      if the array(d) attribute of the $cmp1$ entry of the data < the array(d) attribute of the $cmp2$ entry of the data **then**
        SET x to 0

```
        SET y to 1
        Break out of the loop
      else if the array (d) attribute of the cmp1 entry of the data > the array(d) attribute of the
      cmp2 entry of the data then
        SET x to 1
        SET y to 0
        Break out of the loop
      end if
    end for
  end if
else
  SET x to 1
  SET y to 0
end if
INIT result as a cell array size (2, 1)
{A cell array with size of 2 rows and 1 column}
DETERMINE num as the highest cluster number from table stage1
if num is not a number then
  SET h to 0
else
  SET h to num
end if
SET p1 to the length of the first cell of cell array data_indFinal
for i = 1 to len do
  CALCULATE clust as the sum of h and i
  for d = 1 to p1 do
    if clust is equal to the d element of the first cell of cell array data_indFinal then
      APPREHEND clust to the first cell of cell array result
      APPREHEND x to the second cell of cell array result
      Break out of the loop
    end if
  end for
  if the length of the first cell of cell array result is equal to 0 or the last element of the first cell
  of cell array result is not equal to clust then
    APPREHEND clust to the first cell of cell array result
    APPREHEND y to the second cell of cell array result
  end if
end for
for c = 1 to No_alerts do
  INSERT into table Stage2 the values of the c element, first cell of result and the c element, the
  second cell of result
end for
{Writing data index to a file}
```

OPEN a file named $finalIndex$ for writing; discard existing contents
SET pj to $No\_clusters$
for $z = 1$ to 2 **do**
  SET len to length of the z cell of $data\_indFinal$
  if $len = 0$ **then**
    WRITE "NA"
  else
    WRITE the values of $data\_indFinal$ with cell position z and element number 1 to $(len - 1)$
    INSERT " "
    WRITE the values of $data\_ind$ with cell position z and element number len
    INSERT carriage return
  end if
end for

C.4 Called Functions Pseudocode

C.4.1 Classify data on the map using K-means algorithm and STORE the result into $data\_indFinal$ and rec_post

sA as data struct
$sMap$ as trained map struct
$figureName2$ is a string containing the name of final mapping figure
n_max as maxium number of clusters
c_max as maximum number of k-means runs
$verbose$ as verbose level, 0 by devault
OBTAIN codebook field from $sMap$ and STORE it into D
DETERMINE the size of D and STORE into variable $dlen$ and dim
{$dlen$ is the number of alerts being processed (row), whilst dim is the number of attributes per alert (column)}
CALCULATE cl as $dlen$ divided by 2
if number of input argumens < 4 or n_max is not defined or n_max is not a number **then**
  SET n_max to 2
end if
if number of input arguments < 5 or c_max is not defined or c_max is not a number **then**
  SET c_max to 5
end if
if number of input arguments < 6 or $verbose$ is not defined or verbose is not a number **then**
  SET $verbose = 0$
end if
SET t_max to 500
{t_max is the number of randomised trials run by K-means algorithm}
SET k to 0
{index for the possible cluster solutions (maps)}

INIT sse as an empty array
{sum of squared error for each unique map}
INTI $freq$ as an empty array
{number of occurrence for each unique map}
INIT $clust\_post$ as a cell aray size $(t\_max, 1)$
{A cell array with size of t_max row(s) and 1 column}
INIT $data\_post$ as a zero 1-by n_max matrix
{the matrix has 1 row and n_max column(s)}
{$data\_post$ holds the number of data per cluster}
INIT $data\_ind$ as a cell array size $(n\_max, 1)$
{A cell array with size of n_max row(s) and 1 column}
{$data\_ind$ contains the index number of the first stage clusters in each second stage cluster}
INIT rec_post as a cell array size (2, 1)
{A cell array with size of 2 rows and 1 column}
SET $totSSE$ to 0
SET sol to 0
INIT g as an empty array
INIT no_clust as an empty array
SET $close$ to the largest double precision floating point number
{Choosing the best map}
{Sorting the maps}
for $w = 1$ to t_max **do**
  CLUSTER sMap using K-means algorithm via kmeans_clusters method and STORE the result into c, p, err and ind
  {This called function has been explained in subsection C.2.3}
  {c (cell array) contains cluster centroids, p (cell array) contains cluster indexes, err (row vector) contains squared sum of errors, and ind (row vector) contains Davies-Bouldin index value for each clustering}
  SELECT the minimum values from err and STORE the value as well as its index number into $dummy$ and i respectively
  if (i) entry of err is a member of sse array **then**
    STORE the index number of (i) entry of err in sse into $isavai$
    INCREMENT $(isavai)$ entry of $freq$
  else
    INCREMENT k
    {Counting unique map}
    APPREHEND (i) entry of err to sse array
    APPREHEND 1 to $freq$ array
    STORE cluster indexes of the unique map into (k) cell of cell array $clust\_post$
    APPREHEND the number of clusters of the unique map into array no_clust
  end if
end for
{Compute the frequency rate}

SET $rate$ to zero 1-by-k matrix
for $t = 1$ to k **do**
  COMPUTE the (t) entry of matrix rate as (t) entry of $freq$ divided by t_max
  {calculating the frequency rate}
end for
SELECT the highest frequency rate and STORE the value and its index number into $high$ and $index$ respectively
{Compute the second thresholding value (standard deviation)}
COMPUTE the standard deviation of the maps' frequency rates and STORE the result into variable st
{Apply the thresholding}
if $high > 0.6$ **then**
  SET $bestMap$ to $index$
else
  for $m = 1$ to k **do**
    if $high <= st$ **then**
      SET sol to k
      CALCULATE $totSSE$ as the sum of current $totSSE$ value and the total values of sse array

      STORE the unique map indexes into array g
      Break out of the loop
    end if
    if (m) entry of array $rate >= high$ subtracted by st **then**
      CALCULATE $totSSE$ as the sum of current $totSSE$ and m entry of array sse
      INCREMENT sol
      APPREHEND m to array g
    end if
  end for
  {Calculate the average SSE}
  COMPUTE $aveSSE$ as $totSSE$ divided by sol
  if $sol = 2$ **then**
    if the sse value of the first entry of array $g <$ the sse value of the second entry of array g **then**
      SET $bestMap$ to the first entry of array g
    else
      SET $bestMap$ to the second entry of array g
    end if
  else
    for $n = 1$ to sol **do**
      SET $diff$ to the difference between the sse value of the n entry of array g and the $aveSSE$
      if $close > diff$ **then**
        SET $close$ to $diff$
        SET $bestMap$ to the n entry of array g

end if
end for
end if
end if
SHOW the selected map using som_show method
{som show is a default method from SOM Toolbox}
ADD label automatically to trained struct map ($sMap$) via som_autolabel method
{som_autolabel is a default method from SOM Toolbox}
SHOW the label on the map using som_show_add method
{som_show_add is a default method from SOM Toolbox}
SAVE the selected map as a figure named $figureName1$
{Identifying cluster members}
DETERMINE the index for each unit in $sMap$ that best matched the vectors in sA using som_bmus method and STORE the indexes into $bmus$
{$bmus$ is a column vector}
{som_bmus is a default method from SOM Toolbox}
SET ze to $bestMap$ with array position $index$
COMPUTE the size of input data and STORE the values into variables l and $attr$
{l (row) is the number input clusters from $stage1$, whilst $attr$ (column) is the number of features taken per cluster}
DETERMINE the highest cluster number from table $stage2$ and STORE the value into num
if num is not a number **then**
SET h to 0
else
SET h **to** num
end if
for $s = 1$ **to** l **do**
SET $label$ to the sum of h and s
APPREHEND $label$ to the first cell of cell array rec_post
APPREHEND s to the second cell of cell array rec_post
INCREMENT $ze(bmus(s))$ entry of matrix $data\_post$
APPREHEND $label$ to $ze(bmus(s))$ entry of $data\_ind$
end for

D MATLAB Source Code

D.1 Counting Time Interval and Number of Events

Listing D.1: source–code/countEventTimeTwo.m

```
1  function Timetb = countEventTimeTwo(A, B, dayAlerts)
2
3  %% Calculating the number of events and the time interval
4  add = [0 0 0 2 0 0];
5  curTime = A;
6  while ~strcmp(curTime,B),
7      curVec = datevec(curTime);
8      NextVec =  curVec+ add;
9      NextTime = datestr(NextVec, 31);
10     str = sprintf('select distinct signature from %s where timestamp >= "%s" and timestamp <
           "%s"', dayAlerts, curTime, NextTime);
11     sig = mysql(str);
12     pj = size(sig,1);
13     Timetb = zeros(pj,3);
14      for c = 1:4,
15         if curVec(c) < 10,
16             sx(c) = sprintf('0%d',curVec(c));
17         else
18             sx(c) = sprintf('%d',curVec(c));
19         end
20     end
21     str = sprintf('create table TimeEventTwo%s%s%s%s (Signature char(10), No_of_event int(10),
            Time_interval float)',sx(1),sx(2),sx(3),sx(4));
22     tabl = mysql(str);
23     for co = 1:pj,
24         Timetb(co,1) = sig(co);
25     end
26
27     for x = 1:pj,
28         str = sprintf('select ip_src, ip_dst, timestamp from %s where signature = %d and
               timestamp >= "%s" and timestamp <= "%s" order by timestamp', dayAlerts, sig(x),
               curTime, NextTime);
29         [src, dst, ti] = mysql(str);
30         format long G, ti;
31         format long G, src;
32         format long G, dst;
33         noAlert = size(src,1);
34         for v = 1:noAlert,
35             vec = datevec(ti(v));
36             vec(6) = 0;
37             time(v) = vec;
```

```
38          end
39          for col = 1:2,
40              for row = 1:noAlert,
41                  if col == 1,
42                      data(row,col) = src(row);
43                  else
44                      data(row,col) = dst(row);
45                  end
46              end
47          end
48          t = 0; % time interval
49          i = 1; % index for T array
50          T = zeros(1,noAlert);
51          event = 0; % no of event
52          index = 0; % index of the counted alarm
53          for n = 1:noAlert,
54           % check if the alarm has been covered by previous alarm
55           if find(T==n),
56              event = event;
57           else
58              event = event + 1;
59              if event == 1,
60                  t = 0;
61              else
62                  t = t + abs(etime(time{n},time{index}));
63              end
64              index = n;
65              T(1,i) = index;
66              i = i + 1;
67              j = n+1;
68              % Finding the the alarms covered by the current alert
69              if n ~= noAlert,
70              while abs(etime(time{j},time{n})) <= 180, % if the elapsed time is less than equal
                     to 3 minutes
71                  if data(j,2) == data(n,2) || data(j,1) == data(n,1), % if either the dst IP
                        addresses or src IP addresses are match
72                      T(1,i) = j;
73                      i = i + 1;
74                  end
75                  j = j + 1;
76                  if j > noAlert,
77                      break;
78                  end
79              end
80              end
81           end
82          end
83          Timetb(x,2) = event;
84          if t/event == 0,
85              Timetb(x,3) = 7200; % No of seconds in two hours
86          else
87              format short G, Timetb(x,3) = t/(event-1);
88          end
89          str = sprintf('insert into TimeEventTwo%s%s%s%s (Signature, No_of_event, Time_interval
```

```
            ) values (%d, %d, %.3f)', sx{1},sx{2},sx{3},sx{4},Timetb(x,1), Timetb(x,2), Timetb
            (x,3));
90          tabl = mysql(str);
91      end
92      format short G, Timetb(:,1);
93      format short G, Timetb(:,2);
94      Timetb(:,3);
95      curTime = NextTime;
96  end
```

Listing D.2: source–code/countEventTimeOne.m

```
1  function Timetb = countEventTimeOne(A, B, dayAlerts)
2
3  %% Calculating the number of events and the time interval
4  add = [0 0 0 1 0 0];
5  curTime = A;
6  while ~strcmp(curTime,B),
7      curVec = datevec(curTime);
8      NextVec =  curVec+ add;
9      NextTime = datestr(NextVec, 31);
10     str = sprintf('select distinct signature from %s where timestamp >= "%s" and timestamp <
           "%s"', dayAlerts, curTime, NextTime);
11     sig = mysql(str);
12     pj = size(sig,1);
13     Timetb = zeros(pj,3);
14     for c = 1:4,
15         if curVec(c) < 10,
16             sx{c} = sprintf('0%d',curVec(c));
17         else
18             sx{c} = sprintf('%d',curVec(c));
19         end
20     end
21     str = sprintf('create table TimeEventOne%s%s%s%s (Signature char(10), No_of_event int(10),
            Time_interval float)',sx{1},sx{2},sx{3},sx{4});
22     tabl = mysql(str);
23     for co = 1:pj,
24         Timetb(co,1) = sig(co);
25     end
26
27     for x = 1:pj,
28         str = sprintf('select ip_src, ip_dst, timestamp from %s where signature = %d and
               timestamp >= "%s" and timestamp <= "%s" order by timestamp', dayAlerts, sig(x),
               curTime, NextTime);
29         [src, dst, ti] = mysql(str);
30         format long G, ti;
31         format long G, src;
32         format long G, dst;
33         noAlert = size(src,1);
34         for v = 1:noAlert,
35             vec = datevec(ti(v));
36             vec(6) = 0;
37             time{v} = vec;
38         end
```

```
39        for col = 1:2,
40            for row = 1:noAlert,
41                if col == 1,
42                    data(row,col) = src(row);
43                else
44                    data(row,col) = dst(row);
45                end
46            end
47        end
48        t = 0; % time interval
49        i = 1; % index for T array
50        T = zeros(1,noAlert);
51        event = 0; % no of event
52        index = 0; % index of the counted alarm
53        for n = 1:noAlert,
54         % check if the alarm has been covered by previous alarm
55         if find(T==n),
56            event = event;
57         else
58            event = event + 1;
59            if event == 1,
60                t = 0;
61            else
62                t = t + (data(n,3)- data(index,3)); % calculating the sum of time interval
63                t = t + abs(etime(time(n),time(index)));
64            end
65            index = n;
66            T(1,i) = index;
67            i = i + 1;
68
69            j = n+1;
70            % Finding the the alarms covered by the current alert
71            if n ~= noAlert,
72            %while data(j,3) <= (data(n,3) + 0.0000000208333334), % check the alarms which are
                  under the time frame of 3 minutes
73            while abs(etime(time(j),time(n))) <= 180, % if the elapsed time is less than equal
                   to 3 minutes
74                if data(j,2) == data(n,2) || data(j,1) == data(n,1), % if either the dst IP
                      addresses or src IP addresses are match
75                    T(1,i) = j;
76                    i = i + 1;
77                end
78                j = j + 1;
79                if j > noAlert,
80                    break;
81                end
82            end
83            end
84         end
85        end
86        Timetb(x,2) = event;
87        if t/event == 0,
88            Timetb(x,3) = 3600; % No of seconds in one hour
89        else
```

```
90              format short G, Timetb(x,3) = t/(event-1);
91          end
92           str = sprintf('insert into TimeEventOne%s%s%s%s (Signature, No_of_event,
                 Time_interval) values (%d, %d, %.3f)', sx{1},sx{2},sx{3},sx{4},Timetb(x,1),
                 Timetb(x,2), Timetb(x,3));
93          tabl = mysql(str);
94      end
95      format short G, Timetb(:,1);
96      format short G, Timetb(:,2);
97      Timetb(:,3);
98      curTime = NextTime;
99  end
```

Listing D.3: source–code/countEventTimeHalf.m

```
1  function Timetb = countEventTimeHalf(A, B, dayAlerts)
2
3  %% Calculating the number of events and the time interval
4  add = [0 0 0 0 30 0];
5  curTime = A;
6  while ~strcmp(curTime,B),
7      curVec = datevec(curTime);
8      NextVec =  curVec+ add;
9      NextTime = datestr(NextVec, 31);
10     str = sprintf('select distinct signature from %s where timestamp >= "%s" and timestamp <
           "%s"', dayAlerts, curTime, NextTime);
11     sig = mysql(str);
12     pj = size(sig,1);
13     Timetb = zeros(pj,3);
14     %st = sprintf('%d%d%d', curVec(1),curVec(2),curVec(3), curVec(4), curVec(5));
15     for c = 1:5,
16         if curVec(c) < 10,
17             sx{c} = sprintf('0%d',curVec(c));
18         else
19             sx{c} = sprintf('%d',curVec(c));
20         end
21     end
22     str = sprintf('create table TimeEventHalf%s%s%s%s%s (Signature char(10), No_of_event int
           (10), Time_interval float)',sx{1},sx{2},sx{3},sx{4},sx{5});
23     tabl = mysql(str);
24     for co = 1:pj,
25         Timetb(co,1) = sig(co);
26     end
27
28     for x = 1:pj,
29         str = sprintf('select ip_src, ip_dst, timestamp from %s where signature = %d and
               timestamp >= "%s" and timestamp <= "%s" order by timestamp', dayAlerts, sig(x),
               curTime, NextTime);
30         [src, dst, ti] = mysql(str);
31         format long G, ti;
32         format long G, src;
33         format long G, dst;
34         noAlert = size(src,1);
35         for v = 1:noAlert,
```

```
36          vec = datevec(ti(v));
37          vec(6) = 0;
38          time(v) = vec;
39      end
40      for col = 1:2,
41          for row = 1:noAlert,
42              if col == 1,
43                  data(row,col) = src(row);
44              else
45                  data(row,col) = dst(row);
46              end
47          end
48      end
49      t = 0; % time interval
50      i = 1; % index for T array
51      T = zeros(1,noAlert);
52      event = 0; % no of event
53      index = 0; % index of the counted alarm
54      for n = 1:noAlert,
55       % check if the alarm has been covered by previous alarm
56       if find(T==n),
57          event = event;
58       else
59          event = event + 1;
60          if event == 1,
61              t = 0;
62          else
63              %t = t + (data(n,3)- data(index,3)); % calculating the sum of time interval
64              t = t + abs(etime(time(n),time(index)));
65          end
66          index = n;
67          T(1,i) = index;
68          i = i + 1;
69
70          j = n+1;
71          % Finding the the alarms covered by the current alert
72          if n ~= noAlert,
73          %while data(j,3) <= (data(n,3) + 0.000000020833334), % check the alarms which are
                under the time frame of 3 minutes
74          while abs(etime(time(j),time(n))) <= 180, % if the elapsed time is less than equal
                 to 3 minutes
75              if data(j,2) == data(n,2) || data(j,1) == data(n,1), % if either the dst IP
                    addresses or src IP addresses are match
76                  T(1,i) = j;
77                  i = i + 1;
78              end
79              j = j + 1;
80              if j > noAlert,
81                  break;
82              end
83          end
84          end
85       end
86      end
```

```
87          Timetb(x,2) = event;
88          if t/event == 0,
89              Timetb(x,3) = 1800; % No of seconds in half an hour
90          else
91              format short G, Timetb(x,3) = t/(event-1);
92          end
93            str = sprintf('insert into TimeEventHalf%s%s%s%s%s (Signature, No_of_event,
                Time_interval) values (%d, %d, %.3f)', sx{1},sx{2},sx{3},sx{4},sx{5},Timetb(x,1)
                , Timetb(x,2), Timetb(x,3));
94          tabl = mysql(str);
95      end
96      format short G, Timetb(:,1);
97      format short G, Timetb(:,2);
98      Timetb(:,3);
99      curTime = NextTime;
100 end
```

D.2 Main Correlation Functions

Listing D.4: source–code/CompleteAlarmCorrelationTwo.m

```
1  function data_indFinal = CompleteAlarmCorrelationTwo(A,B)
2  %% input arguments and initialization data
3  add = [0 0 0 2 0 0];
4  curTime = A;
5  dateNumA = datenum(curTime);
6  dateNumB = datenum(B);
7  mysql('open');
8  mysql('use','darpa2');
9  %mysql('alter table acid_event add id int(10) auto_increment primary key first');
10 mysql('create table Stage1Two (id int(10), Cluster_No int(10))');
11 mysql('create table Stage2Two (Cluster_No int(10), Alert_Status tinyint(1))');
12 mysql('create table timeEventRecordTwo (Time_interval float, No_of_event int(10), Cluster_No
       int(10))');
13 countEventTimeTwo(A,B,'acid_event');
14
15 %% Normalising date and time
16
17 while dateNumA < dateNumB,
18     mysql('open');
19     mysql('use','darpa2');
20     curVec = datevec(curTime);
21     len = length(curVec);
22     for c = 1:len,
23         if curVec(c) < 10,
24             sx{c} = sprintf('0%d',curVec(c));
25         else
26             sx{c} = sprintf('%d',curVec(c));
27         end
28     end
29     st = sprintf('tb%s%s%s%s%s%s', sx{1},sx{2},sx{3},sx{4},sx{5},sx{6});
30     dataText1 = sprintf('stg1Two%s.txt', st);
```

```
31      figureName1 = sprintf('stg1Two%s', st);
32      dataText2 = sprintf('stg2Two%s.txt', st);
33      figureName2 = sprintf('stg2Two%s', st);
34      finalIndex = sprintf('FinalTwo%s.txt', st);
35      stage1Res = sprintf('stage1ResTwo%s.txt', st);
36      NextVec =  curVec+ add;
37      NextTime = datestr(NextVec, 31);
38      dateNumNext = datenum(NextTime);
39      if dateNumNext > dateNumB,
40          NextTime = B;
41      end
42      str = sprintf('select count(*) from acid_event where timestamp >= "%s" and timestamp < "%s
            "', curTime, NextTime);
43      count = mysql(str);
44      if count ~= 0,
45          str = sprintf('select min(id) as id from acid_event where timestamp >= "%s" and
                timestamp < "%s"', curTime, NextTime);
46          minID = mysql(str);
47          generateDataStg1(curTime, NextTime ,dataText1);
48          data_ind = alarmAggregateTwo(dataText1, figureName1, stage1Res, minID);
49          generateDataStgTwo(data_ind, dataText2, curVec);
50          data_indFinal = alarmFilterTwo(dataText2, figureName2, finalIndex);
51      end
52      curTime = NextTime;
53      dateNumA = datenum(curTime);
54      mysql('close');
55  end
56
57  return;
```

Listing D.5: source–code/CompleteAlarmCorrelationOne.m

```
1  function data_indFinal = CompleteAlarmCorrelationOne(A,B)
2  %% input arguments and initialization data
3  add = [0 0 0 1 0 0];
4  curTime = A;
5  dateNumA = datenum(curTime);
6  dateNumB = datenum(B);
7  mysql('open');
8  mysql('use','darpa2');
9  %mysql('alter table acid_event add id int(10) auto_increment primary key first');
10 mysql('create table Stage1One (id int(10), Cluster_No int(10))');
11 mysql('create table Stage2One (Cluster_No int(10), Alert_Status tinyint(1))');
12 mysql('create table timeEventRecordOne (Time_interval float, No_of_event int(10), Cluster_No
       int(10))');
13 countEventTimeOne(A,B,'acid_event');
14
15 %% Normalising date and time
16
17 while dateNumA < dateNumB,
18     mysql('open');
19     mysql('use','darpa2');
20     curVec = datevec(curTime);
21     len = length(curVec);
```

```
22      for c = 1:len,
23          if curVec(c) < 10,
24              sx(c) = sprintf('0%d',curVec(c));
25          else
26              sx(c) = sprintf('%d',curVec(c));
27          end
28      end
29      st = sprintf('tb%s%s%s%s%s%s', sx(1),sx(2),sx(3),sx(4),sx(5),sx(6));
30      dataText1 = sprintf('stg1One%s.txt', st);
31      figureName1 = sprintf('stg1One%s', st);
32      dataText2 = sprintf('stg2One%s.txt', st);
33      figureName2 = sprintf('stg2One%s', st);
34      finalIndex = sprintf('FinalOne%s.txt', st);
35      stage1Res = sprintf('stage1ResOne%s.txt', st);
36      NextVec =  curVec+ add;
37      NextTime = datestr(NextVec, 31);
38      dateNumNext = datenum(NextTime);
39      if dateNumNext > dateNumB,
40          NextTime = B;
41      end
42      str = sprintf('select count(*) from acid_event where timestamp >= "%s" and timestamp < "%s
            "', curTime, NextTime);
43      count = mysql(str);
44      if count ~= 0,
45          str = sprintf('select min(id) as id from acid_event where timestamp >= "%s" and
                timestamp < "%s"', curTime, NextTime);
46          minID = mysql(str);
47          generateDataStg1(curTime, NextTime ,dataText1);
48          data_ind = alarmAggregateOne(dataText1, figureName1, stage1Res, minID);
49          generateDataStgOne(data_ind, dataText2, curVec);
50          data_indFinal = alarmFilterOne(dataText2, figureName2, finalIndex);
51      end
52      curTime = NextTime;
53      dateNumA = datenum(curTime);
54      mysql('close');
55  end
56
57  return;
```

Listing D.6: source–code/CompleteAlarmCorrelationHalf.m

```
1  function data_indFinal = CompleteAlarmCorrelationHalf(A,B)
2  %% input arguments and initialization data
3  add = [0 0 0 0 30 0];
4  curTime = A;
5  dateNumA = datenum(curTime);
6  dateNumB = datenum(B);
7  mysql('open');
8  mysql('use','darpa2');
9  %mysql('alter table acid_event add id int(10) auto_increment primary key first');
10 mysql('create table Stage1Half (id int(10), Cluster_No int(10))');
11 mysql('create table Stage2Half (Cluster_No int(10), Alert_Status tinyint(1))');
12 mysql('create table timeEventRecordHalf (Time_interval float, No_of_event int(10), Cluster_No
       int(10))');
```

```
13 countEventTimeHalf(A,B,'acid_event');
14
15 %% Normalising date and time
16
17 while dateNumA < dateNumB,
18     mysql('open');
19     mysql('use','darpa2');
20     curVec = datevec(curTime);
21     len = length(curVec);
22     for c = 1:len,
23         if curVec(c) < 10,
24             sx(c) = sprintf('0%d',curVec(c));
25         else
26             sx(c) = sprintf('%d',curVec(c));
27         end
28     end
29     st = sprintf('tb%s%s%s%s%s%s', sx(1),sx(2),sx(3),sx(4),sx(5),sx(6));
30     dataText1 = sprintf('stg1Half%s.txt', st);
31     figureName1 = sprintf('stg1Half%s', st);
32     dataText2 = sprintf('stg2Half%s.txt', st);
33     figureName2 = sprintf('stg2Half%s', st);
34     finalIndex = sprintf('FinalHalf%s.txt', st);
35     stage1Res = sprintf('stage1ResHalf%s.txt', st);
36     NextVec =  curVec + add;
37     NextTime = datestr(NextVec, 31);
38     dateNumNext = datenum(NextTime);
39     if dateNumNext > dateNumB,
40         NextTime = B;
41     end
42     str = sprintf('select count(*) from acid_event where timestamp >= "%s" and timestamp < "%s
           "', curTime, NextTime);
43     count = mysql(str);
44     if count ~= 0,
45         str = sprintf('select min(id) as id from acid_event where timestamp >= "%s" and
               timestamp < "%s"', curTime, NextTime);
46         minID = mysql(str);
47         generateDataStg1(curTime, NextTime ,dataText1);
48         data_ind = alarmAggregateHalf(dataText1, figureName1, stage1Res, minID);
49         generateDataStgHalf(data_ind, dataText2, curVec);
50         data_indFinal = alarmFilterHalf(dataText2, figureName2, finalIndex);
51     end
52     curTime = NextTime;
53     dateNumA = datenum(curTime);
54     mysql('close');
55 end
56
57 return;
```

D.3 Generating Input Data for Stage 1 Correlation

Listing D.7: source–code/generateDataStg1.m

```
function [a b c] = generateDataStg1(curTime, NextTime, dataText1)

%% Calculating time stamp using 'datenum'

str = sprintf('select ip_src, ip_dst, timestamp, id from acid_event where timestamp >="%s" and
      timestamp < "%s"', curTime, NextTime);
[y, x, z, d] = mysql(str);

len = length(y);
a = zeros(len,1);
b = zeros(len,1);
c = zeros(len,1);
g1 = zeros(len,1);
g2 = zeros(len,1);
stg1data = zeros(len,4);

format long, y;
format long, x;

a = abs(x + y);
b = abs(x - y);
for v = 1:len,
    vec = datevec(z(v));
    vec(6) = 0;
    j(v) = vec;
end

for i = 1:len,
    c(i) = datenum(j(i));
end

format long, c;


for col = 1:4,
    for row = 1:len,
        if col == 1,
            stg1data(row,col) = a(row);
        elseif col == 2,
                stg1data(row,col) = b(row);
        elseif col ==3,
            stg1data(row,col) = c(row);
        else
            stg1data(row,col) = d(row);
        end
    end
end

%% Writing the formatted data to a file
fid = fopen(dataText1, 'wt');
[rows cols] = size(stg1data);
for k = 1:rows,
    fprintf(fid,'%f ', stg1data(k,end-3));
    fprintf(fid,'%d ', stg1data(k,end-2));
```

```
54      fprintf(fid,'%.10f ', stgldata(k,end-1));
55      fprintf(fid,'\''%d\''\n', stgldata(k,end));
56  end
57  fclose(fid);
58  %dlmwrite('stgldata.txt', stgldata, 'precision', 15, 'newline', 'pc', 'delimiter', ' ');
59
60  return;
```

D.4 Generating Input Data for Stage 2 Correlation

Listing D.8: source–code/generateDataStgTwo.m

```
1  function v = generateDataStgTwo(data_ind, dataText2, curVec)
2  %% input arguments and initialization data
3  num = mysql('select max(Cluster_No) as Cluster_No from Stage2Two');
4  d = 0;
5  if isnan(num),
6      pt = 0;
7  else
8      pt = num;
9  end
10
11 %% Extracting data
12 len = size(data_ind,1);
13 for count = 1:len,
14     noPort = 0;
15     mat = {[]}; port = []; src = []; dst = []; t = []; r = []; y = []; prio = []; time = [];
           event = [];
16     if length(data_ind{count}) ~= 0,
17         pt = pt + 1;
18         d = d+1;
19         no_alerts = size(data_ind{count},2);
20         for i = 1:no_alerts,
21             str = sprintf('select signature, ip_proto, layer4_sport, layer4_dport,
                   sig_priority from acid_event where id = %d', data_ind{count}(i));
22             [mat{i,1}, mat{i,2}, mat{i,3}, mat{i,4}, mat{i,5}] = mysql(str);
23         end
24         %% Filtering port numbers
25         for h = 1:no_alerts,
26             if isnan(mat{h,3}) || isnan(mat{h,4}),
27                 port(h) = -1;
28                 noPort = noPort + 1;
29             else
30                 str = sprintf('select distinct portNo from port where portNo = %d', mat{h,3});
31                 src = mysql(str);
32                 str = sprintf('select distinct portNo from port where portNo = %d', mat{h,4});
33                 dst = mysql(str);
34                 if length(src) == 0  && length(dst) ~= 0,
35                     port(h) = str2num(cell2mat(dst));
36                 elseif length(src) ~= 0 && length(dst) == 0,
37                     port(h) = str2num(cell2mat(src));
38                 elseif length(src) == 0 && length(dst) == 0,
```

```
39                  port(h) = min(mat(h,3),mat(h,4));
40              else
41                  port(h) = min(str2num(cell2mat(src)),str2num(cell2mat(dst)));
42              end
43          end
44      end
45      %% Identify no of alerts
46      stg2data(d,1) = no_alerts;
47
48      %% Identify no of signatures
49      c = 1;
50      t = mat(1,1);
51      for s = 2:no_alerts,
52          for ox = 1:s-1,
53              if (mat(ox,1) ~= mat(s,1)),
54                  c = c;
55                  break;
56              end
57              if (ox == s-1),
58                  c = c + 1;
59                  t = [t, mat(s,1)];
60              end
61          end
62      end
63      stg2data(d,2) = c;
64      %% Identify the protocol
65      for k = 1:no_alerts,
66          r = [r, mat(k,2)];
67      end
68      if find(r < 255),
69          if find(r >= 255),
70              stg2data(d,3) = 2;
71          else
72              stg2data(d,3) = 1;
73          end
74      else
75          stg2data(d,3) = 3;
76      end
77      %% Identify port number
78      if noPort == no_alerts,
79          stg2data(d,4) = -1;
80      elseif find(port < 1024 & port >= 0),
81          if find(port >= 1024),
82              stg2data(d,4) = 2;
83          else
84              stg2data(d,4) = 1;
85          end
86      else
87          stg2data(d,4) = 3;
88      end
89      %% Identify priority
90      for p = 1:no_alerts,
91          if mat(p,5) == 1,
92              prio = [prio, 300];
```

```
93              elseif mat(p,5) == 2,
94                  prio = [prio, 200];
95              else
96                  prio = [prio, 100];
97              end
98          end
99          stg2data(d,5) = sum(prio);
100         %% Calculate time interval and no of events
101         for ct = 1:4,
102             if curVec(ct) < 10,
103                 sx(ct) = sprintf('0%d',curVec(ct));
104             else
105                 sx(ct) = sprintf('%d',curVec(ct));
106             end
107         end
108         for p = 1:stg2data(d,2),
109             str = sprintf('select No_of_event, Time_interval from TimeEventTwo%s%s%s%s where
                    Signature = %d', sx(1),sx(2),sx(3),sx(4),t(1,p));
110             [ev, ti] = mysql(str);
111             if isempty(ev),
112                 NextcurVec = curVec + [0 0 0 2 0 0];
113                 for ct = 1:4,
114                     if curVec(ct) < 10,
115                         sn(ct) = sprintf('0%d',curVec(ct));
116                     else
117                         sn(ct) = sprintf('%d',curVec(ct));
118                     end
119                 end
120                 str = sprintf('select No_of_event, Time_interval from TimeEventTwo%s%s%s%s
                        where Signature = %d', sn(1),sn(2),sn(3),sn(4), t(1,p));
121                 [ev, ti] = mysql(str);
122             end
123             event(p) = ev;
124             time(p) = ti;
125         end
126         format long G, event;
127         stg2data(d,6) = max(time);
128         stg2data(d,7) = min(event);
129         stg2data(d,8) = pt;
130         str = sprintf('insert into timeEventRecordTwo(Time_interval, No_of_event, Cluster_No)
                values (%.3f, %d, %d)', stg2data(d,6), stg2data(d,7), stg2data(d,8));
131         mysql(str);
132
133     else
134         d = d;
135     end
136 end
137
138 %% Writing the formatted data to a file
139
140 fid = fopen(dataText2, 'wt');
141 [rows cols] = size(stg2data);
142 for z = 1:rows,
143     if stg2data(z,4) == -1,
```

```
144         fprintf(fid,'%6.2f\t', stg2data(z,1:end-6));
145         fprintf(fid,'%6.2f *\t', stg2data(z,3));
146         fprintf(fid,'%6.2f\t', stg2data(z,5:end-1));
147         fprintf(fid,'\''%d\''\n', stg2data(z,end));
148     else
149         fprintf(fid,'%6.2f\t', stg2data(z,1:end-1));
150         fprintf(fid,'\''%d\''\n', stg2data(z,end));
151     end
152 end
153
154 return;
```

