Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Effective Visualization of Amplification Attacks in Amplifier Networks

Michael Köpferl

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Effective Visualization of Amplification Attacks in Amplifier Networks

Effektive Visualisierung von Amplification Attacks in Amplifier Networks

Author	Michael Köpferl
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisors	Oliver Gasser, Felix von Eye
Date	September 15, 2015

Informatik VIII
Chair for Network Architectures and Services

Abstract

Denial of Service attacks are a threat to computer networks. One variation of them which is even more dangerous are Amplification attacks. Although this attack type is researched well, including multiple proposals to overcome the problem, new attack vectors arise frequently. There are multiple approaches using different detection methods on the victim side. On the amplifier side, where the problem has to be dealt with to reduce the impact of such attacks, surveys present new affected protocols frequently. However, the attacks adapt to the new situation dynamically and abuse other susceptible faults. Therefore, we research the visual approach to gain knowledge about the attacks on basis of an existing attack detection approach. Using the output data of the detection software, we propose software that generates a graphical representation. This can be used to evaluate the attacks within the amplifier's network to immediately receive detailed information about them.

Zusammenfassung

Denial-of-Service-Angriffe haben sich zu Amplification Attacks weiterentwickelt, wodurch die Gefahr für Netwerke steigt. Obwohl diese Art des Angriffs gut untersucht ist und es viele Vorschläge zur Lösung des Problems gibt, entwickeln sich regelmäßig neue Angriffsvektoren. Für die Seite des Opfers gibt es mehrere Vorschläge mit unterschiedlichen Herangehensweisen. Auf Seiten des Amplifiers, wo das Problem gelöst werden muss um die Auswirkungen solcher Angriffe zu verringern, werden regelmäßig neue anfällige Protokolle durch wissenschaftliche Untersuchungen vorgestellt. Dennoch passen sich die Angriffe dynamisch an die neue Situation an und nutzen neue unbekannte Schwachstellen aus. Daher forschen wir an einer visuellen Herangehensweise, um Wissen über die Angriffe auf Basis eines existierenden Angriffserkennungsansatzes zu erlangen. Wir stellen Software vor, die auf Basis der Ausgabe der Erkennungssoftware eine graphische Repräsentation der Daten erstellt. Diese kann genutzt werden, um die Angriffe im Netzwerk des Amplifiers unverzüglich einzuwerten, um detaillierte Informationen darüber zu erhalten.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, downtimes of individual systems or bigger parts of networks are dangerous for business operations and even public safety. Thus, attacks on network infrastructures are a threat to economics, as the internet is a crucial basis to all kinds of communication as well as critical infrastructures.

Denial of Service attacks are one option for attackers to bring down a victim's system by exhausting its resources with a large amount of packets sent to it. Nonetheless, this is also very resource-intensive for the attacker herself. Thus, attacks are commonly conducted by several attackers at once, combining their resources. However, these attacks have evolved to more advanced Amplification attacks, which utilize the fact that some kinds of requests entail a much larger answer. Combined with a forged IP address or conducting the attack from within the victim's network, this overwhelms the target network very fast. There were many attempts to analyze flaws allowing for these attacks. Previous work examined targets and evaluated impact. Despite attempts to sanitize configuration flaws [1, 2], new possibilities are found frequently [2]. Also, attack detection [3, 4] is an important step, but not a final solution, as it does not allow learning about the attacks because it lacks a comprehensible output.

As visualization evolves and facilitates evaluation of large amounts of data, we aim to provice knowledge about these attacks this way. Therefore, our goal is to contribute analysis options to the topic by implementing specialized visualization on basis of a previous detection approach by Böttger [3].

Within this thesis, we use the female form for the attacker, while using the male form for anything else, mainly the person of the network operator.

1.1 Scientific questions

In this thesis, we propose a toolkit which informs about Amplification Attacks in real-time and helps analyzing and understanding them. That way, the underlying problems in system and protocol design can be solved to prevent further exploitation. To achieve this goal, we want to answer the following scientific questions. For each question, we give additional background on its meaning here. We will refer to the question numbers later again for an answer.

