UNIVERSITY OF AGDER

Searchable Privacy-Enabled Information and Event Management Solution.

by

Kristoffer T. Seneger

Supervisors

Nils Ulltveit-Moe

Terje Gjøsæter

This master's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

The University of Agder
Faculty of Engineering and Science
Department of Information and Communication Technology

Grimstad, May 26, 2015

Abstract

With network traffic proliferating over the last couple of decades, there is an increasing need to monitor security information in order to prevent and resolve network security threats. A Security Information and Event Management (SIEM) solution collects all the alerts that the various Intrusion Detection and Prevention Systems (IDS/IDP or IDPS) generates, as well as security logs from various other systems, into one database so that the security analyst (SA) can more easily get an overview of the threat activity. A privacy enhanced anonymization and deanonymization protocol (Anonymiser/ Reversible Anonymiser) has been used to prevent a first-line security analyst, without proper clearance, getting access to personal identifiable information (PII) and/or other types of confidential information that are not allowed to leave the network perimeter. Some examples may be PII sampled in IP packets, critical address information and network architecture. This thesis proposes an architectural design for a new SIEM solution which utilises a reversible anonymizer (RA) for enabling privacy-enhanced data collection and on demand deanonymization of anonymized alarms.

***Keywords**:*

Reversible Anonymiser, IDMEF, NSM, SIEM, Informational Privacy, PHP.

Preface

This thesis has been submitted to the Faculty of Engineering and Science of the University of Agder, Norway in partial fulfillment of the requirements for the degree of Master of Information and Communication Technology (ICT) with specification on Security. Work on this thesis has been carried out between 5th January and 26th May 2015 while working as a Security Analyst at NTT Com Security (formerly Secode) Norway AS.

It is quite overwhelming now, to think back at the start of the project, without any prior knowledge of PHP and system design other than what packet data and various network forensics tools could show, and to see how much I have actually learned during the last months. I chose this project in order to refresh my knowledge of network security after years away fighting a winning (obviously) battle against cancer, and to learn as much as I could in the process. If anything, that goal is a success. There were many things planned to be done during the project, that unfortunately were out of my understanding or knowledge to complete. But I think this report shows that at least some fundamentals has been touched upon, and that with continued work could be something great. Knowing now that I will continue to work and maintain SIEM systems[1], this project have been immensely helpful, even though very hard to complete considering my starting point.

I would like to give my thanks to my supervisors, Nils Ulltveit-Moe and Terje Gjøviksæter, for being understanding of my situation and in constant helping me and guiding me throughout the project.
And last, but not least, I give my all my love to my wife, Sara, for without her continuous love, support and understanding I would have never had the strenght to get where I am now.

[1] http://secode.com/2013/siem-as-a-service/

Now. Witness the power of this fully armed and operational battle sta.... I mean Thesis.

Contents

List of Figures

List of Code Snippets

Glossary

AntiVirus Sofware that recognizes files and programs that may be harmful to the system and prevents them from executing, and in some cases deletes the file completely. 12

Apache The Apache HTTP Server, is the world's most widely used web server software. 29

Asset An asset is anything within your organization that has value. This may include computers, servers, and networking equipment as well as data, people, processes, intellectual property and reputation. ix

Computer Emergency Response Team The name for expert groups that handle computer security incidents. x

Logfile In computing, a logfile is a file that records either events that occur in an operating system, software runs, or messages between different users of a communication software. 11, 12, 16

PHP Hypertext Preprocessor; a server-side scripting language designed for web development but also used as a general-purpose programming language. 29, 32, 36, 40, 41

PostgreSQL An object-relational database management system (ORDBMS) with an emphasis on extensibility and standards-compliance. 20

Deanonymiser The Reversible Anonymiser[17], occasionally described in the thesis as the Deanonymiser for easier readability. xi, 22

Security Analyst In IT; a Security Analyst is a professional that analyses computer logs, network alerts and information to ensure that information stored on computers on the network are not disclosed to unwanted parties or modified inadvertently. Commonly referred to in the thesis as SA. xi, 6, 11, 24, 34

Social Engineering The act of manipulating people, either indirectly, using faux web pages or emails, or directly, using social media, messenger software, or in person. 2

Threat A threat is a party with the capabilities and intentions to exploit a vulnerability in an Asset. 1, 10, 22

Acronyms

AIDS Application-Based Intrusion Detection System. 10, 11

CERT Computer Emergency Responce Team. 4, 22, 24, 25, 27

DOM Document Object Model. 32

FP False Positive. 32

HIDS Host-Based Intrustion Detection System. 10

HTML HyperText Markup Language. 22, 35

HTTP Hypertext Transfer Protocol. 17, 18

HTTPS ”HTTP over TLS”, ”HTTP over SSL”, or ”HTTP Secure”. 18

ID Intrustion Detection. 9, 10, 12, 13

IDMEF Intrusion Detection Message Exchange Format. 4, 5, 9, 14, 16, 17, 19, 21, 22, 24, 30, 32–37, 41

IDP Intrusion Detection and Prevention. 1, 2, 9, 16, 21, 32

IDPS Intrustion Detection and Prevention System. 1

IDS Intrusion Detection System. 9–12, 16, 21, 22, 32

IP Internet Protocol. 22

IT Information Technology. 7

LMS Log Management System. 12

NIDS Network-Based Intrustion Detection System. 10, 11, 16

NSM Network Security Monitoring. 3, 7, 9, 10, 19

OS Operating System. 10, 11, 29

PbD Privacy by Design. 16

PIDS Physical Intrusion Detection System. 11

RA Reversible Anonymiser. viii, 1, 2, 4, 8, 9, 14–16, 19, 21, 25, 36, 41

SA Security Analyst. viii, 1–4, 10, 12, 15, 19, 22, 24, 25, 31, 41

SAX Simple API for XML. 32, 33

SEC Security Event Correlation. 12, 32

SEM Security Event Management. 12

SIEM Security Information and Event Management. 1–5, 8–10, 12, 13, 16, 19, 20, 22, 24–27, 30–32, 34, 39, 41

SIM Security Information Management. 12

SMTP Simple Mail Transport Protocol. 18

SOAP Simple Object Access protocol. 17, 19, 21, 22, 30, 36, 37, 41

SQL Structured Query Language. 20–22, 35

UI User Interface. 21, 26, 27, 29, 34

USB Universal Serial Bus. 2

WSDL Web Services Description Language. 17, 36

XACML eXtensible Access Control Markup Language. 9, 14, 16, 19, 21, 25, 41

XML Extensible Markup Language. 9, 17, 21, 22, 30, 32, 33, 36, 37

Chapter 1

Introduction

This master thesis in IT security presents a design suggestion for a searchable, privacy enhanced Security Information and Event Management (SIEM) solution capable of integrating with the Reversible Anonymiser (RA)[1]. This chapter will give a presentation of this thesis.

With network traffic that have been proliferating over the last couple of decades, the amount of network threats has been growing with it. To combat this, network engineers have developed systems that autonomously monitor and collect information about the various systems and networks to identify and/or prevent threats to their assets. These Intrusion Detection and Prevention Systems (IDPS or just IDP) needs to be constantly configured to accommodate changes on the network and threats. For a Security Analyst (SA) it is also of interest to be able to access these logs in order to verify possible malicious activities on the various systems and to look for false positives. A Security Information and Event Management (SIEM) solution collects all the alerts that the various IDPSs generates as well as security focused information from various system logs into one database so that the Security Analyst (SA) can more easily get an overview of the activity.

[1]Presented in the PhD dissertation: N. Ulltveit-Moe, *Privacy-enhanced network monitoring*[17]

In order to prevent the SA (and other SIEM operators), without proper clearance, getting access to personal identifiable information (PII) or other types of confidential information that may be available in these logs and Intrusion Detection and Prevention (IDP) alerts, a policy filter that anonymizes the Intrusion Detection Message Exchange Format (IDMEF), the standard for sending IDP/IDS messages, has been developed for use in Critical Infrastructures where privacy of information is required.

In this thesis I will have a look at what defines a SIEM‘s basic functions and how I would construct such a solution in order to implement the Reversible Anonymiser. From here I will also take a look at the various information that, while partially anonymized, would still be effective for an analyst without the proper security clearance.

1.1 Background and Motivation

For a SA, it is very important to have a quick and complete overview of what is happening in the systems at any given time. With every system being connected to a network, be it to the Internet or the local network, there will always be of interest to know if there is anything of malicious intent going on. A perfect example of this is the Stuxnet worm incident in the Iranian nuclear power plant, where it is believed that network was compromised due to social engineering and a Universal Serial Bus (USB) stick [14].

While a security analyst will usually need to get the proper security clearance in order to even get employed by certain companies, there will still be cases where the security operator or analyst without such clearance is in need of accessing systems that can contain information of personal identifiable nature.

1.2 Problem Statement/Thesis description

This thesis aims to design and build a search-able, privacy-enhanced Security Information and Event Management (SIEM) solution that collects and anonymizes information defined as private or confidential in security events according to a privacy policy. The SIEM will support on-demand deanonymization of information in a Security Operations Center (SOC). The solution must be secure, also in an outsourced scenario, and should ensure transparency and accountability of access to private or confidential information in a secure and non-repudiable way.

1.3 Importance of Topic

With the network traffic flow that continues to grow on a near exponential level, the need for an effective way to monitor the information flow increases. The now increasingly more popular SIEM solutions are a great way to solve this need. But with the increased traffic, the possibility that a SA without the proper security clearance, unwillingly or not, will be subjected to sensitive information while assessing the network also increases. The implementation of a anonymized alarm database, as well as a SIEM solution that enables the possibility to on-demand deanonymize these alarms could be essential to combat such a problem. In the last years, services like SIEMaaS[2] has arrived. These are services that focus entirely on companies outsourcing their Network Security Monitoring (NSM) needs, further increasing the importance of enhancing the SIEM's design for handling company privacy policies.

[2] `http://secode.com/2013/siem-as-a-service/`

1.4 Key Assumptions

Building a functional solution that manages to integrate all the different logs in the network and systems usually requires a lot of work from multiple people. This workload is reflected with that only the most successful companies have the capital and manpower to build and maintain their own SIEM solutions. Considering this, the solution presented will be limited to focus on the anonymized Intrusion Detection Message Exchange Format (IDMEF) messages that the RA generates.

The RA is based on a hybrid encryption scheme that enables multiple stages of authorization mapping, making it possible for the SIEM to deanonymize different layers of information according to the various SAs secret key (effectively giving them different clearance level). The solution presented in this thesis will assume that there is only one level of anonymization.

I will also limit the use case environment for the project to two main case studies:

- The first case is in a critical infrastructure network, where the SA operates with full security clearance and the possibility to deanonymize private information at will. This scenario might be from a traffic control structure or other high value networks where a low response time is critical for handling events.
- The second case are in a scenario where network security monitoring is outsourced to a private firm focusing on security information log analysis, and the SA requires confirmation from a trusted Computer Emergency Responce Team (CERT) in order to deanonymize alerts. Example for this types of cases could be a hospital, where the patient register information and other types of personal identifiable information is of importance for privacy.

1.5 Thesis Organization

Chapter 2 will explore the state-of-the-art and prior research related to the subject of this thesis regarding privacy, network security monitoring and their systems and appliances, as well a brief overview of the Reversible Anonymiser. Chapter 3 will present the design for the searchable privacy-enabled SIEM, and Chapter 4 will focus on building a prototype for the alert message transfer and XML parsing of IDMEF messages transferred from the Reversible Anonymiser. Chapter 5 will discuss some of the choices that have been made during the design process, as well as some of the main problems that were faced both during the design and prototyping phases of thesis. It will also discuss what should be focused upon for future work on the topic. Chapter 6 will summarize the obtained results and contributions that this thesis has presented.