Listing D.9: source–code/generateDataStgOne.m

```
1  function v = generateDataStgOne(data_ind, dataText2, curVec)
2  %% input arguments and initialization data
3  num = mysql('select max(Cluster_No) as Cluster_No from Stage2One');
4  d = 0;
5  if isnan(num),
6      pt = 0;
7  else
8      pt = num;
9  end
10
11 %% Extracting data
12 len = size(data_ind,1);
13 for count = 1:len,
14     noPort = 0;
15     mat = {[]}; port = []; src = []; dst = []; t = []; r = []; y = []; prio = []; time = [];
           event = [];
16     if length(data_ind{count}) ~= 0,
17         pt = pt + 1;
18         d = d+1;
19         no_alerts = size(data_ind{count},2);
20         for i = 1:no_alerts,
21             str = sprintf('select signature, ip_proto, layer4_sport, layer4_dport,
                   sig_priority from acid_event where id = %d', data_ind{count}(i));
22             [mat(i,1), mat(i,2), mat(i,3), mat(i,4), mat(i,5)] = mysql(str);
23         end
24         %% Filtering port numbers
25         for h = 1:no_alerts,
26             if isnan(mat(h,3)) || isnan(mat(h,4)),
27                 port(h) = -1;
28                 noPort = noPort + 1;
29             else
30                 str = sprintf('select distinct portNo from port where portNo = %d', mat(h,3));
31                 src = mysql(str);
32                 str = sprintf('select distinct portNo from port where portNo = %d', mat(h,4));
33                 dst = mysql(str);
34                 if length(src) == 0  && length(dst) ~= 0,
35                     port(h) = str2num(cell2mat(dst));
36                 elseif length(src) ~= 0 && length(dst) == 0,
37                     port(h) = str2num(cell2mat(src));
38                 elseif length(src) == 0 && length(dst) == 0,
```

```
39                  port(h) = min(mat(h,3),mat(h,4));
40              else
41                  port(h) = min(str2num(cell2mat(src)),str2num(cell2mat(dst)));
42              end
43          end
44      end
45      %% Identify no of alerts
46      stg2data(d,1) = no_alerts;
47
48      %% Identify no of signatures
49      c = 1;
50      t = mat(1,1);
51      for s = 2:no_alerts,
52          for ox = 1:s-1,
53              if (mat(ox,1) == mat(s,1)),
54                  c = c;
55                  break;
56              end
57              if (ox == s-1),
58                  c = c + 1;
59                  t = [t, mat(s,1)];
60              end
61          end
62      end
63      stg2data(d,2) = c;
64
65      %% Identify the protocol
66      for k = 1:no_alerts,
67          r = [r, mat(k,2)];
68      end
69      if find(r < 255),
70          if find(r >= 255),
71              stg2data(d,3) = 2;
72          else
73              stg2data(d,3) = 1;
74          end
75      else
76          stg2data(d,3) = 3;
77      end
78
79      %% Identify port number
80      if noPort == no_alerts,
81          stg2data(d,4) = -1;
82      elseif find(port < 1024 & port >= 0),
83          if find(port >= 1024),
84              stg2data(d,4) = 2;
85          else
86              stg2data(d,4) = 1;
87          end
88      else
89          stg2data(d,4) = 3;
90      end
91
92      %% Identify priority
```

```
93          for p = 1:no_alerts,
94              if mat(p,5) == 1,
95                  prio = [prio, 300];
96              elseif mat(p,5) == 2,
97                  prio = [prio, 200];
98              else
99                  prio = [prio, 100];
100             end
101         end
102         stg2data(d,5) = sum(prio);
103
104         %% Calculate time interval and no of events
105         for ct = 1:4,
106             if curVec(ct) < 10,
107                 sx(ct) = sprintf('0%d',curVec(ct));
108             else
109                 sx(ct) = sprintf('%d',curVec(ct));
110             end
111         end
112         for p = 1:stg2data(d,2),
113             str = sprintf('select No_of_event, Time_interval from TimeEventOne%s%s%s%s where
                    Signature = %d', sx(1),sx(2),sx(3),sx(4),t(1,p));
114             [ev, ti] = mysql(str);
115             if isempty(ev),
116                 NextcurVec = curVec + [0 0 0 1 0 0];
117                 for ct = 1:4,
118                     if curVec(ct) < 10,
119                         sn(ct) = sprintf('0%d',curVec(ct));
120                     else
121                         sn(ct) = sprintf('%d',curVec(ct));
122                     end
123                 end
124                 str = sprintf('select No_of_event, Time_interval from TimeEventOne%s%s%s%s
                        where Signature = %d', sn(1),sn(2),sn(3),sn(4), t(1,p));
125                 [ev, ti] = mysql(str);
126             end
127             event(p) = ev;
128             time(p) = ti;
129         end
130         format long G, event;
131         stg2data(d,6) = max(time);
132         stg2data(d,7) = min(event);
133         stg2data(d,8) = pt;
134         str = sprintf('insert into timeEventRecordOne (Time_interval, No_of_event, Cluster_No)
                values (%.3f, %d, %d)', stg2data(d,6), stg2data(d,7), stg2data(d,8));
135         mysql(str);
136
137     else
138         d = d;
139     end
140 end
141
142 %% Writing the formatted data to a file
143
```

```
144 fid = fopen(dataText2, 'wt');
145 [rows cols] = size(stg2data);
146 for z = 1:rows,
147     if stg2data(z,4) == -1,
148         fprintf(fid,'%6.2f\t', stg2data(z,1:end-6));
149         fprintf(fid,'%6.2f *\t', stg2data(z,3));
150         fprintf(fid,'%6.2f\t', stg2data(z,5:end-1));
151         fprintf(fid,'\''%d\''\n', stg2data(z,end));
152     else
153         fprintf(fid,'%6.2f\t', stg2data(z,1:end-1));
154         fprintf(fid,'\''%d\''\n', stg2data(z,end));
155     end
156 end
157
158 return;
```

Listing D.10: source–code/generateDataStgHalf.m

```
 1 function v = generateDataStgHalf(data_ind, dataText2, curVec)
 2 %% input arguments and initialization data
 3 num = mysql('select max(Cluster_No) as Cluster_No from Stage2Half');
 4 d = 0;
 5 if isnan(num),
 6     pt = 0;
 7 else
 8     pt = num;
 9 end
10
11 %% Extracting data
12 len = size(data_ind,1);
13 for count = 1:len,
14     noPort = 0;
15     mat = {[]}; port = []; src = []; dst = []; t = []; r = []; y = []; prio = []; time = [];
           event = [];
16     if length(data_ind{count}) ~= 0,
17         pt = pt + 1;
18         d = d+1;
19         no_alerts = size(data_ind{count},2);
20         for i = 1:no_alerts,
21             str = sprintf('select signature, ip_proto, layer4_sport, layer4_dport,
                   sig_priority from acid_event where id = %d', data_ind{count}(i));
22             [mat{i,1}, mat{i,2}, mat{i,3}, mat{i,4}, mat{i,5}] = mysql(str);
23         end
24         %% Filtering port numbers
25         for h = 1:no_alerts,
26              if isnan(mat{h,3}) || isnan(mat{h,4}),
27                 port(h) = -1;
28                 noPort = noPort + 1;
29              else
30                 str = sprintf('select distinct portNo from port where portNo = %d', mat{h,3});
31                 src = mysql(str);
32                 str = sprintf('select distinct portNo from port where portNo = %d', mat{h,4});
33                 dst = mysql(str);
34                 if length(src) == 0  && length(dst) ~= 0,
```

```
35                  port(h) = str2num(cell2mat(dst));
36              elseif length(src) ~= 0 && length(dst) == 0,
37                  port(h) = str2num(cell2mat(src));
38              elseif length(src) == 0 && length(dst) == 0,
39                  port(h) = min(mat(h,3),mat(h,4));
40              else
41                  port(h) = min(str2num(cell2mat(src)),str2num(cell2mat(dst)));
42              end
43          end
44      end
45      %% Identify no of alerts
46      stg2data(d,1) = no_alerts;
47
48      %% Identify no of signatures
49      c = 1;
50      t = mat(1,1);
51      for s = 2:no_alerts,
52          for ox = 1:s-1,
53              if (mat(ox,1) == mat(s,1)),
54                  c = c;
55                  break;
56              end
57              if (ox == s-1),
58                  c = c + 1;
59                  t = [t, mat(s,1)];
60              end
61          end
62      end
63      stg2data(d,2) = c;
64
65      %% Identify the protocol
66      for k = 1:no_alerts,
67          r = [r, mat(k,2)];
68      end
69      if find(r < 255),
70          if find(r >= 255),
71              stg2data(d,3) = 2;
72          else
73              stg2data(d,3) = 1;
74          end
75      else
76          stg2data(d,3) = 3;
77      end
78
79      %% Identify port number
80      if noPort == no_alerts,
81          stg2data(d,4) = -1;
82      elseif find(port < 1024 & port >= 0),
83          if find(port >= 1024),
84              stg2data(d,4) = 2;
85          else
86              stg2data(d,4) = 1;
87          end
88      else
```

```
89          stg2data(d,4) = 3;
90      end
91
92      %% Identify priority
93      for p = 1:no_alerts,
94          if mat(p,5) == 1,
95              prio = [prio, 300];
96          elseif mat(p,5) == 2,
97              prio = [prio, 200];
98          else
99              prio = [prio, 100];
100         end
101     end
102     stg2data(d,5) = sum(prio);
103
104     %% Calculate time interval and no of events
105     for ct = 1:5,
106         if curVec(ct) < 10,
107             sx(ct) = sprintf('0%d',curVec(ct));
108         else
109             sx(ct) = sprintf('%d',curVec(ct));
110         end
111     end
112     for p = 1:stg2data(d,2),
113         str = sprintf('select No_of_event, Time_interval from TimeEventHalf%s%s%s%s%s
                where Signature = %d', sx(1),sx(2),sx(3),sx(4),sx(5),t(1,p));
114         [ev, ti] = mysql(str);
115         if isempty(ev),
116             NextcurVec = curVec + [0 0 0 0 30 0];
117             for ct = 1:5,
118                 if curVec(ct) < 10,
119                     sn(ct) = sprintf('0%d',curVec(ct));
120                 else
121                     sn(ct) = sprintf('%d',curVec(ct));
122                 end
123             end
124             str = sprintf('select No_of_event, Time_interval from TimeEventHalf%s%s%s%s%s
                    where Signature = %d', sn(1),sn(2),sn(3),sn(4),sn(5), t(1,p));
125             [ev, ti] = mysql(str);
126         end
127         event(p) = ev;
128         time(p) = ti;
129     end
130     format long G, event;
131     stg2data(d,6) = max(time);
132     stg2data(d,7) = min(event);
133     stg2data(d,8) = pt;
134     str = sprintf('insert into timeEventRecordHalf (Time_interval, No_of_event, Cluster_No
            ) values (%.3f, %d, %d)', stg2data(d,6), stg2data(d,7), stg2data(d,8));
135     mysql(str);
136
137 else
138     d = d;
139 end
```

```
140 end
141
142 %% Writing the formatted data to a file
143
144 fid = fopen(dataText2, 'wt');
145 [rows cols] = size(stg2data);
146 for z = 1:rows,
147     if stg2data(z,4) == -1,
148         fprintf(fid,'%6.2f\t', stg2data(z,1:end-6));
149         fprintf(fid,'%6.2f *\t', stg2data(z,3));
150         fprintf(fid,'%6.2f\t', stg2data(z,5:end-1));
151         fprintf(fid,'\''%d\''\n', stg2data(z,end));
152     else
153         fprintf(fid,'%6.2f\t', stg2data(z,1:end-1));
154         fprintf(fid,'\''%d\''\n', stg2data(z,end));
155     end
156 end
157
158 return;
```

D.5 Alarm Aggregation Process

Listing D.11: source–code/alarmAggregateTwo.m

```
1  function data_ind = alarmAggregateTwo(dataText1, figureName1, stage1Res, minID)
2  %% input arguments and initialization data
3
4  %% Normalising data
5
6  sA = som_read_data(dataText1);
7  sA = som_normalize(sA,'var');
8  sA.data(:,1) = 1.8 * sA.data(:,1);
9  sA.data(:,2) = 1.8 * sA.data(:,2);
10 y = munits_best(sA);
11 sB = som_make(sA,'munits',y);
12 mysql('close');
13 data_ind = kmeans_beststg1(sB, sA, figureName1, minID);
14 mysql('open');
15 mysql('use','darpa2');
16 data_ind(cellfun(@isempty,data_ind)) = [];
17
18 %% Write the output into the database
19
20 No_alerts = length(sA.data);
21 No_clusters = length(data_ind);
22 result = cell(2,1);
23 num = mysql('select max(Cluster_No) as Cluster_No from Stage1Two');
24 if isnan(num),
25     h = 0;
26 else
27     h = num;
28 end
```

```
for i = 1:No_alerts,
    for j = 1:No_clusters,
        clust = h * j;
        len = length(data_ind{j});
        for x = 1:len,
            if minID == data_ind{j}(x),
                result{1} = [result{1}, minID];
                result{2} = [result{2}, clust];
                break;
            end
        end
        if length(result{1}) ~= 0 && result{1}(end) == i,
            break;
        end
    end
    minID = minID + 1;
end
for c = 1:No_alerts,
    str = sprintf('insert into Stage1Two(id, Cluster_No) values(%d,%d)', result{1}(c), result
        {2}(c));
    mysql(str);
end
%% Writing data index into a file
fid = fopen(stage1Res, 'wt');
pj = length(data_ind);
for z = 1:pj,
    len =  length(data_ind{z});
    if len == 0,
        fprintf(fid,'NA\n');
    else
        fprintf(fid,'%d ', data_ind{z}(1:len-1));
        fprintf(fid,'%d\n', data_ind{z}(len));
    end
end

return;
```

Listing D.12: source–code/alarmAggregateOne.m

```
function data_ind = alarmAggregateOne(dataText1, figureName1, stage1Res, minID)
%% input arguments and initialization data

%% Normalising data

sA = som_read_data(dataText1);
sA = som_normalize(sA,'var');
sA.data(:,1) = 1.8 * sA.data(:,1);
sA.data(:,2) = 1.8 * sA.data(:,2);
y = munits_best(sA);
sB = som_make(sA,'munits',y);
mysql('close');
data_ind = kmeans_reststg1(sB, sA, figureName1, minID);
mysql('open');
mysql('use','Carpa2');
```

```
16 data_ind(cellfun(@isempty,data_ind)) = [];
17
18 %% Write the output into the database
19
20 No_alerts = length(sA.data);
21 No_clusters = length(data_ind);
22 result = cell(2,1);
23 num = mysql('select max(Cluster_No) as Cluster_No from Stage1One');
24 if isnan(num),
25     h = 0;
26 else
27     h = num;
28 end
29 for i = 1:No_alerts,
30     for j = 1:No_clusters,
31         clust = h + j;
32         len = length(data_ind{j});
33         for x = 1:len,
34             if minID == data_ind{j}(x),
35                 result{1} = [result{1}, minID];
36                 result{2} = [result{2}, clust];
37                 break;
38             end
39         end
40         if length(result{1}) ~= 0 && result{1}(end) == i,
41             break;
42         end
43     end
44     minID = minID + 1;
45 end
46 for c = 1:No_alerts,
47     str = sprintf('insert into Stage1One(id, Cluster_No) values(%d,%d)', result{1}(c), result
         {2}(c));
48     mysql(str);
49 end
50 %% Writing data index into a file
51 fid = fopen(stage1Res, 'wt');
52 pj = length(data_ind);
53 for z = 1:pj,
54     len =  length(data_ind{z});
55     if len == 0,
56         fprintf(fid,'NA\n');
57     else
58         fprintf(fid,'%d ', data_ind{z}(1:len-1));
59         fprintf(fid,'%d\n', data_ind{z}(len));
60     end
61 end
62
63 return;
```

Listing D.13: source–code/alarmAggregateHalf.m

```
1 function data_ind = alarmAggregateHalf(dataText1, figureName1, stage1Res, minID)
2 %% input arguments and initialization data
```

```
3
4  %% Normalising data
5
6  sA = som_read_data(dataText1);
7  sA = som_normalize(sA,'var');
8  sA.data(:,1) = 1.8 * sA.data(:,1);
9  sA.data(:,2) = 1.8 * sA.data(:,2);
10 y = munits_best(sA);
11 sB = som_make(sA,'munits',y);
12 mysql('close');
13 data_ind = kmeans_beststg1(sB, sA, figureName1, minID);
14 mysql('open');
15 mysql('use','darpa2');
16 data_ind(cellfun(@isempty,data_ind)) = [];
17
18 %% Write the output into the database
19
20 No_alerts = length(sA.data);
21 No_clusters = length(data_ind);
22 result = cell(2,1);
23 num = mysql('select max(Cluster_No) as Cluster_No from Stage1Half');
24 if isnan(num),
25     h = 0;
26 else
27     h = num;
28 end
29 for i = 1:No_alerts,
30     for j = 1:No_clusters,
31         clust = h + j;
32         len = length(data_ind{j});
33         for x = 1:len,
34             if minID == data_ind{j}(x),
35                 result{1} = [result{1}, minID];
36                 result{2} = [result{2}, clust];
37                 break;
38             end
39         end
40         if length(result{1}) ~= 0 && result{1}(end) == i,
41             break;
42         end
43     end
44     minID = minID + 1;
45 end
46 for c = 1:No_alerts,
47     str = sprintf('insert into Stage1Half(id, Cluster_No) values(%d,%d)', result{1}(c), result
         {2}(c));
48     mysql(str);
49 end
50 %% Writing data index into a file
51 fid = fopen(stage1Res, 'wt');
52 pj = length(data_ind);
53 for z = 1:pj,
54     len =  length(data_ind{z});
55     if len == 0,
```

```
56          fprintf(fid,'NA\n');
57      else
58          fprintf(fid,'%d ', data_ind{z}(1:len-1));
59          fprintf(fid,'%d\n', data_ind{z}(len));
60      end
61  end
62
63  return;
```

D.6 Alarm Filtering Process

Listing D.14: source–code/alarmFilterTwo.m

```
1  function data_indFinal = alarmFilterTwo(dataText2,figureName2, finalIndex)
2  %% input arguments and initialization data
3
4  %% Normalising data
5
6  sA = som_read_data(dataText2, '*');
7  len = size(sA.data,1);
8  if length(isnan(sA.data(:,4))) == len,
9      sA.data(:,4) = -1;
10 end
11 sA = som_normalize(sA,'var');
12 sA.data(:,7) = 2.8 * sA.data(:,7);
13 sA.data(:,6) = 2.5 * sA.data(:,6);
14 y = munits_best(sA);
15 sB = som_make(sA,'munits',y);
16 %sB = som_make(sA,'msize',[25 10]);
17 num = mysql('select max(Cluster_No) as Cluster_No from Stage2Two');
18 [data_indFinal, rec_post] = kmeans_bestnew(num, sB, sA, figureName2);
19
20 %% Write output into the database
21 a1 = data_indFinal{1}(1);
22 a2 = data_indFinal{2}(1);
23 for t = 1:len,
24     if a1 == rec_post{1}(t),
25         cmp1 = rec_post{2}(t);
26         break;
27     end
28 end
29 for t = 1:len,
30     if a2 == rec_post{1}(t),
31         cmp2 = rec_post{2}(t);
32         break;
33     end
34 end
35 if sA.data(cmp1,7) > sA.data(cmp2,7),
36     x = 0;
37     y = 1;
38 elseif sA.data(cmp1,7) == sA.data(cmp2,7),
39     if sA.data(cmp1,1) > sA.data(cmp2,1),
```

```
40          x = 0;
41          y = 1;
42      elseif sA.data(cmp1,1) < sA.data(cmp2,1),
43          x = 1;
44          y = 0;
45      else
46          array = [2 5 6];
47          for d = 1:length(array),
48              if sA.data(cmp1,array(d)) < sA.data(cmp2,array(d)),
49                  x = 0;
50                  y = 1;
51                  break;
52              elseif sA.data(cmp1,array(d)) > sA.data(cmp2,array(d)),
53                  x = 1;
54                  y = 0;
55                  break;
56              end
57          end
58      end
59  else
60      x = 1;
61      y = 0;
62  end
63
64  result = cell(2,1);
65  num = mysql('select max(Cluster_No) as Cluster_No from Stage2Two');
66  if isnan(num),
67      h = 0;
68  else
69      h = num;
70  end
71  p1 = length(data_indFinal{1});
72  for i = 1:len,
73      clust = h + i;
74      for d = 1:p1,
75          if clust == data_indFinal{1}(d),
76              result{1} = [result{1},clust];
77              result{2} = [result{2}, x];
78              break;
79          end
80      end
81      if length(result{1}) == 0 || result{1}(end) ~= clust,
82          result{1} = [result{1}, clust];
83          result{2} = [result{2}, y];
84      end
85  end
86
87  for c = 1:len,
88      str = sprintf('insert into Stage2Two(Cluster_No, Alert_Status) values(%d,%d)', result{1}(c
          ), result{2}(c));
89      mysql(str);
90  end
91  %% Writing data index into a file
92  fid = fopen(finalIndex, 'wt');
```

```
93 for z = 1:2,
94     len =  length(data_indFinal{z});
95     if len == 0,
96         fprintf(fid,'NA');
97     else
98         fprintf(fid,'%d ', data_indFinal{z}(1:len-1));
99         fprintf(fid,'%d\n', data_indFinal{z}(len));
100    end
101 end
102
103 return;
```

Listing D.15: source–code/alarmFilterOne.m

```
1  function data_indFinal = alarmFilterOne(dataText2,figureName2, finalIndex)
2  %% input arguments and initialization data
3
4  %% Normalising data
5
6  sA = som_read_data(dataText2, '+');
7  len = size(sA.data,1);
8  if length(isnan(sA.data(:,4))) == len,
9      sA.data(:,4) = -1;
10 end
11 sA = som_normalize(sA,'var');
12 sA.data(:,7) = 2.8 * sA.data(:,7);
13 sA.data(:,6) = 2.5 * sA.data(:,6);
14 y = munits_best(sA);
15 sB = som_make(sA,'munits',y);
16 %sB = som_make(sA,'msize',[25 10]);
17 num = mysql('select max(Cluster_No) as Cluster_No from Stage2One');
18 [data_indFinal, rec_post] = kmeans_bestnew(num, sB, sA, figureName2);
19
20 %% Write output into the database
21 a1 = data_indFinal{1}(1);
22 a2 = data_indFinal{2}(1);
23 for t = 1:len,
24     if a1 == rec_post{1}(t),
25         cmp1 = rec_post{2}(t);
26         break;
27     end
28 end
29 for t = 1:len,
30     if a2 == rec_post{1}(t),
31         cmp2 = rec_post{2}(t);
32         break;
33     end
34 end
35 if sA.data(cmp1,7) > sA.data(cmp2,7),
36     x = 0;
37     y = 1;
38 elseif sA.data(cmp1,7) == sA.data(cmp2,7),
39     if sA.data(cmp1,1) > sA.data(cmp2,1),
40         x = 0;
```

```
41          y = 1;
42      elseif sA.data(cmp1,1) < sA.data(cmp2,1),
43          x = 1;
44          y = 0;
45      else
46          array = [2 5 6];
47          for d = 1:length(array),
48              if sA.data(cmp1,array(d)) < sA.data(cmp2,array(d)),
49                  x = 0;
50                  y = 1;
51                  break;
52              elseif sA.data(cmp1,array(d)) > sA.data(cmp2,array(d)),
53                  x = 1;
54                  y = 0;
55                  break;
56              end
57          end
58      end
59  else
60      x = 1;
61      y = 0;
62  end
63
64  result = cell(2,1);
65  num = mysql('select max(Cluster_No) as Cluster_No from Stage2One');
66  if isnan(num),
67      h = 0;
68  else
69      h = num;
70  end
71  p1 = length(data_indFinal{1});
72  for i = 1:len,
73      clust = h + i;
74      for d = 1:p1,
75          if clust == data_indFinal{1}(d),
76              result{1} = [result{1},clust];
77              result{2} = [result{2}, x];
78              break;
79          end
80      end
81      if length(result{1}) == 0 || result{1}(end) ~= clust,
82          result{1} = [result{1}, clust];
83          result{2} = [result{2}, y];
84      end
85  end
86
87  for c = 1:len,
88      str = sprintf('insert into Stage2One(Cluster_No, Alert_Status) values(%d,%d)', result{1}(c
          ), result{2}(c));
89      mysql(str);
90  end
91  %% Writing data index into a file
92  fid = fopen(finalIndex, 'wt');
93  for z = 1:2,
```

```
94      len =  length(data_indFinal{z});
95      if len == 0,
96          fprintf(fid,'NA');
97      else
98          fprintf(fid,'%d ', data_indFinal{z}(1:len-1));
99          fprintf(fid,'%d\n', data_indFinal{z}(len));
100     end
101 end
102
103 return;
```

Listing D.16: source–code/alarmFilterHalf.m

```
1  function data_indFinal = alarmFilterHalf(dataText2,figureName2, finalIndex)
2  %% input arguments and initialization data
3
4  %% Normalising data
5
6  sA = som_read_data(dataText2, '*');
7  len = size(sA.data,1);
8  if length(isnan(sA.data(:,4))) == len,
9      sA.data(:,4) = -1;
10 end
11 sA = som_normalize(sA,'var');
12 sA.data(:,7) = 2.8 * sA.data(:,7);
13 sA.data(:,6) = 2.5 * sA.data(:,6);
14 y = munits_best(sA);
15 sB = som_make(sA,'munits',y);
16 %sB = som_make(sA,'msize',[25 10]);
17 num = mysql('select max(Cluster_No) as Cluster_No from Stage2Half');
18 [data_indFinal, rec_post] = kmeans_bestnew(num, sB, sA, figureName2);
19
20 %% Write output into the database
21 a1 = data_indFinal{1}(1);
22 a2 = data_indFinal{2}(1);
23 for t = 1:len,
24     if a1 == rec_post{1}(t),
25         cmp1 = rec_post{2}(t);
26         break;
27     end
28 end
29 for t = 1:len,
30     if a2 == rec_post{1}(t),
31         cmp2 = rec_post{2}(t);
32         break;
33     end
34 end
35 if sA.data(cmp1,7) > sA.data(cmp2,7),
36     x = 0;
37     y = 1;
38 elseif sA.data(cmp1,7) == sA.data(cmp2,7),
39     if sA.data(cmp1,1) > sA.data(cmp2,1),
40         x = 0;
41         y = 1;
```

```
42      elseif sA.data(cmp1,1) < sA.data(cmp2,1),
43          x = 1;
44          y = 0;
45      else
46          array = [2 5 6];
47          for d = 1:length(array),
48              if sA.data(cmp1,array(d)) < sA.data(cmp2,array(d)),
49                  x = 0;
50                  y = 1;
51                  break;
52              elseif sA.data(cmp1,array(d)) > sA.data(cmp2,array(d)),
53                  x = 1;
54                  y = 0;
55                  break;
56              end
57          end
58      end
59  else
60      x = 1;
61      y = 0;
62  end
63
64  result = cell(2,1);
65  num = mysql('select max(Cluster_No) as Cluster_No from Stage2Half');
66  if isnan(num),
67      h = 0;
68  else
69      h = num;
70  end
71  p1 = length(data_indFinal{1});
72  for i = 1:len,
73      clust = h + i;
74      for d = 1:p1,
75          if clust == data_indFinal{1}(d),
76              result{1} = [result{1},clust];
77              result{2} = [result{2}, x];
78              break;
79          end
80      end
81      if length(result{1}) == 0 || result{1}(end) ~= clust,
82          result{1} = [result{1}, clust];
83          result{2} = [result{2}, y];
84      end
85  end
86
87  for c = 1:len,
88      str = sprintf('insert into Stage2Half(Cluster_No, Alert_Status) values(%d,%d)', result{1}(
          c), result{2}(c));
89      mysql(str);
90  end
91  %% Writing data index into a file
92  fid = fopen(finalIndex, 'wt');
93  for z = 1:2,
94      len =  length(data_indFinal{z});
```

```
95      if len == 0,
96          fprintf(fid,'NA');
97      else
98          fprintf(fid,'%d ', data_indFinal{z}(1:len-1));
99          fprintf(fid,'%d\n', data_indFinal{z}(len));
100     end
101 end
102
103 return;
```

D.7 k-Means Clusters Process for Stage 1

Listing D.17: source–code/kmeans_beststg1.m

```
1  function data_ind = kmeans_beststg1(sD, sA, figureName1, minID, n_max, c_max, verbose)
2
3  %% input arguments and initialization
4
5  if isstruct(sA),
6    if isfield(sA,'data'), D = sA.data;
7    else D = sA.codebook;
8    end
9  else D = sA;
10 end
11 [dlen dim] = size(D);
12 cl = round(dlen/2);
13 if nargin < 5 | isempty(n_max) | isnan(n_max), n_max = cl; end
14 if nargin < 6 | isempty(c_max) | isnan(c_max), c_max = 5; end
15 if nargin < 7 | isempty(verbose) | isnan(verbose), verbose = 0; end
16 t_max = 1;
17 e = zeros(t_max,1);
18 data_post = zeros(1,n_max);
19 data_ind = cell(n_max,1);
20 errcomp = realmax;
21
22 %% Choosing the best map
23
24 for w = 1:t_max,
25     [c, p, err, ind] = kmeans_clusters(sD, n_max, c_max, w, verbose);
26     [dummy,i] = min(err);
27     e(w,1) = err(i);
28     if err(i) < errcomp,
29         errcomp = err(i);
30         bestMap = p;
31         index = i;
32         iter = w;
33     end
34 end
35
36 som_show(sD,'color',{bestMap{index},sprintf('%d clusters',index)},'bar','none');
37 sD= som_autolabel(sD,sA,'add');
38 som_show_add('label',sD);
```

```
39 saveas(gcf, figureName1, 'fig');
40
41 bmus = som_bmus(sD,sA);
42 ze = bestMap(index);
43 [l attr] = size(sA.data);
44
45 for s = 1:l,
46     data_post(ze(bmus(s))) = data_post(ze(bmus(s))) + 1;
47     data_ind{ze(bmus(s))} = [data_ind{ze(bmus(s))}, minID];
48     minID = minID + 1;
49 end
50 for j = 1:n_max,
51     data_post(j);
52     data_ind(j);
53 end
54
55 return;
```

D.8 k-Means Clusters Process for Stage 2

Listing D.18: source–code/kmeans_bestnew.m

```
1 function [data_ind, rec_post] = kmeans_bestnew(num, sD, sA, figureName2, n_max, c_max, verbose
      )
2
3 %% input arguments and initialization
4
5 if isstruct(sD),
6   if isfield(sD,'data'), D = sD.data;
7   else D = sD.codebook;
8   end
9 else D = sD;
10 end
11 [dlen dim] = size(D);
12
13 if nargin < 5 | isempty(n_max) | isnan(n_max), n_max = 2; end
14 if nargin < 6 | isempty(c_max) | isnan(c_max), c_max = 5; end
15 if nargin < 7 | isempty(verbose) | isnan(verbose), verbose = 0; end
16 t_max = 500; % number of random trials
17 k = 0; % index for the possible cluster solutions (maps)
18 sse = []; % sum of squared error for each unique map
19 freq = []; % number of occurrance for each unique map
20 clust_post = cell(t_max,1); % cluster index
21 data_post = zeros(1,n_max);
22 data_ind = cell(n_max,1);
23 rec_post = cell(2,1);
24 %data_post = cell(t_max,1);
25 totSSE = 0;
26 sol = 0;
27 g = [];
28 nc_clust = []; % no of clusters
29 close = realmax;
```

```
30
31 %% Choosing the best map
32
33 % Sorting the maps
34 for w = 1:t_max,
35     [c, p, err, ind] = kmeans_clusters(sD, n_max, c_max, w, verbose);
36     [dummy,i] = min(err);
37     isavai = find(ismember(sse, err(i)));  %Check if sse array contains err(i) and return the
           index number
38     if ~isempty(isavai), % if the value is not empty
39         freq(isavai) = freq(isavai)+ 1;
40     else
41         k = k + 1; %counting unique map
42         sse = [sse, err(i)];
43         freq = [freq, 1];
44         clust_post{k} = p{i};
45         no_clust = [no_clust, i]; %ideal no of clusters chosen per unique map
46     end
47 end
48
49 % Compute the frequency rate
50 rate = zeros(1,k); % frequency rate
51 for t = 1:k,
52     rate(t) = freq(t)/t_max;
53 end
54 [high, index] = max(rate);
55
56 % Compute the second thresholding value (standard deviation)
57 st = std(rate);
58 % Apply the thresholding
59 if high > 0.6,
60     bestMap = index;
61 else
62     for m = 1:k,
63         if high <= st,
64             sol = k;
65             totSSE = totSSE + sum(sse);
66             g = [1:k];
67             break;
68         end
69         if rate(m) >= (high - st),
70             totSSE = totSSE + sse(m);
71             sol = sol + 1;
72             g = [g, m];
73         end
74     end
75
76     % Calculate the average SSE
77     aveSSE = totSSE/sol;
78     if sol == 2,
79         if sse(g(1)) < sse(g(2)),
80             bestMap = g(1);
81         else
82             bestMap = g(2);
```

```
        end
    else
        for n = 1:sol,
            diff = abs(sse(g(n)) - aveSSE);
            if close > diff,
                close = diff;
                bestMap = g(n);
            end
        end
    end
end

%figure(bestMap);
som_show(sD,'color',{clust_post{bestMap},sprintf('%d clusters',no_clust(bestMap))},'bar','none
    ');
sD= som_autolabel(sD,sA,'add');
som_show_add('label',sD);
saveas(gcf, figureName2, 'fig');
%data_post{bestMap}

bmus = som_bmus(sD,sA);
ze = clust_post{bestMap};
[l attr] = size(sA.data);
if isnan(num),
    h = 0;
else
    h = num;
end
for s = 1:l,
    label = h + s;
    rec_post{1} = [rec_post{1}, label];
    rec_post{2} = [rec_post{2}, s];

    data_post{ze(bmus(s))} = data_post{ze(bmus(s))} + 1;
    data_ind{ze(bmus(s))} = [data_ind{ze(bmus(s))}, label];
end
for j = 1:n_max,
    data_post{j};
    data_ind{j};
end

return;
```

E Functional Requirement Analysis

In order to gain insight into the main features of the prototype, several modelling languages are presented, including use case diagram, activity diagram, sequence diagram and class diagram.

1. Use case diagram

   A use case is a technique, which is typically used for visualising the functionality provided by a system in terms of actors, their actions (represented as use cases) and any interactions between these use cases (Miles, 2006). In general, each use case provides one or more scenarios that meticulously convey how the users should interact in order to perform a task. Bear in mind that a use case diagram does not portray any internal processes of the system nor do they explain the structure of the system. In fact, it is purely a functional description that is completely separate from the system design In order to show the role of the administrator in the system, a use case diagram is presented below:

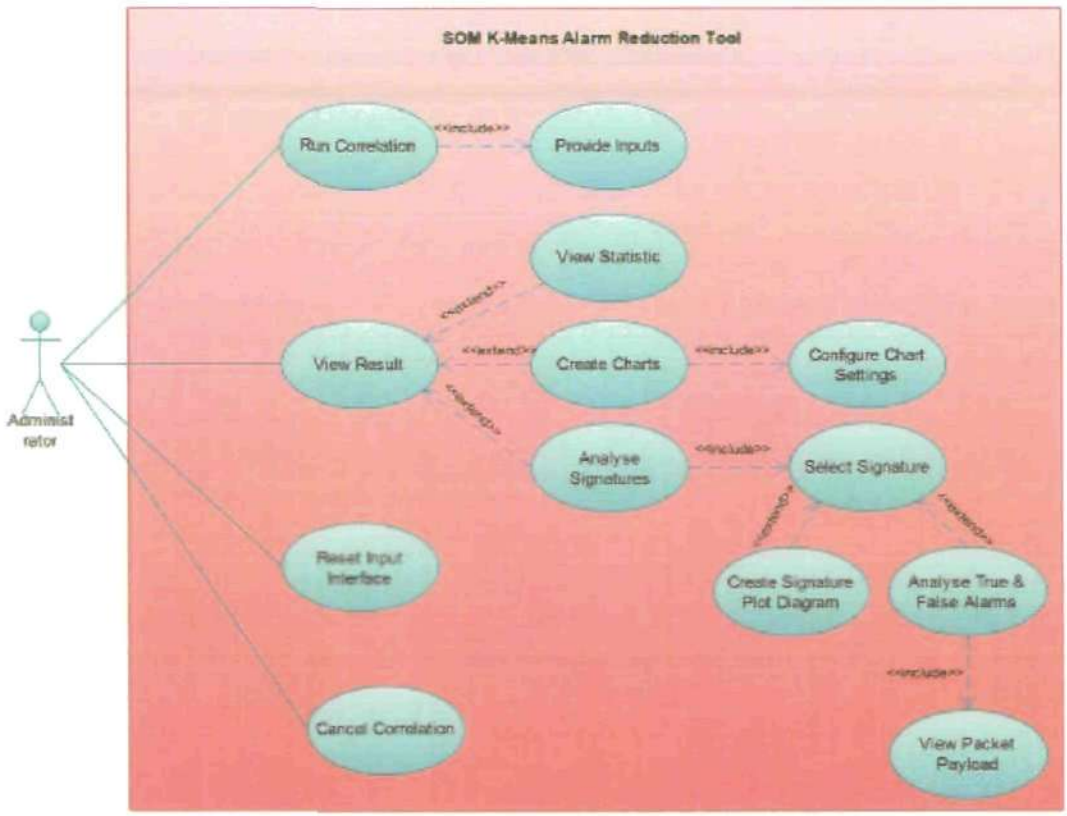


Figure E.1: SMART use case diagram

To summarise the role of the administrator as illustrated in Figure E.1, a brief description of the use case scenarios is given below:

To summarise the role of the administrator as illustrated in Figure E.1, a brief description of the use case scenarios is given below:

- The administrator has an ability to run the correlation after specifying inputs.
- The administrator also has a responsibility to evaluate the result of the correlation. It includes the ability to view the correlation result statistics, to create a chart report and to analyse the IDS signature rules. In terms of the signature analysis, the administrator can inspect the generation of true and false alarms per signature and also to generate a plot diagram that maps the generation of the alarms based on IP addresses and time period. In addition, the system also allows the administrator to view the packet payload and to examine the generated alerts individually.
- Apart from running the correlation, the administrator is also given an opportunity to reset or clear the input interface and to re-run the correlation.
- He is also able to cancel an ongoing correlation.