Q1 *How can we effectively visualize amplification attacks such that a network operator can easily detect them?* This question can be split into the parts of effective visualization and some form of warning in abnormal cases. An answer to the latter could be a warning system to alert the network operator in case of a detected attack. Therefore, the attack has to be detected automatically to be able to classify it as abnormal behavior. Following that, a warning has to be sent out using a communication interface, e.g. a GSM gateway or a mail server. Then, the network operator can check within the system for the cause of the alert. Therefore he can use the timestamp of the message he received as a search criterion. Thus, our task is to develop visualization that immediately informs about the main attack parameters only requiring this information.

Q2 *How can we recognize, which internal and external systems and networks are affected?* To allow for this, detailed investigation on the affected systems must be possible. Therefore, we need information to classify the attack. Possible classification characteristics could be size and severity. Moreover, we require information about affected services and systems. This data hat to be combined to be able to focus on the important events immediately.

Q3 *How can we react accordingly by shutting down or limiting access to systems or services?* This and the next question correlate to each other, as both seek for ways of reaction to an attack. Here, we focus on short-term reactions like shutting down services or taking systems out of the network. Therefore, we need the information from *Q2* and *Q1* to be able to react and determine the affected systems and the type and size of the attack to assess an adequate reaction.

Q4 *How can an attack be investigated later in detail to learn from it?* After remediating an attack using *Q3*, we want to be able to investigate the attack in detail to determine the root cause. Therefore, the visualization has to answer which systems and services were affected, like in *Q2*, but we also need as much information about the attack as possible. Therefore, we aim so store the event information for later re-examination.

Q5 *Does the visualization help to identify false positives or false negatives?* Finally, we want to be able to assess the alert to determine whether it was an attack and differentiate it from false positives. In addition to that, we also want to interpret other eye-catching traffic for suspicious patterns. We will present ideas on how to achieve this in chapter 6.

1.2 Outline

This thesis is structured as follows: In chapter 2, we will introduce the necessary terms to understand the topic and present the basic components of our toolset. Next, we will present related work in chapter 3, split up into approaches to detect Amplification attacks, a listing of vulnerable protocols as well as general and network security specialized visualization approaches. Following that, we will present our proposed System Architecture in chapter 4, describing the black-box view of our design to facilitate the understanding of the actual implementation in chapter 5. We will then check the developed system against our scientific questions to determine functionality coverage in chapter 6. Finally, we will summarize our findings and propose open topics for future work.

Chapter 2

Background

Before diving deeper into the topic, we will use this chapter to clarify some important terms we use in the following. Therefore, we first classify the term *Amplification attack* in the field of network attacks, talk about the environment and the prerequisites it needs to fulfill for the attack to work. We also discuss possible impact and detection options.

2.1 Attack classification

Attacks on computer systems can be split into several groups according to different criteria [5]: By *attack source*, you can split the field of attacks into internal from direct access to a system or application vs. external through some kind of network connection. By *impact*, attacks can be classified as data disclosure or manipulation, *Denial of Service* (DoS), among several others. Data manipulation means that the attacker changes data on the system purposefully to attain some goal, while data disclosure means copying data from the system and maybe publishing it. Using a DoS attack, the attacker in some way disables the attacked system or some service on it.

Other possible classifications contain the type of the attacked system or service, the assessed impact severity, the exploited vulnerability or the pursued objective of the attacker.

2.1.1 Denial of Service attack

As we have clarified above, a DoS attack aims at disabling the attacked system or a service it offers. This can be achieved in several ways: to conduct a *malformed packet attack* [6], the attacker could send a packet which has special impact on the victim like a system crash, thus disabling it. An example would be the Ping Of Death [5] attack, which abuses bugs in the network stack of affected operating systems to crash the victim's computer using a buffer overflow attack. Apart from this, there are lots of attacks that aim at overwhelming a target device by sending a large amount of packets. These can be divided into *Resource Depletion* or *Bandwidth Depletion* attacks [6]. They intend to overrun a host by sending large amounts of computationally intensive packets or a host or network by sending a lot of, sometimes very large packets to fully saturate the inbound network link of the victim. Our approach focuses on the detection of those large amounts of packets, as detection is feasible on a network level for these attacks. Malformed packet attacks can be better detected on the target host, as presented later in chapter 2.3.