Chapter 2

Prior Research

In the last couple of decades, the amount of data that get transmitted around the Internet has grown in order of magnitude, and most likely will continue to do so in years to come. In the same time, different protocols to transmit this data between computers and how to present it to the end user needed to be continuously developed and defined to accommodate the increase in threats to the security. Usually the systems and protocols have a way to debug and keep track of what is going on. Usually this information is stored on a file somewhere on the system. This ”logfile” is kept on the system for the *security analyst* to ”plow” through to find what is wrong.

2.1 Informational Privacy

There are several ways to define privacy both from a legal perspective, company perspective and personal perspective. The most commonly defined way of defining privacy was defined by Warren and Brandeis as *”The right to be let alone”* in 1890 [20]. They discribe privacy as a person’s right to be free of constraint, coercion, and even uninvited observation, or as we today say so commonly as

"Respecting a persons space". In Information Technology (IT), privacy is called *informational privacy*, and James H. Moor has defined privacy as *"The right to the control of access to personal information."*[12] His definition contains four important elements [12]:

1. **Information** - The informational knowledge about someone.
2. **Personal Information** - The personal information that can give access to a subject's person; his/hers identity, thoughts, aspirations, habits, etc.
3. **Control** - A person has the right to choose how much or how little of the information above is known, and to whom the information is revealed.
4. **Right** - A person's control over personal information ought to be respected and protected.

People's tendencies to reveal their personal information to friends on Facebook and other social media as well as surveillance programs sanctioned by foreign governments [10] has made it difficult to ensure that these elements are being preserved completely. There are some theorists that provide a more technical definition of privacy and often define privacy as *equivocation*; that there is some level of privacy and anonymity in a sufficiently large crowd of other information.

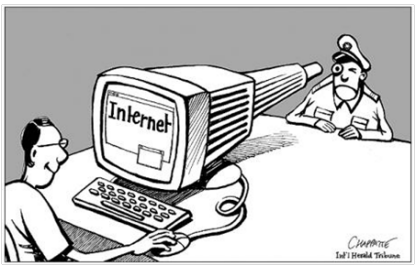


Figure 2.1: Internet privacy

In the physical world, people have become more willing to trade in some privacy and convenience for added security. And in the same sense this applies to IT and NSM as well, as an information leak in the corporate world may result in a great financial loss [11].

2.1.1 Privacy by Design

Ann Cavoukian began in the late 1990s to discuss the virtues of building privacy into technology from the start, and called it *"Privacy by Design"* [5]. It started as a technology concept, but developed into a conceptual model for building an entire privacy program with seven core principles. The principles that should be followed are as follows [5]:

1. Proactive not reactive - the system should aim at preventing privacy issues before they turn into problems.
2. Privacy as the default - ensure that personal data automatically is protected.
3. Privacy embedded into the design - not bolted on as an add-on.
4. Positive sum - integrate privacy enhancing technologies that support both the privacy and security objectives creating a win-win situation.
5. Life cycle protection - ensure that sensitive data is protected from the moment the data is created and until it can be securely destroyed.
6. Visibility/transparency - ensure that operations on privacy sensitive information are traceable.
7. Respect for users - keep the interests of the individual uppermost - offer appropriate notice and empowering user-friendly options.

The RA already upholds these principles [17], and it should also be made sure that the design of the privacy-enabled SIEM, presented in this thesis, are respecting these principles.

2.1.2 Privacy-Enhanced IDS

While there has been some research done on preserving *information privacy* in Intrusion Detection System (IDS) and IDP systems[9], and several attempts at implementing reversible anonymization of packet analysis tools[21, 13], there has been very few attempts on designing or integrating this type of systems into a SIEM system. There may be some corporations or agencies that are privately developing systems, but it is very hard to locate any publicly available information on the subject. The RA are one of the few, if not only, successful attempts to support reversible anonymization of Extensible Markup Language (XML) and IDMEF messages based on eXtensible Access Control Markup Language (XACML) policies.

2.2 Network Security Monitoring

NSM is a term that has evolved after the continued advancement in different fields of network security. Prior to coining the term NSM, this domain of network threat detection was typically described as Intrustion Detection (ID); though the terms are still being used interchangeably, they are not synonymous: Intrusion detection is more of a component of modern NSM[15].

There are four main characteristics of NSM [15]: The first one being that *"Prevention Eventually Fails"*. In contrast to ID centric defenses, where the focus is almost solely on detecting and preventing exploitation of vulnerabilities, NSM focuses on the fact that no matter how strong your defences are, or what proactive steps have been

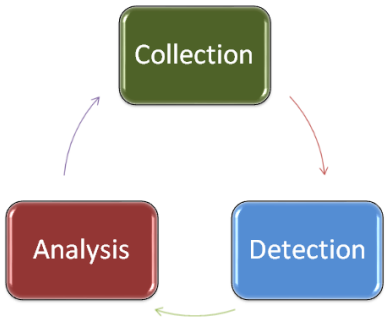


Figure 2.2: The phases of a NSM cycle

taken, a motivated attacker will eventually find a way to get in. This enables the responder (or SA) to focus on detection and response, and thus may respond more efficient and be better suited when a compromise eventually happens. Second is *"Focus on Collection"*. While some IDSs do collect data, the collection gathered is often unfocused and not tied to detection goals. Focusing on collecting the right data can help the SAs in making sound decisions much faster. Third, a *"Cyclical Process"*; this means that the collection of data should feed the detection of threats, detection of threats should feed the analysis, and analysis should feed back into the collection. Lastly is a *"Threat-Centric Defense"*; while ID focuses on a *vulnerability centric defense* that centers on *how* a threat functions, NSM focuses on *threat-centric defence* that centers instead on *who* the attacker is, and *why* the attacher is attacking. This enables the possibility of preventing further attacks.

2.2.1 Intrusion Detection Systems

As stated previously, ID is still a component of NSM, and IDSs are a critical part of any SIEM solution. While there are several different types of IDSs, there are three main categories of IDS [16]: Host-Based Intrustion Detection System (HIDS), Network-Based Intrustion Detection System (NIDS) and Application-Based Intrusion Detection System (AIDS).

HIDSs focus on a specific device, and generally involve an agent getting installed on each system, monitoring and alerting on local Operating System (OS) and application activity. They can analyze activities on the host they monitor at a high level of detail, and can often determine which processes and/or users are involved in malicious activities.

NIDSs attempt to identify unauthorized, illicit, and anomalous behavior based solely on network traffic. NIDSs are passive systems that, when connected to a network, gather information by monitoring the traffic. These systems can moni-

tor entire networks with only a few well placed devices. SNORT[1] is an example of a NIDS. These systems are mostly passive devices that monitor ongoing network activity without adding any significant overhead, or interfering with network operations.

AIDSs are IDSs that concentrate on events that occur within specific applications. They detect attacks through analysis of application logfiles and can sometime even track unauthorized activity from individual users.

There are also a forth minor category; Physical Intrusion Detection Systems (PIDSs)[2], that is the act of identifying threats to physical systems such as security cameras, access control systems and motion sensors.

2.2.2 Security Information and Event Management Systems

Every application, process and OS are usually programmed to generate information of what is happening at any given time. This information is then logged and placed in a logfile. These files are handy for IT personnel when something is wrong with the system, or if one or more programs conflict with each other. For a security analyst, having access to these files can be critical to figuring out the location of a virus outbreak, or for example to see if something/someone is accessing or modifying system configurations or data that, when correlated with what is shown in the network traffic, can indicate a security breach.

When monitoring a large network of systems, the high number of logs available will make it very hard to effectively locate the information required. While many of these logs contain very useful information regarding various security breaches as well as unusual program behaviors, they will be hidden in the pure amount of data.

[1] `https://www.snort.org`
[2] `http://www.sans.org/security-resources/idfaq/what_is_id.php`

In order to help sorting out the relevant information contained in these log files various system engineers came up with several different solutions that got accepted and improved upon. Some of these technologies are[7]:

- **Log Management System (LMS)**. - LMS is a system that collects and stores logfiles from multiple hosts and systems into a single location in order to access them from a centralized location, instead of each systems individually.
- **Security Event Management (SEM)**. - SEM is the same as a LMS, but marketed towards system SAs instead of system administrators; highlighting system security entries.
- **Security Information Management (SIM)** - SIM hosts may have vulnerability information included in their logfiles, as well as ID and antivirus messages that are mapped to the systems. A SIM system collects this data into a central repository for trend analysis and can provide automated reporting.
- **Security Event Correlation (SEC)** - Five consequential failed login attempts may not be more than a five lines in a logfile or five messages from an IDS, but may be wort looking into for a SA. A SEC looks for patterns in the logfiles in order to raise alert when such things happen.

All of these technologies emerged in order for the system administrators and SAs to more easily locate and analyze the contents of all the different logfiles on the network. The system that the industry has now settled on the concept of ”Security Information and Event Management (SIEM)”,

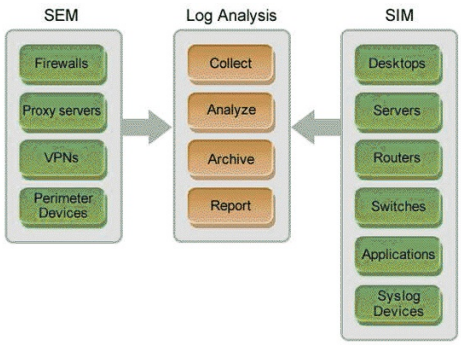


Figure 2.3: SIM + SEM = SIEM[3]

which essentially is a combination of
all of the systems mentioned above.

SIEM is in short a management layer above the existing systems and security controls that connects and unifies all the information from various sources. Making it possible to correlate and cross-reference information from a single interface.

Because of SIEM being a relatively new type of solution in the industry, there are no industry standards for how it should look or function, other than that its function as a faster and more coherent way to display all security logs, ID messages and alerts collected on the network. Several companies have developed different solutions, but what they all have in common is that they are all usually expensive to deploy and maintain, considering the vast amount of logfiles and systems a SIEM needs to support in order to operate on a functional level.

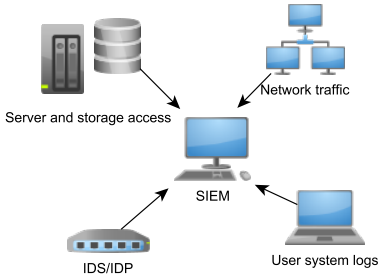


Figure 2.4: SIEM collects a selection of logfiles in the system.

Every year, Gartner Group publishes an report of the status of most of the established and up and coming SIEM contributers according to their ability to execute and their completeness of vision[4]. As seen in Figure 2.5, the top fulfilling contributers as of 2014 were IBM Security, HP (ArcSight), McAfee, Splunk and LogRythm.

Figure 2.5: The Gartner SIEM Magic Quadrant as of June 2014

2.3 The Reversible Anonymiser

The Reversible Anonymiser was developed to enable efficient monitoring of computer networks for signs of malicious activities, and at the same time maintain socially and legally acceptable solutions for handling data privacy and confidentiality [17]. The RA is designed to function as a policy handling middle-ware for use with services utilizing IDMEF messages, the established standard format for transferring security related messages[8]. It uses fine-grained enforcement of XACML-based privacy policies to anonymize different information that are transmitted in the IDMEF messages.

The RA cryptogaphically protects confidential information, so that only authorised personnel can access it, on a needs basis. It also supports different layers of

protection so that different SAs can have access to different information based on their security levels (for example Secret, Confidential and Restricted), but it does not support any relations or semantics between the layers. Considering that it could be desirable for a company with a critical infrastructure to enforce a shared secret scheme, to ensure that two or more parties must agree before revealing information, the possibility to provide key shares has also been implemented in the RA. Figure 2.6 shows an overview of this scheme.