Although the use case diagram has provided a brief description about the functionality of the system and shown a nice roadmap of relationship between the administrator and the system, it does not clarify how those tasks are performed. To provide a clearer picture of the system and to show the steps that an administrator should follow to perform the tasks, an activity diagram is presented in the next sub-section.

2. Activity diagram

   (a) Run correlation

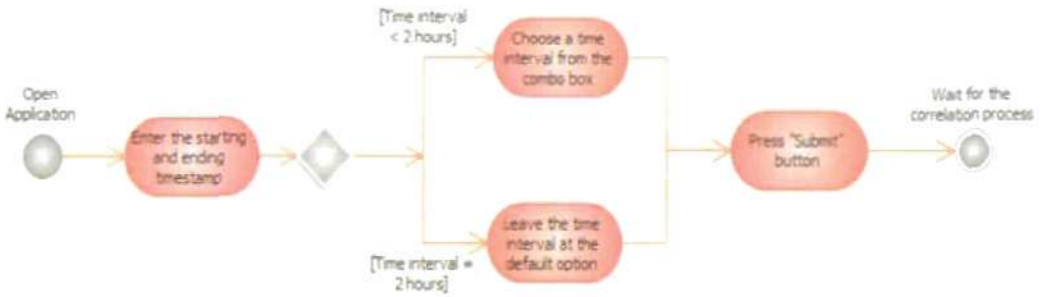


Figure E.2: Activity diagram - Run correlation

To run the correlation process, the administrator is prompted to specify the scope of the alerts to be processed by entering the starting and the ending timestamp of the alerts. Besides, the administrator is also required to make a decision whether to run the correlation for a time interval of 2 hours or less than 2 hours (either every one hour or every half an hour); as shown in Figure E.2.

   (b) View Statistic

Once the correlation is completed, the administrator can view the result of the classification by clicking the "View result" button. In this interface, there are 3 tabs available in a

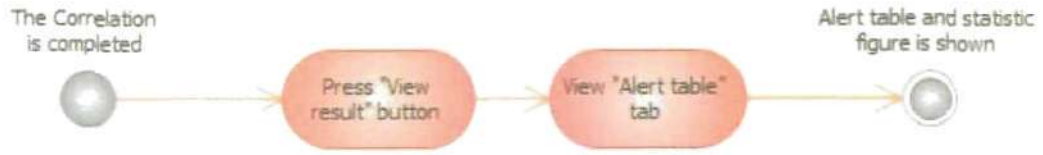


Figure E.3: Activity diagram - View statistic

single frame. The first tab "Alert table" presents a table containing the alerts attributes as well as the final status of the alerts (whether it is a true or false alarm). Furthermore, it also displays a statistic figure, which compares the proportion of the true and false alarms resulted from the correlation. This process is described in Figure E.3

(c) Create Chart

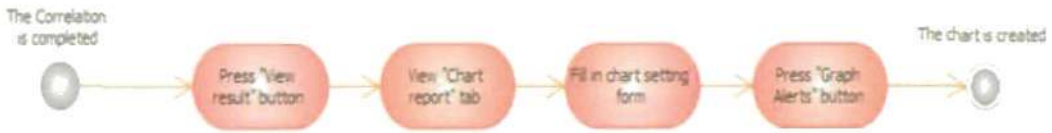


Figure E.4: Activity diagram - Create chart

Besides allowing the administrator to look at the overall correlation result as featured in the first tab, the second tab enables the administrator to fully inspect the generation of the true and false alarms based on their signatures in a particular period of time (Figure E.4). To conduct an alert evaluation, the system allows the administrator to create a chart report, which conveys the trend of the IDS alarms, by filling in the chart setting form.

(d) Analyse Signatures

In the last tab, "Signature analysis" (Figure E.5), the tool allows the administrator to analyse the alarms for each signature rule. A list of signatures is presented and the administrator is required to select one signature to conduct a further investigation. Once selected, a plot diagram and tables of the true and false alarms are automatically generated for the chosen signature. And to facilitate the alert evaluation, the payload of the packet triggering the alert can be examined through the conventional alert management tool, BASE.

(e) Cancel Correlation

Apart from executing the correlation, the administrator is also allowed to stop the ongoing correlation by pressing the "Cancel" button. Before the correlation is terminated, the administrator is prompted to confirm whether the act should proceed or be withdrawn (Figure E.6).

(f) Reset the Interface

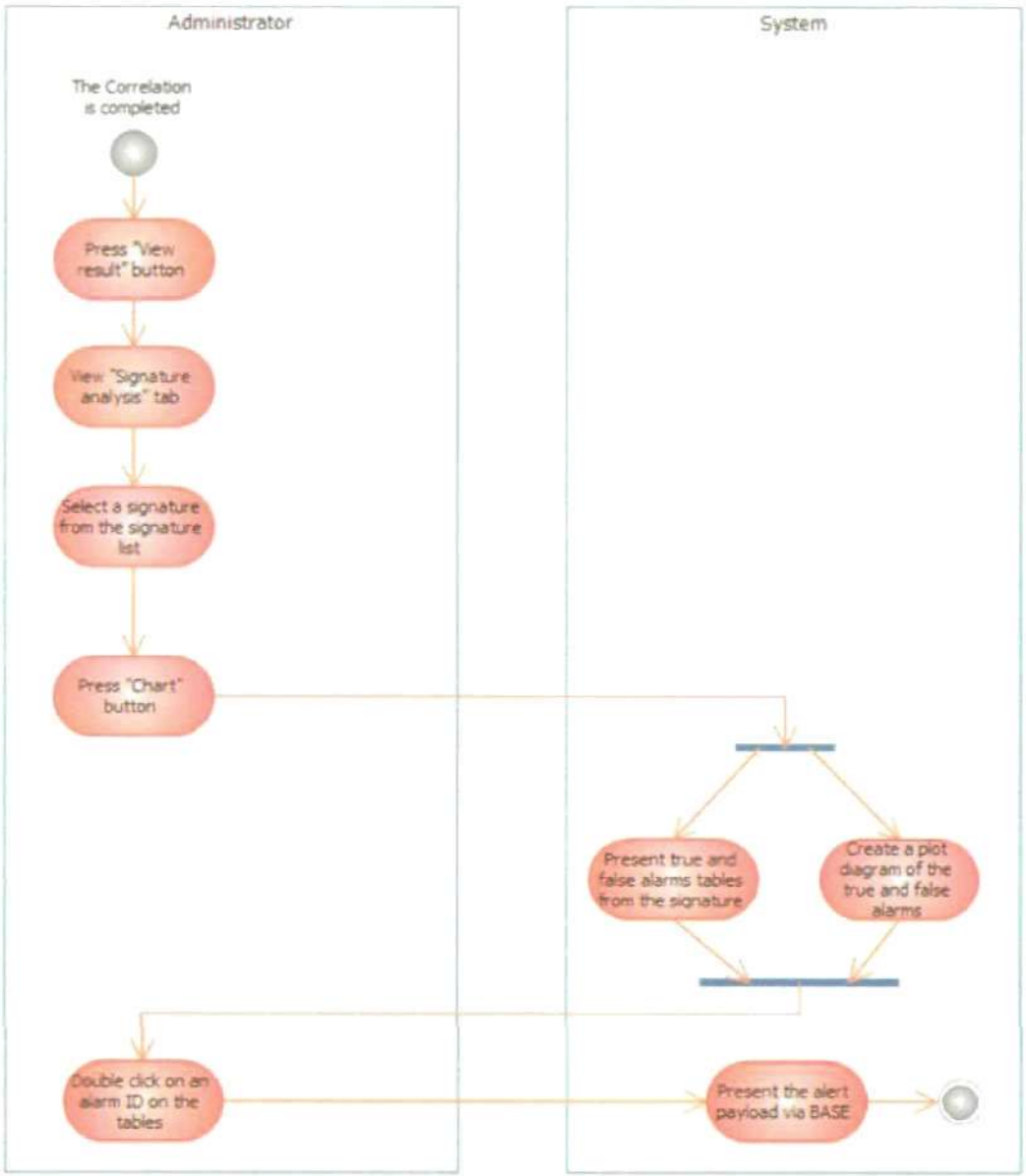


Figure E.5: Activity diagram - Analyse signatures

The last but not least, the administrator can reset the system by pressing the "Reset" button (Figure E.7). This enables the system to clear the input text boxes and return to the start page (home page).

3. Sequence diagram

A sequence diagram is a type of interaction diagram that is primarily used to model the flow of logic and processes within the system. It allows the documentation of the system's run-time scenarios in a graphical manner. The sequence diagram shown in Figure E.8 is created to depict the logic behind the correlation processes and the interaction between objects in sequential order.

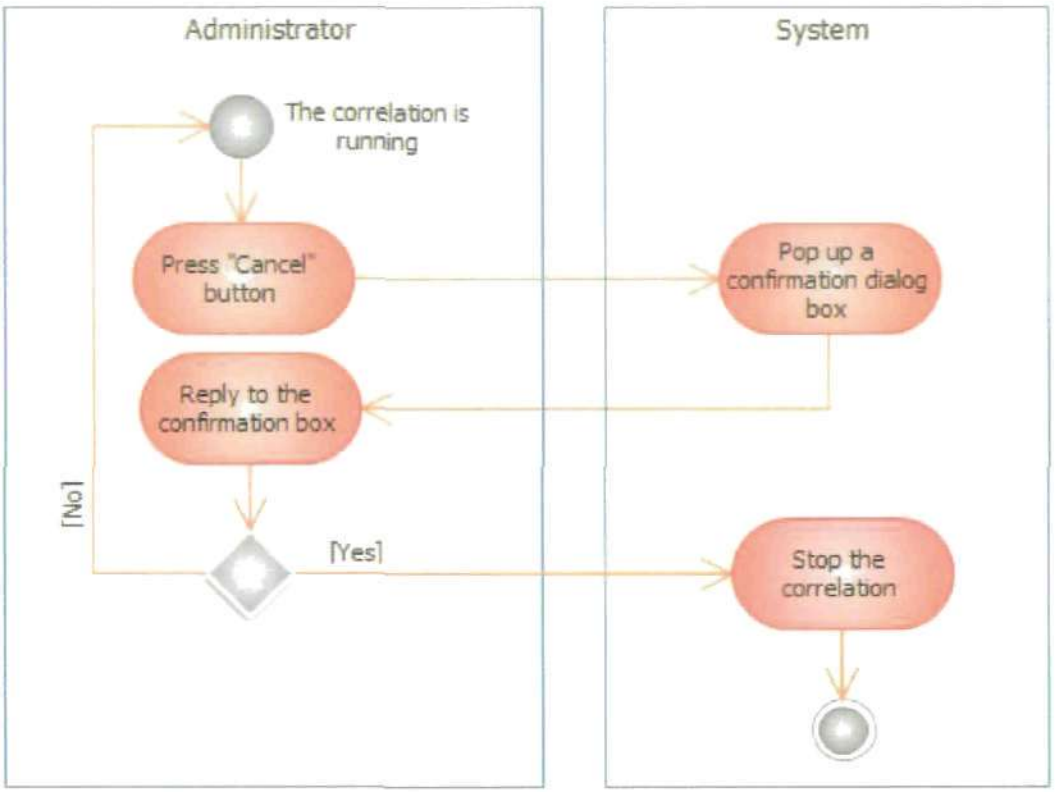


Figure E.6: Activity diagram - Cancel correlation

There are apparently 12 objects involved in the interactions, as described in the sequence diagram (Figure E.8). The first component is the actor (the administrator), which initiates and takes an active part in the scenarios, whilst the rest 11 objects are the components from either the external or internal systems.

(a) External system:

i. Signature-based IDS
ii. MySQL
iii. I/O files

(b) Internal system:

i. Input
ii. Output
iii. Pre-processing System
iv. Normalising System
v. Training System
vi. Aggregating System
vii. Alarm Classification
viii. Best Map Selector

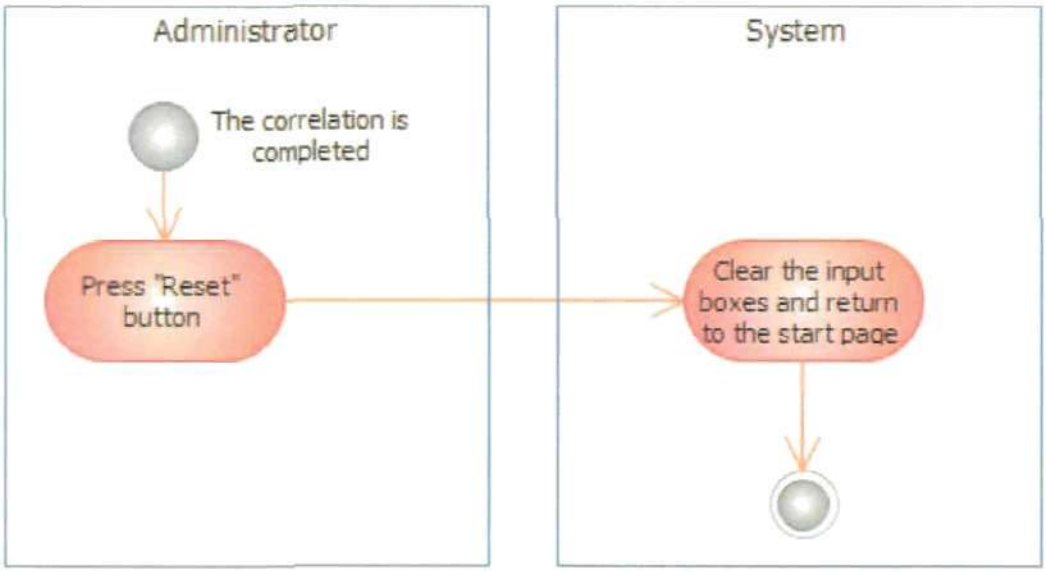


Figure E.7: Activity diagram - Reset the interface

4. Class diagram

   The main idea of creating a class diagram in this context is to describe the structure of the SMART system by showing the system's classes and the relationships between classes. Figure E.9 shows the static structure diagram of the system.

   Instead of focusing on the attributes and methods of the elements in the system, the diagram plainly depicts the association and the inheritance relationships between the classes. The relationship between the administrator, IDS and SMART system classes is defined as a standard kind of association or known as a bi-directional association, which is indicated by a solid line between the classes. On the other hand, a basic aggregation relationship is indicated by a solid line with an unfilled diamond shape between SMART System and File classes. An association with an aggregation relationship suggests that one class is part of another class and the child class instance can outlive its parent class. Moreover, the generalisation relationship can also be noticed from Sig-based IDS and Anomaly IDS classes, which inherit from the parent class IDS.

   To provide a better view of the structure of the proposed SMART system, the following list summarises the classes, which form the structure of the system, as well as the relationships between them.

   (a) The Administrator class takes on the role of "monitored" in the bi-directional relationships between both SMART system and IDS. The multiplicity value next to the Administrator class of 1..* means that when an instance of an IDS or SMART System exists, it can have at least one instance of an Administrator associated with it.

   (b) In the same association, the IDS takes on the role of "manages"; the diagram in Figure E.9 tells that the Administrator instance can be associated with no IDS or with up to an infinite number of IDSs. Conversely, an Administrator instance can be associated with

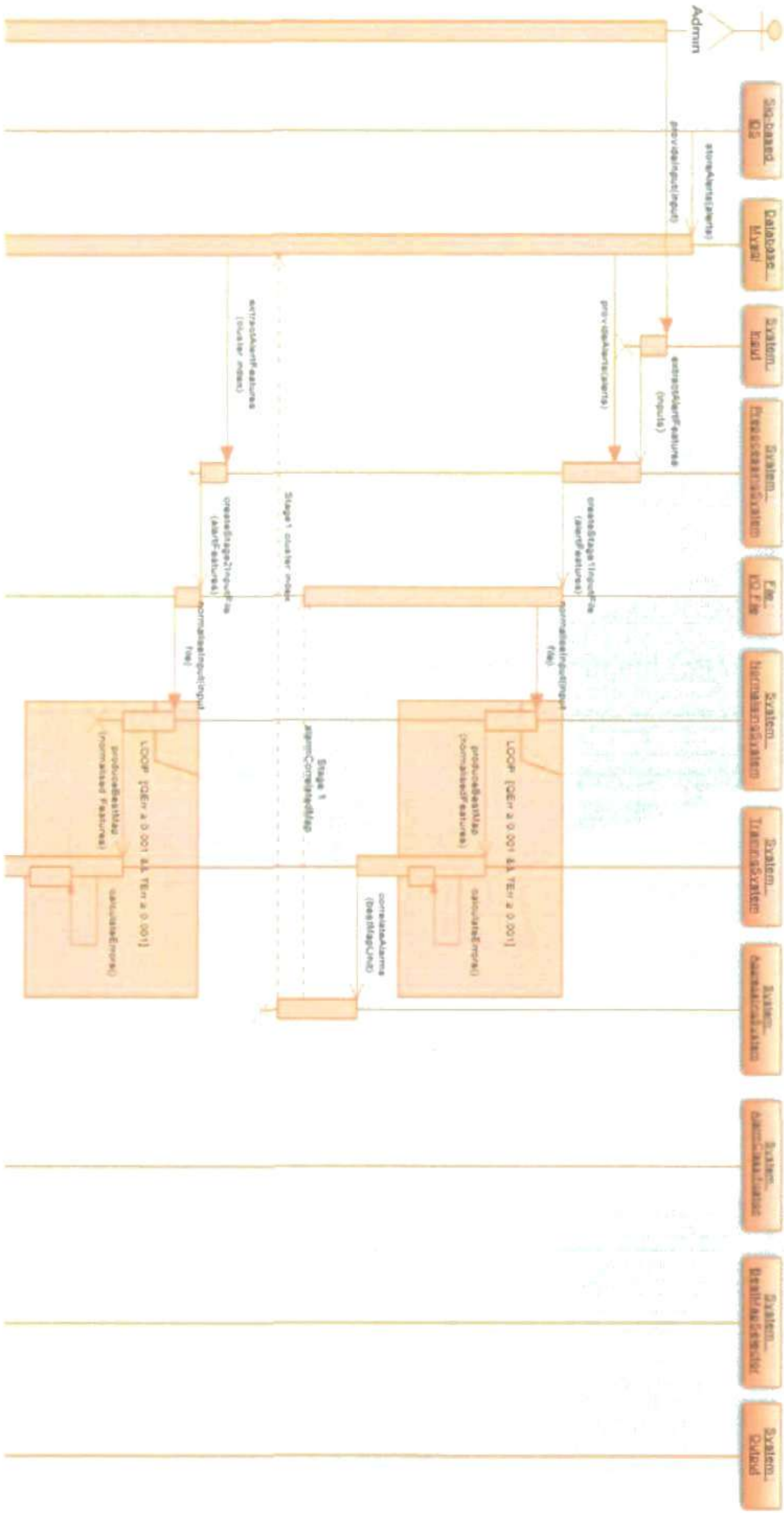
only one SMART System.

(c) Sig-based IDS and Anomaly IDS classes (child class) inherit from the IDS class (parent class). The inheritance relationship refers to the ability of a child class to inherit the identical functionality of another class (super class).

(d) A uni-directional association is explained by the relationship between the SMART System and the Sig-based IDS classes. This association is similar to the bi-directional association except that it only contains the role name and multiplicity value for the know class. For example, the SMART System knows about the Sig-based class, and the Sig-based class plays the role of "connected". However, unlike a standard association, the Sig-based *IDS class has no idea that it is associated with the SMART System class. A multiplicity* value of 1..* next to the Sig-based IDS class means that an instance of SMART System class can be associated with either one Sig-based IDS or an infinite number of Sig-based IDSs.

(e) Database class is an association class, which influences the relationship between SMART System and Sig-based IDS classes. An association class is a cross between an association and is represented as a regular class box that is connected to the association between the other two classes using a dashed line.

(f) The relationship between the SMART System and File classes is regarded as a basic aggregation relationship; in which case the lifecycle of the File class is independent from the SMART System class's lifecycle.


Admin

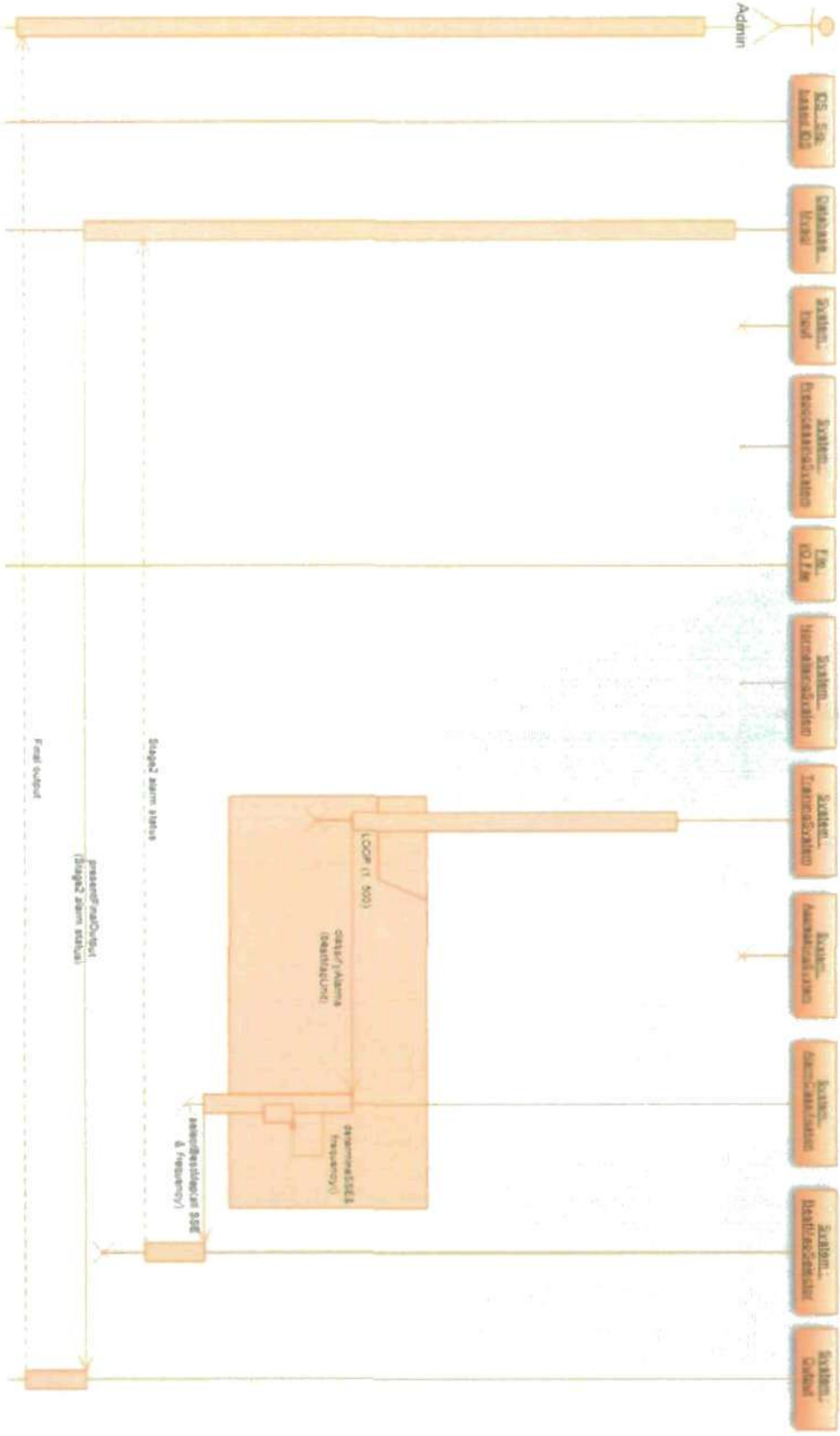


Figure E.8: SMART sequence diagram

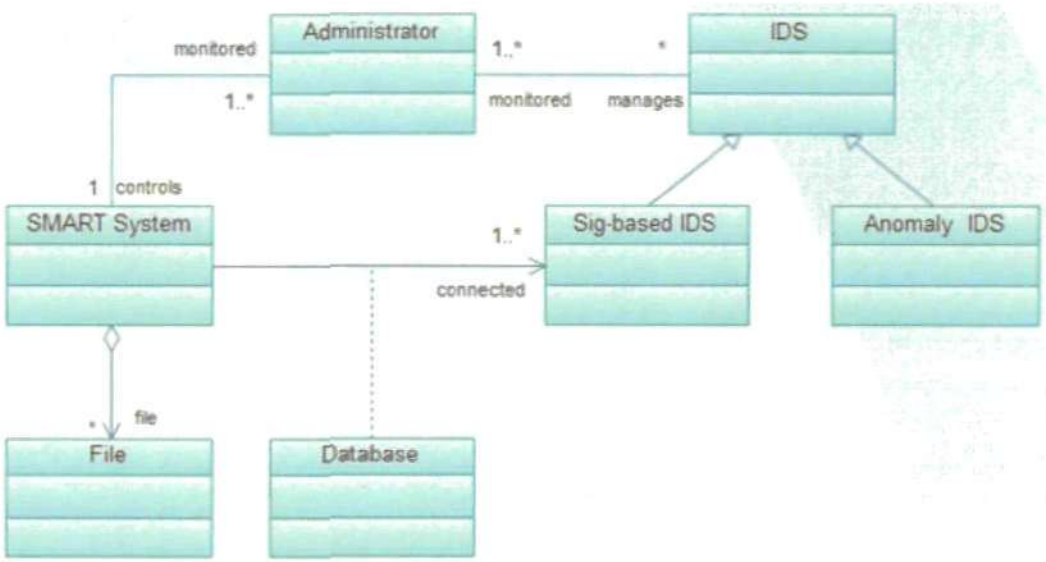


Figure E.9: SMART class diagram

F Running the Correlation System

There are two ways to start the SMART application, namely:

1. Via command line argument

   To open an application using a command line argument, the user needs to open a command prompt and type a command as shown below (or see Figure F.1):

   java -jar *AlarmFrame.jar*

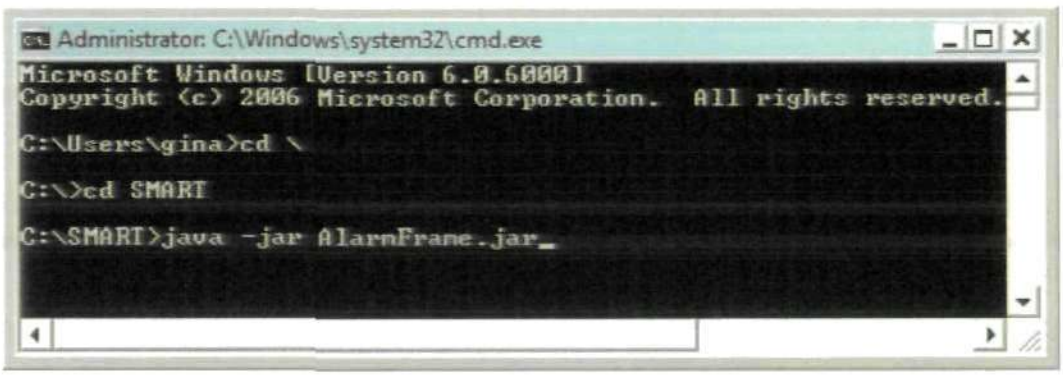


Figure F.1: Opening SMART application - via command prompt

   Make sure the working directory is set to the application folder (that is **SMART**) and the Java path has been added to the system variables. The latter allows the Java commands to be executed outside the Java application folder.

2. Via double-click

   In this option, the user is required to double-click on the executable jar file named "Alarm-Frame.jar" available in the application folder (see Figure F.2).

Once the program is executed, the application main page is presented; as shown in Figure F.3 below.

In this example, two weeks DARPA alerts generated by the Snort IDS are retrieved from the database and fed into the correlation system. Via the interface provided, the user is prompted to enter the starting and the ending timestamp as well as the time frame for each correlation (see Figure 7.3 in Chapter 7). It is worth noticing that the time windows for all correlations performed in this demonstration are set to 2 hours.

As for the input pattern, the system has set a standard timestamp format (YYYY-MM-DD HH:MM:SS) for the first two input arguments, namely the starting and the ending timestamps.

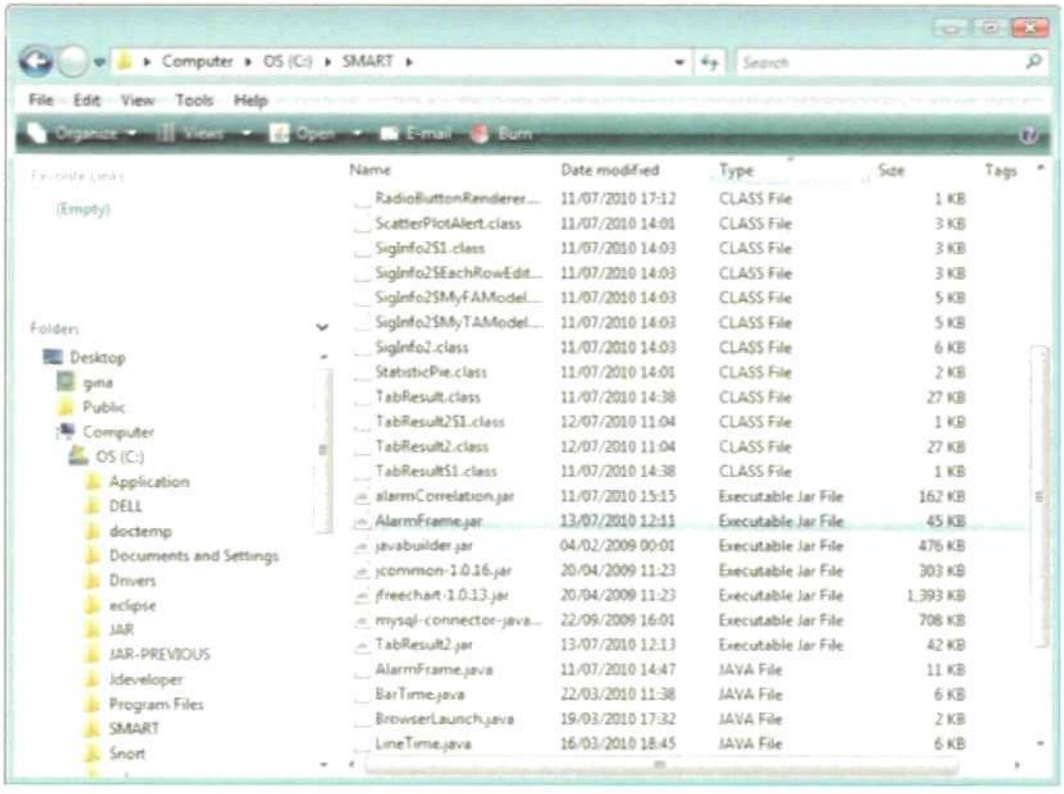


Figure F.2: Opening SMART application - via double-clicked

Failure to comply with the predefined format may result in a runtime error. Besides, an error dialog box will pop up; indicating the incorrect format used for the inputs (see Figure F.4).

Once the correlation is carried out, a progress bar (as depicted in Figure F.5) is shown to monitor the progress of a correlation run by the system.

Apart from executing the correlation, the user is also authorised to halt the ongoing correlation by pressing the "Cancel" button. Before the program is terminated, a confirm box, which pops up with both a Yes and a No button, is used to verify acceptance from the user (see Figure F.6). If the user accepts, then the user presses the "Yes" button and the program exits immediately. If the user rejects with the "No" button, then the correlation process continues.

Figure F.3: SMART - Front page

Figure F.4: SMART dialog box - Incorrect input format

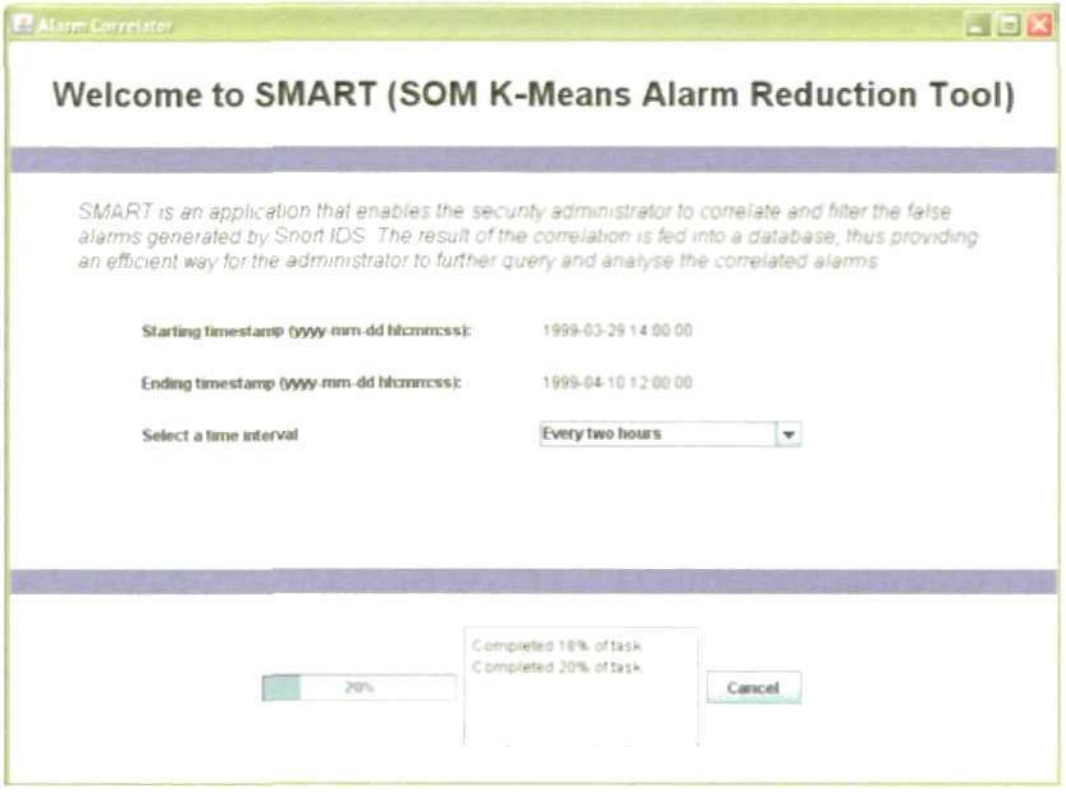


Figure F.5: SMART - Progress bar

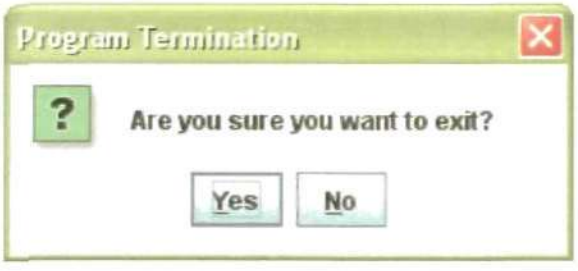


Figure F.6: SMART - Confirmation box

G Publications

1. Tjhai, G.C. (2007), 'Comprehensive approaches of intrusion detection in handling false alarm issue', Proceedings of the Third Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2007), Plymouth, UK, ISBN: 978-1-8410-2173-7, pp. 53-66

2. Tjhai, G.C., Papadaki, M., Furnell, S.M. and Clarke, N.L. (2008), 'The Problem of False Alarms: Evaluation with Snort and DARPA 1999 dataset, 5th International Conference: Trust, Privacy and Security in Digital Business (TrustBus 2008), Lecture Notes in Computer Science, Vol. 5185/2008, 2008, pp. 139-150

3. Tjhai, G.C, Papadaki, M., Furnell, S.M. and Clarke, N.L. (2008), 'Investigating the Problem of IDS False Alarms: An Experimental Study using Snort', In Proceedings of the IFIP TC 11 23rd International Information Security Conference. IFIP International Federation for Information Processing, Vol. 278, Boston: Springer, 2008, pp. 253-267

4. Tjhai, G.C., Furnell, S.M., Papadaki, M. and Clarke, N.L. (2010), 'A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm', Computers & Security, Vol. 29, No. 6, 2010, pp. 712-723

Comprehensive Approaches of Intrusion Detection in handling False Alarm Issue

Gina C. Tjhai

Network Research Group, University of Plymouth, Plymouth, United Kingdom
email: gctjhai@plymouth.ac.uk

Abstract

Intrusion detection is one of the most important tools in computer security. Although the technology has been actively developed for two decades, it is an indisputable fact that the art of detecting an intrusion is still far from perfect. IDS systems tend to generate a large number of false alarms per day, which adds a heavy workload for the administrator responsible in handling the alerts. In this paper, a number of current studies focusing upon the reduction of false alarms are briefly discussed. This paper also critically analyses the approaches implemented by current studies and provides recommendations to improve the performance of IDS in term of its alarm generation.

