Improving the results of intrusion detection systems

Georgios Spathoulas

PhD thesis

University of Piraeus

Department of Digital Systems

Piraeus

Supervisor: Professor Sokratis Katsikas

(July 2013)

This PhD thesis of Georgios Spathoulas, entitled "Improving the results of intrusion detection systems" was examined and approved by the following examining committee:

This PhD Thesis has been authored on Latex (TexLive distribution), by using the TexMaker editor. The document Class used has been authored by Vasilios Plagianakos, Assistant professor of the Department of Computer Science and Biomedical Informatics of the University of Central Greece. The proposed system has been implemented in Java with Netbeans IDE. Matlab has been used for the visualization component of the system. Dia, Gimp and Weka software packages have been used for image production and editing.

To Giouli and Danae...

Abstract

Intrusion detection systems successfully detect intrusions, but the alert-sets they produce suffer from multiple deficiencies. The volume of alerts is difficult to handle, while the percentage of false ones is relatively high. The intruder's attack plan is difficult to be unveiled, as alerts correspond to low level events and the security analyst has to put in a lot of effort, in order to successfully monitor the security status of the protected system.

An alerts post-processing system is proposed to improve the results of intrusion detection systems. It transforms the alert-sets produced by multiple intrusion detection sensors to a meaningful live graphical representation, that can timely inform the analyst about occurring events and enable her to further examine these events and react accordingly. The system consists of sensor managers, each one of which is responsible for an intrusion detection sensor's alert flow. They calculate a validity estimation for each alert and aggregate identical alerts. Their outputs are all led to a single clustering subsystem. This merges these flows into a system-wide flow and commits the required clustering between relevant aggregated alerts. It optionally attempts to estimate information about events missed by the intrusion detection sensors. Finally a visualization subsystem produces a three dimensional live graph of existing clusters, in order to provide the analyst with a compact representation of occurring security events.

Along with the proposed system, an alternative method for false alerts filtering is discussed. It is based on fuzzy inference systems and efficiently evaluates the validity of alerts, eventually filtering out false ones. Finally a platform for conducting alerts post-processing experiments is presented. It provides users with standard ready to use functionality, while it enables them to reuse theirs or others past components.

Greek Abstract

Τα συστήματα ανίχνευσης παρεισφρήσεων ανιχνεύουν με επιτυχία πιθανές εισβολές, αλλά τα σετ συναγερμών που παράγουν χαρακτηρίζονται από σημαντικά προβλήματα. Ο όγκος των παραγόμενων συναγερμών κάνει δύσκολη την διαχείρισή τους, ενώ ένα μεγάλο ποσοστό τους είναι ψευδές. Το σχέδιο του εισβολέα δεν είναι εύκολο να εξαχθεί, καθώς οι συναγερμοί αντιστοιχούν σε χαμηλού επιπέδου πληροφορία και ο αναλυτής πρέπει να καταβάλλει σημαντική προσπάθεια, προκειμένου να παρακολουθεί επιτυχώς την κατάσταση ασφαλείας του συστήματος ύπο προστασία.

Στην παρούσα διατριβή παρουσιάζεται ένα σύστημα επεξεργασίας συναγερμών, με στόχο την βελτίωση των αποτελεσμάτων των συστημάτων ανίχνευσης παρεισφρήσεων. Μετά από την επεξεργασία των συναγερμών πολλαπλών αισθητήρων ανίχνευσης παρεισφρήσεων, το σύστημα παράγει μία ζωντανή γραφική αναπαράσταση των γεγονότων που έχουν ανιχνευθεί. Έτσι ο αναλυτής ενημερώνεται εγκαίρως σχετικά με τα γεγονότα αυτά και είναι σε θέση να τα εξετάσει περαιτέρω, όποτε αυτό κρίνεται αναγκαίο, και τελικά να αντιδράσει κατάλληλα. Το σύστημα αποτελείται από διαχειριστές αισθητήρων που είναι υπεύθυνοι για τις ροές συναγερμών που προέρχονται από τους αισθητήρες ανίχνευσης παρεισφρήσεων. Υπολογίζουν μια εκτίμηση ορθότητας για κάθε συναγερμό και ενοποιούν αυτούς που ταυτίζονται. Οι έξοδοι τους, οδηγούνται σε ένα κεντρικό υποσύστημα ομαδοποίησης. Αυτό συγχωνεύει τις πολλαπλές ροές συναγερμών σε μία ενιαία και ομαδοποιεί τους σχετικούς συναγερμούς. Προαιρετικά το υποσύστημα αυτό εκτιμά παραμέτρους σχετικά με γεγονότα που δεν έχουν γίνει αντιληπτά από τους αισθητήρες ανίχνευσης. Τέλος, ένα υποσύστημα οπτικοποίησης παράγει μία τρισδιάστατη γραφική παράσταση των ομάδων που παρήχθησαν, προκειμένου να παρέχει στον αναλυτή μία συνοπτική εικόνα των γεγονότων ασφαλείας.

Παράλληλα με το προτεινόμενο σύστημα, παρουσιάζεται και μία εναλλακτική μέθοδος αναγνώρισης ψευδών συναγερμών. Βασίζεται στην μεθοδολογία των fuzzy inference systems και αξιολογεί αποτελεσματικά την εγκυρότητα των συναγερμών, με σκοπό την απόρριψη όσων είναι ψευδείς. Τέλος, παρουσιάζεται μια πλατφόρμα για τη διεξαγωγή πειραμάτων σχετικά με την επεξεργασία συναγερμών. Η πλατφόρμα αυτή παρέχει στους χρήστες έτοιμα υποσυστήματα σχετικά με τυπικές επαναλαμβανόμενες λειτουργίες, ενώ τους δίνει και τη δυνατότητα να επαναχρησιμοποιούν υποσυστήματα που έχουν αναπτυχθεί στο παρελθόν από τους ίδιους ή από άλλους.

Acknowledgements

I cannot know if the Phd will reward me in the future for my efforts, but I can surely admit that I have already gained a lot through the whole process. Pursuing my Phd while having a full time job and creating a family in the meantime was one of the most demanding tasks of my entire life. The time available to read relevant literature, imagine my thesis contribution, implement my initial thoughts to then evaluate them as worthless, redesign methods from scratch, write thousands lines of code and finally write this thesis was always insufficient and never continuous. This difficult journey had enforced me to push myself multiple times, in order to stand up with various difficulties, continue to work when it seemed pointless or regain my self-confidence after journal rejections. I am coming out of this adventure feeling more confident, more capable and more poised both as a scientist and a man. This alone is a valuable reward for all my efforts as a Phd candidate.

In this journey I was not alone and the least I can do is to thank all the people that have helped me through these years. I had the luck and satisfaction to cooperate closely with my supervisor Pr. Sokratis Katsikas. Besides being a notable scientist he is an incredible person to work with and be dependent on. He has always been available and willing to help, advice or even just relax me. I would also like to thank Pr. Georgios Vasilakopoulos and Pr. Ioannis Vlahavas for their useful advices, suggestions and feedback on this thesis. Pr. Pericles Mitkas and Pr. Antonios Aletras as Presidents of Department of Computer Science and Biomedical Informatics of the University of Central Greece, have always allowed me to fulfill my obligations as a Phd candidate. I should also thank my colleagues in University of Central Greece, as they have covered for me in all cases I was absent or too focused on my Phd.

I cannot but feel gratitude to my parents for supporting me financially in pursuing my previous degrees but mainly for bringing me up and and more or less making me the person I am today. Above all I should thank my wife Giouli, who has been there for me through the whole time. She had always tried to take everything else of me, in order to enable me to focus on my Phd, even if this demanded from her to neglect her own career. I would not be writing this thesis today if it was not for her. After thanking all these people, I am obliged to apologize to someone. Its my daughter Danae who was born in July of 2012, just when the final and most demanding stage of my Phd was starting. Danae is the person to which the endless hours I devoted to my Phd through the last year originally belonged to. Now I have to give them back to her, starting from tomorrow...

Georgios P. Spathoulas

Piraeus, 2013.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Security has evolved in one of the most critical parameters for employing computer systems in everyday life. Growth of the internet along with its usability have necessitated for all information systems to be interconnected in the world wide web. This has transformed such systems into potential targets for an increasing population of intruders. On the other hand, the value of the information stored in the systems along with the significance of their uninterrupted operation have increased the probability of an intruder being able to benefit from breaking in systems that are not sufficiently protected.

One of the tools in the hands of security analysts, in order to secure computer systems, is intrusion detection systems, which detect possible intrusions and produce alerts in order to notify the analyst. There are multiple types of such systems, as the detection technique and the scope of protected system may vary. Host based systems protect critical hosts, while network systems can protect a whole network. Signature based systems use predefined intrusion profiles and try to match the activity of the protected system to them, while anomaly-based systems search for important deviations in the activity of the system and characterize these as intrusions.

An important problem in the intrusion detection field, regardless of the type of the used system, is the low quality of produced alert-sets, due to which intrusion detection systems may even become unusable. The volume of alerts is usually difficult to manage, while false positives and false negatives are always present. Generally alert-sets, as produced by intrusion detection systems, are hard for security analysts to utilize. The aim of this thesis is to design and implement a prototype system for post-processing of intrusion detection alerts, in order to produce a more qualitative representation of the security state of the protected system.

In this thesis a complete alert port-processing system is implemented. It accepts as input the alert-sets produced by multiple intrusion detection sensors and produces a real-time high level graphical representation of the security state of the protected system. It deals with all common deficiencies of intrusion detection alert-sets and aims to resolve most of the issues, that hinder intrusion detection systems from efficiently inform the analyst. The system consists of three subsystems, namely the Sensor manager, the Clustering subsystem and the Visualization subsystem.

The set-up of the system includes more than one Sensor manager subsystems. For each intrusion detection sensor used as input, a Sensor manager subsystem is responsible for managing the incoming alerts, for producing a validity score for each one of them and for committing the required aggregation to eliminate multiple identical instances of the same alert. The validity score is produced as a combination of four partial validity scores. There are four different components, each one of which examines the existence of a characteristic property of true or false alerts, for each record in the alert-set and produces a partial validity score. Consequently a weights formula is used to produce one final validity score for each alert. The calculated scores are attached to the alerts, which are then subject to an aggregation procedure, that replaces each group of identical alerts, triggered by the same event, with one, sufficiently informative aggregated alert.

The produced aggregated alerts are forwarded to the Clustering subsystem. There is a single such system that takes as input all the outputs of all the Sensor manager subsystems. Its first task is to merge these multiple alert flows into a single one. In order for this to happen the Merging component discards aggregated alerts that relate to the events already detected in other sensors' flows. The Merging component produces a unified flow of aggregated alerts, which is led to the Clustering component. The latter is responsible for detecting relationships between alerts and for grouping them into clusters that correspond to actual actions of the intruder. It is critical for the successful representation of the security state of the system, to approximate the level of detail of real actions. This can only happen if similarities between alerts are thoroughly examined to disclose any indications that more than one alerts may have been caused by the same event. The clustering procedure is periodically executed for all recent alerts. The operation of the Clustering subsystem can optionally be extended by the Clusters generator component, that detects cases where logical gaps between consecutive clusters exist. In such cases an occurred event is considered to have been missed and the component tries to approximate information about it. This estimated information is formed as an additional cluster of alerts which is then embedded into the real clusters list, in order to notify the analyst that there may be undetected activity, that she should investigate about. The final list of clusters produced by the subsystem at each execution is stored in a database.

The third subsystem is the Visualization subsystem, that recalls formed clusters from the database and depicts them on a three dimensional space. The Visualization subsystem is also functioning repetitively, and produces consecutive images, which are then used as frames to form a live representation of the security state of the protected system. At each run the current list of clusters is read from the database. The parameters of the clusters are normalized, to enable the subsystem to produce a comprehensive depiction. The three axes of the produced graph represent time, IP values of the protected network and a calculated danger value. The security state is depicted by a surface parallel to the plane defined by the intersection of time and IP axis. For every cluster a peak which is indicative of its parameters is formed on the surface. The resulting graph is informative for the analyst, while being simple enough to enable her timely reaction when a dangerous event is spotted.

Experiments have been carried out to justify the efficient operation of the system. Two different datasets have been used and the performance of all components has been studied. In both cases the system managed to vastly reduce the volume of alerts by producing a manageable final number of clusters. Most of the false alerts have been successfully detected, therefore misleading clusters relevant to nominal events were filtered out during visualization, in order not to hinder the reading of the resulting graphs. The number of clusters produced for each real intrusion event is very small; this is a great improvement with respect to multiple alerts, initially triggered by the event. The plan of the intruder is easily recognizable through the produced clusters list. Regarding missed event approximation, the relevant testing scenarios have shown that the system can efficiently inform the analyst about missed events, without overwhelming the clusters list with artificial data. Finally, the resulting visualizations are illustrative and provide sufficient data to the analyst, to keep her informed about security events, without much effort.

Additionally to the analysis of proposed system, an alternative false alerts detection methodology is presented. It incorporates a fuzzy inference system, as it is assumed that the claims about certain characteristics of alerts and their validity are not definite but may hold true to some extent. By the use of if-then rules these claims are correlated and a validity estimate for each alert is finally produced. The method is able to vastly reduce false positives, while it filters in alerts for all events detected by the intrusion detection system.

Finally a platform for conducting intrusion detection alerts post-processing experiments has been designed and implemented. Researchers working in the area usually face difficulties when implementing the methods they propose, to prove their validity. The proposed platform enables them to build a complex alerts processing solution on a component-by-component basis. They are provided with ready-to-use components for typical tasks, such as reading alerts from an intrusion detection system or calculating metrics of the performance of their solution. Apart from that, the components are reusable, so the user can easily embed functionality she has already implemented in a new method of her. Additionally users can exchange their components just for comparison or within the context of research collaboration.

CHAPTER 2

Intrusion Detection Systems and their Limitations

Security is one of the most important factors in computer systems applications. Examples of recent security breaches are discussed, to manifest this importance. Intrusion detection systems are analysed and their main taxonomies are presented. They can either monitor traffic of a protected network or events in a protected host. Furthermore the detection technique they employ may vary. Finally the most important limitations regarding the output of intrusion detection systems are presented, to stress out the necessity of an alternative scheme for presenting alerts.

2.1 Computer security importance

Internet extraordinary growth in the last fifteen years has completely altered the way people work, get informed or even have fun. It is present in all aspects of everyday life. The majority of electronic devices is designed to connect to the world wide web and the users of the Internet have exponentially increased. Figure 2.1 shows the increase of the number of internet users in the last fifteen years [1], [2], [3]. This number was 147 millions in 1998, which was about 3.6% of world population. Since then it has been steadily increasing and in 2012, after fifteen years it has become 2,497 millions, which corresponds to 35.7% of the world population.

The growth of the internet user base is obvious and is mainly due to the fact that internet offers the ability to easily do things, that otherwise would be difficult or even impossible. The world wide web is a shared resource used by many people, applications or organizations representing different interests. It is being used by competing businesses, mutually antagonistic governments, and opportunistic criminals. Unless security mea-

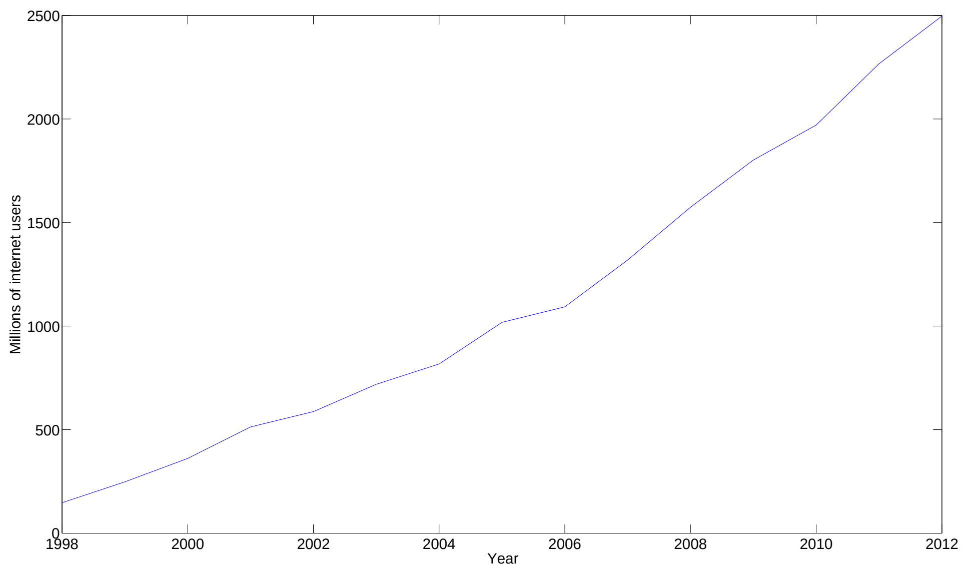

Figure 2.1: Internet users growth in the last fifteen years

sures are taken, a network conversation, a distributed application or a miss-protected system may be compromised by an adversary [4].

People and even businesses enthusiastically employ internet technology, without taking the required measures to protect themselves. Usually security becomes a major concern after a security incident happens. This can cost the privacy or integrity of a user's data or a lot of money, when it regards enterprise systems.

Security breaches make the news on a regular basis: incidents in which the security of a company or a government agency is breached, leading to loss of information, personal records, or other data. There are many ways to measure the size or cost of a security breach. Some result in the loss of millions of data records, some affect millions of people, and some wind up costing the affected businesses a lot of money [5]. Some representative security breaches are briefly discussed below [6].

In June 2005, MasterCard announced that up to 40 million credit card holders were at risk of having their data stolen, and 200,000 definitely had, because of a Trojan on the computers of a credit card processing company. The processor, CardSystems Solutions, had improperly stored the card data, unencrypted, in order to do research on the transactions.

In February 2007, TJX, a discount stores company, disclosed that thieves had stolen information on possibly tens of millions of credit and debit cards. The systems of the company had been compromised for about twenty months. The incident wound up costing TJX millions of dollars and eventually eleven hackers were arrested for the break-in.

The grocery store chain Hannaford Brothers announced in March 2008 that hackers had gained access to more than 4.2 million credit card transactions. By the time word got out, more than 1,800 of the credit card numbers had already been used at the stores of the company.

Heartland was a credit card payment processor for more than 250,000 businesses in 2009, when the company revealed that tens of millions of transactions might have been compromised. The computers of the company were infected with malware that passed information on to outsiders, that would enable them to create counterfeit cards with actual user data. The company claimed that Social Security information, PIN numbers, and other personal data were not affected.

In 2009 in an act of industrial espionage, the Chinese government launched a massive and unprecedented attack on Google, Yahoo, and dozens of other Silicon Valley companies. The Chinese hackers exploited a weakness in an old version of Internet Explorer to gain access to internal network of Google.

The worst gaming community data breach of all-time happened in April of 2011 on the network of Sony Play-station. Of more than 77 million accounts affected, 12 million had unencrypted credit card numbers. According to Sony the source of the hack was never found. Whoever they are gained access to full names, passwords, e-mails, home addresses, purchase history, credit card numbers, logins and passwords.

Throughout 2012 there were again multiple cases, in which major breaches were in the first lines of the news [7].

Wyndham Hotels repeatedly left their database vulnerable to hackers over the course of a few years. It seems that Wyndham failed to utilize industry-wide best practices such as using complex passwords and user IDs or encrypting customer credit card data. Because this information was stored in an improperly secured, centralized data center, hackers were able to easily install phishing software. Furthermore when the company discovered the security lapse, they did not make changes to their security procedures and the hacking continued for years.

In mid-January, Zappos was the victim of a cyber-attack by a criminal who gained access to parts of its internal network and systems. The hackers had managed to compromise over 24 million records which included user names, phone numbers, email addresses, partial credit card numbers, and encrypted passwords. After reviewing the incident, it became clear that Zappos had followed proper security protocols, such as salting their encrypted passwords, and the aftermath was little more than a need for customers to change their passwords.

Not long after Zappos security breach, both LinkedIn and eHarmony had their users passwords published to code cracking forums, presumably by the same hacker. Users were notified via email and urged to change their passwords immediately. Because the posted passwords were quickly brute-forced by security experts, and presumably hackers, it became clear that the companies did not properly hash and salt the encrypted passwords.

In mid-2012, hackers had exploited security of Last.fm to make off with millions of user passwords. The incident was discovered when the passwords were again dumped on a hacking forum. After going public with this incident, the original developer of the company, Russ Garrett, claimed that he was responsible for failing to institute proper password encryption. Additionally, he admitted that the security protocols had not been updated since the site was originally coded in 2003.

Finally hackers broke into a Yahoo sub-domain by sending commands through an inadequately secured URL and managed to steal files from Yahoo's Contributor Network. In total, the files stolen contained about 450,000 user names and passwords. Shockingly, these files were not encrypted and were instead stored in plain text.

All these incidents indicate that taking the required countermeasures against security threats is crucial and failing to do so may expose a system to great danger. As attacks become more sophisticated, tools in the hands of security analysts become more complex. One of these tools is intrusion detection systems, the performance of which is enhanced by the system proposed in this thesis.

2.2 Intrusion detection

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices [8].

The first research effort in intrusion direction is an an audit trail-based intrusion detection system in 1980 [9]. The author tried to detect unauthorized access to files by examining audit trails. His methodology was similar to current host based intrusion detection systems. Since then a lot have changed regarding the requirements that an intrusion detection system should fulfil and such systems have evolved significantly through time.

Intrusion detection systems aim to produce one alert for each abnormal activity happening on the protected system. Such activity may either be an external intrusion to the system or an internal misuse. Real world implementations cannot perform ideally, but try to approximate this aim. The truth is that not all security incidents are monitored, while there are alerts produced with no real corresponding suspicious activity. When designing an intrusion detection system the aim is to maximize the percentage of security incidents for which alerts are produced, while keeping as low as possible the percentage of false alerts (alerts for which no real event occurs).

2.3 Intrusion detection systems taxonomies

The variety of intrusions to be detected has enforced the use of different intrusion detection systems. They are categorized, according to both the detection method they use and the kind of data they monitor [10].

2.3.1 Taxonomies based on monitored data

The main categorization of intrusion detection systems is based on the data they manipulate, in order to detect intrusions; the resulting categories are host-based systems and network-based systems.

Host-based systems

Host-based intrusion detection systems monitor events happening in a single host. They are usually installed on critical hosts, on which valuable data is stored or important services are run. They examine network traffic only for the host they are installed on along with other valuable information found in the host's logs, running processes, file access, file modification and configuration changes [8].

Host-based systems were the first systems to be used. As the years passed, the focus of computing shifted from mainframe environments to distributed networks of multiple computing machines communicating to each other [11]. This made host-based systems inadequate for protecting modern complex systems on their own, but they remain integral parts of an organization's security infrastructure.

Network-based systems

Network-based intrusion detection systems monitor network traffic for the network of an organization or for a particular network segment of it. They analyse the network and application protocol activity to identify suspicious events [8]. They are capable of detecting attacks from the network traffic that these produce, so if a host has already been compromised and the activity of the intruder is limited to this host no alerts will be produced. On the other hand most attacks are network-oriented so network-based systems are the appropriate monitoring tool to detect them.

They are usually deployed at the boundaries between different segments of networks, such as border firewalls or routers. Depending on the size and the segmentation of a network, multiple sensors may be required, to sufficiently monitor the whole system.

2.3.2 Taxonomies based on the detection methodology

There are mainly two approaches to the way an intrusion detection system processes data to find indications of suspicious behaviour. One is misuse detection and the other is anomaly detection.

Misuse detection

In the misuse detection scheme the intrusion detection system employs patterns that correspond to known attacks. The system continuously compares data generated by current activity against these patterns and when a match is found the system generates an alert the details of which depend on the pattern itself and on the data matching it. Patterns are often referred as signatures and the relative systems are characterized as signature-based.

Generally these systems demonstrate good performance on detecting known attacks. Normally attacks, the pattern of which exists in the signatures of the intrusion detection system, are successfully detected. The syntax of the signatures must be flexible enough to match as many transformations of the attack as possible.

On the other hand such systems cannot detect any attack not existing in their signatures list. Even if this list is as fully populated as possible, zero day attacks will always pass by undetected.

Anomaly detection

The anomaly detection approach is characterized by a different logic. Instead of looking for known attacks, a model of normal activity is utilized. Anomaly-based systems continuously compare current activity to this model and when a significant deviation is spotted they generate an alert. Details of this alert depend only on the current activity, as no specific pattern occurs.

Unlike misuse detection systems, these perform better in detecting new attacks as the data they produce is probably different from what has been modelled as normal activity. Thus anomaly detection systems should be preferred if the protected system may be the target of new attacks.

On the other hand because of their way of functioning anomaly detection systems produce too many false alerts. Every activity happening for the first time in the protected system triggers an alert, regardless if it is suspicious or not. So whenever the behaviour of normal users becomes different from what it usually is a false alert is produced.

2.4 Intrusion detection systems limitations

Intrusion detection systems, while providing important information about intrusions that occur in a system, they do not allow the best possible reaction to them. This is because the output of the systems is usually illegible lengthy alert-lists, while an important percentage of alerts they contain are false, as they do not correspond to real events. The major deficiencies of the output of intrusion detection systems output are analysed in this Section [12] [13] [14].

The produced alert-sets are usually large in size, even for small protected systems. In cases of larger protected systems, the volume of the produced alerts set makes it impossible to monitor the security state of the protected systems, through tedious reading of the produced list.

Security events usually cause multiple identical alerts. This happens either because of their nature or because of their time duration. The resulting alert-set is largely characterized by alerts containing overlapping information. In cases where there are more than one installed sensors in a system, it is obvious that when events are perceived by multiple sensors, the corresponding alerts will appear more than once in the final consolidated alert-set.

The validity of alerts is one of the major problems of intrusion detection systems. As the failure of detection of an ongoing attack can be disastrous, the systems are usually configured to be too sensitive. This reduces the chances of not detecting real attacks, but also significantly increases the number of the produced false alerts, as more nominal events are considered as intrusions.

The produced alerts correspond to low-level events. The ideal level of detail for the analyst would be proportional to the level of detail of the actions of the intruder. However, due to functioning of intrusion detection systems, each of the actions of the attacker usually causes multiple alerts. So even if the actions of the intruder are detected by the intrusion detection system, her exact plan is difficult to be detected by the analyst by inspecting the list of generated alerts.

The presentation of alerts in long lists is by itself, even without the aforementioned problems, an obstacle to the analyst in her attempt to identify security incidents as early as possible and to immediately react to them. A brief presentation format of the detected events is necessary for the analyst, in order to take the required actions in time.

Overall, it is obvious that the knowledge produced by intrusion detection systems is not presented in the format it should be. While the actual security incidents may be detected by the systems, the way the data is displayed is not efficient enough. The analyst may identify a detected intrusion too late or even not identify it at all, because of the low quality of the presentation scheme.

CHAPTER 3

Relevant work

Relevant work of others is presented in this chapter. The most influential research efforts in intrusion detection alerts post-processing field are presented. Consequently the most recent work is summarized and presented in different sections. The first section discusses false positives reduction methods proposed in the last years. The next section presents multiple alerts correlation methods based on various approaches to the problem of high alerts' volume. Finally methodologies relevant to both hypothesizing about missed events and visualization are analysed. In both of these cases the relevant papers are significantly less than the first two.

3.1 Introduction

As soon as intrusion detection research matured and the first real world implementations of such systems were installed, the problem of the low quality of the resulting alert-set became evident. In theory intrusions were detected at a high rate, but the alert-sets produced by the intrusion detection systems were inappropriate for use in an environment where instant reaction is critical. Security analysts had to dig through huge alert-sets of low quality to find indications of intrusions. The difficulty in deciding whether an intrusion was really occurring made them to either react with delay or even to not react at all to real intrusions that were indeed detected by the intrusion detection systems.

So in the last two decades serious research work on post-processing of alerts has been conducted, in order to improve their quality. In next Section the most important research efforts in the field are presented. The most recent research efforts are categorised and presented in the rest of the Sections.

3.2 Early important work on post-processing of intrusion detection alerts

3.2.1 Defining alerts similarity

In 2001 Valdes and Skinner proposed using probabilistic similarity between alerts as a means to post-process them [15]. To this end, they defined a method for calculating similarities. They calculate similarity between two alerts as a meter of overlapping between their features. Since then many methods that use this similarity approach or try to enhance it have been proposed.

In this approach alerts for which there is a relevant match are aggregated. For each different alert attribute an appropriate similarity function is defined. Additionally, an expected similarity value is calculated, which in practice is a weight that is later used to calculate the overall similarity. A minimum match specification is also incorporated, that unconditionally rejects a match if any feature similarity is lower than the minimum specified value. For each new alert, the similarity to all existing meta alerts is computed taking into account attribute similarities along with the corresponding expected similarities. The alert is then merged with the best matching meta alert, as long as their similarity is above a threshold value.

An experimental process has been conducted, with a simulated real world network. Normal traffic was artificially generated and at the same time the designed attack was executed. The intrusion detection sensors used were EMERALD eBayes and eXpert-Net. The correlation procedure has achieved a reduction of false alerts at one-half to two-thirds with regards to the initial alert-set.

The concept of combining results of similarity functions for each attribute of alerts, to calculate an overall similarity has influenced the work of other researchers [16] [17] [18].

3.2.2 Discriminating between aggregation and correlation

At about the same time, Debar and Wespi [19] presented the first analytical descriptions of alert aggregation and correlation procedures. They discuss an overall intrusion detection post-processing architecture and their well defined approach remains valid until today, as most methods after theirs have discriminated between aggregation and correlation.

They highlight the most important problems in intrusion detection alert-sets as:

- Intrusion detection systems provide the operator with a large number of alerts; the operator then has difficulties coping with the load.
- Attacks are likely to generate multiple related alerts and it is not easy for operators to logically group them.
- Intrusion detection systems are likely to generate many false alerts, false positives or false negatives.
- Intrusion detection system architectures, at the time, made it difficult to achieve large-scale deployment.

In order to solve these problems, the authors proposed an architecture that consists of multiple detection probes, the outputs of which are fed to aggregation and correlation components.

In the aggregation component the algorithm groups events together according to certain criteria. The aim is to discard multiple identical alerts at the sensor level.

In the correlation component the algorithm creates correlation relationships between related events according to explicit rules. Once events are in a relationship, they are considered as part of the same attack and are processed together. The authors define two kinds of correlation relationships between events: duplicates and consequences.

The detection of duplicates relies on the provision of common information by different intrusion detection sensors. Duplicates are alerts referring to exactly the same event. Consequence chains are sets of alerts linked in a given order, where the link must occur within a given time interval. Consequences are alerts that correspond to consecutive related alerts.

A usage example is given, but thorough experiments are not carried out.

3.2.3 Reconstructing attack scenarios

In [20] the motivation is to provide a framework for constructing attack scenarios through alert correlation, using prerequisites and consequences of intrusions. The approach is based on the observation that alerts correspond to different stages of an attack scenario, with the earlier stages preparing for the later ones. The same idea has been reused in recent years in many research efforts [21] [22] [23].

The authors proposed a formal framework to represent alerts along with their prerequisites and consequences, and developed a method to correlate related alerts. In this framework they define hyper alerts types, which are composed by the intrusion type, the prerequisites and the post conditions of the intrusion. The prerequisites of an intrusion are the necessary conditions for the intrusion to be successful, while the consequences of an intrusion are its possible outcomes. They also developed an off-line tool on the basis of the formal framework, which tries to correlate alerts, by combining post conditions with prerequisites. Specifically, the tool examines each alert and tries to discover possible combinations of its post conditions with the prerequisites of alerts with timestamps in a specific time window which comes later than the time stamp of the alert being examined.

The authors conducted experiments that demonstrated the potential of their method in correlating alerts. While the method is based on manually defining prerequisites and post conditions for all possible attack types, a fact that reduces flexibility and ability to deal with new attack types, the idea of connecting alerts in a logical and chronological manner was important for later post-processing of alerts research.

3.2.4 A complete approach

Perhaps the most influential work in post-processing of alerts was presented in [24]. The authors in [24] have implemented a complete system that tackles most aspects of post-processing of alerts and have conducted experiments on different data-sets to prove the validity of their assumptions.

They provided a detailed analysis of the problem and designed a set of components that focus on different aspects of the overall correlation task. First, a normalization component transforms all alerts to a standardized format, understood by all correlation components. Next, a preprocessing component deals with attributes of alerts that sensors may have omitted and supplies relevant values, as accurate as possible. These attributes may be required for the functioning of other components in sequel. The fusion component is responsible for combining alerts that represent independent detections of the same attack instance by different intrusion detection systems. In a system with multiple sensors, identical alerts may be an important problem to solve. A verification component determines the possible success of the attack which each alert corresponds to; this information is used by the correlation components down the line. Verification is performed either by using passive techniques, such as gathering data for the network in advance, or by using active techniques, such as looking for attack success evidence after the alert has been recorded.

The thread reconstruction component identifies combinations of attacker and target through all alerts, in order to discover series of alerts that refer to attacks launched by a single attacker against a single target. This component is important as it can associate network-based alerts with host-based alerts, both related to the same attack. The task of the focus recognition component is to identify hosts that are either the source or the target of a significant number of alerts. These hosts are likely to be related to a denial-of-service attack or to a port scanning attempt. The multistep correlation component identifies predefined common attack patterns such as island-hopping attacks. Finally, the impact analysis component determines the impact of the detected attacks on the operation of the network being monitored; this information is eventually used by the prioritization component, which assigns an appropriate priority to every alert.

Besides designing a robust system, the authors extensively researched data-sets available at the time, and utilized all of them to experimentally test their system.

3.3 Reducing false positives

Intrusion detection alert-sets are characterized by high false positives rate. Various numerous methods have been proposed to cope with this problem. In this section the latest research efforts to reduce false positives in intrusion detection are presented.

3.3.1 Considering initial classification inadequate

the author in [25] suggests that the actual intrusion detection systems are inadequate, as he proves that checking the TTL of packets that produced the alerts helps reducing significantly the false positives rate. The proposed method is based on clustering the produced alerts on the basis of their TTL values. Experiments on various data-sets show that false positives are included in specific clusters and that it is then easy to discard them. This work has been extended in [26], whereby apart from false positives, redundant alerts are also filtered out through clustering procedures.

3.3.2 Looking at neighboring alerts

The authors in [27] take advantage of mixture models in order to discriminate between true and false positives. They compare the characteristics of each alert to the characteristics of previous ones. It is expected that a true alert will differentiate from its precedents. In this way the alert is classified to the intrusion or to the non-intrusion set. Additionally the protected system is checked for vulnerabilities relevant to each alert, in order to characterize it as critical or not.

While the method seems interesting, it is not well defined and the reader is not convinced of its validity.

Thw authors in [28] also examine neighboring alerts, to decide on the validity of each alert. They calculate the relevant correlation, and try to identify false alerts along with duplicate ones (alerts that tend to reappear multiple times through the dataset). Apart from that, they also use an ensemble-based adaptive learner which, given the expertise feedback, is capable of adapting to environmental changes through automatic tuning. The learner remains effective even if the protected network changes. The implementation is tested by using both the DARPA and a private data-set. The method requires the intervention of the security analysts; this makes it inapplicable in real world, large scale networks.

The alert-set itself is also utilized in [29]. The method proposed therein is based on the calculation of reputation for alerts. The reputation relates to the probability that these alerts are true. It is calculated from the false positives rate of alerts concerning the same IP address or sharing the same signature. The performance of the method is validated through experiments that show significant reduction in the false positives rate. The limitation of the method is that in order to calculate the reputation, the validity of previous alerts needs to be known.

3.3.3 Training required

The authors in [30] propose the use of an adaptive false alert filter that incorporates some of the most common machine learning techniques, such as K-nearest neighbors; Decision trees; and Support vector machines. The results of each method are continuously monitored. The filter examines each algorithm's performance every hour and chooses the best of them to be used until the next evaluation. The filter seems to perform well, but there is an important drawback, as the comparison of the algorithms' performance is based upon the labeled Snort data-set (the validity of alerts can be determined from the DARPA documentation). In a real world scenario no such labels exist.

In [31] the main assumption exploited is that false positives are triggered by causes that are frequent in a specific network. A training phase is described, where the frequency of values for attributes of alerts are calculated. These frequencies are stored in hash tables. They are normalized by weight values and used to determine if a future alert is similar to alerts that frequently appear or if it is significantly different from them. The threshold value that is used to discard frequent alerts is also decided upon during training phase.

3.3.4 Considering attacks as anomalies

In [32] valid alerts are considered as anomalies in an alert-set mainly consisting of false positives. The authors built profiles of usual false positives, for a given protected network. The idea is to use anomaly detection techniques on the produced alert-set, to discriminate true alerts as alerts that are characterized by an important deviation from these profiles. Three different algorithms are used and compared in terms of achieved false positives reduction, given the fact that they do not filter out any true positive. The rationale of this paper seems interesting and promising, but the choice of the threshold that discards alerts would be difficult to make in a real world scenario, where no evidence regarding the validity of the alerts exists.

The same idea is more or less found in [33]. The authors therein state that there are root causes for each group of similar alerts. If the protected system's administrator can discriminate root causes relevant to intrusions from the ones relevant to benign activity, then she could easily reduce the false positives rate by discarding the alerts produced by non intrusive root causes. They propose to use clustering in order to create clusters of similar alerts and then to characterize each of these clusters according to an assumed root cause. Future alerts can be characterized by the cluster they are closer to. An obvious drawback of the proposed system is that it is semi-automatic, as it will always require human intervention in order for root causes to be discovered and characterized as nominal activity or intrusion.

In [34] the idea is to reduce the overall number of alerts and by doing so to reduce the false positives rate. K-means clustering is used to identify main clusters in a huge population of alerts for a specific network, while outliers are ignored. The authors state that if a future alert could be categorized into one of these clusters it would be a strong indication that the specific alert concerns nominal traffic. Upon this hypothesis, they propose to completely ignore such alerts. While they provide proof for the high percentage of alerts that could be ignored in this way (resulting in a much smaller alert-set), they do not provide enough evidence on the validity of their assumption, that the ignored alerts are indeed relevant to nominal traffic.

3.3.5 Getting feedback

A general framework is proposed in [35] that enables the feedback of false positives occurring in results to be fed back into the monitoring process. This way the policy implemented by the intrusion detection system can be altered accordingly to the false positive rate. The framework proposed therein is a solid base on which methods for reconfiguring monitoring policies can be based, but there are important issues that should be addressed in order for the framework to be used in a real world scenario. Specific adjustments should be made to the framework itself as it is strongly coupled to the intrusion detection system used, the procedure chosen for checking the validity of alerts and the method for evaluating the performance of the monitoring policy in use.

3.3.6 Working under uncertainty

The authors in [36] deal with the aggregation of alerts, in order to reduce false positives, but in terms of anomaly detection systems. Alerts, produced by signature based systems, contain useful information for alert correlation methods, such as the relevant signature or the class of the attack. On the other hand, anomaly detection systems produce less information for the attack and the correlation process is more difficult. In this paper the use of fuzzy sets is proposed, in order to avoid missing alerts due to fuzziness issues. The main information to base aggregation in anomaly-based systems is timestamps. Problems may appear in this aggregation procedure due to system latency or wrong sharp threshold values. The authors discuss the criteria required to compute the time distance between alerts and to define threshold values by taking into account the uncertainty factor. The general intrusion detection feasible data-set problem is thoroughly discussed and a framework for evaluating fusion methods is presented.

3.3.7 Considering network's vulnerabilities

The authors in [37] propose to filter out false positives by taking into account the vulnerabilities of the protected systems. They assume that every alert that concerns an ineffective, to the protected network, attack can be treated as a false positive. Even if there is an ongoing intrusion attempt, it will be unsuccessful as the required vulnerability is missing. The proposed method uses vulnerability scanners to create profiles of existing vulnerabilities. Alerts are then correlated to these profiles and the resulting distance vectors are used to classify each alert as true or false. A back-propagation neural network has been trained on vectors that belong to alerts known to be ineffective. Then, this neural network classifies new alerts as effective or ineffective, filtering out the latter ones. The experimental results on a custom data-set indicate that the accuracy of the intrusion detection system has been vastly improved.

3.4 Correlating alerts

Perhaps the most research-intensive field in intrusion detection post-processing is the correlation of alerts. Due to the multiple feasible approaches to the problem, a large volume of relevant papers exist in the literature. In this section we present the most recent works in the field. Some authors extend work that was done previously, while others propose more innovative solutions.

3.4.1 Methods in their early stages

In [38] a simple alert clustering scheme is proposed, in order to reduce the number of alerts. The alerts' attributes that are examined are the alert's signature; the destination IP; and the time stamp. While the clustering algorithm used is not described in detail, each produced cluster is tagged with an attack type at the end of the procedure. This research work seems to be in progress hence, criticizing its results is immature.

An iterative clustering procedure is presented in [39]. The ISODATA algorithm creates clusters of alerts in an iterative manner. The functioning and the finalizing circumstances of the algorithm are based on arbitrarily valued parameters. The experimental results indicate a reduction of the number of alerts, albeit without justifying the validity of the produced aggregations. The method does not seem to contribute much to recent intrusion detection research.

The methodology described in [40], aims to aggregate intrusion detection alerts in a performance-efficient manner, in order to be applicable in an on-line scenario. The authors regard attack instances as random processes producing alerts and they try to model these processes, using approximate maximum likelihood parameter estimation techniques. While they provide a general description of their theory, there is no analysis of the proposed method.

3.4.2 Using correlated alerts to reduce false positives

The authors in [41] have incorporated the results of correlating alerts, to reduce false positives. They use the Self Organising Maps algorithm to create clusters of alerts, triggered by the same security events, in an unsupervised manner. After that, they use the K-Means algorithm to further classify the clusters as true or false. The output of the SOM algorithm is fed as input to the K-means algorithm. The experiments carried on both the DARPA data-set and on a private University of Plymouth data-set prove that two stage clustering is efficient. Another advantage of the proposed method is that the graphical representation of clusters produced by the SOM algorithm may be representative of the relations between correlated alerts.