The role and functions of the RA, will be shown throughout the thesis.

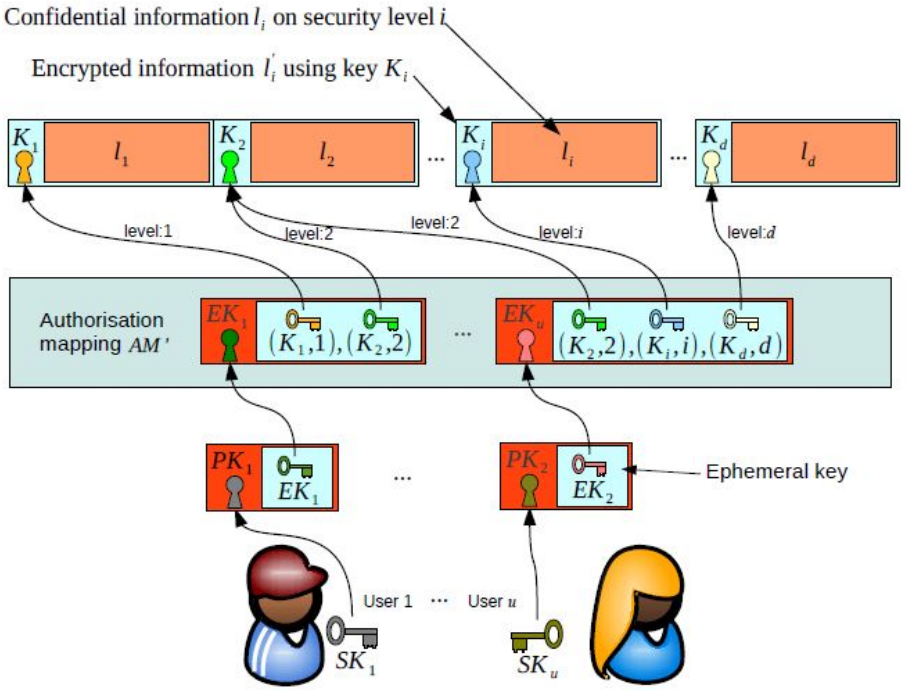


Figure 2.6: Overview of the the encryption sceme used to implement reversible anonymisation.

Chapter 3

Design

This chapter will present a design solution for a Security Information and Event Management system that utilizes the Anonymizer to uphold the Privacy by Design (PbD)[5] principles.

The definition of a SIEM solution is its capability to collect several different security focused logfiles and IDS/IDP alerts into one central database. As defined in chapter 1.2, the focus of this thesis is to design such a system that utilizes the Reversible Anonymiser (RA) for enabling a searchable privacy-enabled data collection and on demand deanonymization of anonymized alarms. Since the RA is designed to anonymize Intrusion Detection Message Exchange Format (IDMEF) messages [17], using an anonymization policy generated with the XACML anonymization scheme policy editor [18], that are recieved from a NIDS and preconfigured for testing using SNORT[1] IDS. The anonymized IDMEF message produced by the RA is the message format that the design solution will focus on.

[1] `https://www.snort.org/`

3.1 Integral formats and systems

Before describing the solution there are some message exchange formats used in the design that need some explanation.

Intrusion Detection Message Exchange Format

The IDMEF message format was presented in RFC 4765[8] as a standard solution for sharing information of interest to intrusion detection and response systems. The format was designed to accommodate the need for a commonly used message exchange format for use in open source and research systems, allowing users to mix between systems to obtain an optimal implementation. IDMEF is implemented with XML, a syntax for specifying text markup[2], and therefore has the same annotation as an XML file.

The alert class structure standard for the IDMEF can be seen in figure 3.1.

Simple Object Access Protocol

Simple Object Access protocol (SOAP) is a messaging protocol that allows programs that run on disparate operating systems, such as Windows and Linux, to communicate using Hypertext Transfer Protocol (HTTP) and XML [1]. It requres a client to send the task/message, and a server to receive the message in order to perform the requested task. It utilizes the Web Services Description Language (WSDL) format to describe its network communication service, and both the SOAP server and client would need to have a *.wsdl* file, describing the WSDL settings and format in order to send and receive messages.

Because of its verbose nature (a trait inherited by XML), it may be slower than other types of middleware standards, but makes up for it with its ability to leverage

[2]Basically a language that describe other languages.

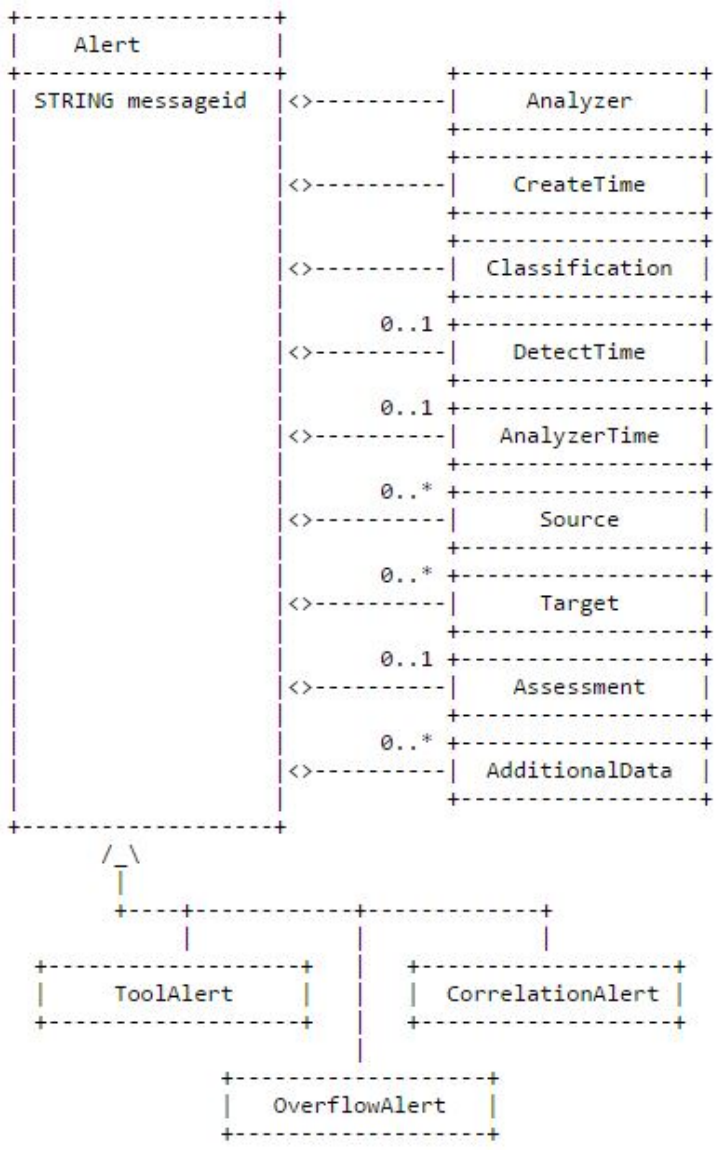


Figure 3.1: The IDMEF Alert class

different transport protocols, including HTTP, HTTPS, Simple Mail Transport Protocol (SMTP), among others.

3.2 The Privacy-Enhanced SIEM

The proposed design will focus on one specific type of information, namely the anonymized IDMEF messages that are produced by the Anonymiser. The SIEM design assumes that the anonymiser is attached to a SNORT IDS server that monitors traffic through a firewall or router, as shown in figure 3.2. The case described in chapter 1.4 (figure **??**) defines that a default *DENY* policy is used for all traffic that is captured by the SNORT IDS. This policy will effectively implement a privacy by default [5] scenario since only information explicitly allowed by the system manager are shown to the first-line SA.

As figure 3.3 shows, the RA listens for a SOAP message on a specified SSL port for IDMEF messages from the Prelude IDS system (in this case port 9990, as seen in Code 4.2). This message is processed by the XACML policy configured for the Anonymiser, which then send the anonymized IDMEF message forward with SOAP. From here it is up to the SIEM controller to ”collect” that message,

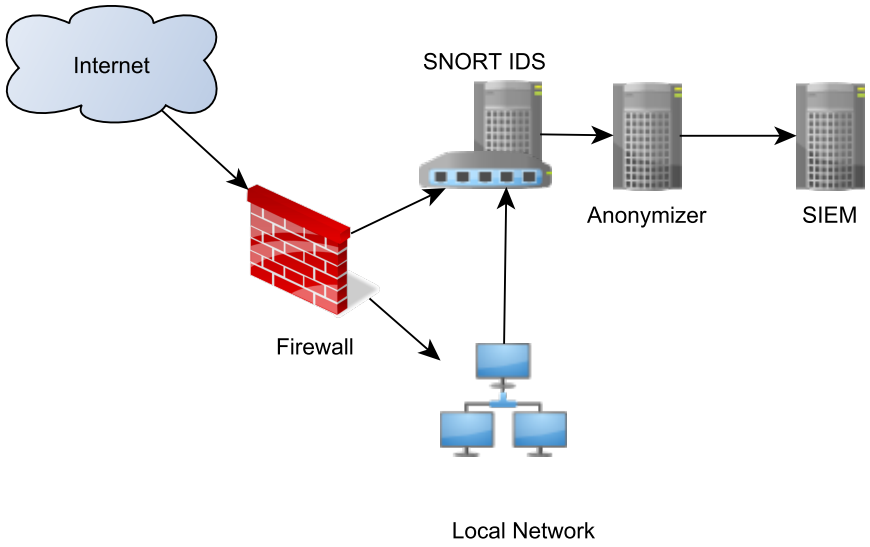


Figure 3.2: The supposed NSM setup

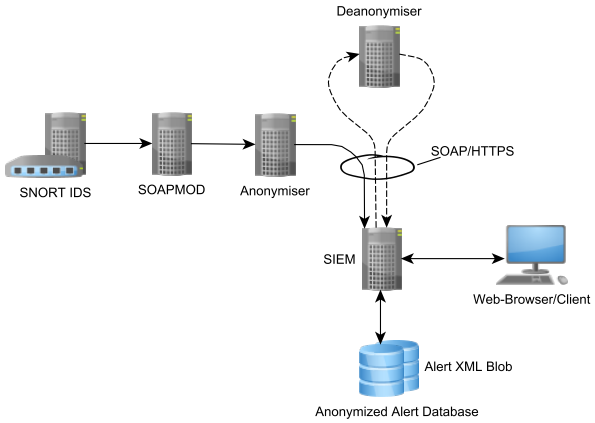


Figure 3.3: Overview of the Anonymiser/Deanonymiser SIEM design

and store it in the database.

3.3 Database Management

The proposed design will also take into account that the user will be able to search for specific alarms stored in the database. By putting the parsed alert information into a SQL database, the information by the design of Structured Query Language (SQL) already search-able. While making pure SQL queries would be the easiest to implement to, these queries may become very verbal and complicated. Though some of the most established SIEM solutions have built their own database software in order to compensate for the ever increasing range of new hardware being plugged in[3], the proposed SIEM solution suggests to use a SQL database (more specifically PostgreSQL[4]) pared with software that translates user input to SQL-queries (Figure 3.4). This could be either from text input, or selecting preset fields presented in the clients User Interface (UI).