Keywords

Intrusion Detection, False alarm, Alert Correlation

1. Introduction

The Internet is an extremely promising mean of facilitating electronic access, thus the profit offered has motivated the growth of the Internet in many fields, such as eCommerce and online banking. This has led to a substantial change in business model of organisations across the world, and today, more and more people are getting connected to the Internet to take advantage of the e-Business model. The effectiveness and efficiency offered has rendered it invaluable for business activities.

Although the efficiency and effectiveness of email and Internet access for carrying out business and email are certainly offering tremendous benefits to the companies, connecting an internal LAN to the external Internet is a risky decision. In recent years, the security of computer networks has become significantly important. Most discussions have been focused on the tools or techniques by which network security could be effectively enhanced. It is also worth noticing that a number of network security measures have been publicly introduced to today's IT community, such as firewall and anti virus systems. However, having firewalls and anti-virus systems installed could not fully protect the network infrastructure from modern network attacks and numerous system vulnerabilities. Indeed, the rapid growth of Internet threats has rendered them inefficient in protecting company's information assets. In spite of those security tools, one of the most apparent network tools being developed, and which has continuously grown in popularity, is the Intrusion Detection System (IDS).

Basically, an IDS is a system which refers to all processes used in detecting an unauthorised uses of network and computer devices (Bruneau, 2001). IDS, much like the security industry itself, has grown rapidly over the past two decades (Goeldenitz, 2002). This measure has become one of the most vital components of defensive measure protecting computer system

and networks from abuse. Even though intrusion detection technology is still in its infancy, and could not act as a complete security defence, it could definitely play a significant role in an overall security architecture.

However, since ensuring security is a dynamic process, security tools are required to keep pace with changes. There is no security measure that can be proved to be 100% effective in protecting a network. Moreover, it is an indisputable fact that the art of detecting intrusions is still far from perfect, and IDS systems tend to generate a large number of false alarms (Allen et al., 2000). Hence a human has to validate alarms before any action can be taken. As IT infrastructure become larger and more complicated, the number of alarms that need to be reviewed can escalate rapidly, making this task very difficult to manage. Although fine-tuning procedures and disabling signatures are known to be one of the most effective ways to reduce false alarm rate in IDS technology, they might also degrade security level and subsequently increase the risk of missing real attacks.

This paper particularly focuses on the extent of the IDS false alarm problem and what current research has been done thus far in improving the performance of IDS by using alert correlation methods. Section 2 provides an overview of IDS technology, as well as the major challenges faced by the existing intrusion detection. Section 3 discusses significant research carried out in the area of intrusion detection. The idea of alert correlation and corresponding studies are presented in section 4. Finally, section 5 discusses significant alert correlation research study that focuses upon Artificial Intelligence techniques. The conclusions and future research direction are presented in section 6.

2. Background

IDS has played a vital role in the overall security infrastructure, as one last defence against computer attacks behind secure network architecture design, secure program design and firewalls (Allen et al., 2000). IDS products have become widely available in recent years, and have started to gain acceptance in the enterprise domain as a valuable improvement on security.

Although an IDS maybe used in combination with a firewall, which are aimed to control and filter the flow of information, these two tools have a different responsibility in safeguarding information security. Although a firewall does a good job in filtering traffic coming from the Internet, there has been a certain way a malicious user can compromise or circumvent the firewall system. The existence of intrusion detection which acts as a second line of defense does offer an adequate level of security, by monitoring, detecting and responding to the unauthorised activities which could bypass the firewall system. In addition, it is worth remembering that IDS is not a silver bullet when it comes to protecting system or network infrastructure. Instead, it is only one aspect of multi-layered protective mechanism, an approach referred to as 'defense in depth' (McHugh et al., 2000).

2.1 Challenges of Intrusion Detection

Today, intrusion detection has become an integral part of multi-layer security infrastructure and evolved into a viable and highly recommended piece of security technology that a company should implement as part of its collection of security tool. However, the art of detection is still far from ideal; intrusion detection technology is still in its infancy. As a result, current IDS technology has faced a number of challenges; one of them is the problem

of controlling a large number of triggered alerts. This issue is aggravated by the fact that some commercial IDSs may generate thousands of alarms per day. Recognising the real alarms from the huge volume of alarms is a frustrating task for security officers. Therefore, reducing false alarms is a serious problem in IDS efficiency and usability. Indeed, a high rate of false alarms is considered to be the limiting factor for the performance of intrusion detection system. False alerts always cause an additional workload for IT personnel, who must handle and verify every single alert generated to inhibit or block possible loss of data confidentiality, integrity and availability. The manual verification of these true and false alarms among the flood of alerts is not only deemed to be labour intensive but also error prone.

False positive alarms are caused by normal non-malicious background traffic. Especially for IDS technology that depends on behaviour modelling (anomaly-based IDS), this appears to be very critical issue. In learning the system or users' behaviours, not all behaviour could be covered and identified in detail. Behaviour can change from time to time. Sometimes, a legitimate user could act in an unusual manner or behaviour; differ from the expected behaviour (i.e. that which is recognised and learnt previously by the system). If the IDS solely relies on this model of normal or valid behaviours, a legitimate user who works in an uncommon way might be suspected as malicious intruder. Moreover, the system might also experience a real attack in learning phase (i.e. when the system is collecting and learning the users behaviours profile) (Lundin and Jonsson, 2003). If this occurs, an intrusive behaviour would be added into the behaviour profile, thus it would never be detected as anomalous.

One of the best ways to reduce the false alarm rate is by performing a tuning procedure. Tuning an IDS can be done by adapting the signature policy to the specific environment and disabling the signatures that are not related to it (Chapple, 2003). This is driven by the fact that some vulnerabilities exist in a particular OS platform only. Although tuning does offer a good solution in reducing a large number false alarms, this procedure could possibly exacerbate the situation by degrading the security level and increasing the risk of missing noteworthy incidents. Therefore, the tuning problem is actually a trade-off between reducing false alarms and maintaining the security level. Furthermore, tuning requires a thorough examination of the environment by qualified IT personnel and requires a frequent update to keep up with the flow of new vulnerabilities or threats discovered.

The number of alerts generated by an IDS could be very large, for example 15,000 alerts per day per sensor (Cuff, 2003). Reducing the false alarm rate is not an easy task. Indeed, it often worsens the situation by causing poorer IDS reliability or accuracy. Due to this issue, a plethora of research has been conducted to address this problem. The rest of this paper examines the nature of the activity to date.

3. Research in alleviating the problem of false alarm

As the false positive alarm has become a universal problem, which affects both signature- and anomaly-based IDS, providing a solution to this issue is critical for enhancing the efficiency and usability of intrusion detection as an effective security tool. One of the reasons why IDS technology generates a high false positive rate is the lack of correlation between input and output traffic, which can essentially look for abnormal output traffic (Bolzoni and Etalle, 2006). The main concept which has motivated this study is the idea that a successful intrusion to a system usually generates an anomaly in the outgoing traffic of the system. Conversely, if there is no anomalous output being produced by the system even though something in the input of the system causes the intrusion detection to raise alarms, those alarms are considered

to be false positives. Significantly, the system proposed, which is known as APHRODITE consists of two main components, namely Output Anomaly Detector (OAD) and correlation engine. OAD has responsibility for monitoring the output of the system and by referring to a statistical model describing the normal output, it flags any behaviour that deviates from the pre-defined model as a possible attack. On the other hand, the correlation engine, which is implemented with a stateful-inspection mechanism, is assigned to correlate the input to the output of the system belonging to a same communication. Through the process of tracking and combining input and output traffic, it would make it easier for IT personnel to learn and identify the possible result of a potential attack.

APHRODITE has various advantages in terms of operational factors. It is considered to work effectively without an optimal training (without using attack-free traffic) and is able to successfully detect an unknown attack without requiring the definition of new signatures. In addition, this system has also been proved to effectively reduce the false positive rate while increasing its detection rate. However, despite these benefits, this system is still not able to reduce the number of redundant alerts produced by the same event, and not able to conduct a real-time inspection, since the output of the event is required as the prerequisite of the detection procedure.

Similar research has also been done to improve the usability and efficiency IDS technology by reducing the number of false alarms while maintaining the level of security achieved (Law and Kwok, 2004). This approach works by monitoring and detecting abnormal patterns, which are then considered to be suspicious incident, from tones of alert generated by the system. It is believed that when an attack occurs, the alerts produced from the IDS will have different patterns from the one generated in an attack-free environment. In this approach, the main idea of the study is to let the false alarms be generated as they are, and then to determine whether the incoming alarm sequence generated are deviated from normal situations. Those alarms, which are classified to be normal, can be ignored (considered as false positives). In this sense, this method will reduce the number of alerts being triggered by the system before they are transferred to the security officer for further investigation.

By using KNN (K-Nearest Neighbour) classification technique, this approach is achieved by modelling normal alarm patterns with an N-dimensional space (using a data point). A new data point will be created once newly arrived alarms have been detected by the system. If the new point is close to the normal point, which has been modelled previously as a rule pattern, the novel data is considered as normal (false alarm), otherwise it is deemed to be a malicious attack. In other words, the distances between the novel point and the normal point indicate the differences between these two data; the further the new point from the normal one, the higher the risk of being attacked.

Although this model is believed to successfully reduce the number of false alarms triggered by the system while maintaining its detection rate, it has not been applied on live data and implemented in the real life environment. For that reason, there is still much more work to be undertaken in order to assert that this idea is applicable to existing IDSs under real life environment.

The idea to perform data mining in order to reduce false alarms has been explored by Julisch (2001). The main idea of this research is to find alarm clusters and generalised forms of false alarms to analyse root causes. Significantly, this study has also found that 90% of false alarms are related to a small number of root causes. By identifying the root causes, it is believed that

human expertise could manage the IDS or remove the root causes in order to reduce the number of false alarms. In addition, looking at potential reason of the alert generation, this research has also been expanded to look for the rules, which could predict a prospective alert when a specific set of alarms has been generated, or known as episode rules (Julisch and Dacier, 2002). With the rule or knowledge of the alarm patterns representing legitimate users being identified beforehand, it would be much easier for the system to filter out any similar patterns (which are supposed to be legitimate as well) in the future. Even though this approach is considered to be outstanding enough to improve alarm handling efficiency, it could only offer a 1% reduction in alarm rate, while 99% of alarms were still left for manual processing.

Another similar piece of research has been conducted to look for anomalous alarm behaviours by using sequential alarm patterns (Alharby and Imai, 2005). The underlying thought of this study is slightly similar to the previous one using the KNN classifier; namely the alarm sequence generated by the system under attack will definitely deviate from the normal alarm pattern. By observing the frequent behaviours within an extended period of time, a normal alarm pattern could be accurately formed. Therefore, through the use of this alarm model, a sudden burst of a sequence of alarms that has never been seen before could be alleged as a suspicious activity.

Given that the historical alarms pattern is used to learn the future alarms in a more efficient way by using the extraction of the sequential pattern, this approach has overcome some limitations of existing detection systems by constructing a more systematic model. Significantly, this method works by matching the extracted newly arrived sequence pattern with the extracted sequential pattern that is represented in the normal sequence patterns. The more matches found in this process, the higher the possibility of it being considered as normal behaviour.

Since this approach is using a threshold value as a measure to determine the class of alarm pattern, deciding the best value of threshold would always be a challenge for a security administrator. A high threshold value offers high security, but it might suffer from a high false alarm rate. Conversely, a lower threshold value solves the problem of false alarms, but might bring a lot of risks, principally causing an IDS to be unable to detect major attacks. The only optimal solution to answer this issue is by setting a security policy, which is always a trade-off between security and the reduction rate. Apart from this limitation, in terms of scalability, this method (the reduction algorithm) shows a good performance in handling such a large volume of data (alarms). Importantly, the aspect of confidentiality is also considered in this model, as there is no prior knowledge about the users, i.e. users' features (source user id, target user id) are required. Lastly, it is also worth noting that this approach is completely independent from the detection function, which means that it could be applied to almost any existing IDS technology.

Besides using a data mining technique to reduce the number of false alarms produced by IDSs, there are still a lot approaches that have been proposed by research thus far. One of the most prominent approaches being presented, which has proved to effectively improve the alarm handling efficiency (especially in false positive rate), is by using co-stimulation mechanism, based on the definition of intrusion and inspiration of immune mechanism (Qiao and Weixin, 2002). This research has been done by building a new network IDS, which is capable of integrating the misuse detection technique with the anomaly detection technique. Basically, the principal concept of this work is the application of the biological immune mechanism into IDSs.

This new network intrusion detection system, which is known as Artificial Immunological Network Intrusion Detection System (*AINIDS*), consists of two main components: the detectors and monitor agents. As in biological immune mechanisms, the monitor agents works by supplying or sending a signal indicating the damage of the system according to the integrity, confidentiality or availability of the system resources. If there is an anomaly case being reported by three agents, a co-stimulation will be sent to the detector, and at the same time a report will also be sent to the system administrator for further action taken; otherwise the activation will be considered as false positive.

Unlike other IDS which constantly monitor the system, this system triggers the monitor agents only when a detector has been activated (several signs of anomalous activity have been identified). Instead of depending upon a system administrator's experience in responding to potential intrusion, this system provides a more objective mechanism with better autonomy in controlling the signal.

Since false alerts have always been a primary issue of current IDS technology, providing alert classification might be a valuable approach in enhancing its performance. A novel system utilising machine learning technique has been proposed to reduce false positive in intrusion detection by correctly identifying true positive (i.e. alerts related to attack) and false positive (Pietraszek, 2004). This new system is known as Adaptive Learner for Alert Classification (ALAC). By building an alert classifier using a machine learning technique, this method works by classifying the alerts and sending the classification to the intrusion detection analyst for further feedback. So, through getting a feedback from the analyst, the system will initially build and subsequently update the classifier, which is then used to classify new alerts in the future (as shown in Figure 1).

The existence of this new system using an adaptive learner does indicate a greater improvement in the area of intrusion detection system. It is worth noticing that this technique offers a great efficiency in term of its operation. ALAC can be set to process autonomously alerts that have been classified previously. For example, this system could remove any alerts that have been classified as false positive in high confidence. In this way the method could successfully trim down the workload presented to the security officer.

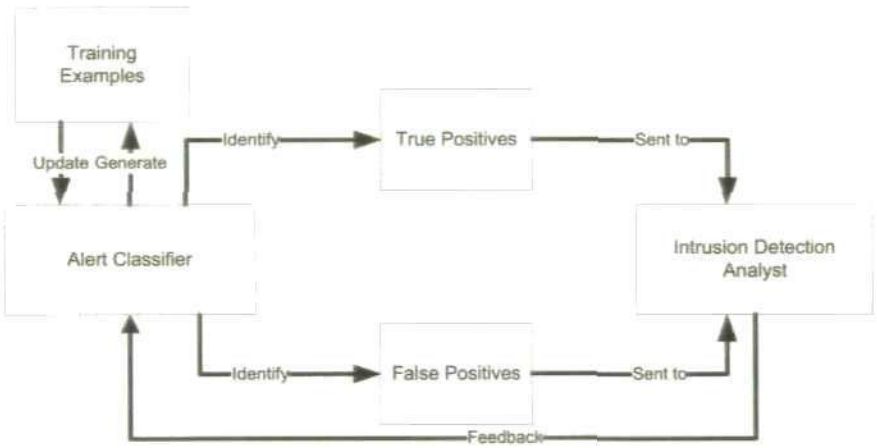


Figure 1: The framework model of ALAC classifier

However, apart from its strength, there are several limitations faced by this system. The ability of the analyst to correctly classify the alerts determines the accuracy or performance of this method. As no alert classification rules are written previously to respond to the alarm sequence generated, the analyst should be an expert in intrusion detection and able to initiate appropriate action to tackle the issue. Hence, this system seems to be inefficient in reducing the human workload. In addition, as the system is required to perform a real-time analysis, adapting to the new changes (new logic) as a new data arrives is its biggest challenge. Moreover, applying additional background knowledge (e.g. network topology, alert database) can become another challenge for the system in building an accurate alert classifier. Indeed, this idea will increase the complexity of learning tasks and only few machine learning techniques could support it. Having said that, from the research which has been done so far, the machine learning technique will produce a better or more concise rule if the background knowledge is used the basis for the classification.

4. Alert Correlation methods

In operation, the false alarm has always been a major factor determining the usability and efficiency of intrusion detection. So, in order to solve the problem of false alarms, simply identifying the false positives from a number of incoming alarms is no longer enough. A better approach is required to analyse the main causes of false alarms and to obtain a better understanding about intrusion behaviours from a set of alarms generated. For that reason, an alarm correlation might be required to describe the relationships and co-dependencies between alarms. Basically, alarm correlation is an important technique that is used to manage large volumes of alerts generated by heterogeneous IDSs. In other words, correlating alerts means combining the fragmented information contained in the alarm sequence and interpreting the whole flow of alarms. Importantly, the key objective of this mechanism is to pinpoint the triggering events from the incoming alarms and to help add meaning to the alarms generated. So, through the use of alarm correlation, it is expected that the number of alerts generated would be significantly reduced (e.g. by removing redundant alarms, filtering out low priority alarms, or even by replacing alarms by something else).

A number of research studies have been conducted to improve the performance of alarm correlation methods in reducing false alarms. Therefore, below are several prominent classes of methods being used for the alarm correlation technique.

- **Correlating alerts based on the prerequisites of intrusions**
  This class of approach is based on the assumption that most intrusions are not isolated, but are related to the different stages of attack sequences, with the early one prepared for the later one. It is believed that most traditional intrusion detection only focuses on low level attacks and raise alerts independently, without considering the possible logical connection between them or the potential attack strategies behind them. Another problem issue is that current IDSs technology cannot fully detect unknown attacks, or the variation of known attacks, without generating a large volume of alerts.

  Several works have been done to apply this class of approach (e.g. Cuppens and Miege, 2002; Ning et al., 2002), which then proposed to correlate alarms by using prerequisites and consequences of corresponding attacks (e.g. the existence of a vulnerable service can serve as the prerequisite for the remote buffer overflow attacks). Furthermore, this approach also provides an intuitive mechanism to represent attack scenarios constructed through alert correlation, known as hyper alert correlation. Even though this kind of

approach gives a better understanding about the intrusions' behaviours through the identification of logical connections between them, it has a major limitation: it cannot correlate unknown attacks (without attack patterns). Since the prerequisites and the consequences are required to build this correlation, only those known intrusions could be successfully identified in this approach (with the prerequisites and consequences having been previously defined).

- **Alert Correlation based on the similarity between alert features**
  This class of methods correlate alerts based on the similarities of selected features, for example source IP address, destination IP address or port number (Debar and Wespi, 2001). Alerts with a higher value of overall feature similarity will be correlated. Another research study has also been conducted in evaluating the use of a feature similarity function to fuse alerts that match closely but not perfectly (Valdes and Skinner, 2001). In this system, the similarity function is used to calculate the likeness of the features that match at least the minimum similarity specification, regardless of the match on the feature set as a whole. Once the alerts are considered having similar features, they will then be correlated using fusion algorithm. Although this method seems to effectively reduce a number of false alarms, it does suffer from one common weakness; it cannot fully discover the causal relationship between related alerts.

- **Alarm Correlation based on known attack scenario**
  This type of approaches correlates alerts based on the known attack scenario. One of the methods used to correlate alerts or fuse alerts into a scenario is by using a data mining technique (Dain and Cunningham, 2001). The data mining technique can be proposed to produce a real time algorithm to combine the alerts produced by heterogeneous intrusion detection system into a scenario. The main purpose of constructing these alert scenarios is to simply group alerts which share a common cause, thus providing a better view of the security issue to the system administrator. Moreover, this approach works well in reducing the number false alarms, since either individual alerts or the whole scenario could be labelled as false alarm possibility. Once a newly arrived alert is received, the probability of this new alert belonging to a specific scenario must be calculated. Significantly, such an approach could effectively uncover the causal relationship between alerts; however, it could not be applied to correlate alerts generated by unknown attack scenarios.

Generally, one of the most significant objectives of applying alert correlation techniques in intrusion detection is to provide a more succinct or high level view of security issues occurring in the protected network (i.e. the knowledge of occurring or attempted intrusions). It is worth remembering that the process of correlating alerts does not only involve a single or few components of procedure, it is a complete process involving various or a comprehensive set of components instead. For that reason, supplying an inclusive formalism and techniques of each component of alert correlation might prove a better result in effectively achieving alert reduction and abstraction.

A study has been done in generating a general correlation model that identifies a comprehensive set of components and a framework that analyses how each component contributes to the overall goal of the correlation (Valeur et al., 2004). As discussed above, a number of alert correlation methods have been introduced so far with the main goal of decreasing the false alarm rate. Unfortunately, not all of the published methods provide a detailed account of how these correlation components should be evaluated and implemented in a real life environment. Besides, current correlation methods only focus on few aspects of

alarm correlation components. For example, the identification and correlation of attacks into scenarios using prerequisites and consequences features do not provide enough detail on how the incoming alerts will be pre-processed before being correlated into scenarios. Due to this issue, providing a functional approach or method of alarm correlation mechanism is not enough; it should be followed by the procedure on how the complete set of components in alarm correlation analysis should work or be implemented in a real life environment. Significantly, the fundamental objective of presenting this comprehensive correlation process is to gain more understanding about the feature of the intrusions (e.g. the alerts generated, the target and source host of the attacks). Lastly, it should also be able to provide enough information about the impact of attacks as well as to assign an appropriate priority for each alert triggered by the events.

5. The application of Artificial Intelligence techniques in IDS alarm correlation methods

Artificial Intelligence (AI) techniques have become one of the most common methods being implemented in intrusion detection technology. Traditional intrusion detection has been previously developed and implemented by current enterprises. However, these systems have difficulty in successfully classifying the intruders, and require a large amount of computational overhead, which then makes it difficult to create robust real time IDS systems. Due to this issue, AI is playing an important role in reducing the human effort required to build these systems and can improve its overall performance.

Generally, there are several types of benefits offered by AI techniques which outperform other existing methods, namely flexibility, adaptability, pattern recognition, faster computing and learning abilities. In term of flexibility, AI techniques enable the system to easily adjust features such as the threshold value. AI also facilitates adaptation to new changes or rules if new data arrives or when the environment of the system has changed. One of the most specific and prominent functions of AI technique is its pattern recognition capability. This functionality is essential in detecting a new pattern of attacks, as no prior knowledge of attack behaviours is required. Moreover, self-learning is also another advantage of AI methods being studied in the intrusion detection research area. The ability of performing self-learning technique enables the system to effortlessly update to new changes (e.g. when a new rule or attack signature of new intrusion has been found).

Several studies have been undertaken to improve the performance of alert correlation mechanisms by using AI techniques. One of the significant studies being conducted in this field is the application of alert fusion to correlate alerts from multiple sensors in a distributed environment (Siraj and Vaughn, 2005). Alert fusion is a process of interpretation, combination and analysis of alerts to determine and provide a quantitative view of the status of the system being monitored. Importantly, this infrastructure consists of three essential components; namely alert prioritization, alert clustering and alert correlation. Thus, in order to fuse the alerts, a causal knowledge-based inference technique with Fuzzy Cognitive Modelling is implemented to find out the causal relationship in sensor data.

Given that alert fusion is a main component this model, the principal objective of this research is to gain an overall understanding or condensed view of the distributed system by assessing the integrity, confidentiality and availability of the system resources in the network. In this work, the Fuzzy Cognitive Map (FCM) is applied in this mechanism to represent our perception or understanding about the network situation or intrusion's behaviours in a more

structured way (e.g. by offering a structural representation of causal knowledge as well as the reasoning for causal analysis of data). Through the idea of using "concept" in this FCM, the relationships of events occurred in the system which generate a sequence of alarms could be described in a more systematic way. Fundamentally, "concept" is an event that originates from the system whose value change over time. The "concepts" typically shows the causality links between them; which then denote how much one "concept" affect the others.

Overall, Fuzzy Cognitive modelling offers a good representation of data that enables the human operator to learn and interpret the data much easier. Moreover, this technique also has advantage in describing an attack scenario for Distributed Denial of Service Attacks (DDoS) by using cause and effect type of the "concepts". Since this method uses cause and effect events to interpret the data, it has a capability in discovering the causal relationship of alerts; which then could lead to the identification of a series of attacks. However, in spite of the fact that this technique offers numerous advantages in correlating the alerts, this mechanism has one major limitation; namely its inability to deal with unknown alerts and mapping requirements of sensor alert features into more a generalised structure.

Most of the existing alert correlation techniques do not provide detailed information about the tactic or strategy of the intrusions, but simply cluster and correlate the alert into a specific class without further investigation of the issue. Another significant model of new alert correlation technique based on a neural network approach has also been proposed so far (Zhu and Ghorbani, 2006). Basically, this research is conducted to focus on the development of new alert correlation technique that can help to automatically extract attack strategies in a huge volume of generated alerts. This proposed alert correlation model is built by using two different neural network approaches; namely Multilayer Perceptron and Support Vector Machine.

One of the most distinctive features of this AI approach is the use of supervised learning technique for creating a function for training data. Once the function has been created, the system could calculate or determine the probability that these two alerts should be correlated. Moreover, in order to make it easier for the correlation engine to correlate the alerts and perform attack strategy analysis, an alert correlation matrix is introduced to define the alert strengths; which then determine whether the corresponding alerts should be correlated. Apart from looking into strength of the alerts in investigating the potential correlation of the alerts, feature selection has also played an important factor determining the probability of correlation. Such features include the timestamp, source IP address, target IP address, source port, destination port, as well as the type of attack.

In general, this proposed model offers tremendous benefits in operational terms. As the major objective of this work is to provide a better or more condensed view of the security situation to the network administrator through the extraction of attack strategy, this approach does outperform other models in offering automated construction of attack graph from a large volume of raw alerts. Unlike other approaches, which use pre-defined rules to correlate the alerts, this model does not require any prior knowledge to correlate the alerts, thus unknown alerts or attacks could be effectively detected. Despite the benefits offered, this approach has not been applied yet in a real-time environment. Correlation methods would be more useful if it could be implemented in a real-time environment, and could provide instant information about the attack strategy or attack patterns of intrusion occurring in the network environment.

6. Conclusions and future work

Even though IDS have been used for years and have demonstrated their worth in protecting organisation's resources, most still suffer from the problem of high false alarms rate and low detection rates. IDS systems are alleged to commonly trigger large volume of alarms, but most alarms are actually false. IDS technology could be fine-tuned as an attempt to reduce false alarm generation, but this may degrade the security level or even such action can be more risky, causing IDS unable to detect real attacks. Therefore, the tuning problem is always about searching for a balance of reducing false alarms while maintaining system security.

Alert correlation could serve as one of the most viable solution in handling the false alarm problem. Various studies have already been conducted in this area, either by using a more logical approach or more complex methods such as Artificial Intelligence techniques. Although a lot of current research has been done by introducing new alert correlation methods, all of these approaches have their own limitations. They either cannot discover the causal relationship among alerts, or they require a large number of pre-defined rules in correlating new alerts (inability in correlating unknown alerts or attacks). For that reason, a better correlation mechanism is required, which enables the system to detect unknown attacks as well as facilitating the security practitioners to learn and gain a better understanding about the attack strategy and the intention of the attackers. Thus, knowing a real security condition of the network and the strategies used by the attacker to launch the attacks would then enable the administrator to take a more appropriate action to stop the attacks and prevent them from worsening.

As AI techniques are deemed to be a powerful approach which could potentially ease human workloads, they can play a key role or act as a key concept in the research of intrusion detection. Hence, designing and developing a new approach using AI techniques for anomaly-based (based on the behaviour modelling) alarm correlation methods is the main idea of the author's ongoing research. Additionally, this research is also directed to improving the performance of alert correlation in providing a better quality of generated alarms and a reduction of false alarm rate. Correlation techniques will become more valuable if they can be designed to perform a real-life correlation and provide instantaneous information to the administrator once the attack has been detected. Furthermore, supplying the information about potential target of the attack can serve as a valuable source in designing an effective response plan, which aims to prevent the attack form re-occurring. Hence, developing a better approach, which focuses upon alarm reduction and enables the administrator to concentrate on more the important decisions, is undoubtedly valuable research.

7. References

Alharby, A. and Imai, H. (2005), 'IDS False alarm reduction using continuous and discontinuous patterns', *Lecture Notes in Computer Science* 3531. *Third International Conference on Applied Cryptography and Network Security, ACNS 2005, New York, United State.*

Allen, J., Christie, A. et al (2000), 'State of the Practice of Intrusion Detection Technologies', Technical Report CMU/SEI-99-TR-028, Carnegie Mellon University, available online: http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html, date visited: 9 January 2007

Bolzoni, D. and Etalle, S. (2006), 'APHRODITE: an Anomaly-based Architecture for False Positive Reduction', available from: http://arxiv.org/PS cache/cs/pdf/0604/0604026.pdf. date visited: 7 November 2006.

Bruneau, G. (2001), 'The History and Evolution of Intrusion Detection', available from: http://www.sans.org/reading room/whitepapers/detection/344.php. date visited: 9 October 2006.

Chapple, M. (2003), 'Evaluating and Tuning an Intrusion Detection System', available from: http://searchsecurity.techtarget.com/tip/1,289483,sid14 gci918619,00.html. date visited: 1 November 2006.

Cuff, A. (2003), 'Intrusion Detection Terminology (Part One)', available from: http://www.securityfocus.com/infocus/1728. date visited: 9 October 2006.

Cuppens F. and Miege A. (2002), ' Alert Correlation in a Cooperative Intrusion Detection Framework', *Proceedings of the 2002 IEEE Symposium on Security and Privacy'*, pp. 202.

Dain O. and Cunningham R. K. (2001), 'Fusing a heterogeneous alert stream into scenarios', *In Proc. of the 2001 ACM Workshop on Data Mining for Security Application, Philadelphia, PA*, pp. 1-13.

Debar H. and A. Wespi. (2001) 'Aggregation and Correlation of Intrusion-Detection Alerts', *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Davis, CA, USA*, pp. 85-103.

Goeldenitz, T. (2002), 'IDS - Today and Tomorrow', available from: http://www.sans.org/reading room/whitepapers/detection/351.php. date visited: 19 October 2006.

Julisch K. (2001), 'Mining Alarm Clusters to Improve Alarm Handling Efficiency', *Proceedings of the 17th Annual Conference on Computer Security Applications.* pp. 12-21.

Julisch K. and Dacier M. (2002), 'Mining Intrusion Detection Alarms for Actionable Knowledge', *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 266-375

Law, K. H. and Kwok, L. F. (2004), 'IDS false alarm Filtering using KNN classifer', *Lecture Notes in Computer Science* 3325. Fifth International Workshop on Information Security Applications, WISA 2004, Jeju Island, South Korea.

Lundin, E. and Jonsson, E. (2003), 'Some Practical and Fundamental Problems with Anomaly Detection', available from: www.ce.chalmers.se/»emilie/papers/Lundin_nordsec99.ps. date visited: 30 October 2006.

McHugh, J., Christie, A. and Allen, J. (2000), 'Defending Yourself: The Role of Intrusion Detection Systems', *IEEE Software* 17(5). available online: http://www.cert.org/archive/pdf/IEEE IDS.pdf, date visited: 5 October 2006.

Ning P. and Cui Y. and Reeves D. S. (2002), 'Constructing Attack Scenarios through Correlation of Intrusion Alerts', *In Proceedings of the 9th ACM Conference on Computer and Communications Security Washington, D.C.,* pp. 245-254.

Pietraszek, T. (2004), 'Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection' *Congres RAID '04:Proc. 7th Symposium on Recent Advances in Intrusion Detection* 3224 pp. 102–124

Qiao, Y., Weixin, X. (2002), 'A Network IDS with Low False Positive Rate', *CEC '02. Proc. IEEE Congress on Evolutionary Computation, IEEE Computer Society Press*, pp. 1121–1126

Siraj, A., Vaughn, R. (2005), 'A Cognitive Model for Alert Correlation in a Distributed Environment', *Lecture Notes in Computer Science 3495*, pp. 218-230.

Valdes, A. and Skinner, K. (2001), 'Probabilistic Alert Correlation', *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Davis, CA, USA*, pp. 54-68.

Valeur F. and Vigna G. and Kruegel C. and Kemmerer R. A. (2004), 'A comprehensive approach to intrusion detection alert correlation', *IEEE Transactions On Dependable and Secure Computing 1(3).* pp. 146-169, available online: http://www.auto.tuwien.ac.at/~chris/research/doc/tdsc04_correlation.pdf

Zhu, B., Ghorbani, A. (2006), 'Alert Correlation for Extracting Attack Strategies', *International Journal of Network Security 3(2)*, pp. 244-258.

The problem of false alarms: Evaluation with Snort and DARPA 1999 Dataset

Gina C. Tjhai[1], Maria Papadaki[1], Steven M. Furnell[1,2], and Nathan L. Clarke[1,2]

[1] Centre for Information Security & Network Research, University of Plymouth, Plymouth, United Kingdom
cisnr@plymouth.ac.uk
[2] School of Computer and Information Science, Edith Cowan University, Perth, Western Australia

Abstract. It is a common issue that an Intrusion Detection System (IDS) might generate thousand of alerts per day. The problem has got worse by the fact that IT infrastructure have become larger and more complicated, the number of generated alarms that need to be reviewed can escalate rapidly, making the task very difficult to manage. Moreover, a significant problem facing current IDS technology now is the high level of false alarms. The main purpose of this paper is to investigate the extent of false alarms problem in Snort, using the 1999 DARPA IDS evaluation dataset. A thorough investigation has been carried out to assess the accuracy of alerts generated by Snort IDS. Significantly, this experiment has revealed an unexpected result; with 69% of total generated alerts are considered to be false alarms.

Key words: Intrusion Detection System, False positive, True positive, DARPA dataset, Snort

1 Introduction

The issue of false positives has become the major limiting factor for the performance of an IDS [5]. The generation of erroneous alerts is the Archilles' heel of IDS technology, which could render the IDS inefficient in detecting attacks. It is also estimated that a traditional IDS could possibly trigger 99% of fake alarms from total alerts generated [10]. Recognising the real alarms from the large volume of false alarms can be a complicated and time-consuming task. Therefore, prior to addressing the issue of false alarm, a quantitative evaluation is required to assess the extent of the false positive problem faced by current IDS.

