Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Response-aware Event Aggregation for Policy Enhancement

Bachelor's Thesis in Computer Science

durchgeführt am
Lehrstuhl fur Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Charis - Nicolas Georgiou

15 April 2014

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Reaktionsgesteuerte Ereignisaggregation zur Verbesserung des Policy-Verhaltens

Response-aware Event Aggregation for Policy Enhancement

Bachelorarbeit in Informatik

durchgeführt am
Lehrstuhl fur Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Charis - Nicolas Georgiou

Aufgabensteller:	Prof. Dr.-Ing. Georg Carle
Betreuer:	Dipl.-Inform. Stephan Posselt, Nadine Herold M.Sc.
Tag der Abgabe:	15. April 2014

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this thesis only supported by declared resources.

Garching, den 15. April 2014

Acknowledgements

This thesis has benefited from the great support received by many people, some of whom I would sincerely like to thank here.

First of all, I would like to thank *Prof. Dr.-Ing. Georg Carle* for the chance to write my thesis at the chair for Network Architectures and Services.

I would like to state my deepest gratitude to my advisors at the chair *Nadine Herold* and *Stephan Posselt* for their continuing support that was given to me by them, the high amount of patience they showed and their constructive criticism which helped shape this thesis.

Additionally I would like to thank my friends and family for the support and courage they have given me the past three years. A special thanks goes to my mother *Heike* and my father *George*, for their continuing moral and financial support in my studies and for giving me the chance to study at the TUM through their personal sacrifices throughout the years. Without them none of this would have ever been possible.

Finally I want to dedicate this thesis to my grandmothers *Nikki* and *Margot* and my grandfathers *Chambis* and *Adolf*.

Abstract:

Nowadays, city infrastructure relies increasingly on interconnected high-speed networks, with most of the critical infrastructures today being in part connected to the internet. This combined with the evolution of malware to more sophisticated levels, raises the need to implement more elaborate security mechanism. Traditional defence mechanism such as firewalls, are not enough to circumvent the threats that may arise in today's networks. To counter this,an increasing number of network administrators deploy Intrusion Detection Systems, which monitor the network and raise alerts if anything suspicious is detected. The traditional approach to defending the system in the case of an Intrusion Detection System, requires a human administrator to implement defensive actions. Since this is sometimes inefficient, for example because of time-delays between when a threat is detected and a measure is implemented, research has been more and more focused on automatizing this process. By automatizing this process, instead of only informing the administrator, or the human interface of the security chain, an entity receives the generated alerts and implements defensive actions based on pre-determined set of actions.

This bachelor thesis is detailing a sub module for an automated reaction system, named ANSII. One of the core problems that is solved in this thesis, huge amount of duplicate alerts that is generated, when a high enough number of IDSs is deployed. The reduction of the duplicates, is an essential part of the Alert Correlation Process, and is called Alert Aggregation. Additionally, this thesis diversifies itself by considering the benefits of including feedback from the enforcement mechanics in the network, and by trying to keep the rule specification for counteractions simple by including dynamic generation of aggregation rules. The inclusion of feedback is important, since it should help gain new insights about the state of the detected threats in the system. The approach detailed in this thesis, aims to improve the overall efficacy and performance of the Policy Component of the ANSII system.

Kurzfaßung:

Moderne städtische Infrastruktur verlaßt sich zunehmend auf hoch-geschwindigkeits und zusammen-geschaltete Netze. Eine große Anzahl der kritischen Infrastruktur heutzutage ist verbunden mit dem Internet, was zusammen mit der Evolution von Malware die Notwendigkeit erschafft st´arkere sicherheits Mechanismen zu implementieren. Traditionele sicherheit, wie eine Firewall, genugt heutzutage nicht um Angriffe zu blocken oder aufzuheben. System Administratoren, benutzen immer mehr Intrusion Detection Systems in ihren Netzwerken, um die Netzwerke zu ïberwachen. Intrusion Detection Systems, generieren Alarme, die Alerts genannt werden, wenn verdächtige Elemente im Netzwerk wahrgenommen werden. Der traditionelle Ansatz bei IDSen ist, das der Administrator die Alerts erhält und defensive Maßnahmen einsetzt. Dieser Ansatz ist manchmal ineffizient, da die Verzögerung zwischen wenn eine Attacke erkannt wurde, und die defensiven Maßnahmen eingesetzt werden, kritisch sein kann. Dies führte zur Forschung für automatisiertes Reaktion, wo die generierten Alerts nicht nur zum Administrator geschickt werden, sonder zu einer Instanz die f´ur die generierten Alerts defensive Ma
ssnahmen ergreift basiert auf vorgegebenen Aktionen.

Die Bachelor Arbeit die hier präsentiert wird, handelt über ein Sub Modul für ein automatisiertes Reaktion System, mit dem namens ANSII. Eines der grundlegenden Probleme das die Arbeit bearbeitet ist, die große Anzahl an Alert Duplikaten die generiert wird, wenn es eine große Anzahl von IDSen im Netzwerk gibt. Die Eliminierung von Duplikaten ist teil eines Prozeß der Alert Correlation heißt, und heißt Alert Aggregation. Zusätzlich zu diesem Problem, diversifiziert sich die Arbeit von anderen Arbeiten, indem der Feedback, der von denn Policy Enforcement Point generiert wird, auch mit einbezogen wird. Dies ist wichtig, denn der Feedback kann helfen um neue Einblicke über denn Status von erkannten Bedrohungen. Das Ziel der Bachelor-Arbeit ist, die allgemeine Effizienz und Performanz der Policy Komponente zu verbessern.

Contents

List of Figures

Listings

1. Introduction

1.1 Modern Networks and Network Security

Present-day communication is characterized by high-speed networks interconnected with each other. High-speed communication networks play an essential role in our everyday activities. A steadily increasing amount of services are offered over the Internet, and new services emerge, which are served exclusively through the internet and computer networks. Critical infrastructure like the electrical grids are increasingly becoming interconnected. Since society relies on the flawless an uninterrupted operation of these infrastructures and communication networks, security in networked environments becomes an important aspect[CBCAD09, PoNe97, GoHK03].

One of the reasons that security needs to evolve is the continuous evolution of computer and network security malware. There is malware that can propagate through networks, like worms. They can carry payloads in the form of viruses which can render systems unusable or contain spyware which enables an attacker to spy on his target. The sophistication of the malware itself has increased. The case of advanced cyberwarfare malware like Stuxnet proves that security hast to be able to adapt, be more autonomous and flexible [FGHW^{+}08, PoNe97, RCHP00].

In addition to the ever increasing sophistication and ability of malware, the internet enables the world access from everywhere. Companies increasingly use XaaS(Everything as a Service) to enable a distributed working environment. Although it being a successful way to work from home, and keep costs low it comes with some risks of itself. The greatest risk which this model bears is that the data itself and the services in use are attackable from anywhere in the world. Since everything is accessible from the internet, the internet itself can be utilized as a tool for attacking systems from anywhere with precision. This increases the need to ensure that customer data and critical infrastructures have adequate actions of protection [GoHK03].

To counter these increasing amounts of security threats, and to safe keep the important parts of networks, system administrators cannot rely solely on the firewall to take care of the defences. A firewall can fall short in blocking more elaborate attacks, since the traffic blocked,forwarded is only known a-priori. For this reason, the utilisation of Intrusion Detection Systems in networked environments is increasingly becoming common-ground [Cupp01].

Intrusion Detection Systems try to detect intrusions in a system or network. Their deployment is essential since they enable monitoring of network components, which is required to ensure an operational state in a network. Monitoring is essential in order to detect attacks which try to compromise the system, and steal data. Monitoring also provides a way to see how an attack evolves in a network [Daya].

The field of Intrusion Detection Systems has been actively researched for over 20 years. Intrusion Detection Systems have become quite adequate in detecting active threats in a network. The increasing importance of distributed systems, has led also to a more active deployment of detection mechanisms [Cupp01]. IDS deployment although necessary does not completely solve the problem and adds a number of problems discussed in Section 1.1.1 .

Traditionally Intrusion Detection Systems are only monitoring a network and notify the administrator. An increasing trend is the automation of this process, the reasons for which will be discussed in Section 1.1.2.

Automated reaction systems, work by monitoring the network through the IDSs but instead of only forwarding the alerts to the system administrator, they contain a dedicated Decision Point which processes the alerts generated by the Intrusion Detection Systems, in order to decide which action is to be implemented.

1.1.1 Problems

The security components used in networks, although inclusive for themselves are not enough. First the structure and placement of these components has become increasingly complex. IDS systems are scattered through the network and independently raise alarms, even if they are overlapping leading to the generation of duplicate alerts [Daya, FGHW$^+$08, HoSi11, Cupp01].

When an attack is launched, a single attack instance which is the occurrence of a single attack type that has been launch in a specific time window from a specific attacker, is distributed throughout many log files and networks. The single attack instance itself might also generate hundreds or thousands of alerts [HoSi11, ZaYZ08].

This is for a traditional network, were the system administrator has to inspect the alerts himself, very complex and generate a relatively high probability for him to decide wrongly for single alerts [HoSi11, AMZh07, ZaYZ08].

Another common problem which can be observed, is that since the systems are tailored for human interaction, the security devices which implement the defensive actions do not provide any form of feedback. In our opinion, feedback would provide the system with additional insight about active attacks, allowing it to adapt its security policies in order to accommodate the changes in the networks environment, for example when a threat cannot be eliminated with the traditional arsenal of available actions.

1.1.2 Reasons for Automation

Section 1.1.1, provided an overview to some of the problems that arise when an increasing number of Intrusion Detection Systems is deployed into a network, through the use of the traditional approach to monitoring and reacting.

One of the problems identified, was the human-interaction tailored approach of network security. Human interaction introduce attack vectors which can be exploited. One of the most common one is the time-delay between when an attack is detected, and when the human security expert will implement defensive actions. Time-delay and human-error create an environment where an automated response would be better suited citecuppensManagement, alertSurvey.

To further understand this reasoning lets look at an example. Lets assume a network is under an active attack, the administrator gets flooded with alerts that the network is getting attacked from all sides. He decides to filter everything except serious exploits, but in turn leaves the network open to a variety of smaller, seemingly unimportant exploits and attacks. These can subsequently be chained together to access crucial business information [Cupp01].

To enable automation, new systems have been created which are called Intrusion Response Systems(IRS). Intrusion Response Systems do not only monitor a network, but also automatically implement defensive actions when they detect any malicious or suspicious activity in the network they are protecting. A variety of systems exist, which will be explored in detail in Section 3.

Automation although helpful is not risk free. Automation is only powerful enough to take control over human actions. This means that a human has programmed it to act assuming normal situations. Automation in all areas has a major caveat, when faced with abnormalities the automation has a limited span of action, and may get confused. In addition automation does not provide adequate feedback as humans would do. All in all though because using automation enables us to act faster, it is still relatively an important feature to have [Norm90].

1.2 Thesis Motivation

Our main focus is to research some of the problems that occur when many IDS systems are contained in one network. One of the problems that occurs is that the network gets flooded with alerts. One of the reasons for this is due to the overlapping domains of many heterogeneous Intrusion Detection Systems. Another aspect being considered is the nonexistent overview of what actions the different security components implement in order to protect the devices contained in a network, when they are under attack.

1.3 Contributions

In this thesis two sub problems of the communication between many Intrusion Detection Systems and the Policy Decision Point will be explored. Particularly the options to deal with the flooding of the network with alerts from the different heterogeneous Intrusion Detection Systems will be explored in depth. In the process the sub problem of providing feedback about the actions the Enforcement Point implements to avert attack instances.

The solution that is provided here is based on alert aggregation, and our main focus will be how to aggregate the alerts in order to reduce the amount of alerts reaching the Policy Decision Point and how eliminate the duplicates that exist in that alert stream. In addition the module that will be developed here, addresses the problem of the nonexistence of feedback from the defensive actions implemented by the security components in the network by receiving it and appending it to the alerts, in order to gain new insights about attacks performed on the network.

Furthermore it will be investigated how to utilize the aggregation in order to append additional information for decision making towards the PDP. This will be found in the form of meta data which are composed of various aggregate information which can be derived by the alert flow and a list of actions performed in the network for each alert. The final target is to improve the overall efficacy of the PDP in general by eliminating and in the process of eliminating the duplicates ensure that no information will get lost.

1.4 Chapter Overview

Chapter 2: Analysis gives a brief overview over the research field, called alert correlation process and which part of the process the thesis is located. It introduces the network environment and the components that are to be investigated, and identifies the problems which are to be solved through this thesis. *Chapter 3: Related Work*, provides an overview of related work, and explains how this thesis distances itself from the related approaches.

Chapter 4:Design introduces the solution to the problems and requirements identified in Chapter 2. It provides a detailed explanation of the different functionalities that the solution contains and how these solve the problems previously identified.

Chapter 5: Implementation provides an overview of the frameworks that will be used in the implementation, and maps each framework to the functionalities defined in Chapter 4.

Chapter 6: Evaluation retouches the requirements identified, the solutions provided and gives an overview over how the solution should be evaluated in order to verify its correct functionality.

Chapter 7: Conclusion and Future Outlook concludes this thesis by providing conclusions that can be derived by the solution, and suggests future work that can be done to further improve automated reaction systems.

2. Analysis

This chapter will introduce the problem domain in which the thesis takes place. It provides an overview of the ANSII Project to which this thesis is part of. It gives a detailed explanation of the environment which this thesis handles about, and explains the correlation process and how this thesis is related to it. Through the environment explanation, the problems will be identified and then posed as requirements to which a solution has to be based upon.

2.1 ANomaly identification and embedded Security in Industrial Information systems (ANSII) and Thesis Subject

ANomaly identification and embedded Security in Industrial Information systems (ANSII) is a research project which aims to create a model, which integrates IT Security solutions in industrial information systems. This model includes the selection of the assets that need to be protected, the threat analysis, risk assessment, and a selection of actions and implementations. Application specific models and solution cores should be developed within the field of embedded systems, aiming at industrial information systems [ANSI].

A part off the ANSII project called policy focuses on automating network security detection and response. The ANSII Policy engine can also analyse attack instances in a network in order to provide more insight about how to deal with specific attack instances.

This thesis details a sub module, which is part of ANSII's policy engine. The module should analyses the alerts generated by the Intrusion Detection Systems in the network. The analysis should provide information if an attack instance in the network is either ongoing, terminated or the security components in place have still not responded to it. Additional information which can be derived from the alerts is also to be provided, in the form of meta data, an example for this would be the count of duplicate alerts contained in the network.

2.2 Environment Specification

This section serves as an introduction to the problem domain and the different components located in it. It will provide an overview of what an Intrusion Detection System is and how it functions, how alerts are presented as a readable format, and what the Policy Decision Points and Policy Enforcement Points functionality encompasses.

2.2.1 Intrusion Detection Systems

Following explanations are taken from [Carv00, FGHW^{+}08, Axel00] except if stated otherwise.

The most well known types of Intrusion Detection Systems are Network Intrusion Detection Systems(NIDS) which monitor network segments for malicious activity and Host Intrusion Detection Systems(HIDS), which monitor a host from the network. An example of an Intrusion Detection System would be an antivirus which classifies as a Host Intrusion Detection System.

One reason to deploy different Intrusion Detection Systems, derives from the heterogeneity of the different Intrusion Detection Systems that exist. Different Intrusion Detection Systems detect different attacks, or classify attacks in another way. In addition to ensure that attacks will definitely be detected, Intrusion Detection Systems need to overlap each other in critical segments of a network, for example a network which contains hosts with access to confidential information [CaCM02].

Intrusion Detection Systems also posses over a variety of detection methods. The first method is used by so called signature based Intrusion Detection Systems. These try to detect threats based on their specific signature. A signature can be envisioned as the unique identifiers of a threat, and can be very specific about what type of threat has been detected.

The second detection method is used by anomaly based Intrusion Detection Systems. Anomaly detectors can detect threats by classifying them as anomalies. Anomalies are not quantifiable and cannot be classified, this happens because the Intrusion Detection System knows how normal network behaviour and network traffic looks like, and if something is misbehaving then it is classified as an anomaly. Anomaly detectors are split into two subtypes the self learning and the programmed anomaly detectors.

Self learning detectors observe the pristine state of a network, building an internal model of how the network behaves. Self learning detectors are split into two subtypes depending on how they observe. The first is the non time series observation which does not consider time constrains and uses stochastic models to create the model of the pristine network. The second one implements a time series which is identical to the previous one except it also accounts for a time, that means for example that it knows how the traffic is at midday with peak traffic, and differentiates it for example with the night time traffic where the activity in the network is not so intense.

The other type of anomaly detectors are the programmed detectors. As the name states, the detector has to be programmed, a person is required to teach the system what an anomalous event looks like. A user controls what is considered normal and what is considered anomalous. These are also split in two categories. The first being the descriptive statistics detection method which builds a normal statistical behaviour from the parameters it collects in a system. An example parameter is the statistical amount of failed logins. The other category is called the default deny. The detector is explicitly told which of the observed data is normal, any deviation to that is considered an anomalous event.

In layman term an anomaly detectors monitor for suspicious activity. Suspicious is not directly malicious so it has to be considered that anomaly detectors are not as precise as signature based, but can detect the attack types which have still to be identified.

When an Intrusion Detection System detects a suspicious event on the network or system it monitors, it will generate an alert that contains the information about the source, the target and the estimated type of the malicious event. An **Event** in this case, is a single attack instance, since an attack is either a single attack or a combination of a variety of

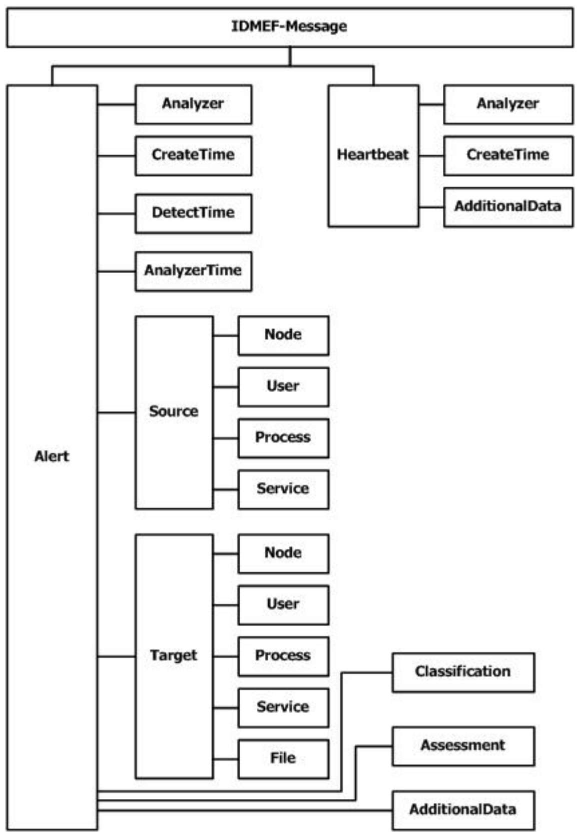

Figure 2.1: Visualised IDMEF Framework [HDFe07]

attacks. Intrusion Detection Systems cannot tell the difference between an *attack type*, that means single attack instance and an attack instance, a multitude of single attack instances, since they are only able to detect the type of a single attack [HoSi11].

2.2.2 Alert Representation

Alerts generated by Intrusion Detection System, are encoded in various formats. In order to process these alerts, they have to be transformed into a common format. Explained in detail in Section 2.3, normalisation is a process which transforms the alerts from the initially encoded format into a common format [MaMZ09, ZaYZ08].