[3] `http://docs.splunk.com/Documentation/Splunk/6.2.3/SearchReference/SQLtoSplunk`

[4] `http://www.postgresql.org/`

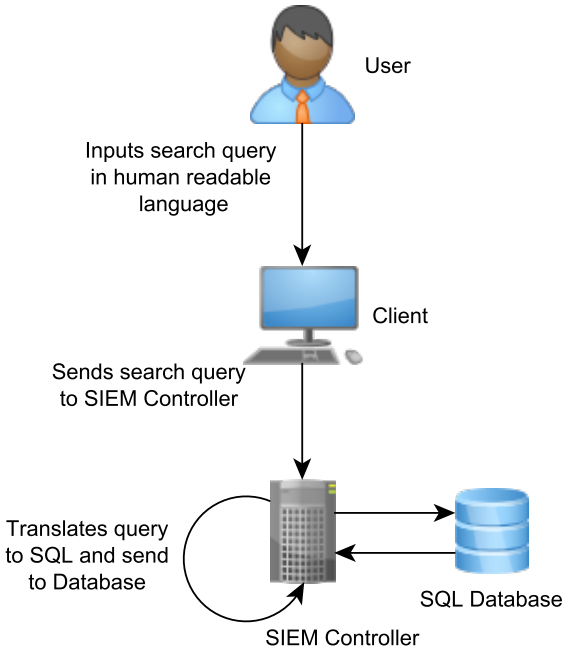


Figure 3.4: User query translation.

3.4 Message Collecting and Deanonymizing

The RA system is divided in two parts: The Anonymiser and the Deanonymiser. When configured, the IDMEF messages are sent from the IDP/IDS to the Anonymiser who runs through the messages and encrypts the XML entities names and attributes values according to what is dictated by the XACML policy. The Deanonymiser can deanonymize these encrypted messages, but only if they are in their original state from when they were sent from the Anonymizer. These entire encrypted messages are put in a *additionalData* XML element in a new IDMEF message that get wrapped around them. These new messages differ from the original by the fact that the anonymized information is replaced with ”dummy” data. Code 3.1 shows an example of the *Source* Element with its *ident* and *address* values anonymized.

As the anonymised IDMEF message is received from the Anonymiser, the message needs to be preserved in its unmodified state in order for the Deanonymiser to successfully deanonymize it [17, page 103]. This message ”blob” gets inserted into its own database table with its extracted *messageid* STRING (as shown in figure 3.1) as a relation.

Code 3.1: Anonymized Source XML Element

```
1 <Source ident="0000" spoofed="yes"><Node ident="000000"
    category="unknown"><Address ident="000000" category="
    ipv4-addr"><address>00000000000</address></Address></
    Node></Source>
```

The alert collection workflow (Figure 3.5) starts with the SOAP Server constantly listening for new messages from the Anonymiser. When it receives a new message, it sends it to the XML Parser who extracts its *messageid* field, and upload the (unparsed) message received to the alert ”blob” database. If the response from the Database comes with an error that the *messageid*, which is a primary key, already exists, it discards the message, and go back to listen for new messages. If the message is unique, the anonymised message is parsed by the XML Parser, and the extracted information is sent to the Anonymized Alerts database. The system will then go back to listen for new messages.

3.4.1 On-demand Deanonymization

Figure 3.6 simulates a scenario where a SA has recieved an alert from the IDS that can indicate a potential threat in the network. The SA are thus in need of deanonymizing the alert information that shows the Internet Protocol (IP)-address of the source. The scenario assumes that the SA has high enough security clearance, and are allowed to deanonymize the information without approval from a trusted CERT.

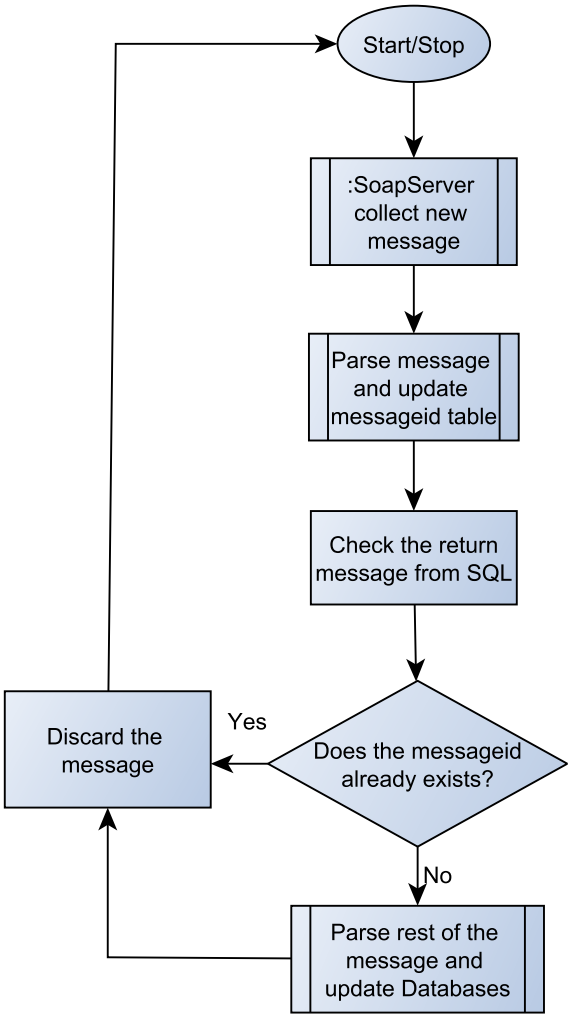


Figure 3.5: The alert collection workflow.

The SA first requests for the alert to be anonymised (1). The SIEM controller then takes the *messageid* and finds the corresponding ”IDMEF blob” from the Alert Database (2 and 3). This message is then sent to the Deanonymiser with SOAP over SSL/HTTPS using a trusted certificate (4). The Deanonymiser then checks the outer XML Signature of the encrypted message to verify that it has not been tampered with, decrypts the file[19], and sends it back to the SIEM (5).

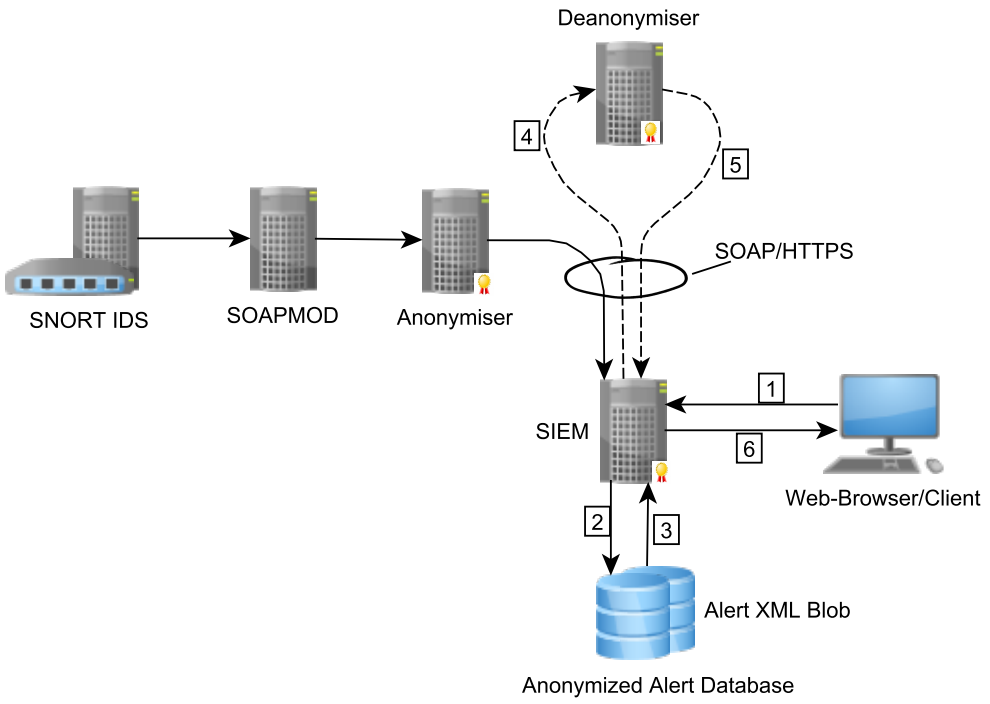


Figure 3.6: Simple flow of Anonymiser/Deanonymiser SIEM design

The SIEM parses the now unanonymized message, and shows it in its appropriate view on the Client (6).

The unanonymized message should not be stored in the SQL database, as this would make the data available for further searches and would not comply with the *End-to-End Life-cycle Protection* Privacy by Design principle[5]. Instead it should only be shown as generated HyperText Markup Language (HTML); text that would dissipate as soon as the text-field is closed.

3.5 User login and Clearance Control

If the user from Figure 3.6 were a first-line SA without proper security clearance, SA would need to get a permission from a trusted CERT, in order to deanonymize the IDMEF message. Figure 3.7 shows what would happen if the user is presented with an alarm that find suspicious (1), and would want to invastigate further. In this scenario, the system is using key share for deanonymization of anonymized alarms and the SA relies on getting permission from a trusted CERT in order to deanonymize the alarm. The SA then request to have the message deanonymized, but because of his/hers limited clearance level, this request needs to be approved by the trusted CERT before anything more is presented. A request for deanonymization of the alarm is sent to the SIEM together with the SAs secret key (2). A CERT personnel is then notified (3) that the SA wants to deanonymize the alarm, and is able to use his secret key together with the secret key of the SA to get the requested message deanonymmized (4). The trusted CERT is then able to see the unanonymized message and can approve the message for to the SA (5).

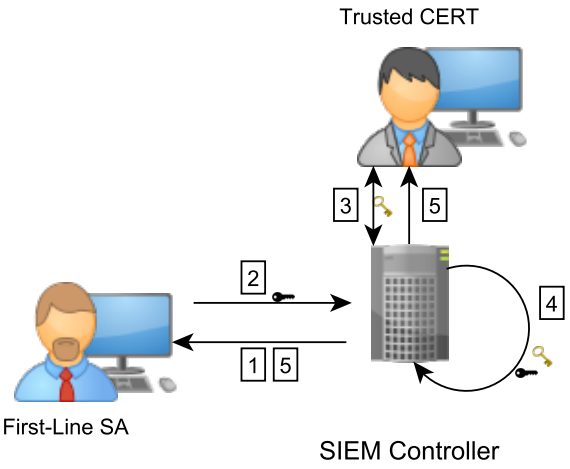


Figure 3.7: A simplified example of how the key exchange works with the RA.

3.5.1 User Interface

One of the functions of the RA and the XACML policy control is to be able to provide different levels of access according to the individual SAs security clearance level. The SIEM should reflect this, and should contain user login systems and a user database, where the users private key and access level would be handled and stored.

Figure 3.8 on page 27 shows the work flow when a user attempt to log in. The system will check the users credentials[5] and if correct, the user is logged in. When the credentials are checked, the user database will also contain the users private key and clearance level that when together with the password would enable the user to deanonymize alerts if high enough. This information could also be used to enable user specific functions in the UI.

After the user has logged in, the UI should present the user with an overview of the SIEM[6]. Complete with statistics of network traffic and alerts. According to his/hers security clearance level the user might also be able to deanonymize anonymized alarms. The work flow for the how the UI should show these alarms is shown on page 29 (figure 3.10), the sequence of events between the SIEMs main components can be found on page 28 (figure 3.9). This sequence diagram simulates a user with top level clearance (a trusted CERT), and are not in need of asking anyone for permission in order to deanonymize the alarm. The users secret key (sequence 11 through 17 in figure 3.9) should have the necessary clearance for this.

[5] Encrypting passwords when stored in databases are essential, and should obviously be implemented in any systems that require user login.