In [42] frequent itemset mining is used in order to discover alerts that are frequent and characterize them as normal. Alerts that look like unusual (interesting) are isolated and promoted as possible intrusions. The authors have collected a private alert-set; a thorough examination of the properties of these alerts has indicated that strong recurring patterns exist in the alert-set. The frequent itemset mining algorithm has been adapted to address problems of the specific domain. For example, unusual and intensive short-term malicious network activity may produce too many alerts and thus trigger relative patterns that will classify similar future activity as normal. On the other hand, too generic produced patterns may be inappropriate for alert classification due to over-generalization issues. The experimental results showed that the classification of alerts as interesting has achieved high rates of precision, as most of the intrusion-relevant alerts were in that category, while the size of the alert-set was substantially reduced.

3.4.3 Socialization between intrusion detection nodes

An interesting approach for optimising collaboration among intrusion detection nodes is presented in [43]. It is based on the assumption that each node can communicate with other nodes and it can appraise their trustworthiness. In this way, each node can get information about ongoing attacks from other nodes and evaluate it according to the confidence it has for them. The authors describe a formal mathematical model that dictates how nodes get to know other nodes, how they manage trustworthiness through time, how they send consultation requests and, finally, how they decide on the validity of alerts based on aggregating advice from other nodes through a Bayesian approach. This method does not deal with correlation in the usual manner, but it provides a new aspect on how alerts produced by different intrusion detection systems can be combined.

The authors in [44] propose a framework that is based on an hierarchical view of the protected system. They first define ontologies for all the involved parts and then describe a model through which correlation of alerts is achieved even though the alerts may concern various kinds of assets of the protected system or they have been generated by various types of sensors. Moreover, a trustworthiness factor for each of these combinations is taken into account in order to produce results that approximate the true security status of the protected system, as accurately as possible. A limited manual experimental procedure is described in order to assess the efficiency of the proposed method, while more extensive tests are required, as the likely diversity of sensors or systems may prove the framework inapplicable.

An important requirement for successful alert correlation between different intrusion detection systems is the existence of a common representation for alerts. In [45] a well defined representation model is presented, based on the first-order logic formalism. Apart from describing a representation model for intrusion detection alerts, the authors have also tried to formalise representations for all other important entities in intrusion detection context. They propose representation schemes for hosts, software products, vulnerabilities, attack classes, intrusion detection systems, events, messages etc. The model is interesting and its use seems promising. Further testing has to be done in order to evaluate if this theoretical model is applicable to real word alert correlation scenarios.

3.4.4 Decentralizing alert correlation

A well defined methodology is proposed in [46], in order to cope with the correlation of alerts produced by intrusion detection systems scattered all over the world. The authors define an alert attribute pattern scheme, that is used to efficiently represent alerts. They use this scheme to commit a two stage correlation, one locally for each intrusion detection system and one for the global system. The two stages of correlation ensure that there will not be any computation overhead issues. A methodology for automatically setting the local and global thresholds is defined. Two different models of processing are examined, one with a central server and the other completely decentralized, built on a peer-to-peer architecture. The evaluation of the algorithm is also concrete, as it takes into account the geographical location of the sensors and it calculates the corresponding communication overheads. In general, the authors have made an important contribution to collaborative intrusion detection research.

Reducing communication overhead is also the motivation in [47]. The use of distributed hash tables in each intrusion detection node is proposed, in order to keep single and correlated alerts. This structure enables efficient and flexible handling of alerts. Computations for each correlated alert are handled to the node with the least load among the nodes relevant to the alert. Moreover, communication issues are also taken into account in the described methodology. Routing of data exchanges between nodes is based on the Kademlia algorithm that ensures that information flow is conducted through the least loaded path. The approach seems interesting, but the actual correlation process is insufficiently analyzed.

The authors in [48] have emphasized on the data fusion part of the alert correlation problem. It is commonly accepted that each intrusion detection sensor is more feasible to detect certain kinds of attacks, according to its nature. The proposed method uses a Neural Network learner unit, that is initially trained with labeled data to decide upon the weights to be used for each intrusion detection sensor. The weights depend on both the sensor itself and the kind of alert it produces. The thresholds used both in each sensor and globally are optimized through the process of observing the data flow and by dynamically modelling normal and anomalous activity distributions. The threshold values are continuously adjusted in order to keep the optimum detection-false alarm trade-off. Experimental results show that the proposed fusion system beneficially combines two different intrusion detection sensors: PHAD and ALAD; the detection rate of the fusion unit is much better than the detection rates of each of the two sensors individually, while the false positive rate is kept at minimum.

3.4.5 Taking into account expert knowledge

Due to the nature of the intrusion detection problem, no automated method can produce a perfect representation of tsecurity state of the protected system. In [49] expert knowledge is used in order to enhance both the intrusion detection and the alert correlation processes. The authors assume that intrusion detection and alert correlation both constitute classification problems. They try to revise results of broadly used classifiers (various Naive Bayes implementations and Decision trees), by taking into account prior expert knowledge. This knowledge is expressed in simple forms, e.g. a certain percentage of traffic is normal or alerts of a certain attack class follow a specific probability distribution. Their algorithm examines this knowledge and tries to alter the results of the classifiers, in order to make them adhere to the relevant limitations, to the extent that this is possible. They finally provide an analytical experimental procedure, using three different data-sets, to show the validity of their approach.

3.4.6 Concentrating on infected hosts

A different approach to the reduction of the size of the alert-set is taken in [50], where the main goal is to find infected hosts. It is difficult to efficiently transform raw alerts to meta-alerts that absolutely correspond to real security events. The authors state that it is easier to just find infected hosts on the protected network, by examining raw alerts and then to further investigate these hosts. They build a novel heuristic to detect infected hosts from a huge alert-set. This heuristic uses a statistical measure to find hosts that exhibit a repeated multi-stage malicious footprint involving specific classes of alerts. Validation of the method showed that it achieves relatively low false positive rates in huge data-sets. It is obvious that the method could be useful to a large network's administrator as she could have a good approximation of infected hosts on her network instead of a very long and impractical alert list.

3.4.7 Alert flows are more informative than single alerts

In [51] the authors propose that investigating flows of alerts is more effective than investigating single alerts. In this way it is obvious that the size of the data for the analyst;s attention is massively reduced, as flows consisting of alerts related to normal system behavior can contain strong regularities, which can be modeled and eventually filtered out. Normal flow behavior is modeled as a weighted sum of previous observations, using non-stationary auto-regressive models. The weights are re-estimated or updated at every new observation. Re-estimation is conducted through the use of a Kalman filter, and it happens on-line, without having to stop examining flows. The most significant differences between forecasts provided by the model and the observations are reported as anomalies and possible intrusions. Finally, these models are used to process voluminous alert flows from an operational network and the results are satisfactory.

3.4.8 Multiple correlators are better than a single one

The authors in [52] propose a system that is based on multiple correlation methods and, for a given data-set, is able to efficiently combine the results of these methods. A learning phase must exist in advance, in which the performance of each of the correlation methods is measured in terms of their alert reduction rate percentages. The best of the methods are then selected and applied in a best to worst fashion, during the real correlation phase. The experimental process has been conducted on various data-sets while a lot of attention has been given to the total correlation time, as the authors' intention is to produce a system capable of working on-line. An important issue is that the achieved high reduction rate is not an adequate indication of required performance, as the quality of the produced correlated alerts should also be examined.

3.5 Hypothesizing on missed events

There are only a few research efforts on hypothesizing on missed events in intrusion detection. Even though false negatives (non-existent alerts for occurred events) may be proven much more dangerous than false positives, there are not many proposals on how to deal with this issue. This mainly happens because of the complexity of the problem and the danger to destroy the initial alert-set by overpopulating it with artificial data, in order to approximate missed events.

In [53], the authors develop a series of techniques to hypothesize and reason about attacks possibly missed by intrusion detection systems. If the sensors miss some critical attacks, alerts from the same attack scenario could be split into multiple attack scenarios. Thus, combining different attack scenarios can potentially reveal alerts for missed events. They first obtain attack scenarios through a correlation method based on prerequisites and consequences of attacks and identify which attack scenarios (and possibly individual, uncorrelated alerts) may be combined by examining the attributes of the alerts in different attack scenarios. If those attribute values satisfy some constraints, they consider integrating the corresponding attack scenarios. They assume the missed attacks are most likely unknown variations of known attacks, or attacks equivalent to some known attacks. They hypothesize and reason about missed attacks based on possible causal relationships between known attacks, aiming at constructing more complete attack scenarios.

In [54], the authors first propose a novel queue graph approach to alert correlation. A queue graph only keeps in memory the latest alert matching each of the known exploits (that is, host-bound vulnerabilities). The approach has a linear time complexity and a quadratic memory requirement (both in the number of known exploits in the given network). Those are both independent of the number of received alerts, meaning the efficiency does not decrease over time. A unified method for the correlation, hypothesis, and prediction of intrusion alerts is also discussed. The method compares knowledge encoded in a queue graph with facts represented by correlated alerts. An inconsistency between the knowledge and the facts implies potential missed events. The method seems interesting but is insufficiently tested.

3.6 Visualizing results

While all methods previously analysed improve the quality of the produced alert-set, none of them can create an easy to read high level representation for the security analyst. This can only achieved by visualizing the produced alert-set. Despite of this fact, the relevant visualization methods in the literature are not many. The most recent among them are analysed in this section.

3.6.1 Tables of aggregated alerts

In [55] the motivation is to produce a graphical representation of all possible aggregations of alerts, in order to help security analysts to easily recognize anomalous activity. A graph of tables is created in an hierarchical manner; the root table of the graph represents all events. Each table on the second level represents all possible aggregations produced by defining a specific value for one of the attributes of alerts. The descendants of each second level table are more specific aggregations as the values of a second attribute is picked. It is obvious that nodes on the higher levels of the graph represent more populated aggregations, while nodes on the lower nodes represent more specific aggregations. Probability distributions of attribute values can be useful when searching for anomalies throughout the graph, as a detailed examination of the graph may provide evidence for actual intrusions.

3.6.2 Limiting the dimensionality of alerts

Usually intrusion detection alerts contain 7-8 interesting attributes. This dimensionality is obviously hard to depict by any visualization method. The motivation of [56] is to research which projection method is the most suitable in order to compress intrusion alert data and make their visualization easier. The methods compared are Principal Component Analysis (PCA), Maximum Likelihood Hebbian Learning (MLHL) and Cooperative Maximum Likelihood Hebbian Learning (CMLHL). The results of relevant experiments have shown that the latter of three methods is the most suitable, as it produces the best results. While this conclusion seems interesting, no indication is given whether CMLHL can be efficiently used in a real world scenario.

3.6.3 Different views for different uses

An interesting multi-view approach for intrusion detection visualization is presented in [57]. The authors have implemented four different representations, each being suitable for a different scenario. Specifically, there is a main system component responsible for preprocessing and aggregation of alerts along with a PostgreSQL database that holds all required data. The four different views are :

- Daily Summary: A customizable overview which shows various daily summary data, such as aggregated flows per minute over the entire network, or over certain ports.
- Intrusion Detection View: A view based on predefined or user-created templates that shows all relevant intrusion detection alerts. The user can set criteria, such as port or IP address, for the alerts being shown, in order to see the part of alert flow she is interested in.
- Home centric flow visualization: It consists of a Tree Map that shows traffic flows between attacking hosts and protected network hosts as splines. The size of the TreeMap rectangles (weight), their background color, and the spline width can be set to a default value or they can be computed by some function of the attributes of aggregated flow data, e.g., log of flow count, transferred packets, or bytes.
- Graph-based flow visualization: This is provided as an alternative to the home-centric flow visualization. The main advantage of the graph view is that it emphasizes on the structural properties of the intrusions such as the connectivity between hosts. It is easier to recognize hosts with an intense participation in the intrusion activity.

The last two of the views seem more interesting, but they are not the appropriate views to provide a satisfactory representation of the overall security state of the protected system.

3.6.4 3d may be better than 2d

In [58] an innovative approach to intrusion detection visualization is proposed. A 3d graphics engine is used to depict the protected network and the security events. Usually, the means used to visualize intrusion detection data are charts, pies or graphs. In this case, a 3d world is created, in which objects, like hosts or network connections, exist and graphical effects indicate the occurrence of an intrusion. The work presented is in its early stages; not enough evidence exists for the validity of the method.

3.6.5 Place everything on wheels

An impressive application of radial visualization in the intrusion detection field is presented in [59]. The authors have implemented AlertWheel, which is an intrusion alerts visualization method, based on the bipartite graphs approach. They depict alerts as edges that connect nodes, representative of source IPs, to a central pie, slices of which represent intrusion categories. the number of possible categories does not exceed thirty, while source IPs can be easily grouped in sub-nets. The edges, which correspond to alerts, usually come in huge numbers. The method tries to group these edges whenever they share the same path, in order to produce a readable graph. The security analyst can set criteria to restrict the alerts shown, in order to be able to read the resulting graph more easily. The method is interesting and produces a nice result. Perhaps more alert attributes can be taken into account in order to create a more informative picture.

CHAPTER 4

Observing data

The alert-sets produced by intrusion detection systems suffer from various deficiencies. In this Chapter a specific traffic data-set is used and the resulting alert-set is thoroughly examined. The redundancy of information in the alert-set is researched and various metrics are calculated to indicate the existence of characteristic properties of alerts that could be used for detecting false positives. Graphs depicting metrics values are presented, in order to justify the validity of the relevant assumptions.

4.1 Introduction

In order to design a system that will be able to enhance the quality of information of intrusion detection alerts, a thorough examination of such alerts has been performed. The Darpa 2000 1.0 traffic data-set [60] has been used to create an alert-set and the relevant documentation has been used to define the validity of each one of the alerts produced. The data-set was used as input to Snort [61] and the resulting alert-set consists of 3646 alerts, 67% of which are false.

The redundancy of information between alerts is examined and the assumptions about different characteristics of true and false alerts are checked, by computing relevant meta-data for each alert. The influence of the meta-data values to the validity of the alerts is depicted by visualizing their relationship.

4.2 Redundancy of alerts

One of the most important problems of the produced alert-set is the redundancy of alerts. Due to the nature or the time range of the events, it is common for them to trigger bursts of identical alerts. These alerts overwhelm the final alert-set and make it hard to read. Additionally each event may produce more than one distinct alert instances, as it can trigger more than one of the intrusion detection system's signatures.

In order to check the existence of this problem in the alert-set under examination, two different metrics have been defined and calculated for each alert. The first metric is called ial; for a specific alert its value is defined to be equal to the number of alerts, identical to it, that have been produced in a very narrow time range (10 seconds) around it. This metric highlights the cases, in which an event produces a burst of identical alerts. The second metric is called ral; for a specific alert its value is defined as the number of alerts similar (having at least one of signature, source IP and destination IP fields identical) to it that exist within a wider time range (120 seconds) around it. This highlights the cases, in which an event produces more than one distinct relevant alerts.

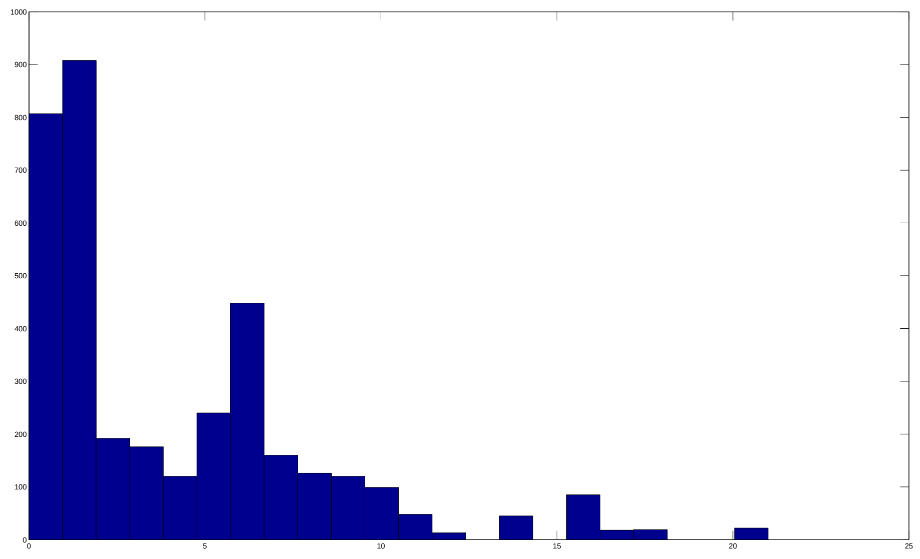

Figure 4.1: Histogram of ial values

Two histograms have been created from these metrics. Figure 4.1 shows the distribution of the ial values, while Figure 4.2 shows the distribution of the ral values for all alerts. Only 860 out of 3646 alerts seem to not have any identical counterparts, while many bursts of identical alerts appear, including up to 20 members each. The percentage of alerts with no relevant alerts at all, is obviously even lower. Only 250 out of 3646 alerts

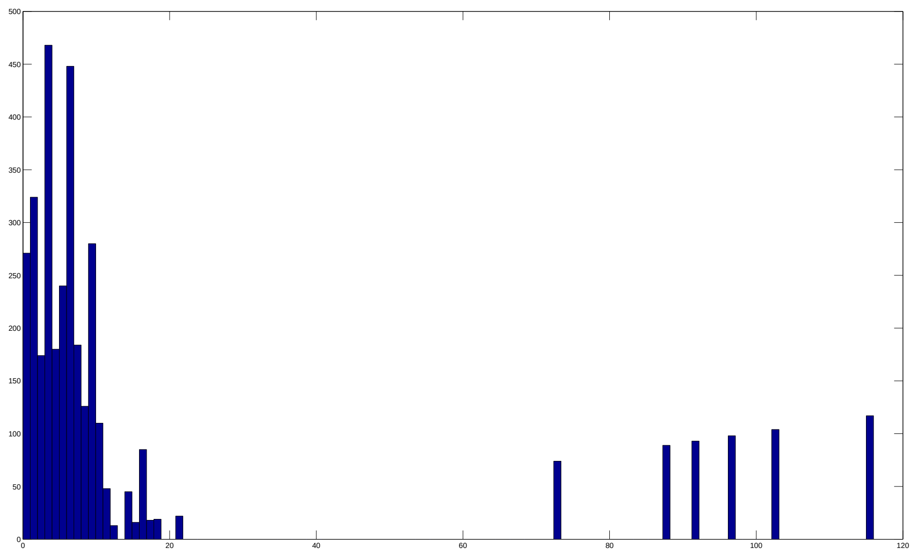

Figure 4.2: Histogram of ral values

seem to not relate to any other alert, while there are alerts measuring up to 100 relevant alerts.

These calculations show that there is a lot of redundant information in the alert-set and that a required quality improvement would be to eliminate this redundancy or to at least reduce it.

4.3 Alerts validity discriminative characteristics

A first assumption made on true alerts is that they are more probable to be part of a group of related alerts, all having been produced by the same or relative causes. On the other hand false alerts are more probable to be less relevant to other alerts, that are close in time to them. A metric of this characteristic has been defined as rel. Specifically for each alert, a range of n alerts around it is selected. Then the percentage of alerts in this range that seem to be relevant to the alert under examination is calculated. Two alerts are assumed to be relevant if at least one equality in the four combinations of their source and destination IP addresses exists. It is expected for this value to be higher for true alerts.

Another assumption made, is that true alerts are expected to produce some kind of deflection in a stable system. In other words, the state of an alert-set in an attack free scenario should be significantly changed when a real attack occurs. In order to produce an appropriate metric, frequency of each kind of alert has been used. Each alert is characterized by a signature field, which corresponds to the kind of attack detected. When no real attacks occur the frequency of appearance of alerts with a specific signature should be more or less stable and relevant to the nature and structure of the protected network. When a real attack producing alerts of the same signature occurs, then the frequency of appearances of the specific signature should temporarily increase. A metric used to examine if this assumption holds is defined as fre. This metric is calculated for each alert by dividing the current frequency of the signature of the alert (the frequency in an alerts' range around the examined alert) to the average frequency (throughout the whole alert-set) for the specific signature. It is expected that this ratio will be higher when calculated for true alerts.

Finally a third assumption made is that the majority of false alerts is the product of periodical lawful tasks in a network. In most cases there are sequences of identical false alerts that appear in more or less fixed time intervals. The metric per which is used to check the validity of this assumption relates to the periodicity of the appearances of identical alerts. Specifically for each alert, identical alerts are discovered (the criteria used are to have the same source and destination IP addresses and the same signature) through the whole alert-set. If less than 10 appearances exist then no recurring scheme is detected and the per metric is set equal to 1. If more than 10 appearances exist then the time distances between them are calculated. The per metric is set equal to the average of the percentages of change between time distances of consecutive records in the identical alerts' list. For a perfectly periodical sequence, where time distances are all equal to each other, this metric averages to 0, while its value increases as time distances start to differ from each other.

In order to efficiently observe the distribution of values for these metrics the data mining software Weka 3 [62] has been used. Its visualizing capabilities enable the production of graphs that indicate the relationships between the validity of alerts and the values of the metrics. Figures 4.3,4.4 and 4.5 show the distribution of each one of the three metrics

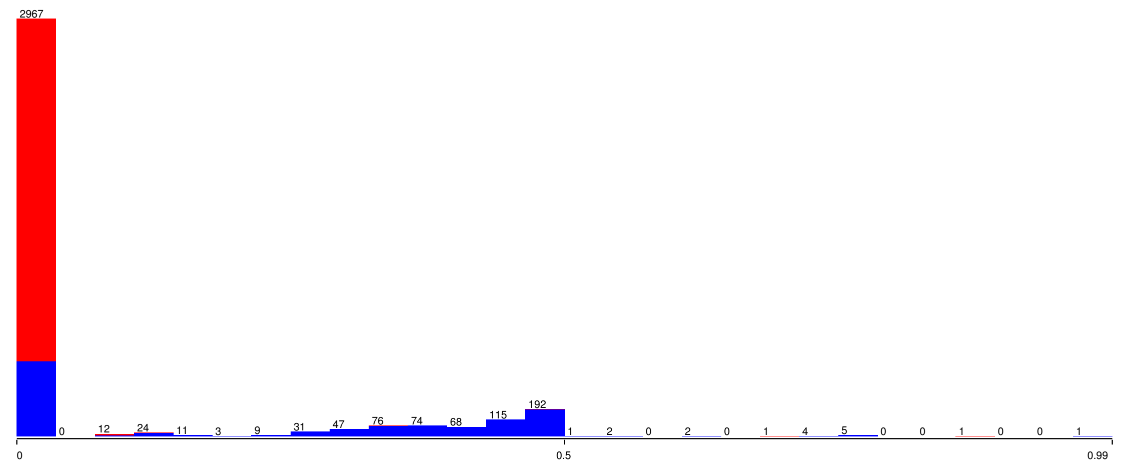

Figure 4.3: Relationship between validity of alerts and rel metric's values

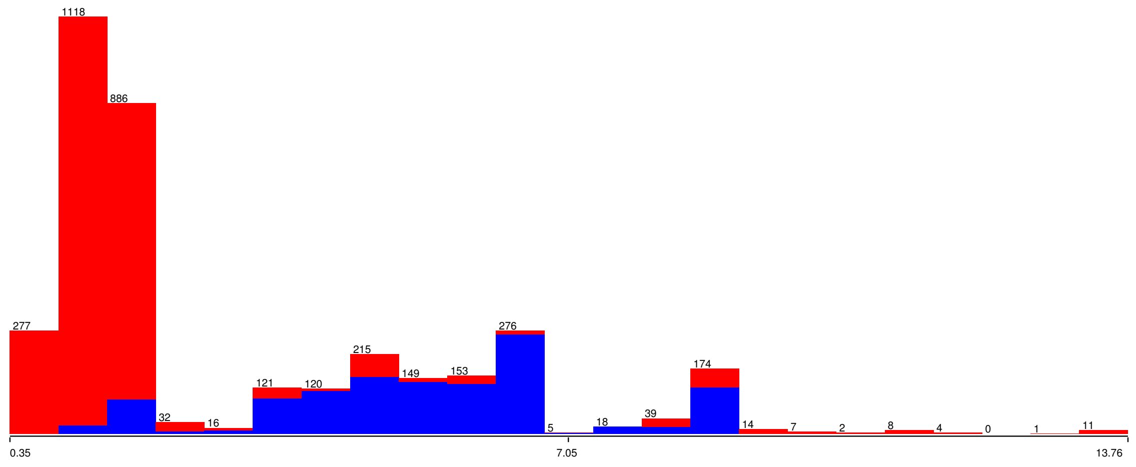

Figure 4.4: Relationship between validity of alerts and fre metric's values

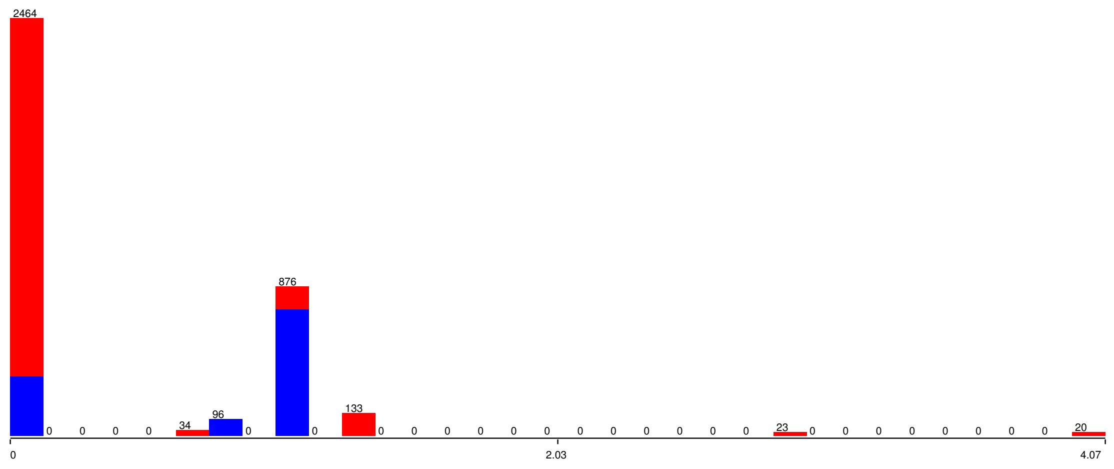

Figure 4.5: Relationship between validity of alerts and per metric's values

values for true and false alerts. The blue bars correspond to true alerts, while the red ones correspond to false alerts.

As it is obvious in Figure 4.3 rel tends to have values close to zero for all false alerts. In other words false alerts tend to not have a significant percentage of other related alerts around them. Figure 4.4 shows that the values of fre are mainly grouped in two different groups, which correspond to true and false alerts. Specifically false alerts have values in the range [0,2], while true alerts mainly have values in the range [3,7]. Finally Figure 4.5 shows that for most of the false alerts, the values of per are close to zero. By examining the three Figures strong relationships of the values of the metrics to the validity of alerts are discovered. The initially made assumptions, initially made, can be used to classify alerts as true or false.

Furthermore the combination of the metrics can reveal additional correlations between the calculated data and the validity of the alerts. Figures 4.6,4.7 and 4.8 show the distribution of true and false alerts on two dimensional graphs the axes of which represent all possible pairs of the thee metrics. In all three graphs the false alerts seem to form clear clusters. The combination of previous assumptions can produce a classification scheme with a high detection rate for false alerts.

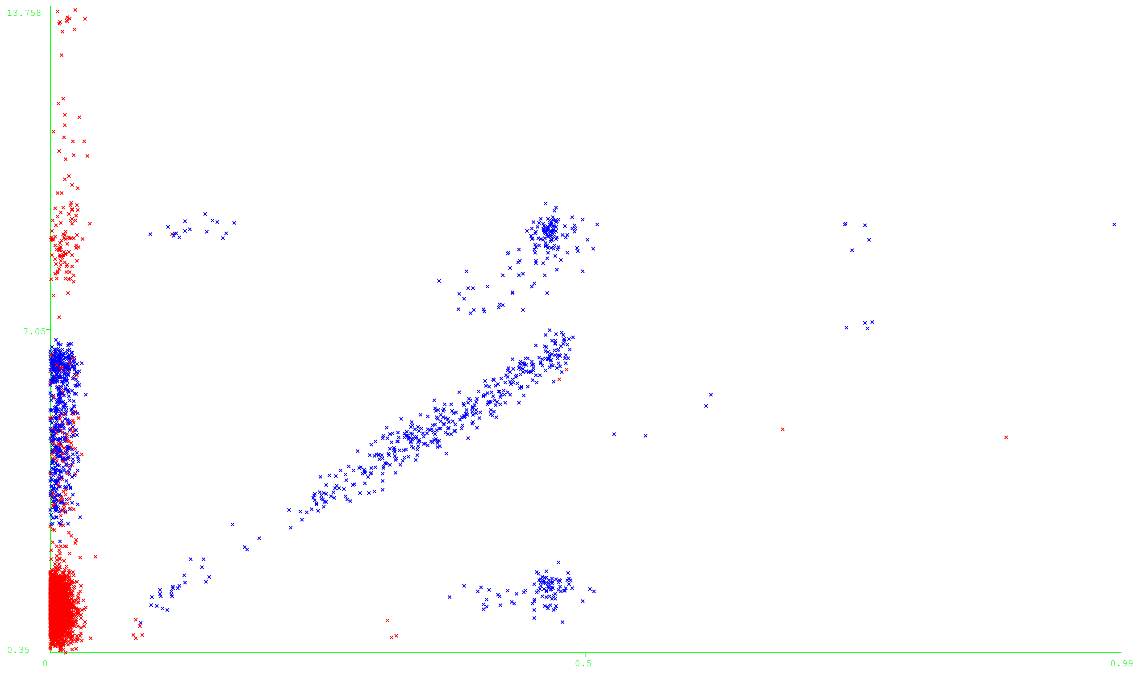

Figure 4.6: Combination of rel and fre

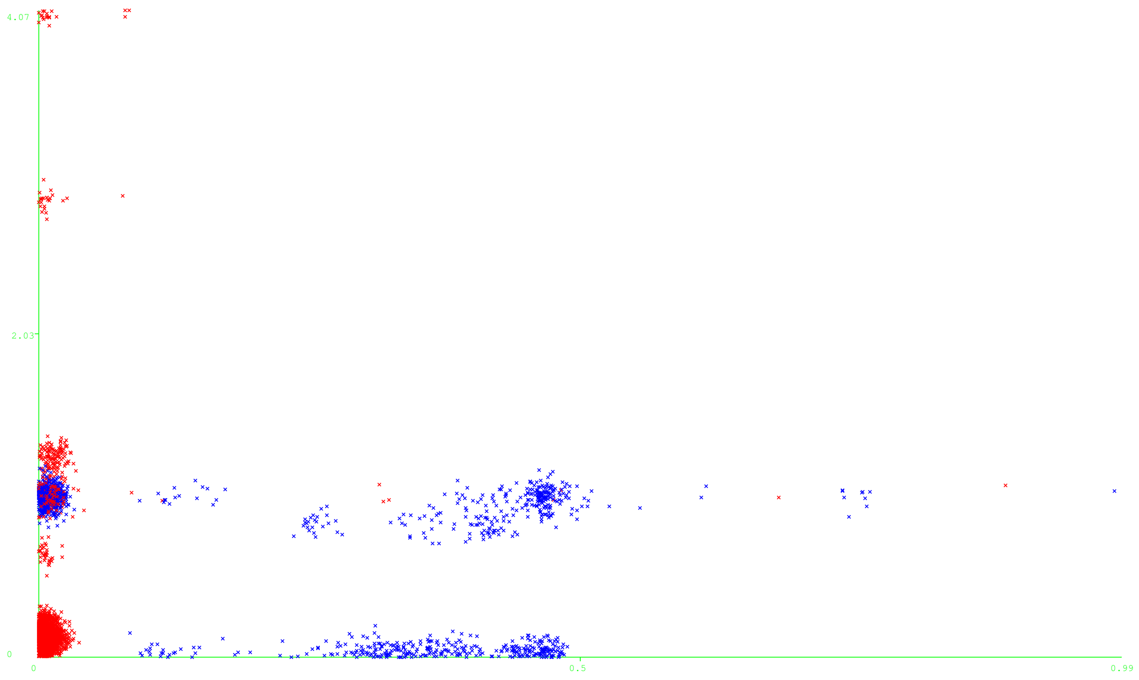

Figure 4.7: Combination of rel and per

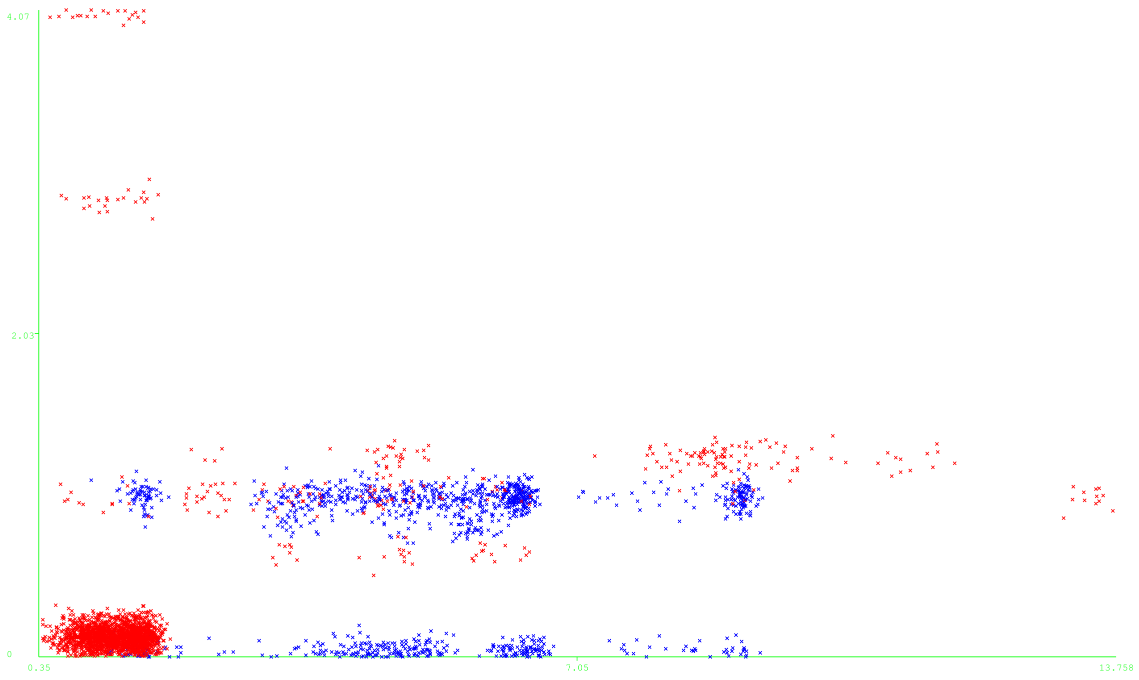

Figure 4.8: Combination of fre and per

CHAPTER 5

Motivation and design

The system proposed in this thesis is a complete post-processing solution that intends to enhance the output of intrusion detection systems. The motivation is to provide security analysts with a more elaborate way of inspecting the security state of the protected system. In this Chapter the main ideas motivating the implementation of the system are presented. Additionally the initial design of the system is analysed and the tasks of the main subsystems are discussed. Sensor managers read alerts from their sources, produce a validity score and commit the required aggregation. The clustering subsystem merges the different inputs from the sensor managers, it creates clusters of related alerts and it produces artificial clusters to approximate alerts for events that have not been detected by sensors. The database of the system holds data for profiling and for communication between different subsystems. Finally the visualization subsystem produces an informative live representation of the security state of the system in the form of a three-dimensional graph.

5.1 Motivation for the system

As discussed in Chapter 2 intrusion detection systems produce alert-sets suffering from significant problems. The motivation for the system of this thesis is to analyse these problems and to create a mechanism that will automatically transform the outcome of intrusion detection systems in something more meaningful and helpful for the security analyst. The main ideas, on which the design of the system, is based are:

- Reducing the volume of alerts
- Reducing the rate of false positives

- Producing alerts relevant to the actions of the intruder
- Hypothesizing about missed intrusion events
- Visualizing the final outcome
- On-line efficient operation

In other words, the design of the system was guided by the motivation to cope with the problems analysed in Chapter 2. These design ideas are analysed in the coming subsections.

5.1.1 Reducing the volume of alerts

The first important problem that a user (security analyst) of an intrusion detection system encounters is the sheer volume of the produced alerts. This volume most of the time makes even observing the alert-set very difficult. The user has to dig through thousands of alerts, in order to find the ones that seem interesting and have potentially been triggered by real intrusion events. Methods relevant to reducing the volume of alerts are aggregation and correlation.

Aggregation of alerts is the procedure that tries to substitute multiple similar or identical alerts that have been triggered by a single event with one representative general alert. A single low level event (a TCP/IP packet or a system call) can produce many identical alerts either because of its time duration or because of triggering multiple intrusion detection sensors, mechanisms or rules. Successful aggregation of alerts is usually feasible, as the similarity of alerts that should be aggregated is high.

On the other hand correlation is the procedure that tries to discover relationships, of any kind, between alerts, triggered by different low level events, and create groups of relevant alerts. The identified relationships can be logical, when the alerts relate to actions that take place consecutively in an attack plan. They can also be chronological, when alerts occur too close in time. Moreover alerts could be related to each other when they pertain to the same asset of the protected system. This common asset can be a host, a subnet of the protected network or a service of a protected host. The groups of relevant alerts can be represented by a more general alert indicating their details.

In the proposed system both of these approaches have been incorporated, in order to reduce the volume of alerts initially produced by the intrusion detection system. Aggregation is performed at the sensor level, while correlation is performed at the system level. For every sensor, each alert is compared in an efficient way to preceding alerts of the same sensor, and if they are found to match they are aggregated. Afterwards when the alert-sets of the multiple sensors are merged, the correlation is performed through similarity-based clustering.

5.1.2 Reducing the rate of false positives

The most known problem of intrusion detection systems is high rate of false positives. The way of operation of any intrusion detection system invariably produces many false positives. The requirement for high detection rate calls for a strict configuration of intrusion detection systems, which causes high false positive rate, as many non-intrusive actions are interpreted as suspicious and relevant alerts are produced.

Many methods have been proposed for discarding false positives. The methodology used in the proposed system is mainly based on statistical observations of various alert-sets and discriminative properties that seem to characterize false alerts or their relationships with their neighbouring alerts.

In order to avoid the danger of discarding useful alerts along with false ones, the idea of filtering out alerts considered to be false was not pursued. Instead the calculation of a probability (score) that indicates whether an alert is true was chosen as more appropriate. The use of a threshold value set by the security analyst in the next stage of processing can easily discard alerts, if this is required.

More than one properties of false alerts are exploited in order to conclude if an alert is true and the combination of the multiple corresponding probabilities is a fundamental issue. A weighted formula has been used, aiming to benefit the parts of the system that seem to produce more reliable estimations.

5.1.3 Producing alerts relevant to actions of the intruder

Apart from containing too many alerts, most of which are false, alert-sets usually contain low level information, which does not correspond to the actions of the intruder.

Each action of the intruder may trigger multiple alerts, similar or not, which make its identification harder for the security analyst.

The proposed system tries to produce information at the same level of granularity with the actions of the intruder. This has been taken into account when designing the clustering procedure, as the production rate of clusters is decisive for the quality of the information finally produced.

Similarity-based clustering was elected as the most suitable way of comparing alerts and deciding which ones of those may constitute a group of indications of the same event or of related events. Each one of alerts' attributes is separately processed and the individual similarities are then combined to decide on whether the alerts are similar or not.

5.1.4 Hypothesizing on missed intrusion events

Besides false positives there are also false negatives, which may be less in number but they may be much more dangerous. False negative stands for a committed intrusion without an associated alert. Intrusion detection systems produce huge numbers of alerts, aiming at as low false negatives rates as possible. This happens because a false negative usually has marginally larger cost for the protected system, than a false positive.

In the proposed system there is a component that aims to hypothesize on information about security events that have been missed by the intrusion detection system. It is obvious that the correct estimation of this information is practically impossible. On the other hand a good approximation of events missed by the intrusion detection system may stimulate the user, in order to protect the system in a better way.

This hypothesizing attempt, may also produce alerts for events that never happened and gradually destroy the produced alert-set. Because of this danger a lot of attention should be given to the configuration of this part of the system, to ensure that the alert-set will not be vastly altered by artificially generated data.

5.1.5 Visualizing the final outcome

Finally all the above ideas will not offer much to the security analyst, if an alternative way to visualize the produced information is not incorporated. Her immediate reaction may be critical for defending the protected system from an attack, so digging through lists of alerts, to check if an intrusion is happening or not, may be dangerous.

An elaborate visualization scheme can inform the analyst intermediately if suspicious behaviour is detected, giving her basic information about it. If she believes that more information could be useful, she can then examine the produced alert-set to extract it.

A three-dimensional space has been used to allow alerts visualization to be as informative as possible. Activity is visualized with respect to IP values, time and estimated danger. The rationale behind this part of the system is to produce a comprehensive image of the security state of the system that will be continuously updated.

5.1.6 On-line efficient operation

An important requirement for designing the system was to be as lightweight as possible, in order to be able to on-line enhance alert-sets of huge sizes. Introducing too complex methods to solve the above problems would prohibit any real world application for the system, as off-line intrusion detection is applicable only in experimental set-ups. Most of the methods presented in Chapter 3 have been tested off-line. Authors have used ready data-sets to conduct experiments that justify the validity of their methods, without checking the application of these in a real world scenario.

A lot of attention was given to the processing rate of the proposed system; how many alerts per second it can process, without introducing any latency in informing the security analyst. A simulation environment has been developed in order for the experiments to be run in a real world scenario. In this way the overhead introduced by each component and the overall performance of the system can be easily measured.