A number of research or efforts have been conducted to evaluate the performance of IDS in terms of its detection rate and false positive rate. One of the most well-known and determined IDS assessments to date was undertaken by Defense Advanced Research Projects Agency (DARPA) IDS evaluation [12]. This quantitative evaluation was performed by building a small network (test bed), which aimed to generate live background traffic similar to that on a government site connected to the Internet. The generated data set, which included a number of injected attacks at well defined points, were presented as tcpdump data, Basic Security Model (BSM), Windows NT audit data, process and file system information. The data were then used to evaluate the detection performance of signature-based as well as anomaly-based IDSs [14].

Although this data set appears to be one of the most preferred evaluation data sets used in IDS research and had addressed some of the concerns raised in the IDS research community, it received in-depth criticisms on how this data had been collected.

The degree to which the stimulated background traffic is representative of real traffic is questionable, especially when it deals with the reservation about the value of the assessment made to explore the problem of the false alarm rate in real network traffic [16]. Significantly, Mahoney and Chan [15] also critically discuss how this data can be further used to evaluate the performance of network anomaly detector. Although the DARPA IDS evaluation dataset can help to evaluate the detection (true positive) performance on a network, it is doubtful whether it can be used to evaluate false positive performance. In fact, the time span between the dataset creation and its application to the current research has resulted in another reservation about the degree to which the data is representative of modern traffic. However, despite all of these criticisms, the dataset still remains of interest and appears to be the largest publicly available benchmark for IDS researchers [16]. Moreover, it is also significant that an assessment of the DARPA dataset is carried out to further investigate the potential false alarms generated from this synthetic network traffic. It is expected that the result of this analysis could describe or provide a general picture of the false alert issue faced by the existing IDSs.

The main objective of the experiment described in this paper is to explore the issue of false alarm generation on the synthesized 1999 DARPA evaluation dataset. An investigation is also conducted to critically scrutinize the impact of false alarms on the IDS detection rate. Section 2 presents a number of related studies carried out to evaluate the performance of IDS. Section 3 discusses the methodology of the experiment. The findings are presented in section 4 and lastly, followed by conclusions in section 5.

2 Related Works

As for IDS performance, a study has also been conducted to further assess the effectiveness of Snort's detection against 1998 DARPA dataset evaluation [8]. Snort is an open source and signature-based IDS [9]. It is a lightweight IDS which can be easily deployed and configured by system administrators who need to implement a specific security solution in a short amount of time [17]. In other words, Snort is a flexible programming tool which enables the users to write their own detection rules rather than a static IDS tool. The evaluation was performed to appraise the usefulness of DARPA as IDS evaluation dataset and the effectiveness of the Snort ruleset against the dataset. Surprisingly, the result showed that Snort's detection performance was very low and the system produced an unacceptably high rate of false positives, which rose above the 50% ROC's guess line rate. Unfortunately, no further explanation was given to describe the nature of false alarms.

Interestingly, a paper by Kayacik and Zincir-Heywood [11] discussed the benefit of implementing intrusion detection systems working together with a firewall. The paper had demonstrated a benchmark evaluation of three security management tools (Snort, Pakemon and Cisco IOS firewall). Significantly, the result showed that none of the tools could detect all the attacks. In fact, Snort IDS was found to have produced 99% of false alarm rate, the highest rate compared to the other IDS (Pakemon). The result had also revealed that Cisco IOS had performed well and raised only 68% of false alarm rate. This has suggested the implementation of a firewall-based detection, which in turn decreases the attack traffic being passed to the IDSs.

Apart from the two studies above, which focused upon Snort performance, there are a large number of studies that have used the 1998 and 1999 DARPA dataset to evaluate the performance of IDSs. One of those studies is that of Lippmann et al [13], which managed to demonstrate the need for developing techniques to find new attacks instead

of extending existing rule-based approach. The result of the evaluation demonstrated that current research systems can reliably detect many existing attacks with low false alarm rate as long as examples of these attacks are available for training. In actual fact, the research systems missed many dangerous new attacks when the attack mechanisms differ from the old attacks. Interestingly, a similar paper had also been written by Lippmann et al [14], focusing upon the performance of different IDS types, such as host-based, anomaly-based and forensic-based in detecting novel and stealthy attacks. The result of this analysis had proposed a number of practical approaches applied to improve the performance of the existing systems.

Alharby and Imai [2] had also utilised 1999 DARPA dataset to evaluate the performance of their proposed alarm reduction system. In order to obtain the normal alarm model, alarm sequence is collected by processing the alerts generated by Snort from the first and third weeks (free-attacks traffic) of DARPA 1999 dataset. From these alarm sequences, the sequential patterns are then extracted to filter and reduce the false alarms. The same dataset (using the first and third weeks of the 1999 DARPA dataset) had also been applied by Bolzoni and Etalle [7] to train and evaluate the performance of the proposed false positive reduction system. Similarly, Alshammari et al [3] had also used such data to experiment their neural network based alarm reduction system with the different background knowledge set. The final result has proved that the proposed technique has significantly reduced the number of false alarms without requiring much background knowledge sets.

Unlike other papers discussed above, our experiment focuses specifically upon the issue of false alarms, rather than the performance of IDS (true alarms) in general. In this study, we propose to investigate in a more detailed manner some of the shortcomings that caused the generation of false alarms.

3 Experiment Description

Given that the 1999 DARPA dataset is deemed to be the largest publicly available benchmark, our experiment was designed to utilize such data as the source of our investigation. The experiment was run under Linux Fedora 7, and Snort version 2.6 was chosen as the main detector. The reason for utilising Snort was due to its openness and public availability. The Snort ruleset deployed in this evaluation is VRT Certified Rules for Snort v2.6 registered user release (released on 14 May 2007). In order to facilitate the analysis of IDS alerts, a front-end tool Basic Analysis and Security Engine (BASE) was utilized as the intrusion analyst console [6].

The primary data source of this evaluation was collected from DARPA evaluation dataset 1999. Without training the Snort IDS with the three weeks training data provided for DARPA off-line evaluation beforehand, two weeks testing data (fourth and fifth week of test data) were downloaded and tested. Snort ran in its default configuration, with all signatures enabled.

The first stage of the experiment was to run Snort in NIDS mode against the DARPA dataset. The manual validation and analysis of alerts produced by Snort were undertaken by matching against the Detection and Identification Scoring Truth. The Detection Scoring Truth is comprised of a list of all attack instances in the 1999 test data, while Identification Scoring Truth consists of alert entries of all attack instances in the test data [12]. A match is identified as same source or destination IP address, port numbers and their protocol type. In this case, timestamp does not really help identifying the true alerts since the attacks were labeled by the time the malicious activities set off while Snort spotted them when malevolent packets occurred. This might render the

system missing numerous matches. Hence, by recognizing the matches for those attack instances, the number of false positives alarms will then be identified.

Once the alerts were manually verified and the false positives were isolated, the results were presented in several diagrams to give a clear picture on the issue of false alarms. Individual Snort rules were examined to further analyse the false alarms issue and the impact of false alarms on IDS detection rate.

4 Results

Our earlier evaluation [21], which was conducted to focus on the issue of false alarms in real network traffic, asserted that the problem remains critical for current detection systems. Hence, this experiment was carried out to endorse our previous findings by highlighting the issue of the false alarm rate on the DARPA dataset.

Snort has generated a total of 91,671 alerts, triggered by 115 signature rules, in this experiment. Of the alerts generated from this dataset, around 63,000 (69%) were false positives. Significantly, this experiment had revealed a similar result to that yielded in our previous evaluation as well as Kayacik and Zincir-Heywood [11]. The false alarms have significantly outnumbered the true alarms.

To obtain a more in-depth understanding of the nature of Snort's alert generation, Figure 1 portrays a ROC plot [4] for the overall result, which illustrates the overall alert generation of Snort's signature rule. Since most plots have the value of X-axis and Y-axis less than 2000, Figure 2 depicts a clearer picture by focusing upon the area in coordinate 0 to 2000. The number of false positives generated is presented per signature for the X-scale, while the true positive is portrayed for the Y-scale. This diagram also describes the random guess line (non-discriminatory line), which gives a point along a diagonal line from the left bottom (0, 0) to the top right corner (10, 10). This diagonal line divides the space into two domains; namely good and bad figures. Ideally, a good detection system should generate a zero value for the X-scale; meaning no false positive has been generated by a particular signature rule. The area below the line represents a higher number of false positives than true positives. Thus, the more plots scattered on this area, the poorer the IDS is.

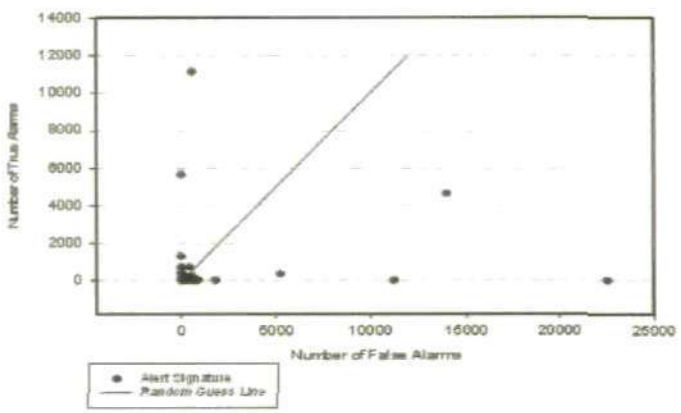


Fig. 1. Overall Alert Generation per Signature

As the plot diagram can only give an overview of IDS alert generation, Figure 3 provides the exact figures of Snort's signatures generating the false and true positive alerts in a Venn diagram [18]. Surprisingly, 73 signatures had raised the false positive alarms; of which 26 of them had triggered both true and false positives. It is also worth noticing that of those 26 rules, 14 signatures had false positives outnumbering the true positives. This seems to be a very critical issue faced by contemporary IDSs. The following subsections discuss this issue in greater detail.

4.1 True Positive

Given that the objective of this experiment is to investigate the issue of IDS false alarms, evaluating Snort's detection performance on DARPA dataset is beyond the scope of this study. In this paper, therefore, we will not further evaluate the extent of Snort's detection performance on a particular attack in a greater detail. However, this subsection presents a brief overview of the Snort detection performance on 4 attack categories, namely probe, Denial of Services (DoS), Remote to Local (R2L) and User to Root (U2R).

In this experiment, 42 of the total 115 signatures had generated pure true positives. Approximately only 31% (27,982 alerts) of total alerts generated by 68 signatures were asserted as true positives. Interestingly, about 72% of them were generated due to the probing activities.

Generally, Snort fares well in detecting probe attacks, which largely generate noisy connections. In this study, we found that Snort has a very low threshold for detecting probing activity; for example in detecting ICMP connections. This had made up of 40% (37,322 alerts) of the total alerts. In spite of its sensitivity, Snort had generated a small number of true ICMP alarms in this experiment, which accounted for only 13% of those 37,322 alerts. This significantly highlights the underlying flaw of Snort IDS alarms.

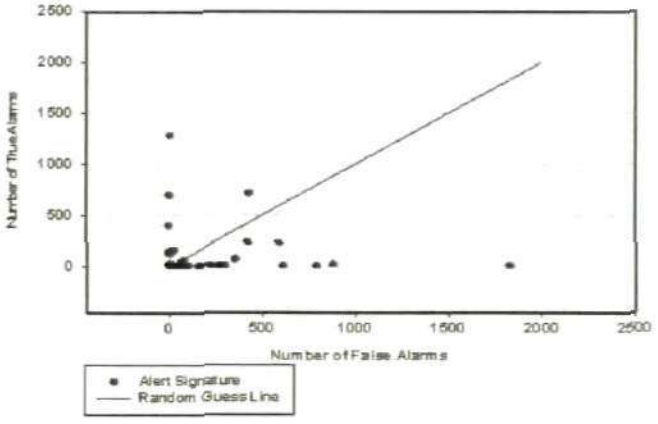


Fig. 2. Alert Generation per Signature within Cartesian Coordinate (2000, 2000)

In term of the DoS attacks, Snort did not perform well. Only one attack, named Back [12], was detected without generating any false positives. This had contributed

to 20% of total true alarms. As for remote to local (R2L) attacks, about 16 out of 20 types of attacks had been detected. This, however, only made up of 2% of true alarms. *Although Snort fares well in this category, it had critically missed numerous attack* instances, such as "ppmacro" attack [12].

The last attack category, user to root (U2R), is the most challenging attack for Snort IDS. Since U2R attack typically occurs on a local machine, which attempts to elevate administrator's privileges, it relies solely on a system log or computer's filesystem. As such, Snort, a network-based IDS that merely depends on network connections, does not work well in detecting such attacks. Indeed, only a small proportion of true alerts (less than 1%) were generated owing to this category.

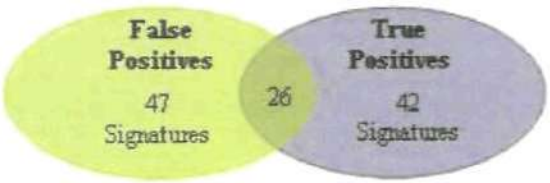


Fig. 3. Snort IDS Alarms - True and False Positives

Overall, the experiment has yielded a similar result as the one revealed by Brugger and Chow [8]. Snort's performance does not look particularly impressive. Although there were quite a significant number of true alarms (27,982 alerts), only 32 from 54 types of attacks were detected. In fact, not all instances from those 32 attacks were perfectly detected by Snort. This emphasises the fact that Snort was not designed to detect all types of attacks, but rather to work in conjunction with other IDSs to achieve the best detection performance.

4.2 False Positive

Approximately, 69% of total alarms are affirmed to be false positives. Figure 4 shows the top 5 false alarms raised by Snort. Interestingly, 48% of the total false alarms were made up of ICMP alerts. Logging every connection associated with probing, for example all ping activities, will only generate a huge number of false positives. In fact, all detected ICMP traffic did not surely imply the occurrence of probing actions, but it was merely an informational event, which possibly indicates the occurrence of network outage.

In term of the alert categories, 39% (24,835 alerts) of the total false alerts were triggered due to a policy violation. Significantly, these types of alerts are more related to irrelevant positives than false positives. Irrelevant positives refer to the alerts generated from unsuccessful attempts or unrelated vulnerability, which do not require immediate actions from the administrators. However, as those informational alerts were not related to any suspicious activity from DARPA attack database and in order to make it simpler, they will be referred to as false positives.

The highest number of false alarms in this experiment was triggered by INFO web bug 1x1 gif attempt signature. This signature rule was raised when the privacy policy violation was detected [20]. Theoretically, the web bug is a graphic on the web page and email message, which is used to monitor users' behaviours. This is often invisible

(typically only 1x1 pixel in size) and hidden to conceal the fact that the surveillance is taking place [19]. In fact, it is also possible to place web bug in a Word document as it allows html in a document or images to be downloaded from the external server. This is particularly useful if the document is supposed to be kept private, and web bug provides the information if the document had leaked by finding out how many IP addresses had looked at it [1]. Since none of these web bug alerts related to any attack instances, our study reveals that no true alarms associated with this signature had been generated. Therefore, 22,559 alerts from this signature were entirely asserted as false positives. This contributed to 35% of the total false alarms raised by the system. Although both ICMP and web-bug alerts can be easily filtered by the administrator through disabling or altering the signature rules, simply tuning the Snort signatures could increase the risk of missing real attacks.

Another similar policy-related alarm logged in this experiment is CHAT IRC alerts. These alerts accounted for 3.6% (2,276 alerts) of the total false alarms. Snort generates these IRC alerts because the network chat clients have been detected. In common with the previous "web bug" signature, IRC alerts were not truly false positives. Principally, Snort, given the correct rule, fares well in detecting policy violation. Indeed, through the investigation of the DARPA packet payload, it was noticeable that the chat activity did take place on a certain time. However, since these alerts did not contribute to any attack instances in the attack list, we would assume these as false positives. These CHAT IRC alerts were triggered by 3 signature rules; namely CHAT IRC message, CHAT IRC nick change and CHAT IRC channel join.

Apart from those top 5 false alarms signatures shown in Figure 4, there were 68 other signatures that generated false alarms. Of the total 115 signatures, 47 of them had triggered one hundred per cent false positives. All these alerts are known as pure false positive alarms since they are not in common with any true alarms. Significantly, 25 of those 47 signatures were web-related signatures. Although port 80 was one of the most vulnerable ports for DARPA attacks, these signatures did not correspond to any attack instances listed in the attack database. The effectiveness of Snort rules in detecting web-related attacks largely hinges on the effectiveness of keyword spotting. Most of the rules looking for web-related attacks are loosely written and merely checked on the presence of a particular string in the packet payload. This renders the system prone generating a superfluous number of false alerts. Aside from the web-related alerts, other 22 signatures, involving ICMP informational rule, policy, preprocessors, exploit attempt and SQL rules, had also generated a significant number of false positives, which accounted for 44% (28340 alerts) of total false alarms raised by the system.

Despite the informational and policy-related alerts, the pure false positives could also be generated due to the loosely written rules of Snort IDS. For example, the vulnerability of Snort in relying on the keyword spotting is intolerable. This has been further discussed in Tjhai et al [21].

As described earlier, exact 14 signatures has produced more false positives than true positives. This highlights the critical issue of false alarms in the real world. The process of identifying the real attacks could be undermined if the false positives per signature highly outnumbered the true positives. In addition, this could render the administrator apathetic; thus tending to conclude that any such alerts as false positives. As a consequence, this problem could seriously inhibit IDS detection performance in a real environment.

While Snort could detect 32 types of attacks, it had produced a large volume of unnecessary alerts; in term of its alerts' quality. One of the good examples can be taken from this experiment is the alerts triggered due to the detection of "Back" DoS attack,

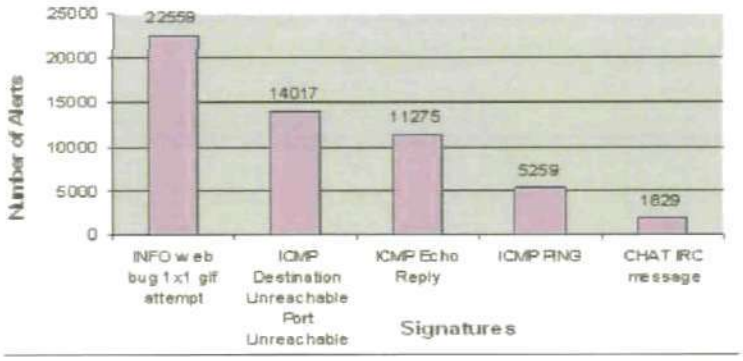


Fig. 4. Top 5 False Alarms

by WEB-MISC apache directory disclosure attempt signature. Only 7 instances from this attack were included into the DARPA dataset, but surprisingly Snort detected all 7 instances by triggering 5,628 alerts from single signature. Obviously, Snort has generated a huge number of redundant alerts in this case. Indeed, this often leaves the administrator with the difficulty of verifying every single alert.

In term of the false positives, the experiment revealed a slightly different result as generated by Brugger and Chow [8]. A smaller number of false alarms, accounted for 11% of total alerts, had been estimated by Brugger and Chow, compared to our result (69% of total alerts). The insignificant number of false alarms was presumably due to the removal of "web-bugs" rule that had generated a very significant number of false alarms. This signature was believed to not provide any true positives, and could potentially prevent an objective evaluation of false positives produced by an IDS. As *for Snort rules, only 36 signatures were triggered in their study. However, the "ICMP* Destination Unreachable Port Unreachable" signature had produced the second highest number of false alarms, similar to our result.

The experiment result has shown that Snort has produced an unacceptable number of false alarms. However, the evaluation of 18 IDSs on 1999 DARPA dataset, had yielded a remarkable result, indicating that most systems had false alarm rates which were low and well below 10 false alarms per day [14]. This might be due to their ability to tune the systems to reduce the false alarms on three weeks of training data prior to running the two weeks of test data.

5 Conclusions

Given the time span between the creation of DARPA dataset and the Snort rules, we initially thought that Snort could fare well in detecting DARPA attacks. What we found instead was that the detection performance was low; only 32 attacks were detected, and Snort has produced a large volume of false positives. Indeed, not all instances of those 32 attacks have been perfectly detected by Snort. From this experiment, it is obvious that the issue of false alarm has become very critical. The false positives outnumbered the true positive by a ratio of 2:1. In fact, more than half of the signatures producing both true and false positives in this evaluation have triggered more false positive than true positive alarms. This issue would critically reduce IDS detection performance; not only in this simulated network environment but also in a real environment.

Regarding the quality of alerts generated, Snort generated a huge number of redundant alerts, which critically highlighted the performance issue of the Snort alert reporting system. This often leaves the administrator with overwhelming alerts, which renders alert validation difficult to manage. Importantly, this issue has also driven the need to have an improved or better alarm reporting system through the implementation of alarm suppression and correlation methods.

Apart from total 39,849 false alerts triggered by 12 signatures generating both false and true alarms, Snort has also produced 28,340 pure false positive alarms, which can be arguably expected to happen in a real-network environment. Interestingly, this has accounted for 31% of alarms. However, in this experiment, we have not had a chance to individually track the cause of these alerts. Having said that, we believe that this might be caused by the nature of Snort IDS, which relies on keyword spotting (i.e. matching the packet content to signature rule) to detect malicious activity. Significantly, this finding underlines another weakness of Snort IDS, which could render the system prone to produce excessive alerts.

Overall, our study has confirmed the criticality of the IDS false alarm issue. Given the findings in this evaluation, endorsed by our previous experimental results, it is clear that false alarm is a never-ending issue faced by current IDS. The sensitivity of Snort rules in detecting probing activities can generate a large volume of false positives.

The ideal solutions to this problem is either to tune the IDS signature rules; this should be done by a skillful administrator who has the knowledge of security and knows well the environment of the protected network, or alternatively to focus upon the alarm correlation, which aims to improve the quality of the alerts generated. The idea of reducing false alarm in alarm correlation system has become the main subject of current IDS research. However, apart from the false alarm reduction, the alert management or alert correlation should also be aimed at the presentation of the IDS alerts itself to the system administrator. This might include the reduction of the redundant alerts and the aggregation of the related alerts (i.e. various alerts generated by a single attack).

References

1. Adoko (2008), 'What Are Web Bugs?', available online: http://www.adoko.com/webbugs.html, date visited: 7 September 2007.
2. Alharby, A. and Imai, H. (2005), 'IDS False alarm reduction using continuous and discontinuous patterns', Lecture Notes in Computer Science 3531.Third International Conference on Applied Cryptography and Network Security, ACNS 2005, New York, United State.
3. Alshammari, R., Sonamthiang, S., Teimouri, M. and Riordan, D. (2007), 'Using Neuro-Fuzzy Approach to Reduce False Positive Alerts', Communication Networks and Services Research, 2007. CNSR '07. Fifth Annual Conference on, pp. 345-349.
4. Anaesthetist (2007), 'The magnificent ROC', available online: http://www.anaesthetist.com/mnm/stats/roc/Findex.htm, date visited: 17 August 2007.
5. Axelsson, S. (2000), 'The Base-Rate Fallacy and the Difficulty of Intrusion Detection', ACM Transactions on Information and System Security 3(3), pp. 186-205, available online: http://www.scs.carleton.ca/ soma/id-2007w/readings/axelsson-base-rate.pdf, date visited: 10 May 2007.
6. BASE (2007), 'Basic Analysis and Security Engine (BASE) Project', available online: http://base.secureideas.net/, date visited: 25 April 2007.
7. Bolzoni, D. and Etalle, S. (2006), 'APHRODITE: an Anomaly-based Architecture for False Positive Reduction', available from: http://arxiv.org/PS cache/cs/pdf/0604/0604026.pdf. date visited: 7 November 2006.

8. Brugger, S. T. and Chow, J. (2005), 'An Assessment of the DARPA IDS Evaluation Dataset Using Snort', available online: http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-1.pdf, date visited: 2 May 2007.
9. Caswell, B. and M. Roesch (2004) Snort: The open source network intrusion detection system, available online: http://www.snort.org/, date visited: 3 October 2006.
10. Julisch K. (2001), 'Mining Alarm Clusters to Improve Alarm Handling Efficiency', Proceedings of the 17th Annual Conference on Computer Security Applications. pp. 12-21.
11. Kayacik, G. H. and Zincir-Heywood, A. N. (2003), 'Using Intrusion Detection Systems with a Firewall: Evaluation on DARPA 99 Dataset', NIMS Technical Report #062003, June 2003, available online: http://projects.cs.dal.ca/projectx/files/NIMS06-2003.pdf, date visited: 9 September 2007.
12. Lincoln Lab (2001), 'DARPA Intrusion Detection Evaluation', available online: http://www.ll.mit.edu/IST/ideval/, date visited: 15 May 2007.
13. Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham and R. K. Zissman, M. A (1999), 'Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation' In Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX), available online: http://www.ll.mit.edu/IST/ideval/pubs/2000/discex00_paper.pdf, date visited: 8 July 2007.
14. Lippmann, R. P., Haines, J. W., Fried, D. J., Korba, J., and Das, K. J. (2000), 'The 1999 DARPA off-line intrusion detection evaluation', Computer Networks 34, pp. 579-595, available online: http://ngi.ll.mit.edu/IST/ideval/pubs/2000/1999Eval-ComputerNetworks2000.pdf, date visited: 20 June 2007.
15. Mahoney, M. V. and P. K. Chan (2003), 'An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection'. In Recent Advances in Intrusion Detection (RAID2003), volume 2820 of Lecture Notes in Computer Science., Springer-Verlag, pp. 220-237, available online: http://cs.fit.edu/ mmahoney/paper7.pdf, date visited: 22 June 2007.
16. McHugh, J. (2000), 'Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory', ACM Trans. Information System Security 3(4), pp. 262-294, available online: http://www.cc.gatech.edu/ wenke/ids-readings/mchugh_ll_critique.pdf, date visited: 19 June 2007.
17. Roesch, M. (1999), 'Snort - Lightweight Intrusion Detection for Networks', Proceedings of LISA '99: 13th Systems Administration Conference, Seattle, Washington, USA, November 7-12, 1999.
18. Ruskey, F. and Weston, M. (2005), 'A Survey of Venn Diagrams', available online: http://www.combinatorics.org/Surveys/ds5/VennEJC.html, date visited: 10 October 2007.
19. Smith, R. (1999), 'The Web Bug FAQ', available online: http://w2.eff.org/Privacy/Marketing/web_bug.html, date visited: 15 August 2007.
20. Snort (2007), 'INFO web bug 1x1 gif attempt', available online: http://snort.org/pub-bin/sigs.cgi?sid=2925, date visited: 9 August 2007.
21. Tjhai, G., Papadaki, M., Furnell, S. and Clarke, N. (2008), 'Investigating the problem of IDS false alarms: An experimental study using Snort', IFIP SEC 2008, Milan, Italy, 8-10 September 2008.

Investigating the problem of IDS false alarms: An experimental study using Snort

G.C.Tjhai, M.Papadaki, S.M.Furnell, N.L.Clarke

Key words: Intrusion Detection System, False Alarm, Snort

1 Introduction

IDS can play a vital role in the overall security infrastructure, as one last defence against attacks after secure network architecture design, secure program design and firewalls [1]. Although IDS technology has become an essential part of corporate network architecture, the art of detecting intrusions is still far from perfect. A significant problem is that of false alarms, which correspond to legitimate activity that has been mistakenly classed as malicious by the IDS. Recognising the real alarms from the huge volume of alarms is a complicated and time-consuming task. Therefore, reducing false alarms is a serious problem in ensuring IDS efficiency and usability [2].

A common technique for reducing the false alarm rate is by performing a tuning procedure. This can be done by adapting the set of signatures to the specific environment and disabling the signatures that are not related to it [8], based on the fact that some vulnerabilities exist in a particular OS platform only. However, although this can offer a means of reducing the number of false alarms, the procedure can also increase the risk of missing noteworthy incidents. Therefore, the tuning process is actually a trade-off between reducing false alarms and maintaining the security level. This often leaves administrators with the difficulty of determining a proper balance between an ideal detection rate and the possibility of having false alarms. Furthermore, tuning requires a thorough examination of the environment by qualified IT personnel, and requires frequently updating to keep up with the flow of new vulnerabilities or threats discovered [26].

The authors are with the University of Plymouth, UK
e-mail: {gina.tjhai,maria.papadaki,s.furnell,n.clarke}@plymouth.ac.uk

This paper investigates the problem of false alarms based upon experiments involving the popular open source network IDS, Snort [7]. A number of potential issues are presented along with the analysis undertaken to evaluate the IDS performance on real network traffic. Section 2 critically reviews background information on the false alarm problem, and provides a critical analysis of existing research in the area. The methodology of the experiment is presented in section 3. Section 4 provides the findings from the private dataset, followed by conclusions in section 5.

2 Related work

The problem of false alarms has become a major concern in the use of IDS. The vast imbalance between actual and false alarms generated has undoubtedly undermined the performance of IDS [9]. For that reason, the main challenge of IDS development is now no longer focusing only upon its capability in correctly identifying real attacks but also on its ability to suppress the false alarms. This issue had been extensively explored and analysed by Axelsson [2] based on the base-rate fallacy phenomenon. At present, a solution to restrain the alarms is not close at hand, as numerous aspects (e.g. attack features) need to be considered as the prerequisites to develop a better alarm reduction technique [12]. Developing an alarms suppressing technique is a continuing process rather than an isolated, one-off action. The number of reported attacks (and the associated IDS signatures), increases each month, with the consequence that tuning becomes a requirement throughout the lifecycle of an IDS.

Similar to our research, an evaluation had been carried out by Brugger and Chow [4] to assess the performance of traditional IDS, Snort. This evaluation had been conducted using the baseline Defense Advanced Research Projects Agency (DARPA) dataset 1998 against a contemporary version of Snort. Although the use of DARPA dataset had been strongly criticised in IDS evaluation, it still serves as a benchmark by allowing the comparison of IDS tools with a common dataset [16]. This assessment was performed to appraise the usefulness of DARPA as an IDS evaluation dataset and the effectiveness of the Snort ruleset against the dataset. In order to analyse Snort's alarms, a perl matcher script was used to report the false negative and positive rates; thus generating the Receiver Operating Characteristic (ROC) curve for a given set of attacks. Given the six year time span between the ruleset and the creation of the dataset, it was expected that Snort could have effectively identified all attacks contained in the dataset. Conversely, what they found instead was the detection performance was very low and the system produced an unacceptably high rate of false positives, which rose above the 50% ROC's guess line rate. This might be due to the fact that Snort has a problem detecting attacks modelled by the DARPA dataset, which focused upon denial of service and probing activities [13]. In particular, Snort is alleged to commonly generate a high level of false alarms [17] and the alarm rate reported in this evaluation was not creditable enough to prove Snort's false positive performance in a real network, which

might be much worse or much better. Moreover, the other experiments took place a few years ago, which means that Snort's performance may have changed since then. In view of that, our research decided to assess the performance of Snort on a more realistic data, as an attempt to critically evaluate the false positive issue of the system.

3 Experiment Description

In order to further explore the problem of false alarms faced by current IDS technology, an experiment was conducted to analyse and evaluate IDS alerts generated by real network traffic. In common with the earlier research referenced in the previous section, Snort, was chosen as the main detector. The reason for utilising Snort was due to its openness and public availability. Moreover, an investigation involving such a commonly used IDS can give an insight into the extent of the false alarm problem in other IDS systems as well.

A number of criticisms had been raised over DARPA dataset, questioning the use of synthetic data to picture a real world network as well as the taxonomy used to categorise the exploits involved in the evaluation [15]. Owing to these issues, our experiments involved the evaluation of Snort on both DARPA [23] and private dataset. However, this paper only presents an experiment using a private dataset, which was collected at University of Plymouth. The data was collected on a public network (100-150 MB/s network) over a period of 40 days (starting from May 17th to June 25th), logging all traffic to and from the University's web server. This includes TCP (99.9%) and ICMP (0.1%) traffic. The traffic collection was conducted with a conventional network analysis tool, tcpdump, and it involved the collection of the full network packet, including the packet payload. Although storing the full packet information significantly increased the storage requirements for the experiment, it was important to maintain this information for the validation and analysis of IDS alarms. The collected payload data was then further processed by Snort IDS in Network Intrusion Detection (NIDS) mode. It should also be noted that traffic containing web pages with the potential of having sensitive / confidential information was excluded from the packet capture, in order to preserve the privacy of web users. This was accomplished by applying filters on the traffic, prior to it being captured by tcpdump. Ngrep was used for this purpose [18].

The first stage of the experiment was to run Snort in NIDS mode, in its default configuration. This means that no tuning whatsoever was conducted. The aim of this phase is to investigate the extent of the false alarm problem with Snort's default ruleset. The next phase of the experiment involved the analysis of the same traffic, after tuning had been performed on Snort. A number of techniques were applied for the tuning, including setting up the event thresholds and adjusting Snort's rules [19]. A necessary requirement for this was the manual validation and analysis of alerts produced by Snort in the first phase, and identification of signatures that are prone to false alarms. The analysis of IDS alerts was supervised by a certified intrusion

analyst, and the front-end tool Basic Analysis and Security Engine (BASE) was utilised to assist the intrusion analysis process [3].

The analysis of alerts was supervised by a GIAC Certified Intrusion Analyst [10]. Once the alerts were manually verified, the result was presented in a ROC diagram; a graphical plot of Snort alarm generation. In order to reveal a clear picture of the false alarm problem, a ROC plot is preferable. This type of graph can demonstrate the trade-off between the ability to identify correctly between true positives and the risk of raising too many false positives. Unfortunately, there were no true negative (number of benevolent activities passed) and false negative (number of real attacks missed) value known in this analysis since real network traffic was used as the input dataset. As an alternative, the plot diagram is drawn to represent the actual number of false and true alarms instead of their alarms rate. This diagram provides a simple graphical representation of the false alarm problem, thus enabling the analyzer to easily comprehend the trend of false alerts. By demonstrating the graphical plot of false positive versus true positive, this approach visibly explains the criticality of the false alarm issue. The alarm rate is calculated as follows:

$$\text{False Alarm Rate} = \frac{\text{False Alarm}}{\text{Total Alarm}} \times 100$$

$$\text{True Alarm Rate} = \frac{\text{True Alarm}}{\text{Total Alarm}} \times 100$$

4 Results

The lack of knowledge or awareness about the complexity of network by IDS technology has led to the generation of excessive amount of false alarms. Generally, there are three possible alert types raised by the system, namely true positives (alerts from real attacks), false positives (legitimate activities thought to be malicious) and irrelevant positive (alerts from unsuccessful attacks or attempts [12]. The last two alerts are the main concerns in this study.

This section presents the results of the experiment. Figure 1 depicts the ROC plot for the overall result, which represents the general detection performance of Snort IDS. In order to create a simpler illustrative graph, which facilitates the comprehension of Snort's detection ability, the false and true positives values are presented in a proportion of thousands. The number of false positives generated is presented per unit time (per day) for the X-scale, while true positives are portrayed for the Y-scale. This diagram also represents the random guess (known as non-discriminatory line), which gives a point along a diagonal line from the left bottom (0,0) to the top right corner (10,10). This diagonal line divides the space into two domains; namely good and bad classification. Ideally, a good detection system should yield a point above the line, meaning the number of real alerts (true positives) triggered should not be exceeded by the number of false positives generated.

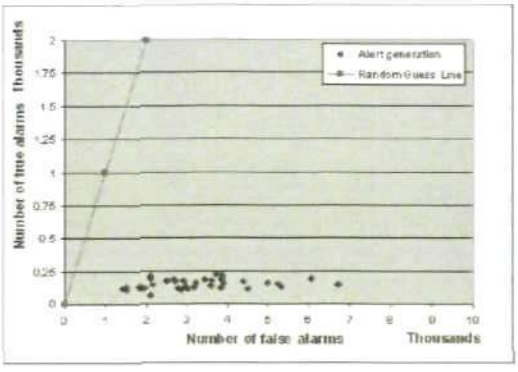


Fig. 1 Generation of alerts

Significantly, our research has also produced a similar result to that yielded in Brugger and Chow's evaluation. The number of false positives generated is massive. This indicates that the Snort's false positive performance on real network could be much worse than described in their evaluation.