[6] An example of a SIEM UI Dashboard, as well an early concept of UI of the alert flow, can be found on page 42, figure 5.1 and **??**

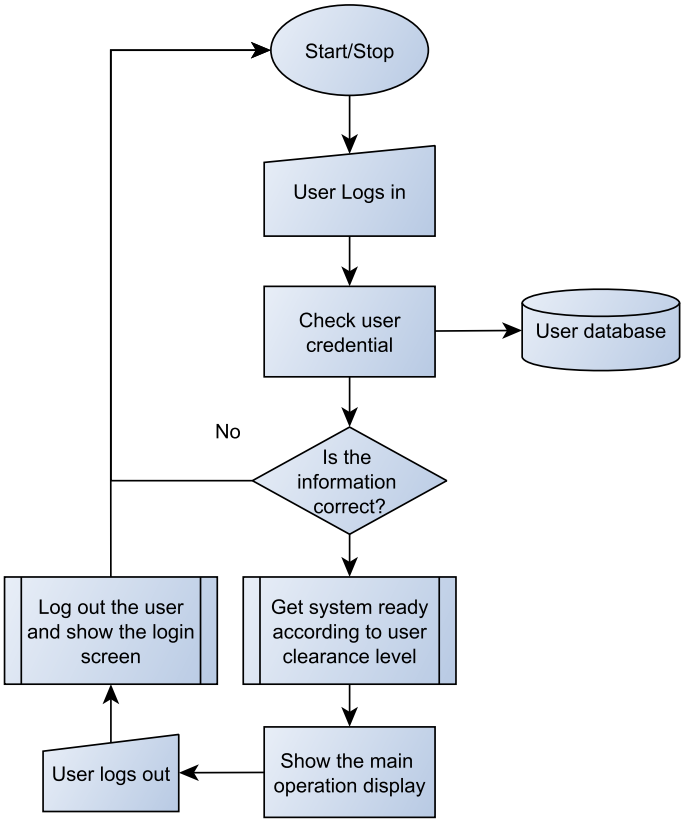


Figure 3.8: SIEM UI User login process

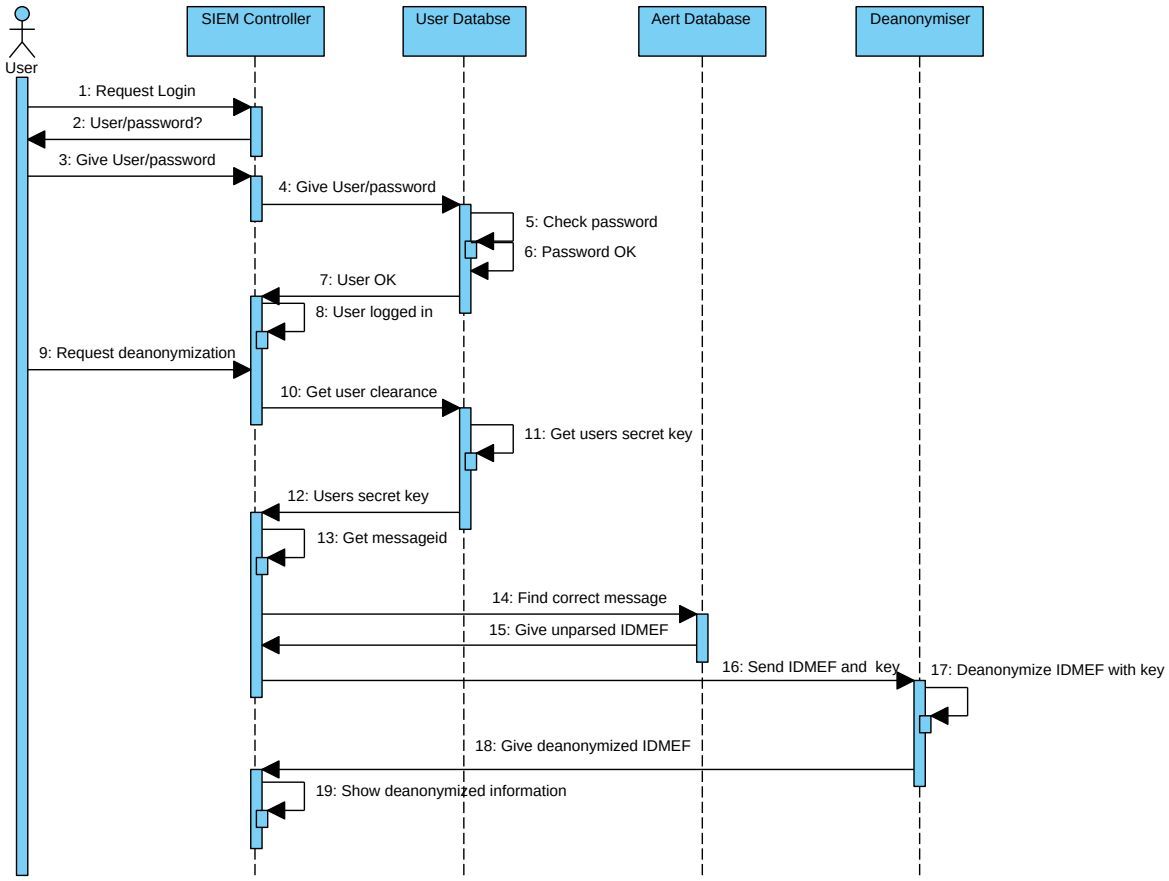


Figure 3.9: The user login and deanonymizing sequence

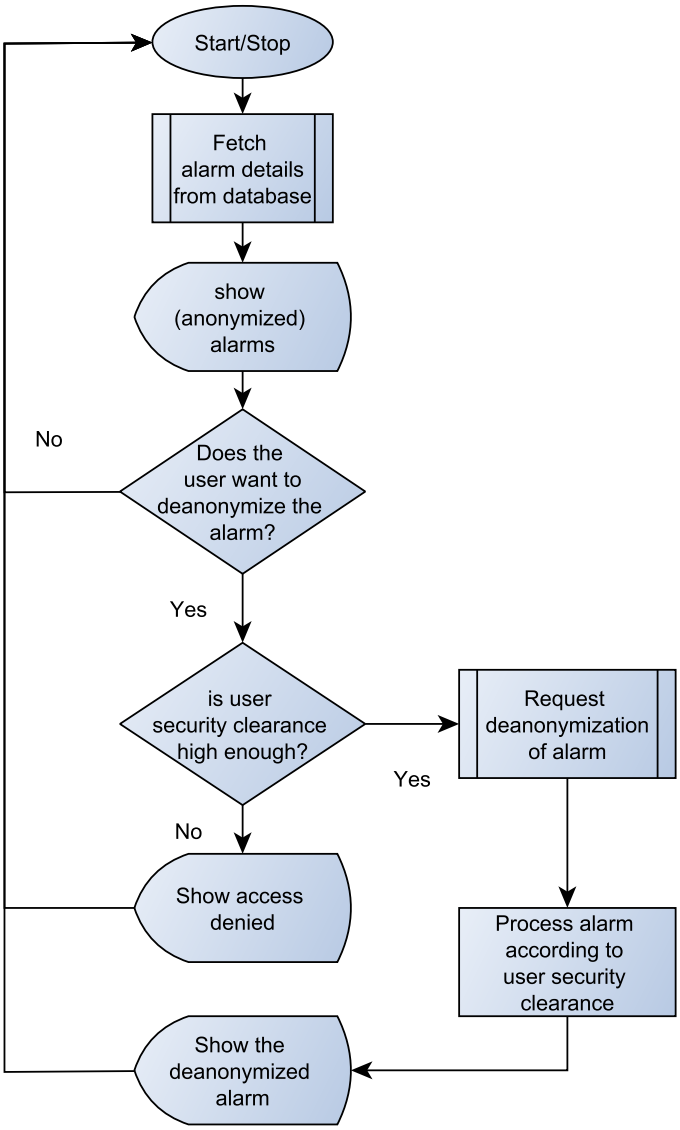


Figure 3.10: Alarm fetching UI flowchart

Chapter 4

Prototyping and Testing

The developing environment was set up on the host workstation (Figure 4.1) using XAMPP[1] and PostgreSQL[2] with its included pgAdmin III[3] database management UI. XAMPP is an Apache distribution containing MySQL, PHP and Perl and is built to make setting up a functioning web server environment easy, regardless of OS[4].

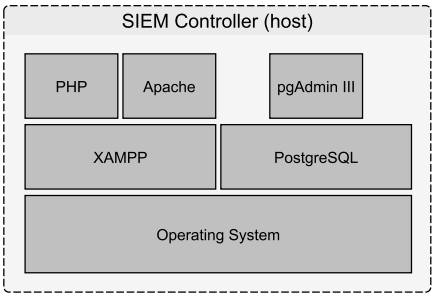


Figure 4.1: The installed components of the SIEM Controller

In order for the PostgreSQL to work with the PHP installation included in XAMPP, it was needed to edit the <*XAMPP/php/php.ini*> file, by removing some comments [6] [5].

[1] `https://www.apachefriends.org/index.html`
[2] `http://www.postgresql.org/`
[3] `http://www.pgadmin.org/`
[4] Works on the latest Windows, Linux and OS X releases
[5] With the newer PHP/Apache versions, by following the guide, Apache crash due to missing dll files if the *LoadFile "C:/xampp/php/libpq.dll"* code is added. It is not required for the test code

PHP was chosen for the server language mainly because it is one of the most used and most powerful web development languages, that suits well for prototyping web based solutions. It also has a large community user-base surrounding it, making it easier to find tutorials and help if unforeseen problems would arise.

A sample IDMEF message was used when developing the code for the testing. The message, with its containing XACML payload removed can be seen in the appendix (Appendix A), and the full test code has been included in Appendix B.

4.1 SIEM Components

Setting up a component overview and flow over the different systems that would make up the SIEM is essential in order to easier get an overview of what components are needed to make the system run. As shown in figure 4.2, the SIEM Controller will consist of four main components: A *User Database*, an *Alert Database*, a *Web Server* and the *IDMEF/XML Parser*. The *XML Parser* receives the anonymized IDMEF messages from the Anonymiser through SOAP over HTTPS/SSL, it then parses through the XML and extracts the information specified. This information is then automatically updated into the Alert database. The *Web Server* is serving the Web UI through HTTPS, and has three members that it can control; the Alert and User Databases, and the XML Parser. It is constantly checking the Alert Database for updates which it presents to the *Web-Browser*. With requests from the user through interactivity, it also has both a SOAP server and client running in order to connect to the *Deanonymiser* for deanonymization of alerts. The SOAP response message would be a standard IDMEF message, and thus there is no need to parse it through the customized XML Parser. This is also to prevent the potentially private information being stored in the database.

to run, so it can be skipped.

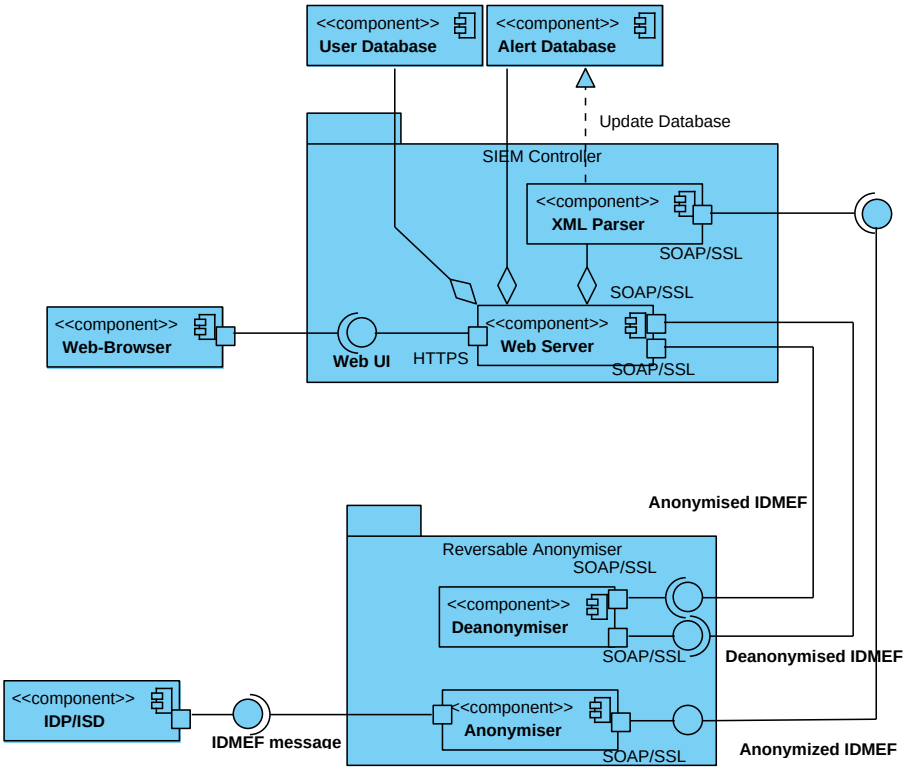


Figure 4.2: High level overview of the SIEM components.