The information that a common format can offer is visualised in Figure 2.1. Figure 2.1 depicts one of the available concepts to represent normalized alerts. The concept in Figure 2.1. contains mandatory and option fields. The mandatory fields are the *Analyser*, *CreateTime* and the *Classification* of the detected Event. The *Classification* contains the name of the attack type, and optional references. The *Analyser* field contains information about the Intrusion Detection System that identified the event, whereas the *CreateTime* field the time the alert was generated. [RoCM09, HDFe07]

The core optional fields are the *Source*, *Target* and *DetectTime* fields. The *Source* and *Target* field contain the source and the target of a detected event. These two optional fields include additional fields, which contain information about a user, service, process or a file that may be causes or receivers of a detected event. The *DetectTime*, which might be different than the *CreateTime*, specifies the exact time an event was detected. [RoCM09, HDFe07]

Although a common format allows the security experts, or the Policy Decision Point to process all alerts in an identical manner, not only do different Intrusion Detection Systems provide differently encoded alerts, they also provide different information. For example an HIDS can identify the particular service an event is targeting, whereas an NIDS can identify the port(s) that a malicious event targeted. [MaMZ09, CaCM02]

2.2.3 Policy Decision Points and Policy Enforcement Points

Following definitions are taken from [G. W99, Reye04], except if stated otherwise.

The Policy Decision Point (PDP), is the component in charge for managing the security in networks and networked components. The Policy Decision Point handles decisions of how a system will react to a detected attack instance. The IETF defines the PDP as the point where decisions that are to be applied on a network are created. The PDP processes the policies and in combination with various network information decides which policy is to be applied. The PDP does not implement the policies, but sends configuration data to the so called Policy Enforcement Points(PEP). The PDP's main responsibility is to locate the policies in a network, and transform the syntax of those policies in a syntax understood by the different PEPs contained in its network. It also monitors the state of the network in order to validate if all the conditions that are defined in its policies are held.

The receiving end of the Policy's Decision Points decisions is the Policy Enforcement Point or PEP. The Policy Enforcement Point handles the actions a system takes based on defined policies. The IETF defines the PEP as the place where decisions are actually enforced. The PEP are devices that participate in the execution of decisions forwarded by the PDP, a prime example of a PEP is a router or a firewall. The PEP functions by receiving update configuration data which has to be implemented and in addition informs the PDP of unknown situations that may occur in the system it is in place.

An example of a Policy Enforcement Point, would be the firewall. The firewall acts as a border control mechanism. Its function is to block traffic from the outside, but can also be utilized to block traffic originating from inside the network [Daya]. Blocking traffic originating from inside a network helps for example to restrict movement of worms. This prevents unauthorized access from and to the network [Daya]. As a PEP the firewalls configuration can be changed by the Policy Decision Point, to block additional ports that might be the target of an attack to the network.

In the networked system that is being investigated, the PDP receives the different alerts and generates decisions upon whom the PEPs implement defensive actions. Before the PDP receives the data, the alerts are processed in order to be presented in a better format, add additional information and derive logical links between alerts. The next section will introduce this process which allows the different alerts to be processed by the Policy Decision Point. A subcomponent of that process is also the main research area for this thesis.

2.3 Correlation Process

The field of research this thesis is based upon is called *Alert Fusion* or *alert correlation process* [VVKK04, MaMZ09]. Figure 2.2 visualises the correlation process as defined in [VVKK04] design. The explanations that follows is also based on the explanation delivered in [VVKK04] with some adjustments taken from [MaMZ09].

Normalisation: Details the process for transforming an alert from the machine generated format generated by an Intrusion Detection System to a common format. Further details on the common formats is presented in Section 2.2.2

Pre Processing: Details the process which is in charge of augmenting the normalised alerts, so that all fields contained in the common format contain meaningful values.

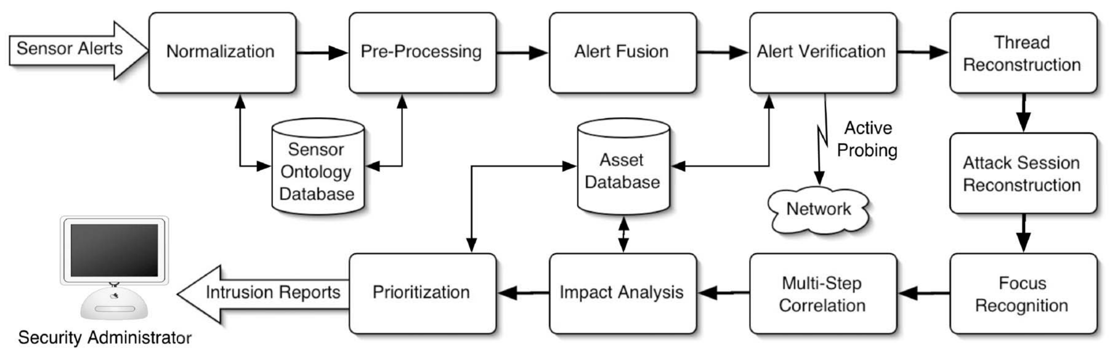

Figure 2.2: Correlation Process [VVKK04]

Alert Fusion: Also defined as Alert Aggregation in [MaMZ09] is the process which combines together alerts which share common traits. This means it combines alerts which are detected independently by different Intrusion Detection Systems, but refer to the same initial event

Alert Verification: This process filters out false positives which are received by the system. In addition it verifies the success an attack had on the network in order to reduce its influence in the correlation process when the attack had failed to produce a meaningful threat to the system.

Thread Reconstruction: Reconstructs attacks which were launched by a single attacker to a single target.

Attack Session Reconstruction: It is in charge of associating alerts which were generated by network Intrusion Detection Systems with alerts generated by host Intrusion Detection Systems.

Focus Recognition: This process operates on alerts which contain a large number of targeted devices. Its target is to identify the original source or the intended target of an attack, for example in the case of a DDoS.

Multi Step Correlation: This component, analyses common patterns in attacks. The patterns are composed of different attack vectors which can be distributed throughout a network, but can be summed up as one attack.

Impact Analysis: Analyses the impact an attack had on the operation of the networks individual components, and the individual assets which are monitored.

Prioritization: Assigns priorities to alerts, in order to quickly discard information which can be considered irrelevant for the safety of a network.

Figure 2.2 is not a standardised concept, and there exists no fixed order for how they can be implemented. Some of these processes can operate in parallel to each other and it is not necessary for every alert to pass through every sub process [VVKK04]. Many researchers define their own way of implementing the correlation process [MaMZ09, VVKK04]. The only exception to this are *Normalisation, Fusion, Correlation*, which have a predefined order. Normalisation always comes before subsequent processing takes place and is always at the start of the correlation process chain. Alert fusion is always before alert correlation, and alert correlation is the last part of the three to be implemented [MaMZ09, VVKK04].

The solution to be developed here, implements the *Alert Fusion* part of the chain visualised in Figure 2.2. Subsequent sections will analyse the problems contained in this domain and pose them as requirements to the solution which is to be to be developed.

2.4 Identified Problems

Section 2.2 introduced the environment, the networked components that are to be investigated. Section 2.3 introduced the *Alert Correlation Process* where the sub process *Alert Aggregation* is essentially the approach used to solve the problems in this thesis. This section will introduce the problems identified in every part of the information chain of the environment presented previously.

2.4.1 Alert Stream

Nowadays network administrators, increasingly deploy a large number of Intrusion Detection Systems, which have overlapping domains. This is done in order to secure critical regions in a network. These critical regions might contain services which are used in critical infrastructures , or sensitive company information. The increasing number of deployed Intrusion Detection Systems, simultaneously increases the amount of alerts generated per event. Thus the alert stream generated is also increased substantially [MaMZ09, Cupp01].

Through the overlapping domains, an increasing number of alerts is going to be referring to the same event. The consequence is that the receiving end of the alerts, the PDP or the system administrator, gets flooded with duplicate alerts [MaMZ09, Cupp01]. In the domain of ANSII the receiving end is the Policy Decision Point. The Policy Decision Point, as explained in Section 2.2.3, receives the events and in turn generates decisions based on conditions contained in its configuration.

The Policy Decision Point, will generate a decision for every alert received, since the system does not posses the ability to remember which alert it processed and which alert is new to the system. This means that if ten alerts arrive referring to the same event instance, the PDP will generate 10 decisions which will be more or less identical if the alerts contain the same information and satisfy the same conditions specified in the PDP for a decision. This happens because, even if the PDP does contain other actions it will always consider the most cost effective measure to implement. If the conditions provided from the alerts do not change between the alert instances, the PDP will always generate the same action. Thus it is unnecessary for the PDP to receive the alerts which provide the same information, that means the same conditions, as in the end the decision that will be generated will be the same.

The decisions themselves are received by the Policy Enforcement Points. These will then implement the decision in form of an action. A prime example for an action would be the reboot of a virtual machine. If the reboot action is executed 10 times in a row, because of the received decisions, the virtual machine would have a significant down time, attributed to the actions undertaken by the system that is meant to protect it.

The solution to this problem is to filter out the alerts that have already been seen by the system from the alert stream. To identify these alerts and classify these alerts, a solution has to, based on how alerts are represented, find their common traits in the mandatory fields of the common alert format, for example if two alerts contain the same classification, originate from the same IDS and target refers to the same host then the alerts would be similar or duplicate. A solution should serve the PDP only the necessary information for generating a decision, in order to increase the efficacy of the PDP in its decision making process, by reducing the process of trying to find a matching rule. In addition the solution to be developed should act as the PDPs memory, by saving the alerts going to towards the PDP, in order to remember which alerts were seen before by the PDP.

This approach would reduce the number of alerts reaching the PDP. But it introduces some problems, as explained in Section 2.2.1 different IDS types provide different type of information. For example, a NIDS and a HIDS generate 2 alerts referring to the same event, and the module blocks the alert generated by the HIDS. If the PDP contains a rule which is tailored to the alert generated by the HIDS, the PDP will not react to the alert forwarded towards it. Subsequent alerts are going to be blocked, and the Policy Decision Point will not generate a decision, thus the Policy Enforcement Points will not react to the attack.

2.4.2 Information Completeness

As highlighted in Section 2.4.1, by blocking alerts, vital information might also get lost. Illustrated in the example of Section 2.5, this information might be essential for the PDP in order to generate a decision, for example the service that is under attack. In addition, there is no information about the alert stream itself, e.g. one could count the amount of alerts coming through that display identical data.

An example to understand the information that can be derived by the alert stream is to consider the following example, an event detected by an IDS classifies as a low severity event. The event itself is not considered as serious, since no action has to be taken, but what happens if the event is repeated 100 times in a row. In this case, if the PDP had this information it could consider the event as a serious threat to the system, and thus generate decisions in order to invoke defensive actions.

To engage and solve the above problems here, the essential solution is to use alert aggregation, in which similar alerts are aggregated to one alert. Existing solutions such as [PoNe97], do provide alert aggregation mechanics, but they do not provide any information about the alert stream. In addition, the focus of most projects is the correlation process itself [MaMZ09] without paying attention to the aggregation part of the process chain displayed in Figure 2.2.

By providing the system with aggregated alerts, the first benefit is that the Policy Decision Point will receive all the information about an a detected event, attack-instance, which deals with the problem introduced by blocking the alerts. The additional information collected by the alert stream, can enable the Policy Decision Point to be more flexible, since the alert aggregation and information gathering can change the conditions that a normal alerts would provide.

2.4.3 Interaction with Policy Enforcement Points

Another subject that is not handled by existing solutions is the process of relaying feedback of the actions enforced in the network. In a traditional system, actions are implemented either by the Policy Enforcement Points or the system administrator. An action can be distinguished into three possible cases, the first being when an action can be successfully applied onto a target device/system/network. This means that the enforcement points did not get any errors in implementing the action. The second case for an action is when the action fails to implement. This means that the Policy Enforcement Points, or the security administrator, could not successfully implement this action, where the cause for this can vary, for example the attack instance is blocking the action from being implemented. Another part of the actions are the actions that have not been executed yet. Although the Policy Enforcement Points and administrators do posses a variety of actions that can be implemented, most of the time the measure which has the lowest cost for all the actors in the network is implemented. The PEP and the administrator, although having information about outcome of the process of implementing an action, they cannot infer if an action successfully alleviated a detected attack instance.

To tackle this problem, the Policy Enforcement Point should relay the feedback about the outcomes when implementing an action. The feedback should be structured in such way, that it includes the actions successfully implemented, those who failed to implement and the actions that have yet to be tried. The successfully implemented actions field, will enable the security system in place to recognize if a measure was successful in dealing with the attack instance, since if any alert arriving after the action was implemented signals that the attack instance is still persisting in the network. The only caveat here is when the attack is performed via an external source. It cannot be distinguished between if an action that was taken was executed successfully, resolved the detected attack instance, only for the attack instance to re- emerge because the target system is getting infected by another source, for example a USB, and an action that although was successfully executed, did not seem to defuse the detected attack instance.

The actions that the PEPs determined to have failed to implement , and those that still haven't been implemented yet, provide the Policy Decision Point with additional actions that still haven't been used to resolve the attack instance. For example for a persistent attack instance, the Policy Decision Point could consider implementing actions that failed to implement or that have yet to be implemented, which although may come at the penalty of increased costs, are able to resolve the attack instance. Some problems that can be considered here is that the feedback from the Policy Enforcement Points might introduce network lag, or be abused by attackers by including spoofed feedbacks. This problems are not handled here, since it is not in the scope of this analysis, but are touched as part of Chapter 6.

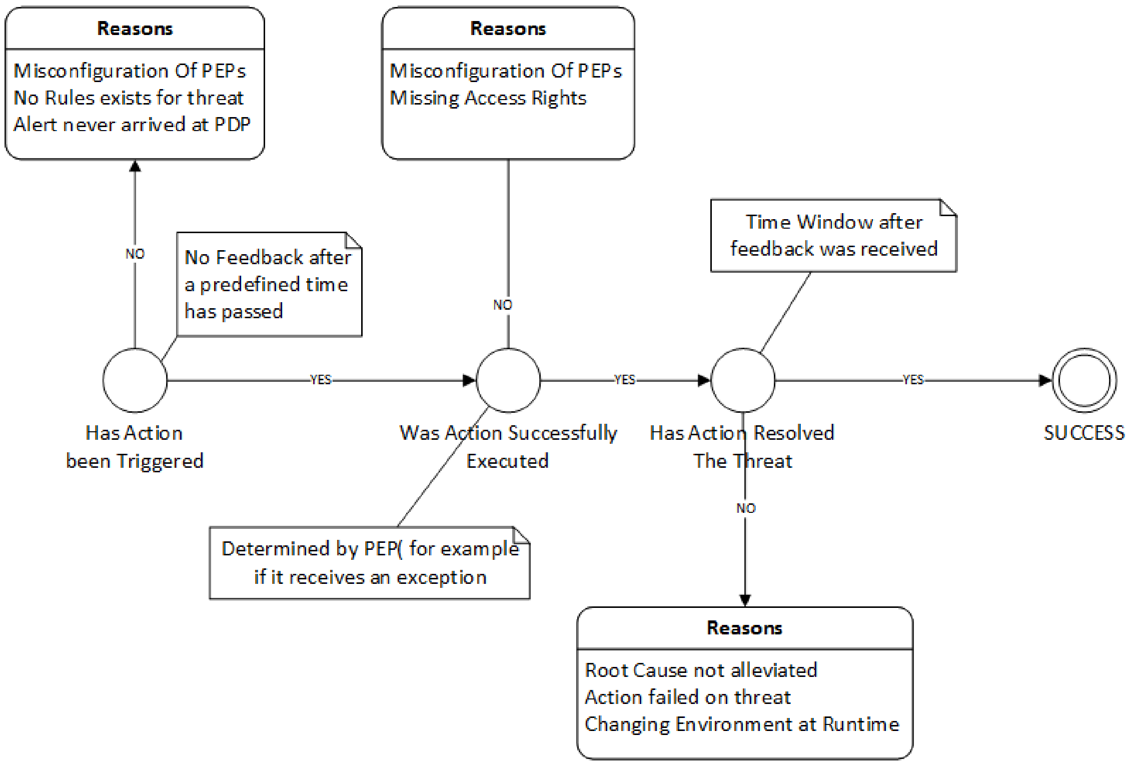
Reasons
Misconfiguration Of PEPs
No Rules exists for threat
Alert never arrived at PDP
Reasons
Misconfiguration Of PEPs
Missing Access Rights
Time Window after feedback was received
NO
No Feedback after a predefined time has passed
NO
YES
YES
YES
Has Action been Triggered
Was Action Successfully Executed
Has Action Resolved The Threat
SUCCESS
NO
Determined by PEP(for example if it receives an exception
Reasons
Root Cause not alleviated
Action failed on threat
Changing Environment at Runtime

Figure 2.3: Decision tree for Actions

Figure 2.3 highlights the benefits of having feedback relayed back from the Policy Enforcement Points in the form of a decision tree. By relaying feedback it can determined if any action that has been triggered by a specific alert also got successfully implemented, by examining if feedback for a specific has been received after the specific alert was forwarded to the PDP.

The Policy Enforcement Points can determine if an action was successfully executed in the target system. This can be identified by them, since for example an action that fails to implement can cause an exception to be thrown. In the end the Policy Decision Point can determine if an action successfully resolved an attack instance, or if it failed to produce any results. The Policy Decision Point cannot determine why the action did not produce any results in resolving the attack instance, since it could be that the root cause of the attack instance was not dealt with.

2.4.4 Dynamic Aggregation and Default Behaviour

Although alert aggregation and the feedback solve most of the problems, a minor problem still exists. Today, a variety of policy specification language and a variety of pre build policy solutions exist. If the above a solution is to be implemented, it would require from the administrators, to adapt the rules in every Policy Decision Point according to the way the aggregation is performed, and to how and what aggregation data is collected. This is especially hard in networks that contain a large enough amount of policy decision points, since everyone of them has to be manually adapted to the solution, and to any updates or changes that the solution introduces later on.

The problem here to only including dynamic generated aggregation rules, is when no rules exist for a specific attack classification. To counter this, the solution should implement a default aggregation behaviour, in order to be able to handle and identify duplicate alerts when no dynamic configuration exists. The default behaviour should aggregate the content based on common fields contained in the alerts.

A core problems of the dynamic rules concept, is that the pre-processing that takes place will inadvertently add time-delay between the time the alerts will reach the Policy Decision Point, thus adding a time-delay to when an action is implemented. In addition the dynamic aggregation rules can also serve as an attack vector, through the configuration file. An attacker could provide a fake configuration which enables him to circumvent the solution, thus the solution would provide false information to the Policy Decision Point. These problems are over the scope of this Chapter, thus want be handled here, but are worth to be mentioned in order to understand some of the trade-offs that might come with the proposed solution. These subjects will be analysed further as part of Chapter 6.

2.5 Sample Scenario

As stated in Section 2.2.3 the Policy Decision Point receives events. An alert describes a detected event in the network, which is thought to be non-conforming(anomaly detection) or malicious(signature detection). In ANSII, the Policy Decision Point, receives the alerts in order to automate the reaction to attack instances. The alerts are previously normalised into a common format, for example the IDMEF format presented in Section 2.2.2. The following example will portray the problems identified above in a simple scenario, in order to enable the visualisation of the concepts presented throughout this chapter.

Figure 2.4 depicts the network that is going to serve as an example. The network here contains *three Intrusion Detection Systems, a Host computer, a Policy Decision Point and a Policy Enforcement Point.* The Intrusion Detection Systems are split in the following way:

- **HIDS** monitors the activity on the host. Is similar to an anti virus system.
- **NIDS1** monitors the network segment, where the Host is located.
- **NIDS2** monitors a network which partly overlaps with the network of NIDS1, but is another type of NIDS.