Another important issue is that when examining alerts off-line all the alert-set is available. On the other hand when examining alerts on-line only the previous alerts are available. The simulation environment mentioned above ensures that the components of our system in order to process an alert use the information they have accepted as input until the time-stamp of this alert. Using all the dataset (previous and next alerts) to process a single alert is obviously not rational and cannot be applicable in a real world implementation.

5.2 Designing the system

In order to implement all the ideas mentioned in Section 5.1 the design of the system has been focused on three different subsystems and a database. These are:

- A sensor manager for each intrusion detection sensor used
- The clustering subsystem
- A database for storing clusters' data
- The visualization subsystem

Basic design and functioning of the system is depicted in Figure 5.1. In the following subsections the task of each subsystem is analysed.

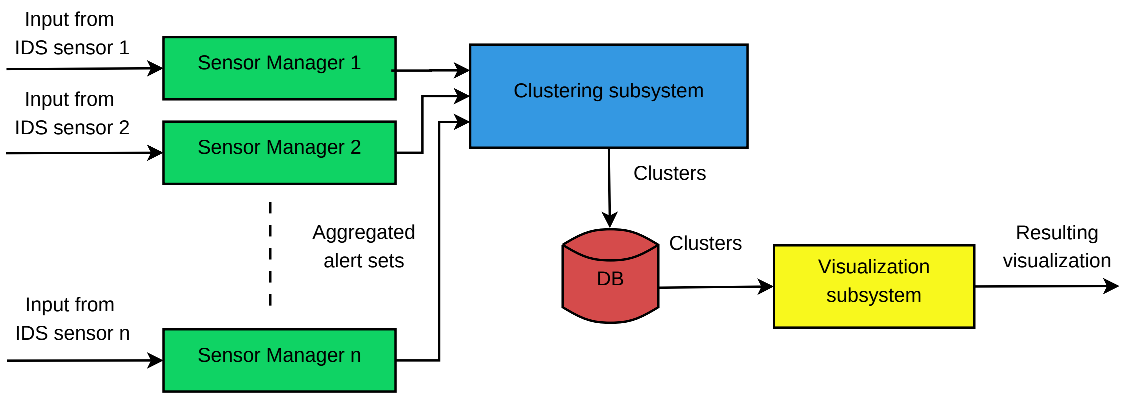

Figure 5.1: Overall design of the system

5.2.1 Sensor manager

Each one of the sensor managers is responsible for manipulating the alert-set of the corresponding sensor. Sensor managers are identical and produce equal in number aggregated alert-sets that are the input for the Clustering component that follows.

The main tasks that Sensor managers perform are:

- Read alert-set from sensor
- Forward each alert to various components, which produce individual scores according to the validity of the alert

- Keep track of the scores produced by each component
- Accordingly configure weights used for the combination of individual scores
- Calculate a combined score according to the validity of each alert and attach it to the alert
- Aggregate alerts, to eliminate groups of multiple identical ones and produce an aggregated alert-set
- Forward this aggregated alert-set to the Clustering subsystem

As indicated in Chapter 4 false positive alerts are characterized by specific statistical properties that could be exploited, in order to successfully classify alerts as true or false. Four different components have been designed and embedded into the Sensor manager subsystem that take advantage of the observations in Chapter 4. These produce four corresponding scores for each alert, that stand for the probability that the specific alert is true or not.

These components may be more or less effective, depending on various parameters such as the nature of the protected network, the kind of attack or even time. The diversity of these parameters may prove components ineffective or make them produce misleading results from time to time. Because of this, a weighting formula has been employed, which enables the Sensor manager subsystem to promote the scores of components that seem to perform better under the current circumstances, while it ignores to a certain extent the scores of the remaining components. The functioning of the weighting formula is presented in Chapter 6, along with the algorithm used to on-line update these weights.

The weights are used to calculate a combined score for each alert, which is then attached to the alert. Discarding alerts has not been preferred, even for alerts with a very low score. All alerts are propagated to the next components, but their scores are used to estimate the validity of aggregated alerts, clusters of alerts produced by them or resulting visualizations. A threshold value can be used, in subsequent stages of processing, to filter out data with low estimated validity.

The last process that the Sensor manager subsystem performs is the aggregation of alerts. As stated in Chapter 2, intrusion detection systems tend to produce multiple similar alerts for each security event. The aggregation component accepts as input the alert-set of the sensor along with the calculated scores. It then aggregates similar alerts that seem to have been triggered by the same event into one aggregated alert, that holds all the relative data. This procedure is important as it significantly reduces the size of the alert-set, and besides improving the quality of information in the alert-set, it enables subsequent subsystems to perform much faster for a given input to the system.

The output of the Sensor Manager subsystem is an aggregated alert-set that is fed to the first component of the next subsystem, the Clustering subsystem.

5.2.2 Clustering subsystem

The Clustering subsystem is important for the overall approach as it adds much value to the initial alert-set. Its main tasks are:

- Accept aggregated alert-sets as input from the Sensor managers
- Merge these aggregated alert-sets, by discarding identical alerts
- Periodically examine clusters and generate artificial ones whenever sensors seem to have missed intrusion events
- Produce final cluster-set and store it to the database

The step of merging is required, as the inputs from multiple sensors must be converted to a uniform alert-set, in order to produce an overall representation of the security state of the protected system. As different sensors may monitor the same assets or the same intrusion may be concurrently happening at different parts of a system, it is usual for different sensors to produce alerts for the same event. The merging component deals with multiple aggregated alerts' streams and unifies them, while it checks for aggregated alerts eligible for discarding, as they may contain redundant information.

The main mission of the Clustering subsystem is to produce clusters of alerts relevant to each other, or having been triggered by the same attack plan. It is obvious that successful clustering is decisive for the quality of the final result. The motivation behind the clustering procedure is to create for each intrusion plan as many clusters as the steps of the plan are. Choosing the most appropriate fields of alerts, to base the clustering algorithm on, is an important factor for producing meaningful clusters. Similarity based clustering is used, to discover correlation between aggregated alerts that are included in the output stream of the merging component.

Additionally this subsystem is designated to reconstruct information for security events that have been missed by intrusion detection sensors. For these events no relevant alerts exist in the incoming aggregated alert-set. The motivation for generating cluster is not to precisely generate alerts, that have not been produced by the sensors, but to generate clusters which in reality are estimates of the clusters that would have existed if all events were successfully detected.

It is expected for this component to produce a percentage of wrong estimates, artificial clusters which will not correspond to any real event. Its designing, has been very careful; while a small percentage of wrong artificial alerts may be acceptable, a higher one may destroy the initial alert-sets altogether.

5.2.3 Database of the system

The database of the system is a MySQL database which is mainly used for two purposes:

- To hold data relevant to all the components of the system, for performance evaluation
- To hold specific data for produced clusters, that enable the Visualization subsystem to operate

The parameters of each of the components, along with the data they manipulate as input or output are important for inspecting the functioning of the system. All components store data to the database as they operate, for profiling reasons.

Apart from that the Visualization subsystem is implemented separately; the best way to make clusters' data required for visualization available to it is to store them in the database. This data is constantly updated by the clustering subsystem. The Visualization subsystem periodically reads this data, to update the resulting visualization. The appropriate update intervals are decided by the speed of the simulation and the density of alerts.

5.2.4 Visualization subsystem

The visualization subsystem is responsible for producing a meaningful representation of the security state of the system. It reads clusters data produced by the clustering subsystem and stored in the database of the system. Then it produces a graph in 3 dimensions that depicts clusters in relevance to time, internal IPs of the protected network, validity estimation and probable danger for the protected system.

It has been designed on the basis of giving to the security analyst an initial general view of intrusions happening in the system and impel her to further investigate initial alert-sets produced by the intrusion detection sensors, when this is required. By first inspecting the result of the Visualization subsystem and then studying alerts accordingly to the information depicted on the graph, the analyst can be more effective in monitoring the security state of the protected system.

This subsystem continuously reads the clusters' information stored in the database at specific time intervals. These intervals should be chosen so as to ensure that the graph produced evolves through time without interruptions and depicts all events happening. The optimal updating speed depends upon both the density of alerts in time and the speed of simulation.

The axis of the three-dimensional graph, depict time, IPs of protected network and a danger estimate of the event. A plane is created in this three dimensional space, to represent the security state of the system. If no clusters exist this plane coincides with the x-y plane (time-IPs plane). At every point where a cluster exists the surface is characterized by a peak. The height of the peak, in the z axis, corresponds to a danger value calculated for the cluster. Finally the color of the plane at each point is decided by an estimated validity value for each cluster.

The plane along with the graph evolve through time to adjust to the clusters' data. For example the time axis should obviously be elongated at each update, and the plane should be altered to depict clusters created since the last update. The user watching the output of the Visualization subsystem, sees a live image showing the exact security state of the protected network, as this can be estimated by the sensors' alerts, at each moment.

5.3 Data formats

Defining of formats used to represent the data exchanged between the components of the system, is an important keystone for the effective and accurate functioning of the system as a whole. Each of the components accepts a specific type of data as input and produces a specific type of data as output. In this Subsection the types of data used are analysed.

5.3.1 Alert

The first type of data used is obviously relevant to alerts. The type of data **Alert** holds all information relative to alerts as they are read from the sensors. **Alert** fields are:

- **Sensor id:** This is an id relevant to the sensor the alert comes from. Each sensor has an id and this id is attached to the **Sensor id** field of its alerts. It is an integer value.
- **Signature:** This is relevant to the kind of the intrusion the alert was triggered by. In signature-based systems matching of traffic with signatures that concern specific attacks are used to produce alerts. It is an integer value.
- **Class id:** Signatures are categorized in different classes, that mainly stand for the phase of an attack plan the intrusion belongs to. The **Class id** of each alert shows at which phase of a designated attack plan the alert has been produced. The higher the **Class id**, the more advanced the relative event is. It is an integer value.
- **Time-stamp:** Time-stamp of alert is the point in time at which the alert was generated. It is calculated in Unix time format, seconds from 00:00:00 UTC on 1 January 1970. It is obviously an integer value.
- **Source IP:** It is the source IP of the IP packet that has triggered the alert. It is an integer value. The conversion of an IP X.Y.Z.W to integer is achieved by Equation 5.1.

$$IP_{int} = X * 256^3 + Y * 256^2 + Z * 256 + W \tag{5.1}$$

- **Destination IP:** It is the destination IP of the IP packet that has triggered the alert. It is also an integer which is calculated by Equation 5.1.

- **Truth:** It holds the validity estimate produced in Sensor Manager for the alert. It is a real number ranging from zero to one.

5.3.2 Aggregated alert

The next type of data used is **Aggregated alert**. This holds the aggregated alerts produced by the aggregation procedure, in the Sensor manager. During this phase the system tries to aggregate alerts that are identical and refer to the same event. **Aggregated alert** is similar to **Alert**, but it has some differences, to hold information for the group of alerts that have been aggregated.

The **Alerts** aggregated to an **Aggregated alert** are identical, so most of their fields hold the same values. So **Sensor id**, **Signature**, **Class id**, **Source IP** and **Destination IP** fields also exist in **Aggregated alerts**. They inherit their values from the corresponding values of the **Alerts** that have been aggregated to produce the specific **Aggregated alert**. Due to the aggregation algorithm used these fields have the same values for all **Alerts** appropriate for being aggregated together. The **Time** field does not exist in the **Aggregated alert** type, as it is replaced by two fields:

- **Start time:** It is equal to the time value of the first one of the **Alerts** aggregated. It is held in Unix time format, and it is obviously an integer value.
- **End time:** It is equal to the time value of the last one of the **Alerts** aggregated. It is held in Unix time format, and it is obviously an integer value.

Additionally there is one more field, with respect to Alert data type:

- **Number of alerts:** It is the number of **Alerts** aggregated to produce the **Aggregated alert**. It is an integer value.

5.3.3 Cluster

Another important data format is **Cluster**. It is used by the Clustering component to store information of clusters produced during the clustering algorithm. The clustering algorithm tries to discover groups of **Aggregated alerts** that relate to each other and transform them to corresponding clusters. The fields of Cluster data type are:

- **List of signatures:** A list that contains **Signatures** existing in at least one of the **Aggregated alerts**, from which the **Cluster** has been produced.
- **List of class ids:** A list that contains **Class ids** existing in at least one of the **Aggregated alerts**, from which the **Cluster** has been produced.
- **Start time:** The earliest of the **Start times** of **Aggregated alerts**, from which the **Cluster** has been produced.
- **End time:** The latest of the **End times** of **Aggregated alerts**, from which the **Cluster** has been produced.
- **List of source IPs:** A list that contains all IPs that exist as **Source IPs** in at least one of the **Aggregated alerts**, from which the **Cluster** has been produced.
- **List of destination IPs:** A list that contains all IPs that exist as **Destination IPs** in at least one of the **Aggregated alerts**, from which the **Cluster** has been produced.
- **Truth:** A validity estimate of the **Cluster** as a whole, that is produced by **Truth values** of **Aggregated alerts**, from which the **Cluster** has been produced.
- **Number of alerts:** The sum of **Number of alerts** fields of all **Aggregated alerts** that produced the **Cluster**. It is an integer value.

Besides these basic fields, there are some auxiliary fields that are used by the Clustering subsystem and the Visualization subsystem, in order to reduce the computation complexity and enable the system to be more effective:

- **Internal IPs:** A list that contains all IPs that belong to the protected network and that appear at least once in either **Source IP** or **Destination IP** of **Aggregated alerts**, from which the **Cluster** has been produced.
- **External IPs:** A list that contains all IPs that do not belong to the protected network protected and appear at least once in either **Source IP** or **Destination IP** of **Aggregated alerts**, from which the **Cluster** has been produced.
- **Minimum internal IP:** The minimum IP in **Internal IPs** list.

- **Maximum internal IP:** The maximum IP in **Internal IPs** list.
- **Minimum class id:** The first in order of **List of class ids**
- **Maximum class id:** The last in order of **List of class ids**
- **Danger value :** It is a value in the [0,1] range that indicates if the **Cluster** refers to initial steps of an attack plan or to advanced ones. It is calculated, when needed, from the **List of signatures** of the **Cluster**. A danger score has been manually assigned to each one of alerts signatures, according to the relevant attack. The **Danger value** of the **Cluster** is the average of the danger values of signatures existing in its **List of signatures**.

5.3.4 Generated Cluster

Generated cluster format is a specialization of the **Cluster** format and it is used to represent the clusters artificially created in the the Clustering subsystem. The fields of **Generated Cluster** is a subset of the fields of **Cluster**, as not all cluster parameters can be estimated during clusters' generation procedure.

The fields included in **Generated cluster** format are **List of class ids**, **Start time**, **End time**, **Minimum internal IP**, **Maximum internal IP** and **Truth** and their usage is similar to the usage of the corresponding fields of **Cluster** format.

5.3.5 Data formats along with their acronyms

In Table 5.1 all four main data formats of the system are shown. Along with their fields the corresponding acronyms are given, which will be used throughout the rest of the thesis.

Table 5.1: Main data formats of the system

Alert (ale)	Aggregated alert (aga)	Cluster (clu)	Generated cluster (gcl)
Sensor id (sid)	Sensor id (sid)	Signatures list (sig)	Class ids list (cid)
Signature (sig)	Signature (sig)	Class ids list (cid)	Start time (sti)
Class id (cid)	Class id (cid)	Start time (sti)	End time (eti)
Time-stamp (time)	Source IP (sip)	End time (eti)	Min internal IP (minii)
Source IP (sip)	Dest. IP (dip)	Source IPs list (sip)	Max internal IP (maxii)
Dest. IP (dip)	Truth (truth)	Dest. IPs list (dip)	Truth (truth)
Truth (truth)	Start time (sti)	Truth (truth)	
	End time (eti)	Alerts number (num)	
	Alerts number (num)	Internal IPs (iip)	
		External IPs (eip)	
		Min internal IP (minii)	
		Max internal IP (maxii)	
		Min class id (mincid)	
		Max class id (maxcid)	

CHAPTER 6

Sensor manager

The Sensor manager accepts as input alerts from a sensor, decides a validity score for each one and then aggregates identical alerts. The validity score is calculated by combining four partial validity scores produced by corresponding components. These components examine various metrics, indicative of the validity of the alerts. Four corresponding weights are used for the scores produced by the components. These weights are continuously updated in order to make the combination of scores as efficient as possible. The final score calculated for each alert is attached to it, before the alerts are sent to the aggregation component, that reduces the volume of alerts by transforming groups of identical alerts to representative aggregated alerts.

6.1 Design of Sensor manager

The functioning of the Sensor manager subsystem is presented in this Section. As it has been mentioned in Section 5.2 the task of the Sensor manager is to read alerts from its corresponding sensor, to decide a validity score for each one of the alerts and then to conduct the required aggregation, in order to eliminate multiple identical alerts. The main components of the Sensor manager subsystem along with its design are depicted in Figure 6.1.

Sensor manager contributes to the functioning of the whole system, as it is responsible for two important quality improvements on the alerts-set. The first one is to produce a validity score for each alert, which is an estimate of the probability that the alert is true, given the parameters of the actual alert-set. The second task is to perform aggregation of alerts, in order to cope with the known deficiency of intrusion detection systems to

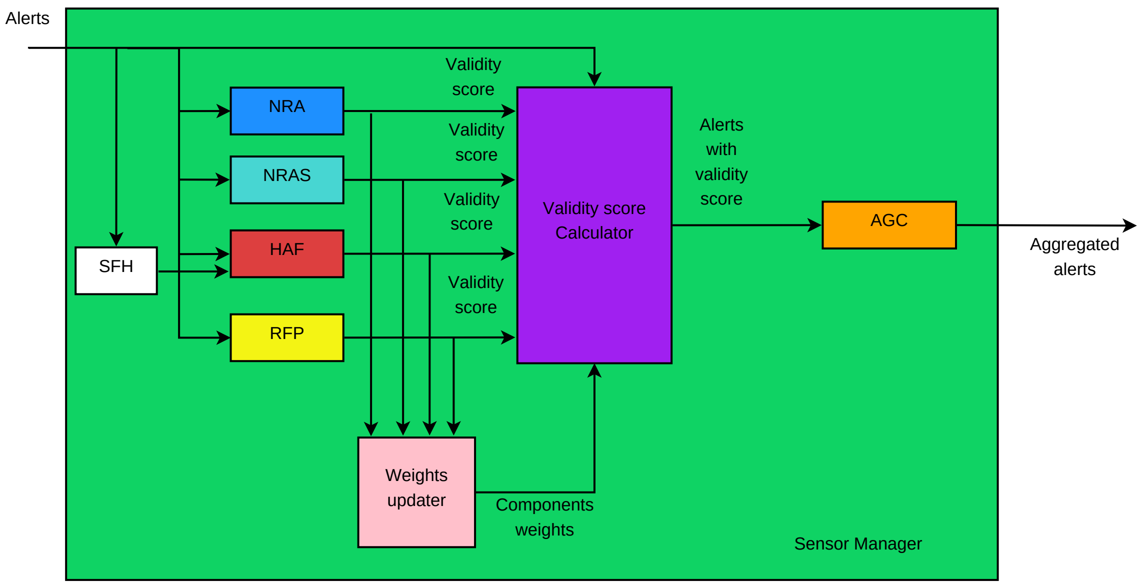

Figure 6.1: Sensor manager subsystem

produce multiple identical alerts for single events.

Estimation of validity score is carried out by the combination of six components namely NRA, NRAS, HAF, RFP, Weights updater and Validity score calculator. The AGC component is responsible for performing the required aggregation procedure. These components are analysed in the following sections.

6.2 Calculating a validity score for each alert

In order for the system to finally produce the best possible representation for the security state of the protected system, the false positives issue must surely be taken into account. As it has been analysed in Chapter 2, the most important problem of intrusion detection is that the majority of alerts they produce are false; they do not correspond to real intrusions. In order to offer to the security analyst information of high quality, these false alerts should be detected.

Most of the functioning of the Sensor manager subsystem is relevant to producing a validity score for each alert. It has been preferred not to discard alerts characterized by low validity scores at this phase. The aggregation, merging and clustering procedures that follow, group multiple alerts together to create more complex data forms. An alert with low validity score may be useful at later stages of the procedure. The calculated score is attached to each alert and is eventually used to calculate validity scores for more complex data forms, to which alerts are transformed by the following components.

The score is calculated by combining scores produced by four different components. Each one of these components has been designed to take advantage of the specific property of false alerts. They calculate partial validity scores by examining each alert with respect to this property. It is obvious that each one of these components is not able to produce correct scores for all alerts contained in the alert-set, on its own. Each one of them may be efficient for specific cases of false positives, which are characterized by the examined property. It surely can not produce useful estimates for the whole alert-set. The combination of the four components builds upon the partial scores and can produce good validity estimates for most of the false positives, as it takes into account all four parameters examined by the components.

The four components producing partial validity scores are:

- The NRA (Neighbouring Related Alerts) component that searches for neighbouring related alerts and uses them as indication of the validity of an alert.
- The NRAS (Neighbouring Sweeping Related Alerts) that is based on the same logic but is modified to discover special relations of alerts that hold in sweeping attacks.
- The HAF (High Alert Frequency) component that detects peaks in signature related frequency of alerts and uses it as an indication of the validity of the alert. It utilises another component of the Sensor manager, namely the SFH (Signature Frequencies Holder) component.
- The RFP (Recurring False Positives) component that detects recurring patterns of alerts and penalizes the alerts conforming to them as probable false positives.

The combination of the four scores is ruled by four corresponding weights that may set each of the components to be more or less decisive for the final estimate. The weights are continuously updated during the actual functioning of the system according to the scores produced by the four components. In this way the Weights updater protects the system from being biased by components that may malfunction in certain circumstances and produce too high or too low scores for all alerts.

When the four scores are sent to the weights updater it adds them to its data and recalculates the four weights. These are used by the Validity score calculator, to produce the final validity score for the examined alert. The procedure is repeated for every alert.

In the next subsections the logic and the algorithm of each one of these components are presented.

6.2.1 Configuration of components

The components deciding the validity of each alert are parametrized, in order to be eligible to produce correct scores for diverse alert-sets. The Sensor Manager subsystem reads the incoming alert-set and calculates two variables, which depend on its nature. The two variables are Alerts density den_{alerts} and Alerts variance var_{alerts} and are consequently used in the calculation of parameters used for the configuration of the four validity estimation components.

The den_{alerts} variable shows how dense the alerts are. It is high if alerts are close in time and lower if they are sparse. Its calculation is easy for the Sensor Manager as it is produced by the number of alerts and the time range of the alert-set. For an alert-set with n alerts den_{alerts} is calculated as in Equation 6.1.

$$den_{alerts} = \frac{n}{time_{n-1} - time_0} \tag{6.1}$$

The var_{alerts} variable shows the diversity of alerts in terms of kind of attack (signature). An alert-set with variety in alerts' signatures is characterized by high values of var_{alerts} , while another one with signatures recurring through adjacent alerts has lower values of var_{alerts} . It is obvious that, while the possible signatures are in the magnitude of hundreds, an alert-set with thousands of alerts cannot have a different signature for each one of its alerts. What is important for calculating var_{alerts} is changes of signature in adjacent alerts. Specifically the alert-set is split to chunks of ten alerts each. The number of chunks is easily calculated by Equation 6.2. The distinct signatures in each chunk are counted and their population is divided by ten to produce the local variance, as shown in Equation 6.3. The global variance for the whole alert-set is the average of the local variances and is calculated as shown in Equation 6.4.

$$num_{chunks} = (alertset_{size} \div 10) + 1 \tag{6.2}$$

$$var_n{}^{local} = \frac{\text{number of distinct signatures}}{10} \tag{6.3}$$

$$var_{alerts} = \frac{\sum_{i=1}^{num_{chunks}} var_i{}^{local}}{num_{decs}} \tag{6.4}$$

6.2.2 NRA component

For each alert the NRA component examines the existence of other alerts that are close in time and seem to be related to it. The assumption on which NRA is based is that true alerts seem to have more neighbouring related alerts, while false ones seem to have less.

When a real attack occurs, and the intrusion detection system detects it, a group of alerts, related to the attack, is produced. According to the nature of the attack the number of produced alerts varies. For specific attacks such as port scanning this number can be extremely large. These alerts are related to each other and this relation can be indicated by similarities in their source and destination IP addresses.

On the other hand, false positives are false alerts which are evenly distributed throughout the huge amount of alerts produced by the intrusion detection system. Therefore, a decision on whether an alert is true or false can be based on the number of alerts that occur in a time window around the specific alert and have common values in the source and destination IP address fields with the specific alert.

The input of the NRA component is of type **Alert**, the fields of which are shown in Table 5.1, while its output is the corresponding validity score.

The operation of NRA is configured by two parameters, namely the density limit dl_{NRA} and the time window tw_{NRA}. These are calculated from the variables of the alert-set mentioned in subsection 6.2.1, as shown in Equations 6.5,6.6. These relations have been decided empirically, by observing the performance of the component on various alert-sets.

$$dl_{NRA} = \frac{0.4}{var_{alerts}} \tag{6.5}$$

$$tw_{NRA} = \frac{100}{den_{alerts}} \tag{6.6}$$

The component scans the alert-set backwards until it finds the first alert of which the time difference with the alert being examined is greater than tw_{NRA} value. Practically it scans all alerts in a time window that ends at the alert being examined and has width equal to tw_{NRA}. While scanning these alerts the component counts how many of them are related to the alert being examined. This relation's conditions are:

- There is at least one equality of source and destination IP addresses between them.
- They do not share the same signature.

One of the source or destination IP addresses of the scanned alert should be equal to one of the source or destination IP addresses of the alert being examined. The equality may exist between the source IP of one alert and the destination IP of the other, or vice versa. Because it is not always the case that only outgoing or only incoming packets produce the alerts, it is possible to find two related alerts, produced by traffic of different directions. In this case their source and destination IP addresses equalities will hold crosswise.

The condition for the two related alerts to have different signatures prevents alerts that reappear many times from being wrongly estimated by the component. If this condition were not present, then if the component examined an alert that appeared many times, which is a very common scenario in intrusion detection alert-sets, it would count its past appearances as related alerts and would produce a high validity score for it.

In order to produce the component validity score the percentage of related alerts out of all alerts in the previous tw_{NRA} time range is calculated. The percentage is normalized by using as maximum value the dl_{NRA}.

$$density = \frac{\text{number of related alerts in } tw_{NRA}}{\text{number of alerts in } tw_{NRA}} \tag{6.7}$$

$$score_{norm} = \frac{min(density, dl_{NRA})}{dl_{NRA}} \tag{6.8}$$

The logic of the NRA component is shown in Algorithm 6.1.

```
1: i is the index of alert being examined
2: $time_{now} \leftarrow alert_{time}(i)$
3: $neigbours \leftarrow 0$
4: $counter_{run} \leftarrow i - 1$
5: $time_{diff} \leftarrow alert_{time}(i) - alert_{time}(counter_{run})$
6: while $time_{diff} < tw_{NRA}$ do
7:   if $alert_{sip}(i) == alert_{sip}(counter_{run}) \vee alert_{sip}(i) == alert_{dip}(counter_{run}) \vee alert_{dip}(i) == alert_{sip}(counter_{run}) \vee alert_{dip}(i) == alert_{dip}(counter_{run})$ then
8:     if $alert_{sig}(i) \neq alert_{sig}(counter_{run})$ then
9:       $neigbours \leftarrow neighbours + 1$
10:    end if
11:  end if
12:  $counter_{run} \leftarrow counter_{run} - 1$
13:  $time_{diff} \leftarrow alert_{time}(i) - alert_{time}(counter_{run})$
14: end while
15: $density = \frac{neigbours}{i - counter_{run} + 1}$
16: $score = \frac{min(density, dl_{NRA})}{dl_{NRA}}$
```

Algorithm 6.1: Algorithm of NRA component

6.2.3 NRAS component

NRAS is similar to the NRA component, but some of the rules that should hold for an alert to be counted as related to the one examined are loosened. The rationale behind this is to produce high scores for specific kinds of attacks, in which produced alerts may relate in an uncommon way, e.g. sweeping attacks.

Specifically the conditions that should hold for assuming two alerts as related are:

- The two alerts should have one equality either between their source IPs or their destination IPs, while the difference of the IPs in the other pair should be less than 256.
- The signatures of the two alerts should be different.

Usually in sweeping attacks the intruder scans the whole IP range of a network and takes action against hosts found alive. This means that the the produced alerts will probably share the same source IP, while their destination IPs will have continuous values. If the outgoing traffic produces the alerts then they will have common destination IPs and their source IPs will have continuous values. Usually sweeping is committed for /24 networks, so the relevant condition demands the distance between varying IPs to be less than 256.

As it has been done in NRA it is required for the signatures of the examined alerts to be different, to consider the alerts relevant. For sweeping attacks it is common to produce multiple alerts with the same signature but varying IP addresses. It has been opted not to use the existence of such alerts as a validity indication, because it would hinder correct estimation for the majority of alerts, as it has been analysed in the previous Subsection.

The NRAS component is also configured by the same parameters as the NRA component, dl_{NRAS} and tw_{NRAS}. They are calculated as it is shown in Equations 6.9, 6.10. These relations have been decided empirically, by observing the component performance on various alert-sets.

$$dl_{NRAS} = \frac{0.4}{var_{alerts}} \tag{6.9}$$

$$tw_{NRAS} = \frac{1000}{den_{alerts}} \tag{6.10}$$

The operation of NRAS component is identical to that of the NRA component, except of the part of deciding the relation or not between two alerts. The logic of the NRA component is Algorithm 6.2.

```
1: i is the index of alert being examined
2: $time_{now} \leftarrow alert_{time}(i)$
3: $neigbours \leftarrow 0$
4: $counter_{run} \leftarrow i - 1$
5: $time_{diff} \leftarrow alert_{time}(i) - alert_{time}(counter_{run})$
6: while $time_{diff} < tw_{NRAS}$ do
7:    if $(alert_{sip}(i) == alert_{sip}(counter_{run}) \wedge |alert_{dip}(i) - alert_{dip}(counter_{run})| < 256) \vee$
      $(alert_{dip}(i) == alert_{dip}(counter_{run}) \wedge |alert_{sip}(i) - alert_{sip}(counter_{run})| < 256)$ then
8:       if $alert_{sig}(i) \neq alert_{sig}(counter_{run})$ then
9:          $neigbours \leftarrow neighbours + 1$
10:      end if
11:   end if
12:   $counter_{run} \leftarrow counter_{run} - 1$
13:   $time_{diff} \leftarrow alert_{time}(i) - alert_{time}(counter_{run})$
14: end while
15: $density = \frac{neigbours}{i - counter_{run} + 1}$
16: $score = \frac{min(density, dl_{NRAS})}{dl_{NRAS}}$
```

Algorithm 6.2: Algorithm of NRAS component

6.2.4 HAF component

A metric used in the HAF component is the signature-related frequency. For a specific alert ale with time-stamp ale_{time} and signature ale_{sig}, the signature-related frequency is the frequency with which alerts with signature ale_{sig} appear in a time window around ale_{time}.

For the time window [$ale_{time} - d_t$,$ale_{time} + d_t$] the signature-related frequency for signature ale_{sig} can be calculated by the number of alerts that exist in the time window and carry signature ale_{sig}. If this number is denoted by n then the signature-related frequency $fre_{ale_{time}}^{ale_{sig}}$ for the specific signature at the specific time can be calculated as shown in Equation 6.11.

$$fre_{ale_{time}}^{ale_{sig}} = \frac{n}{2 * d_t} \tag{6.11}$$

The functioning of HAF is based on the observation that it is more probable for an alert to be a true positive if it appears in higher frequency compared to the mean frequency of alerts describing the same attack (alerts of the same signature). In other words, an occurring attack increases the signature-related frequency of the alerts it produces.

The HAF component is configured by one parameter, the frequency limit fre_{limit}. This is calculated as shown in Equation 6.12. This relation has been decided empirically, by observing the performance of the component on various alert-sets.

$$fre_{limit} = \frac{5}{den_{alerts}} \tag{6.12}$$

In order for the HAF component to function, the Sensor Manager needs to hold signature-related frequencies and update them as new alerts come in. This is done by the SFH component that will be analysed in the next section. The HAF component calculates for each alert the relative current signature frequency. It then compares it to the average frequency for the signature of the alert, that is returned by the SFH component. The ratio of the current to the average frequency is then used as indication of validity for the examined alert.

In order to calculate the current signature-related frequency for an alert the component should scan alerts in a time window before it. It should then sum up the appearances of the alert's signature and divide them by the number of alerts existing in this window. For efficiency reasons this logic has not been used.

Instead the component calculates the time distance *timedist* of the examined alert to the closest alert that shares the same signature. In most of the cases, this provides a satisfactory estimate of the current signature-related frequency. The time distance is divided by 1000 as it is in milliseconds and 1 sec is added to avoid division by 0, in cases where two alerts with the same signature share the same time-stamp. If no alert with the same signature exists in the alert-set then the frequency is set to 0. The calculation of current signature-related frequency is as shown in Equation 6.13.

$$fre^{cur}(sig) = \begin{cases} \frac{1}{1+\frac{timedist}{1000}}, \text{if an alert with same sig exists} \\ 0, \text{if an alert with same sig does not exist} \end{cases} \tag{6.13}$$

Regarding the average frequency, 0 is returned from the SFH component if no other appearance of the specific signature exists. In this case the ratio value is set to 1. The frequencies ratio is calculated as shown in Equation 6.14.

A limit is used to normalize the outcome of the ratio of the current to the average frequency, the value of fre_{limit}. The normalized frequency ratio, which is the final validity score outputted by the component, is calculated by Equation 6.15

$$fre^{ratio} = \begin{cases} \frac{fre^{cur}(sig)}{fre^{avg}(sig)}, if fre^{avg}(sig) > 0 \\ 1, otherwise \end{cases} \tag{6.14}$$

$$score_{norm} = \frac{min(fre^{ratio}, fre_{limit})}{fre_{limit}} \tag{6.15}$$

The logic of the HAF component is presented in Algorithm 6.3.

6.2.5 SFH component

The SFH component is an auxiliary component that provides information to the HAF component. Its mission is to hold signature-related frequency values for all signatures and update them at each new alert. When a new alert arrives, the HAF component requests from SFH to provide it with the average signature-related frequency for the alert under examination, which is calculated by previous alerts.

```
i is the index of alert being examined
$i_{run} \leftarrow i - 1$
$found \leftarrow false$
while $i \geq 0 \wedge found = false$ do
    if $alert^{sig}(i_{run}) = alert^{sig}(i)$ then
        $found \leftarrow true$
    end if
end while
if $found = false$ then
    $fre^{cur} \leftarrow 0$
else
    $timedist = alert^{sig}(i) - alert^{sig}(i_{run})$
    $fre^{cur} \leftarrow \frac{1}{1+\frac{timedist}{1000}}$
end if
Read $fre^{avg}$ from SFH
if $fre^{avg} > 0$ then
    $score_{HAF} \leftarrow \frac{min(\frac{fre^{cur}}{fre^{avg}}, fre_{limit})}{fre_{limit}}$
else
    $score_{HAF} \leftarrow 1$
end if
```

Algorithm 6.3: Algorithm of HAF component

The component holds the number of appearances of each signature in a hash table. Each hash table record consists of a key-value pair. The key is the signature and the value is the number of its appearances. The component runs periodically to update this table. The period used, denoted by T_{SFH}, is calculated using the density of the alert-set, as shown in Equation 6.16.

$$T_{SFH} = \frac{0,3}{den_{alerts}} \tag{6.16}$$

In order to update the hash table SFH stores $index_{last}$, the index of the last alert of the alert-set during its last update. At each run it checks if any new alerts have been produced by the sensor. If there are new alerts, then it scans them and for each one of them it increases the appearances of its signature by one. If the signature of a new alert is not present in the hash table, which means that SFH examines the first alert carrying it, then a new key-value pair (sig,1) is added to the table. The $index_{last}$ is updated to the last alert of the alert-set.

When an average frequency is requested for a signature by the HAF component SFH recalls the number of appearances for the specific signature, divides it by the total number of alerts in the alert-set and returns the result to HFA.

6.2.6 RFP component

False Positives usually derive from causes that may be related to the topology of the network, mis-configured hosts or periodical lawful services and tasks that are carried out in the network. All these causes are time invariant and produce recurrent patterns of false positives. The rationale of the RFP component is to use these patterns, in order to detect false positives. If the alerts produced for a specific network are examined thoroughly for a long period, then the signatures that consistently produce alerts in identifiable patterns can be detected.

The RFP component examines the alert-set and detects patterns of recurring alerts that could have been produced by persistent causes, existing in the protected network. For each one of the discovered false positives patterns the component calculates a validity score fpp_{val}. This score corresponds to the probability that alerts matching the pattern are in fact false positives. For every new alert the component checks if it matches to any of these patterns. If it does, the component calculates a relatively low validity score for it, using the fpp_{val} value of the pattern. Otherwise the validity score produced for an alert for which no match occurs is equal to 1.

During the detection of the pattern, an important parameter is the average period between recurring alerts. This should not be too small as multiple patterns would be produced by bursts of alerts. It is common for many identical alerts, sharing the same time-stamp or having minimal differences in time, to have been produced by a single event. In these cases no pattern should be detected by the RFP component.

Even if continuously searching for new patterns in the growing alert-set is not a lightweight task for the system, the component keeps updating the patterns found at each new alert. This ensures that no false positives will go unnoticed because of slow pattern detection.

In order to find the patterns the alert-set is scanned from first to last alert. The patterns consist of three values, namely signature, source IP and destination IP. An important assumption for the efficiency of the component is that these patterns should refer to alerts of low danger (with low class id values). Periodic lawful tasks in a network usually produce such alerts and additionally this is a safety choice, in order not to detect as false positive any recurring dangerous alert.

The conditions for a pattern to be detected are :

- The class id ale_{cid} of the alert should be less or equal to 2.
- There should be at least ten alerts in the alert-set that match to the pattern (carrying pattern's sig, sip, dip).
- The average period of the recurring patterns should be more than 2 seconds.

If these conditions are met then a false positives pattern validity score fpp_{val} is calculated with respect to its average period. It is considered that a period of 15 seconds is an appropriate representative value for periods of recurring nominal tasks. A function built upon two different Gaussian distributions is used in order to transform the average period value of the pattern to the pattern validity fpp_{val} score. The score should be low for patterns with periods close to 0 seconds, it should then peak at values equal to 1 for patterns with periods close to 15 seconds and then slowly approximate 0 again for period values around 5 minutes. The function used is shown in Equation 6.17.

$$fpp_{val} = \begin{cases} \exp -\frac{(per_{avg}-15)^2}{2*4^2}, & \text{if } per_{avg} < 15. \\ \exp -\frac{(per_{avg}-15)^2}{2*100^2}, & \text{otherwise.} \end{cases} \tag{6.17}$$

The graphical representation of this function is shown in Figure 6.2.

After the rescanning of the updated alert-set for new possible patterns the component checks the new alert against all patterns. If a match exists the validity score for the alert is calculated as shown in Equation 6.18. Otherwise the component sets the validity score for the alert equal to 1. The score produced by the component is representative of the probability of the alert being true, so high fpp_{val} should produce high scores and vice versa.

$$score_{RFP} = \begin{cases} 1 - fpp_{val}, & \text{if a match exists.} \\ 1, & \text{otherwise.} \end{cases} \tag{6.18}$$

The logic of the RFP component is shown in Algorithm 6.4.

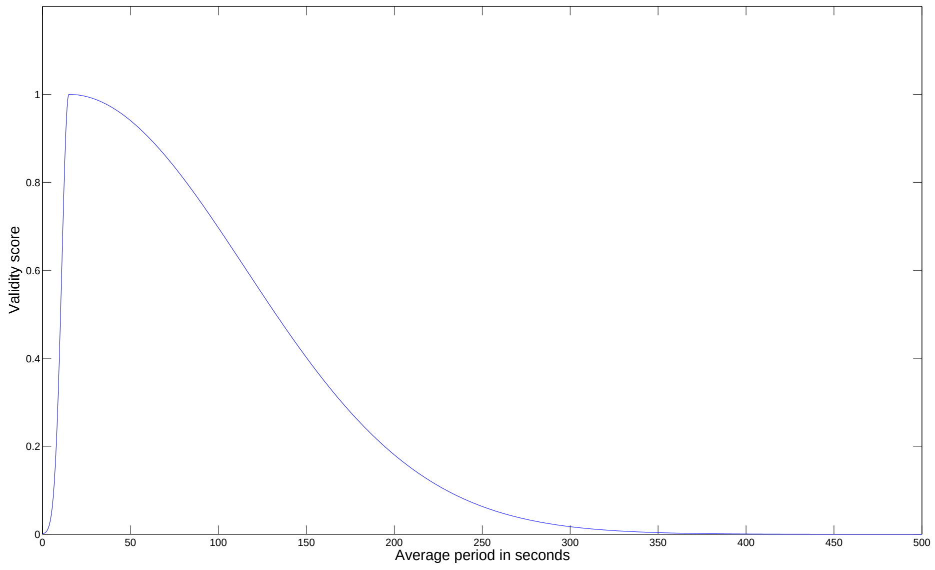

Figure 6.2: False positives pattern validity function

6.2.7 Weights updater and Validity score calculator

The Weights updater is the component responsible for updating the weights used to fuse the four scores produced by the NRA, NRAS, HAF, RFP components to one final validity score. The four weights sum up to one, while at the start of the functioning of the system, they are all initiated to 0,25.

$$w_{NRA} + w_{NRAS} + w_{HAF} + w_{RFP} = 1 \tag{6.19}$$

$$w_{NRA}{}^{init} = w_{NRAS}{}^{init} = w_{HAF}{}^{init} = w_{RFP}{}^{init} = 0,25 \tag{6.20}$$

The Weights updater keeps monitoring the score values produced by each component. It adjusts the weights, in order to increase the influence to the final score for components that seem to perform better, while decreasing the corresponding influence for components that seem not to perform well.