This experiment focused on the analysis of false positive alarms, as opposed to other studies [14, 4], which were directed to explore the issue of false negatives. The main objective of this analysis is to merely provide a general view of the scale of false positives that may be generated by current IDS. The following subsections discuss this case in greater detail.

4.1 False Positives

A large volume of alerts, largely comprised of false alarms and irrelevant positives, drives the need to verify the validity of the alerts generated. Interestingly, apart from the false positives, our study reveals that some alerts were raised due to informational events, which merely occurred as a result of a network problem, not owing to the detection of real attacks. These types of alerts are known as irrelevant positives. Indeed, the unsuccessful attacks, or attempts that aim at an invincible target, might cause the system to generate such alarms.

Figure 2 provides a clear picture of the number of true and false alarms generated per day. In this context, it is obvious that the false alarms highly outnumbered the true alarms. Approximately 96% of alerts generated are asserted as false positives, while less than 1% of the total alerts are affirmed to be irrelevant positives. In order to make it simpler, irrelevant alarms are regarded as false positives alerts in this case since no immediate and crucial responses required from these events. By looking at the Snort alerts generated from the University's web server, most of the false positive alarms came from the category of web application activity. Table 1 shows

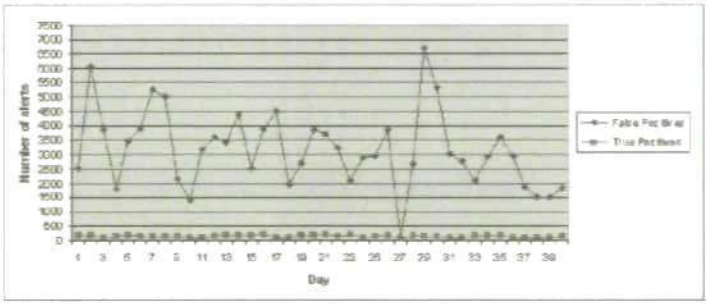


Fig. 2 Comparison between False Positive and True Positive alarms

a complete list of the Snort alerts triggered by the data. The first 3 alerts are the false positives alerts, which will be further investigated later in the subsubsections. The reason for focusing upon these alerts is due to the quantity generated, which is made up of more than 80% of total alerts raised by the system.

4.1.1 WEB-IIS view source via translate header

This event is categorized as web application activity, which targets the Microsoft IIS 5.0 source disclosure vulnerability [20]. Since Microsoft IIS has the capability of handling various advanced scriptable files such as ASP, ASA and HTR, the use of specialized header "Translate f" on HTTP GET request might force the web server to present the complete source code of the requested file to the client without being executed first by the scripting engine. In addition, this attack only works well if the trailing slash "/" is appended to the end of requested URL [5, 6].

Surprisingly, this single alert accounted for 59% of the total alerts. Therefore, approximately 1970 alerts were generated per day by this event. Although this event is deemed to be an attack that targets the Microsoft IIS source disclosure vulnerability, this could possibly be a false positive. Some applications, for example Web-based Distributed Authoring and Versioning (WebDAV) that make use of "Translate f" as a legitimate header, might cause this rule to generate an excessive amount of false alarms [25]. Moreover, WebDAV PROPFIND and OPTION methods also make use of this "Translate f" as a legitimate header to retrieve the information or properties of the resources identified by the Uniform Resource Identifier (URI) (nearly 96% of alerts generated by this event were not HTTP GET requests). Significantly, in this experiment, there is no alert generated by this signature, which required immediate action or indicated the occurrence of the real attack.

Table 1 Complete list of Snort alerts

No	Signatures	Total alerts
1	WEB-IIS view source via translate header	78865
2	WEB-MISC robots.txt access	30011
3	ICMP L3retriever Ping	10254
4	BARE BYTE UNICODE ENCODING	6392
5	POLICY Google Desktop activity	3258
6	SPYWARE-PUT Trackware funwebproducts mywebsearchtoolbar-funtools runtime detection	1873
7	ATTACK-RESPONSE 403 Forbidden	720
8	ICMP PING Cyberkit 2.2 Windows	651
9	DOUBLE DECODING ATTACK	504
10	ICMP Destination Unreachable Communication Administratively Prohibited	151
11	TCP Portsweep	124
12	SPYWARE-PUT Hijacker searchmiracle-elitebar runtime detection	80
13	WEB-MISC .DS_Store access	60
14	IIS UNICODE CODEPOINT ENCODING	49
15	WEBROOT DIRECTORY TRAVERSAL	35
16	SPYWARE-PUT Adware hotbar runtime detection - hotbar user-agent	27
17	WEB-IIS asp-dot attempt	26
18	TCP Portscan	19
19	SPYWARE-PUT Trackware alexa runtime detection	19
20	WEB-PHP IGeneric Free Shopping Cart page.php access	17
21	ICMP PING NMAP	17
22	ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited	13
23	WEB-CGI calendar access	11
24	MULTIMEDIA Quicktime User-Agent Access	10
25	WEB-MISC intranet access	8
26	ICMP redirect host	8
27	ICMP PING speedera	7
28	SPYWARE-PUT Hijacker marketscore runtime detection	7
29	WARNING: ICMP Original IP Fragmented and Offset Not 0!	6
30	WEB-MISC WebDAV search access	5
31	WEB-FRONTPAGE /_vti_bin/access	5
32	Open Port	5
33	WEB-PHP remote include path	4
34	WEB-CGI formmail access	3
35	WEB-FRONTPAGE_vti_inf.html access	3
36	SPYWARE-PUT Trickler teomasearchbar runtime detection	2
37	WEB-PHP xmlrpc.php post attempt	2
38	WEB-CLIENT Microsoft wmf metafile access	2
39	WEB-MISC Domino webadmin.nsf access	2
40	OVERSIZE CHUNK ENCODING	2
41	ICMP Source Quench	2
42	WEB-PHP test.php access	2
43	WEB-PHP calendar.php access	1
44	WEB-PHP admin.php access	1

4.1.2 WEB-MISC robots.txt access

This event is raised when an attempt has been made to access robots.txt file directly [21]. Basically, robots.txt file is a file that is created to keep the web pages from being indexed by search engines. More to the point, this file provides a specific instruction and determines which part of a website a spider robot may visit. Interestingly, the problem is that the webmaster may detail sensitive and hidden directories or even the location of the secret files within the robots.txt file. This is considered extremely unsafe since this file is located in web server's document root directory, which can be freely retrieved by anyone.

Although this event is raised as the indicator of vulnerable information attack, there exists high possibility that all these alerts were raised due to legitimate activities from web robots or spiders. A spider is software that gathers information for search engines by crawling around the web indexing web pages and links in those pages. Robots.txt file is basically created to restrict the web spider from indexing pages that should not be indexed (e.g. submission pages or enquiry pages). As web indexing is regular and structurally repetitive, this activity tends to cause the IDS to trigger a superfluous amount of alerts. In this study, approximately 23% of total alerts (approximately 750 alarms per day) were accounted for by this web-misc activity. Given that all alerts generated from this event are owing to the activities of web spider, they are considered to be false positives. Significantly, this issue has apparently disclosed the drawback of Snort IDS in distinguishing legitimate activity from the malicious one; especially when it deals with the authorization or file permission.

4.1.3 ICMP L3retriever Ping

ICMP L3retriever Ping is an event that occurs when ICMP echo request is made from a host running L3Retriever scanner [22]. This type of ICMP echo request has a unique payload in the message, which significantly designates its distinctive characteristic. This traffic is considered to be an attempted reconnaissance since the attackers may use the ping command to obtain ICMP echo reply from a listening host. Surprisingly, in this analysis, quite a few alerts were generated from this event; contributing to 8% of the total alerts generated. This figure indicates that approximately 250 alerts were generated by this signature rule every day.

Considering the source IP address associated with these alerts, it is obviously clear that all ICMP requests were sent from the external hosts. Further investigation was conducted to critically analyse and discover if possible malicious events happened subsequent to the ICMP echo request. Surprisingly, there were no malevolent activities detected following the ICMP traffic. In addition, normal ICMP requests generated by Windows 2000 and Windows XP are also known to have similar payloads to the one generated by L3Retriever scanner [24]. Generally, this traffic is routine activities run by computer systems (especially Windows 2000 and XP systems) to communicate with their domain controllers or to perform network discov-

ery. In view of this issue and given that no suspicious output detected following these ICMP requests; these alerts were likely false positives.

4.2 Fine Tuning

False alarm for one system might not be an erroneous alert for other systems. For example, port scanning might be a malicious activity for normal users, but it is a legitimate activity if it is performed by a system administrator. Figure 3 shows an example of an event which triggered both false alarms and true alarms from the experiment. From the IDS's perspective, as long the activity's pattern match to the signature defined in the rule database, it is considered to be a malicious event. In view of this, fine tuning is exceptionally required to maintain the IDS's performance and enable the administrator to adapt the signature rule to the protected environment.

In order to optimize Snort's performance, fine tuning is necessary to reduce the number of alerts raised. Since only 3 signatures were tuned in this experiment, the false alarm rate accounted for 86.8% of total alarms after tuning was performed. Figure 4 depicts the ROC plots for the overall result after tuning was performed. Obviously, only less than two thousands alerts per alert type have been generated by Snort. In order to understand the effectiveness of fine tuning, the alarm rate between default and tuned Snort is presented in Figure 5. This figure does not seem particularly impressive but fine tuning did fare well on those signatures; reducing up to 90% of false alarms per signature, excluding WEB-MISC robots.txt access. The following subsections discuss tuning processes in more details.

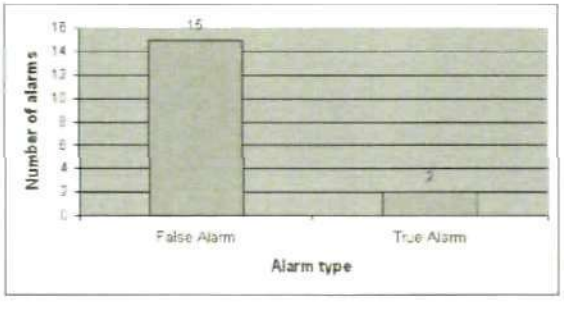


Fig. 3 "ICMP PING NMAP" event

4.2.1 WEB-IIS view source via translate header

Regarding the information disclosure vulnerability attack, Snort does not seem proficient enough to detect this type of event. The signature rule appears to be very loosely written, by searching for a particular string in the packet payload (in this case, "Translate: f"). Since the "Translate: f" is a valid header used in WebDAV application, as discussed previously, this rule tends to trigger a vast volume of alerts from the legitimate activities. Hence, tuning is needed to search for a more specific pattern of the attack signature.

As this attack is basically launched through HTTP GET request, searching for "GET" command in the content of analyzed packet can be a good start. Principally, this attack is performed by requesting a specific resource using HTTP GET command, followed by "Translate: f" as the header of HTTP request. In this case, a tuning can be performed by modifying the signature rule to:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS view source via translate header";
flow:to_server,established; content:"GET|20|";content:
"Translate|3A| F"; distance:0; nocase; reference:arachnids,
305; reference:bugtraq,14764; reference:bugtraq,1578;
reference:cve,2000-0778; reference:nessus,10491;
classtype:web-application-activity; sid:1042; rev:13;)
```

The tuning process significantly reduced the number of alerts, with only 3463 generated by this rule as against 78865 alerts in the first case (i.e. without tuning). Significantly, this tuned rule had been proved to effectively reduce up to 95% of the initial false alarms from this event.

Although the tuning process had decreased the volume of alerts, there is still a possibility that those 5% alerts were false positives. Searching for GET command and the Translate f header is not effective enough to detect such attack. Putting trailing slash "/" at the end of requested URL to HTTP request for example could lead in *the security bug [5]. Thus, matching the "/" pattern against the packet payload will* be helpful. Unfortunately, this idea seems hardly possible to achieve. Snort does not have a specific rule option that can be used to match a specific pattern at a particular location.

As to Snort's signature, looking for an overly specific pattern of a particular attack may effectively reduce the false alarms; however, this method can highly increase the risk of missing its range. A skilful attacker can easily alter and abuse the vulnerability in various ways as an attempt to evade the IDS. This might lead to false negatives as a consequence.

4.2.2 WEB-MISC robots.txt access

Since accessing the robots.txt file is a legitimate request for Internet bots (web spiders), a subjective rule, which mainly focuses on the source IP addresses, is necessary to verify user authorization in accessing a certain file. This approach, however, seems to be hardly feasible to deploy. Of course, identifying all authorized hosts from their source IP addresses is impractical. There is an infinite number of IP addresses need to be discovered before the rule can be written. Indeed, lawfully allowing specific hosts to access certain file might increase the risk of having false negatives.

In this case, the only solution to suppress the number of false alarms generated is by using event thresholding [19]. As robots.txt access requests generate regular and repetitive traffic, a "limi" type of threshold command is the most suitable tuning in this case. Such a threshold configuration would be as follows:

```
threshold gen\_id 1, sig\_id 1852, type limit,
track by\_src, count 1, seconds 60
```

This rule logs the first event every 60 seconds, and ignores events for the rest of the time interval. The result showed that approximately 10% of false alarms had been effectively reduced. This indicates that only an insignificant number of false alarms can be reduced in this scenario. The frequency of fetching robots.txt files greatly depends on the web spider's policy. Hence, deploying event suppression and thresholding cannot effectively trim down the number of false alarms logged by the system. Additionally, suppressing the number of alerts generated can also create a possibility of ignoring or missing significant alerts. A malicious user can hide

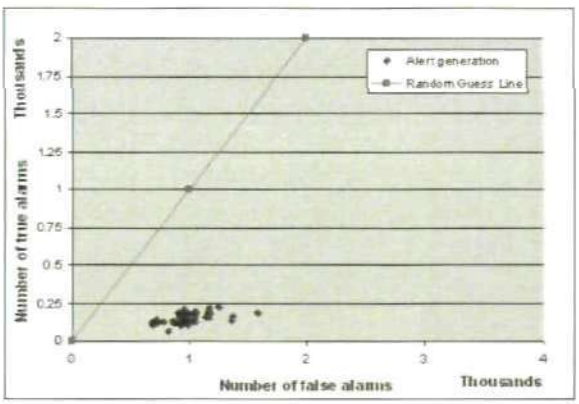


Fig. 4 Alerts generation after fine tuning

his/her action within the excessive number of alerts generated by using a spoofed address from web spider agent.

4.2.3 ICMP L3Retriever Ping

The only method that can be deployed to suppress the number of false positive triggered from this event is by applying event suppressing or thresholding command. Similar to the one applied to "WEB-MISC robots.txt access" signature, a threshold command is written to limit the number of alarms logged. Instead of using "limit" type of threshold command as previous signature, this rule utilized "both" type of command to log alerts once per time interval and ignore additional alerts generated during that period:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP
L3retriever Ping"; icode:0; itype:8; content:
"ABCDEFGHIJKLMNOPQRSTUVWABCDEFGHI"; depth:32; reference:
arachnids,311; classtype:attempted-recon; threshold: type
both, track by_src, count 3, seconds 60; sid:466; rev:5;)
```

Similar to the previous signature (robots.txt access), the threshold applied will not prevent the generation of false positives, but it will highly reduce the number of redundant false positives triggered. Importantly, the threshold is written to detect brisk ICMP echo requests by logging alerts once per 60 seconds after seeing 3 occurrences of this event.

The result showed that only 1143 alerts had been generated from this event in 40 days experiment data. This experiment has also proved that the event thresholding can successfully reduce up to 89% of the false alarms generated by this activity. Despite its ability in suppressing redundant alarms, the system is prone to missing stealthy ICMP requests (e.g. requests sent once every 60 seconds can be missed by the system).

5 Conclusions and Future Work

The issue of false positives has become a critical factor in determining the success of IDS technology. Not only must an IDS be accurate in detecting real attacks, but it must also have the ability to suppress the number of unnecessary alerts generated. The experiment presented in this paper has revealed a similar result to the work of Brugger and Chow [4]. Over a span of two years since their research was published, the issue of false positives remains a critical challenge for the current Snort IDS. Obviously, Snort's performance does not look particularly remarkable as illustrated

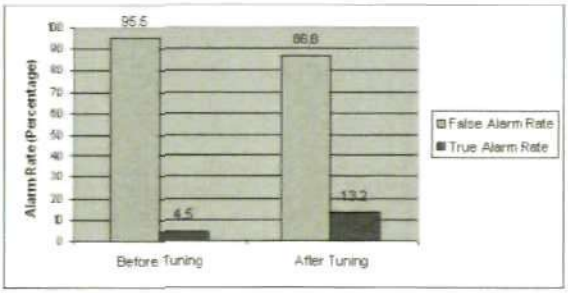


Fig. 5 Alarm rate before and after tuning

in Figure 1. The bottom right scattered plots demonstrate that the number of false positives largely overwhelms the number of true positives generated. Approximately 3,000 alerts had been generated per day, requiring manual verification to validate their legitimacy. Although the administrator can effectively distinguish the false and true positives from the alerts generated, the massive amount of false alarms triggered by one signature rule might cause the administrator to miss a malicious attack.

Principally, the overall effectiveness of Snort greatly hinges on the effectiveness of keyword spotting (i.e. matching the packet content to the signature rule). This has rendered the system prone to generating a superfluous number of false alerts. Interestingly, most of the rules looking for web traffic related attacks are loosely written and merely check for the presence of a particular string in the packet payload. This could trigger a large number of false alerts if a particular string is included in the content distributed by the web server. Hence, from this perspective, Snort is deemed not to be ideal enough to detect more complex attacks, which are not detectable by a pre-defined signature.

In view of these issues, an improvement is required to advance the performance of IDS technology. This involves developing an automatic alert verification, which no longer relies on human participation. Through this enhancement, it is expected that the number of false alarms can be substantially suppressed without increasing the possibility of false negatives. Also, a more intelligent system is required to help discover the logical relationship between alerts generated and to reveal the potential attack scenario; thus providing a better picture of the security issue to the system administrator. Given the complexity of systems and the ingenuity of attacks, an IDS will never be perfect, and there is still significant scope to enhance its performance.

Acknowledgements We want to thank Dr. Bogdan Ghita of University of Plymouth for his help in capturing the network traffic and for his support until the completion of this paper.

References

1. Allen J, Christie A, Fithen W, McHugh J, Pickel J, Stone E (2000) State of the Practice of Intrusion Detection Technologies. Available via Software Engineering Institute. http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html. Cited 9 January 2007
2. Axelsson S (2000) The Base-Rate Fallacy and the Difficulty of Intrusion Detection. ACM Transactions on Information and System Security 3(3), 186-205
3. BASE (2007) Basic Analysis and Security Engine (BASE) Project. Available via BASE Project. http://base.secureideas.net/. Cited 25 April 2007
4. Brugger ST, and Chow J (2005) An Assessment of the DARPA IDS Evaluation Dataset Using Snort. Available via UCDAVIS department of Computer Science. http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-1.pdf. Cited 2 May 2007
5. Bugtraq (2007a) Microsoft IIS 5.0 "Translate: f" Source Disclosure Vulnerability. Available via Security Focus. http://www.securityfocus.com/bid/1578. Cited 9 June 2007
6. Bugtraq (2007b) Microsoft IIS WebDAV HTTP Request Source Code Disclosure Vulnerability. Available via Security Focus. http://www.securityfocus.com/bid/14764. Cited 9 June 2007
7. Caswell B and Roesch M (2004) Snort: The open source network intrusion detection system. Available via Snort. http://www.snort.org/. Cited 3 October 2007
8. Chapple M (2003) Evaluating and Tuning an Intrusion Detection System. Available online: SearchSecurity.com. http://searchsecurity.techtarget.com. Cited 1 November 2006
9. Chyssler T, Burschka S, Semling M, Lingvall T and Burbeck K (2004) Alarm Reduction and Correlation in Intrusion Detection Systems. Available via The Department of Computer and Information Science Linkopings Universitet. http://www.ida.liu.se/ rtslab/publications/2004/Chyssler04_DIMVA.pdf. Cited 15 June 2007
10. GCIA (2008) GIAC Certified Intrusion Analyst (GCIA). Available via Global Information Assurance Certification. http://www.giac.org/certifications/security/gcia.php. Cited 8 May 2007
11. Koziol J (2003) *Intrusion Detection with Snort*, 2Rev edition. Sams Publishing, United States of America
12. Kruegel C and Robertson W (2004) Alert Verification: Determining the Success of Intrusion Attempts, Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004). Available via Department of Computer Science, University of California, Santa Barbara. http://www.cs.ucsb.edu/ wkr/publications/dimva04verification.pdf. Cited 19 May 2007
13. Lippmann RP, Haines JW, Fried DJ, Korba J and Das KJ (2000) The 1999 DARPA off-line intrusion detection evaluation. Computer Networks 34:579–595
14. Mahoney MV and Chan PK (2003) An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In Recent Advances in Intrusion Detection (RAID2003), Lecture Notes in Computer Science, Springer-Verlag 2820:220–237
15. McHugh J (2000) Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. ACM Transactions on Information and System Security 3(4), 262-294
16. Mell P, Hu V, Lippmann R, Haines J and Zissman M (2003) An Overview of Issues in Testing Intrusion Detection Systems. NISTIR 7007. Available via National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistir/nistir-7007.pdf. Cited 7 July 2007
17. Patton S, Yurcik W and Doss D (2001) An Archilles' Heel in Signature-Based IDS: Squealing False Positives in SNORT. Recent Advanced in Intrusion Detection (RAID), Univ. of California-Davis.

18. Ritter J (2006) Ngrep - network grep. Available via SourceForge.net. http://ngrep.sourceforge.net. Cited 30 June 2007
19. Snort (2007a) Event Thresholding. Available via Snort. http://www.snort.org/docs/snort_htmanuals/htmanual_2.4/node22.html. Cited 1 July 2007
20. Snort (2007b) WEB-IIS view source via translate header. Available via Snort. http://snort.org/pub-bin/sigs.cgi?sid=1042. Cited 9 June 2007
21. Snort (2007c) WEB-MISC robots.txt access. Available via Snort. http://www.snort.org/pub-bin/sigs.cgi?sid=1:1852. Cited 9 June 2007
22. Snort (2007d) ICMP L3retriever Ping. Available via Snort. http://www.snort.org/pub-bin/sigs.cgi?sid=1:466. Cited 13 June 2007
23. Tjhai GC, Papadaki M, Furnell SM and Clarke NL (2008) The problem of false alarms: Evaluation with Snort and DARPA 1999 Dataset. Submitted to TrustBus 2008, Turin, Italy, 1-5 September 2008
24. Web Server Talk (2005) L3Retriever false positives. Available via Web Server Talk. http://www.webservertalk.com/message893082.html. Cited 12 July 2007
25. WebDAV (2001) WebDAV Overview. Available via Samba Server Documentation. http://www.samba.com/syshelp/webdav.htm. Cited 20 June 2007
26. Zhou A, Blustein J, and Zincir-Heywood N (2004) Improving Intrusion Detection Systems Through Heuristic Evaluation. 17th Annual Canadian Conference on Electrical and Computer Engineering. http://users.cs.dal.ca/ jamie/pubs/PDF/Zhou+CCECE04.pdf. Cited 25 June 2007

COMPUTERS & SECURITY 29 (2010) 712–723

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm

Gina C. Tjhai, Steven M. Furnell, Maria Papadaki, Nathan L. Clarke*

Centre for Security, Communications and Network Research, University of Plymouth, Plymouth PL4 8AA, United Kingdom

ARTICLE INFO

Article history:
Received 21 July 2009
Received in revised form
5 February 2010
Accepted 25 February 2010

Keywords:
Intrusion Detection System
False alarm
Self Organising Map (SOM)
K-means clustering
Alarm correlation

ABSTRACT

Intrusion Detection Systems (IDSs) play a vital role in the overall security infrastructure. Although the IDS has become an essential part of corporate network infrastructure, the art of detecting intrusion is still far from perfect. A significant problem is that of false alarms, as generating a huge volume of such alarms could render the system inefficient. In this paper, we propose a new method to reduce the number of false alarms. We develop a two-stage classification system using a SOM neural network and K-means algorithm to correlate the related alerts and to further classify the alerts into classes of true and false alarms. Preliminary experiments show that our approach effectively reduces all superfluous and noisy alerts, which often contribute to more than 50% of false alarms generated by a common IDS.

1. Introduction

Networked systems have become increasingly prevalent, fast, and inexpensive, leading to a rapid growth in both demand and complexity of the computing system. Unfortunately, this has also been accompanied by a growth in the threats to the systems. In 2008, the number of new malicious code signatures increased over 265 percent over 2007; more than 60 percent of the total code threats were detected in 2008, as reported by Symantec (2009). The huge increase in the number of malicious code threats demonstrates the growing need for more responsive and reliable security measures.

An Intrusion Detection System (IDS) is a component of a network security architecture, which involves the monitoring of computer systems for intrusive activities (i.e. those behaviours that infringe the established security model). The rise of cybercrime on the global network has entailed a great demand for a remarkable use of IDS, which in turn forms the necessity of developing a better detection system. Although IDS has become an essential part of a corporate network infrastructure, the art of detecting intrusion is still far from perfect. IDS tends to generate a huge amount of alerts, which can be mixed with false alarms. False alarms, also known as false positives (Type I errors), occur when a legitimate activity has been mistakenly classified as malicious by the IDS. The vast imbalance between the actual and false alarms generated has undoubtedly undermined the performance of IDS (Chyssler et al., 2004b). An alarm reduction system is an absolute need for this problem.

This paper proposes a two-stages clustering system to reduce false alarm rate. The proposed method can classify the alarms generated by the IDS into false and true alarms. The main objective of this approach is to correlate the alerts into a more manageable form before they are presented to the administrator; and to reduce the sheer volume of alerts generated.

Section 2 provides a critical analysis of existing research in the area. The framework of the proposed system is presented in Section 3, whilst the concept and algorithm of the methodology is described in Section 4. Section 5 discusses the

* *Corresponding author.*
E-mail address: info@cscan.org (G.C. Tjhai).
0167-4048/$ – see front matter
doi:10.1016/j.cose.2010.02.001

performance of the proposed model in term of its alarm reduction rate, followed by conclusions in Section 6.

2. Related works

There have been a large number of IDS research efforts dealing with the alarm handling problem, but only few have considered applying alarm frequency rate when performing feature construction. Some techniques focus upon association-rule-based feature selection to classify the alarms, as proposed by Shin et al. (2004). The authors suggested a mining-based false alarm classification model that filters the false alarms by using alarm classification rules. This approach used decision tree algorithm (C4.5) (Quinlan, 1993), which was extended to association based classification, to predict class labels of unknown objects (alarms) with high accuracy. The effectiveness or accuracy of these classifiers largely depends on the training data set. Although this method had been empirically evaluated to work effectively in reducing the false alarm rate, it is considered not efficient enough since the classification rules should be created for each type of attacks. The model was implemented in the domain of DDOS (Distributed Denial of Service) attack only and no other attacks rules had been tested in their experiment.

Law and Kwok (2004) proposed to model the normal alarm patterns of IDSs and detect anomaly from incoming alarm streams using a K-Nearest-Neighbour classifier. In contrast, Alharby and Imai (2005) looked for anomalous alarm behaviour by using sequential alarm patterns. They believed that when an attack is occurring, the alerts triggered by the IDSs will have different patterns from that in an attack-free environment. The classification accuracy of both approaches relies upon the length of the time window for each alarm set. Since the alarm patterns are varied depending on the allocated time frame, the anomalous alarm pattern will share similarity with the normal pattern if the time allotment is amiss.

In terms of the alarm clustering, Julisch (2001) suggested a technique to efficiently handle large groups of redundant alerts by identifying and removing the root cause of an alarm. The author observed that over 90% of all alarms corresponded to a small number of root causes. The study reported that by knowing the root causes, the IDS can be regularly adjusted and root causes can be removed, reducing the false alarms by 82%. Unfortunately, as such method focused merely upon a large group of superfluous alarms, it was considered not effective enough in identifying false alarms in a small cluster. Julisch and Dacier (2002) had also developed a technique to mine historical IDS alarms for episode rules. The rules are created to predict a prospective alert when a specific set of alarms had been generated. Whilst this approach is deemed outstanding enough to give an insight into the pattern of false alarms and the potential future attacks, it could only offer 1% reduction in alarm rate, whilst 99% of alarms were still left for manual processing.

Other approaches correlate and classify the false alarms by using conceptual clustering techniques (focusing upon the alarm attributes). Cuppens and Miege (2002) pursued an attribute-based correlation function that clusters and merges the alarms by using prerequisites and consequences of attacks. Similarly, Ning et al. (2002) suggested an approach to construct attack scenarios by correlating alerts on the basis of prerequisites and consequences of intrusions. Both approaches provide an intuitive mechanism to represent attack scenarios constructed through alert correlation. However, neither of these can correlate unknown attacks since the prerequisites and consequences of the new attacks are not identified beforehand.

The likeness of alert features is considered effective to correlate and reduce the false alarms, as proposed by Debar and Wespi (2001) and Valdes and Skinner (2001). The authors conducted research to evaluate the use of a feature similarity function to fuse alerts that match closely but not perfectly. The similarity function is used to calculate the likeness of the features that match at least the minimum similarity specification, and correlate them using a fusion algorithm. Although this method seems to effectively reduce a number of false alarms, it does suffer from one common weakness; it cannot fully discover the causal relationship between related alerts. To solve this issue, Dain and Cunningham (2001) produced a real-time algorithm to combine the alerts into a scenario; thus effectively uncover the causal relationship between alerts. Unfortunately, such an approach cannot be applied to correlate alerts generated by unknown attack scenarios.

Unlike previous approaches, which used pre-defined rules to correlate the alerts, Zhu and Ghorbani (2006) proposed a correlation technique that cannot only correlate the generated alerts but also automatically extract attack strategies from a huge volume of intrusion alerts. The technique was developed based on the use of a neural network supervised learning approach. The correlation system was designed to automatically determine or calculate the probability that the alerts should be correlated by using an alert correlation matrix. The experiments conducted using the DARPA 2000 intrusion detection scenario specific data set demonstrated that this technique can successfully correlate a large number intrusion alerts into scenarios. Despite the benefits offered this approach had only been evaluated using the synthetic evaluation data set (DARPA 2000) and has not been applied yet in a real-time environment.

Data mining technologies have broadly evolved and have shown their capabilities to reduce more than a half of false alarms. Our approach presented in this paper will trim down the number of false alarms by implementing a data mining, unsupervised neural network technique.

3. Proposed technique

The idea of this false alarm classification system is to filter the false alarms from Intrusion Detection Systems and minimise the false alarm rate by unsupervisedly clustering the alerts based on the their attributes. For this purpose, the proposed system will apply data mining techniques using a neural network, i.e. SOM (Kohonen, 1995) and K-means (MacQueen, 1967), for the classification. The data mining techniques are commonly used in data reduction and data clustering. The reason of choosing these algorithms is because it is easy to implement and has the ability to show and clarify the relationship between the classified alerts. Our classification system is developed to identify the potential false alarms from

a huge number of alarms generated by the IDS. In general, with the extracted features from the alerts, the main task of the classification is to map the input data into 2 classes of true and false alarms. The alert features are the attributes that are measured in order to identify an intrusion, for example the IP addresses or the protocols. The multidimensional input data, which consist of more than one attribute, are mapped into a 2-dimensional space and are divided into two clusters, i.e. true and false alarms.

The framework of our approach comprises two main stages; first is alarm aggregation and second is classification, as shown in Fig. 1.

From the IDS sensors, the alert data are collected and stored in a database. The system then retrieves the data from the database and classifies them by extracting the attributes from the alerts and feeding them into the unsupervised SOM-based clustering system. Building accurate and efficient classifiers largely depends on the accuracy of the attributes used for the classification. Our method applies unsupervised SOM and K-means-based feature classification, whereby the attributes of the alerts are treated as the input data.

The whole procedure consists of four phases: feature extraction, alarm aggregation, cluster analysis and classification. In the feature extraction phase, the system will use several attributes extracted from the alert database, which are considered effective to correlate alerts generated from a single activity. The extracted data are then normalised since the value of the data is varied depending on the type of attributes used. A huge variance between attributes value will produce an uneven or biased result. Given a set of input vectors from the first phase, the SOM-based system is trained unsupervised in the second phase to map the data so that similar vectors are reflected in their arrangement. The distance between two input vectors is presented on the map, not by their absolute dissimilarity (which can be calculated), but the relative differences of the data properties. The maps created by SOM would be especially useful as a visual feedback to the user (network administrator), which is one of the main reasons why this approach is used. Whilst many other techniques offer a better or more accurate result for clustering and multidimensional scaling (Flexer, 1997), they were deemed not suitable for online, real-time data processing as SOM (a future enhancement for our system).

The SOM training process, through which the relationships are built, is fairly simple. During the training, SOM is expected to randomise the map's prototype vector elements within the range of the input value. The iteration is carried out to obtain a prototype which is most similar to the input vector. Once it is found, the prototype and its neighbours on the map are incrementally adjusted to more closely resemble the data.

As soon as the final Kohonen map is produced, the trained SOM can be automatically visualised using U-Matrix method. Having said that, SOM clustering alone is not good enough to describe the boundary between the data items since there are no clear walls to separate them from the other items. Classifying the data without any prior knowledge, thus, is rather inconsistent and difficult. The result of this U-Matrix can merely be used for visualisation purpose and the interpretation of the U-Matrix values is considered subjective. To avoid this issue, therefore, our approach applies a traditional

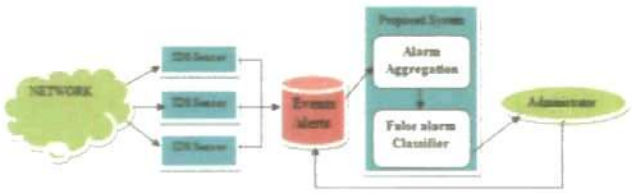


Fig. 1 – Framework of false alarm classification model.

clustering method, K-means clustering. Based on the map produced by the trained SOM, K-means clustering is implemented to further define the boundary between the data and concurrently classify the input vectors into a number of predefined clusters. At the end of the second phase, the system is expected to form clusters by correlating all alerts generated by a single activity, i.e. one cluster for each event/activity.

In the third phase, cluster analysis, the result of the classification is further evaluated to attain a set of attributes from each cluster created in the previous phase (the first classification). Seven alert attributes (features) were chosen to represent the value of each input vector in the second classification. Two out of the seven attributes, namely the frequency of alarm signatures and the average time interval between the alerts each day will be computed. These features are considered to be the most crucial attributes influencing the magnitude of the alert signatures.

The frequency of alarm signature defines the number of occurrences of an alarm signature from an event within a one-day period. The recurrence rate of a signature provides insight into the issue of superfluous alarms such as the noisy false alerts triggered by the ICMP traffic. The higher the frequency rate of an alarm signature, the more likely it is a noisy alert. In order to account for denial of service attacks, which could also generate high alarm frequency, the average time interval between events triggering a particular signature is chosen to describe the density of the signature and also to determine the validity of the triggered alerts.

In the last phase, the SOM and K-means algorithm are applied for a second time to re-classify the data based on the attributes extracted in the third phase into the classes of true and false alarms. In this stage, the frequency and time interval features are emphasised and it is necessary to examine how the attributes' weights from the two features can greatly affect the outcome of the classification. In which case, a fine-tuning is performed to adjust the attributes' weights and to ensure that such attributes contribute more to the grouping processes. The details of how the fine-tuning is performed will be presented later in Section 5.2. The final classification reveals that a cluster containing a higher frequency rate and a shorter time interval is prone to represent a false alarm class. The architecture of the false alarm classifier and the relationships among the components appear in Fig. 2.