4.2 IDMEF Message Parsing

With the default *DENY* policy, a lot of the information that usually gets parsed and displayed by existing SIEM solutions would be of no use for the SA. As the example shown in screenshot in figure 4.3, taken from an instance of Prewikka (the Prelude-SIEM[6]) that is connected to the Anonymiser, most of the information that the SIEM finds important to show as default are hidden, replaced with

[6] `https://www.prelude-siem.org/projects/prelude/wiki/Prewikka`

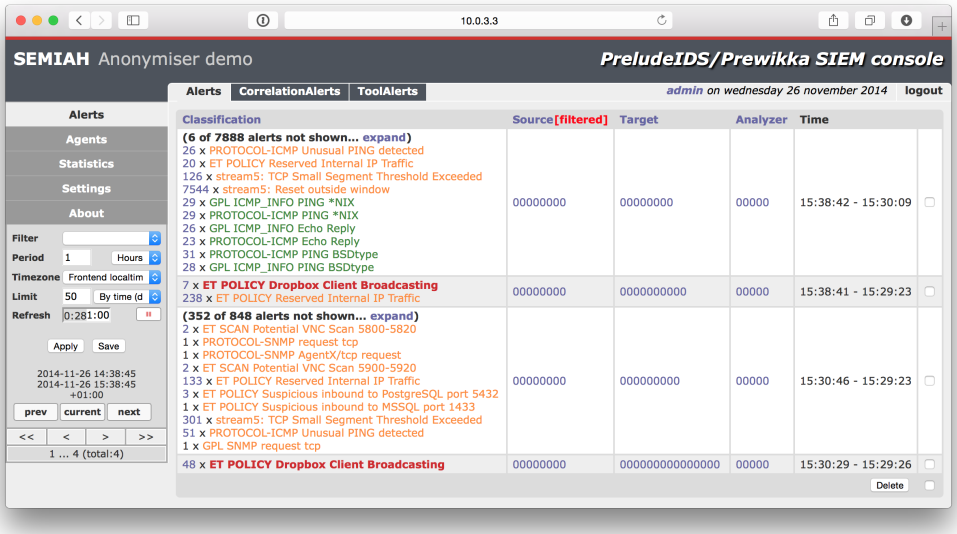


Figure 4.3: Prewikka SIEM connected to the Anonymiser

zero's as specified in the policy. The exception being *Classification*. This further complicate the case beyond the visual component as the SIEM would treat all these values the same: Alerts would be shown as originating from the same source, heading towards the same destination, triggered by the same process, and so on. If the SIEM were to be analyzing the alerts as a SEC, this could result in several critical alerts that would actually be normal traffic (False Positives (FPs)).

As IDMEF messages are implemented in XML, they can be parsed as an XML message. It has been developed several different PHP classes for reading XML files, but they vary in speed and implementation[2]. It may seem that using a Document Object Model (DOM) parser would be the easiest. But it would not necessarily allow the speed a IDP/IDS would require of the SIEM, if the data stream were to high. The DOM parsers would infact insert the whole XML tree to the memory before processing it, and would need to be told where to specifically start and stop the parsing.

The other two of the most used alternatives, Simple API for XML (SAX) and

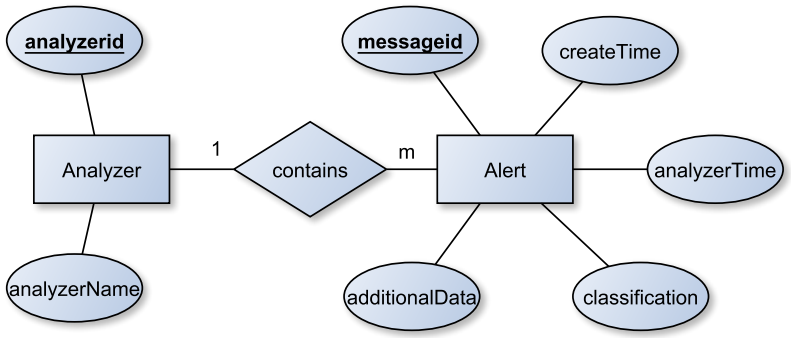


Figure 4.4: The alert entity relation after parsing

Pull Parser, are consistently mentioned around the community as almost equal in speed and performance. The SAX parser reads the XML as a stream of events, from top to bottom, which in return makes it complicated to get specific information when run through a IDMEF message since several nodes and attributes are named the same. An example: if a SAX parser were told to get the value of the weak node *name* related to the strong node *Process* from the IDMEF message shown partially in Code 4.1 on page 35, it would return with two different values, *prelude-manager* and *idmefServer*, even though the only value we wanted was the former. The solution would be to compare the different *pid* values, and separate them, but that would (assumed) require more code and processing time than what would be ideal. The choice would then be left to use a Pull parser, specifically *XMLReader*[7].

The Pull parser reads the XML the same way as the SAX, but it keeps track on where it is at any moment, and thus can be told where in the XML-tree it will find the attribute, and it will stop when it gets the specified information there.

Code 4.1: Sample of an IDMEF message

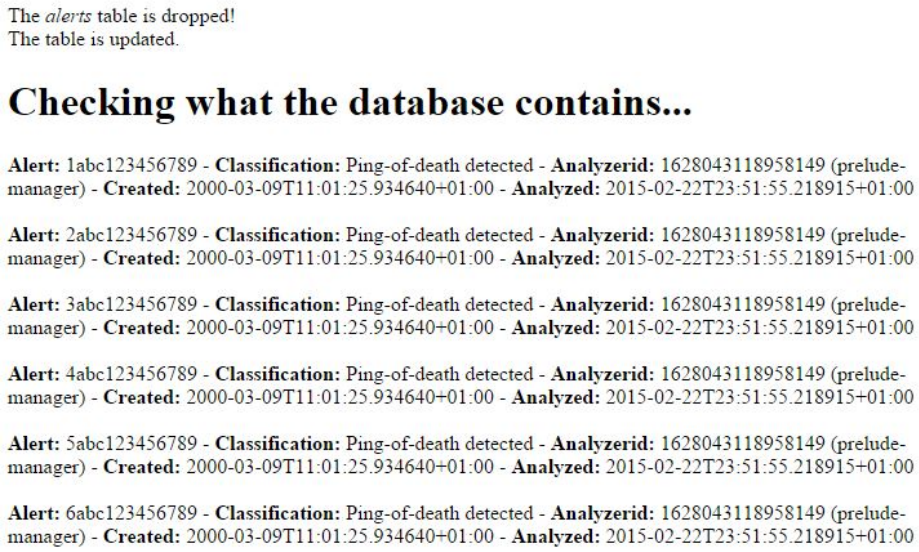
```
1  <Analyzer analyzerid="1628043118958149" name="prelude-
       manager" manufacturer="http://www.prelude-ids.com"
       model="Prelude Manager" version="1.2.5" class="
       Concentrator" ostype="Linux" osversion="3.13.0-29-
       generic">
2    <Process>
3      <name>prelude-manager</name>
4      <pid>1041</pid>
5      <path>/usr/sbin/prelude-manager</path>
6    </Process>
7    <Analyzer analyzerid="3085355587262395" name="PRECYSE
       " manufacturer="http://www.precyse.eu" model="
       idmefServer" version="v0.4" class="PRECYSE ESB to
       PreludeIDS IDMEF proxy server." ostype="Linux"
       osversion="3.13.0-29-generic">
8      <Process>
9        <name>idmefServer</name>
10       <pid>1018</pid>
11       <path>/usr/bin/idmefServer</path>
12     </Process>
```

4.3 The Alert Database.

When looking at the original IDMEF relation entity in Figure 3.1 on page 18 and comparing it to the information that are unanonymized in the sample IDMEF message, the information that was considered to be of most value for use in an alert flow UI got chosen as a test entity. Figure 4.4 shows the resulting relation entity of the alert.

With a the IDMEF message sample, as a reference point, the classification, analyzerTime/createTime and analyzerName are to be considered what are necessary attributes for a security analyst to identify what is going on.

[7] http://php.net/manual/en/class.xmlreader.php


The *alerts* table is dropped!
The table is updated.

Checking what the database contains...

Alert: 1abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Alert: 2abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Alert: 3abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Alert: 4abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Alert: 5abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Alert: 6abc123456789 - **Classification:** Ping-of-death detected - **Analyzerid:** 1628043118958149 (prelude-manager) - **Created:** 2000-03-09T11:01:25.934640+01:00 - **Analyzed:** 2015-02-22T23:51:55.218915+01:00

Figure 4.5: Multiple alerts parsed and inserted to database

After configuring the XMLReader to extract the values of these nodes into an array, the script would then connect to the database and update each table with value specified. A simple script that would print out the results from a *"SELECT * FROM alerts"* SQL query into HTML were created to see that the *Alert Database* were updated with the information sendt.

To simulate multiple alarms, the entire *Alert* element, with all its sub elements, was repeated inside the IDMEF message several times and with some minor edit to their *messageid*. The result printed by the HTML are shown in figure 4.5, show that the parser were able to handle multiple alerts if inside the */IDMEF-Message*. The *additionalData* contains the, encrypted, original IDMEF message, and is stored in the database, but it does not give any value of being presented here because of the amount of encrypted data.

Figure 4.6: The SOAP server publishing its WSDL

4.4 Testing SOAP with PHP.

The Anonymiser has a SOAP client running that sends out anonymized IDMEF messages as they are produced by the Anonymiser. The *XMLParser* would then need to have a SOAP Server running that listens for these messages as they arrive, and forward them to the parser.

After testing different PHP classes made for setting up SOAP servers, nuSOAP[3] ended up being the easiest to use and were chosen for this test. The Anonymiser SOAP client were configured with a WSDL named *DomainStreamService.wsdl* and utilized two XML schema's; *DomainStreamService_schema1.xsd* and *DomainStreamService_schema2.xsd* for describing the anonymized IDMEF message to send. By copying the *DomainStreamService.wsdl* file, and the corresponding schema's, to the host machine, it were possible to set up the server by pointing it to the copied *.wsdl* file. As figure 4.6 shows, the server were publishing the WSDL.

To ensure privacy, the RA are configured to connect securely over SSL using

Code 4.2: DomainStreamService.wsdl service settings

```
1 <wsdl:service name="DomainServiceService">
2   <wsdl:port name="DomainServicePort" binding="tns:
      DomainServiceServiceSoapBinding">
3     <soap:address location="http://10.0.3.1:9990/
      DomainServicePort"/>
4   </wsdl:port>
5 </wsdl:service>
```

port 9990 as shown in the wsdl extraction (Code 4.2). This became a problem since it would need to manually create new certificates specified for the host in order to accept a connection. It was decided to instead simulate the message exchange locally by setting up a SOAP client that sent out the test IDMEF message used for parsing. Using nuSOAP to point the client to the *DomainStreamService.wsdl* published by the server and manually configure the SOAP call to match the XML schema this was done successfully. A packet capture dump using wireshark[8] revealed the transfered message (Code 4.3 and 4.4).