The Weights updater checks the validity scores produced by each component. It takes into account a number of alerts (most recent ones) and calculates a deviation metric for each component. The main idea behind this is that a component that is not able to cope with a specific running taxonomy of alerts will probably consistently produce too low or too high validity score values. Low deviation is taken as an indication of low performance, while a higher one is a characteristic that should increase the weight of the corresponding component.

The four deviation values calculated are normalized to sum up to one. In the case of alerts' series that force all components to low deviation their weights won't be lowered as during the normalization phase they will be forced to sum up to 1.

The Weights updater keeps a record of the last n_0 validity scores produced by each one of NRA,NRAS,HAF and RFP ($n_0 = 500$ has been used). As a new score comes in the first one of the n_0 scores stored is discarded and the new one is added to the list. Then the weights deviation w^{dev} is calculated for each one of the four components. In Equation 6.21 the calculation of the mean of weights is shown for the NRA component, while in Equation 6.22 the deviation for the same component is calculated.

$$w^{mean}{}_{NRA} = \frac{\sum_{i=1}^{i=n_0} score^{NRA}(i)}{n_0} \tag{6.21}$$

$$w^{dev}{}_{NRA} = \sqrt{\frac{\sum_{i=1}^{i=n_0} (score^{NRA}(i) - w^{mean}_{NRA})^2}{n_0}} \tag{6.22}$$

The same calculation is conducted for the other three components. Then the normalization takes place. Each of the four components deviation is divided by their sum to produce the final weight. This is shown for NRA in Equation 6.23

$$w_{NRA} = \frac{w^{dev}_{NRA}}{w^{dev}_{NRA} + w^{dev}_{NRAS} + w^{dev}_{HAF} + w^{dev}_{RFP}} \tag{6.23}$$

After calculating the four updated weights, the Weights updater sends them to the Validity score calculator, which uses them to combine the four different scores. The final score is computed by Equation 6.24.

$$score_{final} = w_{NRA} * score_{NRA} + w_{NRAS} * score_{NRAS} + w_{HAF} * score_{HAF} + w_{RFP} * score_{RFP} \tag{6.24}$$

The functioning of this component is simple and could have been incorporated in another component. It has been implemented in a separate component, in order to test more complex fusion ideas for the four scores in the future.

6.3 Aggregating alerts

The second main task of the Sensor manager is to aggregate identical alerts, produced by the corresponding sensor. The AGC component transforms such groups of identical alerts to aggregated alerts that hold all the information that existed on identical alerts aggregated along with other data such as their number and the time space in which they existed. A validity score is attached to the produced aggregated alerts, by calculating the mean value of the validity scores of alerts aggregated.

The task of the AGC component is to compare on the fly incoming alerts to close in time previous ones and to discover alerts candidate for aggregation. While the comparison is a relatively simple procedure, attention has been focused on enabling the component to be effective even if high volume of alerts is produced by the sensor.

It is essential to prevent latency being introduced in the system functioning at such an initial stage of the procedure. Otherwise this latency will be propagated and multiplied in next components that accept as input the output of the AGC components. For this reason AGC component keeps two lists for its aggregated alerts. A live aggregated alerts list contains all recent aggregated alerts that are eligible for aggregating to them future alerts. A final aggregated alerts list is used to move aggregated alerts from the live list, when they become too old for aggregating future alerts to them.

The AGC component sends aggregated alerts to the Clustering subsystem as soon as they become final and no further evolution is possible. Practically alerts that are produced from sensors are grouped to aggregated alerts, when they are identical to their neighbouring alerts. If no match exists they are transformed to aggregated alerts (containing only one alert). The produced aggregated alerts are then propagated to the Clustering subsystem, with a small delay to ensure that no other single alert should be aggregated to them.

The component upon each new alert arrival performs two tasks. The first one is to scan the live aggregated alerts' list for too old entries and move the ones found to the final aggregated alerts list. The second task is to examine if the new alert can be aggregated to any of the aggregated alerts, remaining in the live list, or if a new aggregated alert should be created.

The aggregated alerts list update is committed before trying to aggregate each new alert. There is a $time_{width}$ parameter that is used for deciding the discarding of a live aggregated alert. If the end time of an aggregated alert is more than $time_{width}$ old then this aggregated alert is removed from live list and added to the final list. In the implementation of the system this value has been set to 120 seconds, assuming that differences between identical alerts produced by the same event should not exceed this value.

$$time_{width} = 120 \text{ secs} \tag{6.25}$$

In order to add the new alert to the aggregated alert-set the component scans the live aggregated alerts list and finds the most recent aggregated alert that matches with the new alert. The matching conditions are:

- The new alert and the aggregated alert should share the same signature.
- The new alert and the aggregated alert should share the same source and destination IP addresses.

If a matching aggregated alert is found then the new alert is added to it. In practice the end time of the aggregated alert is set equal to the time of the new alert, its number of alerts is increased by one, and its truth value is updated to be the average of the scores of all alerts, as shown in Equation 6.26

$$truth_{new} = \frac{aga_{truth} * aga_{num} + ale_{truth}}{aga_{num} + 1} \tag{6.26}$$

If no matching alert is found then a new aggregated alert is created. Sensor id, Signature, Class id, Source IP, Destination IP and Truth attributes inherit their values from the initial alert. Start time and end time of the aggregate alert are set equal to the time field of the initial alert. The number of alerts value is obviously set to be equal to 1.

The logic of aggregation component is shown in Algorithm 6.5.

```
{Updating found patterns}
$patterns_{scanned} \leftarrow$ new empty patterns list
$size \leftarrow$ size of alert-set
for $i = 1 \rightarrow size$ do
  $alert^{base} \leftarrow alert(i)$
  if $alert^{base}_{classid} \leq 2$ then
    $pat \leftarrow$ pattern matching $alert^{base}$
    if $\neg patterns_{scanned}$ contains pat then
      add pat to $patterns_{scanned}$
      $times \leftarrow$ new empty time values list //list of times of appearances
      $time_{last} \leftarrow alert^{base}_{time}$ // last time value scanned
      add $alert^{base}_{time}$ to $times$
      for $j = i + 1 \rightarrow size$ do
        $alert_{run} \leftarrow alert(j)$
        if $alert^{run}$ matches to pat $\wedge alert^{run}_{time} > time_{last}$ then
          $time_{last} \leftarrow alert^{run}_{time}$
          add $alert^{run}_{time}$ to $times$
        end if
      end for
      if size of times > 10 then
        $per_{avg} \leftarrow$ average period between values of times
        if $per_{avg} < 15$ then
          $fpp_{val} \leftarrow \exp -\frac{(per_{avg}-15)^2}{2*4^2}$
        else
          $fpp_{val} \leftarrow \exp -\frac{(per_{avg}-15)^2}{2*100^2}$
        end if
        add pat,$fpp_{val}$ to found patterns Hashmap
      end if
    end if
  end if
end for
{Checking new alert}
if last alert matches to a pattern $fpp^{match}$ then
  $score_{RFP} \leftarrow 1 - fpp^{match}_{val}$
else
  $score_{RFP} \leftarrow 1$
end if
```

Algorithm 6.4: Algorithm of RFP component

```
{Updating live aggregated alert}
$i \leftarrow$ index of last live aga
Read system's time $t_{now}$
while $i \geq 0$ do
    if $t_{now} - aga_{eti}(i) > time_{width}$ then
        remove aga(i) and add it to final list
    end if
    $i \leftarrow i - 1$
end while
{Aggregating new alert}
$found \leftarrow false$
$i \leftarrow$ index of last live aga
while $i \geq 0 \wedge \neg found$ do
    if $aga_{sip}(i) = alert_{sip} \wedge aga_{dip}(i) = alert_{dip} \wedge aga_{sig}(i) = alert_{sig}$ then
        $aga_{truth}(i) \leftarrow \frac{aga_{truth}(i)*aga^{num}(i)+alert_{truth}}{aga_{num}(i)+1}$
        $aga_{num}(i) \leftarrow aga_{num}(i) + 1$
        $aga_{eti}(i) \leftarrow alert_{time}$
        $found \leftarrow true$
    end if
    $i \leftarrow i - 1$
end while
if found=false then
    Create new $aga$
    $aga_{sid} \leftarrow alert_{sid}$
    $aga_{sig} \leftarrow alert_{sig}$
    $aga_{cid} \leftarrow alert_{cid}$
    $aga_{sip} \leftarrow alert_{sip}$
    $aga_{dip} \leftarrow alert_{dip}$
    $aga_{truth} \leftarrow alert_{truth}$
    $aga_{sti} \leftarrow alert_{time}$
    $aga_{eti} \leftarrow alert_{time}$
    $aga_{num} \leftarrow 1$
    add aga to live aggregated alerts
end if
```

Algorithm 6.5: Algorithm of AGC component

CHAPTER 7

Clustering subsystem

The clustering subsystem clusters related aggregated alerts, to produce a more meaningful representation of the security state of the system. The approach used is similarity based clustering. The subsystem monitors the nature of incoming aggregated alerts and adjusts its operation to it. Its main objective is to produce one or at most a few clusters for each occurred event. Moreover the Clustering subsystem attempts to hypothesize on events missed by the intrusion detection sensors and estimate parameters of missing clusters. The combined clusters' list is stored in the database of the system, to be read by the Visualisation subsystem.

7.1 Design of the Clustering subsystem

The Clustering subsystem is responsible for transforming alerts, that correspond to low level events, to clusters, that correspond to higher level intruder actions. Its functioning is fundamental for the production of a reasonable representation at the output of the system. The wrong level of clustering (producing either too many or too few clusters) is capable of destroying the final representation, even if all other subsystems perform well.

Along with conducting the necessary clustering to present information about detected events in an efficient manner, this subsystem tries to hypothesize on events that have been completely missed by the intrusion detection sensors and to supply the security analyst with estimated information about these events. This hypothesis and estimation procedure should also be fine-tuned, because otherwise there is significant risk to flood the real alert-set with artificial data.

The Clustering subsystem may function in either simple or advanced mode. In simple mode clusters generation for hypothesizing on missed events is omitted, while in advanced mode it is incorporated. Then main components of this subsystem are depicted in Figures 7.1 and 7.2. The first figure shows simple mode functioning, where only Merger, Clusterer and Weights updater are used. In the second Figure advanced mode functioning is depicted where Clusters generator is also used.

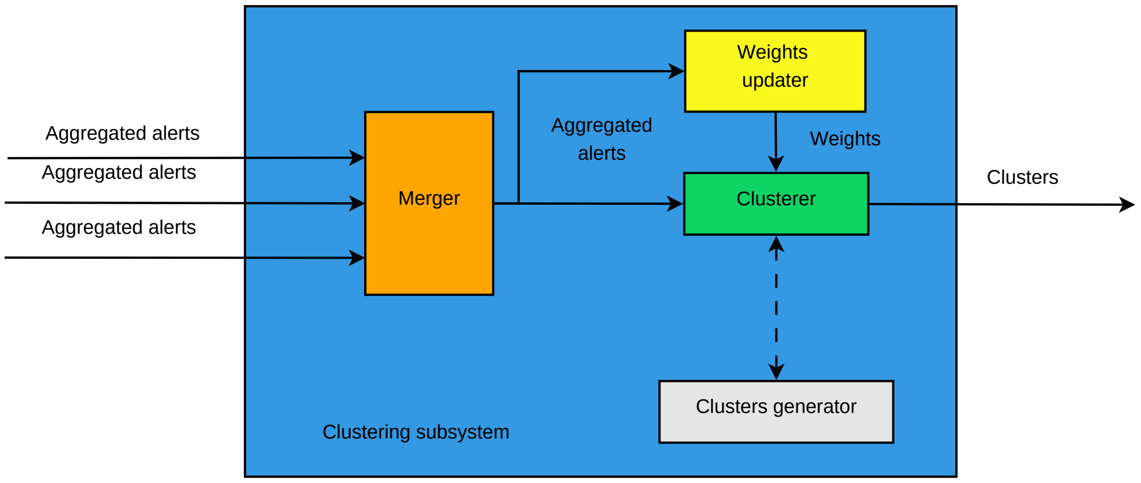

Figure 7.1: Clustering subsystem (simple mode)

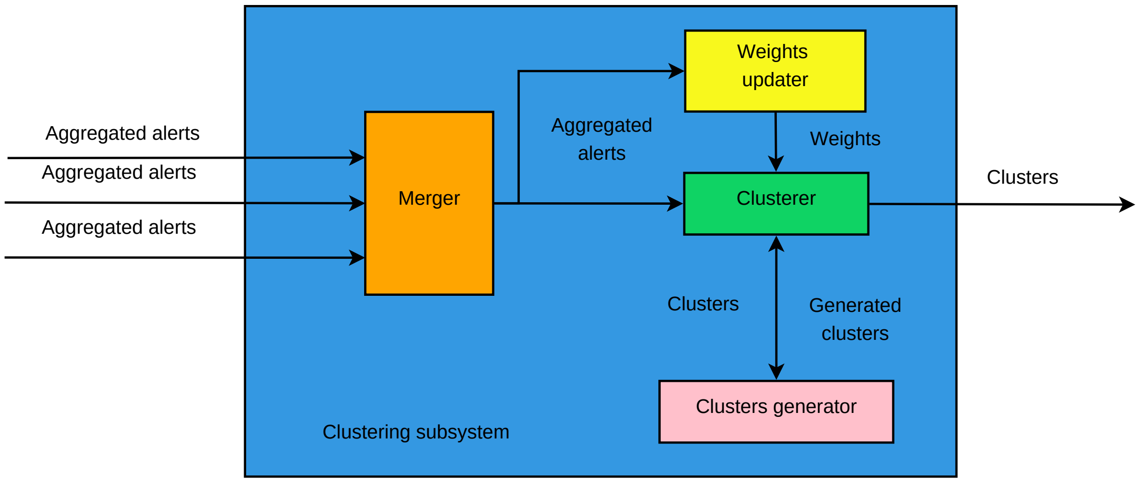

Figure 7.2: Clustering subsystem (advanced mode)

7.1.1 Simple mode

The component that accepts the output of the previous subsystem, is the Merger. It is responsible for merging multiple flows of Aggregated alerts coming from the corresponding

Sensor managers into one single flow. During merging, Aggregated alerts, that seem to refer to the detection of the same event by different sensors, are discarded.

The Aggregated alerts produced by the Merger are sent to the Clusterer. Each step in the plan of an intruder may produce more than one different alerts, which are propagated through previous components as different Aggregated alerts. Additionally consecutive steps may also produce related alerts. The Clusterer tries to construct clusters of groups of alerts related to each other. For each new alert it calculates similarity values to all existing clusters that have been formed from previous alerts and according to these values it either adds the alert to an existing cluster or it creates a new cluster consisting only of this single alert.

The Clusterer runs periodically every two seconds. The output of each run is a set of clusters that corresponds to the security state of the protected system from the start of monitoring up to the time of the run. The set of clusters is stored in the database, from where it is continuously read by the Visualization subsystem, which consequently updates a graph of the security state of the protected system.

The Weights updater is responsible for optimizing weight parameters used by the Clusterer. The latter needs to be configured according to the nature of the Aggregated alerts' set. The Weights updater checks the set everytime a new alert arrives and calculates values of parameters representative of the nature of the alert set. Based on these parameters the component recalculates the weights to be used in the Clusterer.

7.1.2 Advanced mode

In the advanced mode an additional component is used, namely the Clusters generator. Its task is to estimate information about security events missed by intrusion detection sensors.

The idea behind hypothesizing on missed events is that most of the time these events are part of a general attack plan, so they are correlated to the steps that come before or after them in this plan. By examining the detected events and searching for logical gaps between them, the Clusters generator can detect cases where an event has been probably missed and estimate information about it.

This component examines the clusters' set periodically and discovers candidate cases for missing events. The detection is carried out by checking for all possible pairs of clusters, that have a minimum time difference, if there is detected activity between them that seems to logically connect them, or not. If no such activity exists, then the pair is assumed as a candidate case for cluster generation. For each one of the candidate cases the steps described below are followed:

- An artificial cluster is created in the form of generated cluster described in Table 5.1. Its parameters are calculated by the parameters of the real clusters of the relevant pair.
- The artificial clusters generated in the previous step are examined in terms of the quality of information they add to the alert-set. A quality metric has been defined for this examination and clusters with higher quality are propagated to the next step, while the rest are discarded.
- The clusters that have not been discarded at the previous step are fine tuned in order to touch up their quality of information without significantly altering their state.

7.2 Merger

The task of the Merger is to combine multiple flows of aggregated alerts, into a single flow. The main issue in this procedure is to avoid incorporating into the produced aggregated alerts set multiple similar aggregated alerts, describing the same event.

The Merger accepts input from more than one Sensor managers. It provides them with an interface to add their aggregated alerts to its alert-set. As it has been described in Section 6.3 the AGC component keeps a list of live aggregated alerts. As soon as an alert is discarded from this list, which means that it will not change in the future, it is also sent to the Merger by using this interface.

Practically the Merger adds each incoming aggregated alert to its global aggregated alerts list. Before doing so, it checks whether there is an existing similar aggregated alert added by another Sensor manager in the list. If there is, the new aggregated alert is discarded. If there is not, the new alert is added to the global list and at the same time it is sent to the Clustering component.

The Merger is configured by a time width value $time_{width}$, which is the maximum time difference allowed for two alerts to be considered as similar and eventually discard one of them. Using a high value for this parameter will decrease the performance of the system, while it will ensure that no identical alerts will be propagated to next components, even if there is a larger time difference between them. During the implementation of the component, this $time_{width}$ parameter value has been set to 10 seconds.

$$time_{width} = 10 \text{ seconds} \tag{7.1}$$

The logic of handling a new alert, coming in the component, is shown in Algorithm 7.1.

```
1: $aga \leftarrow$ existing aggregated alerts list
2: $a \leftarrow$ new incoming alert
3: $mark\_for\_deletion \leftarrow false$
4: $size \leftarrow aga.size()$
5: $i \leftarrow size - 1$
6: while $i > 0 \wedge aga_{sti}(i) > a_{eti} + time_{width}$ do
7:    if $(aga_{sip}(i) = a_{sip}) \wedge (aga_{dip}(i) = a_{dip}) \wedge (aga_{sig}(i) = a_{sig})$ then
8:       $mark\_for\_deletion \leftarrow true$
9:    end if
10:   $i \leftarrow i - 1$
11: end while
12: if $mark\_for\_deletion = false$ then
13:    aga.add(a)
14:    Send a to next component
15: end if
```

Algorithm 7.1: Merger algorithm

7.3 Clusterer

The Clusterer component decides which of the aggregated alerts produced by the Merger are related to each other and groups them to relative clusters. Similarity based clustering has been chosen as the methodology that efficiently detects relations between aggregated alerts. A similarity metric is defined between an aggregated alert and a cluster, the values of which range from 0 to 1.

The approach used is to add each of the incoming aggregated alerts to one of the existing clusters. If the aggregated alert is not relevant enough to any cluster then a new one is created by it. There is a similarity value sim_{thre} that is used as the minimum threshold for an aggregated alert to be clustered to an existing cluster. In the implementation of the Clusterer component a sim_{thre} value equal to 0.6 was used. If for a new aggregated alert the similarity values to all existing clusters are below sim_{thre}, then a new cluster is created, otherwise the alert is added to the cluster to which the larger similarity value corresponds. The logic of handling each new aggregated alert is shown in Algorithm 7.2.

```
1: $aga \leftarrow$ incoming aggregated alert
2: $clu \leftarrow$ existing cluster's list
3: $size \leftarrow clu.size()$
4: $i \leftarrow 0$
5: $sim_{max} \leftarrow 0$
6: $index_{found} \leftarrow -1$
7: while $i < size$ do
8:     $sim \leftarrow sim(aga, clu(i))$
9:     if $(sim > sim_{thre}) \wedge (sim > sim_{max})$ then
10:        $index_{found} \leftarrow i$
11:        $sim_{max} \leftarrow sim$
12:    end if
13:    $i \leftarrow i + 1$
14: end while
15: if $clu_{found} = -1$ then
16:    $clu_{new} \leftarrow create\_new\_cluster(aga)$
17:    $clu.add(clu_{new})$
18: else
19:    $clu(index_{found}).add(aga)$
20: end if
```

Algorithm 7.2: Clustering a new aggregated alert

7.3.1 Creating a new cluster from an aggregated alert

In order to create a cluster from a new aggregated alert, the cluster's fields values are concluded from the values of the corresponding alert's fields values. Specifically:

- The list of cluster's signatures is initiated with only one value, the signature of the aggregated alert.
- The list of cluster's class ids is initiated with only one value, the class id of the aggregated alert.
- The Cluster's start time is set equal to aggregated alert's start time.

- The Cluster's end time is set equal to aggregated alert's end time.
- List of cluster's source IPs is initiated with only one value, the source IP of the aggregated alert.
- List of cluster's destination IPs is initiated with only one value, the destination IP of the aggregated alert.
- Cluster's truth is set equal to the aggregated alert's truth.
- Cluster's alert number is set equal to the aggregated alert's number of alerts.
- Auxiliary cluster's fields are updated accordingly.

7.3.2 Adding an aggregated alert to an existing cluster

If the similarity of the examined aggregated alert to at least one of the clusters is more than the threshold value, then it is added to the cluster with the highest similarity value. In order to update the cluster's fields values the corresponding values of the aggregated alert's fields are used:

- Aggregated alert's signature is added to the list of cluster's signatures, if it is not already included.
- Aggregated alert's class id is added to the list of cluster's class ids, if it is not already included.
- If aggregated alert's start time is earlier that cluster's start time then the latter is set equal to aggregated alert's start time.
- If aggregated alert's end time is later that cluster's end time then the latter is set equal to aggregated alert's end time.
- Aggregated alert's source IP is added to the list of cluster's source IPs, if it is not already included.
- Aggregated alert's destination IP is added to the list of cluster's destination IPs, if it is not already included.

- Cluster's truth is calculated by the it's initial truth value and the aggregated alert's truth value.

$$clu_{truth}^{new} = \frac{clu_{num} * clu_{truth} + aga_{num} * aga_{truth}}{clu_{num} + aga_{num}} \tag{7.2}$$

- Cluster's alert's number is set equal to the sum of its initial alert's number and the number of alerts of the aggregated alert.

- Auxiliary cluster's fields are updated accordingly.

7.3.3 Calculating similarity

The main parameter defining the functioning of the Clusterer is the similarity between an aggregated alert aga and a cluster clu. This similarity sim(aga,clu) is decided by individual similarities relative to IP, time and signature values of the aggregated alert and the cluster. Specificaly:

$$sim(aga, clu) = w_{IP} * sim_{IP}(aga, clu) + w_{time} * sim_{time}(aga, clu) + w_{sig} * sim_{sig}(aga, clu) \tag{7.3}$$

Each one of the three individual similarities is calculated upon the respective values of the aggregated alert and the cluster. Additionally there are three corresponding weight values that are updated by the Weights updater component. They are used to alter the similarity calculation centre of gravity according to the nature of the alert-set. The calculation of each one of the three individual similarities is analysed as follows:

IP similarity

The aggregated alert has a source and a destination IP address, aga_{sip} and aga_{dip}. The IP similarity of these to the IP addresses of the cluster is calculated and the maximum value is picked as the similarity value for the aggregated alert.

$$sim_{IP}(aga, clu) = max(sim_{IP}(aga_{dip}, clu), sim_{IP}(aga_{sip}, clu)) \tag{7.4}$$

In order to calculate these two similarities the component needs minimum and maximum internal IPs of the cluster along with its minimum and maximum external IPs.

The first two exist in the **Cluster** data format and are automatically updated, as they are required by the Visualization subsystem. The limits for the external IPs (clu_{minei} and clu_{maxei}) are calculated before calculating the similarity. When examining the similarity of an internal IP to a cluster, the internal minimum and maximum IPs of the cluster are utilized. On the other hand external minimum and maximum IPs of the cluster are used when the similarity of an external IP is calculated.

If the IP in examination is inside the IP range of the cluster used, then 1 is returned as similarity. If the IP is outside the IP range of the cluster then the multiplicative inverse of its distance from it is returned as similarity. The distance of the IP from the range of the IPs of a cluster is equal to the minimum of its distances from the two limits of the range as shown in Equation 7.5.

$$dist_{IP}(IP, range_{min}, range_{max}) = min(|IP - range_{min}|, |IP - range_{max}|) \tag{7.5}$$

The similarity of an internal IP to a cluster clu is calculated as shown in Equation 7.6.

$$sim_{IP}(IP, clu) = \begin{cases} 1, & \text{if } clu_{minii} \geq IP \geq clu_{maxii} \\ \frac{1}{dist_{IP}(IP, clu_{minii}, clu_{maxii})}, & otherwise \end{cases} \tag{7.6}$$

If the IP is an external IP the same equation holds, by replacing clu_{minii} and clu_{maxii} with clu_{minei} and clu_{maxei} respectively.

Time similarity

In order to calculate the similarity of time values between an aggregated alert and a cluster, the time ranges (start time to end time) of both are compared. Two factors decide time similarity, namely the percentage of the time range of the aggregated alert that overlaps with the time range of the cluster and the distance between the two ranges if they don't overlap. These two factors are expressed in two variables: relativeness factor rf and distance d.

The rf value is calculated as shown in Equation 7.7, while the d value is calculated as shown in Equation 7.8.

$$rf = \frac{\text{length of part of aga time range that overlaps with clu}}{\text{aga time range}} \tag{7.7}$$

$$d = \begin{cases} aga_{sti} - clu_{eti}, & \text{if aga is after clu} \\ clu_{sti} - aga_{eti}, & \text{if aga is before clu} \\ 0, & \text{if aga and clu overlap} \end{cases} \tag{7.8}$$

The time similarity is calculated by these two parameters as shown in Equation 7.9.

$$sim_{time} = \frac{rf - \frac{max(d,d_0)}{d_0} + 1}{2} \tag{7.9}$$

The d_0 parameter is used to normalise time distances. The ratio $\frac{max(d,d_0)}{d_0}$ is close to zero for small distances and is maximized to 1 for time distances d_0 or larger. In the implementation of the system d_0 has been set equal to 120 seconds. In other words all time distances above 120 seconds are considered to be infinite.

Signature similarity

Signature similarity is simple in calculation. The aggregated alert has a signature, while the cluster has a signatures list. If the signature of the aggregated alert is included in the list, then similarity is equal to 1 , while if it is not similarity is equal to 0.

$$sim_{sig} = \begin{cases} 1, & \text{if } aga_{sig} \in clu_{sig} \\ 0, & \text{otherwise} \end{cases} \tag{7.10}$$

7.4 Weights updater

The diversity in protected networks, intrusion detection sensors' installations and intruders' activity makes using a static configuration for the three similarity weights ineffective. The weights used by the Clusterer should be adjusted according to the nature of the alert-set in examination. This is the task of the Weights updater component.

After each new alert, the component reads the aggregated alert-set and calculates three parameters representative of its nature. For performance reasons, the parameters can be updated after several new alerts (e.g. after every 5 or 10 alerts). These parameters are:

- Time density : Ratio of alerts number to time range
- Network density : Ratio of alerts number to size of the protected network
- Signatures variance : Variance of neighbouring alerts signatures

The calculation of the first two parameters is shown in Equations 7.11 and 7.12.

$$den_{time} = \frac{\text{aggregated alerts number}}{\text{time from the start time of the first aggregated alert}} \tag{7.11}$$

$$den_{net} = \frac{\text{aggregated alerts number}}{\text{max internal IP address - min internal IP address}} \tag{7.12}$$

The var_{sig} is similarly calculated to the corresponding parameter for the initial alert-set presented in subsection 6.2.1. It monitors changes of signature in adjacent aggregated alerts. The alert-set is split into chunks of tens of aggregated alerts and a local variance is calculated for each one of them. The global variance for the alert-set is the average of the local variances.

$$num_{decs} = (alertset_{size} \div 10) + 1 \tag{7.13}$$

$$var_n{}^{local} = \frac{\text{number of distinct signatures}}{10} \tag{7.14}$$

$$var_{sig} = \frac{\sum_{i=1}^{num_{decs}} var_i{}^{local}}{num_{decs}} \tag{7.15}$$

These three parameters characterize the incoming alert-set. In order to deduce the weight values from them a reference alert-set has been used. Specifically a default alert set with specific parameter values has been defined. These values are:

$$den^0_{time} = 0.3 \tag{7.16}$$

$$den^0_{net} = 4 \tag{7.17}$$

$$var^0_{sig} = 2 \tag{7.18}$$

The default aggregated alert-set has been defined to be as balanced as possible and the appropriate weights for it have been chosen to be equal:

$$w_{IP} = w_{time} = w_{sig} = 0.33 \tag{7.19}$$

The weights updater measures the deviation of the parameters of the real alert-set from the parameters of the default one and according to this deviation it adjusts the three weights. In order to avoid extreme circumstances, in which one weight could be over increased at the expense of others, the range of the weights has been set to [0.2,0.6]. Three factors $factor_{time}, factor_{net}, factor_{sig}$, each one of which is relevant to each one of the parameters describing the nature of the alert-set are calculated.

$$factor_{time} = \frac{den_{time}}{den^0_{time}} \tag{7.20}$$

$$factor_{net} = \frac{den_{net}}{den^0_{net}} \tag{7.21}$$

$$factor_{sig} = \frac{1}{\frac{var_{sig}}{var^0_{sig}}} \tag{7.22}$$

$$\tag{7.23}$$

These three factors are then used to calculate the three weights.

$$w_{time} = 0.2 + 0.4\frac{factor_{time}}{factor_{time} + factor_{net} + factor_{sig}} \tag{7.24}$$

$$w_{net} = 0.2 + 0.4\frac{factor_{net}}{factor_{time} + factor_{net} + factor_{sig}} \tag{7.25}$$

$$w_{sig} = 0.2 + 0.4\frac{factor_{sig}}{factor_{time} + factor_{net} + factor_{sig}} \tag{7.26}$$

7.4.1 Taking care of detached clusters

It is possible for alerts to not have an internal IP in none of their source or destination IP fields. It is common for intruders to have taken control of an internal host to attack another site, while using IP spoofing techniques to hide their intermediate location.

In such cases it is possible to have a cluster produced by these alerts with no internal IP addresses. A cluster like this will not be visualized at all by the Visualization subsystem. These clusters are characterized as detached clusters and the Clusterer should specifically examine them.

The Clusterer checks nearby clusters for each detached cluster, while giving priority to the ones overlapping with the detached cluster. It uses their internal IP addresses to solve the problem and populate the internal IPs list of the detached cluster. In this way there may be some ambiguity inserted in the final result, but all clusters, including detached ones, are visualized by the Visualization subsystem.

7.5 Clusters generator

The task of the Clusters generator is to produce data for cases in which an occurred event seems to not have a relevant cluster in the clusters' list. The Clusters generator runs periodically to discover such cases and to produce estimated information about missed incidents.

The high level logic of the component is to study every possible pair of clusters with a minimum time difference and conclude if there is a logical gap between them, that could be filled by an artificially generated cluster. The clusters between the pair are examined and if they do not seem to provide the required logical connection, an artificial cluster candidate is created. Its data is produced by the pair's data and it is subsequently validated and tuned, in order to enhance the outcome. During validation the component checks whether the cluster offers a minimum additional amount of information to the existing list of clusters, otherwise it is discarded. During tuning its parameters are tuned in order to maximize the quality of added information.

The Clusters generator operation consists of searching for artificial clusters candidates, generating them, validating them and finally tuning them. These procedures are analysed in the next subsections.

7.5.1 Searching through clusters

The component searches for pairs of real clusters with a minimum time distance and no significant activity monitored between them. In order to calculate the volume of activity between the pair of clusters a geometric approach has been incorporated.

All clusters are plotted on a three dimensional space. X-axis corresponds to IP values, y-axis to time values and z-axis to danger values. Each cluster is plotted as a rectangle parallel to x-z plane and perpendicular to the x-y plane. Its height in z-axis is relevant to its danger value. Its position in the y-axis is relevant to its time centre (the average between its start and end times). Its limits on the x-axis are relevant to its minimum and maximum internal IP addresses. An example of plotting a cluster is shown in Figure 7.3.

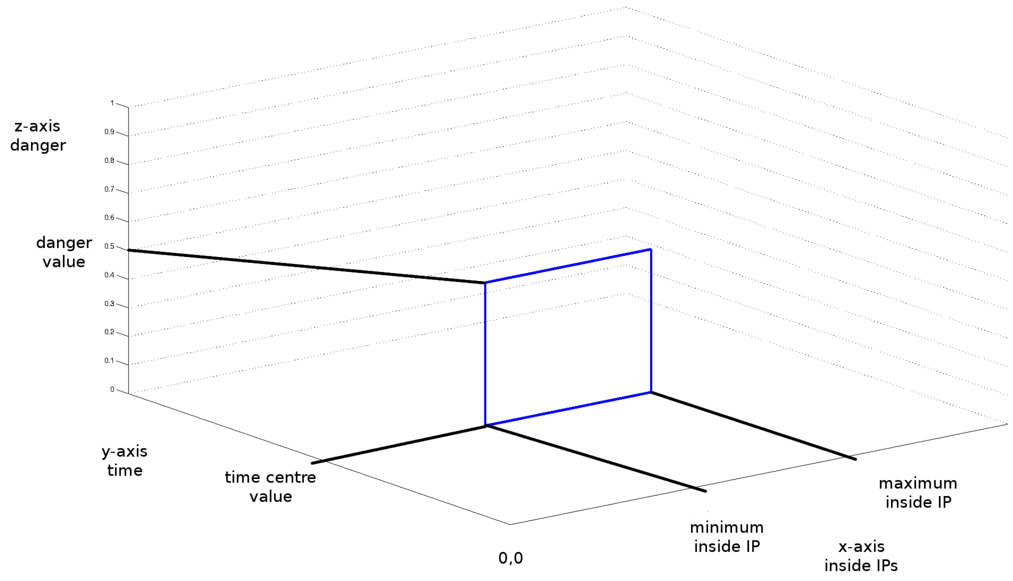

Figure 7.3: Plotting a cluster

All pairs of clusters are then examined as long as they abide by the following rules :

- The time difference of their time centres is at least 15 minutes. This is essential in order not to produce artificial clusters for pairs that simply are close in time and normally do not have any activity between them.

- Their internal IPs range is at least 10 IPs. This rule also reduces the number of candidates, in order not to flood the cluster-set with artificial data.

For all pairs that abide by these rules their intermediate clusters are examined one by one. The intermediate clusters, whose rectangles intersect with the volume between the pair's rectangle (a three dimensional trapezoid) are used to estimate the activity between the pair. For each one of them its projection to the second in time cluster of the pair is calculated, as shown in Figure 7.4. The pair of the clusters in examination are the blue and the orange rectangles, while the intermediate cluster is the green rectangle.

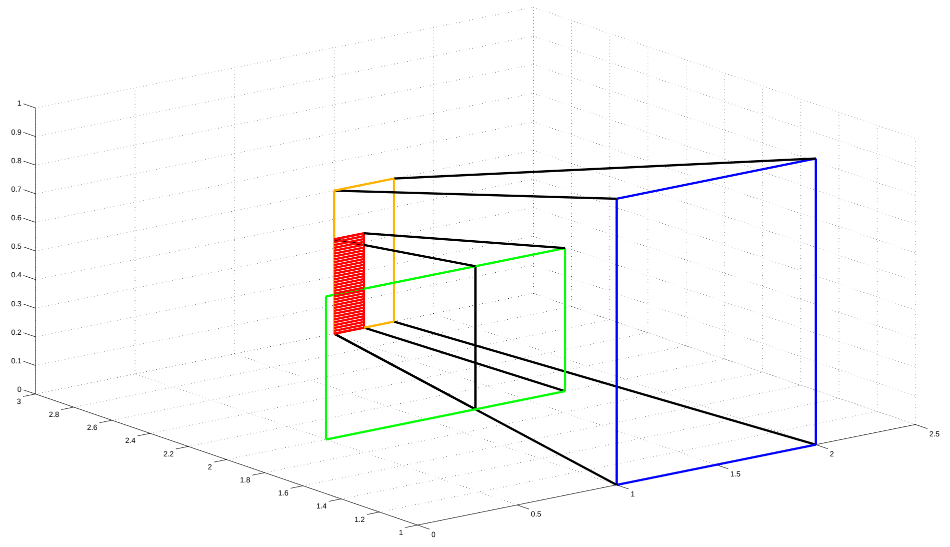

Figure 7.4: Examining an intermediate cluster

The main idea of searching for candidates is that a large projection means that there is important activity between the pair of clusters for the same IP space, while a smaller projection means that there is not enough activity between the two clusters and that the production of an artificial cluster may be suitable.

For intermediate clusters examined, their projections are produced. Then the union of these projections is created and its area is calculated as a percentage of the area of the second cluster of the pair. If this value exceeds a certain threshold then the activity between the pair is assumed as sufficient; otherwise an artificial cluster candidate is created.

Projecting clusters

Regarding the intermediate cluster the projection calculation aims to discover its relation of it to the clusters of the pair and if it can be considered as an intermediate step between them. The relevant factors are:

- The cluster should be occuring between the pair
- The cluster should refer to a similar IP range
- Its danger values should be relatively high or at least close to the corresponding values of the clusters of the pair

The parameters of the intersection of the rectangle of the intermediate cluster with the volume between the rectangles of the clusters of the pair are calculated. These are the minimum and maximum IP values along with the danger value. Then they are normalized with respect to the parameters of the intersection of the plane of the intermediate cluster with the volume between the rectangles of the clusters of the pair. After normalization the three parameters have values in [0,1] range and are identical to corresponding values of the projection rectangle with respect to the rectangle of the second cluster of the pair.

So if the pair of clusters is clu^{start}, clu^{end} and the intermediate cluster is clu^{inter} their parameters used in the calculation are shown in Table 7.1.

Table 7.1: Parameters of searching for artificial cluster candidates

Cluster	Minimum internal IP	Maximum internal IP	Time centre	Danger
clu^{start}	clu^{start}_{minii}	clu^{start}_{maxii}	$clu^{start}_{t} = \frac{clu^{start}_{sti}+clu^{start}_{eti}}{2}$	clu^{start}_{dan}
clu^{end}	clu^{end}_{minii}	clu^{end}_{maxii}	$clu^{end}_{t} = \frac{clu^{end}_{sti}+clu^{end}_{eti}}{2}$	clu^{end}_{dan}
clu^{inter}	clu^{inter}_{minii}	clu^{inter}_{maxii}	$clu^{inter}_{t} = \frac{clu^{inter}_{sti}+clu^{inter}_{eti}}{2}$	clu^{inter}_{dan}

The first parameters to be calculated are these of the rectangle created by the intersection between the volume between clu^{start} and clu^{end} and the plane of the clu^{inter}. The relevant calculations are shown in Eguations 7.27, 7.28 and 7.29.

$$intersection_{minii} = clu^{start}_{minii} + (clu^{end}_{minii} - clu^{start}_{minii}) * \frac{clu^{inter}_t - clu^{start}_t}{clu^{end}_t - clu^{start}_t} \tag{7.27}$$

$$intersection_{maxii} = clu^{start}_{maxii} + (clu^{end}_{maxii} - clu^{start}_{maxii}) * \frac{clu^{inter}_t - clu^{start}_t}{clu^{end}_t - clu^{start}_t} \tag{7.28}$$

$$intersection_{dan} = clu^{start}_{dan} + (clu^{end}_{dan} - clu^{start}_{dan}) * \frac{clu^{inter}_t - clu^{start}_t}{clu^{end}_t - clu^{start}_t} \tag{7.29}$$

Then the component checks if the previous rectangle and the clu^{inter} rectangle intersect. If they do, the parameters of their intersection rectangle which is called projection rectangle can be calculated as shown in Eguations 7.30, 7.31 and 7.32. If they do not intersect, no projection rectangle is calculated.

$$projection_{minii} = max(intersection_{minii}, clu^{inter}_{minii}) \tag{7.30}$$

$$projection_{maxii} = min(intersection_{maxii}, clu^{inter}_{maxii}) \tag{7.31}$$

$$projection_{dan} = min(intersection_{dan}, clu^{inter}_{dan}) \tag{7.32}$$

$$\tag{7.33}$$

Finally these parameters are normalized with respect to the intersection rectangle in order to have values in the [0,1]. Specifically:

$$projection^{norm}_{minii} = \frac{projection_{minii} - intersection_{minii}}{intersection_{maxii} - intersection_{minii}} \tag{7.34}$$

$$projection^{norm}_{maxii} = \frac{projection_{maxii} - intersection_{minii}}{intersection_{maxii} - intersection_{minii}} \tag{7.35}$$

$$projection^{norm}_{dan} = \frac{projection_{dan}}{intersection_{dan}} \tag{7.36}$$

The same projection procedure is carried out for all clusters. For each cluster that abides by the initial rules and its projection produces a projection rectangle, this rectangle is added to the projection rectangles list. After all clusters have been checked the rectangles included in the list are combined.

For every point in the [0,1] range of the IPs axis that corresponds to the $[clu^{end}_{minii}, clu^{end}_{maxii}]$ range the maximum danger value of all rectangles is used, in order to create a single union area for all projections. This combined area on the clu^{end} corresponds to the part of it that is covered by projection rectangles. If n projection rectangles have been produced the area of the total projection is calculated as in Equation 7.37.

$$area_{proj} = \int_0^1 max(projection^1_{dan}(x), ..., projection^n_{dan}(x))dx \tag{7.37}$$

In order to calculate the integral of Equation 7.37 a sampling approach has been used. The x variable is increased from 0 to 1 with step 0.01. In each loop the higher danger value between projections is added to a sum, which is finally divided by 100. This procedure is efficient in terms of performance terms and it produces a fine approximation of the area of total projection. The logic of this calculation is described in Algorithm 7.3.

```
1: $sum \leftarrow 0$
2: **for** $i = 0$ to 1 with step 0.01 **do**
3:   $dan_{max} \leftarrow 0$
4:   **for all** projections **do**
5:     **if** $(i \geq projection^{norm}_{minii}) \wedge (i \leq projection^{norm}_{maxii})$ **then**
6:       **if** $projection^{norm}_{dan} > dan_{max}$ **then**
7:         $dan_{max} \leftarrow projection^{norm}_{dan}$
8:       **end if**
9:     **end if**
10:   **end for**
11:   $sum \leftarrow sum + dan_{max}$
12: **end for**
13: $area \leftarrow \frac{sum}{100}$
```
Algorithm 7.3: Calculation of area integral

The area calculated in Algorithm 7.3 is in [0,1] and is indicative of the percentage of the clu^{end} that is covered by projections.