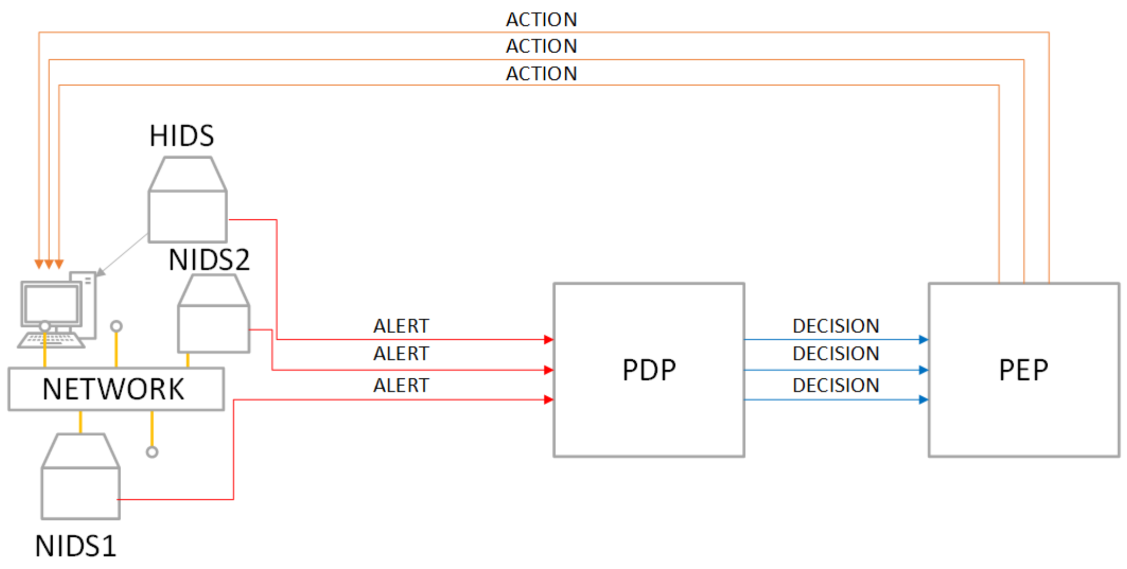

Figure 2.4: Sample Network

An attacker starts using a Distributed Denial of Service(DDoS) attack against the Internet Information Services service running on the Host. The Intrusion Detection Systems, will detect the DDoS as nonconforming, non regular traffic event, and will therefore generate an alert containing the information about the source, target, classification of the attack.

Although the alerts describe the same malicious event(the DDoS), they contain different information. The HIDS, captures the attack from the host's point of view. The information contained in the alert generated by the HIDS contains more information about which service was specifically targeted and which files might have been targeted. The NIDSs contain more network specific information, the individual ports and have a more accurate picture from where the attack is coming from, for example multiple sources. Considering that the NIDSs are heterogeneous, NIDS1 and NIDS2 contain different information since each one of the has a different view of the system. Here the concept talked about in Section 2.4.1 should become clear. Although the alerts contain different types of information, they are similar, since they describe the same event just from different point of views.

The alerts, will reach the Policy Decision Point. The PDP is in charge of processing the Alerts, and to generate decisions on what actions are to be used against the underlying event. The decisions generated in the Policy Decision Point, are specified in a *condition then action* concept. The conditions are mapped to one ore more actions, and the actions, are chosen to be implemented based on their cost. The PDP in the scenario of Figure 2.4 contains the following condition: *When Classification equal to DDoS*, for which it executes the following action: *Restart the Host.*

The actions can change if the conditions change, as mentioned in 2.4.3. Since the alerts generated correspond to the same attack instance, no matter how many alerts arrive, since the environment of the conditions does not change, the same action is going to be executed for those alerts. If in the example above, the DDoS is performed continuously, it will lead to the generation of a huge number of alerts which always invoke the same action, as the environment does not change and thus the same conditions will be activated over and over again. This is also a core problem in a network with a large number of deployed Intrusion Detection Systems [MaMZ09]. One of the reasons that this can be considered a problem, is that when enough alerts have accumulated, the Policy Enforcement Points in the network will be too involved in reacting to the one event that generated the alerts, that it may not react to alerts which refer to an event of a lower priority. A solution to his would be the requirement posed as part of Section 2.4.1 where duplicate alerts are to be blocked.

As explained in Section 2.4.1 and 2.4.2, if duplicate alerts are to be blocked the information which may be crucial for a decision, that means information that brings a change to the environment of the detected attack instance, may also accidentally get blocked, even if it may invoke a new type of decision. If the rule specifies a specific service that should be the target of the attack before enforcing any actions, then if the Alert generated by the HIDS gets blocked, no measure will be implemented to resolve the event that generated the Alerts. To alleviate this, in Section 2.4.2 it was defined that no information should be discarded, for example information about the targeted service, but rather be saved together with informations which can be derived by inspecting the Alert Stream. This should ensure that even if the solution does not know the rules contained in the Policy Decision Point, it will always gather additional Information in order to force a change in environment conditions, for example through the total amount of accumulated alerts.

After the PDP generates a decision, it forwards updated configurations to the PEPs in order to enforce the actions decided by the PDP. The PEP will implement the actions chosen by the PDP. In the case of the Policy Enforcement Point, it does not provide any additional information on whether or not an action was successfully implemented (not successful in resolving the attack instance) or failed because of an error. This was detailed in 2.4.3, and Figure 2.3 illustrates the insight that can be gained by providing and saving feedback information on the actions. It also helps to provide a more complete picture of what happens in the network. The feedback will also help determine which actions are not effective against a particular attack instance, if they were successfully executed but did not have the capability to resolve the attack instance.

2.6 Summary

and how each of these devices function. By having IDSs distributed in overlapping domains a number of alerts is created that may be identical to each other. In order to understand links between events, alerts are analysed through a correlation process. Alert aggregation is part of the correlation process, which is an essential tool in order to derive more information from the alerts, by eliminating duplicates and fusing information, and the main solution domain of this thesis.

This chapter identified four problems which are mapped to following requirements:

- **R1**: Elimination of duplicates in the alert stream, and as a subsequent side effect reduce the volume of alerts.
- **R2**: Prevent the loss of information when blocking alerts
- **R3**: Communicate with the Policy Enforcement Point, in order to receive feedback about performed actions.
- **R4**: Flexibility in policy specification, through dynamic provided rules

R3 is an essential part of the solution since as mention in Section 2.4.3, through the feedback provided more insight can be gained on how effective a defensive measure is against a specific attack instances.

R4 should simplify the rule specification in PDP, since it tries to reduce the process of adapting the rules to the content that would be provided by the solution which is to be developed.

3. Related Work

3.1 Field of Research

As explained in Section 2 the research field is called alert correlation process or Alert Fusion. The main focus of the research is to solve the problem that arises when many IDS systems are present in a networked environment. Alert fusion describes the format conversion phase and is divided in the processes specified in Section 2.3.

Alert aggregation has as its main function to group alerts together which present similar traits, the alerts are called similar since the word identical denotes that they contain the same information which is not the case, since the information differs from alert to alert(a prime example would be the time of detection which varies between alerts almost always even if the are talking about the same alert) [VVKK04, MaMZ09]

Alert correlation tries to identify logical relations between the alerts. Alert correlation is usually performed after alert aggregation, and this explains also way a number of papers discussed below feature alert aggregation even if it is not the main focus of the papers. Although the main focus of this thesis is alert aggregation in automated environments, most of the research presented here is rather either describing the process as a whole, or focusing on alert correlation [VVKK04, MaMZ09]

Similar to the system being specified here are so-called Intrusion Response Systems or IRS. These systems combine monitoring and reacting in one package. The approaches themselves are divided in three principal categories [Carv00].

1. Notification Systems: If an attack is detected, then the system administrator or the security system is to be notified. The response is left up to the System Administrator.

2. Manual Response: Offers a variety of pre-programmed responses that can be initiated at the request of the System Administrator. Again it requires human interaction in order to repel attacks.

3. Automatic Intrusion Response: The systems do not wait for the System Administrator to respond, but rather contain a set of responses. These type of systems are again split into their own subcategories.

    - Simple Decision Table: Responses associated with an attack that can be detected. This is a one to one mapping of reactions to certain attack classifications, there are no alternative actions to be taken by the system.

- Rule-Based decision module: This type of IRS implements a variety of rules to allow the system to implement varying responses based on those rules. These systems are very similar to the concept detailed in this thesis.

At the time of writing this thesis, of the 56 IRS systems available, only 17 implement an automatic response behaviour. Of these 17 only 4 of the systems implement a Rule-Based decision module [Carv00].

The most notable of the Rule-Based IRS systems, is EMERALD. EMERALD stands for Event Monitoring Enabling Responding to Anomalous Live Disturbances is on the most similar approaches to the approached that is detailed in this thesis. It is diversified implementing a system that can be distributed throughout network domains, and implements misuse and anomaly detection systems. Target base of EMERALD are large based enterprise networks with the need for a scalable intrusion detection and response system [Carv00, FGHW$^+$08].

EMERALDs architecture is hierarchical. It includes its own custom IDS system, which is again has a strict hierarchy. It is split in following components [PoNe97, FGHW$^+$08]:

- *Service Analysis*: Detects misuse of individual network components in a single domain
- *Domain-wide Analysis*: Detects misuse across a number of components in a single domain, the difference to the service analysis is that instead of looking for misuses on single components it targets attacks that are distributed on a number of components in one domain.
- Enterprise-wide Analysis: Performs both of the above functionalities but across all domains contained in the enterprise network.

The NIDSs in EMERALD are called Monitor. They are only responsible to monitor the network and detect any event classified by them as an attack-instance. EMERALDs Monitor architecture is the core of the architecture. The monitor itself also follows a hierarchical pattern implementing a number of components at its basis.

- *Profiler Engine*: a statistical anomaly detection engine.
- *Signature Engine*: implements the signature based detection engine.
- *Resolver*: A coordinator for analysis and response
- *Resource Object*: A library that can be plugged in containing the data of the three other components

The automatic response system,is handled by the Resolver. By employing expert rules, it receives the events as alerts and invokes a variety of response handlers. The responses here are handled by two specific metrics.

- **Threshold**: which defines how much evidence should be present before using a measure
- **Severity**: which defines the cost of using the specific response actions

As explained in Section 2.3 alert aggregation is part of the correlation process. EMERALD also uses aggregation as part of the correlation engine. Aggregation is implemented by employing a concept called *alert threat*. Alerts received by EMERALDS modules, are considered part of the same threat if they come from the same sensor, and contain the same classification. EMERALD treats the subsequent alerts arriving by adding them into the existing threat [Carv00, PoNe97, FGHW$^+$08].

The core difference to our approach that can be observed in regards to EMERALD, is that our module also tries to address the problem of missing feedback in order to provide more flexible solutions to the Policy Decision and Enforcement Point. In addition no advanced rules are specified apart from the expert rules, aggregation data is collected using static modifiers.

In [DeWe01], Debar and Wespi describe an aggregation method again as part of their overall correlation process. Alerts are treated the same if identical in the three fields *Classification, Target and Source Address*, based on the representation described in 2.2.2. The aggregation system portrays common alerts as the triple *(src,dest,class)*. To denote similar fields the algorithm employed uses a wild card ***.

For example:

- (src,dest,*) if alerts are containing the same classification
- (*,dest,class) if alerts are contain the same source
- (*,dest,*) if they contain the same classification and same source

The aggregation method is only used to reduce the volume of received alerts. The correlation component works on the situations provided by the aggregation component. The difference to our approach is that the feature selection here is static, and alerts are categorized on only three features. No feedback is included in the alerts forwarded and in addition no aggregation data is collected.

Another correlation engine that employs aggregation is CRIM from the MIRADOR project. CRIM uses predicate logic to quantify the alerts, and the alerts are stored in a relational database. The relational database is used to identify the similarity on the different alert fields. Through expert rules the similarities between the alerts create clusters. The clusters then are merged together using an aggregation algorithm. The algorithm produces an indicator called stability, that represents when an alert can be reported. The cluster stability is a variable that depends on the type of the attack.

[FaJM09] proposes an architecture to aggregate alerts in distribute IDS implementations. A centralized IDS collects alerts from client IDSs. An aggregation agent aggregates the alerts which are forwarded to it from the central IDS agent. The aim is to improve performance between the heterogeneous central IDSs that are placed in the network. The aggregation algorithm is as follows:

1. Alerts are analysed and placed into four classes: *discover, scan, dos, privilege escalation.*
2. The class placement depends on the type of attack detected. For example a port scan would be placed in the *dos* class.
3. Alerts are treated similar based on the class they have been placed

4. Alert fields are divided into four classes depending on the type of their value. The classes are *categorical, numerical, temporal and character* types. For example the time of detection is a temporal value.

5. The classification serve as a precursor for implementing a feature similarity computation as categorical and numerical values yield either true if their identical and false otherwise.

6. Their algorithm takes as input the original alert, a meta-alert list, a predefined computing library and a configuration file. The output creates a new or an updated meta-alert (fused alert).

The approach differs to then detailed in this paper is that the feedback is not considered in order to keep track of what was implemented as a counter measure.

[MaMZ09] focus is alert aggregation. It tries to reduce the false positives from alerts generated by anomaly detection IDS. They advocated the use of fuzzy time sets, instead of fixed time intervals, to determine similarity of alerts. Because on anomaly based IDSs the anomaly is not known a-priori, this breaks the limiting factor introduced by this. What is again not considered here is the feedback and the insight that can be won out of it was explained in Section 2.4.3.

3.2 Difference of Thesis Approach

The approaches presented as part of the Related work, do not consider the insight that can be learned by analysing the that the system implements and the effect that they produce. In addition the expert rules create the need for creating different policies, since no rules for aggregating are directly derived from the policies. This thesis takes under consideration the matter of relaying feedback about the defensive actions from the PEPs and how to derive aggregation rules based on the rules that the Policy Decision Point contains. Through the feedback, the solution tries to provide insight on what actions can be used to resolve a specific threat.

From the systems and solutions detailed in related works, EMERALDs aggregation approach is the most similar to the solution that is presented in this thesis. Although it eliminates the problem of fusing alerts together and presenting them as one unique instance, it does not solve the problem posed in **R3** of Section 2.6. No feedback is relayed on the actions the IRS implemented in order to resolve the threat.

Most of the related work, details the correlation process as a whole, and not the aggregation part itself. Most of the solutions as in the case of EMERALD, reduce the number of duplicates using alert aggregation, fuse alerts together in order to provide complete informations about a specific threat, but do not consider feedback as an essential information for creating automatic response mechanisms. In addition, none of the related work presented here consider providing a dynamic aggregation method which adapts to the changing requirements of the Policy Decision Point.

4. Design

Section 2 explored the problems that arise when having a large number of Intrusion Detection Systems in a network. The core problems that will be addressed by our Aggregation Module are:

- Remove duplicate alerts from the alert stream
- Aggregate alerts with common traits to a larger alert, with no information loss
- Provide additional data that can be inferred from the alert stream
- Provide additional insight to the events that generated the alerts, by including feedback on actions performed by the PEPs
- Provide an easy way to scale the solution, by dynamically aggregating using the configuration of the local Policy Decision Point

Figure 4.1 depicts the *Aggregation Module* that needs to be added to the network. The Aggregation Module's position has to be between the alerts and the PDP, in order to control the flow of alerts towards the PDP. Its main mission is to aggregate alerts, enhance them with additional information and to block the duplicate alerts contained in the alert stream, in order to provide the necessary information for the Policy Decision Point to make a decision.

The alert produced in the Aggregation Module is going to be called a fused alert. A fused alert contains all the information of the individual alerts that share common traits, such as having the same classification. This is further elaborated in Section 4.3.2.

The Aggregation Module can also be viewed as the brain of the PDP since it stores alert instances, filters them out based on the PDPs specific decision making need, in order to make this information available when needed. The Aggregation Module will in addition retain a list of the actions that were implemented in the network. From this point, the Policy Decision Point can, through the feedback, use alternate actions to combat attack instances which are unresolved.

If the Aggregation Module would have been placed in another location in the network, it would not provide any benefit since it would not be able to pass any additional insight to other security components, and would loose its purpose in pre-processing the alerts

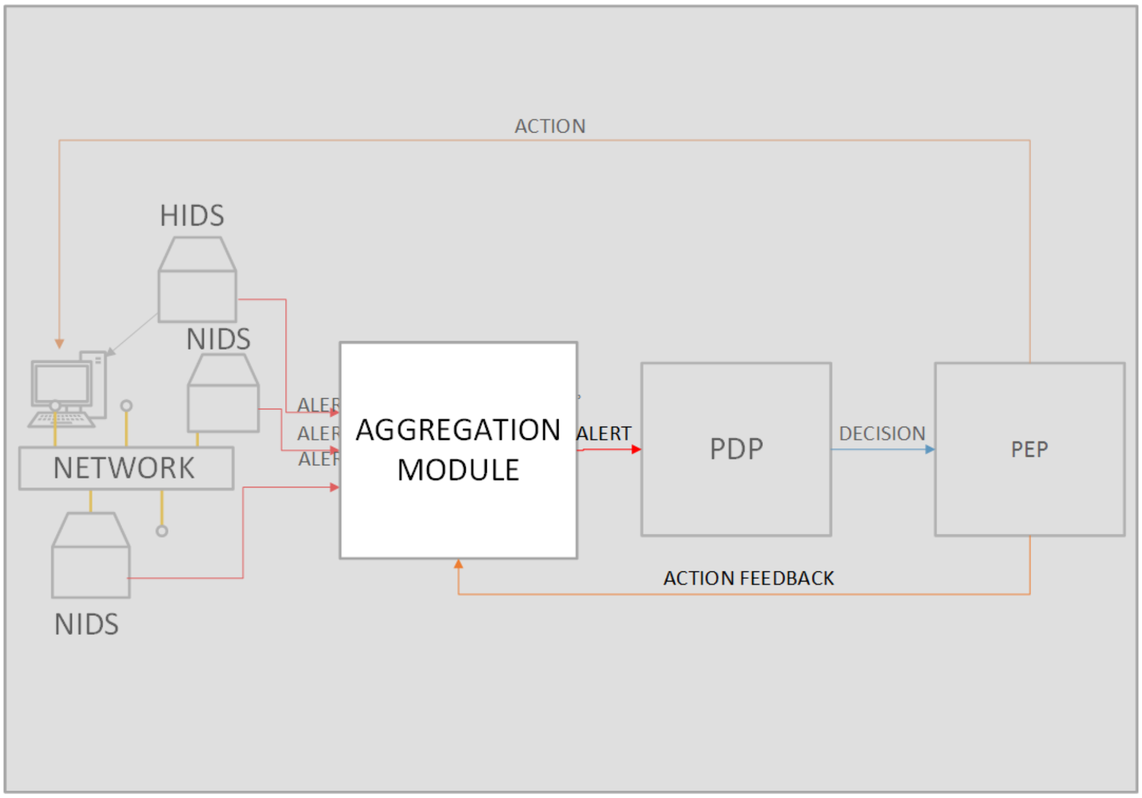

Figure 4.1: Alert Aggregation Module

before they reach the Policy Decision Point. In addition to this, the feedback which will be provided by the Policy Enforcement Points, needs to be handled by the Aggregation Module in order to be forwarded together with the alert that caused the action to be implemented.

4.1 Definitions

In the design chapter, there is going to be an increased use of custom definitions. These definitions will be elaborated here.