2.1.2 Distributed Denial of Service attack

A *Distributed* DoS attack (DDoS) differs from a normal DoS attack by making use of a large amount of attackers. The Agent-Handler model [6] describes a common structure used to conduct these attacks: the handlers are responsible for communication between attacker and agents and therefore interconnected. The agents are also called secondary victims or zombies. Therefore, the attacker has to send an instruction to a handler controlling the required amount of agents to conduct an attack on the (primary) victim. This way, the attacker cannot easily be detected, while controlling a large amount of attacking devices.

2.1.3 Reflected Denial of Service attack

A *Reflected* DoS attack (RDoS) differs from the ones mentioned before by utilizing a remote system as a traffic reflector for an attack. This can be achieved by forging the source address of outgoing packets which require an answer by the receiver. Therefore, the answer will be reflected to the victim. The reflector cannot easily detect the attack, as it is indistinguishable from a legitimate request for him. In addition to that, neither the reflector nor the victim knows where the attack came from. [7]

Distributed and reflected DoS attack schemes can be combined to a *DRDoS* attack, making use of their joint advantages. [6, 8, 9]

2.2 Amplification attack

An Amplification attack belongs to the field of DoS attacks, as its aim is to flood a computer system, network or connecting link and thus make it unavailable to legitimate users. However, instead of directly flooding a system, rendering it unresponsive to legitimate usage, it achieves this goal by using an amplifier. An amplifier can be any

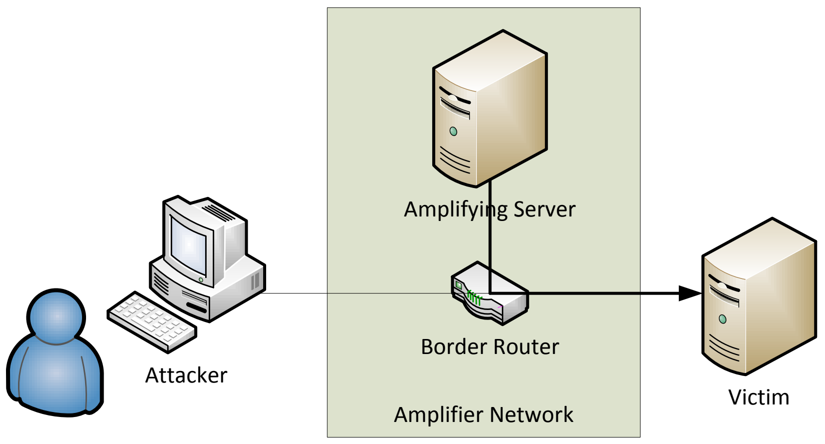


Figure 2.1: Amplification setup

service which answers with a higher amount of data than the query size[1]. As the attacker generally aims at a remote system which she is outside of, she has to carry out a reflected DoS attack which is able to hit a remote target, usually achieved by sending a forged source IP address with the query. Using this, the attacker sends queries to the amplifier which in return sends the amplified answers to the victim. [6, 10]

Amplification attacks can make use of several faults outlined later. Thus, it is possible that the answer packet to a request is much larger, but it is also possible that an answer packet is repeated many times. The difference in size or packet count is used to calculate the Amplification Factor α:

$$\frac{\text{TrafficToVictim [B]}}{\text{TrafficToAmplifier [B]}} = \alpha \tag{2.1}$$

2.2.1 Impact

An Amplification attack allows the attacker to flood the victim with a lot of traffic, only spending a relatively small amount herself. Thus, she can run this type of attack even with low resources on computing time and network bandwidth. Due to that, her attack cannot be detected easily. For the amplifier, the attack looks like legitimate traffic which cannot be separated from normal requests due to IP address spoofing. Therefore, without any of the detection approached outlined during this work, he cannot distinguish the

[1]Apart from the amount of data, there are other possibilities to do an Amplification attack which will be outlined later in this chapter.