[8] https://www.wireshark.org/

Code 4.3: SOAP exchange: Request

```
1  POST /Server/server.php HTTP/1.1
2  Host: localhost
3  Connection: Keep-Alive
4  User-Agent: PHP-SOAP/5.4.40
5  Content-Type: text/xml; charset=utf-8
6  SOAPAction: "urn:#publishAlerts"
7  Content-Length: 9255
8
9  <?xml version="1.0" encoding="UTF-8"?>
10 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap
       .org/soap/envelope/" xmlns:ns1="http://precyse.eu/ns/
       service"><SOAP-ENV:Body><ns1:publishAlerts><ns1:
       headerconsumerId="SIEM" requestTime="2015-04-22
       23:00"/><ns1:messages>&lt;IDMEF-Message&gt;&lt;
       Alertmessageid="abc123456789"&gt;&lt;Analyzer
       analyzerid="1628043118958149" name="prelude-manager"
       manufacturer="http://www.prelude-ids.com" model="
       Prelude Manager" version="1.2.5" class="Concentrator"
       ostype="Linux"
11 ... ...
12 </ns1:messages></ns1:publishAlerts></SOAP-ENV:Body></SOAP
       -ENV:Envelope>
```

Code 4.4: SOAP exchange: Reply

```
1  HTTP/1.1 200 OK
2  Date: Thu, 23 Apr 2015 19:03:43 GMT
3  Server: Apache/2.4.10 (Unix) PHP/5.5.14
4  X-Powered-By: PHP/5.5.14
5  Content-Length: 238
6  Keep-Alive: timeout=5, max=100
7  Connection: Keep-Alive
8  Content-Type: text/xml; charset=utf-8
9
10 <?xml version="1.0" encoding="UTF-8"?>
11 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap
       .org/soap/envelope/" xmlns:ns1="http://precyse.eu/ns/
       service"><SOAP-ENV:Body><ns1:publishAlertsResponse/></
       SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Chapter 5

Discussion and Future work

This chapter will look at some of the design choices and discuss the reasoning, as well as some of the problems that were faced during the prototyping of the IDMEF message parsing and SOAP server setup.

5.1 Design discussion

While the design itself should be viable for prototyping, it is in general a very high level design. There are several factors, like details in how secret key sharing should be handled. It would need to be fleshed out and prototyped before implemented fully into the design. The solution proposed in this thesis assume that handling the secret keys together with login information and users security clearance levels in the main user database, used for the web interface menu, is a viable solution. Further research on the topic is needed to be sure. Another factor that are not taken into account of the design, are the handling of deanonymized IDMEF messages. Since this format has become the standard for transferring security related messages through the network, it is assumed that there already exists program components that are configured for transferring such messages directly into

existing SQL databases. Open source SIEM software like Prelude[1] also utilized SQL for its alert database, and the *ParseXML* component of the presented solution are made with the function as a plug-in for existing SIEM systems in mind. This should could be the case.

There are several subroutines presented in some of the flow charts (Figure 3.5). These are mostly related to how the web-server would handle different user security clearance and user profiles. These were not directly the focus for the thesis and were left for further explanation if time would allow it. Though some research into UI design were made, there was not enough time to properly present it. Figure 5.1 were the closest form of an imagined alert flow interface prototype that were made.

5.2 Prototyping

The choice of information to extract from the anonymized IDMEF message were made from what was deemed important from a personal perspective and of what information is commonly displayed in a Security Operation Center (SOC) environment. The sample anonymized IDMEF message actually contain more unanonymized information than the seven values shown in figure 4.4, but most of it were of no real value for the sake of prototyping.

It is considered that the *ParseXML* component of the XML Parser could be generated together with the XACML Policy when generating new policies for the Anonymiser, but this is more in the thought process rather than prototype. It could prove valid if there were resources available and should perhaps be studied further.

Since the knowledge level of PHP scripting were very low before starting the thesis project, a lot of trivial problems turned to be very time consuming in the beginning. Similarly, setting up a SOAP server over SSL in order to connect

[1] `https://www.prelude-siem.org/`

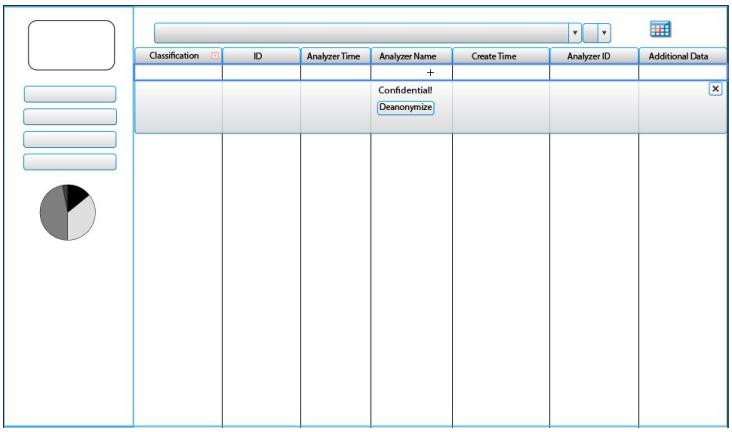


Figure 5.1: A basic representation on how the web user-interface might look.

directly with the Anonymizer became more of a problem than first anticipated. The result of this ended with settling on the proof of a locally transfered message. This were out of the need to shift the focus of the project slightly and to finish it in time. .

Chapter 6

Conclusion

In this thesis it have been proposed a design solution for how the Reversible Anonymiser (RA) can enable privacy in a Security Information and Event Management Solution (SIEM). This design describes the architecture needed so that the RA can be implemented either as a plugin to existing SIEMs, or to lay the groundwork for a new SIEM solution that uphold the principles of Privacy by Design.

While there were some unforeseen problems during prototyping of the solution, the results from the tests done to parsing of sampled anonymized alerts show that importing of usable information from anonymized IDMEF messages in to a PostgreSQL database, as well as transferring said messages locally through SOAP are possible using PHP.

6.1 Future Work

Because of the nature of designing a system like the one presented in this thesis, there are multiple fields of research that can be done going forward. But the most notable ones that does not involve further development and prototyping of the privacy enhanced SIEM's system design.

- When removing a large amount of important information from alerts that get collected in a SIEM, a good deal of usable statistics extracted from this information could be lost. Research into how this affects the main functions of a SIEM solution, and the efficiency of the SA operating it would be of importance.
- Comparison of an anonymized alert flow versus an unanonymized alert flow during different types of incidents in order to locate weakness‘ in an privacy enabled SIEM workflow.

Bibliography

[1] "What is SOAP (Simple Object Access Protocol)? - Definition from WhatIs.com." [Online]. Available: http://searchsoa.techtarget.com/definition/SOAP

[2] *The Performance of Open Source Applications: Speed Precision, and a Bit of Serendipity*. Lulu Com, 2013.

[3] D. Ayala and S. Nichol, "NuSOAP - SOAP Toolkit for PHP," June 2013. [Online]. Available: http://sourceforge.net/projects/nusoap/

[4] J. Burnham, "Gartner Publishes 2014 Magic Quadrant for SIEM," 2014. [Online]. Available: http://securityintelligence.com/gartner-2014-magic-quadrant-siem-security/

[5] A. Cavoukian, S. Taylor, and M. E. Abrams, "Privacy by Design: essential for organizational accountability and strong business practices," *Identity in the Information Society*, vol. 3, no. 2, pp. 405–413, June 2010. [Online]. Available: http://link.springer.com/article/10.1007/s12394-010-0053-z

[6] A. Collins, "How to Integrate postgreSQL Database to XAMPP in Windows," 2014. [Online]. Available: http://w3guy.com/integrate-postgresql-database-xampp-windows/

[7] C. Constantine, "Everything You Wanted to Know about SIEM and Log Management but Were Afraid to Ask – Part 1: What is a SIEM?" 2015. [Online]. Available: https://www.alienvault.com/blogs/security-essentials/everything-you-wanted-to-know-about-siem-and-log-management-but-were-afraid

[8] H. Debar, D. Curry, and B. Feinstein, "The intrusion detection message exchange format (IDMEF)," 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4765.txt

[9] U. Flegel, *Privacy-Respecting Intrusion Detection*. Springer Science & Business Media, Aug. 2007.

[10] G. Greenwald, "XKeyscore: NSA tool collects 'nearly everything a user does on the internet'," July 2013. [Online]. Available: http://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data

[11] D. Lazarus, "Bank of America data leak destroys trust," *Los Angeles Times*, May 2011. [Online]. Available: http://articles.latimes.com/2011/may/24/business/la-fi-lazarus-20110524

[12] J. H. Moor, "How to Invade and Protect Privacy with Computers," in *The Information Web: Ethical and Social Implications of Computer Networking*. Boulder, CO: Westview Press, 1989, pp. 57–70.

[13] R. Pang and V. Paxson, "A high-level programming environment for packet trace anonymization and transformation," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 339–351. [Online]. Available: http://dl.acm.org/citation.cfm?id=863994

[14] J. F. T. reporter and B. B. C. News, "Stuxnet worm 'targeted high-value Iranian assets'," 2010. [Online]. Available: http://www.bbc.com/news/technology-11388018

[15] C. Sanders and J. Smith, *Applied Network Security Monitoring: Collection, Detection, and Analysis*. Elsevier, Nov. 2013.

[16] R. J. Shimonski, "What You Need to Know About Intrusion Detection Systems," Nov. 2002. [Online]. Available: http://www.windowsecurity.com/articles-tutorials/intrusion_detection/What_You_Need_to_Know_About_Intrusion_Detection_Systems.html

[17] N. Ulltveit-Moe, *Privacy-enhanced network monitoring*. Universitet i Agder / University of Agder, 2013, doktorgradsavhandling i informasjons-

og kommunikasjonsteknologi, Universitetet i Agder, 2014. [Online]. Available: http://brage.bibsys.no/xmlui/handle/11250/194485

[18] N. Ulltveit-Moe, H. Nergaard, T. Gjøsæter, and J. Betts, "XACML Privacy Policy Editor for Critical Infrastructures," *Springer-Verlag*, 2011.

[19] N. Ulltveit-Moe and V. Oleshchuk, "A novel policy-driven reversible anonymisation scheme for XML-based services," *Information Systems*, vol. 48, pp. 164–178, Mar. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030643791400091X

[20] S. D. Warren and L. D. Brandeis, "The Right to Privacy," *Harvard Law Review*, vol. 4, no. 5, pp. 193–220, Dec. 1890. [Online]. Available: http://www.jstor.org/stable/1321160

[21] A. N. Yannacopoulos, C. Lambrinoudakis, S. Gritzalis, S. Z. Xanthopoulos, and S. N. Katsikas, "Modeling privacy insurance contracts and their utilization in risk management for ICT firms," in *Computer Security-ESORICS 2008*. Springer, 2008, pp. 207–222. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-88313-5_14

Appendix A

Sample IDMEF message

The *xml* value is replaced with "..." because of problems with linebreak.