- *Actions:*Measures undertaken by the Policy Enforcement Points in a network or host, in order to defend the network from detected attacks. Actions are derived from the feedback received from the Policy Enforcement Points.
- *Similar Alert:* An alert which refers to the same instance of an attack, as a previously saved one. Are also called duplicate alerts in other papers. We refrain from using the term duplicate as duplicate refers to identical objects.
- *Alert Aggregation:* describes the combination of alerts which are similar. These alerts may contain heterogeneous information, but they are referring to the same event(Single attack-instance).
- *Fused Alert:* The product of alert aggregation. The alert that is going to be produced by the *Aggregation Module*, which combines the heterogeneous information
- *Repeated Alert:* A similar alert which was received at the Aggregation Module, after the Policy Decision and Enforcement Point hat already implemented an action against the identified attack.

- *Triggered Alert:* A fused alert which gets forwarded based on certain triggers. An example trigger would be no action has been performed for the alert in the last 10 minutes.
- *Meta data:* Meta data are the additional information appended to the alert. Meta data contains a number of aggregate data. and the actions performed by the Policy Enforcement Points.
- *Aggregate Data:* Data that is derived from the alert stream. An example for aggregate data is the amount of similar alerts received.

4.2 Aggregation Module Components Overview

This section provides an overview of the Components that compose the Aggregation Module. A detailed explanation of how the interplay and how the components functions is provided in Section 4.3

4.2.1 Stream Filtering

Stream filtering is the component which is responsible for filtering the alert stream. Its basic function is to filter the stream based on rules derived by the Policy Decision Point. If for a specific classification, no rule exists in the PDPs configuration then, the Stream Filtering component should define a default behaviour. The default behaviour filters alerts that have already been seen by the Aggregation Module, and forward alerts that are per 4.1 repeated or triggered. This behaviour is discussed in more detail in Section 4.5.

The component satisfies the requirements set that not every alert should be forwarded and duplicate alerts should be blocked, as stated in **R1** of Section 2.6. In addition as a subsequent positive side effect it helps reduce the traffic that is directed towards the PDP. Last but not least, the component itself is here to change the PDP from a stateless machine to a state-full machine by remembering which attack instances have already been reported to the PDP.

4.2.2 Alert Fusion

Alert fusion is the component that will perform the alert aggregation and meta aggregate data gathering. Its main functionality is to gather aggregate data from the alert stream and provide an interface where the similar alerts are combined together, based on their common traits. The data to aggregate are either derived from the rules contained in the Policy Decision Point in the case of dynamic aggregation, or when no rules exist using a pre-defined rules specified and executed by the default aggregation behaviour. The aggregation process and the default data collected are detailed in Section 4.3.2.

Alert fusion is the component that will perform the alert aggregation and meta aggregate data gathering. Its main functionality is to gather aggregate data from the alert stream and provide an interface where the similar alerts are combined together, based on their common traits. The data to aggregate are either derived from the rules contained in the Policy Decision Point in the case of dynamic aggregation, or when no rules exist using a pre-defined rules specified and executed by the default aggregation behaviour. The aggregation process and the default data collected are detailed in Section 4.3.2.

The component should satisfy the requirement posed in Section 2.4.2 in which the individual alert information should not be lost, and that the solution to be developed should also derive information from the alert stream. Without this component the Aggregation Module would not be able to provide the Policy Decision Point, with the information it may need to generate a decision. It gives the Policy Decision Point the complete picture about an alert instance.

The Alert Fusion component will include a Memory Container in which all this information is saved. This Memory Container should be accessible by other functional components contained in the Aggregation Module, and from outside which may need to form statistics about the alert, e.g. what attack classification is the hardest to react to with a specific action.

4.2.3 Action Inclusion

This component is responsible for handling the feedback provided by the PEP. Using this component the Aggregation Module can provide a list of actions that have been performed. As explained in Section 2.4.3 action that are received from the Policy Enforcement Points as feedback can be distinguished between actions that:

- Have been successfully Executed and Implemented (No error was generated)
- Failed to Execute or Implement (e.g. An error was generated)
- Actions that were never Executed

This enables the Policy Decision Point to revise and develop new strategies for dealing with specific attack instances. Together with the Alert Fusion and Stream Filtering components, it enable the Aggregation Module to inform the Policy Decision Point if an action which was successfully executed, had any effect on the intended targeted threat.

This component handles the feedback by fusing it together with the alerts that generated the actions in the Memory Container of the Alert Fusion component. It satisfies the requirement that the Aggregation Module should manage the feedback received from the Policy Enforcement Points. One of the main features of the Aggregation Module are the triggers. The triggers enable the *Aggregation Module* to forward alerts that either have yet to be dealt with. The feedback in cooperation with the Stream Filtering component, enables the Aggregation Module to access these triggers, since the main way to recognize if the Policy Enforcement Point have performed an action for a specific alert is through the feedback.

4.2.4 Dynamic Aggregation Rules

In order to simplify the definition of policies in the Policy Decision Point, the *Aggregation Module* provides an interface which is in charge of receiving the rules specified in the PDP as a configuration file.

To achieve this, a grammar is specified in Section 4.7. The main functionality of this grammar is to explain how entries should look like in the configuration file. Each entry represents one rule that is specified in the PDP. The transformation from rules to entries in the configuration file is not going to be detailed here, as it is beyond the scope of the thesis.

4.2.5 Component Order

Figure 4.2 depicts how the different components are placed together. *Stream Filtering* is the first component to receive an alert. The Stream Filtering component sets the rules for when an alert is forwarded. The *Alert Fusion* component is responsible to save the individual alert information and combine similar alerts together in its Memory Container.

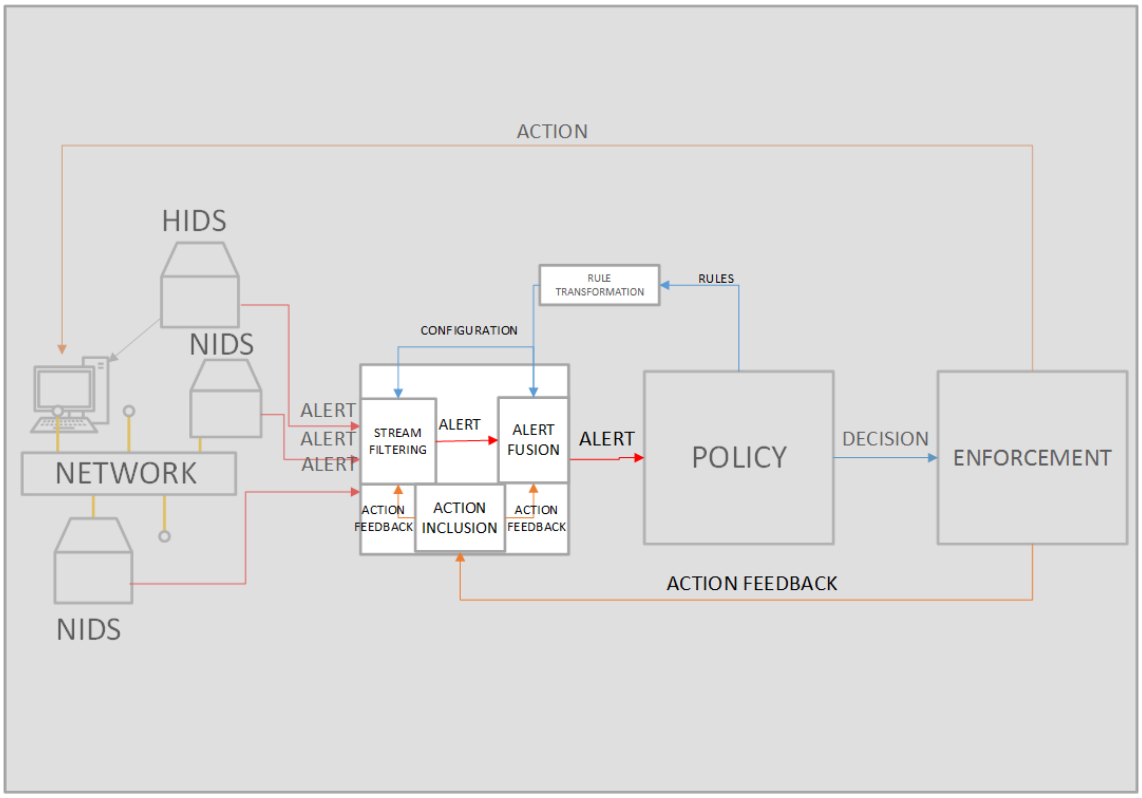

Figure 4.2: Functional Components

The Memory Container should be accessible from the other components, in order to have one central referral point for looking up if an alert was already seen by the Aggregation Module.

The feedback received by the *Action Inclusion* component, has to be forwarded towards the Alert Fusion component, in order for it to be saved into the Memory Container of the Alert Fusion component. The Stream Filtering component also needs to know about the feedback, in order to determine if an alert that was already seen is a *Repeated Alert*. Through the actions, the Aggregation Module can also provide triggers for the alerts in order for them to be forwarded which will be detailed in Section 4.5. In addition, if an action has successfully been applied to an event, this means that the underlying event which generated the alert has stopped existing, which raised an alert, and there is no similar alert produced by the same event in the near future, then the alert can be regarded as resolved and thus be removed from the Alert Fusions Memory Container.

Stream filtering and Alert Fusion form the basis of the forwarding logic. Alert fusion, combines similar alerts together and the Stream Filtering takes care of the alert to be forwarded. For this to work one of the two components must have a t Memory Container in order to remember information about the different alerts. The Memory Container is included as part of the Alert Fusion, where the Stream Filtering component can query for further detail if it has to forward an alert. This design decision has been explained in Section 4.2.2, and the Memory Container will be further detailed in Section 4.3.1 that follows.

4.3 Functionality Logic

The components represent functional components which are mapped to the requirements specified in Section 2. Below the functional components, the Aggregation Module has to implement some sort of logic in order to sort, combine and forward alerts. These in addition to questions that arise are handled in this section.

4.3.1 Memory Container

The Memory Container should be accessible to the different components of the Aggregation Module. Stream filtering should access the Memory Container in order to determine which alerts are to be forwarded, by comparing the alert received with any saved instances. The Action Inclusion should be able to access the Memory Container in order to append information about the received feedback to the appropriate alert.

The Alert Fusion component, as mentioned in Section 4.2.2 has to contain the Memory Container. The Memory Container is needed in order to save the first instance of new alerts. This is done in order to remember which alerts have already passed through the Aggregation Module, thus have already been seen by the Aggregation Module. In addition to this the Memory Container will serve as the location where the alert aggregation happens, new information is added to the alert instance saved or update.

The Memory Container serves also as the location in which the information about the feedback for the individual alerts is stored and where additional data from the alert stream is contained for each alert is contained. A special case for the Memory Container are the dynamic aggregation rules. For each rule that is derived, the Memory Container needs to add a bucket, in which the individual alerts matching these rules are placed inside.

4.3.2 Aggregation Process

The aggregation process details how the individual alerts are aggregated together. For the dynamic methods, aggregation data gathering is performed based on a-priori knowledge of the conditions defined in the PDP, the aggregation of the individual alert field is as specified in the default behaviour. In order for the Aggregation Module to aggregate alerts that are considered similar through the dynamic rules, the alerts have to be grouped together. This is done through the buckets located in the Memory Container of the Aggregation Module. The buckets each contain separate meta data in which the content to be aggregated are saved into.

For the default aggregation behaviour one of the core subjects in order to define how to aggregate is the representation of the alerts. The Aggregation Module has to provide default similarity and aggregation rules. For this three fields are to be considered here. The source which specifies the source(s) of the attack, the target field which specifies the target(s) of the attack, and the classification field, which specifies the type of the attack. The most important factor for similarity is the classification field, if the classification is the same then the attacks are more or less similar. If two alerts have the same target(s) or source(s) or both, it can also be said that they are similar [LeLN04].

For the default aggregation behaviour the Aggregation Module needs to also collect aggregate data, which serves the purpose of providing information which is not stored in the alerts representation and will be lost otherwise. The default behaviour should collect information about:

- The amount of alerts fused together
- The time when the first instance of the alert was received by the Aggregation Module
- The time when the last occurrence of the alert was received by the Aggregation Module

In addition, since one of these traits is always present in an alert, it even be said that this method would work for anomaly detectors. To aggregate alerts from anomaly detectors, one can use static time windows[MaMZ09] between the detection times. These will be pre-set time-windows in which an alert will be considered similar if it displays the same source(s) or target(s) or both.

When an alert arrives, it gets stored in the Memory Container, in addition the detection time specified in the common format will be saved as the time when the alert was first seen. The next alert arriving will be checked in the fields classification, source and target. If an alert displays the same classification following happens:

- The target(s), source(s) of the attack get combined together.
- For the aggregation data collected it will additionally increment a count which denotes how many alerts are combined together. Integer values in the alert are also displayed as an average over all alerts.

  For example if the first alert arriving has a severity of 1, and the subsequent similar alert arriving a severity of two, the average value between the two,which since it is always an integer value so in the example presented here the average is 1.5 and would translate to 2, then the fused alert will contain a severity of 2.
- The detection time of every last received similar alert is saved as the time the alert has last been seen.

If now an alert displays similar source(s) and/or target(s) they can be considered as not being similar. This happens because the modules duty is to aggregate alerts and not correlate them, since the classification points to the single attack instance detected. If a rule is contained in the PDP specifying that an alert displaying similar source(s) and/or target(s)is to be reacted upon, the Aggregation Module will treat the alert as similar but not as part of its default behaviour but rather through the dynamic aggregation methods.

The aggregation behaviour using dynamic rules, derive the data to be aggregate and the fields that produce similar alerts through a configuration file provided by the Policy Decision Point. Each alert passing through the Aggregation Module is first queried from the a-priori rules. The configuration received by the Policy Decision Point, also contains what aggregation data should be provided. For example if the Policy Decision Point contains a condition that says *Do action A after target Host1 has been attacked by 5 different attack instances* the a-priori method does the following:

- The target field is considered matching if it is host 1.
- The alerts information aggregated together are provided in the configuration file. In this case it would be the classification
- The meta data would contain the number of alerts received, and the time of first encounter plus the last encounter and in addition the amount of different classifications
- When the count of different classifications reaches the number 5, the alert will be forwarded to the Policy Decision Point

An exception to this rule are anomaly detectors. Since they do not provide a concrete classification but rather only refer to it as anomaly the default aggregation should treat them as a special case. When the classification anomaly is encountered, the main aspects of similarity would be the source(s) and target(s) contained in the alert. In this case a static or fuzzy set time window can be used as in [PoNe97] or [MaMZ09].

4.4 Feedback

Through the feedback, the Aggregation Module and the Policy Decision Point can detect in which state a specific attack instance is, and develop new strategies for dealing with the attack instance. Feedback also signifies a change in environment conditions, since for examples actions that have been unable to resolve a specific attack instance will not be considered for re-execution.

For example consider the sample network which is under attack from Figure 2.4, if an action has been performed, but failed to produce any results, the subsequent alert detected, states that the action has indeed failed. The Policy Enforcement Point can now implement another action which may produce results. This way the PDP and PEP can be more flexible in the face of attack instances which are difficult to resolve.

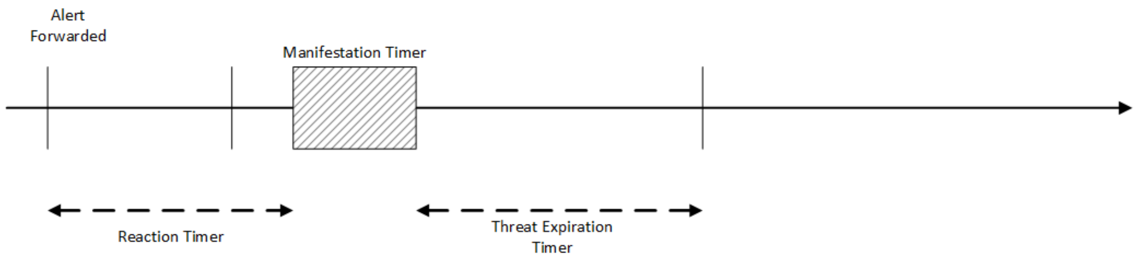

Figure 4.3: Timers

Reaction Timer: This timer is there to account for the time between when an alert has been forwarded and when the feedback from the Policy Enforcement Points is expected to arrive. Similar alerts that arrive in this period are blocked and stored by the Aggregation Module and are not forwarded to the PDP.

Manifestation Timer: After an action has been performed, there is a critical area in which the Aggregation Module has to allow the actions that have been performed, to take effect. For example if an action restarts a host, the Aggregation Module has to wait for the host to boot again before the action can manifest itself. Similar alerts arriving in this time period, are again not forwarded to the Policy Decision Point, as the event that generated the alerts do not carry a statement if the actions where indeed successful in resolving the threat.

Threat Expiration Timer: After an action has been manifested the Aggregation Module has to determine if the threat that was detected has been resolved. Any similar alert arriving in this time window is classified as a repeated alert. The repeated alert is forwarded to the Policy Decision Point together with the actions that have been performed to resolve the threat. When the timer expires the alert can be cleaned by the *Cleaner*. The cleaning process is detailed in Section 4.6.

4.4.1 Action Saving logic

The actions are split in three categories for all actions that can be successfully implemented against a threat:

- **Successful Actions:**

  Actions that have been successfully applied against a threat. This does not mean that the action successfully cleaned or blocked a threat it mainly symbolises that the action was successfully executed.

- **Failed Actions:**

  The exact opposite of a successful action. The PEP agent tried to implement an action but failed. There can be various reasons why the action has failed, for example that the threat is preventing the action from successfully executing. This has to be included in the feedback in order for the Policy Decision Point to know that the actions themselves could maybe alleviate a threat but were not able to be executed.

- **Reserved Actions:**

  These are the actions that have yet to be tried out by the Policy Decision and Enforcement mechanism. Since actions are implemented based on their effective cost, most of the time the actions included here are the more expensive solutions. They have to be included as part of the feedback in order for the Policy Decision Point to know what actions have yet to be tried out.

The feedback also contains the ID of the alert that has been responded to. This is important in order for the Action Inclusion component to append the action to that specific alert contained in the Memory Container of the Alert Fusion component. A fused alert contains the list of the actions as part of its meta data field.

4.4.2 Problems

One of the core problems in this approach is when no condition is defined, thus no action can be executed by the Enforcement Points. The default behaviour of the Aggregation Module tries to address this problem. By forwarding using the default behaviour, a triggered alert is always executed after a certain time window T or at an amount X of accumulated alerts as these variables are presented in more detail is Section 4.5. It is then up to the PDP to determine that something is wrong and some action has to be taken. Since no information is lost, the PDP should have enough data to execute a default routine in order to address the exception.

4.5 Forward Logic

In this section the co operation between the thee component is explained. The forward logic explains what alerts are forwarded and how this process determines which alerts have to be forwarded. The analysis is done based on the steps displayed in Figure 4.4. In order to filter alerts correctly while still being careful enough not to withhold operation critical information happens as a cooperative process between Alert Fusion and Stream Filtering for the initial alerts, and Action Inclusion and Stream Filtering for subsequent incoming alerts.

Figure 4.4 depicts how alert forwarding is handled by the Aggregation Module. It can be as in all cases in the aggregation be distinguished between the default and the dynamic behaviour of the Aggregation Module, even if Figure 4.4 combines the two behaviours.

Alert Arrives: In this part of the forwarding logic process chain, the alert arrives in the Aggregation Module and is added to a queue to be processed.

Information Saving, Memory and Enhancing are the parts of the Forwarding Process that can be directly mapped to the Alert Fusion component.

Information Saving: Here there are again two different approaches on what kind of information gets saved. The information that gets aggregated and how it is aggregated is the same as detailed in Section 4.3.2. If an alert instance does not exist yet, one is created in the Memory Container of the Alert Fusion component. For the dynamic aggregation rules, there is no need to create new entries since every rule has its own predefined bucket.