attack from a high volume of traffic or even notice it, depending on the size of the attack and the usual load on his systems. The victim, however, can commonly detect the attack quite fast, as the network link is expected to be saturated, because usually the attacker selects an amplifier with a much faster connection than the victim has. Also, the internet provider of the victim could detect the attack due to suspiciously high traffic volume without any corresponding requests from the victim. Due to that, he could block the amplifier as a sender of the packets from his network, and thus make its services unavailable to customers. This unavailability could imply sales or repudiation losses, for example. In addition to that, costs could incur for the amplifier by the high amount of traffic, or legal problems could arise. Therefore, an amplifier network should be protected against such attacks. [6]

2.2.2 Remediation

In chapter 2.1.1 we stated that the root cause of reflection attacks is IP address spoofing. Therefore, the IETF published BCP 38 [7] containing a solution to the problem. The problem of IP address spoofing exists due to the fact that many routers do not verify the source addresses of the incoming packets. Therefore BCP 38 proposes *ingress filtering*, which means that source addresses of packets are checked against a list of addresses expected to arrive from that interface. If that would be done on every router, address spoofing would be impossible. Although this proposal is more than 15 years old, the problem still exists. Due to the fact that this solution cannot be incorporated by a single network to solve the problem, but must be rolled out broadly, it does not help us.

Thus, we focus on remediation of such attacks using a detection approach. Detection is possible on the victim and amplifier side. Solving the problem, however, is only possible on the amplifier, as the victim is not capable of acting against the attack. [10]

Additionally, the attack can be solved by disabling public reachability of affected services. Thus, our approach focuses on finding such services using visualization to be able to eliminate the issues, as described in *Q4*. Within the related work chapter, we will describe several examples and mitigation options.

2.3 Introduction to Security Software

Within this thesis, we heavily rely on two groups of security products: Intrusion Detection Systems (IDS) and Security Information and Event Management systems (SIEM). Before we continue, we therefore have to clarify the terms. We thus describe the main tasks and features of these two product types.

Viinikka et al. [11] give a broad introduction into the topic which we thus take as input. According to them, the main reason for the necessity and development of such software is insufficient access control to computer systems. Due to that, it is an IDS' task to detect events which breach the security policy. To achieve that, it analyzes the data stream in question, which in our case is the data traveling through the network interfaces of the border router. Two different approaches to detect suspicious events exist: misuse-detection and anomaly-detection. The first one depends on a rule configuration representing the security policy as input, while the latter one tries to detect suspicious events by reacting to anomalies within the data stream, and therefore needs no explicit ruleset. Wherever we know the attacks we want to secure our systems against, we should therefore define rules for misuse-detection carefully to catch all cases. This also applies to our approach. However, if we do not know which attacks to expect, anomaly-detection can be an option to identify questionable actions. An IDS can have a special task like monitoring one runtime environment or service, while it can also be responsible for the security of single host (*H*IDS) or a whole network (*N*IDS). Therefore, the existing systems have different functionality. It is then the task of a SIEM system to collect the alerts the different IDS systems, also called sensors, created and combine them to provide a global[2] view of the security state. This is achieved by functions like alert correlation, display and threat management. The collection of data from the sensors is done by logfile analysis or exchange via specified interfaces. A SIEM system also has the responsibility to alert the network operator in configured cases, e. g. via mail.

2.4 Software

As a practical part, we have implemented visualization for Amplification attacks. Therefore, we have chosen existing standard systems to build upon which we describe in the following. As a general overview, we use Suricata for detection of Amplification attacks, Prelude as event storage and database functionality to close the gap by supporting aggregated and sorted output to D3.js. Figure 2.2 on page 9 gives a graphical representation of the system design.

2.4.1 Suricata IDS

Suricata [12] is a Network Intrusion Detection System (NIDS), also including an intrusion prevention module which allows it to detect and also block network traffic according to a predefined rule set. Suricata knows many high level protocols due to its included protocol identification, allowing for easy and complete rule creation on a basis of protocol names instead of ports. For detected events, Suricata can log address information to a file, or even store the whole packet data. Likewise, it provides interfaces to some other tools in the IDS field. However, Suricata is not able to output

[2]Global in this case means including all systems within the network or company.