4. Methodology

The basic concept, architecture and implementation technique of SOM can be found in Kohonen (1995). A Self

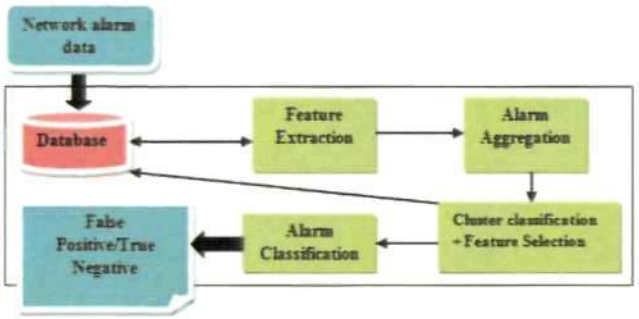


Fig. 2 – Architecture of false alarm classifier.

Organising Map (SOM) is an unsupervised neural network which produces a feature map that maintains the topology of the input data according to their similarity. Unlike typical neural networks that need to be trained with their desired outputs, SOM can automatically categorise the varieties of input presented during training without any external supervision whatsoever and assess the accuracy of its classification.

Unsupervised learning using SOM offers a simple and efficient way of clustering data sets. It is empirically proven that SOM is best suited to data classification due to their high speed and fast conversion rates as compared with other learning techniques (Labib and Vemuri, 2002). Also, in terms of its data representation, this method is deemed to outperform other algorithms owing to its ability to preserve topological mappings between the input data. This represents a significant feature, which is desired when introducing the relationship between the generated alerts.

The idea of the SOM algorithm is to perform a data compression technique (vector quantisation) where a high dimensional data is represented or mapped into something that is better understood visually such as a 2-dimensional array. The approach is considered as being highly effective as a complex visualisation tool for picturing extensive, multidimensional space with the intrinsic relationship among the various attributes comprising the data.

Interestingly, the dimensional transformation of the input data leads to an automatic classification, whereby similar data items are mapped in proximity; thus forming clusters. To obtain distinct classes of similar data elements from the trained SOM, a clustering algorithm, K-means, is applied to formally determine the clusters inherent in the structure of the data at the SOM's output layer. The basic algorithms for the applications of SOM and K-means clustering can be found in SOMToolbox (CIS, 2005). Unlike K-means, SOM algorithm is resistant to the presence of outliers in the data, and is also robust with regard to the choice of the number of classes to divide the data into, which is a desirable property and the main reason of applying SOM in between (Zanero, 2005). K-means (MacQueen, 1967) is a simple unsupervised learning algorithm that answers the well-known clustering problem by grouping n objects based on attributes into k partitions, where $k < n$. The implementation of K-means assumes all attributes to be independent and normally dispersed. The main concept of this approach is to define k appropriate centroids, one for each cluster and then group all data into the pre-defined k subsets. The grouping is done by calculating the sum of distances or sum of squared Euclidean distances from the mean of each cluster, as shown below.

$$\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2} \approx \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}, \qquad (1)$$

where p and q are the cluster points and n is the number of attributes.

Hence, the objective of this clustering is to minimise a measure of dispersion within the clusters and to maximise the distance between clusters.

Similar to other algorithms, K-means clustering also has weaknesses. K-means is considered to be unstable; running the procedure several times will give several different cluster solutions (Van der Heijden et al., 2004). Depending on its initial condition, the algorithm may converge or be trapped in the local optimum (minima). In addition, when the number of pre-specified classes is high, it often happens that some clusters are ignored during the classification as no sufficient support is given. In that case, the number of effective clusters will turn out to be much less than k.

With the issues of classification in mind, our approach is directed to focus upon an interaction between the intrinsic structures (alerts' attributes) in the instances, the representation of the data and the definition of the clustering problem. Solutions are suggested to overcome the issues, at least partially, and the effect of the problems will be considered when representing the data and interpreting the result of the clustering process.

The K-means procedure will be started by assigning the data to k initial clusters at random. It is also worth noting that the cluster solutions can be influenced by the order of the input data. The randomised trials therefore involve randomising both the initial clusters and the data order. To get the best clustering solution, the proposed system looks for the top solution by exploring a range of cluster solutions produced by the procedure and examining their criterion value; involving the minimum sum of squared error and the highest frequency rate.

5. Experiments and evaluation

This section presents the experimental results of our false alarm classifier. Two experiments were performed. For the experiments, we used two types of data sets; the public and the private data sets. Given that the 1999 DARPA evaluation data set is deemed to be the largest publicly available benchmark, our experiments aimed to utilise such data as the source of our evaluation. A number of criticisms had been raised over the DARPA data set, questioning the use of synthetic data to picture a real world network, as well as the taxonomy used to categorise the exploits involved in the evaluation. Owing to these issues, our experiments involved evaluation on both DARPA and a private data set. The private data were collected at the University of Plymouth, on a public network (100–150 MB/s network), logging all web traffic to and from the University's web server. It should also be noted that traffic containing web pages with the potential of having sensitive/confidential information was excluded from the packet capture, in order to preserve the privacy of the users. As the

main objective of the system is to facilitate alarm management for the administrator, the proposed technique is designed to process the generated IDS alerts every 2 h. So, instead of using a whole data set, the experiments evaluated only a chunk of DARPA 1999 and the private data as the input of the IDS system. In this case, only week 4 testing data from DARPA 1999 evaluation data set is fed into the system. Table 6 presents the properties of data set selected for the experiments.

To obtain a set of network alarm data for our classification system, we firstly run the Snort IDS (Caswell and Roesch, 2004) under Linux Fedora 7 against the DARPA and private data set. The reason for utilising Snort is due to its openness and public availability. In order to facilitate the analysis of IDS alerts, a front-end tool Basic Analysis and Security Engine (BASE, 2007) was then utilised as the intrusion analyst console. Regarding the neural networks, the SOM-based and K-means system is implemented on the SOMToolbox 2.0 (CIS, 2005) which is run on MATLAB 7.5.0.

5.1. STAGE 1 – *alarm aggregation*

Traditional IDS is believed to commonly produce a large volume of alerts, consisting of redundant and low priority alarms. To reduce the number of redundant alerts generated from the same event, therefore, our study proposes an alarm aggregation approach that effectively combines all alerts triggered from a single activity or event in a particular time frame. The key objective of this mechanism is to pinpoint the triggering events from the incoming alarms and to help add meaning to the alarms generated. It is not unusual for IDS to generate more than one signature from a single event. Presenting those alerts individually could degrade the value of the alarms. By contrast, correlating all alerts triggered by a single event, could increase the meaning of the alarms, and *make it possible to discover the potential attack scenario.*

In order to correlate related alarms, we need to remove the inapt attributes and select only appropriate attributes. After evaluating a number of potential features, three significant attributes have been chosen to represent the relationships between alerts. Those are the timestamp, the source and destination IP addresses. IP address is deemed to be the most critical feature determining the subject of the occurrence. Conversely, the timestamp determines the time of the event and whether a particular alert within a specific time period should be aggregated. By using the combination of these features, we expect to correlate alerts triggered by particular IP addresses within a particular period of time.

In order to correctly spot the events triggered by particular hosts, we decided to use the combination of both source and destination IP addresses. So, instead of using the original IP addresses, the system is designed to compute the addition and the subtraction between the source and destination IP addresses. Before the computation, the IP addresses were converted into their decimal value from the common dotted decimal notation (e.g. 123.7.1.10 becomes 2064056586). The computed value is then fed into the system for the classification. The main objective of this pre-processing step is to obtain a distinctive pair of IP addresses from an alert without the need of identifying the source and destination addresses. Such approach enables us to connect all alerts which involve the two IP addresses within a particular time frame. For example, alerts generated by ICMP Ping and ICMP Echo Reply signatures can be correlated since they commonly associate to a same pair of IP addresses. In order to obtain a same pair of addition and subtraction values of two IP addresses in any order, only the absolute value of the subtraction is taken. For example, if the subtraction between 2886758706 (source) and 2886759119 (destination) is −413, then the absolute value 413 is selected. So, although the source/destination has changed (reply), the subtraction will still yield the same value. As this technique uses the characteristics of both difference and addition of IP addresses, which are taken in time context, the likelihood of having collisions is low (different pairs of IP addresses are mapped into the same cluster) (Chyssler et al., 2004a). A unique combination of the value, hence, indicates a unique event triggered by the corresponding IP addresses.

Apart from the IP addresses, the third attribute, timestamp, also requires a slight conversion. As the timestamp is represented as date string format rather than a number, an alteration is necessary. The timestamp is normally presented as date vector, consisting of 6 elements specifying year, month, day, hour, minute and second. So, in order to perform the conversion whilst keeping the value of the attribute, we utilise "datenum" function from MATLAB to convert the string or date vector into a serial date number.

Using the three-dimensional vectors to build SOM map directly is likely to be biased to a certain dimension, as different attributes values tend to be in different units. If some vector components have variance which is considerably higher than other components, they will certainly dominate the map formation. Therefore, normalisation is performed to control the variance of the vector components. Our experiments utilise variance normalisation method, which is known as "var" (CIS, 2005). This is a linear transformation which *scales the values such that their variance is equal to 1.*

The number of neurons or the size of the map itself greatly influences the performance of SOM system. In the classical SOM, the number of neurons should usually be selected as big as possible, with the neighbourhood function maintaining the efficiency and generalisation of the mapping. The increase of the map size, however, could cause the training phase become computationally and impractically heavy for most applicants. With the aim of gaining the best map result, we decided to select the number of neurons based on the smallest quantisation and topographic errors, where errors < 0.1. In order to do so, we run a loop programme creating maps with different number of units and the programme will be terminated once the map has the quantisation and topographic errors less than 0.1. The quantisation and topographic errors are computed after training to measure the quality of the generated map. However, bear in mind that a low quantisation error does not necessarily mean a good result; it might lead to the issue of overfitting. This may happen when the numbers of units are larger than the number of training data (CIS, 2005). Having said that, overfitting is not a real problem since K-means is applied as a second classifier. In fact, the implementation of multi-stage classifiers can actually avoid the issue of overfitting (Weijters et al., 1997). The number of units, the topographic and quantisation errors of the data are presented in table 6.

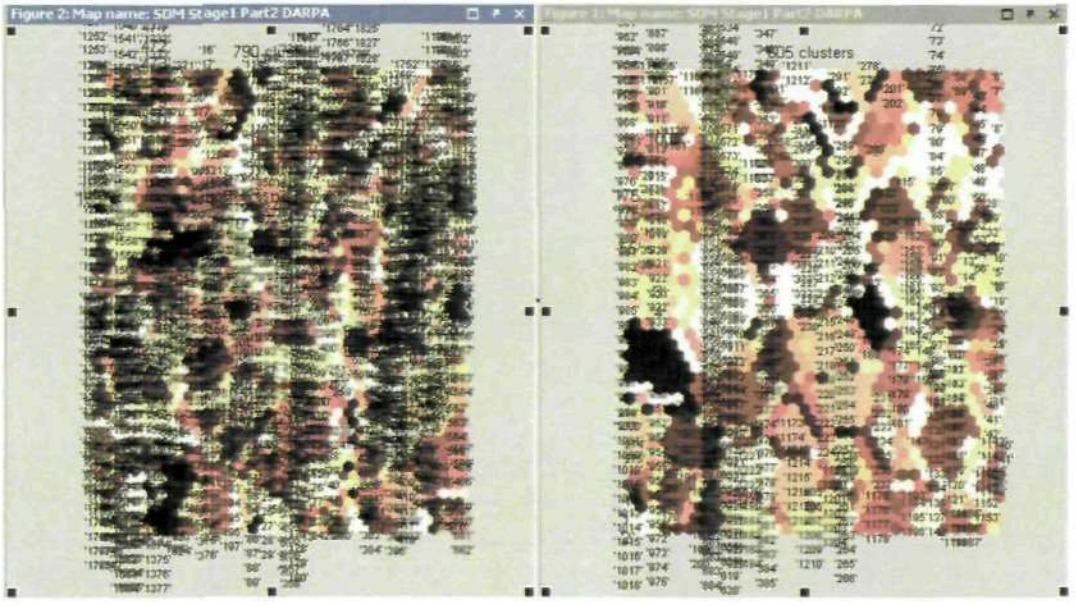


Fig. 3 – Stage 1 classification using DARPA 1999 dataset.

One of the most significant weaknesses of K-means clustering is the need to determine the number of clusters prior to classification. The default setting for K-means initialisation value k (maximum number of clusters), set by SOMToolbox, is the square root of the length of data. As in the first stage, we never know the real number of the clusters, and we believe that the data can be classified into more clusters than specified by the default setting above. To affirm this idea, 4 sample data were taken and manually analysed to estimate the expected number of clusters and three of them had clusters reached up to two fifth of the length of data. Hence, in order to avoid possible misclassification, our system determines to increase the "k" value for K-means to half of the length of data. Again, the problem of overfitting is very common in the subject of data mining and neural network. Such issue occurs when the number of nodes (clusters) is as large or larger than the number of training cases (CIS, 2005). Since the number of training data used in our experiment is two times more than the clusters (k = ½ data), the network is unlikely to suffer from overfitting.

To overcome the weakness of K-means clustering, the system generates 500 randomised trials, involving randomising both the initial k clusters and the data order. The best classification is then selected based upon the highest frequency and the minimal sum of squared errors. The sum of squared error refers to the least distance between the data and the corresponding cluster centroid. In the K-means algorithm, each attribute is assumed to have the same weight; which then makes it impossible to know which feature contributes more to the grouping process. Having said that, the value of the attributes' weights can be completely adjusted if the fine-tuning is desired.

As mentioned before, two data sets are utilised in the experiments. For DARPA data set, 4 h of data (total 3062 alerts) was extracted from the first day of 4th week testing data and was evaluated as two separate inputs. Fig. 3 presents the result of the stage 1 DARPA classification.

A total of 790 clusters have been generated in the first part of classification; shown on the left map. Interestingly, only 203 of them are active whilst the rest are considered dead centres. Similarly, from 605 clusters generated in the second part (shown on the right map), only 86 classes are active. This seems obvious that K-means clustering tends to generate a significant number of dead centres. Enhancing the K-means performance, however, is out of the scope of the study and is not discussed in this paper.

In general, the classification has demonstrated a reasonable outcome. Approximately 93% of data from the first part classification have been mapped and classified into the correct clusters, i.e. accuracy = 0.93. Conversely, 0.9 is revealed from the second classification. In terms of clustering accuracy (the number of clusters with the correct data), the first classification shows 0.86 accuracy, whilst second classification reveals 0.81.

Similar to the DARPA data set, 4 h data from University's network data (2556 alerts) was analysed using two separate inputs. Fig. 4 presents the result of the classifications.

The classification from this network data shows a slightly better result compared to those from DARPA data set. Approximately 0.92 and 0.94 are computed for the first and second classifications, whilst the cluster accuracy accounts for 0.89 and 0.93 respectively.

5.2. *STAGE 2 – false alarm classification*

Having correlated the related alerts into a number of clusters, the second stage of classification is carried out to further label the alerts into true and false alarms. The main objective of this false alarm classification stage is to obtain a better alarm management by reducing the number of false alarms generated before being presented to the administrator. Besides, the organisation of the data using the SOM-based system enables

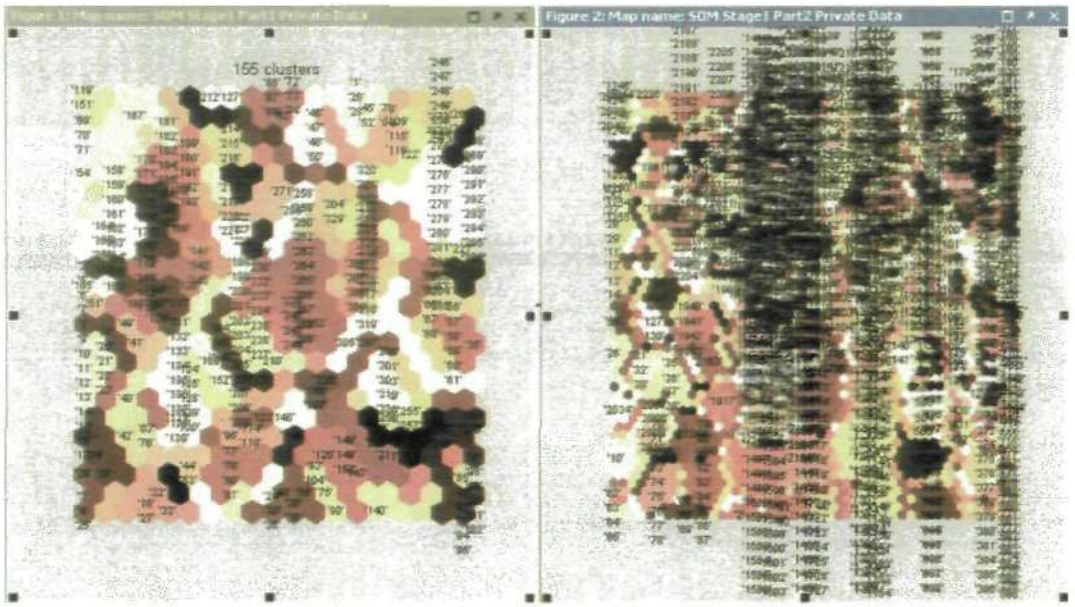


Fig. 4 – Stage 1 classification using University of Plymouth Network data.

us to learn the relationship between alerts based on the defined attributes.

Similar to the first mapping, the alert attributes are selected and pre-processed prior to the classification. However, in this case, the outcome of the first classification will be fed into the second alarm classifier; meaning that each cluster is an input for the second classification. Using the association-rule method (Piatetsky-Shapiro, 1991), which looks for the most frequent itemsets, 7 alert features are selected from each alert cluster. The selected attributes are the number of alerts, number of signatures, port number, protocol, priority, time interval and the number of events, have been chosen as the dimensions of the input data. The attributes are carefully selected to describe the inherent relationship between alerts. Table 1 presents a brief description of the selected alarms' attributes and their data collection methods.

The final result of the K-means classification largely hinges upon the dimensions or the attributes applied in the SOM mappings. Typically K-means algorithm treats all features fairly and distributes the weights on all attributes equally. The features' weights can be derived based on the importance of the feature to the clustering quality. The higher the attribute's weight, the more the contribution it has on the clustering

Table 1 – The interpretation and data collection methods of the alarm attributes for second classification.

Alert features	Description	Collection methods
No of alerts	Total number of alerts grouped in one cluster	N/A
No of signatures	Total number of signature type in a cluster	N/A
Protocol	Type of traffic from event triggering the alerts	There are only three values that can be assigned to this feature. Alert with the protocol number below 255 is assigned to a value of 1 and 3 for protocol number 255. If there are two types of protocol number found in a cluster, the value is set to 2.
Port number	Only the service port number is applied in the classification.	If the alert contains a well-known port number (<1024), the value will be set to 1; if not (≥1024) value of 3 will be given. If the cluster has two types of port numbers, then the value will be set to 2.
Alert priority	Criticality of the alerts. There are 3 types of alert priority, namely 1st, 2nd, and 3rd.	Based on the type of signature, alert with the 1st priority is assigned to a value of 300, 2nd to 200 and 3rd to 100. If multiple signatures are found in a cluster, the priority value for each signature could be added together.
Time interval	Time interval between events[a] from a particular signature	Should an alert signature occur in 3 different events in a particular time frame (one-day), the mean of the time interval between each event is calculated. This attribute is computed in seconds. However, if there are multiple signature types in one cluster, the highest time interval will be selected.
No of events	The number of events[a] in which a particular alert signature is triggered within a day	If there are multiple signature types in a cluster, the lowest no of events is selected.

a One event equals to a number of alerts from a single signatures, which are triggered by a particular activity.

Table 2 – SSE and frequency rate from DARPA data set part 1.

MAP	1	2	3	4	5	6	7	8	9	10	11
SSE	6.3276	6.335	6.5053	6.5056	6.8716	6.8721	6.8731	6.8767	6.9091	7.6474	7.6807
Frequency	0.362	0.332	0.062	0.064	0.004	0.004	0.004	0.044	0.002	0.056	0.066

process. Amongst the seven features two attributes, namely the alarm frequency rate and the time intervals between events are deemed to be the most influential one. So, to ensure such features contribute more to the clustering processes than the remaining five, we decided to carry out the fine-tuning by properly adjusting the attributes' weights.

In order to achieve an ideal weight, we conducted an experiment to search for the best classification outcome by randomly setting the weights of the two leading attributes to be higher than the remaining features. Although the main objective of this weight adjustment method is to prioritise the attributes on the classification processes, we do not want to "over-weight" the features; leading to a biased clustering result. From our observation, we had found that the data started producing inequitable classification once an attribute's weight was set more than 3 times higher than other features. So, to avoid this problem, the weight values of the primary attributes (w) will be selected from 1 to 3 times higher than the other attributes ($1 < w \geq 3$). To test the attributes' weights, the first selected value (e.g. $w = 1 + 0.1 = 1.1$) is multiplied to the corresponding attribute's value. The updated attribute is then evaluated and run on the classification algorithm. The experiment will be further conducted to assess different weight values by gradually increasing the selected value by 0.1 (e.g. $w = 1.1 + 0.1 = 1.2$). The ideal weight is determined by the best classification outcome. In our system, the finest weight values for the two leading attributes are set to be 2.8.

As mentioned before, 500 randomised trials are taken; the comparison is made based on their sum of squared error (SSE). In K-means, the most essential approach in determining the best classification result is by looking into its SSE value (MacQueen, 1967). Hence, this feature is taken as one of our selection criteria to select the finest cluster solution. A map is considered equal to other maps if they have the same SSE value.

In K-means algorithm, the lower sum of squared error the more accurate the classification should be. This theory, however, in some cases, might not apply to our system. In the map with the lowest SSE value, the algorithm tends to assign the centroids to the data points with the farthest distance; generating two clusters with highly unbalanced cluster sizes. This definitely indicates two poor outcomes; either a tighter security level with a lower reduction rate or a loose security level with the risk of false negatives. Such an issue clearly demonstrates the trade-off between maintaining the security level and the need for reducing the false alarms.

In view of this trade-off issue, thresholding is required to balance the security issue and the alarm reduction. Since we are using the randomised experiments to select the best cluster solution, evaluating the frequency distribution of each solution is necessary. So, instead of merely focusing upon the lowest SSE value, the best map is also selected based on its frequency rate (frequency distribution). The frequency refers to the number of occurrences a map with a particular SSE value is created within the 500 randomised trials. In order to properly evaluate the maps, we conducted 5 other classifications using 5 sets of sample data to examine the maps' frequency distribution. From our observation, a cluster solution with a frequency rate above 0.6 (300 out of 500) had the best classification result compared to other solutions. From our study, it is evident that a solution with a high probability distribution (reassuringly occurs in at least the third fifth of the random trials) generates a better grouping compared to those with low frequency rates. We, therefore, decided to select the best classification solution based on the highest frequency rate and set the thresholding value to 0.6 since it appears to practically balance both security and alarm reduction. Any map with a frequency rate exceeding the thresholding value will be automatically selected as the finest choice without any further evaluation. Conversely, if the highest frequency rate falls below the value (it does not dominate other solutions), further evaluation will be required in this case. To find which of the maps are worth considering, it is necessary to set another threshold to select the dominant solutions. The second thresholding (s), which is derived from a standard deviation of the maps' probability distribution, will determine which of the cluster solutions need further investigation. The standard deviation represents the average variation of the frequency rates from the mean distribution. From our 5 sample classifications, it is apparent that the maps with frequencies ranging from t to $t - s$ are likely to produce better clustering results compared to those with low frequency rates. So, for this reason, only those solutions with frequency rates that fall between t (highest frequency) and $(t - s)$ are evaluated.

Tables 2–5 show the value of the SSE and the frequency rate from the University's network and the DARPA classifications.

There were eleven maps (cluster solutions) that had been produced in the first part of DARPA classification, as shown in

Table 3 – SSE and frequency rate from DARPA data set part 2.

MAP	1	2	3
SSE	2.4721	2.9135	4.0591
Frequency	0.786	0.154	0.06

Table 4 – SSE and frequency rate from private data set part 1.

MAP	1	2	3
SSE	2.7161	2.8643	2.8645
Frequency	0.636	0.182	0.182

Table 5 – SSE and frequency rate from private data set part 2.

MAP	1	2	3	4
SSE	3.0918	5.3372	5.6775	5.6784
Frequency	0.818	0.122	0.032	0.028

Table 2. Since none of the maps has a frequency rate above 0.6, further evaluation is required in this case. We need to re-evaluate other maps whose frequency range from t (*highest frequency rate*) to $(t - s)$. The average SSE of the selected maps is then computed. The aim of calculating the average SSE is to approximate the value of SSE; thus enabling us to select the best map. The map, whose SSE value is closest to the average SSE, will be determined to be the most optimal solution. If only two maps are being selected, there is no need for computing the average SSE, the map with the lowest SSE will be chosen.

Hence, to conclude this, we will label the cluster solution to be the most appropriate model if only it follows one of the following rules:

1. It has the frequency rate above 0.6; if not
2. Its frequency rate greatly exceeds other solutions ($t - s > y$ (*second highest frequency*)); if not
3. The average SSE value of the maps whose frequency rates between t and $t - s$ is calculated, as shown in equation (2). The cluster solution with the SSE value closest to the average SSE is then selected as the best map choice.

$$\left(\sum_{i=1}^{n} \mathrm{SSE}_i\right) \div n. \tag{2}$$

where n is the number of map solutions whose frequency rates range from t to $(t - s)$.

The answers for our second stage clustering are presented in the following figure.

Only two clusters, the true and the false alarm classes, are desired in this stage. The result shown on the left map from Fig. 5 is corresponding to criterion value in Table 2. It is obvious that the solution with the highest frequency rate in Table 2 does not conform to the first and second criterion rules (MAP 2 has the frequency rate higher than $t - s$; t = MAP 1's frequency rate; $t = 0.362$; $s = 0.129$). Since only two maps (MAP 1 and MAP 2) have frequency rates higher than $t - s$ ($0.362 - 0.129 = 0.233$), the one with the lowest SSE value is selected. In this scenario, MAP 1 is selected as the best map choice (presented on the left side of Fig. 5). On the other hand, the second part of DARPA classification, which is shown on Table 3, presents 3 possible cluster solutions. The solution with the highest frequency (MAP 1) is automatically chosen as the best map since the frequency rate (0.786) has exceeded the first thresholding value. The mapping result is presented on the right map in Fig. 5.

Unlike the first part of DARPA classification, the classifications on private data set reveal quite a straightforward result. The computation of the average SSE value is not required in this scenario as the highest frequency rates from both classifications shown on Tables 4 and 5 conform to the first criterion. In view of this, the solutions with the highest frequency rate are determined to be the best maps. In addition, the selected maps have the lowest SSE value among all cluster solutions. The final results of both classifications are presented in Fig. 6.

Regarding the DARPA data set, the proposed system is considered effective in reducing the number of false alarms; with 95% being correctly labelled in the first classification, whilst the second categorisation has reduced approximately 99% of the total false alarms. Those alarms located in the upper portion are labelled as true alarms, whilst the lower portion is for the false alarms. The system appears effective in reducing the false alarms generated by a noisy traffic such as the ICMP traffic (ICMP Ping and Echo Reply) and the web-bug alerts, which have formed the highest number of false alarms triggered in the experiment (Tjhai et al., 2008a).

In our previous experiment, false alarms such as ICMP and INFO web-bug alerts had contributed to 62% of total alerts *generated from DARPA 1999 data set (Tjhai et al., 2008a)*. Logging every connection associated with probing, for example all ping activities, will only generate a huge number of false positives. In fact, all detected ICMP traffic did not surely imply the occurrence of probing activities, but it was merely an information event, which possibly indicates the occurrence of network outage. The highest number of false alarms was triggered by INFO web-bug 1 × 1 gif attempt signature. Theoretically, the web bug is a graphic on the web

Table 6 – Properties of DARPA and Plymouth private data sets.

DARPA	No of Alerts	Stage 1			Stage 2			Result
		Map units	Errors	"k" Value	Map units	Errors	"k" Value	
Part 1	1224	1333	Q = 0.001 T = 0.019	612	190	Q = 0.048 T = 0.012	2	FA = 1131 TA = 93
Part 2	1838	1924	Q = 0.009 T = 0.041	919	299	Q = 0.097 T = 0.054	2	FA = 1297 TA = 541
Plymouth private data								
Part 1	330	437	Q = 0.050 T = 0.048	165	290	Q = 0.097 T = 0.043	2	FA = 260 TA = 70
Part 2	2226	2385	Q = 0.003 T = 0.011	1113	260	Q = 0.044 T = 0.077	2	FA = 2139 TA = 87

Q = quantisation error, T = topographic error, FA = false alarm, TA = true alarm.

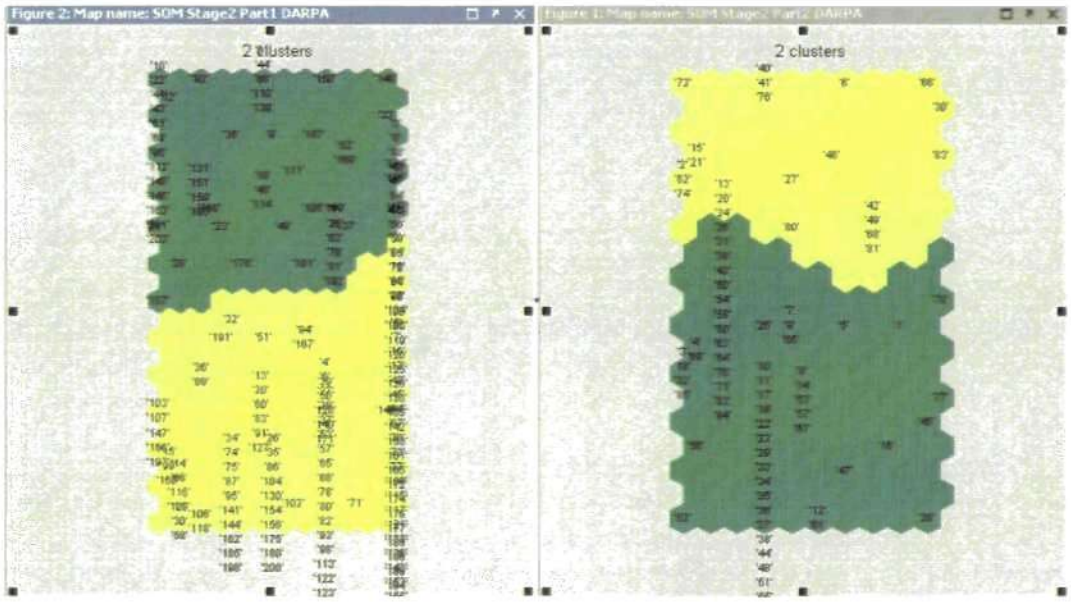


Fig. 5 – Stage 2 alarm classifier using DARPA dataset.

page and email message, which is used to monitor users' behaviours. This is often invisible and hidden to conceal the fact that the surveillance is taking place (Smith, 1999). Since none of these web-bug alerts related to any attack instances, the study revealed that no true alarms associated with this signature had been generated.

As for the private data, the classification reveals that about 78.8% of false alarms have been identified in the first map, whereas 96% of them have been detected in the second mappings. It is notable that our system has shown promising result in filtering all hectic and unnecessary alerts triggered by the IDS. For example, the alerts from WEB-IIS view source via translate header and WEB-MISC robots.txt access signatures, which had caused 82% of false alarms from the entire private data (Tjhai et al., 2008b).

WEB-MISC robots.txt access is a signature raised when an attempt has been made to access robots.txt file directly. Robots.txt file provides a specific instruction and determines which part of a website a spider robot may visit. Although this signature is triggered as the indicator of vulnerable information attack, there exits high possibility that all these alerts were raised owing to the legitimate activities from web robots or spiders. As the spider's web indexing is regularly and structurally repetitive, this activity tends to cause the IDS to trigger a superfluous amount of false alerts. On the other hand, WEB-IIS view

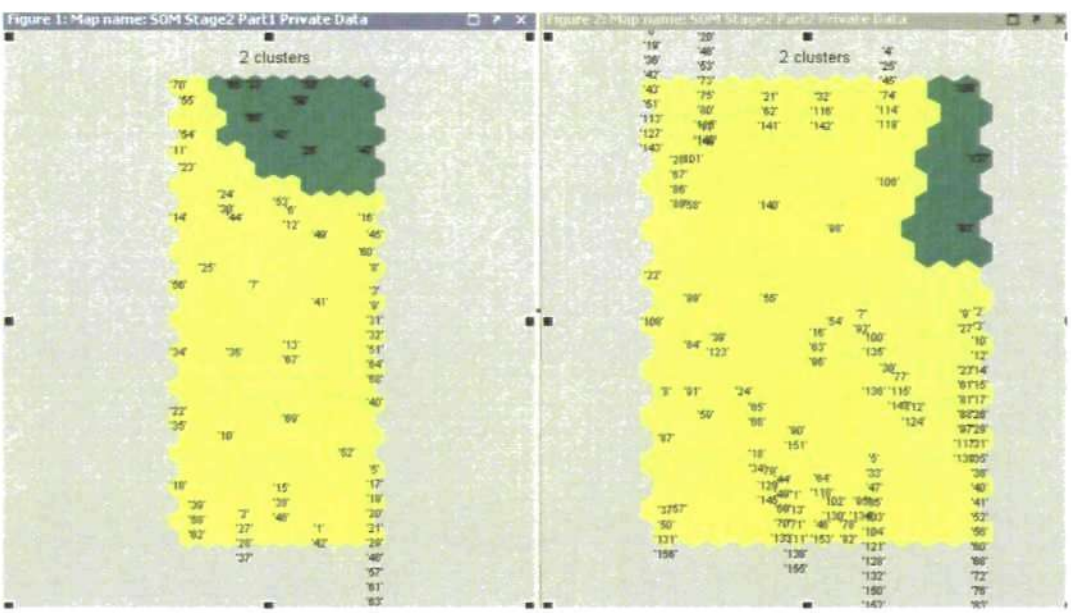


Fig. 6 – Stage 2 alarm classifier using private data.

source via translate header targets the Microsoft IIS 5.0 source disclosure vulnerability by using a specialised header "Translate f" on an HTTP GET request (Snort, 2007). Although this event is deemed to be an attack that targets the Microsoft IIS source disclosure vulnerability, this could possibly be a false positive. Some applications, for example Web-based Distributed Authoring and Versioning (WebDAV) that make use of "Translate f" as a legitimate header, might cause this rule to generate an excessive amount of false alarms (WebDAV Overview, 2001).

Our suggested alarm filtering system is believed to significantly outperform other existing methods. Unlike many proposed systems that need to be trained with a considerable volume (gigabytes) of attack-free data, our system applies unsupervised training to train the classifier; hence no attack-free data are necessary. In terms of its configuration, our approach is considered efficient enough as it is easy to set up and no knowledge of the attacks is required to filter the alarms. Moreover, the system's filtering processes are independent from the intrusion detection process. Therefore, we believe that our model can be applied to other signature-based IDSs without changing the existing filtering configuration. As to its performance, the system does not only provide a better alarm management, but also shows the relationship between the generated alerts, thus enabling administrator to discover the potential attack scenarios.

6. Conclusion and future work

In this paper, we have proposed a technique to detect and to subsequently reduce the number of IDS false alarms. We developed a two-stage classification system using the combination of two data mining techniques, namely SOM and K-means clustering. The first stage classification was developed to properly correlate alerts related to a particular activity. All alerts, regardless the signature type, triggered by a single event are mapped and grouped into one cluster. In addition, the main objective of the second stage is to subsequently label all clusters produced in the first classification into groups of true and false alarms.

To verify the idea, we carried out preliminary experiments with two different data sets; the 1999 DARPA IDS evaluation data set and our own set based upon private network data. The result shows that more than 90% of false alarms from DARPA data set were filtered without ignoring the true alarms whilst approximately 87% of false alarms from private data set can be correctly identified. Despite the lower false alarm detection rate than the DARPA data set, our system has demonstrated its effectiveness in filtering all noisy and unnecessary IDS alerts, which have usually contributed to more than 50% of false alarms from the common IDSs.