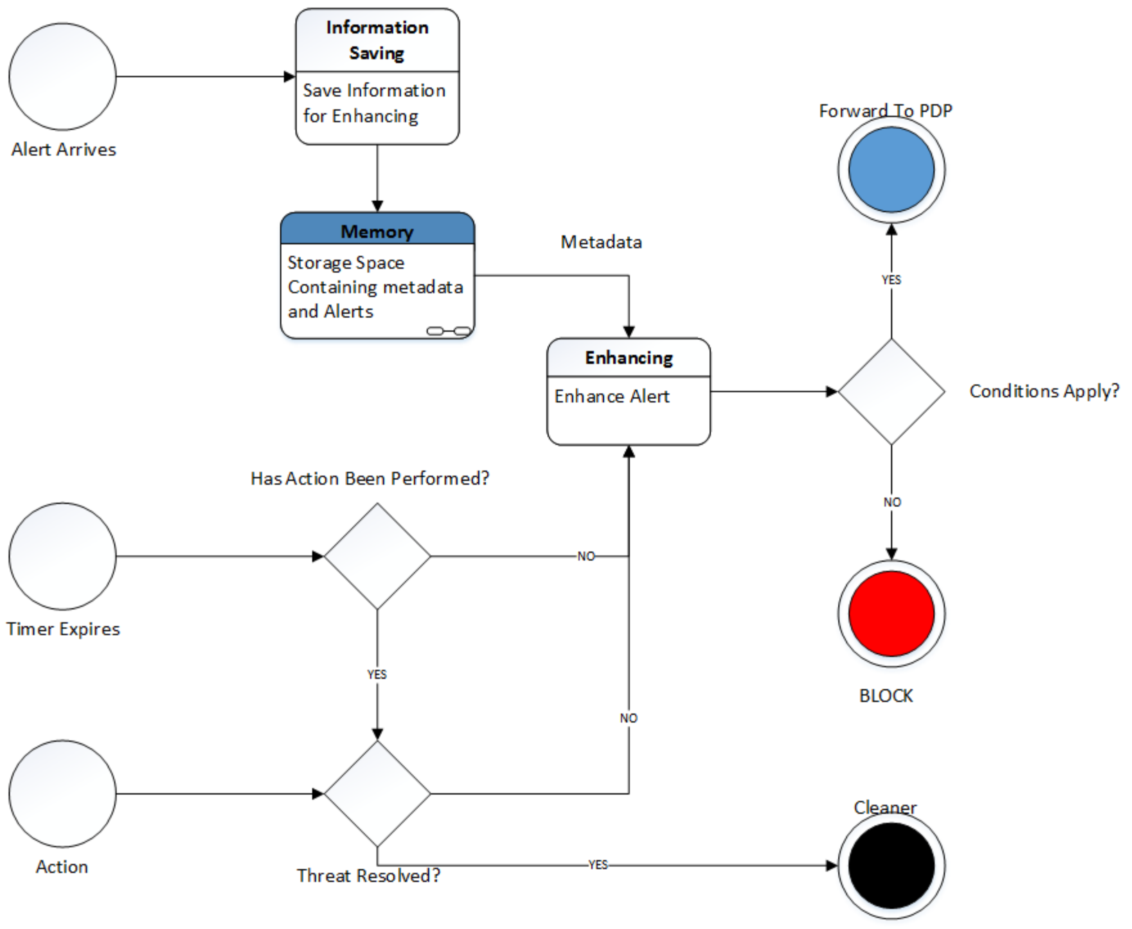

Figure 4.4: Module Forward Logic

Memory: This is the Memory Container as detailed in Section 4.3.1. It is their to store the information of the alerts, the fused alert components. Additionally it contains the buckets for the dynamic aggregation rules. The meta data transferred to the *Enhancing* part of the chain are the Meta data detailed again in Section 4.3.2

Enhancing: This is the sub process of fusing the alerts. This is again happening in the Memory Container, and follows the steps specified in Section 4.3.2. The enhancing process is the alert aggregation process.

Conditions Apply? If an alert is a similar alert the process checks the different types of alerts. The initial conditions are as follows:

- *Dynamic Aggregation Behaviour:* These conditions are directly specified by the conditions defined in the Policy Decision Point. An example would be if an alert contains the classification port scan to forward the alert when it occurs again in the next three days.

- *Default Aggregation Behaviour:* For the default aggregation behaviour, the initial condition for an alert to be forwarded is for the alert to be a new alert instance.

The initial conditions apply to alerts that have still not been forwarded or reacted upon. The forwarding logic has to also account for *Repeated and Triggered Alerts*. For this the Aggregation Module employees the three timers which can be visualised in Figure 4.3 and were explained in Section 4.4. These add additional conditions which are considered for alerts that already exist in the Memory Container and which have been forwarded to the PDP. This is a synergy between the Action Inclusion and Stream Filtering component.

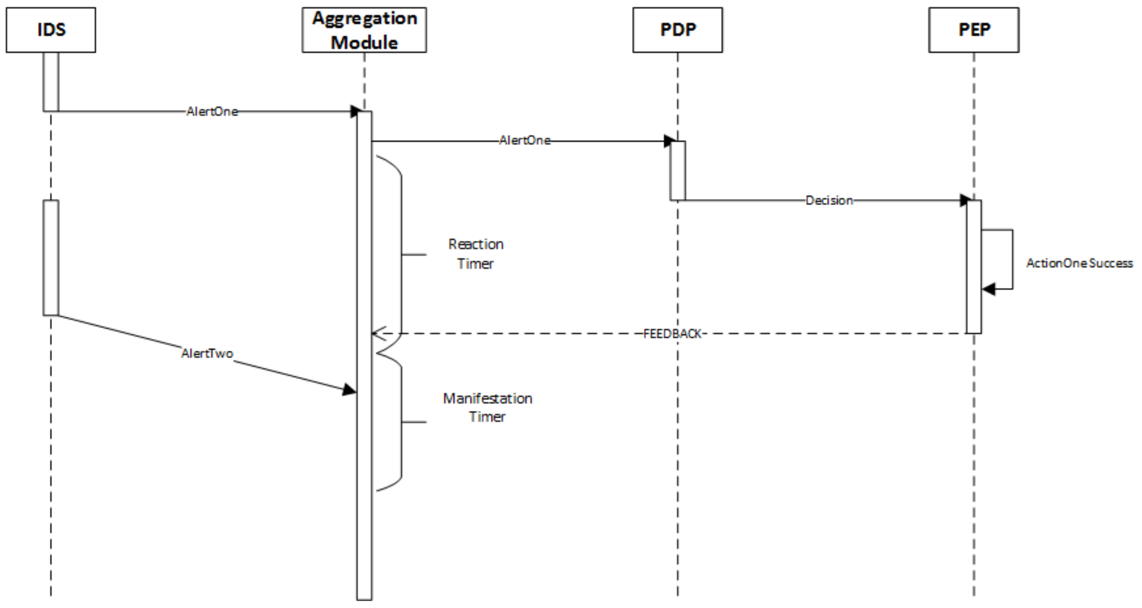

Figure 4.5: Timer Example

Figure 4.5 visualises an example to understand the logic of timers, and how they function in order to enable the Stream Filtering component to make a decision if an alert is going to be forwarded. An IDS detects an event, and for which it generates *AlertOne*. *AlertOne* is received by the Aggregation Module and forwarded to the PDP as defined by the default behaviour. The PDP generates a Decision which is forwarded to the PEP to be implemented.

The PEP generates an action called, *ActionOne* which was successfully implemented. Subsequently the PEP generates a feedback message about the action and forwards it to the Aggregation Module. Parallel to this the IDS generates *AlertTwo* which arrives at the Aggregation Module after the feedback. Since the Aggregation Module implements the manifestation timer, *ActionOne* is not considered to have failed, since it still hasn't manifested in the targeted Host. Subsequently AlertTwo is not forwarded to the PDP.

One of the problems to consider here is the time-delay which can occur between when an alert is sent by an IDS and when it should arrive at the Aggregation Module. Alerts have to be sorted based on the time they have been detected, in order to account also for the possibility of latency influencing the arrival time of the alerts. This is important in order to identify repeated alerts. The case of a repeated alert occurs only if the *Detection Time* contained in the IDMEF is after the *Implementation Time* of the action and after the *Manifestation Timer* has expired.

Through the timers, the Aggregation Module can implement the concept of triggered alerts. Triggered alerts can be used for the Dynamic Behaviour as well as the Default Behaviour.

Triggered Conditions : If an alert is not a repeated alert, the Aggregation Module needs to check if any trigger applies to the alert. There are two different cases to consider here. The default aggregation behaviour and the dynamic aggregation behaviour both use a timed trigger by employing a variable T, which is the same as specified in Section 4.4.2.

Variable T:

Denotes the maximum time that needs to pass without receiving feedback of a performed action. This variable reflects the *Reaction Timer*.

In addition to variable T, the default aggregation behaviour employees a variable X, and is also the one defined in 4.4.2.

Variable X:

Denotes the size, the amount of similar alerts that have been aggregated together, a fused alert needs to reach in order to be considered by the Aggregation Module as significant, should the Policy Decision and Enforcement mechanism not react at the first forwarded alert. This variable is needed in order to up the significance of an alert, even before a timer expires, in the case no immediate feedback is received, and the event that generated the first alert is generating a huge amount of alerts.

BLOCK: The last step in the forwarding chain. The alert can be safely considered a similar alert, and thus does not need to be forwarded. Block denotes the action of not forwarding the alert to the Policy Decision Point. In Section 2, it was defined that no information should get lost, and in addition the problem of blocking alerts was explored. The conception of the triggered alerts and timers, helps defuse this problem at its core. The Policy Decision Point will always get reminded about alerts which have yet to be resolved.

4.6 Alert Cleaning Logic

An alert can be cleared from the Memory Container after it has been marked as resolved by the Aggregation Modules components. The cleaning logic describes the process that should be used in order to free space in the Memory Container and to clean the Memory Container of alerts that have been dealt with.

The cleaning frees the space taken by alerts and saved meta data, which in turn enhances the performance when looking for alerts, since the *Stream Filtering* component has to go through fewer entries, in order to determine if an alert is new, similar or repeated.

In addition since saved alert information helps us to determine the state of the networks attacks, a saved instance that does not reflect the current situation should be removed as it provides inaccurate information. The cleaning also serves as an orientation point for the forwarding logic, since the look up for the entries is done on the Memory Container.

Again as with every process in the Aggregation Module,a distinguishment has to be made between the default behaviour of the Aggregation Module and the rule-based(a-priori) behaviour which is determined by the configuration provided by the Policy Decision Point.

Another distinguishment that has to be made is between an asynchronous and a synchronous cleaning function. The cleaning function can either run as a separate component, or as part of the Aggregation Module. Before introducing the logic to cleaning, the benefits and negatives of each approached have to be weighed.

Figure 4.6 depicts a graph which visualises the theoretical difference between a synchronous and asynchronous cleaner would have on operating memory when the Aggregation Module is under heavy load. The asynchronous mode would be able to deal with more alerts, since the Aggregation Module will have more processing time to operate.

In contrast using a synchronous process to analyse the Memory Container each time an alert is received, will certainly slow down the operation of the Aggregation Module, and take up resources, thus reducing the amount of alerts that can be processed. If a large

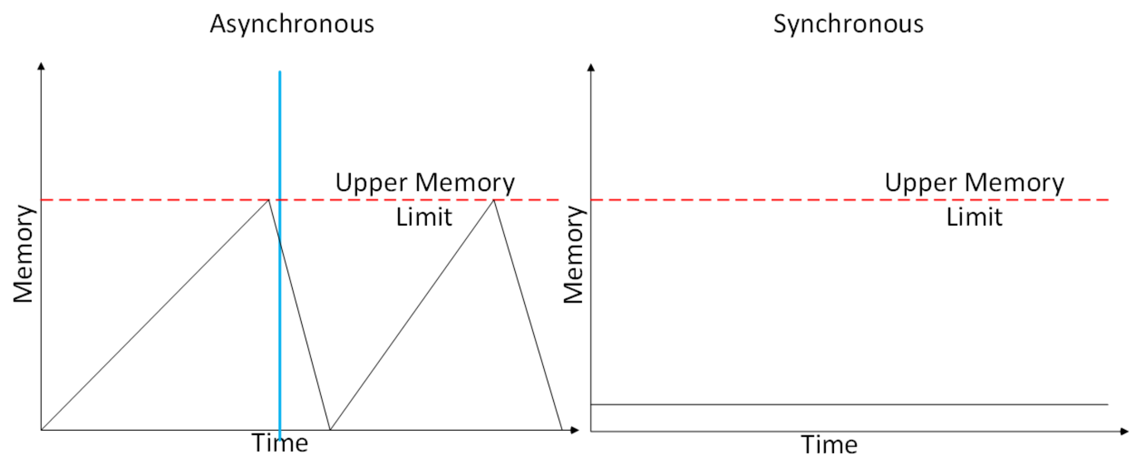

Figure 4.6: Theoretical Limits of A/Synchronous Cleaning on Memory

number of alerts arrive at the Aggregation Module, running synchronously would have the problem, that a lot of alerts may get discarded if the Aggregation Module reaches its maximum processing potential.

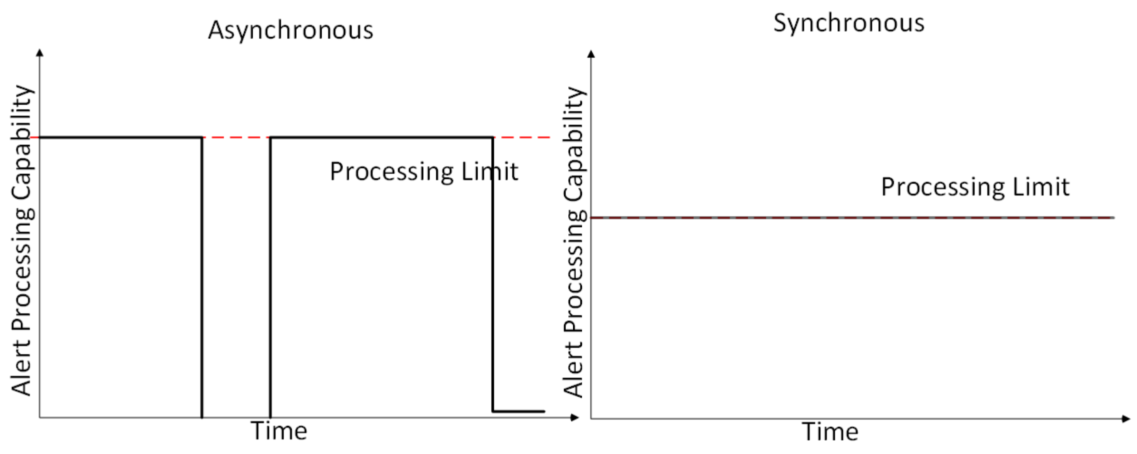

Figure 4.7: Theoretical Limits of A/Synchronous Cleaning on Processing

Although the asynchronous mode might have its benefits in the processing sector, there is a problem concerning the space the alerts would take in the Memory Container. Figure 4.7 depicts the theoretical memory consumption of the Aggregation Module for the asynchronous and synchronous cleaner, again the Aggregation Module being under heavy load.

If the cleaning function runs in synchronous manner, memory space is not a consideration, since the alerts are cleaned up immediately when they are due. In the asynchronous version, memory consumption is a problem, which has to be dealt with, since the space alerts take can exponentially rise when a substantial amount of new alerts arrive, since the function would run only in specific time-windows or triggers.

Another approach would be to combine the benefits of both. Since the cleaning logic has to clean alerts using two different approaches, the asynchronous and synchronous functionality can be implemented in another manner. For alerts that are saved using the default behaviour the Aggregation Module can use an asynchronous approach. A synchronous mode can be used to deal with alerts that are gathered based on the a-priori rules provided by the Policy Decision Point.

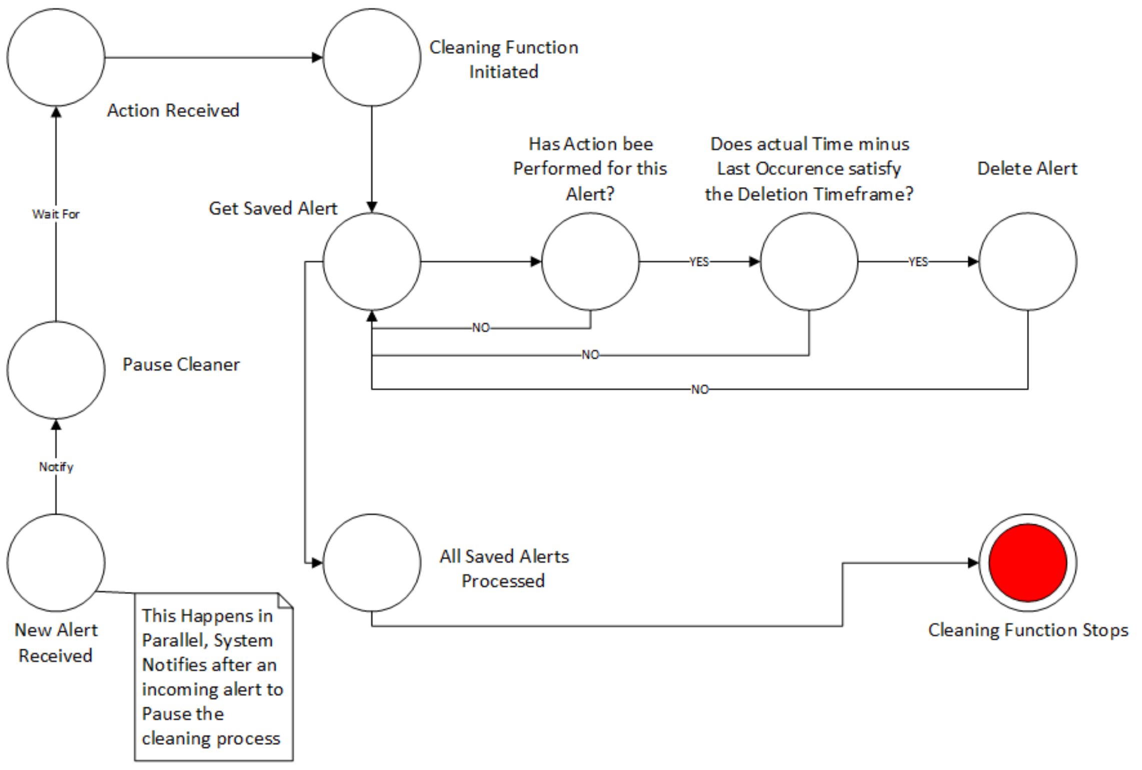
Cleaning Function Initiated
Action Received
Has Action bee Performed for this Alert?
Does actual Time minus Last Occurence satisfy the Deletion Timeframe?
Delete Alert
Wait For
Get Saved Alert
YES
YES
NO
NO
NO
Pause Cleaner
Notify
All Saved Alerts Processed
New Alert Received
This Happens in Parallel, System Notifies after an incoming alert to Pause the cleaning process
Cleaning Function Stops

Figure 4.8: Alert Cleaning Logic for Default Behaviour

Figure 4.8 depicts how the asynchronous functionality has to be implemented. In the default behaviour in order for an alert to be removed from the Memory Container following conditions have to apply:

- An action has been performed to deal with the alert: This means that no similar alert arrived after the action was performed, thus is not a repeated alert.
- For each alert a variable Y has to be set. Variable Y denotes a static time window. A static time window is the time in which two alerts are similar when they display a common trait [MaMZ09].

  In the case of the Cleaner, Y is the time variable between when an action has been performed, and the next arriving similar alert is considered a repeated alert. It reflects the sum of the Timers defined in Section 4.4.