If this percentage is greater than a threshold value $area_{thre}$, then the intermediate activity between the pair is considered adequate and no new artificial cluster is created. If this percentage is less than $area_{thre}$ then it is assumed that a logical gap may exist between the pair of examined alerts and a candidate artificial cluster is produced as is described in the next subsection.

The initial value of $area_{thre}$ parameter is set equal to 0.5. This means that the calculated projection area should be less than half of the area of the rectangle of the second cluster, in order to create an artificial cluster. If no cases where an artificial alert should be created are found, the Clusters generator increases $area_{thre}$ by 0.1 to 0.6 to take into account more pairs of clusters. This increase in the threshold value is repeated until Clusters generator is able to produce some artificial clusters.

7.5.2 Creating artificial clusters

If for a pair of clusters the total calculated projection area is greater than $area_{thre}$ then an artificial candidate cluster is created. Its parameters are calculated by the corresponding parameters of the pair clusters. As it is shown in Table 5.1 the Generated cluster data type includes a subset of the parameters of Cluster data type, which are required for its visualization by the Visualization subsystem. Almost all parameters' values of the Generated cluster are calculated as the average of the two values of the corresponding parameters of the clusters of the pair.

$$clu^{art}_{minii} = \frac{clu^{start}_{minii} + clu^{end}_{minii}}{2} \tag{7.38}$$

$$clu^{art}_{maxii} = \frac{clu^{start}_{maxii} + clu^{end}_{maxii}}{2} \tag{7.39}$$

$$clu^{art}_{sti} = \frac{clu^{start}_{sti} + clu^{end}_{sti}}{2} \tag{7.40}$$

$$clu^{art}_{eti} = \frac{clu^{start}_{eti} + clu^{end}_{eti}}{2} \tag{7.41}$$

$$clu^{art}_{dan} = \frac{clu^{start}_{dan} + clu^{end}_{dan}}{2} \tag{7.42}$$

$$clu^{art}_{num} = \frac{clu^{start}_{num} + clu^{end}_{num}}{2} \tag{7.43}$$

$$clu^{art}_{truth} = \frac{clu^{start}_{truth} + clu^{end}_{truth}}{2} \tag{7.44}$$

$$ \tag{7.45}$$

The last parameter of the Generated cluster that has to be calculated is its class ids list. Because the artificial cluster is created in order to logically connect the pair of clusters, its class id values should ideally be between the class ids of the pair clusters. So the list of class id values in each of the pair clusters is scanned and class ids are added to the relevant list of the generated artificial cluster, according to the following logic rules:

$$\text{if } \exists\, cid\ (cid \in clu^{start}_{cid} \bigwedge cid \in clu^{end}_{cid}) \longrightarrow \text{ add } cid \text{ to } clu^{arti}_{cid} \tag{7.46}$$

$$\text{if } \exists\, cid\ (cid \in clu^{start}_{cid} \bigwedge cid+1 \in clu^{end}_{cid}) \longrightarrow \text{ add } cid, cid+1 \text{ to } clu^{arti}_{cid} \tag{7.47}$$

$$\text{if } \exists\, cid\ (cid \in clu^{start}_{cid} \bigwedge cid+2 \in clu^{end}_{cid}) \longrightarrow \text{ add } cid+1 \text{ to } clu^{arti}_{cid} \tag{7.48}$$

If a class id is present in both lists of pair clusters it is also added to the artificial cluster generated. If the two pair clusters contain two consecutive class ids then they are both added to the artificial cluster. If the two pair cluster contain class id values that differ by 2 then the intermediate class id is added to the artificial cluster.

7.5.3 Validating artificial clusters

In the validation sub process the clusters generated in the previous step are evaluated, and the ones carrying insignificant information are discarded. A metric of the quality of the information contained in each cluster is needed to form the basis of this reasoning. This metric is called Cluster's Quality CQ and it is defined to be representative of the value of information of each cluster. It is calculated as a function of three other variables namely:

Distance from the closest cluster (D): The minimum of the distances of the generated artificial cluster from the existing clusters, in the two-dimensional space defined by the time axis and the IP axis. The distance between two clusters A,B is defined as the Euclidean distance of their centres. The coordinates of their centres are

$$clu^A : (\frac{clu^A_{minii} + clu^A_{maxii}}{2}, \frac{clu^A_{sti} + clu^A_{eti}}{2})$$
$$clu^B : (\frac{clu^B_{minii} + clu^B_{maxii}}{2}, \frac{clu^B_{sti} + clu^B_{eti}}{2})$$

The distance between the clusters A,B is calculated as in Equation 7.49.

$$D_{A,B} = \sqrt{(x_{clu^A} - x_{clu^B})^2 + (y_{clu^A} - y_{clu^B})^2} \tag{7.49}$$

In order to calculate the CQ of a cluster, the minimum of its distances to other clusters D_{min} is calculated. Clusters that lie away from neighbouring clusters are more likely to be useful, in contrast to clusters that are close to neighbouring ones.

Area of the cluster (A): The area of the ellipse that corresponds to the cluster. This ellipse for clu^A is defined by Equation 7.54.

$$x_0 = \frac{clu^A_{minii} + clu^A_{maxii}}{2} \tag{7.50}$$

$$y_0 = \frac{clu^A_{sti} + clu^A_{eti}}{2} \tag{7.51}$$

$$x_r = \frac{clu^A_{maxii} - clu^A_{minii}}{2} \tag{7.52}$$

$$y_r = \frac{clu^A_{eti} - clu^A_{sti}}{2} \tag{7.53}$$

$$\frac{(x - x_0)^2}{x_r} + \frac{(y - y_0)^2}{y_r} = 1 \tag{7.54}$$

The larger an ellipse is, the more general the corresponding cluster is and vice-versa. A large area ellipse means either that the cluster spans across multiple IPs, or it has a large time range or both.

Too general clusters cannot be much helpful for the user, whereas more specific clusters are more likely to contain valuable information. The area of the ellipse is calculated from time and IP values of the cluster as shown in Equation 7.55:

$$A = \frac{(clu^A_{eti} - clu^A_{sti})(clu^A_{maxii} - clu^A_{minii}) * \pi}{4} \tag{7.55}$$

Quality of cid values list (CIDQ): It has been mentioned in previous sections that each cluster has a class ids list that consists of the different attack class values related to it. These values represent the step of the attack plan, in which the specific cluster belongs. In Snort the class ids are numbered from 1 to 7. The variance of cid values indicates whether the cluster spans along multiple steps of the attack. Alerts with different cid values belong to different steps of the attack plan, and their coexistence in the same cluster can potentially reveal the logical connection among the steps of the attack.

The quality $CIDQ$ for a list with only one value is defined to be equal to 1, whereas for a list with k values it is calculated as shown in Equation 7.56.

$$CIDQ = k * \frac{\sum_{n=1}^{k-1} \frac{1}{(a_n - a_{n-1})^2}}{k-1} \tag{7.56}$$

A compact list (a list with continuous values) seems to be more consistent and should have a higher CIDQ value. On the other hand a list, the members of which are not continuous or/and have large distances between them, is not likely to represent a meaningful series of the attack's steps and should have a lower $CIDQ$ value. In table 7.2 the CIDQ value is calculated for some sample cid lists.

Table 7.2: Calculating CIDQ values for sample cid lists

cid list	CIDQ
3	1
3,4	2
3,4,5	3
3,4,6	1.875
3,4,7	1.65
3,7	0.125

Generally the Quality of the cluster CQ is proportional to the distance from the closest cluster D_{min} and the quality of the *cid* values $CIDQ$, while it is inversely proportional to the Area of the cluster A. CQ calculation is shown in Equation 7.57.

$$CQ = \frac{D_{min} * CIDQ}{A} \tag{7.57}$$

The calculated CQ values are used to sort generated clusters in terms of the quality of information they carry. The validation process ensures that the most useful of the generated clusters are injected in the real clusters list, without over populating it. The generated clusters used are at most as many as 20% of the real clusters. If the generated clusters are less than this threshold, they are all incorporated, otherwise the ones with the highest CQ values are selected until the threshold is reached.

7.5.4 Tuning artificial clusters

In the last stage of the Cluster generation process, the clusters that have been produced in the first stage and have not been discarded in the second are fine tuned in order to enhance the contribution of the component to the overall system.

The idea of tuning the cluster is to alter the A and CIDQ parameters (D_{min} cannot be usefully altered) in such a way that the cluster becomes more informative, while it keeps its main initial characteristics.

Area tuning

In general smaller clusters (reducing A) seem more informative about the actual intrusion activity. The value of A is calculated as a function of time and the IP boundaries of the cluster. The IP boundaries are decided upon the IP boundaries of the real clusters, that are logically connected by the generated ones; hence they should not be altered.

On the other hand, the time span of the cluster could be reduced around its initial center, as long as this reduction does not completely alter the geometry of the cluster. Specifically this reduction should not be more than 20% while the ratio of the IP range of the cluster over the time range of the cluster does not become less than $\frac{1}{10}$. The simple logic used is described in Algorithm 7.4.

```
1: $tr_{init} \leftarrow clu_{eti} - clu_{sti}$
2: $ipr \leftarrow clu_{maxii} - clu_{minii}$
3: $tr \leftarrow clu_{eti} - clu_{sti}$
4: **while** $(tr \geq 0,8 * tr_{init}) \wedge (\frac{tr}{ipr} \geq \frac{1}{10})$ **do**
5:     $clu_{sti} \leftarrow clu_{sti} + 1$
6:     $clu_{eti} \leftarrow clu_{eti} - 1$
7:     $tr \leftarrow clu_{eti} - clu_{sti}$
8: **end while**
```

Algorithm 7.4: Area tuning

CIDG tuning

Tuning $CIDQ$ is done in a similar way; it can be altered in order to increase the quality of information as long as the initial character of the cluster is not lost. One change is permitted in the values of the cid list (one value can be replaced by another) as long as the produced increase in the CIDQ value of the cluster is no more than 50% of the initial value. The small space of values for *cid* list, enables exhaustive search to be used as the method to find the optimal *cid* values for the cluster. The logic is depicted in Algorithm 7.5.

```
1: $CIDQ_{init} \leftarrow CIDQ$value of the cluster
2: $CIDQ_{max} \leftarrow CIDQ_{init}$
3: **for all** cid list values **do**
4:    **for all** possible alternate cid values **do**
5:       replace cid value with the alternate one
6:       calculate $CIDQ_{run}$
7:       **if** $(CIDQ_{init} < CIDQ + run < 1.5 * CIDQ_{init}) \wedge (CIDQ_{run} > CIDQ_{max})$ **then**
8:          $CIDQ_{max} \leftarrow CIDQ$
9:          keep in record the replace of values
10:      **end if**
11:   **end for**
12: **end for**
13: **if** no replace of values kept in record **then**
14:    cluster left intact
15: **else**
16:    commit last replace in record
17: **end if**
```

Algorithm 7.5: ACLQ tuning

CHAPTER 8

Visualization subsystem

The visualization subsystem creates a graphical representation of the security state of the protected system. It reads clusters produced by the Clustering subsystem, commits the required transformations to their parameters and combines them in a unique representation. It produces a surface in three dimensions, which is characterized by peaks for each of the clusters in the list. In this way suspicious activity on the system is depicted in terms of IPs of the protected network and time. The resulting graph also presents information about danger and validity estimates for each cluster.

8.1 Design of the Visualization subsystem

The Visualization subsystem is responsible for reading the current clusters list from the database and producing a graph containing a high level representation of the security state of the protected system. This procedure is recurring as it produces a still image of the security state at a specific moment. It has to be continuously repeated, to produce successive frames that are displayed as an evolving animation.

The visualization concept is to produce a surface in a three dimensional space that will summarize the security state of the protected system. The three dimensions of the graph are related to the internal IPs space of the protected network (x axis), time (y axis) and an estimated danger value (z axis). For a system without any intrusive events the produced surface is flat and it matches the x-y plane. For each detected cluster of alerts a peak is created on this level surface.

The height of the peak is related to the danger value of the cluster. The area of the peaks on the x-y plane is related to the combination of the range of internal IPs that are included in the cluster and its time range. The color of the surface at the peak area is related to the validity score of the cluster.

The platform chosen to implement the Visualization subsystem was Matlab 2010a [63], as its ready-to-use visualization functions made it easy to depict the surface that combines all existing clusters.

In Figure 8.1 an example frame, containing peaks corresponding to two clusters is given. The data for the two clusters are shown in Table 8.1.

Table 8.1: Example clusters data

Cluster	IPs space	Time range	Danger value	Truth value
A	76.213.183.186 - 226	17.58.22 - 18.07.39	0.80	0.43
B	76.213.183.175 - 196	18.06.24 - 18.07.06	0.20	0.84

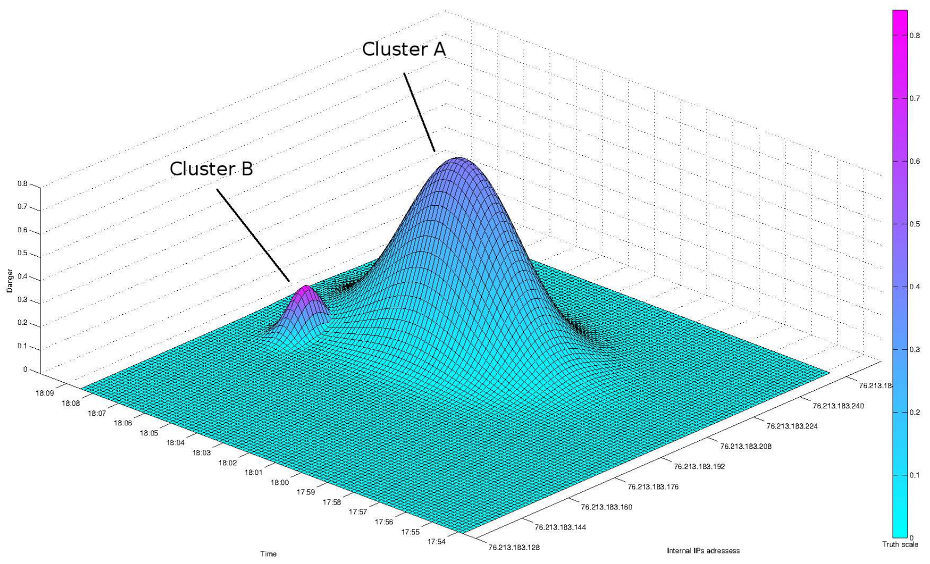

Figure 8.1: An example frame

Cluster A contains activity lasting for almost 10 minutes and affecting a range of 40 IPs of the internal network. On the other hand cluster B lasts for less than a minute and it concerns 21 internal IPs. The internal IPs and time ranges of each cluster define a base area on the x-y plane on which the corresponding peak is situated. As cluster A lasts more and concerns more IPs than cluster B, its base area is significantly larger and in general its peak on the surface is more voluminous. The height of the peaks is defined by the danger value calculated by the Clustering subsystem for each one of the clusters. Cluster A with danger value equal to 0.8 produces a much higher peak than the peak produced by cluster B with danger value 0.2. Finally, the color of the surface at the area of each peak is defined by the validity score of each cluster. Cluster B is characterized by a higher validity score, so its peak is almost red while the peak of cluster A is blue. The colour-map used is shown in the bar at the right side of the frame.

Another important visualization feature, that is obvious from Figure 8.1 is that a single surface is produced for the system, so a single event can be present in the graph for each grid point (IP and time combination). The values in table 8.1 show that both clusters A and B are related to IPs between 76.213.183.186 and 76.213.183.226 and are active in time range from 18.06.24 to 18.07.39. The two peaks overlap in a certain area of the surface. During surface production the cluster with the highest danger value is preferred for each grid point, while the others are ignored. This happens because it is essential to produce a simple informative graph and not a complex representation of each event that happens. The security analyst will be informed of intrusive activity for a certain IP range at a certain time by the graph and she may examine in detail relative alert-sets to discover more details about it.

The subsystem consists of three components that are depicted in Figure 8.2. These components are DB reader, Clusters analyser and Plotter.

The DB reader is responsible for reading clusters' data from the database of the system and committing the required normalizations in IP and time values, in order for the resulting graph to be as easy to read as possible.

The produced normalized clusters are then sent to the main component of the Visualization subsystem, which is the Clusters analyser. Its task is to iterate through all clusters and produce two arrays that contain height and color values for each point on the surface. No matter how many clusters may exist, a single surface is generated that approximates the security state of the protected system.

These arrays along with other auxiliary data are the input to the last component of this subsystem, the Plotter. The Plotter produces the final frame by depicting the surface and taking care of other important graph parameters, such as labelling axis and efficiently

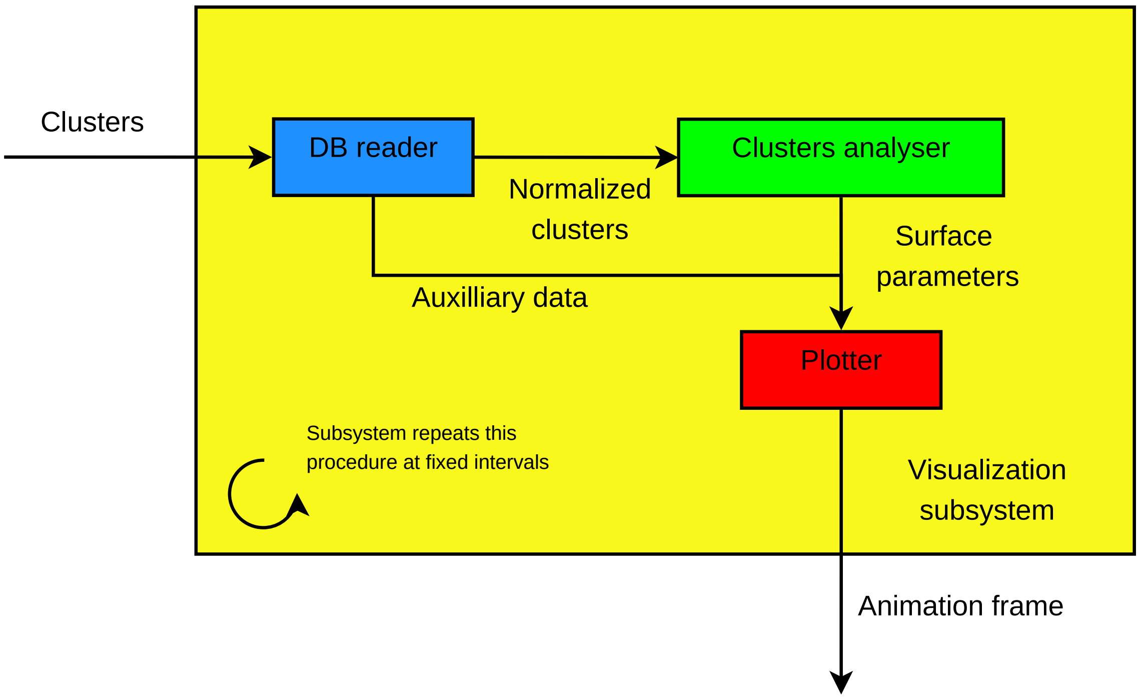

Figure 8.2: Visualization subsystem

placing the camera in the three-dimensional space.

8.2 DB reader

The DB reader reads clusters from the system database and normalizes their parameters, to make a meaningful representation feasible. As it has been mentioned in previous chapters the clusters' data stored in the database are in the format shown in Table 8.2.

Table 8.2: Format of clusters data in the database

Field	Description	Format
min IP	The first internal IP related to the cluster	IP in decimal format
max IP	The last internal IP related to the cluster	IP in decimal format
min time	The start time-stamp of the cluster	Date in Linux format
max time	The end time-stamp of the cluster	Date in Linux format
num of alerts	Number of initial alerts included in the cluster	Integer value
danger	Danger value calculated for the cluster	Double value
truth	Truth value calculated for the cluster	Double value

The required transformations concern IP and time data of the clusters and enable meaningful annotation of the x and y axes. The produced graph concerns all internal IP values of the protected network, even if not all of them are intrusion targets. On the other hand, the time range of the graph is decided from the clusters themselves. The graph concerns only the time range in which the detected clusters exist.

The global minimum and maximum IP values are predefined for the protected network and correspond to the smallest and the largest of its internal IPs. The global min IP value is subtracted by all IP values of the clusters to create a representation in the [0, globalmaxIP - globalminIP] space.

The time values are in Linux format. Assuming that the live representation being produced is not for periods larger for 24 hours, the date part of the time values is discarded. The values are converted from milliseconds to seconds and the necessary shift for synchronizing with the valid time zone is performed. The global minimum and maximum time values are calculated and the minimum time value is subtracted from all time values to create a representation in the [0, globmaxtime-globmintime] space.

The algorithm of the DB reader component is shown below :

```
1: Read $IP_{min}^{glob}, IP_{max}^{glob}$
2: Read tzd (Time zone's hours difference)
3: **for all** clusters **do**
4:    $IP_{min} \leftarrow IP_{min} - IP_{min}^{glob}$
5:    $IP_{max} \leftarrow IP_{max} - IP_{min}^{glob}$
6: **end for**
7: **for all** clusters **do**
8:    $time_{min} \leftarrow \frac{time_{min}}{1000}$
9:    $time_{max} \leftarrow \frac{time_{max}}{1000}$
10: **end for**
11: Calculate $time_{min}^{glob}, time_{max}^{glob}$
12: **for all** clusters **do**
13:    $time_{min} \leftarrow time_{min} - time_{min}^{glob} + tzd * 60 * 60$
14:    $time_{min} \leftarrow time_{min} - time_{min}^{glob} + tzd * 60 * 60$
15: **end for**
```

Algorithm 8.1: DB reader algorithm

8.3 Clusters analyser

8.3.1 Choosing the appropriate models

The most important component of the Visualization subsystem is the Clusters analyser as it transforms multiple clusters into a unique surface, representative of all of them. The input to this component is the normalized clusters' set produced by the DB reader and its output are the arrays holding the parameters of the surface, which are sent to the Plotter component for the production of the frames.

In order to represent each cluster a peak is formed on the surface, as it has been depicted in Figure 8.1. The rationale is to form a three-dimensional shape that will be indicative of as many cluster's parameters as possible.

Gaussian distribution has been chosen for creating the shape of the peak. The bell shape produced by the Gaussian distribution is depicted in Figure 8.3 and is appropriate for representing the clusters.

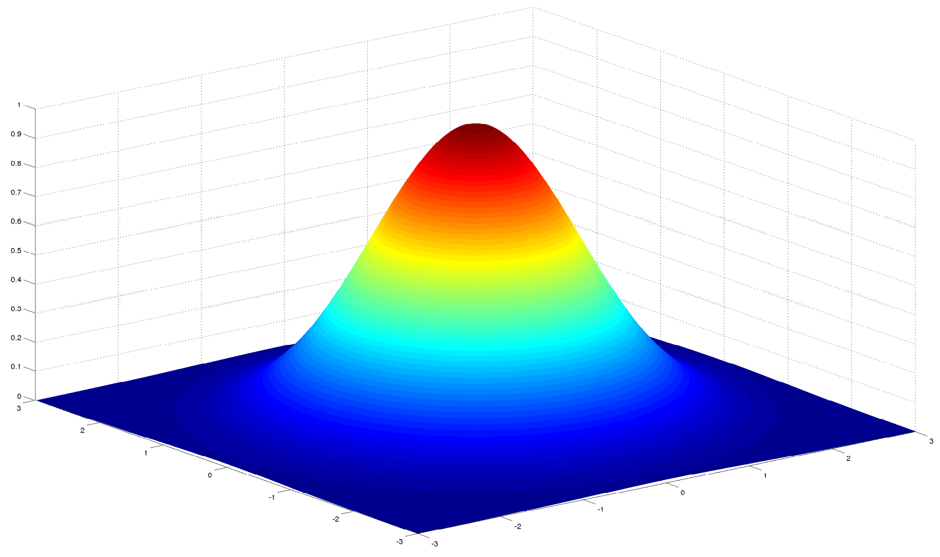

Figure 8.3: Graph of Gaussian distribution

In order to produce narrower peaks the square of the Gaussian distribution was preferred. The resulting peak is depicted in Figure 8.4. Assuming that the area of the x-y plane that is relevant to the cluster is the square defined by the x [-1, 1] range and the y [-1, 1] range the bell produced by the square of the Gaussian distribution seems more accurate than the one produced by the distribution itself.

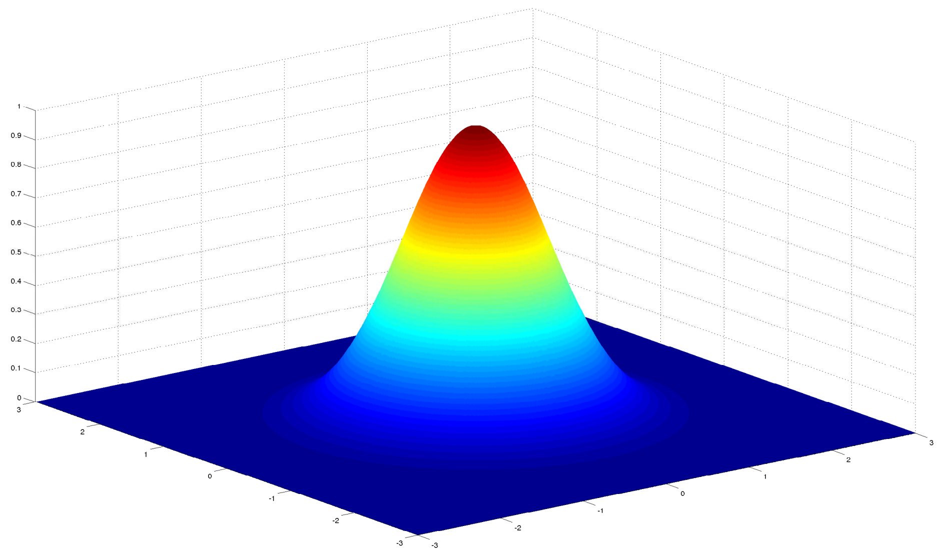

Figure 8.4: Graph of the square of Gaussian distribution

The function of the square of Gaussian distribution is :

$$z = \left(e^{-(x^2+y^2)}\right)^2 \tag{8.1}$$

8.3.2 Producing a peak for each cluster

The produced bell shape should approximate the cluster as accurately as possible. The bell is placed on the area of x-y plane that is defined by the time and IP values of the cluster. The formation of a rectangle on the x-y plane by time and IP ranges is shown in Figure 8.5.

The x_c and y_c coordinates of the centre of the rectangle are easily calculated, as shown in Equation 8.2, while the z_c value is zero as the center point is on the x-y plane. The distances of the rectangle's sides from the centre are shown in Equation 8.3. The peak is situated around point (x_c, y_c, z_c).

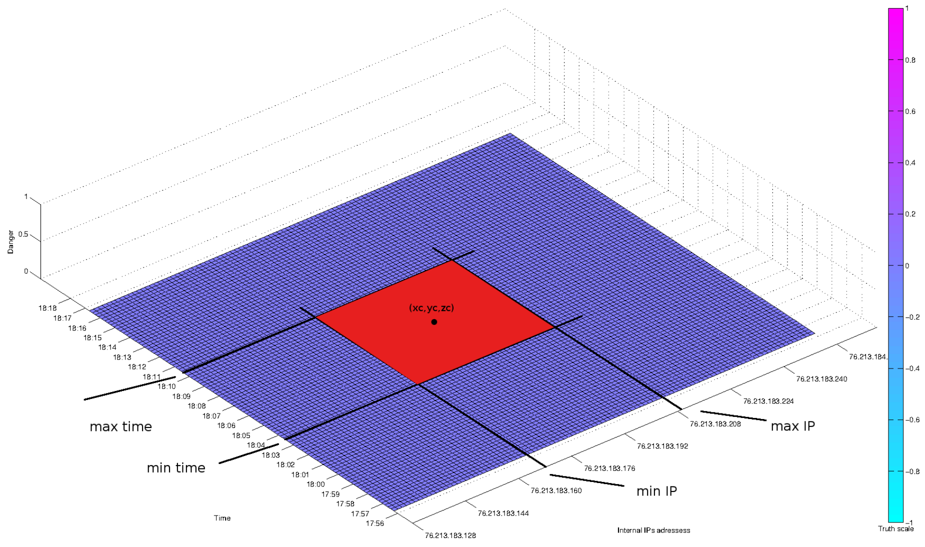

Figure 8.5: Time and IP ranges define the base area of the bell

$$
\begin{aligned}
x_c &= \frac{IP_{min} + IP_{max}}{2} \\
y_c &= \frac{time_{min} + time_{max}}{2} \\
z_c &= 0
\end{aligned}
\tag{8.2}
$$

$$
\begin{aligned}
x_r &= \frac{IP_{max} - IP_{min}}{2} \\
y_r &= \frac{time_{max} - time_{min}}{2}
\end{aligned}
\tag{8.3}
$$

In order to produce a bell that peaks at value 1 at the centre of the rectangle and fades out to zero towards its sides, the x and y variables in Equation 8.1 have been replaced by $\frac{(x-x_c)^2}{{x_r}^2}$ and $\frac{(y-y_c)^2}{{y_r}^2}$ respectively.

The next parameter of the cluster that needs to be calculated is its density, which shows how dense or sparse is the distribution of alerts in the clusters time range. The number of alerts is divided by the duration of the cluster.

$$density = \frac{num_{alerts}}{time_{max} - time_{min}} \tag{8.4}$$

The density calculated in Equation 8.4 ranges from 0 to infinity and is not appropriate for visualization purposes. In order to normalize it, the sigmoid function shown in Equation 8.5 is used.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8.5}$$

Because this function for x values in range $(0, \infty)$ outputs values in range $(0.5, 1)$, it has been slightly transformed by subtracting 0,5 from it to force zero density to produce zero normalized density and the result has been multiplied by 2, in order to make the normalized density approximate 1 for large values of density.The graphs of the sigmoid function and the transformation used are shown in Figure 8.6.

$$density_{norm} = (\frac{1}{1 + e^{-density}} - 0,5) * 2 \tag{8.6}$$

The $density_{norm}$ value is used to emphasize the danger of the cluster. Normally the height of the peak is decided by its danger value, which also ranges between 0 and 1. Because the density of alerts is also an indication of cluster's danger, if $density_{norm}$ is larger than the danger of the cluster, then it is used to define the height of the peaks, instead of the danger value.

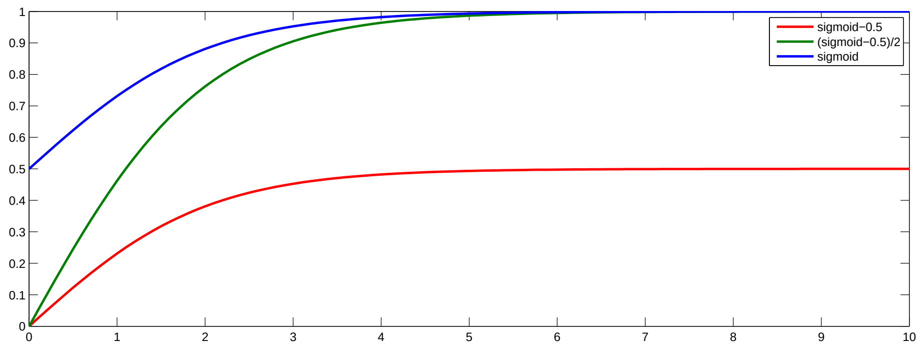

Figure 8.6: Sigmoid function and its transformation

$$cluster_{height} = \begin{cases} danger, & \text{if } danger \geq density_{norm}. \\ density_{norm}, & \text{otherwise.} \end{cases} \tag{8.7}$$

So Equation 8.8 is used to compute the height of peaks in (x,y) grid point. In this way a two-dimensional array is created which contains the height of the surface in each grid point on the x-y plane for the specific cluster.

$$height(x,y) = cluster_{height} * (e^{-(\frac{(x-x_c)^2}{x_r{}^2}+\frac{(y-y_c)^2}{y_r{}^2})})^2 \tag{8.8}$$

The last parameter that needs to be calculated for the specific cluster is the color of the peak. The color is relevant to the truth value of the cluster. The centre of the peak is coloured with the color that corresponds to the cluster's truth value, according to the colormap in Figure 8.7.

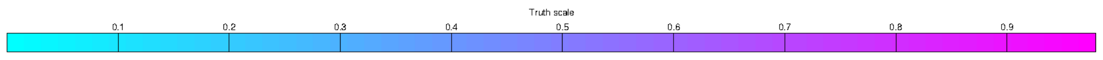

Figure 8.7: Truth value scale colormap

The color of the peak is gradually changing from the cntre of the cluster to the outer points of its area, to correspond to zero truth value color in the borders of the area. In this way the color differences in the final surface are much more visible and the truth parameter is distinguishable for each cluster. This is obvious in Figure 8.1.

In order to achieve this gradual transition of color along the surface of the peak the truth value has been combined with the calculated height value. So the color that corresponds to a point x,y, according to the truth value of a cluster is shown in Equation 8.9.

$$color(x,y) = height(x,y) * cluster_{truth} \tag{8.9}$$

For each one of the clusters received by the DB reader the Clusters analyser produces a two dimensional array that contains its height values and another one that contains its color values at each point of the grid.

8.3.3 Combining multiple peaks to one surface

The main idea is to produce a simple representation that will be easily inspected by the security analyst. A large number of clusters obviously produces a large number of peaks to depict on the surface. In order to keep the graph readable, the color and the height values for each point of the surface are decided by the cluster that has the largest height value at the specific point. This means that only the most dangerous cluster is depicted at each grid point.

The trade-off for keeping the surface simple is small. Overlapping clusters are going to be visible as two overlapping peaks. The only case in which a cluster will not be visible at all is to be included in the base area of another cluster with marginally larger height value. In this case the security analyst will not see the specific peak, but will be informed that there is intrusive activity for the specific IP and time values combination. The peak of the cluster with higher danger value, will motivate the analyst to further explore the alerts related to this combination of time and IP values.

The surface that will be finally visualised by the Plotter is defined by two two-dimensional arrays relevant to height and color, similar to the ones already produced for each cluster. To produce the two arrays the analyser initiates their values to 0. Then it iterates through clusters and for each one of them it scans all grid points. At each point, if the height array of the cluster has higher value than the height array of the surface, the array values of both surfaces for this grid point are set equal to the corresponding values of the arrays of the cluster.

After this procedure the values that are stored in the arrays of the surface, for each grid point, correspond to the values of the cluster with the higher $cluster_{height}$ value. The two arrays are then propagated to the Plotter.

8.3.4 Clusters analyser algorithm

The algorithm of the Clusters analyser component is given below in Algorithm 8.2.

```
1: $x_r^{min} = \frac{IP_{max}^{glob} - IP_{min}^{glob}}{20}$
2: $y_r^{min} = \frac{time_{max}^{glob} - time_{min}^{glob}}{20}$
3: for $i = 1 \rightarrow 100$ do
4:     $X[i] \leftarrow \frac{IP_{max}^{glob} - IP_{min}^{glob}}{100} * i$ {Create x and y values of the grid points}
5:     $Y[i] \leftarrow time_{min}^{glob} + \frac{time_{max}^{glob} - time_{min}^{glob}}{100} * i$
6:     $Z[i] \leftarrow 0$ {Array to store height values}
7:     $C[i] \leftarrow 0$ {Array to store color values}
8: end for
9: for all clusters do
10:    $x_c \leftarrow \frac{IP_{min} + IP_{max}}{2}$
11:    $y_c \leftarrow \frac{time_{min} + time_{max}}{2}$
12:    $z_c \leftarrow 0$
13:    $x_r \leftarrow \frac{IP_{max} - IP_{min}}{2}$
14:    $y_r \leftarrow \frac{time_{max} - time_{min}}{2}$
15:    $density \leftarrow \frac{num_{alerts}}{time_{max} - time_{min}}$
16:    $density_{norm} \leftarrow (\frac{1}{1+e^{-density}} - 0,5) * 2$
17:    $height \leftarrow max(cluster_{danger}, density_{norm})$
18:    for $x = X[1] \rightarrow x[100]$ do
19:       for $y = Y[1] \rightarrow Y[100]$ do
20:          $Z_c(x,y) \leftarrow cluster_{height} * (e^{-(\frac{(x-x_c)^2}{x_r{}^2} + \frac{(y-y_c)^2}{y_r{}^2})})^2$
21:          $C_c(x,y) \leftarrow cluster_{truth} * Z(x,y)$
22:          if $Z_c(x,y) > Z(x,y)$ then
23:             $Z(x,y) \leftarrow Z_c(x,y)$
24:             $C(x,y) \leftarrow C_c(x,y)$
25:          end if
26:       end for
27:    end for
28: end for
```

Algorithm 8.2: Clusters analyser algorithm

8.4 Plotter

The last component of the Visualization subsystem is the Plotter. It gets as input the color and height arrays of the surface and it uses the Matlab surf() function to create the frame. I also adjusts the axis labelling, as the global time and IP ranges of the cluster-set, should be taken into account to produce an informative graph.

The production of the graphical representation of the surface is performed by the surf(X,Y,Z,C) function of Matlab. X and Y are the arrays produced by the creation of the grid in the x-y plane in relevance to IP and time values. Z is the two dimensional array containing the height of the surface at each point and C is the two-dimensional array that contains the color values of the surface.

The labelling of the axis is based on two principles :

- If the graph is to be viewed in a pc monitor then fifteen labelling points along the time and IP axis is an efficient choice.
- The label points of the time axis should be placed in specific intervals such as 5 mins, 10 mins, 20 mins, 30 mins or 60 mins.

The Plotter starts by setting the IP step equal to 2 and doubles it, until the number of the sampling points become less than 15. For the time step it iterates through the predefined step values described above. It calculates for each one of them the difference between the number of produced sampling points and 15. It finally chooses the one with the least difference. The Plotter calculates sampling points on the time axis and converts their values from integers to time format. It also calculates the sampling points on the IP axis and converts their values from long values to IP format. The produced labels are attached to the graph axis.

The algorithm of the Plotter is shown in Algorithm 8.3.

```
1: $IP_{range} \leftarrow IP^{glob}_{max} - IP^{glob}_{min} + 1$
2: $IP_{step} \leftarrow 2$
3: **while** $\frac{IP_{range}}{IP_{step}} > 15$ **do**
4:     $IP_{step} \leftarrow IP_{step} * 2$
5: **end while**
6: **for** $i = 1 \rightarrow IP_{range} \div IP_{step}$ **do**
7:     $IP_{label}(i) \leftarrow ipconvert(i * IP_{step})$
8: **end for**
9: $time_{step} \leftarrow 1$
10: $time_{range} \leftarrow time^{glob}_{max} - time^{glob}_{min}$
11: $points_{num} \leftarrow timerange \div 60$
12: $delta \leftarrow |points_{num} - 15|$
13: $time^{choices}_{step}[1] \leftarrow 5$
14: $time^{choices}_{step}[2] \leftarrow 10$
15: $time^{choices}_{step}[3] \leftarrow 20$
16: $time^{choices}_{step}[4] \leftarrow 30$
17: $time^{choices}_{step}[5] \leftarrow 60$
18: **for** $i = 1 \rightarrow 5$ **do**
19:     $points_{num} \leftarrow timerange \div time^{choices}_{step}[i] * 60$
20:     $delta_{cur} \leftarrow |points_{num} - 15|$
21:     **if** $delta_{cur} < delta$ **then**
22:         $delta \leftarrow delta_{cur}$
23:         $time_{step} \leftarrow time^{choices}_{step}[i]$
24:     **end if**
25: **end for**
26: **for** $i = 1 \rightarrow time_{range} \div time_{step}$ **do**
27:     $time_{label}(i) \leftarrow timeconvert(i * time_{step})$
28: **end for**
```

Algorithm 8.3: Algorithm of the Plotter component

CHAPTER 9

Experiments and results

In this Chapter the experimental procedure, used to test the performance of the system, is presented, along with the relative results. Initially the two traffic datasets used to produce the alert-sets required for testing the system are discussed. The performance of the system regarding reducing the volume of alerts is documented by comparing the volume of sensors' alert-sets to the respective volumes of aggregated alerts and clusters produced by them. ROC analysis is used to show the ability of the system to efficiently classify alerts as true or false. Subsequently the results of the Visualization subsystem are presented and analysed for each one of the data-sets. The ability of the system to hypothesize on missed events is tested by using subsets of a specific alert-set. Finally the data-processing performance of the system is measured and discussed.

9.1 Set-up of experiments

In order to test the performance of the system the widely used network intrusion detection system Snort [61] has been used against two different datasets.

The first dataset is the Darpa 2000 1.0 [60]. The Cyber Systems and Technology Group of MIT Lincoln Laboratory, has collected and distributed standard corpora for evaluation of computer network intrusion detection systems. One of these is the dataset used for testing the performance of the proposed system. This dataset has been extensivelly used in the past and the results produced on it are comparable to those of other methods proposed in the literature. Apart from that, the corresponding documentation contains the intrusion steps performed throughout the simulation process, so it facilitates the evaluation of the results.