As our classifier is in a preliminary stage and was only evaluated using a small chunk of DARPA and private data, we plan to further assess the system using the complete DARPA and Plymouth data sets. We also focus on improving the approach by applying it to live data in order to prove that our model is applicable under a real life operational environment.

REFERENCES

Alharby A, Imai H. IDS false alarm reduction using continuous and discontinuous patterns. In: Third international conference on applied cryptography and network security, ACNS, New York, United State. Lecture notes in computer science, vol. 3531; 2005.

Basic Analysis and Security Engine (BASE) Project, http://base.secureideas.net/; 2007.

Caswell B, Roesch M. Snort: the open source network intrusion detection system, <http://www.snort.org/>; 2004.

Chyssler T, Nadjm-Tehrani S, Burschka S, Burbeck K. Alarm reduction and correlation in defence of IP networks. In: Proceedings of the 13th IEEE international Workshops on enabling technologies: infrastructure for collaborative enterprises; 2004a, p. 229–34.

Chyssler T, Burschka S, Semling M, Lingvall T, Burbeck K. Alarm reduction and correlation in intrusion detection systems. The Department of Computer and Information Science Linkopings Universitet, <http://www.ida.liu.se/rtslab/publications/2004/Chyssler04 DIMVA.pdf>; 2004b. Available via:.

CIS. SOM toolbox 2.0, <http://www.cis.hut.fi/projects/somtoolbox/>; 2005.

Cuppens F, Miege A. Alert correlation in a cooperative intrusion detection framework. In: Proceedings of the 2002 IEEE symposium on security and privacy; 2002, p. 202.

Dain O, Cunningham RK. Fusing a heterogeneous alert stream into scenarios. In: Proceedings of the 2001 ACM workshop on data mining for security application, Philadelphia, PA; 2001, p. 1–13.

Debar H, Wespi A. Aggregation and correlation of intrusion-detection alerts. In: Proceedings of the fourth international symposium on recent advances in intrusion detection, Davis, CA, USA; 2001, p. 85–103.

Flexer A. Limitations of self organizing map for vector quantization and multidimensional scaling. In: Mozer MC, et al., editors. Advances in neural information processing systems, vol. 9. MIT Press/Bradford Books; 1997. p. 445–51.

Julisch K, Dacier M. Mining intrusion detection alarms for actionable knowledge. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining; 2002, p. 266–375.

Julisch K. Mining alarm clusters to Improve alarm handling efficiency. In: Proceedings of the 17th annual conference on computer security applications; 2001, p. 12–21.

Kohonen T. Self-organizing maps. 1st ed. Germany: Springer, ISBN 3540620176; 1995.

Labib K, Vemuri R. NSOM: a real-time network-based intrusion detection system using self-organizing maps. In: Networks and security; 2002.

Law KH, Kwok LF. IDS false alarm filtering using KNN classifier. In: Fifth international workshop on information security applications, WISA, Jeju Island, South Korea. Lecture notes in computer science, vol. 3325; 2004.

MacQueen JB. Some methods for classification and analysis of Multivariate Observations. In: Proceedings of fifth Berkeley symposium on mathematical statistic and probability, vol. 1. Berkeley: University of California Press; 1967. p. 281–97.

Ning P, Cui Y, Reeves DS. Constructing attack scenarios through correlation of intrusion alerts. In: Proceedings of the ninth ACM conference on computer and communications security Washington, D.C.; 2002, p. 245–54.

Piatetsky-Shapiro G. Discovery, analysis, and presentation of strong rules. In: Piatetsky-Shapiro G, Frawley WJ, editors. Knowledge discovery in databases. Cambridge, MA: AAAI/MIT Press; 1991.

Quinlan JR. C4.5: programs for and neural networks, machine learning. 1st ed. United State of America: Morgan Kaufman Publishers, ISBN 1558602380; 1993.

Shin MS, Kim EH, Ryu KH. False alarm classification model for network-based intrusion detection system, intelligence data engineering and automated learning. In: Lecture notes in computer science, vol. 3177/2004; 2004. 259–265.

Smith R. The web bug FAQ, <http://w2.e®.org/Privacy/Marketing/webbug.html>; 1999.

Snort: WEB-IIS view source via translate header, <http://snort.org/pub-bin/sigs.cgi?sid=1042>; 2007.

Symantec: symantec global internet security threat report: trends for 2008, vol. XIV; April 2009, http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiv_04-2009.en-us.pdf; April 2009.

Tjhai GC, Papadaki M, Furnell SM, Clarke NL. The problem of false alarms: evaluation with Snort and DARPA 1999 dataset. In: Trust, privacy and security in digital business. Lecture notes in computer science, vol. 5185/2008; 2008a. p. 139–50.

Tjhai GC, Papadaki M, Furnell SM, Clarke NL. Investigating the problem of IDS false alarms: an experimental study using Snort. In: Jajodia Sushil, Samarati Pierangela, Cimato Stelvio, editors. Proceedings of the IFIP TC 11 23rd international information security conference. IFIP international federation for information processing, vol. 278. Boston: Springer; 2008b. p. 253–67.

Valdes A, Skinner K. Probabilistic alert correlation. In: Proceedings of the fourth international symposium on recent advances in intrusion detection, Davis, CA, USA; 2001, p. 54–68.

Van der Heijden F, Duin R, Ridder D, Tax DMJ. Classification, parameter estimation and state estimation. 1st ed. United State of America: Wiley, ISBN 978-0-470-09013-8; 2004.

WebDAV overview, <http://www.sambar.com/syshelp/webdav.htm>; 2001.

Weijters T, Van Den Herik HJ, Van den Bosch A, Postma E. Avoiding overfitting with BP-SOM. In: Proceedings of the fifteenth international joint conference on artificial intelligence; 1997.

Zanero S. Analyzing TCP traffic patterns using self organizing maps. In: Proceedings 13th international conference on image analysis and processing. Lecture notes in computer science, vol. 3617/2005; 2005. p. 83–90.

Zhu B, Ghorbani A. Alert correlation for extracting attack strategies. International Journal of Network Security 2006;3(2): 244–58.

Gina Tjhai received a Bachelor Degree in Computer Science from the University of Wollongong, Australia, in 2005, and the MSc in Information System Security from the University of Plymouth, UK, in 2006. She is currently a PhD candidate in Centre for Security, Communications and Network Research at University of Plymouth, UK. Her current research interests include network intrusion detection and prevention, pattern classification, neural network and data mining.

Steven Furnell heads the Centre for Security, Communications and Network Research at the University of Plymouth in the United Kingdom, and is an Adjunct Professor with Edith Cowan University in Australia. He specialises in computer security and has been actively researching in the area for fifteen years, with current areas of interest including security management, computer crime, user authentication, and security usability. Prof. Furnell is a Fellow and Branch Chair of the British Computer Society (BCS), and a UK representative in International Federation for Information Processing (IFIP) working groups relating to Information Security Management (of which he is the current chair), Network Security, and Security Education. He is the author of over 190 papers in refereed international journals and conference proceedings, as well as the books Cybercrime: Vandalizing the Information Society (Addison Wesley, 2001) and Computer Insecurity: Risking the System (Springer, 2005). Further details can be found at www.plymouth.ac.uk/cscan.

Maria Papadaki is a lecturer in Network Security, at University of Plymouth, UK. Prior to joining academia, she was working as a Security Analyst for Symantec EMEA Managed Security Services (MSS), UK. Her postgraduate academic studies include a PhD in Intrusion Classification and Automated Response, and an MSc in Integrated Services and Intelligent Networks Engineering, both awarded from University of Plymouth, UK. Her research interests include intrusion prevention detection and response, network security monitoring, asset classification, threat management, security usability, and security education. Dr Papadaki is a GIAC Certified Intrusion Analyst, and is a member of the British Computer Society. Further details can be found at www.plymouth.ac.uk/cscan.

Nathan Clarke graduated with a BEng (Hons) degree in Electronic Engineering in 2001 and a PhD in 2004 from the University of Plymouth. He has remained at the institution and is now a senior lecturer in Information Systems Security within the Centre for Security, Communications and Network Research. Dr Clarke is also an adjunct scholar at Edith Cowan University, Western Australia. His research interests reside in the area of user identity, mobility and intrusion detection; having published 40 papers in international journals and conferences. Dr Clarke is a charted engineer, member of the British Computer Society (BCS), the Institute of Engineering Technology (IET) and a UK representative in the International Federation of Information Processing (IFIP) working groups relating to Information Management Identity Management and Information Security Education.

References

360IS (2010). Preparing Security Event Management, available from: http://www.windowsecurity.com/uplarticle/NetworkSecurity/360is-prep-sem.pdf. (Cited on page 33.)

Al-Mamory, S. and Zhang, H. (2009). Intrusion Detection Alarm Reduction Using Root Cause Analysis and Clustering, *Computer Communications* **32**(2): 419–430. (Cited on pages 20, 25, 60 and 73.)

Albayrak, S., Scheel, C., Milosevic, D. and Muller, A. (2005). Combining Self-Organizing Map algorithms for robust and scalable intrusion detection, *In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA 2005)*, Washington, DC, USA, pp. 123–130. (Cited on page 58.)

Alharby, A. and Imai, H. (2005). IDS False alarm reduction using continuous and discontinuous patterns, *Lecture Notes in Computer Science – Third International Conference on Applied Cryptography and Network Security*, Vol. 3531, New York, USA, pp. 192–205. (Cited on pages 21, 23 and 34.)

Amoroso, E. (1999). *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Traps, Trace Back and Response*, 1st edn, Intrusion.Net Books, United States of America. ISBN 0966670078. (Cited on page 13.)

Axelsson, S. (2000). The Base-Rate Fallacy and the Difficulty of Intrusion Detection, *ACM Transactions on Information and System Security* **3**(3): 186–205. available online: http://www.scs.carleton.ca/~soma/id-2007w/readings/axelsson-base-rate.pdf, date visited: 22 January 2008. (Cited on pages 2, 12 and 15.)

Baba, T. and Matsuda, S. (2004). A Proposal of Protocol and Policy-Based Intrusion Detection System, *Systemics, Cybernetics and Informatics* **2**(3): 57–62. (Cited on page 11.)

Bace, R. (2000). *Intrusion Detection*, 1st edn, Sams, United States of America. ISBN 1578701856. (Cited on page 1.)

BASE (2009). Basic Analysis and Security Engine (BASE) Project, available from: http://base.secureideas.net/. (Cited on pages 39, 71 and 93.)

Bashah, N., Shanmugam, B. and Ahmed, A. (2005). Hybrid Intelligent Intrusion Detection System, *Transactions on Engineering, Computing and Technology* **6**: 291–294. (Cited on page 9.)

Beale, J. and Caswell (2004). *Snort 2.1 Intrusion Detection*, 2nd edn, Syngress, United States of America. ISBN 1931836043. (Cited on pages 16, 44 and 52.)

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, 1st edn, Clarendon Press, United States of America. ISBN 0198538642. (Cited on page 30.)

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*, 1st edn, Springer, United States of America. ISBN 0387310738. (Cited on page 29.)

Bolzoni, D. (2009). *Revisiting Anomaly-based Network Intrusion Detection*, Ph.D disertation, University of Twente, Netherlands. ISBN 978-90-365-2853-5. (Cited on page 14.)

Bolzoni, D., Crispo, B. and Etalle, S. (2007). ATLANTIDES: An Architecture for Alert Verification in Network Intrusion Detection Systems, *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, Dallas, Texas, USA, pp. 141–152. (Cited on page 2.)

Bolzoni, D. and Etalle, S. (2006). APHRODITE: an Anomaly-based Architecture for False Positive Reduction, available from: http://arxiv.org/PScache/cs/pdf/0604/0604026.pdf. ISSN 1381-3625. (Cited on page 22.)

Bon, K. (2005). Signature-based Approach for Intrusion Detection, *Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science* **3587**: 526–536. (Cited on page 9.)

Brugger, S. and Chow, J. (2007). An Assessment of the DARPA IDS Evaluation Dataset Using Snort. (Cited on pages 45 and 54.)

Bugtraq (2010a). Microsoft IIS 5.0 "Translate: f" Source Disclosure Vulnerability, available from: http://www.securityfocus.com/bid/1578. (Cited on pages 48 and 52.)

Bugtraq (2010b). Microsoft IIS WebDAV HTTP Request Source Code Disclosure Vulnerability, available from: http://www.securityfocus.com/bid/14764. (Cited on page 48.)

Cannady, J. (1998). Artificial Neural Networks for Misuse Detection, *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98)*, Arlington, Virginia, USA, pp. 443–456. (Cited on page 32.)

Carlo, C. (2003). Intrusion detection evasion: How Attackers get past the burglar alarm, available from: http://www.sans.org/reading_room/whitepapers/detection/intrusion-detection-evasion-attackers-burglar-alarm_1284. (Cited on page 10.)

Carter, E. (2002). Intrusion Detection Systems, available from: http://www.ciscopress.com/articles/. (Cited on page 8.)

Caswell, B. and Roesch, M. (1998). Snort: The open source network intrusion detection system, available from: http://www.snort.org. (Cited on pages 4, 37, 38 and 71.)

Chapple, M. (2003). Evaluating and Tuning an Intrusion Detection System, available from: http://searchsecurity.techtarget.com/tip/1,289483,sid14gci918619,00.html. (Cited on page 15.)

Chyssler, T., Nadjm-Tehrani, S., Burschka, S. and Burbeck, K. (2004). Alarm reduction and correlation in defence of IP networks, *Proceedings of the 13th IEEE International Workshops on enabling technologies: infrastructure for collaborative enterprises (WETICE04)*, Modena, Italy, pp. 229–234. (Cited on page 63.)

CIS (2005). SOM Toolbox 2.0, available from: `http://www.cis.hut.fi/somtoolbox`. (Cited on pages 63, 64, 65, 71 and 92.)

CISCO (2010). CISCO Security Monitoring Analysis and Response System (MARS), available from: `http://www.cisco.com/en/US/products/ps6241/`. (Cited on page 9.)

Cox, K. and Gerg, C. (2004). *Managing Security with Snort and IDS Tools*, 1st edn, O'Reilly, United States of America. ISBN 0-596-00661-6. (Cited on page 13.)

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, 1st edn, Cambridge University Press, United Kingdom. ISBN 0521780195. (Cited on page 32.)

Cuff, A. (2006). Intrusion Detection Terminology (Part One), available from: `http://www.securityfocus.com/infocus/1728`. (Cited on page 2.)

Cuppens, F. and Miege, A. (2002). Alert Correlation in a Cooperative Intrusion Detection Framework, *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 202–215. (Cited on pages 20, 27, 34 and 57.)

Dain, O. and Cunningham, R. (2001). Fusing a heterogeneous alert stream into scenarios, *In Proceedings of the 2001 ACM Workshop on Data Mining for Security Application*, Philadelphia, PA, USA, pp. 1–13. (Cited on page 20.)

Debar, H. (2000). An Introduction to Intrusion Detection Systems, *Proceedings of Connect'2000*, Doha, Qatar. (Cited on page 12.)

Debar, H. and Wespi, A. (2001). Aggregation and Correlation of Intrusion-Detection Alerts, *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Davis, CA, USA, pp. 85–103. (Cited on pages 14, 20, 28, 34 and 57.)

Dineley, D. and Mobley, H. (2009). The greatest open source software of all time, available from: `http://www.infoworld.com/d/open-source/greatest-open-source-software-all-time-776?source=fssr`. (Cited on page 126.)

Dondo, M., Japkowicz, N. and Smith, R. (2006). AutoCorrel: a neural network event correlation approach, *Proceedings of SPIE, Data Mining, Intrusion Detection, Information Assurance, and Data Network Security*, Vol. 6241. (Cited on page 27.)

El-Hajj, W., Hajj, H., Trabelsi, Z. and Aloul, F. (2010). Updating snort with a customised controller to thwart port scanning, *Security and Communication Networks, Wiley InterScience*. (Cited on page 2.)

Enterasys (2010). Intrusion Prevention System (IPS) - Distributed Intrusion Prevention & Responses for Edge-to-Core and Data Centre, available from: `http://www.enterasys.com/company/literature/ips-ds.pdf`. (Cited on page 11.)

Flanagan, D. (2005). *Java in a Nutshell*, 5th edn, O'Reilly Media, United States of America. ISBN 0-596-00773-6. (Cited on page 92.)

Flexer, A. (1997). Limitations of self organizing map for vector quantization and multidimensional scaling, *Advances in Neural Information Processing Systems, Proceedings of the 1996 Conference, MIT Press*, Vol. 9, pp. 445–451. (Cited on page 61.)

Garcia-Teodoro, P., Diaz-Verdejo, J., Marcia-Fernandez, G. and Vazquez, E. (2009). Anomaly-based network intrusion detection: Techniques, Systems and Challenges, *Computers & Security* **28**(1-2): 18–28. (Cited on page 9.)

Goeldenitz, T. (2002). IDS - Today and Tomorrow, available from: `http://www.sans.org/reading_room/whitepapers/detection/ids-today-tomorrow_351`. (Cited on page 2.)

Grandison, T. and Terzi, E. (2007). Intrusion Detection Technology, available from: `http://www.almaden.ibm.com/cs/projects/iis/hdb/Publications/papers/ID.pdf`. (Cited on page 8.)

Greenwood, B. (2007). Tuning an IDS/IPS From The Ground UP, available from: `http://www.sans.org/reading_room/whitepapers/detection/tuning-ids-ips-ground_1896`. (Cited on page 49.)

Grunwald, P. (2005). A tutorial introduction to the Minimum Description Length principle, *In Advances in Minimum Description Length: Theory and Applications*, pp. 3–81. (Cited on page 127.)

Heady, R., Luger, G., Maccabe, A. and Servilla, M. (1990). The architecture of a network level intrusion detection system, available from: `http://www.cs.unm.edu/~treport/tr/90/tr.pdf`. (Cited on page 1.)

Herberlain, L., Dias, G., Levitt, K., Mukherjee, B., Wood, J. and Wolber, D. (1990). A Network Security Monitor, *IEEE Symposium and Security Privacy* pp. 296–303. (Cited on pages 8 and 10.)

Jan, N., Lin, S., Tseng, S. and Lin, N. (2009). A decision support system for constructing an alert classification model, *Expert Systems with Applications* **36**(8): 11145–11155. (Cited on pages 23, 34 and 60.)

Julisch, K. (2001). Mining Alarm Clusters to Improve Alarm Handling Efficiency, *Proceedings of the 17th Annual Conference on Computer Security Applications*, pp. 12–21. (Cited on pages 12, 20, 22, 25, 28, 29, 60 and 73.)

Julisch, K. and Dacier, M. (2002). Mining Intrusion Detection Alarms for Actionable Knowledge, *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 366–375. (Cited on pages 25, 28, 34 and 60.)

Kanellopoulos, I., Wilkinson, G., Roli, F. and Austin, J. (1997). *Neuro-computation in Remote Sensing Data Analysis*, illustrated edn edn, Springer, United States of America. ISBN 3540633162. (Cited on page 32.)

Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R. and Wu, A. (2002). An efficient k-means clustering algorithm: Analysis and implementation, *IEEE Transaction Pattern Analysis and Machine Intelligence* **24**: 881–892. (Cited on page 59.)

Kark, K. (2010). The changing threat landscape, available from: `http://www.csoonline.com/article/602313/the-changing-threat-landscape`. (Cited on page 1.)

Kayacik, H., Zincir-Heywood, A. and Heywoood, M. (2007). A hierarchical SOM-based intrusion detection system, *Engineering Applications of Artificial Intelligence* **20**(4): 439–451. (Cited on pages 31, 34 and 58.)

Khan, L., Awad, M. and Thuraisingham, B. (2007). A new intrusion detection system using support vector machine and hierarchical clustering, *The VLDB Journal - The International Journal on Very Large Data Bases* **16**(4): 507–521. (Cited on page 32.)

Khanchi, S. and Adibnia, F. (2009). False Alert Reduction on Network-based Intrusion Detection Systems by Means of Feature Frequencies, *International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 513–516. (Cited on page 73.)

Khosravifar, B., Gomrokchi, M. and Bentahar, J. (2009). A Multi-agent-based Approach to Improve Intrusion Detection Systems False Alarm Ratio by Using Honeypot, *International Conference on Advanced Information Networking and Applications Workshops*, pp. 97–102. (Cited on page 2.)

Kohonen, T. (1990). The Self-Organizing Map, *Proceedings of the IEEE* **78**(9): 1464–1480. (Cited on page 61.)

Kohonen, T. (1995). *Self-Organizing Maps*, 1st edn, Springer, Germany. ISBN 3540620176. (Cited on pages 30, 57, 58 and 92.)

Kohonen, T., Kaski, S., Lagus, K., Salojarvi, J., Honkela, J., Paatero, V. and Saarela, A. (2000). Self organization of a massive document collection, *IEEE Transactions on Neural Networks* **11**: 574–585. (Cited on page 58.)

Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map, *Proceedings of ECAI'94, 11th European Conference on Artificial Intelligence*, New York, USA, pp. 211–215. (Cited on page 58.)

Koziol, J. (2003). *Intrusion Detection with Snort*, 1st edn, Sams Publishing, United State of America. ISBN 1-57870-281-X. (Cited on page 13.)

Kruegel, C. and Robertson, W. (2004). Alert Verification: Determining the Success of Intrusion Attempts, *In Proceedings of the Detection of Intrusion and Malware and Vulnerability Assessment (DIMVA'04)*, Dortmund, Germany. (Cited on pages 21 and 45.)

Kumar, S. and Spafford, E. (1994). A Pattern Matching Model for Misuse Intrusion Detection, *In Proceedings of the 17th National Computer Security Conference*. (Cited on page 10.)

Labib, K. and Vemuri, R. (2002). NSOM: A real-time network-based intrusion detection system using self-organizing maps, *Technical report, Department of Applied Science*, University of California, Davis, USA. (Cited on pages 30 and 58.)

Laskov, P. (2007). Machine Learning for Intrusion Detection, available from: `http://langtech.jrc.it/mmdss2007/htdocs/Presentations/Docs/MMDSS_Laskov.pdf`. (Cited on page 12.)

Law, K. and Kwok, L. (2004). IDS false alarm Filtering using KNN classifer, *Lecture Notes in Computer Science, Fifth International Workshop on Information Security Applications (WISA'04)* , Vol. 3325, Jeju Island, South Korea, pp. 114–121. (Cited on pages 21, 23, 29 and 34.)

Li, Y., Fang, B., Guo, L. and Chen, Y. (2007). TCM-KNN Algorithm for Supervised Network Intrusion Detection, *Lecture Notes in Computer Science*, Vol. 4430, pp. 141–151. (Cited on page 29.)

Libeau, F. (2008). Automating security events management, *Network Security* **2008**(12): 6–9. (Cited on page 33.)

Lichodzijewski, P., Zincir-Heywood, A. and Heywood, M. (2002). Dynamic Intrusion Detection Using Self-Organizing Maps, *In 14th Canadian Information Technology Security Symposium*. (Cited on pages 31 and 34.)

Lincoln Lab (2010). DARPA Intrusion Detection Evaluation, available from: `http://www.ll.mit.edu/mission/communications/ist/CST/index.html`. date visited: 2 May 2006. (Cited on pages 37, 39 and 42.)

Lippmann, R. and Cunningham, R. (2000). Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks, *Computer Networks* **34**(4): 597–603. (Cited on page 33.)

Lippmann, R., Haines, J., Fried, D., Korba, J. and Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation, **34**(4): 579–595. (Cited on pages 15, 37 and 54.)

Lundin, E. and Jonsson, E. (2002). Survey of research in the intrusion detection area, *Technical Report 02-04, Department of Computer Engineering, Chalmers University of Technology*, Goteborg. (Cited on pages vii and 7.)

MacQueen, J. (1967). Some methods for classification and analysis of Multivariate Observations, *In Proceedings of fifth Berkeley Symposium on Mathematical Statistic and Probability*, Vol. 1, Berkeley: University of California Press, pp. 281–297. (Cited on pages 57, 59 and 67.)

Maggi, F., Matteucci, M. and Zanero, S. (2009). Reducing false positives in anomaly detectors through fuzzy alert aggregation, *Information Fusion* **10**(4): 300–311. (Cited on pages 20, 26, 34 and 57.)

Mahoney, M. and Chan, P. (2003). An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection, *Lecture Notes in Computer Science: In Recent Advances in Intrusion Detection (RAID 2003)*, Vol. 2820, pp. 220–237. (Cited on pages 37 and 45.)

MathWorks (2010). MATLAB Builder JA for Java language, available from: `http://www.mathworks.com/products/javabuilder/`. (Cited on page 92.)

McAfee (2010). 2010 Threat Predictions, available from: `http://www.mcafee.com/us/local_content/white_papers/7985rpt_labs_threat_predict_1209_v2.pdf`. (Cited on page 1.)

McHugh, J. (2000). Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, *ACM Trans. Information System Security* **3**(4): 262–294. (Cited on pages 37 and 38.)

McHugh, J., Christie, A. and Allen, J. (2000). Defending Yourself: The Role of Intrusion Detection Systems, *IEEE Software* **17**(5): 42–51. (Cited on page 2.)

McLain, C., Studer, A. and Lippmann, R. (2007). Making network intrusion detection work with IPSec. (Cited on page 13.)

Miles, R. (2006). *Learning UML 2.0*, 1st edn, Pragma, United State of America. ISBN 0596009828. (Cited on page 193.)

Miller, D., Harris, S., Harper, A., Vandyke, S. and Blask, C. (2010). *Security Information and Event Management (SIEM) Implementation*, 1st edn, McGraw-Hill Osborne, United States of America. ISBN 0071701095. (Cited on page 33.)

Necker, M., Contis, D. and Schimmel, D. (2002). TCP-Stream Reassembly and State Tracking in Hardware, *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)*, pp. 286–287. (Cited on page 11.)

Nicolett, M. and Kavanagh, K. (2008). Magic Quadrant for Security Information and Event Management. (Cited on page 33.)

Nilsson, N. (1996). Introduction to Machine Learning, available from: `http://ai.stanford.edu/~nilsson/MLBOOK.pdf`. (Cited on page 29.)

Ning, P., Cui, Y. and Reeves, D. (2002). Constructing Attack Scenarios through Correlation of Intrusion Alerts, *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, D.C., pp. 245–254. (Cited on pages 3, 20, 28, 34 and 57.)

Ning, P., Cui, Y., Reeves, D. and Xu, D. (2004). Techniques and tools for analyzing intrusion alerts, *ACM Transactions on Information and System Security (TISSEC)* **7**(2): 274–318. (Cited on page 22.)

Ong, J. and Abidi, S. (1999). Data Mining Using Self-Organizing Kohonen Maps: A Technique for Effective Data Clustering & Visualisation, *In International Conference on Artificial Intelligence (IC-AI'99)*, Las Vegas. (Cited on page 59.)

Paxson, V. (1999). Bro: A System for Detecting Network Intruders in Real Time, *Computer Networks* **31**(23-24): 2435–2463. (Cited on pages 14 and 29.)

Perdisci, R., Giacinto, G. and Roli, F. (2006). Alarm clustering for intrusion detection systems in computer networks, *Engineering Applications of Artificial Intelligence* **19**(4): 429–438. (Cited on pages 26 and 73.)

Pietraszek, T. (2004). Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection, *Proceedings of 7th Symposium on Recent Advances in Intrusion Detection (RAID'04)*, Vol. 3224, pp. 102–124. (Cited on pages 24, 30 and 34.)

Pietraszek, T. and Tanner, A. (2005). Data Mining and Machine Learning—Towards Reducing False Positives in Intrusion Detection, *Information Security Technical Report Journal* **10**(3): 169–183. (Cited on page 14.)

Porras, P. and Valdes, A. (1998). Live traffic analysis of tcp/ip gateways, *In Proceedings of the 1998 ISOC Internet Society Networks and Distributed Systems Security Symposium*, San Diego, CA. (Cited on page 12.)

Powers, S. and He, J. (2008). A hybrid artificial immune system and Self Organizing Map for network intrusion detection, *Information Science* **178**(15): 3024–3042. (Cited on pages 31, 34 and 58.)

Ptacek, T. and Newsham, T. (1998). Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, available from: `http://insecure.org/stf/secnet_ids/secnet_ids.html`. (Cited on pages 13 and 14.)

Raikar, A. and Ramarao, G. (2007). Method for configuring a network intrusion detection system, available from: `http://www.patentstorm.us/patents/7228564-description.html`. (Cited on page 16.)

Ramadas, M., Ostermann, S. and Tjaden, B. (2003). Detecting Anomalous Network Traffic with Self-Organizing Maps, *Lecture Notes in Computer Science, Recent Advances in Intrusion Detection*, Vol. 2820, pp. 36–54. (Cited on pages 30 and 58.)

Rhodes, B., Mahaffey, J. and Cannady, J. (2003). Multiple Self-Organizing Maps for Intrusion Detection, *In Proceedings of the 23rd National Information Systems Security Conference*, Baltimore, MD. (Cited on page 30.)

Rissanen, J. (1978). Modelling by shortest data description, *Automatica* **14**(5): 465–471. (Cited on page 127.)

Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks, *Proceedings of LISA '99, 13th System Administration Conference*, Seattle, WA. (Cited on page 15.)

Sadoddin, R. and Ghorbani, A. (2009). An incremental frequent structure mining framework for real-time alert correlation, *Computers & Security* **28**(3-4): 153–173. (Cited on pages 26, 34, 60 and 73.)

Sambamoorthi, N. (2003). Hierarchical Cluster Analysis: Some Basics and Algorithms, available from: http://www.crmportals.com/hierarchical_cluster_analysis.pdf. (Cited on page 32.)

Scarfone, K. and Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS), available from: http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf. (Cited on page 8.)

SecurityFocus (2010). SPADE (Statistical Packet Anomaly Detection Engine), available from: http://www.securityfocus.com/tools/1767. (Cited on page 9.)

Seung, S. (2003). Multilayer Perceptron and back propagation learning, available from: http://hebb.mit.edu/courses/9.641/2002/lectures/lecture04.pdf. (Cited on page 32.)

Shah-Hosseini, H. and Safabakhsh, R. (2000). TASOM: The Time Adaptive Self Organising Map, *Proceedings of International Conference Information Technology: Coding and Computing*, pp. 422–427. (Cited on page 116.)

Shin, M., Moon, H., Ryu, K., Kim, K. and Kim, J. (2003). Applying Data Mining Technique to Analyze Alert Data, *Lecture Notes in Computer Science, Proceedings of the 5th Asia-Pacific web conference on Web technologies and applications*, Vol. 2642, pp. 193–200. (Cited on page 29.)

Siraj, A. and Vaughn, R. (2005). A Cognitive Model for Alert Correlation in a Distributed Environment, *Lecture Notes in Computer Science*, Vol. 3495, pp. 218–230. (Cited on pages 26 and 27.)

Smith, D. (2006). A Practical Application of SIM/SEM/SIEM Automating Threat Identification, available from: http://www.sans.org/reading_room/whitepapers/logging/practical-application-sim-sem-siem-automating-threat-identification_1781. (Cited on page 11.)

Smith, R. (1999). The Web Bug FAQ, available from: http://w2.eff.org/Privacy/Marketing/web_bug.html. (Cited on page 43.)

Smith, R., Japkowicz, N., Dondon, M. and Mason, P. (2008). Using unsupervised learning for network alert correlation, *Lecture Notes in Computer Science, Proceedings of the Canadian Society for Computational Studies of Intelligence, 21st Conference on Advances in Artificial Intelligence*, pp. 308–319. (Cited on pages 60 and 119.)

Snort (2010a). ICMP L3Retriever Ping. (Cited on page 49.)

Snort (2010b). INFO web bug 1x1 gif attempt, available from: http://www.snort.org/search/sid/2925?r=1. (Cited on page 42.)

Snort (2010c). WEB-IIS view source via translate header, available from: http://www.snort.org/search/sid/1042?r=1. (Cited on page 48.)

Snort (2010d). WEB-MISC robots.txt access, available from: http://www.snort.org/search/sid/1852?r=1. (Cited on page 49.)

Song, D., Shaffer, G. and Undy, M. (1999). Nidsbench - A network intrusion detection test suite, *1999 Recent Advances in Intrusion Detection, Second International Workshop, RAID 1999*, West Lafayette, Indiana. (Cited on page 10.)

Sophos (2010). Security Threat Report 2010, available from: `http://www.sophos.com/sophos/docs/eng/papers/sophos-security-threat-report-jan-2010-wpna.pdf`. (Cited on page 1.)

Spathoulas, G. and Katsikas, S. (2010). Reducing False Positives in Intrusion Detection Systems, *Computers & Security* **29**(1): 35–44. (Cited on pages 22, 34, 57 and 73.)

Stergiou, C. and Siganos, D. (1996). Neural Networks, available from: `http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html`. (Cited on page 30.)

Symantec (2010). Symantec Global Internet Security Threat Report: Trends for 2009, available from: `http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xv_04-2010.en-us.pdf`. (Cited on page 1.)

Thomas, C., Sharma, V. and Balakhrisnan, N. (2008). Usefulness of DARPA dataset for intrusion detection system evaluation, *Proceedings of SPIE International Defense and Security Symposium*, Vol. 6973. (Cited on page 38.)

Timberline Technologies (2009). Alphabetical List of Intrusion Detection Products, available from: `http://www.timberlinetechnologies.com/products/intrusiondtct.html`. (Cited on page 9.)

Valdes, A. and Skinner, K. (2001). Probabilistic Alert Correlation, *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Davis, CA, USA, pp. 54–68. (Cited on pages 20 and 28.)

Valeur, F., Vigna, G., Kruegel, C. and Kemmerer, R. (2004). A comprehensive approach to intrusion detection alert correlation, *IEEE Transactions on Dependable and Secure Computing* **1**(3): 146–169. (Cited on pages vii, 20, 21 and 27.)

van der Heijden, F., Duin, R., Ridder, D. and Tax, D. (2004). *Classification, parameter estimation and state estimation*, 1st edn, Wiley, United States of America. ISBN 978-0-470-09013-8. (Cited on page 59.)

Viinikka, J., Debar, H., Me, L., Lehikoinen, A. and Tarvainen, M. (2009). Processing intrusion detection alert aggregates with time series modelling, *Information Fusion* **10**(4): 312–324. (Cited on pages 24 and 34.)

Vokorokos, L., Balaz, A. and Chovanec, M. (2006). Intrusion Detection System using Self Organizing Map, *Acta Electrotechnica et Informatica* **6**(1): 1–6. (Cited on page 31.)

WebDAV (2001). WebDAV Overview, available from: `http://www.kadushisoft.com/syshelp/webdav.htm`. (Cited on page 48.)

Weijters, T., den Herik, H. V., den Bosch, A. V. and Postma, E. (1997). Avoiding overfitting with BP-SOM, *In Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Vol. 2, pp. 1140–1145. (Cited on page 64.)

Wireshark (2010). Wireshark: The world's foremost network protocol analyzer, available from: `http://www.wireshark.org/`. (Cited on page 39.)

Xiao, Y. and Han, C. (2006). Correlating intrusion alerts into attack scenarios based on improved evolving self-organizing maps, *International Journal of Computer Science and Network Security* **6**(6): 199–203. (Cited on pages 31, 35 and 58.)

Yu, Z., Tsai, J. and Weigert, T. (2008). An adaptive automatically tuning intrusion detection system, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* **3**(3): 1–25. (Cited on page 25.)

Zaiane, O. (1999). Introduction to Data Mining, available from: `http://www.exinfm.com/pdffiles/intro_dm.pdf`. (Cited on page 28.)

Zhu, B. and Ghorbani, A. (2006). Alert Correlation for Extracting Attack Strategies, *International Journal of Network Security* **3**(2): 244–258. (Cited on pages 27, 32 and 34.)

Zoho (2007). Analyzing Logs For Security Information Event Management, available from: `http://www.manageengine.com/products/eventlog/Analyzing-Logs-for-SIEM-Whitepaper.pdf`. (Cited on page 9.)

Zurutuza, U. and Uribeetxeberria, R. (2004). Intrusion Detection Alarm Correlation: Survey, *Proceedings of the IADAT International Conference on Telecommunications and Computer Networks*, Donostia, Spain. (Cited on page 22.)