From the first condition provides the trigger criteria for the cleaner to run as an asynchronous instance. Through the first condition it can be inferred that the cleaner should run only after an action has been received. If no action has been received there is no point for the cleaner to run since no alert would adhere to the condition, thus no alert needs to be cleaned from the Memory Container of the Aggregation Module.

In the case an alert is a match to one of the rules provided by the Policy Decision Point, the cleaner functionality has to be implemented in a synchronous manner.

In the case of dynamic aggregation rules the Memory Container contains a bucket for each rule. Buckets are needed in order to group alerts which match a rule together in order to perform the fusion process a-posteriori Each matching alert will be saved to the bucket corresponding to the rule which applies to its case.

The cleaner will check the bucket of this alert first, and then proceed to check further buckets. This way it is ensured that only the up-to-date necessary information is included in the fused alerts. The buckets are restricted in size by the rules, and the entries might also be restricted through. If for example the PDP only needs the alerts that come concurrently in 24 hours, all alerts that are expired and of no interest after their occurrence will be cleaned by the cleaning function.

The cleaner process stops processing in both cases when a new alert arrives. The methods above partly ensures that the Memory Container will stay at a stable level. In order to ensure though that there is always enough memory available, the operator of the Aggregation Module should set a thresh hold for the Aggregation Module, that initiates an non-interruptible cleaning process when memory consumption is nearing critical levels.

The caveat to this approach us that some alerts are going to get dropped during the non-interruptible cleaning process. One approach that could be used to avoid dropping alerts is to allocate some of the memory to a queue which would store the alerts for processing. This way the amount of alerts that would be dropped, could be minimized.

4.7 Dynamic Aggregation Grammar Description

R4 stated that the Aggregation Module should provide the Policy Decision Point the exact information it needs. The Aggregation Module has to have a-priori knowledge of what the Policy Decision Point reacts to. For this a universal grammar should be created, in which the rules specified in the PDP can be translated as aggregation rules into the Aggregation Module. This will grant the PDP more flexibility as it can define more flexible rules.

This section details the grammar to which the configuration files provided by the PDP has to adhere to. The grammar serves the purpose of defining a common format to witch policy specification languages can be translated. The configuration serves as a reference point to the dynamic aggregation behaviour of the Aggregation Module which is to be developed.

4.7.1 Concept

The grammar to be developed has to display some form of logic. The universal grammar is needed, since there are various languages in which a Policy Decision Point can be programmed. The configuration will be in context free grammar in order to be able to impose limits to the conditions, with respect to the Policy's conditions.

Reason for dynamic aggregation method:

- Ease in configuration of policy decision points, no special rules have to be specified to adhere to the aggregation the Aggregation Module would provide on its own
- No need to adapt the Policy Decision Point to the changing aggregation data collection or aggregation conditions.
- The dynamic derivation of aggregation rules enables content to be aggregated based on the policies that are defined in the PDP.
- If a rule is not specified, then the Aggregation Module aggregates using the default behaviour. If one exists then the exact information specified in the rule will be aggregated and forwarded to the PDP.

The rules are written in CFG notation and explain in the section that follow. The rules have to be provided in the following specific order:

$$\begin{array}{lcl} . & \rightarrow & AdivBdivC \\ A & \rightarrow & WindowRules \mid \epsilon \\ B & \rightarrow & FilterRules \mid \epsilon \\ C & \rightarrow & AggregationRules \mid \epsilon \\ div & \rightarrow & \_ \end{array}$$

Listing 4.1: Individual Entry Specification(CFG Notation)

The *div* terminal symbol is meant to divide the Rules. This has to happen in order for them to have a boundary which denotes when one of the rules end and another rule begins. ϵ denotes the empty set. The empty set is needed for when a Policy Decision Point has no such rule. In addition since the configuration file has to be a common format, ϵ is there to ensure that every rule is translate in the exact format provided by *AdivBdivC*, even if one of the entities is missing.

The Aggregation Module receives this rules as a configuration file, which generates aggregation methods based on the rules contained in the file. This enables the Aggregation Module to reduce traffic towards the PDP even more, since alerts that interest the PDP are forwarded when the conditions are an exact match. It also allows the operators to manage the network, without concerning themselves with adapting the PDP to changing aggregation rules. These rules will be provided in a file in which each line is a rule entry.

4.7.2 Semantic Order of an Entry

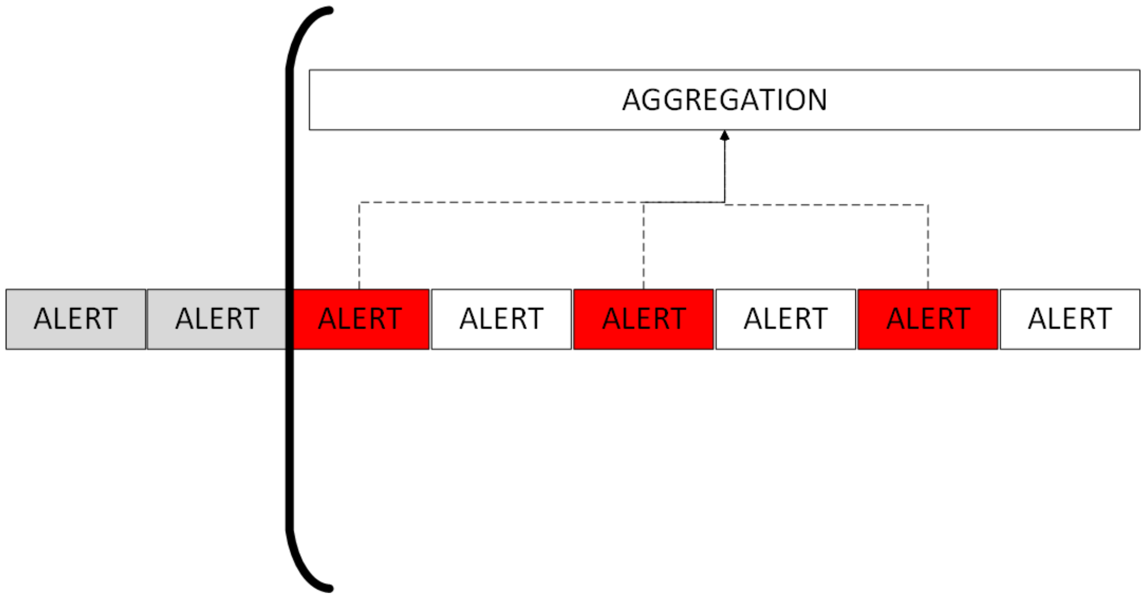

Figure 4.9: Alert Stream Filtering based on Policy Provided Rules

The entries contained in the configuration which is provided, must have a logical semantic order order. Figure 4.9 depicts an alert stream, which is arriving at the Aggregation Module. The first priority is to set a limit to the amount of alerts that are going to be processed, in the figure this is indicated by the black brace. The window set here, can be either based on time(for example every minute), or based on the a size of alerts e.g. first 100 alerts. The reason for first filtering the alert stream, is observational in nature.

```
WindowRules → T | S | T 'or' S
T → xTM
TM → 'd'| 'h' | 'm' | 's' | 'ms'
S → x
x → (1..9)(,)?(0..9)*
```

Listing 4.2: Window Rules CFG

If information is for example gathered in time, one can observe an alert stream every 24 hours in order to determine the impact the identified alerts have on the network.

After imposing a limit to the alerts, the Aggregation Module has to filter out the alerts that are of interest to the PDP, for the particular rule implemented. For example out of the 100 alerts contained in the window, the Aggregation Module is only interested in alerts containing the classification port scan. This has to be implemented in order to specifically target the alerts that satisfy conditions defined in the PDPs rule set, but before gathering any information about these alerts or fusing them together. Together with the time window defined above, if a rule exists that observes the alert stream in 24 hour periods, it can filter those alerts that are of interest in quantifying how serious a port scan attack might be.

After the alerts are filtered, aggregation data can be collected. For this any aggregate information needed to be passed to the Policy Decision Point has to be placed at the end. An example of aggregate information would be the count of alerts.

Continuing the example of the port scan classification, the PDP might want to count how many alerts have been identified to be containing the classification port scan. All three rules together now display a logical statement. An example would be that the Policy Decision Point, contains a rule that implements an action against a port scan if in 24 hours, alerts that display the classification port scan have been observed more than 10 times.

4.7.3 Window Rules

Window rules allows the Aggregation Module, to observe the alerts based on their occurrence in a network. For example the PDP can have a rule to react to a port scans. Port scans can happen in irregular time windows, but they do not have a high severity rating.

Because of this, the PDP might not react to the alert, since the severity is low and might be a false positive. By specifying for example a time-window, for example 3 occurrences in 3 days, the PDP can have a more flexible rule that it considers the port scan as high severity when it occurs in this way. The Aggregation Module would save the alert and discard it after 3 days.

Considering the above example the window rules has to consist of:

- **Size of the window in measurements of time**: The values has to specify a time window. An example would be 24 hours.
- **Numeric size of the window**: The amount of alerts contained in the window to be analysed.
- **A delimiter which allows the input of both values** This is needed in order to provide both values. The delimiter here should be an OR expression in order to achieve the FCFS logic in triggers

Listing 4.2 visualises the concept of Window Rules as a CFG. This is a continuation of Figure 4.1 from Section 4.7.1.

- **T**: Symbolises the time value.
- **S**: Symbolises the numeric length of the window
- **TM**: Symbolises the concrete time specification. d for day, h for hour, m for minutes, s for seconds, ms for milliseconds.
- **x**: Symbolises the numeric value of, it has to be at least one digit long. Terminal Symbol
- **T 'or' S**: In order to allow both arguments to exist the connector between them should be an OR statement. The reason for this is to adhere to the FCFS principle.

4.7.4 Filter Rules

The filter rules serve as a basis in responding to different attack scenarios. In the example of a port scan attack, of interest to the Policy Decision Point is to react when an port scan targets a specific *host*. When the alert is received by the Aggregation Module the Aggregation Module will see that the alert contains the classification port scan, and in the target field it contains host 1.

The filter rules have to include the following:

- Specify the fields of interest together with the value of interest (For example *classification = "port scan"*)

  The value should only be mapped with the following mathematical operands if its a number: **=, >, <, !=** . For strings, since the Aggregation Module will receive alerts that have been normalised as explained in Section 2.3, the mathematical operands **'=', '!='** should be used.

  In the case of strings, the mathematical operands denote string equality and string in-equality, and do not check for semantic or other equalities. This is again attributed to the fact that the alerts received by the Aggregation Module are normalised and pre-processed before being received.

  The dynamic aggregation method will quantify an alert as a match to the rules when all the specified fields contain data for which the mathematical operation or the string comparison provide the value of TRUE.

- Functions which operate on alert fields, which will produce a boolean value as their end result.

  This enables the Aggregation Module to include advanced functions. An operator might want to include custom functions in the policy conditions, which take have more advanced functionality than the plain mathematical functions. The function has to yield a boolean function in order for it to be compatible to the previously specified mathematical functions.

- Functions which operate on fields, but produce a number value as their end results.

  In order to use more advanced calculations, there should be an options to use a custom function. For example one could implement a function that assesses if an alert has a classification of 'port scan' and the severity is LOW, but contains several targets that are high assets to produce a value which determines the impact for the overall network.

```
FilterRules→ L
L → E | (L) | L CON L | not L
E → boolFunction | AnField SIGN y | AtField SIGNs
    s | numFunction SIGN y
CON → 'and' | 'or' | 'xor' | 'nand' | 'and not'
SIGN → = | > | < | !=
SIGNS → '=' | '!='
s → string
y → number
```

Listing 4.3: Filter Rules CFG notation

Figure 4.3 depicts the CFG notation of the Filter Rules. It is a continuation of Figure 4.3 in Section 4.7.1

- **L**: Denotes that more than one filter rule can be specified.
- **E**: Denotes a filter expression.
- **CON**: Denotes the allowed connections between the different filter rules.
- **boolFunction**: Denotes a custom function. This function is only allowed to provide a boolean value.
- **numFunction**: Denotes a custom function. This function is only allowed to provide a numerical value.
- **AnField**: Denotes the field of an Alert that contains an arithmetic value.
- **AtField**: Denotes the field of an Alert that contains a text value.
- **SIGN**: Denotes the allowed mathematical comparison between a field and a number value
- **SIGNS**: Denotes the allowed string operation between a field and as string value.
- **y**: Denotes the numerical value a numerical Alert field has to be compared with. Has to have at least 1 digit.
- **s**: Denotes the textual value an Alerts text field has to be compared with. Has to have at least 1 symbol.

AnField and **AtField** have to be considered separately, since they contain two different types of values. For the numerical values the Aggregation Module can perform 2 more mathematical functions, namely $>$ *and* $<$. Whereas the text fields of an alert are restricted to the *= and !=* operators. The Function has to yield a boolean value. The functions themselves have are to be restricted in operating only on alert fields. As stated above an operator might choose to perform more advanced mathematical or comparison functions which cannot be attained by the simple mathematical operands *=, !=, >, <* .

SIGNS symbolises this comparison functions, they symbolise lexicographical string operations, the Aggregation Module is allowed to compare to strings if they are identical, since the alerts that will be arriving will have been normalised and pre-processed. This means that for example a DDoS will always be displayed as a Distributed Denial of Service, rather than its abbreviation.

```
AggregationRules → H
H → N | H CONA (P) | H CONA H | (H)
N → (distinct)? aggrFunction
P → P CON Z | (P CON (Z) | P CON P
Z → (distinct)? aggrFunction SIGN y
CON → 'and' | 'or' | 'xor' | 'nand' | 'not'
CONA → ' || '
SIGN → = | > | < | !=
y → numbers
```

Listing 4.4: Aggregation Rules CFG notation

4.7.5 Aggregation Rules

The aggregation rules describe what aggregation data the Policy Decision Point needs in order to create a decision for an alert. The port scan example above can help to visualise this concept. The Policy Decision Point, as a continuance to the example in Section 4.7.4 contains a rule in which the Policy Enforcement Point will react if the port scan targets a specific host.

Since a port scan might also be a false positive, the PDP reacts only if the average severity of the alerts received equals 2 or if it has been received a total of 10 times. The counting measurement on how often the alert has been received is different to the one specified in the Window rules, since it is the count of filtered alert occurrences in the window specified above. The Aggregation Module will then count the amount of alerts contained in the bucket of the specific rule, and calculate their average severity. This data is then added to the meta data of the bucket.

The aggregation rules have to include the following:

Functions that yield numerical values

These functions operate on the total of alerts contained in a bucket for a specific rule. The Policy Decision Point can include functions that work on the entirety of the alert stream such as the count of the alerts contained in a bucket, or the count of individual analysers which identified an event.

- **H**: There are two cases of aggregation functions to be handled here
  - **N**: A single aggregation functions that only collects data
  - **P & Z**: An aggregation function which collects data, but also allows to filter alerts and forward them based on triggers
- **CON**: Same as CON from Listing 4.3 explained in Section 4.7.4
- **aggrFunction**: A data collection functions, which aggregates alerts or alert fields. An example would be 'count'. It can be differentiated between a distinct function and a normal function. As in SQL the distinct functions only counts unique occurrences of a field, entry or alert.
- **SIGN**: Same as in 4.7.4
- **CONA**: Symbolises the delimiter between different aggregation functions. For example count || average
- **y**: A real number

Aggregation rules provide two functionalities. The first functionality is to gather aggregate data for the meta data which is going to be provided together with the alert. The second functionality is to provide triggers for alert forwarding as specified in Section 4.5. An example of a trigger in this case would be *count = 10*. This is provided through the **P** letter of the grammar in Listing 4.4. It is different to the filters from Section 4.7.4 and from the window specification in 4.7.3.

4.7.6 Example Entries

In order to visualise the concept of these entries, some samples are provided here, the description specify the rule that is included in the Policy Decision Point. The rule is when the Policy Decision Point will be able to make a decision.

```
<empty>_target ='host1 ' and classification='denial
    of service '_distinct count(analyzerID) > 2
```

Listing 4.5: Denial of Service on Host 1, with more than 2 Alert Occurences

Listing 4.5, displays the entry in the configuration file where the Aggregation Module forwards the alert only if a Denial of Service is performed on Host 1, and it was detected by two IDS systems <*empty*> denotes the Window, as there is no need for one no window is specified. *target = 'host1' and classification = 'denial of service'* denote the filter rules, alerts matching this filter are chosen from the alert stream in order to be aggregated. *distinct count(analyzerID) > 2* denote the aggregation rules. The alert is only forwarded if the fused alert have been generated by two unique IDSs.

```
<empty>_distinct count(target) = 1_distinct count(
    classification) > 5
```

Listing 4.6: A single host targeted by more than 5 Attacks

Listing 4.6 depicts the entry that corresponds for a rule when a host is targeted by five or more attacks. <*empty*> denotes the Window Size, as there is none.*(*distinct count(target) =1) is the filter rule when the unique targets contained in the alert are not exceeding the number one. *distinct count(classification) > 5*, denotes the trigger condition for the aggregation meta data collected. The alert will be forwarded if the amount of unique attack instances performed on the targeted host exceeds the number 5.

```
5 m_classification='denial of service '_distinct
    count(analyzerID) > 4
```

Listing 4.7: Any denial of service targeting a variety of hosts. Identified by 4 or more analysers in the last 5 minutes

Listing 4.7 depicts the entry that corresponds to a condition that will implement measure if a DDoS attack targets a variety of hosts, and it is identified by 4 different IDSs. *5 m* denotes the window the alerts are chosen, this means that for every alert arriving in the last 5 minutes it will check if any of its rules match. *classification = 'denial of service'* denotes the filter rule, only alerts containing as classification DDoS are of interest here. *distinct count(analyzerID) > 4*, denotes the aggregation and trigger rule, the alert is only forwarded if this attack instance has been detected by 4 or more unique IDSs.

4.8 Summary

Chapter 4 provided a solution to the four requirements posed in Section 2.6.

- **Solution to R1**: Stream Filtering limits the number of alerts forwarded to the PDP by reducing duplicates.
- **Solution to R2**: Alert Fusion aggregates similar alerts, collects information from the data-stream and creates an alert that contains all the information collected.
- **Solution to R3**: Action Inclusion receives the feedback and appends it to the alert which caused the action.
- **Solution to R4**: The language defined in Section 4.7 takes care of dynamically providing aggregation rules to the *Alert Fusion* component via a configuration that is created.

The solution that is provided in this chapter should take care of the identified problems. As input the *Aggregation Module* takes the alerts that are generated by the IDSs and outputs it as combined alerts, and in addition appends additional information gathered from the alerts.

It receives feedback of the implemented actions, and forwards alerts when they are still representing not resolved threats to the PDP. This enables to monitor the state of threats in the network through the alerts that are generated.

5. Implementation

In Section 4 the concept of the solution to be implemented was presented. This chapter introduces the implementation of this solution. This chapter is structured in the following way:

- 5.1 and 5.2: Section 5.1 introduces the ESPER Framework used to implement the Aggregation Module functionality. Section 5.2 ANTLR is detailed which was used to implement the parser for the Dynamic Rule Generation. In the end of this section the reader should be able to understand how each framework is to be used to implement the functionalities defined in 4
- 5.4: This Section introduces the code created for the implementation. Since the implementation was not finished in time only minor parts of the code are going to be explained.

5.1 Introduction to ESPER

ESPER is a framework that implements Complex Event Processing(CEP). Event processing is a method which focuses in tracking and analysing large stream of information which contain information(data) about events that are going to, or have happened, with the purpose of deriving a conclusion from this information stream. CEP combines data from multiple sources. Its purpose is to understand underlying events or patterns that imply a more complicated conclusion than the original events. The final goal of CEP, is to explain and identify more elaborate events, for example a threat, and respond to them as quickly as possible[Luck12, Schm08, Schm11].