The Darpa dataset has been heavily criticised [64], [65], [66] and is certainly out of date as it does not contain modern attacks. However in view of the absence of an alternative widely accepted benchmark it is still being almost exclusively used for testing intrusion detection systems. Because of the above an alternative dataset has also been used. It has been created from live traffic captured from an academic network, combined with a specific attack plan (built up using modern tools) that have been conducted during the time of traffic capture. Although this data-set cannot be used for exhaustive testing of the system (due to incomplete knowledge of the possible attacks launched in the time of traffic capture), it gives a good indication of the performance of the system against modern attacks.

9.1.1 DARPA 2000 1.0 dataset

The specific dataset includes a distributed denial of service attack. This attack scenario is carried out over five different phases. The attacker probes the network, breaks in to some hosts by exploiting the Solaris sadmind vulnerability and installs trojan mstream DDoS software. She finally launches a distributed denial of service (DDoS) attack at an off site server from the compromised hosts. The steps of the attack are analysed in the MIT Lincon Laboratory website [60] and are described as follows.

1. The adversary performs a scripted IP sweep of multiple class C subnets on the Air Force Base. The networks swept are 172.16.115.0/24, 172.16.114.0/24, 172.16.113.0/24 and 172.16.112.0/24.

2. The hosts discovered in the previous phase are probed, to determine which hosts are running the "sadmind" remote administration tool.

3. The attacker then tries to break into the hosts found to be running the sadmind service in the previous phase. The attack script attempts the sadmind Remote-to-Root exploit several times against each host, each time with different parameters.

4. Entering this phase, the attack script has built a list of those hosts on which it has successfully created a new user. These are 172.16.115.20, 172.16.112.50 and 172.16.112.10. For each host on this list, the required scripts for DDos are installed.

5. In the final phase, the attacker manually launches the DDos from the infected hosts against the victim.

The DARPA 2000 1.0 dataset contains network traffic data from two sensors (inside and DMZ). These streams of data have been both used as input to a Snort 2.8.5.3 installation and two corresponding alert sets have been produced. These alert-sets have been used as input to the proposed system. The produced alert-set has been labelled (alerts have been recognized as true or false) by using the documentation of the DARPA dataset, which contains info about real intrusion activity. The results of this labelling are summarized in Table 9.1.

Sensor	True alerts	False alerts
Inside	764 (76%)	237 (24%)
DMZ	430 (15%)	2215 (84%)

Table 9.1: Darpa alert-set statistics

9.1.2 Live academic dataset

The academic network used as test-bed for this experiment consists of 25 servers hosting various services and around 100 desktops used by members of the academic community. For the purposes of the attack simulation, an image of a vulnerable host the Metasploitable VM [67] and an image of a host, hosting common LAMP stack, Ultimate-Lamp [68] have been used. Two virtual machines have been initiated by these images and added to the existing network structure. The simulation lasted for approximately one hour. During this time range a specific attack scenario was executed, while live network traffic was monitored by two Snort sensors.

The executed attack scenario consists of the following steps :

1. An external host scans all hosts in network's range.
2. Alive hosts found are scanned for known vulnerabilities.
3. The Metasploitable VM is selected as a target.
4. The Tomcat 5.5 vulnerability is chosen to be exploited and shell access is gained to the specific Host.

5. XerXes DoS attack is successfully executed from Metasploit VM against UltraLamp VM.

Two Snort sensors were used to produce alerts. One was located at the main switch of the DMZ network, while the other was located outside the firewall of the network.

Real world conditions have been ensured, as live traffic of an academic network has been used as the platform, on which the attack plan has been executed. On the other hand, this choice has created a lot of ambiguity in the experimental procedure, as the background traffic can not be characterized with certainty as legitimate use or intrusion.

The academic network consists of a /25 network. The addresses depicted in the results have been obfuscated for privacy reasons. The hosts and the attack plan used are described in Tables 9.2 and 9.3, respectively.

Host	Shortname	IP
External intruder	EI	-
Metasploit VM	MVM	76.213.183.136
UltraLamp VM	UVM	76.213.183.231

Table 9.2: Hosts used in the academic dataset attack plan

for

No	Time	Source	Destination	Incident
1	17:58-59	EI	/25 network	Scanning for alive hosts
2	18:01	EI	Alive hosts	Scanning for vulnerabilities
3	18:03	EI	MVM	Tomcat 5.5 vulnerability (unsuccessful)
4	18:12	EI	MVM	Tomcat 5.5 vulnerability (successful)
5	18:13	MVM	UVM	XerXes DoS

Table 9.3: Description of academic dataset attack plan

The alerts produced by the Snort sensors have been examined and manually labelled as true or false according to the attack plan mentioned in Table 9.3. The results of this labelling are summarized in Table 9.4.

Sensor	True alerts	False alerts
DMZ	448 (58%)	316 (42%)
Outside	178 (43%)	237 (57%)

Table 9.4: Academic alert-set statistics

9.2 Reducing the volume of alerts

The main component that reduces the volume of alerts is the AGC component, which transforms alerts to aggregated alerts. Table 9.5 shows the numbers of alerts produced by sensors for both data-sets and the numbers of aggregated alerts to which these alerts are transformed, by the AGC components. The corresponding reductions are presented as percentages.

	Darpa dataset		Academic dataset	
	inside	DMZ	DMZ	outside
Number of alerts	1001	2645	415	764
Number of aggregated alerts	357	164	84	111
Percentage of reduction	65%	94%	80%	86%
Total percentage of reduction	86%		84%	

Table 9.5: Performance of AGC component for all sensors

The AGC component reduces the overall volume of alerts by approximately 85% in both datasets. This surely makes the alert-set easier to be examined and additionally enables next components to function in a more efficient way.

The Merger also helps in the alerts volume reduction direction, by discarding identical aggregated alerts, produced by different sensors. In Table 9.6 the performance of the Merger component is shown for both data-sets.

	Darpa dataset	Academic dataset
Number of aggregated alerts	521	195
Number of aggregated alerts after merging	440	113
Percentage of reduction	16%	42%

Table 9.6: Performance of the Merger component for both datasets

The reduction of alerts that happens in the Merger component is relevant to the percentage of network traffic existing in both sensors' placements. If the firewall blocks most of the traffic, low redundancy between the aggregated alert-sets will exist and the percentage of reduction will be relatively low. On the other hand if the firewall allows most of the traffic in, then higher redundancy will exist between the two alert-sets and higher reduction of alerts will occur in the Merger component.

The last component, in which a reduction in volume of data takes place, is the Clusterer. The Clusterer detects related aggregated alerts and groups them in clusters. The numbers of clusters created in relevance to the numbers of aggregated alerts, fed as input to the Clusterer, are depicted in Table 9.7.

	Darpa dataset	Academic dataset
Number of aggregated alerts	440	113
Number of clusters produced	24	56
Percentage of reduction	95%	61%

Table 9.7: Performance of the Clusterer for both datasets

The Clusterer stores in the database of the system an impressively low number of data instances, as compared to the numbers of alerts produced by Snort sensors. Specifically for the Darpa dataset, the sensors of which initially produced 3646 alerts, 24 clusters are produced and for the academic data-set, the sensors of which initially produced 1179 alerts, 56 clusters are produced. This reduction in the volume of data is important, as it enables both the analyst to easily read the clusters' list and the Visualization subsystem to efficiently visualize it.

9.3 Reducing false positives

9.3.1 Validity scores

The main task of the Sensor manager is to calculate validity scores for the alerts produced by the corresponding sensor. As it has been analysed in Chapter 6, four components NRA, NRAS, HAF, RFP calculate partial scores, which are then combined to produce a final score. The average scores produced by the components and by the system as a whole for true and for false alerts have been calculated. These are shown in Table 9.8 for the Darpa dataset and in Table 9.9 for the academic dataset.

	Inside sensor		DMZ sensor	
Component	TA avg	FA avg	TA avg	FA avg
NRA	0.313	0.135	0.951	0.014
NRAS	0.140	0.021	0.287	0.002
HAF	0.663	0.564	0.652	0.229
UFP	1.000	0.795	1.000	0.090
SYS	0.467	0.327	0.835	0.085

Table 9.8: Average validity scores for Darpa dataset

	DMZ sensor		outside sensor	
Component	TA avg	FA avg	TA avg	FA avg
NRA	0.791	0.378	0.874	0.174
NRAS	0.305	0.140	0.293	0.029
HAF	0.999	0.586	0.999	0.563
RFP	1.000	0.610	1.000	0.633
SYS	0.706	0.425	0.590	0.349

Table 9.9: Average validity scores for academic dataset

It is obvious from Tables 9.8 and 9.9 that the Sensor manager discriminates between true and false alerts, which was the initial motivation for this subsystem. Depending on the dataset and the location of each sensor the differences between the validity scores of true and false alerts vary. In all cases there is a significant difference between scores for true and false alerts.

Regarding the Darpa dataset, the system has produced an average validity score of 0.476 for the true alerts of the inside sensor, while the corresponding average validity score for false alerts is 0.327. The true alerts' validity score is on average 42% higher than the score of false alerts. For the DMZ sensor of the same dataset the results are almost perfect. The true alerts' average score is 0.835, which is 980% higher that the corresponding false alerts score (0.0835). In this case the system has successfully detected most of the false alerts, with a high confidence, as it has produced minimal validity scores for the majority of them.

On the other hand, regarding the academic dataset the system has performed almost the same for both sensors. Specifically for the DMZ sensor true alerts' scores average at 0.706 which is 66% higher than false alerts scores' average (0.425). For the outside sensor the average score of true alerts is 0.590 and is 69% higher than false alerts' average score, which is 0.349.

In general the true alerts' average validity score is significantly higher than that of the false alerts for all cases. This enables the system to attach useful truth values on each alert, which are utilized by subsequent components. Even if the Darpa DMZ sensor high difference should be regarded as a special case, the results for the remaining three sensors prove that Sensor manager component is capable of efficiently discriminating between true and false alerts.

The four components seem to have deviations in their performance. This is normal as they examine specific alerts' properties and the results obtained are normally different for different intrusions, or even different sensors' placements. The logic of using multiple components is not to utilize them on their own, but to combine their results, to get a useful validity estimate for as more cases as possible.

9.3.2 ROC analysis

As it has been stated in Chapter 6, no discarding of alerts is conducted in Sensor Manager. In order to examine NRA, NRAS, HAF, RFP components and Sensor manager as a whole in terms of classification performance, a ROC curve analysis [69] has been conducted. The use of a threshold value val_{thr} indicates that every alert with higher validity score than val_{thr} is considered as positive, while every alert with validity score lower than val_{thr} is considered as negative and is discarded. The use of a threshold creates four different categories of alerts:

- **True positives :** True alerts that have been classified as positives.
- **False positives :** False alerts that have been classified as positives.
- **True negatives :** False alerts that have been classified as negatives.
- **False negatives :** True alerts that have been classified as negatives.

These categories are also shown in Table 9.10.

From these values some interesting metrics can be calculated. True positives rate (TPR) is the fraction of real alerts, classified as true, while False positives rate (FPR) is the

		True class	
		p	n
Hypothesized class	Y	TP	FP
	N	FN	TN

Table 9.10: Categories of alerts after threshold use

fraction of false alerts, classified as true. These metrics are calculated in Equations 9.1 and 9.2.

$$TPR = \frac{TP}{p} = \frac{TP}{TP+FN} \tag{9.1}$$

$$FPR = \frac{FP}{n} = \frac{FP}{FP+TN} \tag{9.2}$$

(9.3)

The ROC curve is a graph where FPR is denoted in the x-axis and TPR is denoted in the y-axis. A classifier corresponds to a point in this graph according to its TPR and FPR. If all possible threshold values are examined for a classifier, then the corresponding points form a curve called ROC curve. The y=x line corresponds to the random choice classifier. Every classifier's curve should be as much higher as possible from this line, to justify its good performance. Curves close to this line or even under it, indicate inadequate classifiers, while curves close to $(0,1)$ indicate well performing classifiers.

For each one of the two sensors, for both datasets all possible threshold values in the $[0,1]$ range have been examined. By using a threshold value the scores produced by each one of the components and the system as a whole can be used in a classification procedure. The relevant rates have been calculated and the resulting ROC curves are shown in Figures 9.1,9.2,9.3 and 9.4.

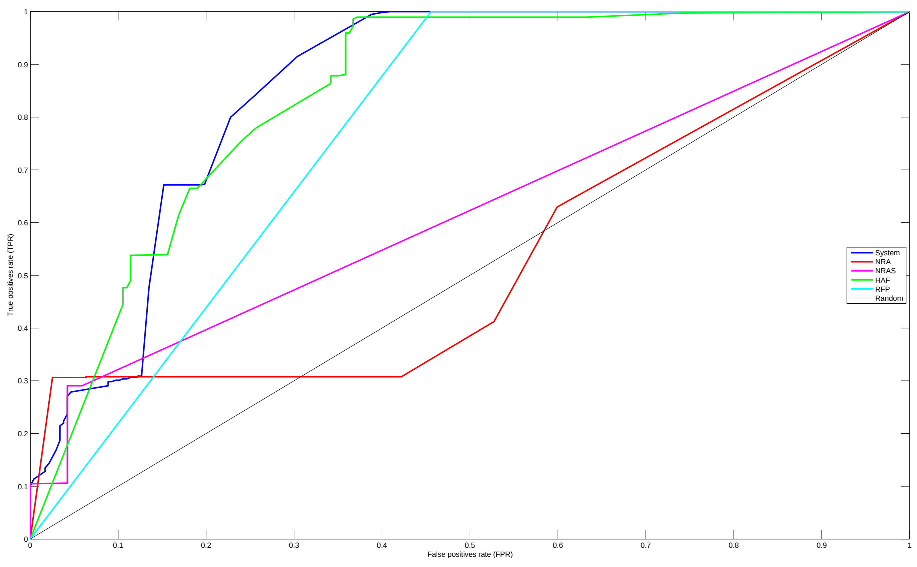

Figure 9.1: Darpa inside ROC curve

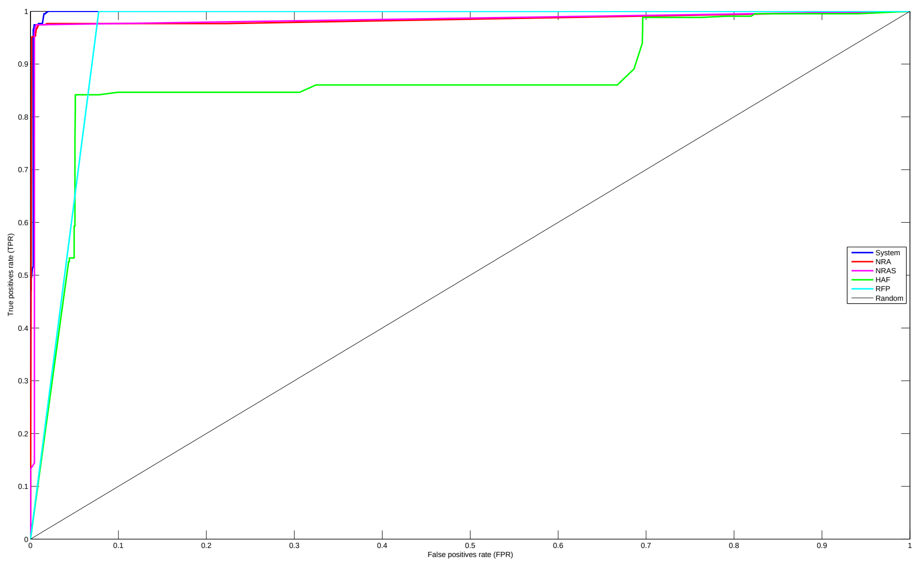

Figure 9.2: Darpa DMZ ROC curve

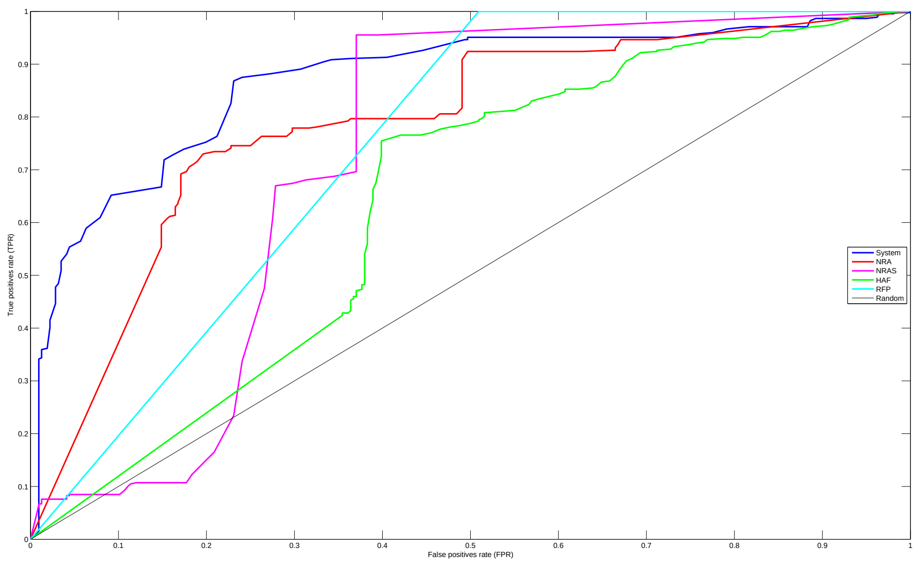

Figure 9.3: Academic DMZ ROC curve

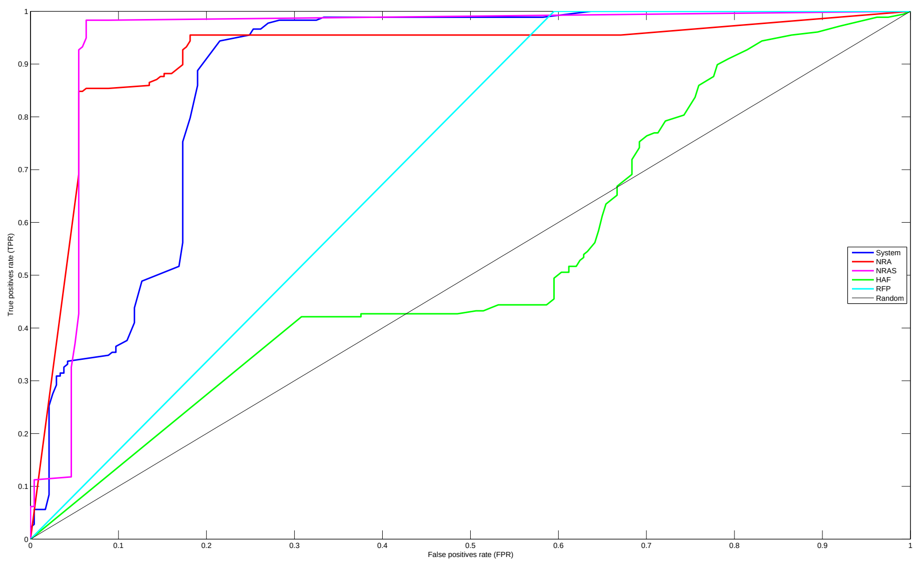

Figure 9.4: Academic outside ROC curve

By examining the ROC curves for the Darpa dataset, it is obvious that the blue lines that correspond to the output of the system indicate a well performing classifier for both sensors. In the inside sensor's case, the individual components' performance varies.

While the curve of the HAF component is close to the system's one, the curve of the NRA component is close to the random classifier's line. On the other hand in the DMZ sensor's case, all components perform well. The ROC curves of the academic dataset show a similar situation, with the curve of the system situated significantly higher than the random classifier. The components' curves vary from being very good to being almost unacceptable.

As it has been analysed, the individual performances of the components are not significant as long as the overall system performs well. The NRA components in the Darpa inside case or the HAF component in the academic outside case surely do not perform well in general. However they are useful, as they may produce critically low or high scores for a small subset of alerts, that enable the system as a whole to perform well.

A metric for the performance of a classifier is the area under the ROC curve (AUC). The larger this area is, the better the classifier is. AUC values have been calculated for all components in all cases and are shown in Table 9.11.

	Darpa dataset		Academic dataset	
Component	inside	DMZ	DMZ	outside
NRA	0.528	0.984	0.787	0.917
NRAS	0.616	0.983	0.722	0.944
HAF	0.841	0.869	0.635	0.528
RFP	0.772	0.961	0.745	0.702
SYS	0.854	0.998	0.869	0.877

Table 9.11: AUC values for all components in all cases

As it was expected, the AUC values for the individual components range from 0.528 to 0.944. The most important indication of the good performance of the system is that in all cases examined the AUC value for the Sensor manager subsystem is above 0.85.

9.4 Visualization results

All the enhancements mentioned in previous Sections have an impact on the visualization produced by the Visualization subsystem. The three dimensional graphs produced by this subsystem are informative for the security state of the system, without demanding much effort from the security analyst. In Figures 9.5 and 9.6 the final visualizations produced for both datasets are depicted.

In the specific visualizations a validity filter has been used. Clusters with validity scores lower than 0.3 have been ignored in the formation of the resulting surfaces, to produce a clearer result.

9.4.1 Darpa data-set visualization analysis

Regarding the Darpa visualization, there are indicative peaks for all attack steps of the intruder, while irrelevant peaks are a minority. The same visualization is depicted in Figure 9.7 with all peaks corresponding to intruders actions labelled accordingly to the relative phase of the attack.

For phase 1 the cluster is not depicted as clearly as possible. The sweeping of the network has not produced a continuous, in terms of IP values, cluster, so three different peaks correspond to this phase. The first two peaks are also related with traffic of the next phases so they are longer in the time dimension, while the third one is correctly positioned on the y axis. In general the analyst is informed about suspicious traffic on time, but the characteristics of the actions of the intruder are not obvious on the graph.

For phase 2, on the other hand, there is a clear peak showing the probing activity on alive hosts discovered during the previous step.

For phase 3 there are two different peaks that clearly show the occuring activity. The graph depicts the IP range of the cluster instead of its individual IPs, so there is no clue about the specific IPs related to activity. The analyst can get these information by inspecting the clusters list stored in the database. For phase 4 there is also a representative corresponding peak.

Finally for phase 5 there is a corresponding peak which stimulates the analyst to further investigate the alerts produced by the sensors. The peak is large in width (in terms of IP values), because of the nature of DDos attack.

Except for the peaks described above, there are some peaks irrelevant to the actions of the intruder, which are also labelled on the graph. They correspond to clusters formed from false alerts. Their impact on the final visualization is significantly lower than that of the high percentage of false alerts in the initial alert-sets.

9.4.2 Academic data-set visualization analysis

The visualization of the academic dataset has been labelled according to the phases of the attack executed and is shown in Figure 9.8

The IP sweeping of phase 1 is depicted by three different peaks, which cover most of the IP range of the protected network. The same stands for phase 2. The difference is that the peaks seem more specific in terms of IP addresses. This is expected, as in the first phase all IPs of the network are scanned, while in the second phase only hosts found to be alive in the first phase, are probed.

Regarding phase 3, there are three peaks indicating activity around the IP of the MVM host, and span a 6 minute range, in which the intruder unsuccessfully tried to exploit the Tomcat vulnerability.

Regarding the representation of the last two phases the system under-performs as the alerts for both steps are included in the same cluster and inevitably they produce only one peak. This peak is extended between the two IPs of MVM and UVM so the analyst cannot clearly understand what is really happening in the system. For both of these phases the sensors produce very few alerts in the initial alert-set, so the problem in the resulting visualization is partly due to the poor performance of the intrusion detection sensors.

Another important remark is that peaks, irrelevant to actions of the intruder, are more in number than they were in the Darpa dataset case and make the reading of the graph slightly harder. Like with the Darpa dataset, the overhead imposed by irrelevant peaks, is significantly lower than that of false alerts in the initial alert-sets.

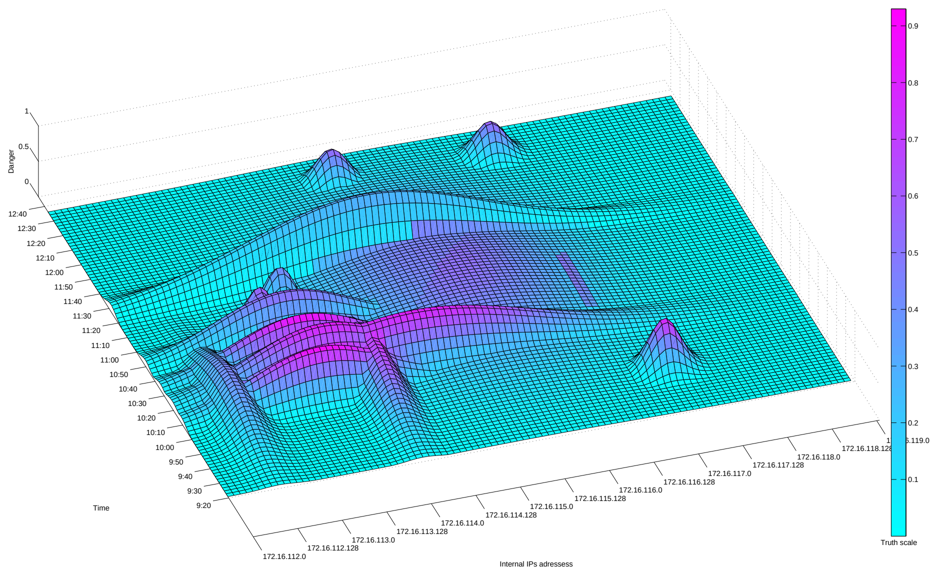

Figure 9.5: Visualization for Darpa dataset

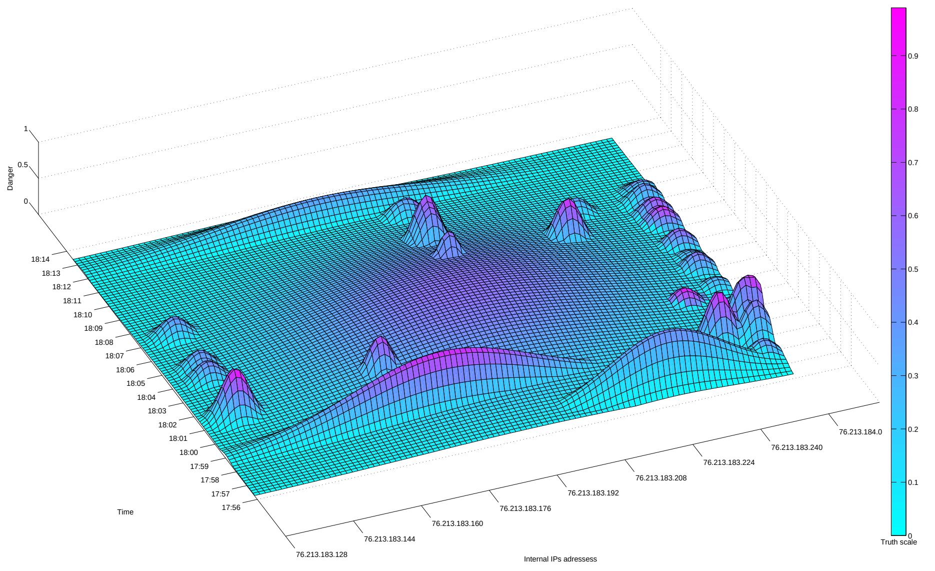

Figure 9.6: Visualization for academic dataset

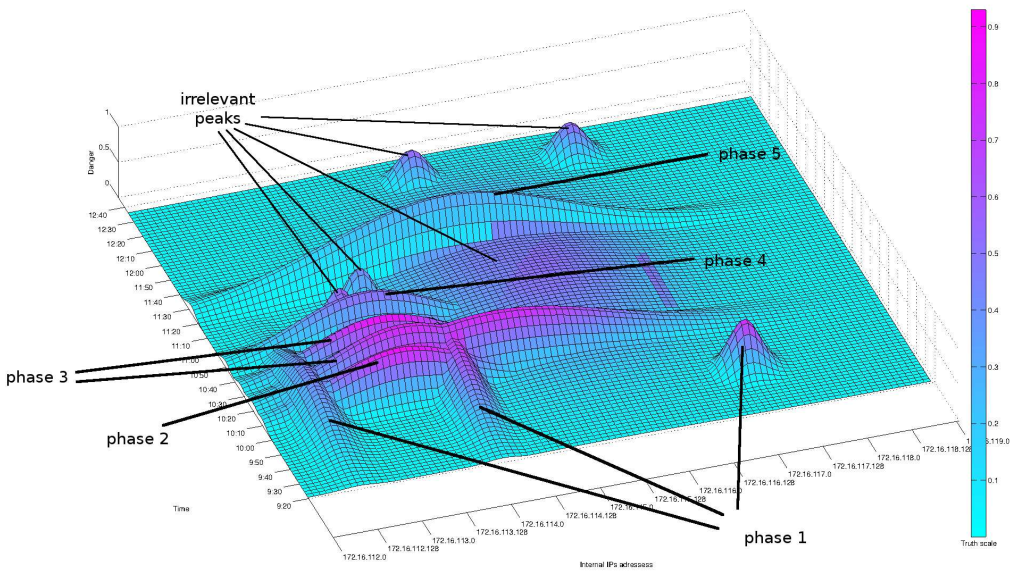

Figure 9.7: Visualization for Darpa dataset with phase labels

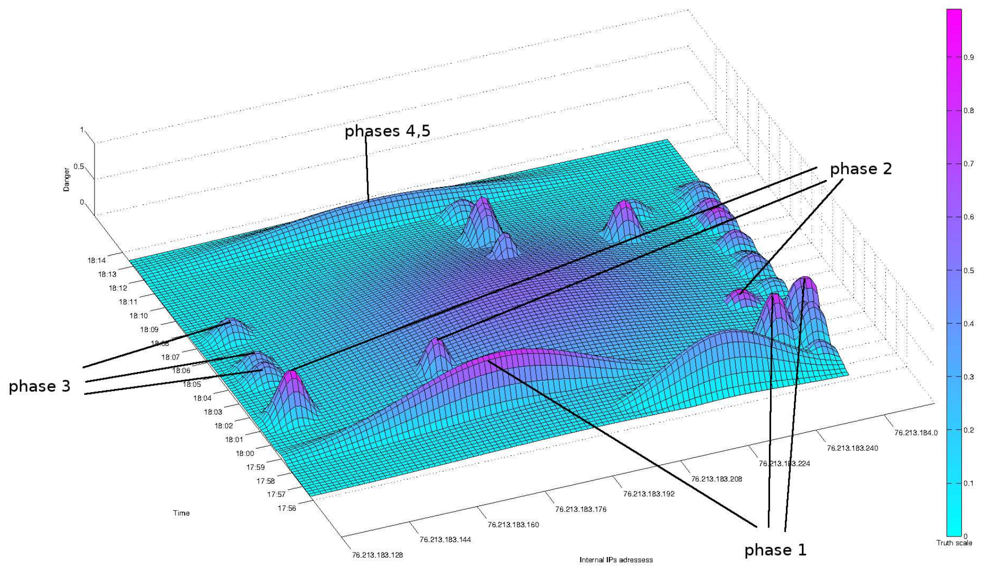

Figure 9.8: Visualization for academic dataset with phase labels

9.5 Hypothesizing on missed events

An additional task for the Clustering subsystem is to hypothesize on missed intrusion events and approximate relevant information. In order to test the performance of the system for this task, subsets of the Darpa dataset have been used. Specifically hypothetical alert-sets were created, which reflect the cases in which distinct phases of the attack scenario have been missed by both sensors.

The Darpa dataset has been examined and the procedure followed was to manually recognise the alerts pertaining to each attack phase (the intermediate phases 2,3,4) and to remove them in an iterative mode. Three alert-sets were created, each one of which corresponds to the scenario that one of the three intermediate phases has been missed by the Snort sensors. These three scenarios were then used as input to the system in both simple and advanced mode. All scenarios are shown in Table 9.12.

Testing Scenarios	
Name	Description
Phase 2 out	Alerts of Phase 2 are dropped
Phase 3 out	Alerts of Phase 3 are dropped
Phase 4 out	Alerts of Phase 4 are dropped

Table 9.12: Testing hypothesizing performance scenarios

Figures 9.9 and 9.10 show the results of simple and advanced functioning of the system for the scenario "Phase 2 out". Phase 2 regards probe of live IP to look for the sadmind daemon, running on Solaris hosts. Phase 2 time range spans from 10:07:07 to 10:18:05. The Clusters generator produces an artificial cluster that sufficiently approximates the probe regarding both time and IP values.

Figures 9.11 and 9.12 show the results of simple and advanced functioning of the system for the scenario "Phase 3 out". Phase 3 regards breaking in via the sadmind vulnerability, on the hosts found during phase 2. It includes both successful and unsuccessful attempts. This phase lasts from 10:33:10 to 10:34:59. Again the Clusters generator produces a very good estimate of the missed real event.

Finally Figures 9.13 and 9.14 show the results of simple and advanced functioning of the system for the scenario "Phase 4 out". Phase 4 regards installation of the trojan mstream DDoS software on three of the hosts discovered in previous phases. The time range of this phase is from 10:46:28 to 11:00:32. The Clusters generator produces a relative cluster depicted in Figure 9.14. The peak is situated at the end of the time range of the phase, while the IPs range is a good approximation of the internal IPs related to this phase. Because the IPs of the three hosts are not close to each other (172.16.115.20, 172.16.112.50 and 172.16.112.10), the IP range of the produced cluster is not sufficiently informative for the analyst. In this scenario there are two additional artificial clusters which are situated close to phase 2 clusters, so they are not distinguishable in the graph and do not confuse the analyst.

In general the estimation of missed events is satisfactory. It enables the analyst to be more effective in guessing events not detected by the sensors. The motivation behind the component was not to produce alerts for missed events, which is obviously impossible, but to produce estimates about missing clusters. The Figures mentioned in previous paragraphs show that informative estimates are achieved.

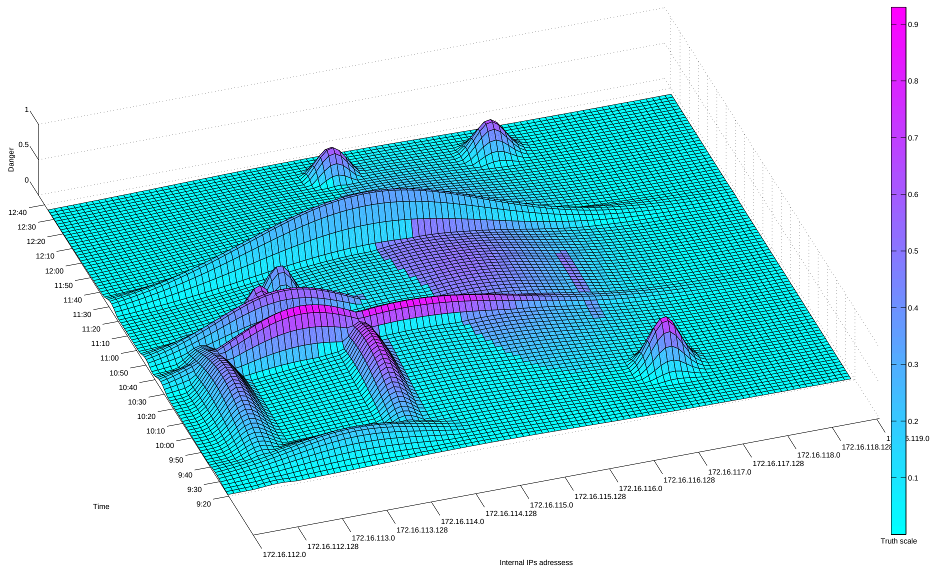

Figure 9.9: Darpa phase 2 out scenario (simple mode)

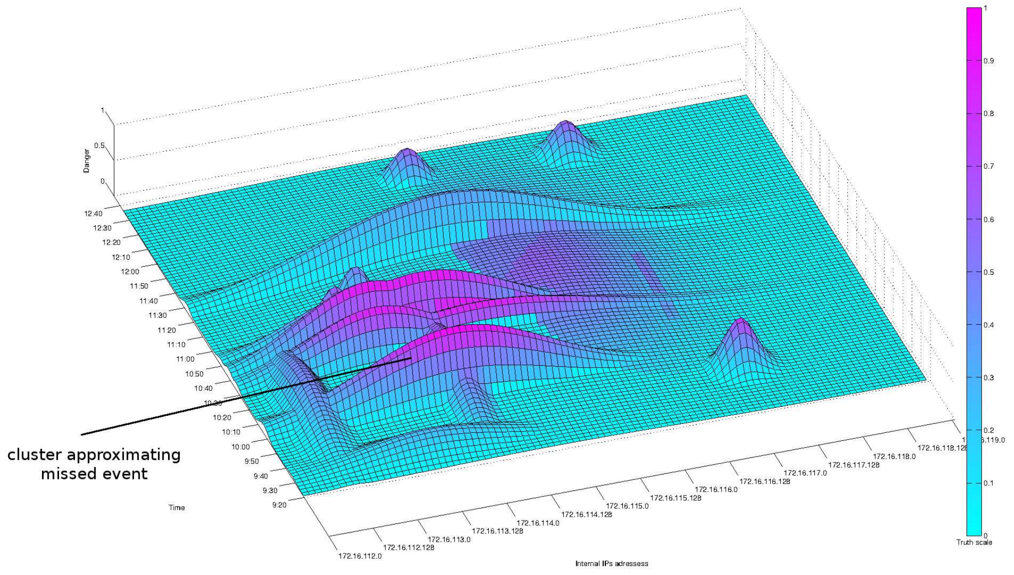

Figure 9.10: Darpa phase 2 out scenario (advanced mode)

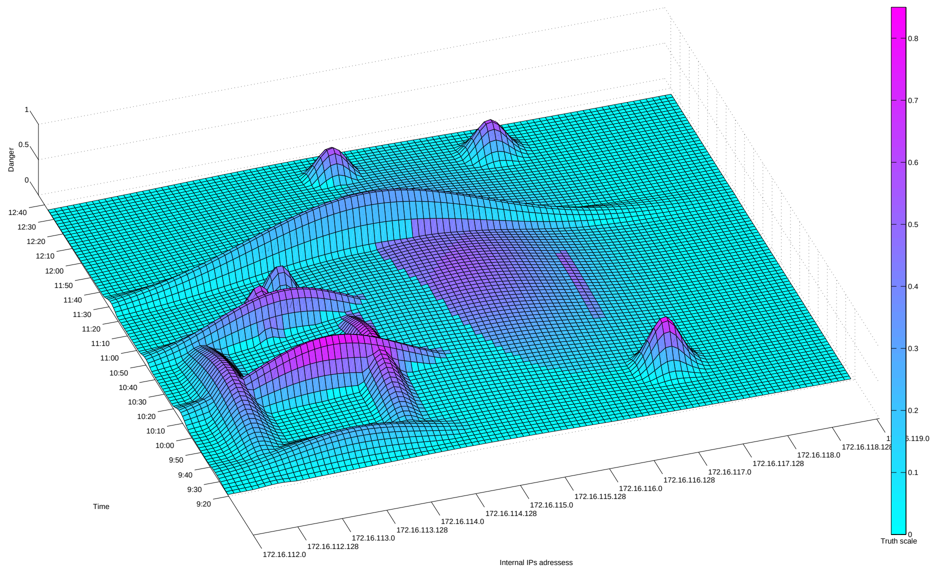

Figure 9.11: Darpa phase 3 out scenario (simple mode)

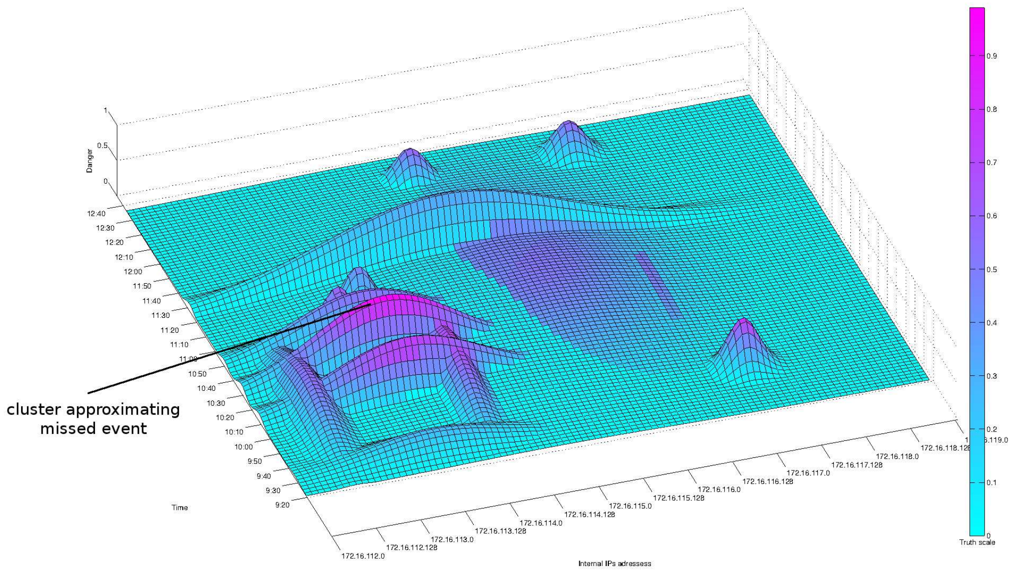

Figure 9.12: Darpa phase 3 out scenario (advanced mode)

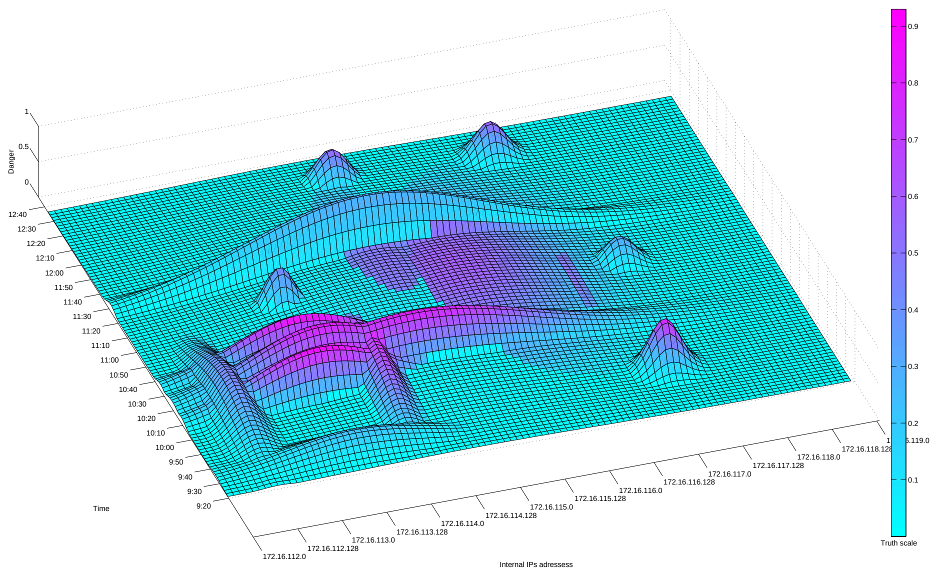

Figure 9.13: Darpa phase 4 out scenario (simple mode)

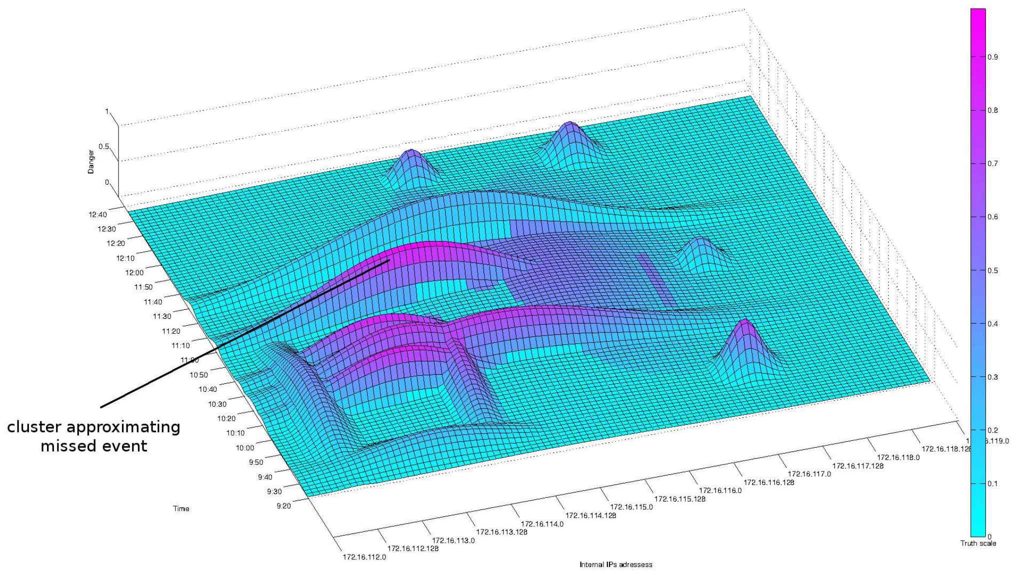

Figure 9.14: Darpa phase 4 out scenario (advanced mode)

9.6 Data processing performance of the system

The data processing performance of the system has also been measured. The Darpa dataset has been used to measure the time required by the system, to process various numbers of alerts. The simulation speed has been set to maximum, so the only latency that exists in the system is due to processing overhead. The algorithm was executed on a Intel Core i7 with 4GB of RAM. Table 9.13 depicts the execution times (in ms) for various numbers of alerts :

Number of alerts	Time (ms)
1000	3081
1250	3683
1500	4757
1750	5462
2000	5973
2250	6611
2500	7416
2750	8095
3000	9254
3250	9673
3500	9769

Table 9.13: Execution times

These measurements have been marked on a graph, depicted in Figure 9.15, which shows the time values in relevance to the numbers of alerts. The points in the graph are sufficiently approximated by the line :

$$time = 2.9 * number\ of\ alerts \tag{9.4}$$

This means that each alert needs 2.9 milliseconds, on average, to be processed. A more detailed examination of the system has shown that some of this time is used to read the alerts from the database, during simulation. During real world application this won't happen, so the net processing time needed is more or less 2 milliseconds. In other words the system can process around 500 $\frac{alerts}{sec}$, without introducing processing overhead. If a more powerful machine is used, then this rate will be much higher. It is obvious that the system can be applied to modern, large, high speed networks with multiple intrusion detection sensors.