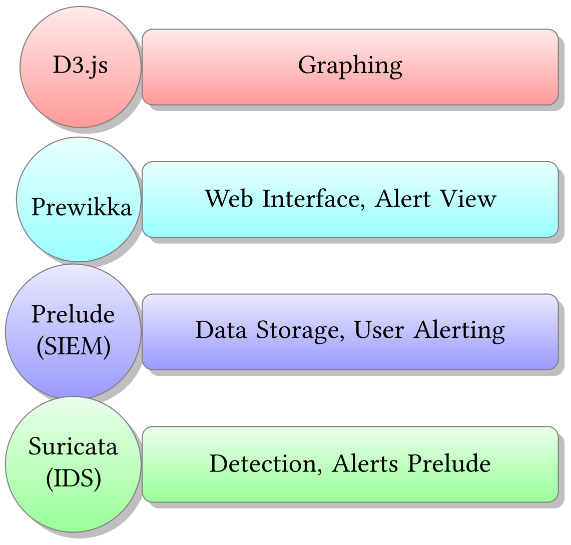

D3.js
Graphing
Prewikka
Web Interface, Alert View
Prelude (SIEM)
Data Storage, User Alerting
Suricata (IDS)
Detection, Alerts Prelude

Figure 2.2: Software

event data in any easily understandable form for large amounts of data. The Suricata Engine is multi-threaded, thus scalable, and automatically splits the workload of packet capturing, processing and detection onto multiple CPU cores, This allows for usage also in large environments without configuration or scalability problems. Suricata is an open-source project maintained by the Open Information Security Foundation (OISF) which is a non-profit organization, and supported by various development members. As of writing the current version of Suricata is 2.0.8, released May 6, 2015. In many of the base features like logging format or rule definition, Suricata is very similar to SNORT IDS.

During the former work by Timm Böttger [3] which we build upon, the Suricata IDS has been extended by Amplification attack detection. Also, Suricata supports alerting for configured network events using the IDMEF, a standard for alert exchange within IDS systems. Due to these features, we reuse Suricata for the detection in this work. We also define an extension to the rule set and the standard IDMEF interface to include additional information in chapter 4.

2.4.2 Prelude SIEM and Prewikka web interface

Prelude [13] is, as described by the authors, a "universal Security Information & Event Management (SIEM) system", which itself is agentless. This means that it is not directly able to detect network events. However, it is also universal which means that it can basically work with any kind of input by evaluating output files. Apart from evaluating log files, it also can receive IDMEF messages which is one of the reasons why we use it. The knowledge Prelude learns from the attached input data is stored to the Prelude database which is designed according to the IDMEF message layout. This fact simplifies interface development, as it is easily comprehensible and well documented. Evaluation of the database entries is possible within Prewikka, the web interface to Prelude which allows for sorting, filtering and grouping of alerts according to alert type, address and time criteria. We therefore use Prelude as a database backend and Prewikka as a user interface which we build visualization upon.

The current version of Prelude Open Source Edition (OSS) is 1.2.6, released August 2015 by Systèmes d'Information. There also is a Pro Edition which has additional features, but is closed source and thus not usable for us as an extension basis.

2.4.3 D3.js

For the visualization tool, we need a highly extensible platform to build upon to create a diverse set of graphs on top of it. As an input type, we require the import of tables we receive as an output from Prelude database queries. Also, we require basic support for aggregation of entries to reduce the number of database query definitions, while still being able to rearrange graphs in any dimension to outline important details. For usability of the visualization, we require flexibility to allow the network operator to change the view to data range and types according to his needs to simplify analysis of attacks.

As such flexibility can be achieved easier using a client-side solution, and we aimed at integrating it into Prewikka which is running on the server-side, we looked for a web-based solution and found the JavaScript Library Data Driven Documents [14, 15] (D3) to support our requirements. D3.js, current version 3.5.6 released Jul 4, 2015, is maintained by Mike Bostock as a GitHub project with many supporting developers and released under the open-source BSD license. Complete developer documentation and many examples on the internet further facilitate usage and extension of D3.js according to our requirements.