The events processed in CEP may happen across a variety of organisational layers. A core application field of CEP are brokerage house, in which stock market feeds are analysed in order to react to the fast changing values of stocks [Schm11]. The events analysed can also be called a "change of state" when measurements exceed predefined thresholds. Analyst imply that CEP introduces a new way to identify patterns in real-time, and can act as an enabler for better communication between businesses and changing IT and service departments [McKa09].

In the case of the Aggregation Module, CEP can be used to identify common patterns in Alerts. Since CEP relies on a variety techniques to infer the patters. These techniques are according to [EtNi10]:

- Event-pattern Detection
- Event-Abstraction
- Event filtering
- Event aggregation and transformation
- Modelling event hierarchies
- Detecting relationships between different events
- Abstracting an event-driven process

Since one of the main functionalities of the Aggregation Module is to filter and aggregate events, CEP is a suitable method to implement in the Aggregation Module. In addition since CEP can detect relationships between different events, it can also help detect the relationship between an alert and the feedback about the actions implemented for a specific alert. It automates the process of implementing an automation module.

There are a variety of solutions available, with one of the being ESPER. The reasons the ESPER framework was chose for implementing the Aggregation Module are following:

- It allows to process a live-data stream using SQL-Like Statements ESPER does not operate on a database, as SQL does, but it merely uses queries on data streams passing through its engine.
- ESPER does not need a database in order to keep track of events, thus simplifying the process of managing a memory container Since ESPER does not operate on a database, it also does not need one to save events. This alleviates the problem of managing a memory system, which are introduce for example when using a database.
- ESPER can be configured to automate the cleaning functionality of the Aggregation Module ESPERs can be configured with rules, which automate the cleaning of entries contained in its engine. This simplifies the process of creating a separate Cleaner.
- Since ESPER is a framework facilitating CEP it can detect patterns in events ESPER can be used to identify common occurrences in events. In the case of the alerts, the common occurrences are the similarities between the different alert fields.
- Allows to specify time-specific windows in which alerts are saved, enabling a more manageable timer environment A detailed explanation to this characteristic is provided in Section 5.1.2

The next Sections give a detailed explanation of the individual components employed by ESPER.

5.1.1 Event Objects

The first component of ESPER are **Event Objects**. Event Objects are wrapper Objects, which represent the Event Class to be analysed. Since the implementation of the Aggregation Module is going to be in Java, Event Objects in this case are Plain Old Java Objects (POJOs). The term POJO is used to indicate a normal Java Object with no special functions. This means they are only restricted to the definitions of objects in the Java Language Specification. A POJO does not extend pre-defined classes, does not implement any interfaces and does not contain predefined annotation. It is mainly composed of attributes, getters and setters. In ESPER POJOs are called Event beans [TeIn].

5.1.2 Event Processing Language(EPL)

The event processing language or EPL, is the CEP implementation of ESPER. It is an SQL-like language which includes a number of improvements, and where queries are so-called statements. The improvements over SQL are mainly event patterns and windows.

- **Patterns**:

  Patterns are a powerful tool which defines patterns between events in a language. For example, one can specify that a code segment is to run only if Event B occurs before an Event A. Consider following use case to understand the concept, a security system locks a user out when the user tries to type in his password(Event B), before using the biometric scanner(Event A).

- **Windows**

  One of the main differences of EPL to SQL. When ESPER queries a stream, as opposed to SQL it also considers time and space. This introduces a variety of applications, for example that a statement saves only the first 100 alerts that are a match to it.

An EPL statement is an equivalent of an SQL Query. ESPER contains a variety of objects which manipulate, control and react to statements:

- **Listener**

  A Listener in ESPER is an object which is initiated when a statement is activated. An active statement means that the query contained in the statement found a matching event object. Every statement produced by ESPER can be assigned multiple listeners, and a single listener can be assigned to multiple statements. Further detail is provided on the listener object in Section 5.1.3.

- **Subscriber**

  A subscriber object in ESPER is similar in functionality to the Listener Object. The main difference between the two constructs is that a subscriber receives the fields received from a statement have be statically programmed. It is faster than a listener object in processing event objects since the data a statement produces are immediately forwarded to the subscriber.

- **Event Processing Service Provider(EPService Provider)**

  The Event Processing Service Provider or EPService Provider is an object which configures the ESPER Engine. Each instantiated ESPER Engine contains its own Service Provider, in which the events to be processed have to be registered, and the EPL statements have to be declared and registered [TeIn].

5.1.3 ESPER Listener

As explained above, Listeners are objects which are instantiated when a statement is activated. This Section will further detail the functionality that Listeners can facilitate in order for the reader to understand how the ESPER Engine handles matching statements.

When a Listener is instantiated by a statement which he is registered to, it receives an update of the fields defined in a statement. A listener then can manipulate the data received, by invoking methods which process the received data. It can in addition interact with the ESPER Engine. This interaction is important, since listeners can forward manipulated objects again to the engine in order to invoke a more elaborate statement.

For example in the case of the default aggregation behaviour, an alert instance is cleaned by the Aggregation Module if feedback has been received for the specific alert and a predefined buffer period has passed. ESPER can contain four statements which take care of the above logic:

- *StatementOne*: In charge of analysing the event stream to find common fields in alerts.
- *StatementTwo*: In charge of analysing the alert stream for feedback matching alerts contained in the engines memory.
- *StatementThree*: This statement is a pattern which clears an alert for cleaning if feedback for the alert has been received.
- *StatementFour*: This statement cleans an alert instance and its feedback out of the engines memory when certain pre-defined time limits have passed

For the statements above ESPER has to contain three listeners:

- *ListenerOne*: ListenerOne is directly registered to *StatementOne.* This listener fuses similar alerts together, and has to forward this alerts to *StatementThree* in order to activate the Pattern.
- *ListenerTwo*: ListenerTwo is directly registered to *StatementTwo.* It is in charge of forwarding the feedback to *StatementThree* in order to validate the pattern defined in *StatementThree.*
- *ListenerThree*: ListenerThree is directly registered to *StatementThree.* Its main functionality is to forward the alert together with its feedback to StatementFour in order to activate the cleaning timer.

The registered listeners are instantiated as defined in 5.1.2 only when the statement is activated to which they are registered is activate. With the example above the reader should now be able to understand the operational logic of Listeners, Statements and Patterns, and how these can be used to implement the logic defined in Chapter 4. For more elaborate examples to further understanding the reader should visit [ESPEa, ESPEb, TeIn].

5.2 ANTLR

Another Tool for Language Recognition or ANTLR, is a parser generator framework. ANTLRs input is a Context Free Grammar, expressed using the Extended Backus Naur Form (EBNF), to specify a certain language pattern. It will then proceed to generate a recognizer for the specified grammar, which will read the input stream and in the case of non conforming entries it will generate errors. Non conforming entries, are the entries that do not follow the explicitly defined syntax form. The default action if no syntax errors are found is for the program to exit. Custom actions can be defined for the elements of the grammar, and will be automatically executed when an entry is identified which contains the conforms to the syntax defined in the grammar element.

5.3 Core Implementation Concept

This section maps the different Frameworks and Framework Components to functionalities defined in Chapter 4.

ESPER will receive the alerts as event beans. A matching event will be forwarded to the ESPER engine in order to be processed by a listener. Depending on whether the default behaviour is to be engaged or the dynamic behaviour the events are stored in its dedicated window. After that phase the last listener will forward the fused alert to the Policy Decision Point. The default behaviour statement will be mapped to a subscriber since the fields that are received from the engine are defined a-priori.

ANTLR will implement the grammar defined in Section 4.7. ANTLR will generate the parser and lexer in order to generate EPLStatements based on the input received by the configuration file. The statements will be mapped to a listener, since the content of the alerts and the statements are not known a-priori. For the implementation the only aggregation functions that will be allowed are the count,average, minimum and maximum statements.

Alerts are received normalised in the format specified by the IDMEF specification in 2.1. In order to provide flexibility the received alerts are going to be transformed to an Event Bean. For the implementation the event bean will only contain the classes *Analyser, Target, Source and Classification.* The actions are going to be implemented as their own Event Bean in order to be able to simulate the Action Inclusion Component.

5.4 Code Analysis

Since the implementation was not finished in time, there is no code analysis to be made.

5.5 Summary

This chapter provided an overview over the framework the *Aggregation Module* was to be implemented with were presented. The implementation framework can be mapped as follows considering the solutions presented in 4.8:

- Solution to R1,R2,R3: These parts of the solution can be mapped to ESPER. **R1** gets implemented through the patterns that can be defined in ESPER, **R2** can be implemented through the different EPLStatements and listeners, and **R3** can be implemented again through an alternative event stream which utilizes EPLStatements to aggregate alerts with their actions.

- Solution to R4: The custom grammar defined in Chapter 4 can be implemented by implementing the same grammar into ANTLR. By mapping the different grammar definitions to generate EPLStatements, ANTLR provides the best solution to implement the dynamic alert aggregation.

Additionally this chapter explained how these frameworks interact and why the frameworks were chosen for this implementation. No code was explained, sine at the time of writing no implementation was ready.

6. Evaluation

6.1 Evaluation Overview

Section 4 detailed how the solution should be implemented in order to alleviate the problems identified in Section 2. This Section describes the methods to be used, in order to identify if the proposed solution has indeed alleviated the problems and at what trade-off theses problems have been solved. First, an overview the identified problems is presented. The problems are connected to the different components of the solution, and the criteria for evaluating the solution are introduced. The subsequent Sections detail how the solution should be tested.

6.1.1 Evaluation Target

Section 2 identified following problems:

1. **Alert Stream**:

   It has been identified in Section 2 that one of the core problems of having a high number of heterogeneous IDSs in the system is the generation of many alerts. A number of these alerts are a duplicated of each other, since they either contain identical information or are describing the same event.

   To counter this in 2 a solution is presented which is implemented as part of the *Aggregation Module*. The solution is called **Stream Filtering**. This components functionality is detailed as part of Section 4.5. It is responsible for reducing the alert flow, by forwarding only alerts that have to be forwarded. The advanced forwarding functionality, e.g. the triggers and timers being set, is based on the synergy of all three components.

2. **Information Completeness**:

   Filtering the alert stream introduces a sub-problem. As detailed in Section 2.4.2, by blocking Alerts some of the information might get lost, since heterogeneous IDSs contain different type of information. In addition based on the location of the IDS on the network the Alert generated may contain more information about the event that generated the alert.

   To counter this, the *Aggregation Module* implements the component called Alert Fusion detailed in Sections 4.2.2, 4.3.1, 4.3.2 and 4.5. The Alert Fusion component aggregates alerts based on two behaviours the default behaviour and the dynamic behaviour. The default behaviour is in-place in order to guarantee that alerts are fused together based on commonly found traits. In the dynamic aggregation behaviour the engine combines alerts based on a-priori provided configuration which is derived of the rules contained in the corresponding Policy Decision Point. It pertains to-be resolved alerts in its memory, and derives information from the alert stream in order to display the larger picture relating to a specific event. It is the solution for the problem of loosing information and is used towards reducing the stream of alerts that reaches the Policy Decision Point.

3. **Missing Feedback**:

   One of the core problem that this thesis addresses is interaction missing between the Policy Enforcement Points and the Policy Decision Point, or administrator. Detailed in Section 2.4.3, and illustrated in Figure 2.3 there is a lot of insight that can be gained by enabling communication between the actual enforcement of actions and the Policy Decision Point. One example is to determine if a specific action was successful in resolving a detected attack instance. To alleviate this problem the *Aggregation Module* implements the component with the name **Action Inclusion**. This component intercepts the feedback which is provided by the Policy Enforcement Points. The function it provides is detailed in Section 4.4. This component allows the security systems to understand if a measure that was implemented successfully resolved a detected attack instance. This should provide a solution to the requirement of missing feedback.

4. **Dynamic Aggregation Rule Generation**:

   One of the minor, yet important problems identified was the need to adapt the Policy Decision Point to the content generated by the *Aggregation Module*. To solve this a grammar was developed in Section 4.7 which is read by the Aggregation Module as a configuration file. Each grammar entry denotes a rule defined in the Policy Decision Point. It is in its basic form an interface with which the different policy languages can be translated in order for the Aggregation Module to generate custom rules based on the data the Policy Decision Point expects to receive.

6.1.2 Evaluation Criteria

Before evaluating the Aggregation Module, there have to be base criteria in which the tests to be created have to be oriented upon. The first criteria is the modules effectiveness, the question to be asked is whether or not the Aggregation Module works as intended. To achieve this each component of the *Aggregation Module* has to be tested. This criterion will be called **Efficacy of the System**.

Stream Filtering has to be tested in order to verify that the alerts are forwarded according to the rules that were specified in 4. **Alert Fusion** needs to be tested on whether or not it aggregates Alerts correctly. This means it identifies and fuses similar alerts together. In addition the dynamic aggregation has to be tested in both cases in order to determine that it functions as intended.

After determining the functionality of the Aggregation Module, the second concern is the performance. Since the Aggregation Module is added between the alerts and the PDP it will definitely add some latency from the time the alerts were intended to arrive, and when they will arrive. The performance has to be investigated in order to determine the trade-off the theoretical benefits of the Aggregation Module come with. In addition since the Aggregation Module will be part of a larger system, it needs to be investigated if the resources the Aggregation Module is using are kept low enough. This criterion can be called the **Efficiency of the System**.

The last criterion to be considered is the overall security of the Aggregation Module and its individual components. Each component has to be tested in order to determine if the Aggregation Module adds additional attack vectors to the Aggregation Module. This is important, since the Aggregation Module operates to enhance the efficacy of an existing security solution. By decreasing security, it would also undermine the benefit of having the *Aggregation Module* in the network.

6.2 Conventional Testing Methods

To test the Aggregation Module, the Intrusion Detection Systems in the network have to be stimulated by generating events. These events have to malicious actions which will cause the Intrusion Detection Systems to generate alerts. There are two different approaches for generating events [ZhGh05, MaMZ09].

The first approach is by using pre-generated data sets, that come in the form of network traffic dumps. Pre-generated data sets are observations of network activity, which contain malicious attacks that will subsequently generate cause the Intrusion Detection Systems to generate alerts, based on the activity that is detected. This is one of the common testing methods in the field, since it offers a simple way to stimulate the Intrusion Detection Systems.

For the pre-generated datasets, there exists a number of them that are being used in the research field. The most notable are the DARPA 99 and 200 data set, and the DEFCON CTF dataset. DARPA 99 and 2000 are a dump of network traffic, they contain following attack steps:

- An attacker breaks into a host computer
- After the attacker gains access to the host computer, he proceeds in installing malicious components which are needed in order to launch a DDoS(Distributed Denial of Service) attack.
- The DDoS components are used to attack an off-site server located outside of the network.

The DARPA 2000 set contains 2 versions of the said attack, with one of them being a more sophisticated version of the attack described above [ZhGh05, MaMZ09]. The DEFCON CTF is a collection of network traffic recorded in a DEFCON convention Capture the Flag tournament [MaMZ09].

Although the pre-generated datasets are flexible enough, they include a number of problems. As explained by [MaMZ09] the data sets themselves are not sufficiently describe, and not enough information exists about what exactly is contained in those dumps. An additional problem is that a network containing a considerable number of different Intrusion Detection Systems, is difficult to test since every Intrusion Detection System will generate another description for the attacks or may not be able to detect some of the attacks. Even if researchers tried to asses the validity of the datasets, the datasets themselves contain several inconsistencies which make it difficult to asses if the data is indeed real network traffic.

From the pre-generated datasets the most favourable is still the DARPA sets. This is due to them being thoroughly researched and documented and obtainable free of charge

[MaMZ09]. Still one of the problems that can be detected here, is that the datasets are outdated stemming from the early 2000's which is hardly identical to the attacks that can target modern systems.

The second approach is to simulate attacks. This process has to be recorded in order to be reproducible for both cases, the one without the Aggregation Module and the one with the Aggregation Module. In order to simulate attacks, background research has to be conducted in order to identify which attacks the Intrusion Detection Systems employed in a sample network are able to detect. Since the Common Vulnerabilities and Exposures database(CVE) contains reported exploits, it should be used to identify potential attacks that can be used.

For the purpose of the evaluation of this Aggregation Module, the second approach would be the most preferable. The second options allows us to generate and document the attacks performed on the network, and also to simulate the Scenarios defined in Section 6.3. This will allow the Aggregation Module to be tested exactly as stated above, and the research being able to produce clear results of whether or not the functionality of the Aggregation Module is complete.

6.3 Evaluation Scenarios

Before describing concrete testing environments and methods, some scenarios have to be developed in order to derive test-cases out of them. The following scenarios are abstracted concepts, meant to reflect the situations which are the most or least favourable towards the *Aggregation Module.*

- Scenario No1: Huge number of different Alerts, with no common traits.

  This scenario is the worst case for the Aggregation Module. This case can demonstrate the performance and stability of the Aggregation Module under the heaviest load possible. The Aggregation Module will be occupied the whole time. This is characterized by the constant lookups the Aggregation Module has to perform on its memory in order to find similar alerts. This will demonstrate that the modules functionalities can work in a stable manner together. It will also show if the Aggregation Module can correctly aggregate based on the dynamic rules provided. This scenario can be used to measure the performance and efficacy of the Aggregation Module under heavy load.

- Scenario No2: Huge number of different Alerts, randomly similar alerts.

  This scenario should demonstrate the theoretical real world capability of the Aggregation Module. As with scenario No2 this will demonstrate the stability and performance of a Aggregation Module in a ”real world” situation. It will also help understand if the default and dynamic aggregation behaviour, correctly aggregate events.

- Scenario No3: Medium sized alert stream, which contains a large number of similar alerts.

  This Scenario has to evaluate the benefit of the Aggregation Module. It answers the questions such as, does the default behaviour aggregate correctly, how many alerts are aggregated correctly(in %), what the error rate aggregation is. In addition it displays how well the default behaviour operates. In addition through this scenario, the feedback received by the Aggregation Module can be quantified. This scenario can be used to test if the Aggregation Module operates as intended. The amount of exploits used can be kept at a low range(3-4 exploits).

In 6.2 the two testing methods were explained, and method two was chosen as a way to generate alerts. Together with the scenarios described here, testing cases can be constructed. These will be detailed below together with the data that needs to be extracted in order to understand. The tests will be performed on a test-bed containing networked hosts and heterogeneous IDSs.

6.4 Experiment Environment

The environment for the test, will be a virtualized network, with no prior traffic. The system will have 3 to 4 hosts, and 2 to 3 heterogeneous IDSs with overlapping domains. As mentioned in 6.2 for the evaluation the approach chosen to generate alerts is to simulate and document attacks.

In the pristine network, without including the *Aggregation Module*, the first step is to start recording network traffic and performing a number of exploits in order to gather data. The data that needs to be collected here is as follows:

- Details of the Exploits:

  One of the most important factors to grant us the ability to test is repeatability. Every attack performed on a host or network in the virtual space has to be documented. The most important data is the exploit name, the target machine and the time frame it was performed. In addition document if the exploit was resolved by the traditional system in place.

- Initial Alerts:

  The initially generated alerts have to be collected. This helps identify which IDSs could detect the exploit, and what information the IDS provide in the first place. In addition the total count of generated Alerts has to be collected, which indicates how many alerts have also been received by the Policy Decision Point.

- Record network traffic:

  The actions that have been performed, exploits and the network traffic being scanned by the IDSs has to be recorded. This ensures that the experiment can be replayed with the *Aggregation Module* in place, with everything else kept the same.

This has to be performed in three different scenarios which are explained in 6.3. After collecting the data, it can be replayed into the network in order to simulate the above scenarios every time a function has to be tested.