```
1 <IDMEF-Message><Alert messageid="abc123456789"><Analyzer
    analyzerid="1628043118958149" name="prelude-manager"
    manufacturer="http://www.prelude-ids.com" model="Prelude
     Manager" version="1.2.5" class="Concentrator" ostype="
    Linux" osversion="3.13.0-29-generic"><Process><name>
    prelude-manager</name><pid>1041</pid><path>/usr/sbin/
    prelude-manager</path></Process><Analyzer analyzerid
    ="3085355587262395" name="PRECYSE" manufacturer="http://
    www.precyse.eu" model="idmefServer" version="v0.4" class
    ="PRECYSE ESB to PreludeIDS IDMEF proxy server." ostype
    ="Linux" osversion="3.13.0-29-generic"><Process><name>
    idmefServer</name><pid>1018</pid><path>/usr/bin/
    idmefServer</path></Process><Analyzer analyzerid
    ="00000000000"><Node ident="0" category="unknown"><name>
    sensor.example.com</name></Node></Analyzer></Analyzer></
    Analyzer><CreateTime ntpstamp="0xbc71f4f5.0xef449000
    ">2000-03-09T11:01:25.934640+01:00</CreateTime><
    AnalyzerTime ntpstamp="0xd894dc0b.0x380ad000">2015-02-22
    T23:51:55.218915+01:00</AnalyzerTime><Source ident
    ="0000" spoofed="yes"><Node ident="000000" category="
    unknown"><Address ident="000000" category="ipv4-addr"><
    address>00000000000</address></Address></Node></Source><
    Target ident="0000" decoy="unknown"><Node ident="0"
```

```
category="unknown"><Address ident="000000" category="
ipv4-addr"><address>0000000000</address></Address></Node
></Target><Classification ident="0" text="Ping-of-death
detected"><Reference origin="cve"><name>000000000000</
name><url
>00000000000000000000000000000000000000000000000000000000000000000000</
url></Reference></Classification><AdditionalData type="
xml" meaning="EncryptedData"><xml>...</xml></
AdditionalData></Alert></IDMEF-Message>
```

Appendix B

insertToDatabase.php

```
1  <?php
2  /**
3   * Created by PhpStorm.
4   * User: Kristoffer
5   * Date: 17.03.2015
6   * Time: 09:15
7   */
8
9  /**
10  * Autoload Classes.
11  * @param $class_name
12  */
13 function __autoload($class_name) {
14   include './lib/class_' . $class_name . '.inc';
15 }
16
17 $dropping = drop_table('alerts');
18 echo "</br>$dropping";
19
20 $alert_log = './prelude-xml.log';
21 $messageId = parseXML($alert_log, 'messageid');
22 $analyzerId = parseXML($alert_log, 'analyzerid');
23 $analyzerName = parseXML($alert_log, 'analyzername');
24 $createTime = parseXML($alert_log, 'createtime');
25 $analyzerTime = parseXML($alert_log, 'analyzertime');
```

```
26 $classification = parseXML($alert_log, 'Classification');
27 $additionalData = parseXML($alert_log, 'AdditionalData');
28
29 create_Table('alerts');
30 $updating = update_alarms($messageId, $classification,
       $analyzerId, $analyzerName, $createTime, $analyzerTime);
31 echo "</br>$updating";
32
33 echo "</br><h1>Checking what the database contains...</h1>"
       ;
34 $results = pgsql_query('SELECT * FROM alerts');
35 if (!$results){
36   echo "Something fishy is going on...";
37   exit;
38 }
39
40 while ($row = pg_fetch_row($results)){
41   echo "<b>Alert:</b> $row[0] - <b>Classification: </b>
         $row[1] - <b>Analyzerid: </b> $row[2] ($row[3]) - <b>
         Created:</b> $row[4] - <b>Analyzed:</b> $row[5]";
42   echo "<br /><br />\n";
43 }
44
45
46 /**
47  * Function for parsing the XML.
48  * Takes only messageid, analyzerid, analyzername,
       createtime, analyzertime, classification and
       AdditionalData at the moment.
49  * @param string $logFile
50  * @param string $tagName
51  * @return array
52  */
53 function parseXML($logFile, $tagName){
54   $xml = new XMLReader();
55   $xml->open($logFile);
56   $result = array();
57
58   if ($tagName === 'messageid'){
59     while ($xml->read()) {
```

```
60          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'Alert'){
61              $result[] = $xml->getAttribute('messageid');
62          }
63      }
64  } else if ($tagName === 'analyzerid'){
65      while ($xml->read()) {
66          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'Analyzer' && $xml->depth == 2){
67              $result[] = $xml->getAttribute('analyzerid');
68          }
69      }
70  }else if ($tagName === 'analyzername') {
71      while ($xml->read()) {
72          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'Analyzer' && $xml->depth == 2) {
73              $result[] = $xml->getAttribute('name');
74          }
75      }
76  }else if ($tagName === 'createtime') {
77      while ($xml->read()) {
78          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'CreateTime') {
79              $result[] = $xml->readInnerXml();;
80          }
81      }
82  }else if ($tagName === 'analyzertime') {
83      while ($xml->read()) {
84          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'AnalyzerTime') {
85              $result[] = $xml->readInnerXml();;
86          }
87      }
88  }else if ($tagName === 'AdditionalData') {
89      while ($xml->read()) {
90          if ($xml->nodeType == XMLReader::ELEMENT && $xml->
                name === 'xml') {
91              $result[] = $xml->readInnerXml();;
92          }
93      }
```

```
94    }else if ($tagName === 'Classification'){
95      while ($xml->read()) {
96        if ($xml->nodeType == XMLReader::ELEMENT && $xml->
              name === 'Classification') {
97          $result[] = $xml->getAttribute('text');
98        }
99      }
100   } else $result[] = "There is no such attribute!";
101   $xml->close();
102   return $result;
103 }
104
105 /**
106  * Dropping table if it exists. Returns a message that it
        is dropped.
107  * @param string $tableName
108  * @return string
109  */
110 function drop_table($tableName)
111 {
112   $db = PgSQL_connect::getInstance();
113   $pgsql = $db->getConnection();
114   $query = "DROP TABLE IF EXISTS $tableName";
115   $ret = pg_query($pgsql, $query);
116
117   if(!$ret){
118     $result = pg_last_error($pgsql);
119   }else $result = "The <i>alerts</i> table is dropped!";
120   return $result;
121 }
122
123 /**
124  * Function for creating the alerts table.
125  * @param string $tableName
126  * @return string
127  */
128 function create_Table($tableName){
129   $db = PgSQL_connect::getInstance();
130   $pgsql_connection = $db->getConnection();
131
```

```
132   if ($tableName === 'alerts'){
133     $query = <<< EOF
134       CREATE TABLE IF NOT EXISTS alerts
135       (
136       messageid TEXT NOT NULL,
137       classification TEXT,
138       analyzerid TEXT NOT NULL,
139       analyzername TEXT NOT NULL,
140       createtime TEXT NOT NULL,
141       analyzertime TEXT NOT NULL,
142       additionaldata TEXT,
143       CONSTRAINT alerts_pkey PRIMARY KEY (messageid)
144       )
145 EOF;
146     $ret = pg_query($pgsql_connection, $query);// sending
          the create table query to the database
147     if(!$ret){
148       $result = pg_last_error($pgsql_connection);
149     }else $result = "The <i>alerts</i> table is good!";
150   }
151
152   return $result;
153 }
154
155 /**
156  * Function for updating the 'alerts' table with
       information from the XMLparser.
157  * @param array $messageId
158  * @param array $classification
159  * @param array $analyzerId
160  * @param array $analyzerName
161  * @param array $createTime
162  * @param array $analyzerTime
163  * @return resource
164  */
165 function update_alarms($messageId, $classification,
      $analyzerId, $analyzerName, $createTime, $analyzerTime){
166   $db = PgSQL_connect::getInstance();
167   $pgsql_connection = $db->getConnection();
168   $sql = array();
```

```
169
170   for ($i=0; $i < count($messageId); $i++) { //going
          through the arrays
171     $sql[$i] = 'INSERT INTO alerts (messageid,
            classification,analyzerid,analyzername,createtime,
            analyzertime) ';
172     $sql[$i] .= "VALUES ('$messageId[$i]','$classification[
            $i]','$analyzerId[$i]','$analyzerName[$i]','
            $createTime[$i]','$analyzerTime[$i]')";
173     //echo "</br>$sql[$i]</br>"; // just for printing what
            the SQL query looks like.
174
175     $ret = pg_query($pgsql_connection, $sql[$i]);// sending
            the insert data query to the database
176
177     if (!$ret){
178       $result = pg_last_error($pgsql_connection);
179     }else $result = 'The table is updated.';
180   }
181   return $result;
182 }
183
184 /**
185  * Just a simple function to make queries less a hassle.
186  * @param $query
187  * @return resource
188  */
189 function pgsql_query($query){
190   $db = PgSQL_connect::getInstance();
191   $pgsql = $db->getConnection();
192   $ret = pg_query($pgsql, $query);
193   return $ret;
194 }
```

Appendix C

SOAP Code

C.1 client.php

```
1  <?php
2  /**
3  * Created by PhpStorm.
4  * User: Kristoffer
5  * Date: 18/04/15
6  * Time: 21:33
7  */
8  require_once "../lib/nusoap.php";
9
10
11 $client = new nusoap_client("http://localhost/thesis/Server
       /server.php?wsdl", true);
12
13 $error = $client->getError();
14 if ($error){
15 $alert = "<h2>Constructor error</h2><pre>". $error . "</pre
       >";
16 }
17
18 $alarm_log = file_get_contents('../prelude-xml.log');
19 $messages = array();
```

```
20 $messages[] = new soapval('message', 'xsd:string,',
       $alarm_log);
21
22 $objDateTime = new DateTime('NOW');
23 $objDateTime->format('c');
24
25 $result = $client->call("publishAlerts", array("header" =>
       array("consumerId" => "SIEM", "requestTime" =>
       "2015-04-20 14:00"), "messages" => $messages));
26
27 if ($client->fault) {
28 $alert = "<h2>Fault</h2><pre>";
29 $alert .= print_r($result);
30 $alert .= "</pre>";
31 }
32 else {
33 $error = $client->getError();
34 if ($error){
35 $alert = "<h2>Error</h2><pre>" . $error . "</pre>";
36 }
37 else {
38 $alert = $result;
39 print_r($alert);
40 }
41 }
42
43 //Debugging the nusoap messages.
44 // echo '<h2>Debug</h2>';
45 // echo '<pre>' . htmlspecialchars($client->debug_str,
       ENT_QUOTES) . '</pre>';
46
47 // //showing the nusoap request and response messages.
48 // echo "<h2>SOAP Request</h2>";
49 // echo "<pre>" . htmlspecialchars($client->request,
       ENT_QUOTES) . "</pre>";
50 // echo "<h2>SOAP Response</h2>";
51 // echo "<pre>" . htmlspecialchars($client->response,
       ENT_QUOTES) . "</pre>";
```

C.2 server.php

```
1  <?php
2  /**
3  * Created by PhpStorm.
4  * User: Kristoffer
5  * Date: 18/04/15
6  * Time: 21:33
7  */
8  require_once "./lib/nusoap.php";
9
10 function publishAlerts($header, $messages){
11 //print_r($header);
12 //$xml = $messages[0]->saveXML($messages[0]->
       documentElement);
13 //echo $xml;
14 // print_r($header);
15 // print_r($messages);
16 // return $messages;
17 //$length = sizeof($messages);
18 /*for ($i = 0; $i < $length; $i++){
19 echo $messages[$i];
20 }*/
21 }
22
23 $server = new soap_server('./Server/DomainStreamService.
       wsdl');
24
25 $POST_DATA = isset($GLOBALS['HTTP_RAW_POST_DATA']) ?
       $GLOBALS['HTTP_RAW_POST_DATA'] : '';
26
27 // pass our posted data (or nothing) to the soap service
28 $server->service($POST_DATA);
29 exit();
```