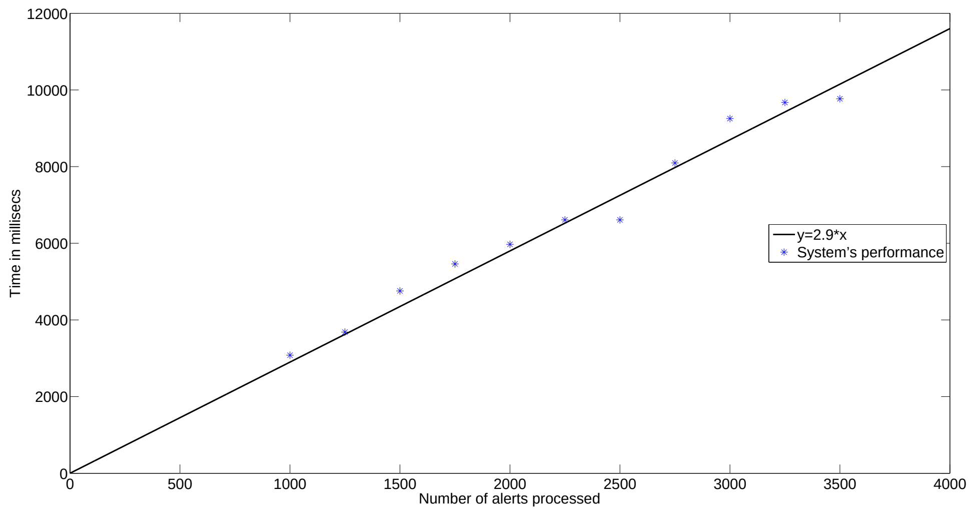

Figure 9.15: Execution time with respect to number of alerts

The calculated processing rate is a multiple of the rate required in current intrusion detection systems deployment scenarios. A research on high alert volume scenarios in recent literature has indicated that the maximum rate required is 170 $\frac{alerts}{sec}$.

Specifically the authors in [70] have collected traffic in the ETH campus and their alert rate was 3 million alerts per day. In [71] experiments have been conducted in the campus network of the Xian Jiaotong University. The alerts collected were 1 million per day. Researchers in [72] have used alerts submitted to DShield service [73] and had to cope with processing 15 million alerts per day. The authors in [74] have also worked on a University campus network and collected 117 millions alerts throughout the period of one month. The alert rate referred in each one of the above research efforts is shown in Table 9.14, in terms of alerts per second.

The processing rate of the system is on average $500\frac{alerts}{sec}$, much higher than the alert rates found in the literature; thus the proposed methodology is capable of coping with real world networks, characterized by high volume of alerts.

Reference	Rate mentioned	Rate (alerts/sec)
[70]	3M / day	34
[71]	1M / day	11
[72]	15 M /day	170
[74]	117 M / month	45

Table 9.14: Referenced alert rates

CHAPTER 10

Using Fuzzy Inference Systems to reduce false positives in IDS

Even if intrusion detection systems have marginally improved in the past few years, they still face the problem of high false positives rate. In this Chapter the use of a fuzzy inference system, which filters out false positives, without missing on any of the detected attacks, is proposed. The design of the system is based on meta-alerts, which carry special information about the nature of alerts. The system has been tested against the DARPA dataset and has exhibited a significant reduction (83%) of false positives.

10.1 Introduction

In this chapter an alternative approach on reducing false positives is described. As it has been discussed in previous Chapters intrusion detection systems exhibit high false positive rates, to the extent that the alert-sets they produce consist mainly of false alerts. This makes them difficult to use; consequently, much research effort has been invested into devising ways for filtering false positives.

The approach discussed in this chapter is a false positives filtering method based on fuzzy inference systems (FIS) [75]. The rationale of the method is related to the validity score producing scheme existing in Sensor Manager, where statistical observations on the nature of false and true positives are utilized. Herewith, the method is enhanced by using a robust and competent tool, such as fuzzy inference systems. The result of this enhancement is a clear improvement in performance, enhanced configurability and feasibility for supervised learning.

10.2 Related Work

A lot of research has been conducted in improving the performance of intrusion detection systems. Fuzzy logic and fuzzy inference systems have been widely used in this research.

In [76] a FIS (Fuzzy Inference System) was used within an intrusion detection system. The result was better detection rates, regarding port scanning attacks. In [77] fuzzy logic was incorporated in order to change the detecting method of intrusion detection systems. The authors of [78] created a two layer architecture, whose first layer consists of five trainable FISs, while the second consists of one FIS, which is based on expert knowledge.

Several experiments on anomaly based intrusion detection have been based on FISs. In [79] a FIS is one of the three parts of an anomaly-based intrusion detection system. In [80] another anomaly-based system uses fuzzy logic. In [81] an interesting variation of FIS is presented. EFIS which stands for evolvable FIS tries to automatically follow the changes happening regarding the normal behaviour model, used in anomaly-based intrusion detection.

It is common to use Neural Networks along with a FIS. This usually happens in order to define the FIS parameters, through the training process of the neural network. Examples of such methodologies are [82] and [83].

Fuzzy logic has also been used in [84], to improve Hidden Markov Models detection capabilities. Fuzzy logic and fuzzy sets are much more appropriate than crisp logic for describing the existence of an attack. It has also been used in [36] to cope with another challenge in intrusion detection research, namely alert aggregation.

In [85] and [86] a FIS was used to fuse the results of other methods. The architecture of a FIS, which will be analysed in Section 10.3, allows multiple rules that can support many inputs to the system. These inputs may as well be the outputs of other systems.

While all the above work has been conducted in order to change the way intrusion detection systems function, in [87] the authors try to post process an intrusion detection alert set. This approach resembles work presented in this Chapter and the results obtained are comparable. The method of [87] is based on training, which makes the proposed system very dataset-dependent. Authors achieved a slightly higher percentage on false positive reduction; however, it is not stated therein whether the filtered alert set contains alerts for all attacks.

10.3 Fuzzy Logic And Fuzzy Inference Systems

Fuzzy logic is based on the idea that the claim about the truth of a fact is not necessarily absolute (meaning that the fact can be either true or false). In fuzzy logic this claim can also be relative (the fact can be true in some degree).

FIS are based on fuzzy logic. They are systems that take one or more inputs and export results for one or more outputs. The mapping of the value of inputs and outputs to a degree of membership (a number between 0 and 1) is conducted through the use of membership functions. The relation between inputs and outputs is declared by if-then rules. The format of these rules is:

$$\text{If input1 is low and input2 is high then output is high} \tag{10.1}$$

All the inputs are fed into the FIS. Through the use of membership functions, input values are translated into degrees of membership regarding predefined fuzzy sets. Then if-then rules are applied and specific results are produced for the outputs by each rule. The results of all rules are aggregated into one single result for each output. These results consist of the combination of truncated diagrams into one diagram for each output. The last phase that takes place is defuzzification, where these combined fuzzy diagrams are converted into single numbers, through the use of appropriate methods.

FIS can be very useful in domains where human knowledge can be translated into membership functions and if-then rules. If the parameters for these components are wisely set, then FISs can achieve great results in terms of predicting values or classifying instances.

10.4 Logic of the filter

The system is based on the logic used in the Sensor manager subsystem. False and true positives can be identified by different characteristics that seem to describe each category.

For specific kinds of attacks multiple alerts are produced; these can either be identical or have noticeable similarities to each other. These similarities in most cases can be denoted by their source and destination IP addresses. By checking whether an alert belongs to a group of related alerts, an indication of whether it is true or not can be derived.

Another common observation for true alerts is that they tend to increase the frequency of appearance for their signatures. Generally speaking, an occurring attack should probably create an anomaly in the usual distribution of alerts. If the standard frequency of alerts carrying a specific signature can be calculated, then the alerts that cause a substantial raise in this frequency can be considered true ones.

False positives on the other hand usually derive from specific causes, which may be related to the topology of the network, mis-configured hosts or periodical nominal services and tasks that occur in the network. All these causes are stable and produce recurrent patterns of FPs. If these patterns of FPs could be extracted, then they could be used to detect false alerts and to filter them out.

All these observations were used in Sensor Manager subsystem presented in Chapter 6. By incorporating the same logic in a FIS, a flexible system can be built on the basis of fuzzy sets.

10.5 FIS Application

The input to FIS used consists of meta-alerts created from the intrusion detection alerts. For each alert, meta-data, relative to the observations stated in Section 10.4, are calculated. The calculations for each specific alert are based on an alert window around it. For example a window of width equal to 200 alerts consists of 100 alerts before the specific alert and 100 alerts after it. The relevant meta-data for a specific alert are:

- The percentage of alerts in the window that carry the same signature with the specific alert.
- The percentage of alerts in the window that have a similarity in source or destination IP addresses with the specific alert.
- The percentage of alerts in the window that have a similarity in the subnets of source or destination IP addresses with the specific alert.

- The average time difference between timestamps associated to alerts within the window and the timestamp of the specific alert.
- The number of appearances of the signature of the specific alert in an attack free period.
- The percentage of alerts in the window, of which source and destination IPs are related to the source and destination IPs of the specific alert in a sweep-suspicious manner (they have one equality of IPs and one equality of subnets of IPs).

For each alert all these data are calculated and combined to one corresponding meta-alert. The features of meta-alerts are summarized in Table 10.1.

Description	Acronym
Percentage with the same signature	same_sig
Percentage with the same source or destination IPs	same_s_d_ip
Percentage with the same source or destination subnet	same_s_d_sub
Average time difference	av_t_d
Number of appearances in the attack free period	afp_ap
Percentage of sweep related alerts	sweep_rel

Table 10.1: Meta-alerts' features

These meta-alerts are imported into the FIS. For each input a membership function that converts its values to degrees of membership regarding fuzzy sets is needed. By observing the values of the meta-alerts features for true and false alerts it is easy to deduce that they follow different distributions. Based on these distributions, it is easy to create membership functions that can discriminate between false and true positive alerts. For example Figure 10.1 and Figure 10.2 show the distribution of same_sig metadata for FPs and TPs respectively.

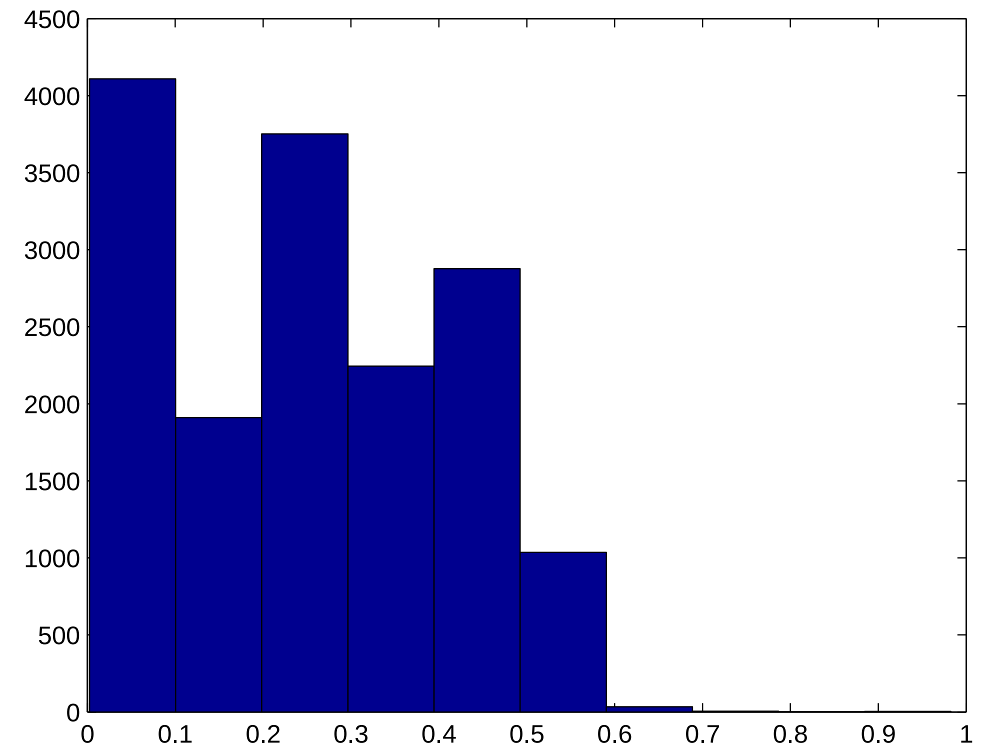

Figure 10.1: Distribution of FP alerts according to same_sig

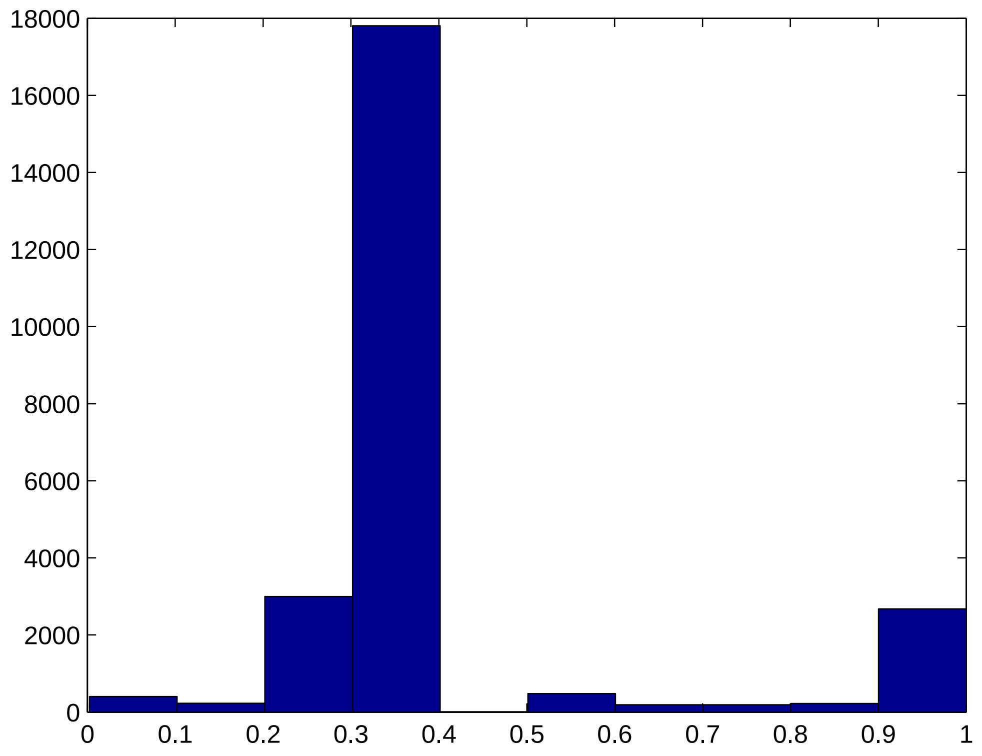

Figure 10.2: Distribution of TP alerts according to same_sig

Figure 10.3 shows the membership function deduced. This feature will be high for many TPs, while it will not be high for almost all FPs. Figure 10.4 depicts the membership functions for all 6 inputs, which are designed in a similar manner.

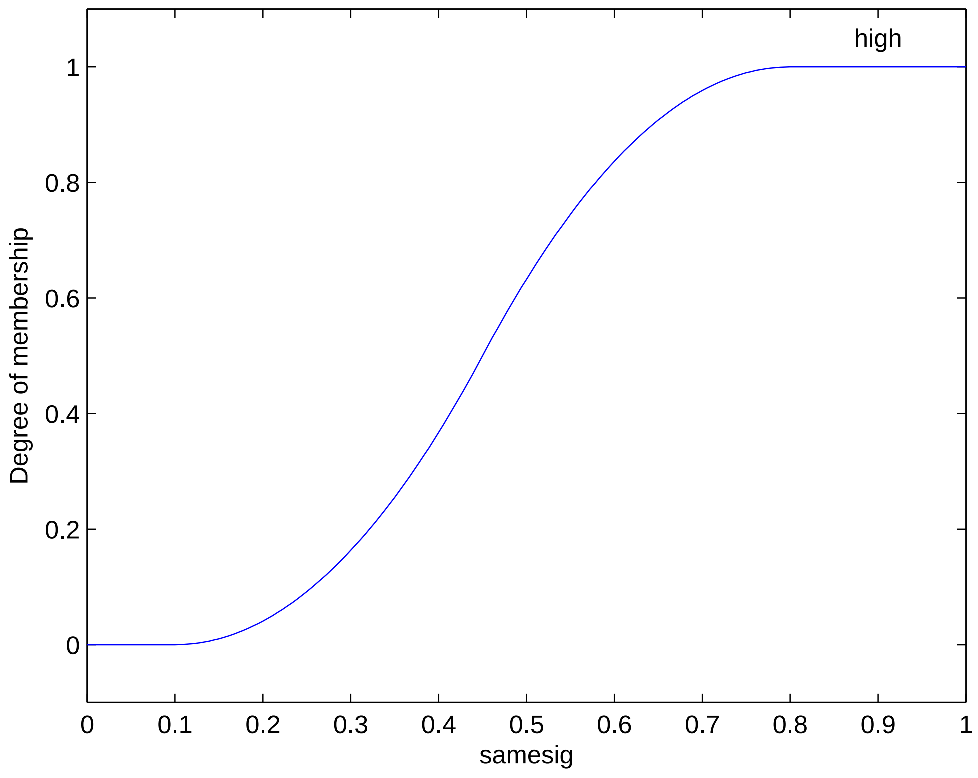

Figure 10.3: Membership function for input of same_sig feature

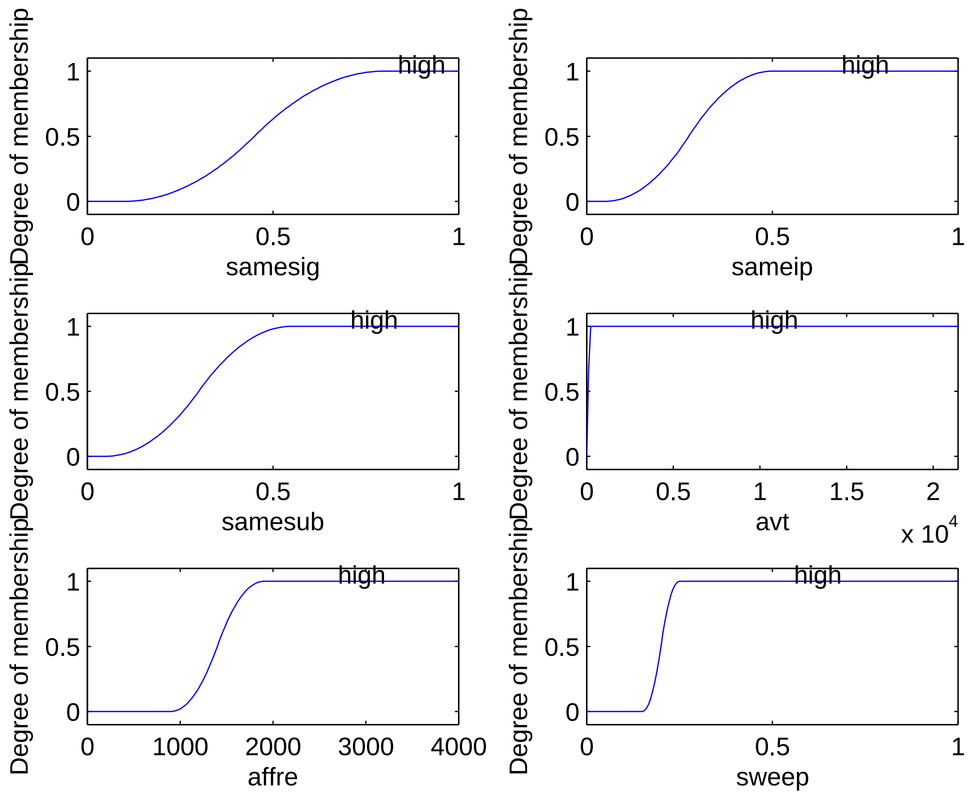

Figure 10.4: Membership functions for all inputs

The next step is to create the if-then rules, which will connect the inputs to the output. Based on the observations mentioned in Section 10.4 the rules used are:

- if (same_sig is high) then (alert is tp)
- if (same_s_d_ip is high) or (same_s_d_sub is high) then (alert is tp)
- if (avt is not high) then (alert is tp)
- if (afp_ap is not high) then (alert is tp)
- if (sweep_rel is high) then (alert is tp)

The rules are aggregated by picking the maximum out of the five rules results (degrees for the output to be TP), while the defuzzification method used is centroid, which is the most common choice.

The output is a value in the interval [0,1] which corresponds to the degree in which the specific alert is a true positive. A threshold is then used in order to determine which alerts are true and which are not.

10.6 Experimental Results

The dataset used for designing and testing the filter has been created by the DARPA off-line intrusion detection evaluation in 1999 [88]. A simulation of a real world network was created and more than 200 attacks against it were attempted. The network traffic was captured during simulation to create the data-set.

This traffic has been replayed and Snort has been used in order to produce a relevant alert set. This alert set is the input to the system. The features of alerts used are their signature ids, their timestamps and their source and destination IPs.

The resulting FIS was tested against the 2nd week's data of DARPA 1999 dataset network traffic. Snort (version 2.6) has been used to detect attacks from these traffic data. These data include 43 real attacks. Snort produces 41,164 alerts which are related to 24 out of 43 real attacks. 38.79% (15,970 alerts) of the alerts produced are false positives.

These alerts were used to produce meta-alerts and the meta-alerts were used as input to the system. A threshold should be defined, to classify alerts into TPs and FPs. Various threshold values have been tested. In Figure 10.5 the number of alerts filtered in is depicted against the value of the threshold.

As is shown in Figure 10.5, the best value for the threshold parameter is 0.726. If this value is used, the system filters in alerts for all attacks (all attacks detected by Snort), while it reduces FPs by 83%. The comparison of the results to the results of using only Snort are shown in Table 10.2.

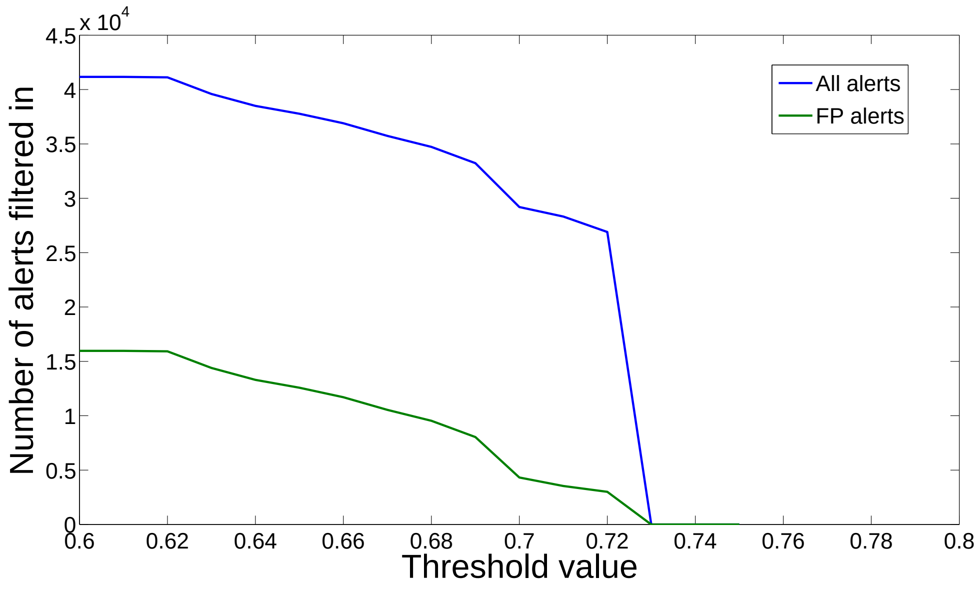

Figure 10.5: Membership functions for all inputs

It is obvious that the proposed fuzzy inference system greatly improves the quality of the alerts, as it vastly reduces the rate of false positives, while it filters in alerts for all attacks detected by Snort.

	FIS	Snort
Number of filtered in alerts	23833	41164
Number of FPs	2624	15970
Percentage of FPs	11,01%	38,8%
Number of attacks detected	24	24

Table 10.2: Results of FIS compared to Snort

CHAPTER 11

A test-bend for intrusion detection systems results post-processing

Intrusion detection systems produce alert-sets of low quality. Many post-processing methods have been proposed to make alert sets more meaningful to security analysts. Relevant research has to deal with an important task; implementing proposed methods and carrying out required experiments. In this Chapter a platform which can be used as a test-bend for conducting intrusion detection alerts post-processing experiments is proposed. All the standard functionality is already implemented for the user, as she has to implement only the core logic of her method. Additionally the platform offers important reuse and evaluation capabilities. Finally a previously implemented method has been reimplemented on the platform to test its usefulness.

11.1 Introduction

A lot of research has been focused on post-processing of intrusion detection alerts in recent years. Intrusion detection systems usually produce alert-sets of low quality. These alert-sets are characterized by their enormous size, which is disproportional to the size of the relevant protected systems, their high rate of false positives and false negatives and their inconsistency in regards to the real attack plan committed. Researches have been recently working on this field intensively and have proposed a lot of interesting methods that utilize ideas from various science fields such as machine learning, data mining, fuzzy logic,time series etc.

These research efforts always include commonly used procedures such as reading alerts or evaluating results. Apart from that, researchers usually implement functionality they have already used in their previous work, while another important problem is that comparison of experimental results is most of the time a big issue.

If segments of the methodologies proposed were formally defined as components, then they would be suitable for reuse. They could be reused by their authors or even by others. Apart from that, standard evaluation components could make the direct comparison of results easier and more elaborate.

This Chapter presents the development of a software platform, implemented in Java, that enables researchers to implement the post-processing solutions they have designed, as interconnected components. The platform contains well defined models of all the standard functionality needed by researchers and provides it to them as ready components. Additionally it offers to them a clear and easy way to inject their methods in the solution. Emphasis has been given on the re-usability of the parts of the solution, in order for the user to be able to reuse her methods or distribute them to others. The evaluation part has also been standardized, in a way that results of different implementations are directly comparable.

11.2 Related work

Researchers in intrusion detection alerts post-processing have to make important implementation efforts in order to test their systems. The proposed platform provides them with the tools to efficiently implement their post-processing methods. No similar platform, that tests intrusion detection alerts post-processing methods, has been proposed in bibliography. There are some systems presented in articles published in the early days of intrusion detection research, that focus on testing the intrusion detection systems themselves.

Authors in [89] and [90] propose a platform on which the user can create scripts that simulate intrusive or normal behaviour, in order to test an intrusion detection system. They then systematically try to evaluate the performance of the intrusion detection system in test by observing the detected intrusions.

In [91] the system proposed injects dummy intrusion network traffic into the normal live traffic of a network. In this way a data-set for testing an intrusion detection system is created, while normal traffic is as realistic as possible. Moreover the user can test any intrusion she wishes, without creating any real security issues for the protected network.

In [92] signatures of Snort network-based intrusion detection system are used as input to an event stream generator that produces randomized synthetic events that match the input signatures. The resulting events stream is then used to trigger a number of different intrusion detection systems and the results are analysed.

It is obvious that these efforts are related to testing the actual intrusion detection systems. The motivation for the platform implemented is to provide researchers with an elaborate alerts post processing methods development environment. There is no other system proposed in bibliography, that shares the same motivation.

11.3 The problem

Intrusion detection has been a very intensively researched area in recent years. Many researchers work on this field and try to improve the quality of the results obtained by intrusion detection systems. While others try to achieve this by proposing improvements of the detection techniques, many researchers use post-processing of the produced alerts. They try to extract additional valuable knowledge of the security state of the protected system from the actual alert-set.

Generally produced alert sets are of low quality. The most common problem is high false positives rate. The percentage of false alerts, is usually so high that it is hard to isolate the real alerts from false ones. Additionally the relevance between events and alerts is not always obvious. A single event may produce multiple identical instances of the same alert or it can produce many alerts that differ in a small subset of their fields. Generally alerts are usually in lower level of complexity than the events that trigger them. All these factors contribute to the low quality of the produced alert set.

In general reading alert sets is impractical as they contain thousands of alerts, which are not all useful or they overlap, while many of them are false. The motivation of researchers dealing with alert post-processing is to improve the results obtained from intrusion detection systems in every possible way. The main concepts in post-processing of intrusion detection alerts are :

- False positives reduction (filtering)

- Aggregation
- Correlation
- Clustering
- Visualization

Many researchers are working in the field of post processing of intrusion detection alerts, in order to enhance the produced alert-set. An important part of their efforts is dedicated to implementing their methods and justifying their performance with relative experiments. The main problems hindering these efforts are :

- A lot of standard functionality, irrelevant to the methods' core logic needs to be implemented. Code has to be written for reading the alert set out of the intrusion detection system and for transforming it to a format suitable for processing or for measuring the method?s performance in order to evaluate its efficiency.
- Additionally, if a researcher wants to extend or enhance an existing method, she has to re-code all the functionality that exists in the previous implementation. While the functionality of the code will be similar, a lot of attention must be paid to the changes needed to the existing code in order to function properly in the new implementation.
- An important issue comes up when comparing methods of different authors. Their implementations vary along with the data they use or the evaluation methods they choose. This makes the comparison process problematic, as different parameters in each implementation may induce doubts on the validity of the comparison itself.

11.4 Designing the system

A platform, which will help the researchers on implementing their methods has been designed. The main ideas behind it were :

- Re-use of components implemented in previous methods
- Ready to use components for standard procedures (e.g. reading data from intrusion detection sensors)

- Included alert sets for widely used cases (e.g. alert set produced by Snort from DARPA [88], [93] datasets)
- Ready to use performance measuring components
- Ready to use visualization components

The proposed platform should enable researchers, working in the field of intrusion detection alerts post-processing, to test the methods they propose efficiently. The main concept is that they should be able to develop components with one or more alert sets as inputs and one alert set as output. They should then be able to use a graphical tool to connect these components (send a component's output to the input of another component). The components along with the connections structure in which they are connected make up a solution in the platform. In this way researchers will be able to build sophisticated methods to improve the quality of the initial alert set, with the minimum effort.

The main building blocks of the solution are the components; these are either generic or special. Generic components may be extended by the user of the platform, in order to achieve the post-processing functionality she has designed. Special components are used to achieve specific functionality and the user is responsible for setting their parameters.

The flow of alert sets, or in other words the connections between the components of the solution, along with the details of each component are stored in an XML file. The XML file contains all the required information about each of the components of the solution.

11.4.1 Abstract solution component

The user should implement the functionality she has designed. The platform provides her with all the infrastructure needed in order to start coding her logic. Every other aspect of the problem besides logic of the method such as reading data, sending data to other components, checking the validity of these data exchanges or measuring performance should be taken care of by the platform. The researcher should be focused only on implementing her methods.

The abstract solution component is a Java Class that contains all the required characteristics that a component should have.

- Minimum number of inputs
- Maximum number of inputs
- List of accepted input types
- Output type
- Void execute() method that should be overridden by the Class implemented by the user

For each of the custom components used, the user has to develop her own Java Class which will extend the abstract solution component Class. The only requirement for the custom Classes is to override the execute() method of the abstract solution Class to implement the logic of the component.

11.4.2 Special components

There are special components that are used to achieve specific tasks needed for the experiments, such as reading data from an intrusion detection system source or measuring the performance of the system.

IDSDataReader:

The IDSDataReader component is responsible for reading data (intrusion detection systems alerts) from a source and importing them into the system. This component is specific for each possible case of input. Input cases are characterized by two parameters; the IDS used and the format it keeps its data in. For example a IDSDataReader component can be developed to read data from a Snort installation that keeps data in a MYSQL database, while another would be needed to read data from a Snort installation that keeps alerts in a log file and a third one would be required to read data from a Bro IDS installation.

IDSEvaluator:

The IDSEvaluator is another special component responsible for evaluating the performance of the solution proposed. The performance of the solution can be measured in various ways, e.g. how many false positives (alerts without a corresponding event) exist in the final alert set or how many false negatives exist (events without a corresponding alert). The evaluation is performed upon data that represent the real events that have taken place, while the alerts data-set was being collected. The format in which these data are fed to the IDSEvaluator component has to be predefined. An example is the XML format used by DARPA for the real events of DARPA data-sets.

11.4.3 Connecting the components

After the researcher has implemented the required components, then the next step is to combine them, in order to produce a solution. The system stores the produced solution in an XML file. This file contains information about each component such as :

- Id of the component
- Ids of previous components (their output is connected to the input of the component)
- Ids of next components (the output of the component is connected to their inputs)
- Map containing values of configuration variables of the component

A subsection of a solution XML file that refers to a specific component is shown below :

```
<bean id="id1" class="component_class">
    <property name="previous">
        <list>
            <ref bean="id2"/>
        </list>
    </property>
    <property name="next">
        <list>
            <ref bean="id3"/>
            <ref bean="id4"/>
        </list>
    </property>
    <property name="configMap">
        <map>
            <entry key="var1" value="value1"/>
            <entry key="var2" value="value2"/>
        </map>
    </property>
</bean>
```

This entry for the component with id1 defines that its input comes from the output of the component with id2 and that its output is connected to the inputs of components with id3 and id4. This is depicted in Figure 11.1.

The functioning of the specific component is configured by the configMap property shown in the XML file that contains two configuration variables var1 and var2 along with the respective values.

11.4.4 Using the system

The proposed software contains a few initial components that implement working post-processing methods, which are enough for researchers to get the system going. The aim is to create a public library of components, to which every researcher will be able to submit her components. If this library is sufficiently populated then :

- Everyone will have a lot of ready to use components to experiment with
- Developers of well performing components, will receive the analogous recognition

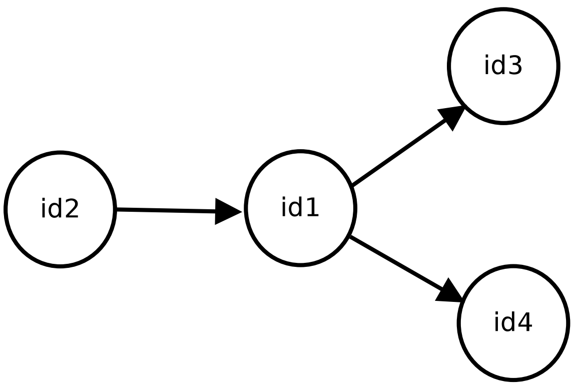

Figure 11.1: The connections between components

- Researchers will easily expand the work of others
- Comparison of the performance of different methods will be trivial

Of course every user is able to develop from scratch new components. As mentioned in Section 11.4 the user has to define the acceptable range for the number of inputs, the accepted input types and the produced output type. The logic of the users method has to be implemented in execute() method which should override an execute() method existing in the abstract component Class.

Then a graphical user interface, enables the user to create a structure of components. Through a drag and drop procedure the user can place components on the solution's canvas and then connect them in serial or parallel manner. The first component used has to be an IDataReader component, while the last should usually be an IDataEvaluator component. The structure created can be saved and loaded in the future. It can also be loaded in another installation of the platform as long as the required components exist in it. The user can export her components' Classes and import to them to another installation of the platform.

The solution (structure of components) created by the user is implemented as a directed graph of Java objects. Each component used is a node in this graph. When the user executes her solution the nodes of the graph are visited in a Breadth First Search (BFS) manner, beginning from the root of the graph. The execute() method of each component is run, while logic of BFS algorithm ensures that no component's execution is attempted, without first producing the required input.

11.5 Implementing the system

The platform has been developed in Java. In this section its main Classes are analysed. They are presented in three subsections relevant to the components of the system, the data exchanged between these components and the user interface of the system.

11.5.1 Components Classes

The main Class of the system is the AbstractSolutionComponent Class. Developers that want to create their own components must write Classes that implement this Class.

It contains all common functionality that components should have. AbstractSolution-Componnet Class main properties are :

- An id field, which is unique and representative of each component
- A set of Java Lists of Alert objects, that contain input alert sets
- A Java List of Alert objects, that contain output alert set
- Two Java Lists that contain previous and next components respectively and provide the means to create an interconnected diagram of components
- A boolean flag that shows if the component has been executed or not

The AbstractSolutioncomponent Class also has all the required methods, such as getters and setters for its fields.

Classes of custom made components inherit AbstractSolutioncomponent Class. All the standard functions (input, output, etc) that a component should contain are implemented in AbstractSolutioncomponent Class. The only task that remains to the developer is override the execute() method and embed into it the core logic of her method.

The execute() method of each component should read input alert-sets from the inputAlertSet Lists, conduct the processing it has been designed to do and then store the resulting alert-set to the outputAlertSet List. Before the execution of any other component which accepts this components output as input, the system will copy the outputAlertSet List of this component to the other components inputAlertSet Lists.

There is also an InitSolutionComponent Class, the objects of which are responsible for handling all the standard procedures for the solution. Reading data, validating solutions or evaluating results is committed by cooperating with other special Classes such as IDataReader, IdataEvaluator and InitializingComponent interface. All these Classes are depicted in Figure 11.2.