6.5 Efficacy

6.5.1 Alert Stream

By replaying the data set collected in 6.4 into the network, since every attack step is documented a test-case can be created in which it is observed if any Alert forwarded from the *Aggregation Module* to the Policy Decision Point is a duplicate. If no significant number of duplicated Alerts are forwarded to the Policy Decision Point, and the number of alerts received by the Policy Decision Point is reduced then it can be concluded that the desired functionality for this component has been achieved.

To test the functionality, the first step is to use the Data-Set collected for Scenario No3. Scenario No3, as explained in 6.3 is the ideal scenario to test if the system is operating as intended. Following data has to be collected:

- *The amount of alerts forwarded*:

  Using this number it can be verified if anything happens at all. If the number of forwarded alerts is significantly lower than the alerts generated in the initial data-set creation then the Aggregation Module does filter alerts and reduces the alert flow.

- *Forwarding of appropriate Alerts*:

  Forwarded alerts have to be inspected in order to determine the contents of those alerts. Since the attack steps and attack instances in the system were documented as part of the data generation process, it can be safely determined which alerts should have been forwarded and were not, which alerts should not have been forwarded but were forwarded. if the alerts that *should have been forwarded but were not* is an insignificant amount, then the functionality of the Stream Filtering component is verified.

The results expected here would be that the alert reduction is significant and that the amount of not forwarded alerts is insignificant or near zero. If the outcome of the tests verifies our expected results then it can be safely said that the *Stream Filtering* component functions as intended.

6.5.2 Information Completeness

To determine the efficacy of implementing information completeness, the **Alert Fusion** component and the **Cleaner** have to be evaluated. Again the dataset from 6.4 which corresponds to Scenario 3 is to be used. The dataset has to be replayed into the network, and following data has to be collected:

- *Percent of correctly aggregated alerts*:

  By inspecting the Alerts contained in the memory of the *Alert Fusion* component, the number of aggregated alerts can be determined. From there by inspecting the contents of the alerts it can be seen which alerts from all the aggregated alerts contain information that was intended to be fused together. This metric will reflect the efficacy with which the Alert Fusion component aggregates alerts, and also answer the question if it aggregates any alerts.

- *Up-to-date alert memory*:

  In Section 4.6 it has been detailed how the cleaner influences the state of alerts contained in the memory, and the importance of its function. Part of the information completeness is to deliver the correct information to the Policy Decision Point. As such it has to be tested if alerts are removed from the memory when the attack instances that were described expired. This can be derived by inspecting the memory of the *Alert Fusion*, since the resolved attack instances were documented as part of the dataset generation.

The expected result is that the aggregation process produces a significant number of correctly aggregated alerts, and that expired alerts are cleaned when the attack instance that produced them expires. This results will prove that the functionalities of providing information are indeed working and that the Aggregation Module produces the desired output.

6.5.3 Feedback

The efficacy of the *Action Inclusion* component can be determined through the question:

Do attack instances that were previously unresolved get resolved?

In generating the test-data, exploits that were successfully resolved were documented. If yet unresolved attack instances are resolved by the Aggregation Module i.e. they do not generate any more alerts after a measure has been invoked, then it can be concluded that the feedback produced an environment that allowed alternative actions.

If the desired results are produced in this test, then it can be concluded that the feedback produces the desired effect. The environment changed and a new set of actions can be chosen to be implemented.

6.5.4 Dynamic Aggregation Rules

This can be tested by crafting custom rules in the policy corresponding to the exploits used in the data generation. This can be tested with any dataset, for which the dataset for Scenario No3 is actually best suited since it contains a lower number of alerts. Data can be obtained here by inspecting Alerts forwarded to the policy. If the Alerts correspond to the rules that were provided through the configuration file it can be safely said that dynamic aggregation indeed works as intended.

6.6 Performance

Performance here are two metrics:

- **Resource Consumption**:

  This metric is defined from the Memory Usage and the Processing Power Usage. In the case of the *Aggregation Module*, the maximum resource consumption in a worst case scenario has to be investigated. This also researches the stability of the Aggregation Module under duress.

- **Processing Time**:

  This metric can be defined from the time an alert enters the *Aggregation Module* until the alert is forwarded or blocked. This metric reflects the trade-off to the benefits gained by the functionality of the Aggregation Module. Since the Aggregation Module is between the alerts and the PDP, it can be safely assumed that the alerts will arrive later at the PDP which might influence the reaction cycle of the security system, since actions would be implemented at a later stage, than without the Aggregation Module. If the processing time, is significantly low then the Aggregation Module does not influence the security system negatively.

- **Percent Alert Reduction**

  This metric should derive itself from the *Total Alerts Generated* and the *Amount of Forwarded Alerts*. Dividing the second with the first, should yield the percent amount of reduction. It is considered a metric here, since it states how much of a performance benefit the duplicate eliminations has brought(the PDP processed fewer alerts, fewer actions have been executed) to the system as a total.

6.6.1 Alert Stream

In this phase the average processing time of alerts is calculated before they are forwarded. The data-set that should be applied here is the one derived from Scenario No 1 & 2. The average processing time of each scenario, when added together, gives the overall average processing time for heavy load with no processing and heavy load with processing. It can be assumed, that if the overall average is significantly low, that the trade-off of implementing the Aggregation Module is also significantly low and will not impact the day-to-day operations of the Aggregation Module.

6.6.2 Information Completeness

For this the *Alert Fusion* component will be tested again. The datasets from Scenarios No1 & No2 should bring the Aggregation Module to its limits. In these scenarios following data should be collected:

- Processing Power Consumption: Important since the Aggregation Module is a sub Aggregation Module of a larger engine. The observed processing needs of the Aggregation Module give a detailed picture of how many resources the system will need, in order to perform its job. The ideal case is that it consumes as less as possible although if the processing needs are not significantly large it is safe to implement the *Aggregation Module* without considering processing a cost.
- Memory Consumption:

  Another important factor for the Aggregation Module to be labelled efficient, is the memory consumption. Ideally memory consumption should be also as low as possible. The *Aggregation Module* should also not consume more memory than the one available since this would classify the Aggregation Module as unstable. This can be observed throughout the performance of the dataset.Scenarios No1 & 2 should be used to determine the stability of the Aggregation Module and its memory consumption under full load. In addition Scenario No3 should be also used at this stage in order to determine the average memory consumption in a scenario which does not produce a full load.

6.7 Security

Security is an important aspect for evaluating the *Aggregation Module.* Since the Aggregation Module will be placed as part of a more elaborate security system, it should not be one of the attackable vectors. Since this is not part of the thesis only the what can be done in order to evaluate will be described.

6.7.1 Alert Stream

The Stream Filtering component, affects which alert is forwarded or not to the PDP. If the Stream Filtering can get exploited to forward false alerts to the module it could undermine the safety of the system. To test if this is the case, false alerts should be generated and forwarded to the Aggregation Module, in addition to the alerts that are generated by using Scenario 3. The alerts should be constructed in such a way that the Stream Filtering component will always forward them, a possible approach to achieve this is to generate alerts with different classifications every time.

The desired observation would be that if accidentally false information is forwarded, the normal operation of the PDP will not get hindered, and that the bogus alerts do not block actions from being executed.

6.7.2 Information Completeness

A core security problem would be the memory. The memory can be clogged with spoofed alerts, making it hard to process or even cause the system to become unstable and terminate. This is not a behaviour that is requested from the Aggregation Module, because this would render the Aggregation Module not operational, and thus not forward security alerts to the PDP. The ideal way to test this was described in Section 6.6.

By using Scenario No 1 and No 2 one can determine if the Aggregation Module will stop functioning if it receives more load than it can process. In addition it will test the capability the cleaner function has to intervene and reduce memory consumption.

The desired and result here would be that the memory itself, cannot be exploited to terminate the Aggregation Module.

6.7.3 Feedback

The **Action Inclusion** component can be misused in order to provide false information to the Aggregation Module. Here custom fake feedbacks should be generated corresponding to one of the initial alerts that were saved. The desired result is that the system will not be influenced by the fake message, since Alerts are not marked as resolved when feedback is received.

6.7.4 Dynamic Aggregation Rules

In order to generate dynamic aggregation rules, first the Aggregation Module has to receive a configuration file as its input. One of the core concepts that can be tested here is if, the configuration file can be tampered with. This would happen by creating a fake configuration file based on the grammar and giving it as input to the Aggregation Module.

The subsequent part would be to log what the Aggregation Module does with the configuration file. One very desired outcome would be that the Aggregation Module recognizes the faked configuration and does not produce any rules. Another outcome would be that the Aggregation Module does process the file, generates aggregation rules but that the rules themselves do not hinder the correct functionality of the *Aggregation Module.*

6.8 Summary

In this chapter techniques were presented in order to evaluate the requirements posed in Chapter 2.

These were the following:

- **R1**: Elimination of duplicates in the alert stream, and as a subsequent side effect reduce the volume of alerts.
- **R2**: Hinder loss of information by blocked alerts
- **R3**: Provide an interface to communicate with the Policy Enforcement Point
- **R4**: Flexibility in policy specification, through dynamic provided rules

R1 is implemented through the solution presented in Section 4.2.1. Stream Filtering is evaluated by recording a simulated attack and documenting each step of the simulation. By replaying the simulation with the Aggregation Module in place, it is possible to test if for the pre-given input the expected output is generated. If the expected output is received, without containing duplicate information and the amount of false negatives is significantly low then it can be stated that this component is functioning correctly.

R2 is implemented by the *Alert Fusion* component. Again by using a traffic dump of documented simulated attacks one can create the input needed, and predetermine the output that is going to be generated by the component. In this case the output is inspected in order to verify that the output which is generated also contains the correct information. As in the above case of *Stream Filtering*, a significantly low number of wrongly aggregated or not aggregated alerts, can let us conclude that this components functions as intended.

R3 is implemented through *Action Inclusion* as discussed in Section 4.4. In order to determine if the feedback does indeed bring change in the behaviour of the Policy Decision Point, it has to be inspect if the attack instances which were not resolved when the attack simulation is running without the *Aggregation Module*, have been resolved in the implementation containing the Aggregation Module.

R4 can be verified by using the same method as the one that was used in verifying the solution to **R2**. By inspecting the alerts which were generated by the dynamic aggregation rules it can be observed if the function does indeed function as intended.

Performance can be quantified over the components as a total by measuring the consumed resources, and the delay which arises when the Aggregation Module processes the alerts. Using the worst case scenario defined in Section 6.3 the system can be tested as to how it performs under full load. This can be used to verify the stability of the system and the resource consumption of the Aggregation Module.

Security is another essential evaluation target. Since the *Aggregation Module* operates as a sub-Aggregation Module of a much larger security application, it has to be tested in order to verify that the Aggregation Module itself is secure from tampering. Not all requirements can be exhaustively tested, but the core is to test the *Alert Fusion* component if its memory can be overrun, if the feedback can be faked, and if faulty configuration files can be recognized.

7. Conclusion and Future Work

The work presented here proposed a sub-module for the ANSII project, which is responsible for aggregating alerts, eliminating duplicates in the process and providing new insight, through feedback, about the state of threats in the system. Chapter 2, gave a brief overview about the ANSII Project, and then proceeded to analyse the environment which this thesis studied. It proceeded to identify the problems which exist in the traditional design of the network. The most important problems presented were the non-existent communication with the Policy Enforcement Point, and the huge number of duplicate alerts generated by the IDSs.

Chapter 4 presented a solution to the problems which were identified in the Analysis problem, which is called *Aggregation Module* and has as its main mission to aggregate similar alerts, forward alerts only when needed thus eliminating any duplicate alerts reaching the PDP, and receiving and appending feedback to the alerts which are contained in its memory. Through a special configuration file which contains the rules defined in the PDP, the module generates aggregation rules on the fly thus removing the need of defining special rules in every PDP. The implementation chapter presented the frameworks chosen to implement the module, and how each framework can be mapped to the functionalities defined in chapter 4.

In the evaluation section, evaluation methods were derived from the requirements posed in Chapter 2, which intent to produce tests that can verify whether the module functions as intended, does this without significant trade-off and that the module cannot be compromised through malicious actions. Although no implementation was made for the concept envisioned in this thesis and thus no evaluation, the concepts provided here should add a contribution to the alert aggregation field, and to the total of the correlation process.

This thesis, proposed to not only aggregate alerts based on common characteristics, but introduces the idea of deriving forwarding rules of alerts from the Policy Decision Points rules. Through the point of view, of intrusion response systems, this tries to simplify the rule specification in those systems. There is no need to specify special rules for the content that is going to be forwarded to the Policy Decision Point of an IRS.

Another re-approach that this thesis adds to the concept of IRSs, and alert aggregation in general is the consideration of feedback from the different Policy Enforcement Points in the network. Through the feedback new insight can be gained about resolved, and unresolved threats in the system. These insights were presented in Figure 2.3 from Section 2.4.3.

Through the inclusion of feedback as part of a larger alert, can allow more flexible and adaptable reactions from the point of view of the decision and enforcement mechanism.

Subject of further research should be the successful implementation of the module, and evaluation. A core subject that can be considered after this is to expand the modules functionality. The alert correlation process and the concepts introduced in it, can also be applied in a number of sensory equipment. An approach here is to accommodate as part of the correlation process the analysis of alerts generated by other security components and not only the Intrusion Detection Systems.

Bibliography

[AMZh07] Safaa O. Al-Mamory und Hong Li Zhang. A Survey on IDS Alerts Processing Techniques. In *Proceedings of the 6th WSEAS International Conference on Information Security and Privacy*, ISP'07, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS), S. 69–78.

[ANSI] ANSII Website. http://www.ansii-projekt.de/.

[Axel00] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technischer Bericht, Technical report, 2000.

[CaCM02] Nathan Carey, Andrew Clark und George M. Mohay. IDS Interoperability and Correlation Using IDMEF and Commodity Systems. In *Proceedings of the 4th International Conference on Information and Communications Security*, ICICS '02, London, UK, UK, 2002. Springer-Verlag, S. 252–264.

[Carv00] Curtis A Carver. Intrusion Response Systems: A Survey. *Department of Computer Science, Texas A&M University, College Station, TX*, 2000, S. 77843–3112.

[CBCAD09] Nora Cuppens-Boulahia, Frederic Cuppens, Fabien Autrel und Herve Debar. An Ontology Based Approach to React to Network Attacks. *Int. J. Inf. Comput. Secur.* 3(3/4), Januar 2009, S. 280–305.

[Cupp01] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *Proceedings of the 17th Annual Computer Security Applications Conference*, ACSAC '01, Washington, DC, USA, 2001. IEEE Computer Society, S. 22–.

[Daya] Bhavya Daya. Network security: History, importance, and future. *University of Florida Department of Electrical and Computer Engineering.*

[DeWe01] Hervé Debar und Andreas Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, RAID '00, London, UK, UK, 2001. Springer-Verlag, S. 85–103.

[ESPEa] *ESPER Articles and Presentation.* Available at : http://esper.codehaus.org/tutorials/tutorial/presentations.html.

[ESPEb] *ESPER Website.* http://esper.codehaus.org/.

[EtNi10] Opher Etzion und Peter Niblett. *Event Processing in Action.* Manning Publications Co., Greenwich, CT, USA. 1st. Auflage, 2010.

[FaJM09] Guo Fan, Ye Jihua und Deng Mingxing. Design and implementation of a distributed IDS alert aggregation model. In *Computer Science & Education, 2009. ICCSE'09. 4th International Conference on.* IEEE, 2009, S. 975–980.

[FGHW+08] Bingrui Foo, Matthew W Glause, Gaspar M Howard, Yu-Sung Wu, Saurabh Bagchi und Eugene H Spafford. Intrusion response systems: a survey. *Information Assurance: Dependability and Security in Networked Systems, Spafford EH (ed.). Morgan Kaufmann Publishers: Burlington, MA*, 2008, S. 377–412.

[G. W99] Nortel Networks A. Westerinen L. Rafalow R. Moore G. Waters, J. Wheeler. Policy Framework Architecture. Technischer Bericht, Microsoft, IBM, Network Working Group, 1999.

[GoHK03] Seymour Goodman, Pam Hassebroek und Hans Klein. Network Security: Protecting Our Critical Infrastructures. *ITU Background Paper. http://www. itu. int/osg/spu/visions/networksecurity/paper3. html*, 2003.

[HDFe07] D. Curry H. Debar und B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, SecureWorks, Inc., March 2007.

[HoSi11] Alexander Hofmann und Bernhard Sick. Online Intrusion Alert Aggregation with Generative Data Stream Modeling. *IEEE Trans. Dependable Secur. Comput.* 8(2), März 2011, S. 282–294.

[LeLN04] Seong Ho Lee, HyungHyo Lee und BongNam Noh. A Rule-Based Intrusion Alert Correlation System for Integrated Security Management. In Marian Bubak, G. Dick van Albada, Peter M. A. Sloot und Jack Dongarra (Hrsg.), *International Conference on Computational Science*, Band 3036 der *Lecture Notes in Computer Science*. Springer, 2004, S. 365–372.

[Luck12] David C. Luckham. *Event processing for business : organizing the real-time enterprise.* Hoboken, N.J. John Wiley & Sons. 2012.

[MaMZ09] Federico Maggi, Matteo Matteucci und Stefano Zanero. Reducing False Positives in Anomaly Detectors Through Fuzzy Alert Aggregation. *Inf. Fusion* 10(4), Oktober 2009, S. 300–311.

[McKa09] Lauren McKay. Forrester Gives a Welcoming Wave to Complex Event Processing. http://www.destinationcrm.com/Articles/CRM-News/Daily-News/Forrester-Gives-a-Welcoming-Wave-to-Complex-Event-Processing-55492.aspx, August 2009.

[Norm90] Donald A Norman. The'problem'with automation: inappropriate feedback and interaction, not'over-automation'. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 327(1241), 1990, S. 585–593.

[PoNe97] Phillip A. Porras und Peter G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *In Proceedings of the 20th National Information Systems Security Conference*, 1997, S. 353–365.

[RCHP00] Daniel Ragsdale, Curtis Carver, Jeffery Humphries und Udo Pooch. Adaptation Techniques for Intrusion Detection and Intrusion Response Systems. In *In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics at.* Society Press, 2000, S. 2344–2349.

[Reye04] María Angélica Reyes Muñoz. *Contributions to an advanced design of a Policy Management System.* Dissertation, Universitat Politècnica de Catalunya. Departament d'Enginyeria Telemàtica, 2003-07-04.

[RoCM09] Sebastian Roschke, Feng Cheng und Christoph Meinel. An extensible and virtualization-compatible IDS management architecture. In *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on*, Band 2. IEEE, 2009, S. 130–134.

[Schm08] Ivy Schmerken. Deciphering the Myths Around Complex Event Processing. http://www.wallstreetandtech.com/data-latency/deciphering-the-myths-around-complex-eve/207800335?pgno=2, Mai 2008.

[Schm11] Ivy Schmerken. Secrets Revealed: Trading Tools Uncover Hidden Opportunities. http://fixglobal.com/home/secrets-revealed-trading-tools-uncover-hidden-opportunities/, Juni 2011.

[TeIn] Esper Team und EsperTech Inc. *Esper Reference*. EsperTech Inc.

[VVKK04] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel und Richard A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Trans. Dependable Secur. Comput.* 1(3), Juli 2004, S. 146–169.

[ZaYZ08] Tianning Zang, Xiaochun Yun und Yongzheng Zhang. A Survey of Alert Fusion Techniques for Security Incident. In *Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management*, WAIM '08, Washington, DC, USA, 2008. IEEE Computer Society, S. 475–481.

[ZhGh05] Bin Zhu und Ali A. Ghorbani. Abstract Alert Correlation for Extracting Attack Strategies, 2005.