11.5.2 Data Classes

Data that flows between the components is mainly Java Lists of Alert Class objects. There is also an AlertId Class that relates alerts with their categories. Apart from that

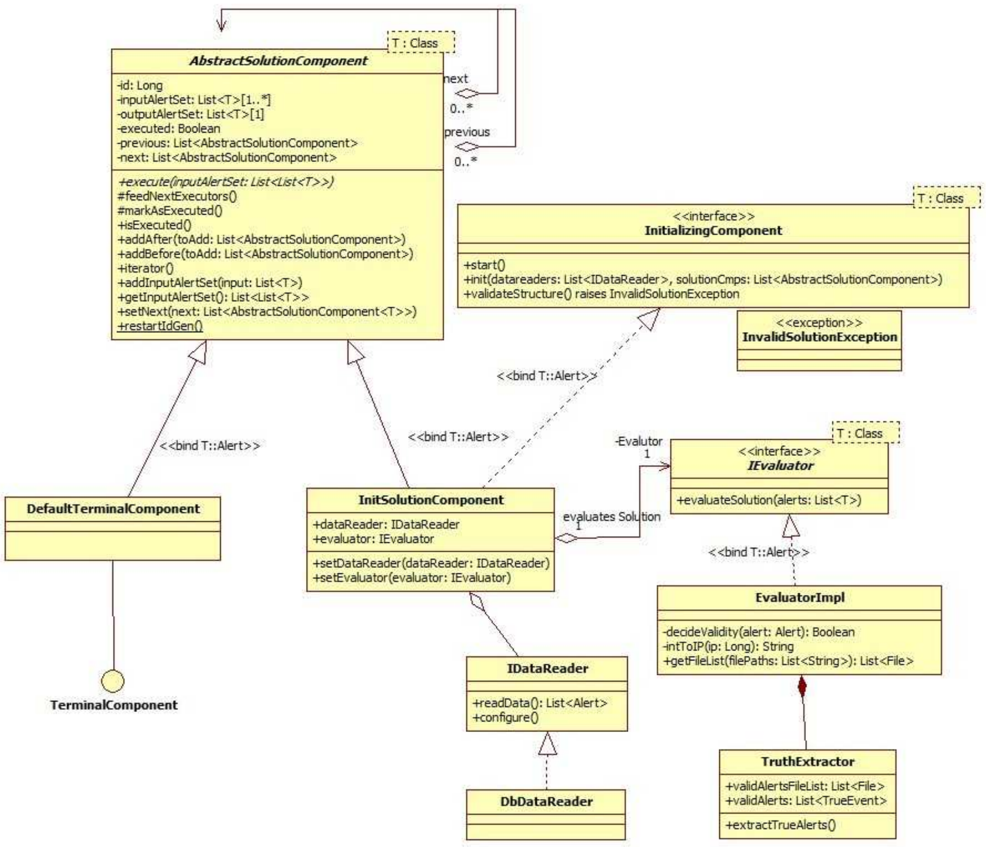
T : Class
AbstractSolutionComponent
-id: Long
-inputAlertSet: List<T>[1..*]
-outputAlertSet: List<T>[1]
-executed: Boolean
-previous: List<AbstractSolutionComponent>
-next: List<AbstractSolutionComponent>
+execute(inputAlertSet: List<List<T>>)
#feedNextExecutors()
#markAsExecuted()
+isExecuted()
+addAfter(toAdd: List<AbstractSolutionComponent>)
+addBefore(toAdd: List<AbstractSolutionComponent>)
+iterator()
+addInputAlertSet(input: List<T>)
+getInputAlertSet(): List<List<T>>
+setNext(next: List<AbstractSolutionComponent<T>>)
+restartIdGen()
next
0..*
previous
0..*
T : Class
<<interface>>
InitializingComponent
+start()
+init(datareaders: List<IDataReader>, solutionCmps: List<AbstractSolutionComponent>)
+validateStructure() raises InvalidSolutionException
<<exception>>
InvalidSolutionException
<<bind T::Alert>>
<<bind T::Alert>>
<<bind T::Alert>>
-Evalutor
1
T : Class
<<interface>>
IEvaluator
+evaluateSolution(alerts: List<T>)
DefaultTerminalComponent
InitSolutionComponent
+dataReader: IDataReader
+evaluator: IEvaluator
+setDataReader(dataReader: IDataReader)
+setEvaluator(evaluator: IEvaluator)
evaluates Solution
1
<<bind T::Alert>>
EvaluatorImpl
-decideValidity(alert: Alert): Boolean
-intToIP(ip: Long): String
+getFileList(filePaths: List<String>): List<File>
TerminalComponent
IDataReader
+readData(): List<Alert>
+configure()
DbDataReader
TruthExtractor
+validAlertsFileList: List<File>
+validAlerts: List<TrueEvent>
+extractTrueAlerts()

Figure 11.2: Components Class Diagram

another Class used is TrueEvent that holds data of real events happened and is used by the IDataEvaluator Class. Figure 11.3 shows the relevant part of the Class diagram.

The Alert Class holds data relevant to alerts such as the time-stamp, the signature of the alert, source IP, destination IP etc. There is a special field in the Alert Class, which holds a Java Map and is called properties. This can be utilized by the user, in order to enhance the basic data type (simple alert) with meta-data. For example a component can calculate a validity score for each alert. This score can be attached to the alert itself, by including it in the properties map. The next component that will accept the enhanced alert set as input will be able to read and utilize this validity score.

AlertId Class relates alerts to their Classes by defining alert id to Class id relationships. This may be used by a component that needs attack class information for its processing.

The TrueEvent Class holds data relevant to the true security events occurred. Its objects contain information such as the time stamp, the duration and source and destination IPs. This is used by the IDataEvaluator Class, in order to check if the alerts of the finally produced alert-set are valid or not. If events, relevant to an alert, exist in the List of TrueEvent objects then this alert is marked as valid, otherwise it is marked as false.

11.5.3 User interface Classes

Finally the third part of Classes of our platform enables the user to create her solution with a intuitive graphical interface.

The main Class in this part of the platform is ConfigUI. This Class holds all the information required for the graphical representation of the solution. Its main properties

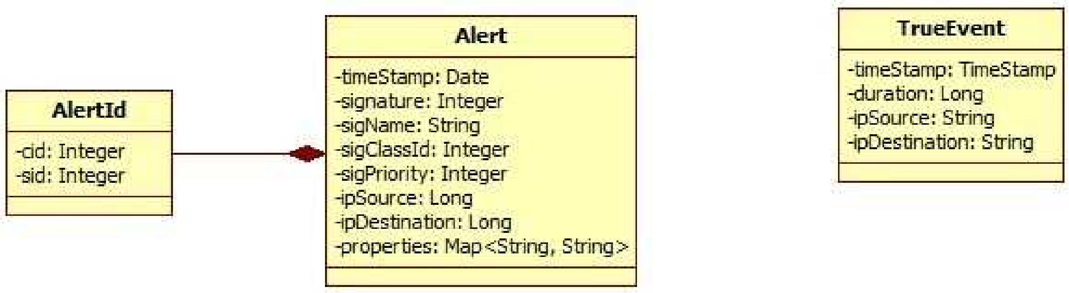

Figure 11.3: Data Class Diagram

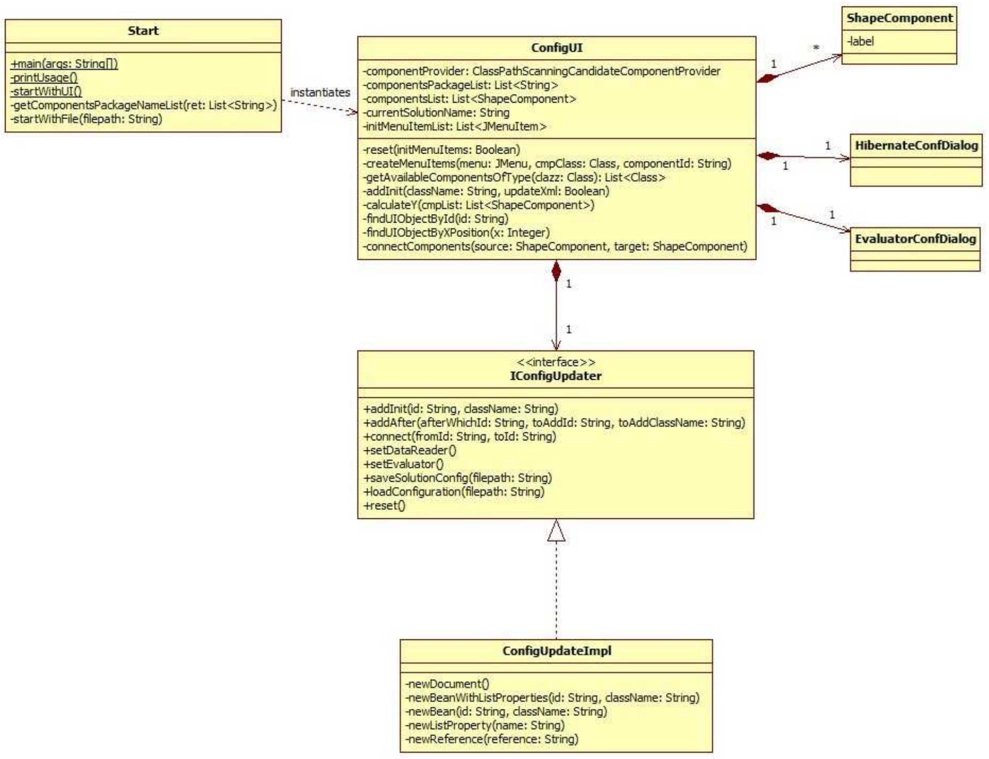
Start
+main(args: String[])
-printUsage()
-startWithUI()
-getComponentsPackageNameList(ret: List<String>)
-startWithFile(filepath: String)
instantiates
ConfigUI
-componentProvider: ClassPathScanningCandidateComponentProvider
-componentsPackageList: List<String>
-componentsList: List<ShapeComponent>
-currentSolutionName: String
-initMenuItemList: List<JMenuItem>
-reset(initMenuItems: Boolean)
-createMenuItems(menu: JMenu, cmpClass: Class, componentId: String)
-getAvailableComponentsOfType(clazz: Class): List<Class>
-addInit(className: String, updateXml: Boolean)
-calculateY(cmpList: List<ShapeComponent>)
-findUIObjectById(id: String)
-findUIObjectByXPosition(x: Integer)
-connectComponents(source: ShapeComponent, target: ShapeComponent)
ShapeComponent
-label
1
*
HibernateConfDialog
1
1
EvaluatorConfDialog
1
1
1
1
<<interface>>
IConfigUpdater
+addInit(id: String, className: String)
+addAfter(afterWhichId: String, toAddId: String, toAddClassName: String)
+connect(fromId: String, toId: String)
+setDataReader()
+setEvaluator()
+saveSolutionConfig(filepath: String)
+loadConfiguration(filepath: String)
+reset()
ConfigUpdateImpl
-newDocument()
-newBeanWithListProperties(id: String, className: String)
-newBean(id: String, className: String)
-newListProperty(name: String)
-newReference(reference: String)

Figure 11.4: The UI Class Diagram

are a list of all the components of the solution, a list of all the required packages in an installation for the solution to work and the name of the solution. All the required methods that enable the graphical user interface to function, also exist in this Class.

11.6 Testing the system

In order to test the system it have been used to re-implement a previous work in alerts post processing. The effort needed in both cases was compared and the qualitative advantages of using the proposed platform were highlighted.

The implemented work contains a post-processing filter, which reduces false positives in network-based intrusion detection systems. The filter consists of three components, the functioning of which is based upon statistical properties of the input alert set. The filter shown in Figure 11.5 was developed in Java for the purposes of the relevant experiments that justified that it is able to drastically filter out false alerts. There are three components; namely NRA, HAF and UFP. Each one produces a score for each alert, which indicates the probability of this alert to be true. Afterwards these three scores produced for each alert are combined into one final score. A threshold is finally used to identify the alerts that will be rejected.

The same filter has been developed by using the proposed system. The development has been easier because ready to use functionality has been used. Procedures such as reading alerts from Snort and measuring the performance of the filter has not been implemented, as this functionality is offered by standard components of the system. For each component of the original filter a component in the platform has been created by inheriting the AbstractSolutionComponent Class. Additionally a component responsible for the fusion of the results obtained by each component have been implemented.

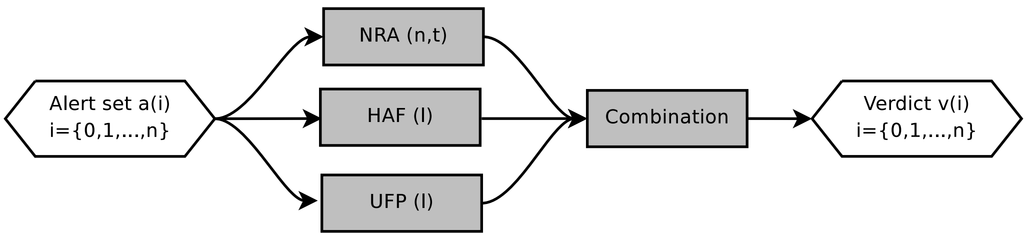

Figure 11.5: The filter

As it was expected implementing the filter with the proposed system demanded marginally less lines of code and less effort from the developer. It has been calculated that the lines of code needed to be written in the scenario of using the system were approximately 40% of the lines of code in the original implementation of the filter. This mainly happened because the original implementation was characterized by a lot of code redundancy. Parts of code, irrelevant to filters logic, appears multiple times throughout the initial implementation. This code handled standard procedural functioning such as receiving an alert-set, exporting an alert-set from a component, calculating false positives rate etc. This functioning is already implemented in the platform and the developer can focus on writing code only for the logic of her method.

Moreover each component developed is built independently of the rest of the solution. This means that it can be easily moved out of the solution, edited, replaced by another or even distributed to others. So if this method or part of it is needed to be used in a future work, the procedure of re-using it will be trivial.

CHAPTER 12

Conclusions and future work

The results of the experiments presented in Chapter 9 are discussed and conclusions on whether the proposed system fulfils the objectives, set during its design, are made. The results of the proposed system are also commented with respect to other relative methods existing in literature. Aspects of the proposed system that can be enhanced are identified and functionality that can be added to it is discussed. Additionally the systems proposed in Chapters 10 and 11 are discussed and corresponding future work opportunities are presented.

12.1 Main system

12.1.1 Conclusions

The proposed system performed efficiently on all the experiments conducted. The main conclusion is that there is a lot of room for quality improvement, regarding the information presented to the analyst about the security state of the protected system. It is obvious that lists of thousands of alerts is not the best way to timely trigger the required protection or recovery action from the analyst. The graphical representation, produced by the proposed system, is capable of informing the analyst about events in a fast and efficient manner. If the presented information is not sufficient, then she can further examine the alert-sets produced by the intrusion detection systems.

Alerts volume reduction

It has been proved that the proposed system can significantly reduce the volume of data produced by the intrusion detection sensors. As it has been analysed in Chapter 9 this reduction is concurrently achieved by:

- Aggregating multiple identical alerts produced by the same event
- Merging aggregated alerts produced for the same event by different sensors
- Clustering related aggregated alerts

The results show that population of produced clusters is an order of magnitude less than the population of alerts. For the Darpa data-set 3646 alerts produced 24 clusters and for the academic data-set 1179 alerts produced 56 clusters.

Reducing false positives

The performance of the system in detecting false positive alerts has also been important for the final result. If clusters with low truth values are ignored by the Visualization subsystem, then a sharper and clearer representation of the security state of the protected system can be produced.

The Sensor manager subsystem does not clearly classify alerts into true or false positives, but attaches a score to them that corresponds to a validity estimate value. This score is then used by subsequent components. In order to asses the system's classifying capability the average validity scores for true and false alerts can be analysed. The scores for true alerts are on average around 50% greater than scores for false alerts. It is easy to conclude that the use of an appropriate threshold value can transform the Sensor manager subsystem to an efficient classifier.

Producing informative results

One of the main motivations in designing the system was to solve the problem of different level of detail between true events and alerts they trigger. As it has been stated in previous Chapters each event produces multiple alerts identical or similar to each other. Aggregation, merging and clustering procedures embedded in the proposed system aim to gradually eliminate this disagreement between real events and the information detected in relevance to them.

As it has been previously discussed, the system produces 24 clusters for the Darpa dataset and 56 clusters for the academic dataset, both of which relate to a five steps attack plan. It is obvious that these numbers of clusters are a significant improvement with respect to the thousands of alerts in the initial alert-sets. A more thorough examination of the produced clusters list reveals that some of them are relevant to normal or non-existent events. The average number of clusters produced for each step of the intrusion steps is 1.8 clusters per step, for the Darpa data-set and 2.2 clusters per step for the academic data-set. The low average number of clusters per event is an important result as it enables the system to produce a high quality and meaningful representation at the end.

Visualizing results

The Visualisation subsystem produces an elaborate live graphical representation of the security state of the protected system. It enables the analyst to easily and instantly spot events happening on the system. The depicted level of detail is more than enough for high level inspection. If the analyst needs, she can recall lists of clusters or even alerts from the database for a more thorough examination.

The three dimensional graph enables the system to depict multiple characteristics for each cluster, namely is its time range, the relevant IP addresses range, a danger value along with a validity estimation. The production of a single surface provides the required level of simplicity, in order not to over populate the resulting graph with insignificant information.

Apart from that, the efficient generation of the graph images, enables the subsystem to produce a real-time representation of events as they happen. The low complexity of alerts' processing makes it possible to produce the clusters and subsequently the visualization frames in real time, without demanding significant hardware resources.

Hypothesizing on missed events

Another important aspect of intrusion detection systems results' quality improvement is the estimation of information about missed events. Missing on some events is inevitable for intrusion detection systems. Not all the missed events can be reconstructed, but when these are the intermediate steps of an organized attack plan, then the previous and next steps may be informative.

The tests on the hypothetical scenarios produced by the Darpa datasets have shown that the estimation of missing information is feasible to some extent. In all three cases the missing event was decently estimated. It must be noted that the produced estimates are not accurate enough to be used when taking preventive actions, but can be used to make the analyst more suspicious about possibly missed intruder's actions.

12.1.2 Comparison to other methods

The proposed system in the thesis is a complete solution, producing an elaborate result to the security analyst by combining various approaches to alerts post-processing. Its main advantage is the produced informative visualization, which is made possible by using appropriate false reduction, aggregation, clustering and hypothesizing techniques in previous subcomponents of the system. There is no other system proposed in the literature that approaches the intrusion detection alerts low quality problem, in such a global way. So a direct comparison to similar systems is not feasible, but each of the procedures of the system can be compared to similar methods existing in the literature.

Regarding reducing false positives there are many proposed methods that detect false alerts and filter them out. In general these methods are usually characterized by low false positives rate. Comparing such methods is complex as along with the percentage of false positives filtered out, the true alerts filtered in should also be examined. The Sensor manager subsystem produces a validity estimate for each alert but does not use a threshold value in order to classify alerts as true or false . Its estimation performance is depicted by the ROC curves in Figures 9.1, 9.2, 9.3 and 9.4. The estimated validity scores are significantly higher for true alerts, so the use of an appropriate threshold value would enable the Sensor manager component to efficiently classify alerts as true or false and filter out the false ones. The false positives reduction rate would be comparable to the higher corresponding percentages in the literature, while almost all true alerts would be filtered in.

The system has performed excellently in reducing the volume of the alert-set by aggregating identical alerts, merging alerts of different sensors and finally clustering relevant alerts. The methods existing in the literature also reduce the size of the alerts, but the correspondence between the final data and occurred events is usually not discussed at all. The Clustering subsystem on average created 2 clusters per event, which enables the visualization subsystem to produce elaborate graphs as visualization frames.

The Clustering subsystem also hypothesizes on missed events. There are not many relevant methods in the literature to compare its performance to. Only one of the methods presented in Chapter 3 has been tested in hypothesizing missed events and the results are comparable. The authors mainly reunite clusters parts instead of creating clusters from scratch, so the success of their method depends on the existence of some alerts for the missed event.

Furthermore the produced real time visualization is of high quality. It enables the analyst to be quickly informed about the events occurring in the protected network. In comparison to other visualization methods in the literature the produced results are more informative and easier to read. Most of other methods produce representations of high complexity in order to depict as much information as possible. The successful clustering performed by the proposed system makes simple but informative visualization feasible.

Finally an important advantage of the proposed system against most of relative methods is that it has been designed for real world implementation. The complexity of all components has been kept at minimum levels, to avoid introducing any latency to the system. Additionally, all components process only past alerts. This is fundamental for real world application of the system. Most methods proposed in the literature are tested against a previously produced alert-set and usually use this as a whole. For example a clustering method that examines the whole alert-set and produces clusters would not be applicable in a real world scenario, as the alert-set is never available as a whole but is populated gradually through time.

Generally, the proposed system is a complete approach to the low quality problem of intrusion detection alerts. It is applicable in real world scenarios, at least against the two data-sets used, it performs very well. Alternative system that have been proposed in the literature are too theoretical, do not tackle all the aspects of the problem and are not appropriate for real world application.

12.1.3 Future work

The proposed system is indicative of the possibilities that exist for intrusion detection alerts' post-processing. The obtained results are encouraging, but there is a lot of room for further improvement.

The implemented system accepts as input alert-sets produced by Snort. While Snort is broadly used, it would be interesting to enable the system to use input from other intrusion detection systems. The use of a different network-based system is feasible, without many changes. The information the system uses from Snort alerts surely exists in all network-based systems' alerts. The only issue that should be sorted out is to create a common representation for signatures [94], in order for alerts from different systems to be comparable and eligible for merging or clustering.

Another, more difficult, improvement would be to incorporate host-based systems along with the network-based ones. In this case the different nature of alerts produced by host-based systems makes unifying alerts of different types of systems a more complex task. Apart from a global representation for types of attacks, source and destination IP addresses for the host-based alert must also be extracted.

Alerts refer to activity relevant to the protected system. One way to properly process these alerts is to take into account parameters of the protected system. These parameters can improve the calculations regarding both danger and validity scores for the detected events. An enhancement to the proposed system would be to continuously monitor the protected network along with its hosts. Simple information such as which hosts are alive, what is their operating system or what services run on them is valuable in determining the validity of alerts and the likelihood of the corresponding intrusion to be effective, even if it is really happening. Alerts that are related to non alive hosts may be considered false, while alerts that are related to services which do not run on the target host of the intrusion may be considered of low danger.

The evolution of intrusion detection systems has produced intrusion prevention systems, which do not only detect intrusions but take actions to stop them or to protect the system's assets from them. An improvement to the proposed system would be to take advantage of its notable detection performance, in order to efficiently prevent the ongoing intrusions. Taking preventive actions with respect to an alert-set with high false positives rate, can make the system unusable, as each legitimate action, wrongly classified by the intrusion detection system, would be blocked along with other similar actions. By the use of appropriate agents, installed on critical hosts or network boundaries such as firewalls, the high quality information produced by the proposed system could be incorporated to prevent the actual intrusions from happening. Prevention policies should be defined and communication issues should be sorted out before extending the system in this way.

Finally the processing rate of the system could be further improved by using parallel programming techniques. In the implementation used throughout the tests, the processing of the alerts was sequential. When a new alert was imported into the system it was propagated to the Clusterer component, which run periodically to create clusters. The system could not process the incoming alert until the previous one had reached the Clusterer component. One obvious improvement would be to implement components to be executed in parallel. In this way, when a component processes an alert and sends it to the next components, it is ready to accept another alert. A more advanced implementation would be to initiate a new execution thread for each different alert, but in this case significant issues would be raised with regards to defining the neighbouring alerts for each alert and calculating various metrics for it. If these issues are solved, then the performance of the system can be significantly increased.

In general the system can be improved in various ways, in order to take advantage of its results. A lot of attention should be given to not over-increase complexity, as the real-time application of the system in real world networks is one of its most important advantages.

12.2 Using Fuzzy Inference Systems to reduce false positives

The fuzzy inference system methodology has been used to create a system able to filter out most of the false alerts, while filtering in alerts for all detected attacks in Chapter 10. The core logic of the system comes from statistical observations about the distribution of true and false alerts. Fuzzy logic seems to serve well the need for relative assumptions about the nature of alerts, and maximizes the quality of the results obtained.

The system can be further improved by adjusting the parameters in a more efficient way or by importing additional parameters. Apart from that, there are training schemes for fuzzy inference systems, which function by trying to minimize the false positives rate. If these schemes are enhanced to also maximize the percentage of attacks for which alerts are filtered in instead of only minimizing the FPR, and over-fitting issues are carefully treated, then the resulting system may perform even better.

12.3 Test-bend for intrusion detection systems results post-processing

It is generally accepted that post-processing of alerts is a significant area of intrusion detection research. All the authors proposing a relevant method have to put a lot of effort on implementation, in order to prove the validity of their method. The implementation part is always difficult and time consuming. There are no tools, that can help researchers on this problem, so they have to manually code everything.

The platform, presented in Chapter 11, fills this gap. It offers researchers ready to use functionality, the ability to reuse theirs or others older functionality and a standard evaluation environment that enables reliable comparisons between different methods. The tests committed have demonstrated that using the platform makes the implementation easier and less time-consuming.

The platform is in its initial steps and future work can add value to its use from researchers. A lot of attention has to be given on making users publish their components. The true power of the platform is that it enables easy reuse of previous methods. The researcher has to just import others' components to her installation to make use of them, so testing or extending others' work is very easy. So if researching community made their components available for public use, then the community itself would benefit from an important repository of ready to use components.

Apart from that, obliging users to write in Java is a limitation that should be vanished. If a researcher has already implemented her methods in another programming language (C or Matlab), she will not re-implement it in Java just to make it public to others. A generic component should be implemented, that will be able to communicate with external software (functionality implemented in other languages) and use it in terms of the solution designed in the platform.

Finally, in order to accommodate functionality, that demands excessive processing power, a second abstract component that will be designed in parallel programming approach, should be implemented. A relevant hardware platform should be chosen and a component that will execute different parts of its execute() method on different processors available should be created. In this way the user that has in her disposal the required hardware, will be able to exploit it in order to implement and test a complicated, processing-power demanding method.

Candidate’s publications

Journal publications:

[J1] Spathoulas, G.P., Katsikas, S.K., Reducing false positives in intrusion detection systems (2010) Computers & Security, 29 (1), pp. 35-44.

[J2] Georgios P. Spathoulas, Sokratis K. Katsikas, Enhancing IDS performance through comprehensive alert post-processing, Computers & Security, Volume 37, September 2013, Pages 176-196, ISSN 0167-4048

[J3] Georgios P. Spathoulas, Sokratis K. Katsikas, Methods for post-processing alerts in intrusion detection : A survey, International Journal of Information Security Science, 2(2), 64-80.

Conference publications:

[C1] Spathoulas, G.P., Katsikas, S.K., Using a fuzzy inference system to reduce false positives in intrusion detection (2009), 2009 16th International Conference on Systems, Signals and Image Processing, IWSSIP 2009

[C2] Georgios P. Spathoulas, Sokratis K. Katsikas, A test-bend for intrusion detection systems results post-processing, accepted in Euro PKI 2013,

Bibliography

[1] Internet world stats. History and growth of the internet from 1995 till today. `http://www.internetworldstats.com/emarketing.htm`, 2013. Online; accessed 10-May-2013.

[2] ITU. International telecommunication union. `http://www.itu.int/en/about/Pages/default.aspx`, 2013. Online; accessed 10-May-2013.

[3] Computer industry almanac inc. `http://www.c-i-a.com/index.htm`, 2013. Online; accessed 10-May-2013.

[4] James Joshi. *Network Security: Know It All*. Elsevier Inc, 2008.

[5] Information security week. 10 massive security breaches. `http://www.informationweek.com/security/attacks/10-massive-security-breaches/229300675?pgno=2`, 2011. Online; accessed 10-May-2013.

[6] CSO. The 15 worst data security breaches of the 21st century. `http://www.csoonline.com/article/700263/the-15-worst-data-security-breaches-of-the-21st-century`, 2012. Online; accessed 10-May-2013.

[7] Symform. Five worst cyber security breaches of 2012. `http://www.symform.com/blog/5-worst-security-breaches-2012/`, 2013. Online; accessed 10-May-2013.

[8] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems. Technical report, NIST, September 2006.

[9] James P. Anderson. Computer security threat monitoring and surveillance. *Fort Washington, PA*, 1980.

[10] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, 2000.

[11] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Comput. Netw.*, 31(9):805–822, April 1999.

[12] Rodrigo Werlinger, Kirstie Hawkey, Kasia Muldner, Pooya Jaferian, and Konstantin Beznosov. The challenges of using an intrusion detection system: is it worth the effort? In *Proceedings of the 4th symposium on Usable privacy and security*, SOUPS '08, pages 107–118, New York, NY, USA, 2008. ACM.

[13] R. Pietro and L.V. Mancini. *Intrusion Detection Systems*. Advances in Information Security. Springer, 2008.

[14] Dingbang Xu. *Correlation analysis of intrusion alerts*. PhD thesis, 2006. AAI3223224.

[15] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection (RAID 2001)*, number 2212 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[16] Hanli Ren, Natalia Stakhanova, and AliA. Ghorbani. An online adaptive approach to alert correlation. In Christian Kreibich and Marko Jahnke, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6201 of *Lecture Notes in Computer Science*, pages 153–172. Springer Berlin Heidelberg, 2010.

[17] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, November 2003.

[18] Soojin Lee, Byungchun Chung, Heeyoul Kim, Yunho Lee, Chanil Park, and Hyunsoo Yoon. Real-time analysis of intrusion detection alerts via correlation. *Computers & Security*, 25(3):169 – 183, 2006.

[19] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, RAID '00, pages 85–103, 2001.

[20] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 245–254. ACM, 2002.

[21] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 370–379, 2004.

[22] Shanchieh J. Yang, Adam Stotz, Jared Holsopple, Moises Sudit, and Michael Kuhl. High level information fusion for tracking and projection of multistage cyber attacks. *Information Fusion*, 10(1):107 – 121, 2009.

[23] Peng Liu, Wanyu Zang, and Meng Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Trans. Inf. Syst. Secur.*, 8(1):78–118, February 2005.

[24] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1:146–169, 2004.

[25] Joshua Ojo Nehinbe. Automated method for reducing false positives. In *Intelligent Systems, Modelling and Simulation (ISMS), 2010 International Conference on*, pages 54–59, January 2010.

[26] Joshua Ojo Nehinbe. Concurrent reduction of false positives and redundant alerts. In *Information Society (i-Society), 2010 International Conference on*, pages 318–323, June 2010.

[27] GJ Victor, MS Rao, and VCH Venkaiah. A bayesian classification on asset vulnerability for real time reduction of false positives in ids. *International Journal of Network Security and Its Applications (IJNSA)*, 4(2):63–73, March 2012.

[28] Heng-Sheng Lin, Hsing-Kuo Pao, Ching-Hao Mao, Hahn-Ming Lee, Tsuhan Chen, and Yuh-Jye Lee. Adaptive alarm filtering by causal correlation consideration in intrusion detection. In *New Advances in Intelligent Decision Technologies*, volume 199 of *Studies in Computational Intelligence*, pages 437–447. Springer Berlin Heidelberg, 2009.

[29] A. Thomas. Rapid: Reputation based approach for improving intrusion detection effectiveness. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 118 –124, August 2010.

[30] Yuxin Meng and Lam-for Kwok. Adaptive false alarm filter using machine learning in intrusion detection. In *Practical Applications of Intelligent Systems*, volume 124 of *Advances in Intelligent and Soft Computing*, pages 573–584. Springer Berlin Heidelberg, 2012.

[31] S. Khanchi and F. Adibnia. False alert reduction on network-based intrusion detection systems by means of feature frequencies. In *Advances in Computing, Control, Telecommunication Technologies, 2009. ACT '09. International Conference on*, pages 513 –516, December 2009.

[32] J.J. Treinen and R. Thurimella. Finding the needle: Suppression of false alarms in large intrusion detection data sets. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 2, pages 237 –244, August 2009.

[33] Safaa O. Al-Mamory and Hongli Zhang. Intrusion detection alarms reduction using root cause analysis and clustering. *Computer Communications*, 32(2):419 – 430, 2009.

[34] T Alapaholuoma and J Nieminen. A behavior-based method for rationalizing the amount of ids alert data. *ICCGI 2012, The Seventh International Multi-Conference on Computing in the Global Information Technology*, June 2012.

[35] Sangkyum Kim, Winnie Cheng, Shang Guo, Laura Luan, Daniela Rosu, and Abhijit Bose. Polygraph: system for dynamic reduction of false alerts in large-scale it service delivery environments. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11. USENIX Association, 2011.

[36] Federico Maggi, Matteo Matteucci, and Stefano Zanero. Reducing false positives in anomaly detectors through fuzzy alert aggregation. *Inf. Fusion*, 10(4):300–311, October 2009.

[37] Neminath Hubballi, Santosh Biswas, and Sukumar Nandi. Network specific false

alarm reduction in intrusion detection system. *Security and Communication Networks*, 4(11):1339–1349, November 2011.

[38] A.B. Mohamed, N.B. Idris, and B. Shanmugum. Alert correlation using a novel clustering approach. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 720 –725, May 2012.

[39] Dapeng Man, Wu Yang, Wei Wang, and Shichang Xuan. An alert aggregation algorithm based on iterative self-organization. *Procedia Engineering*, 29(0):3033 – 3038, 2012.

[40] A.Charan Kumar, A. Vivekanand, K. Kavitha, M. Gracevennice, and T.Bharath Manohar. Data stream intrusion alert aggregation for generative data stream modelling. *International Journal of Advanced Research in Computer Engineering and Technology(IJARCET)*, 1(7), 2012.

[41] Gina C. Tjhai, Steven M. Furnell, Maria Papadaki, and Nathan L. Clarke. A preliminary two-stage alarm correlation and filtering system using som neural network and k-means algorithm. *Computers and Security*, 29(6):712 – 723, 2010.

[42] R. Vaarandi and K. Podins. Network ids alert classification with frequent itemset mining and data clustering. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 451 –456, October 2010.

[43] C.J. Fung, Quanyan Zhu, R. Boutaba, and T. Basar. Bayesian decision aggregation in collaborative intrusion detection networks. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 349 –356, April 2010.

[44] Massimo Ficco and Luigi Romano. A correlation approach to intrusion detection. In *Mobile Lightweight Wireless Systems*, volume 45 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 203–215. Springer Berlin Heidelberg, 2010.

[45] Benjamin Morin, Ludovic Me, Herve Debar, and Mireille Ducasse. A logic-based model to support alert correlation in intrusion detection. *Information Fusion*, 10(4):285 – 299, 2009.

[46] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. Decentralized multi-dimensional alert correlation for collaborative intrusion detection. *Journal of Network and Computer Applications*, 32(5):1106 – 1123, 2009.

[47] Zoltan Czirkos, Marta Rencz, and Gabor Hosszu. Improving attack aggregation methods using distributed hash tables. In *ICIMP 2012, The Seventh International Conference on Internet Monitoring and Protection*, pages 82–87, May 2012.

[48] C. Thomas and N. Balakrishnan. Improvement in intrusion detection with advances in sensor fusion. *Information Forensics and Security, IEEE Transactions on*, 4(3):542 –551, September 2009.

[49] Salem Benferhat, Abdelhamid Boudjelida, Karim Tabia, and Habiba Drias. An intrusion detection and alert correlation approach based on revising probabilistic classifiers using expert knowledge. *Applied Intelligence*, pages 1–21, 2012.

[50] Elias Raftopoulos and Xenofontas Dimitropoulos. Detecting, validating and characterizing computer infections in the wild. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 29–44. ACM, 2011.

[51] Jouni Viinikka, Herve Debar, Ludovic Me, Anssi Lehikoinen, and Mika Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10(4):312 – 324, 2009. <ce:title>Special Issue on Information Fusion in Computer Security</ce:title>.

[52] Ayman E. Taha, Ismail Abdel Ghaffar, Ayman M. Bahaa Eldin, and Hani M. K. Mahdi. Agent based correlation model for intrusion detection alerts. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on*, pages 89 –94, May 2010.

[53] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):591–627, 2004.

[54] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communica-*

tions, 29(15):2917 – 2933, 2006. <ce:title>Computer Communications</ce:title> <xocs:full-name>Internet Communications Security</xocs:full-name>.

[55] John R. Goodall Jason Laska Bogdan Denny Czejdo, Erik M. Ferragut. Network intrusion detection and visualization using aggregations in a cyber security data warehouse. *International Journal of Communications, Network and System Sciences*, 5(9):593–602, September 2012.

[56] Urko Zurutuza, Enaitz Ezpeleta, Alvaro Herrero, and Emilio Corchado. Visualization of misuse-based intrusion detection: Application to honeynet data. In *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*, volume 87 of *Advances in Intelligent and Soft Computing*, pages 561–570. Springer Berlin Heidelberg, 2011.

[57] Florian Mansmann, Fabian Fischer, Daniel A. Keim, and Stephen C. North. Visual support for analyzing network traffic and intrusion detection events using treemap and graph representations. In *Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology*, CHiMiT '09, pages 3:19–3:28. ACM, 2009.

[58] Nurbol, Huan Xu, Hao Yang, Fan-Er Meng, and Liang Hu. A real-time intrusion detection security visualization framework based on planner-scheduler. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*, pages 784 –788, December 2009.

[59] M. Dumas, J.-M. Robert, and M.J. McGuffin. Alertwheel: radial bipartite graph visualization applied to intrusion detection system alerts. *Network, IEEE*, 26(6):12 –18, November-December 2012.

[60] MIT. MIT Lincoln Laboratory, 2000 darpa intrusion detection scenario specific data sets, 2000. Online; accessed 04-June-2012.

[61] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. *System Administration Conference*, 1999.

[62] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann,

and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[63] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[64] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000.

[65] Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, pages 220–237. Springer-Verlag, 2003.

[66] S Terry and Brugger Jedadiah Chow. An assessment of the darpa ids evaluation dataset using snort, 2005.

[67] Rapid7Community. The metasploitable vm. `http://www.offensive-security.com/metasploit-unleashed/Metasploitable`, 2010. Online; accessed 04-June-2012.

[68] VMWare. Ultimate lamp vm. `http://www.vmware.com/appliances/directory/189`, 2006. Online; accessed 10-September-2011.

[69] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[70] E. Raftopoulos and X. Dimitropoulos. Detecting, validating and characterizing computer infections in the wild. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pages 29–44, 2011.

[71] Q. Qia and Z. Wang. A new attack detection in large scale network based on entropy. *Journal of Networks*, 7(5):863–868, 2012.

[72] Z. Li, Y. Chen, and A. Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense, LSAD'06*, volume 2006, pages 115–122, 2006.

[73] DShield. Dshield database. `http://www.dshield.org`, 2012. Online; accessed 20-November-2012.

[74] T. Alapaholuoma, J. Nieminen, J. Ylinen, T. Seppala, and P Loula. A behavior-based method for rationalizing the amount of ids alert data. In *Proceedings of the Seventh International Multi-Conference on Computing in the Global Information Technology, ICCGI 2012*, page 302 to 307, 2012.

[75] E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1 – 13, 1975.

[76] W. El-Hajj, F. Aloul, Z. Trabelsi, and Nazar Zaki. On detecting port scanning using fuzzy based intrusion detection system. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC '08. International*, pages 105–110, 2008.

[77] A. El-Semary, J. Edmonds, J. Gonzalez, and M. Papa. A framework for hybrid fuzzy logic intrusion detection systems. In *Fuzzy Systems, 2005. FUZZ '05. The 14th IEEE International Conference on*, pages 325–330, 2005.

[78] Adel Nadjaran Toosi and Mohsen Kahani. A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Comput. Commun.*, 30(10):2201–2212, July 2007.

[79] K.S. Kumar and V. Nandamohan. Novel anomaly intrusion detection using neuro-fuzzy inference system. *International Journal of Computer Science and Network Security*, 8:6–11, 2008.

[80] J.H. Graham and Y. Yu. Computer system security threat evaluation based upon artificial immunity model and fuzzy logic. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1297–1302, 2005.

[81] MuhammadFermi Pasha, Rahmat Budiarto, Mohammad Syukur, and Masashi Yamada. Efis: Evolvable-neural-based fuzzy inference system and its application for adaptive network anomaly detection. In DanielS. Yeung, Zhi-Qiang Liu, Xi-Zhao Wang, and Hong Yan, editors, *Advances in Machine Learning and Cybernetics*, volume 3930 of *Lecture Notes in Computer Science*, pages 662–671. Springer Berlin Heidelberg, 2006.

[82] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptive neuro-fuzzy intrusion detection systems. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, volume 1, pages 70–74 Vol.1, 2004.

[83] Dwen-Ren Tsai, Wen-Pin Tai, and Chi-Fang Chang. A hybrid intelligent intrusion detection system to recognize novel attacks. In *Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on*, pages 428–434, 2003.

[84] Yongzhong Li, Yang Ge, Xu Jing, and Zhao Bo. A new intrusion detection method based on fuzzy hmm. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 36–39, 2008.

[85] J.E. Dickerson, J. Juslin, O. Koukousoula, and J.A. Dickerson. Fuzzy intrusion detection. In *IFSA World Congress and 20th North American Fuzzy Information Processing Society (NAFIPS) International Conference, Vancouver, British Columbia*, pages 1506–1510, 2001.

[86] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. *J. Netw. Comput. Appl.*, 32(6):1219–1228, November 2009.

[87] Riyad Alshammari, Sumalee Sonamthiang, Mohsen Teimouri, and Denis Riordan. Using neuro-fuzzy approach to reduce false positive alerts. In *Proceedings of the Fifth Annual Conference on Communication Networks and Services Research*, CNSR '07, pages 345–349, Washington, DC, USA, 2007. IEEE Computer Society.

[88] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Comput. Netw.*, 34(4):579–595, October 2000.

[89] N.J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R.A. Olsson. A methodology for testing intrusion detection systems. *Software Engineering, IEEE Transactions on*, 22(10):719 –729, oct 1996.

[90] N. Puketza, M. Chung, R.A. Olsson, and B. Mukherjee. A software platform for testing intrusion detection systems. *Software, IEEE*, 14(5):43 –51, sep/oct 1997.

[91] Tao Wan and Xue Dong Yang. Intrudetector: a software platform for testing network intrusion detection algorithms. In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pages 3 – 11, dec. 2001.

[92] D. Mutz, G. Vigna, and R. Kemmerer. An experience developing an ids stimulator for the black-box testing of network intrusion detection systems. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 374 – 383, dec. 2003.

[93] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 12–26 vol.2.

[94] Herve Debar, David A Curry, and Benjamin S Feinstein. The intrusion detection message exchange format (idmef). 2007.