To connect Prelude to the D3.js library, an interface is required to extract the datasets from the database in an aggregated and sorted file format which D3.js understands as standard input. We will design and implement this as a part of this thesis, as outlined in chapter 4. In chapter 3.2, we will present D3.js in more detail and justify our choice.

Chapter 3

Related work

In the previous chapter, we introduced the necessary background which we now build upon to outline the literature related to our work. We will split this into the following parts: First, we will focus on Amplification attack detection, where we will contrast other approaches of Amplification attack detection with our visualization. Afterwards, we describe visualization toolsets which we evaluated during this work and outline their differences which lead to the choice we introduced in chapter 2.4.3. Finally we will look at visualization, where we compare existing attack visualization techniques to our development. For reference within the evaluation, we will also present a subset of protocols vulnerable to Amplification attacks for a deeper understanding of the threat and its forms. For each related work, we will give a short summary of the relevant parts and subsequently set it in context to our approach.

The literature related to the IDMEF standard which forms the interface from Suricata to Prelude will be outlined in chapter 4 together with the developed interface.

3.1 Amplification attack detection

There were numerous previous approaches to detect Amplification attacks. However, many of them focused on detection at the victim. As we focus on attack visualization at the amplifier, only parts of these approaches are usable for us. Therefore, we will first introduce the relevant parts of this side, and eventually concentrate on detection approaches at the amplifier. As DNS Amplification attacks are a well-researched topic due to numerous previous attacks [16], we will refer to these quite often. However our approach is not protocol-specific.

3.1.1 Victim-based detection

At the victim, detection of an Amplification attack is quite easy, due to incoming traffic that cannot be matched to any existing connection on the edge router [17, 18]. The exception is a network including externally reachable servers, which is also covered by the research presented below. However, remediation is difficult or not possible at all, as the traffic induced by it can utilize large amounts or all of the incoming bandwidth. Due to that, legitimate traffic cannot be transferred anymore. Therefore, a victim that detected an Amplification attack is dependent on his provider to block the traffic on his behalf [8].

Kambourakis et al. [17] propose a detection algorithm for DNS Amplification attack detection based on the matching of incoming and outgoing packets. They start by classifying Amplification attacks, like we do in chapter 2, but not as detailed, as they do not differentiate between resource and bandwidth depletion. Subsequently, they give examples of conducted attacks against DNS servers and explain how attacks are conducted actually and describe the vulnerabilities within the DNS system allowing this. Their general detection approach makes use of the fact that there has to be a DNS query originating from their network to justify an answer, and answers without a query packet are evil. Therefore, they operate on basis of a strict one-to-one mapping for DNS requests and responses which they measure on an edge router of their network. If an incoming DNS packet can be matched to an outgoing one, it is therefore legitimate. However, on receiving an incoming packet with no preceding request, it is classified malicious. If a predefined amount of packets is reached, an alert is generated and the sender is blocked. In their evaluation, their approach proves useful. However, the database keeping state of the system grows very large, as they have to store source and target port and IP addresses for every connection attempt. They conducted the evaluation and attack simulation within their university network, which also was accessible via internet.

Sun et al. [18] improved that algorithm by not keeping state about every connection, but only comparing the amount of incoming to outgoing packets. They argument that in normal operation, the difference in number should be minimal due to the fact that there exists a response to every packet. As they count packets within fixed length timeframes, it could however happen that a response is captured in the timeframe following to the one with the query. Thus, minimal differences can appear. Taking that into account, they set a threshold to differentiate between normal operation and an attack scenario. This way, they overcome the state keeping problem of Kambourakis et al. and create a solution that uses a static amount of memory and low computing resources also on heavy usage.

As both did detection on the victim side, their approach is not useful for us, because we do have a request for every response. However, the comparison of incoming to outgoing packets, although on a packet size and content level, can help us to classify an attack.

The reduction on state keeping could however be useful to reduce the amount of data processed for a summary graph.

3.1.2 Amplifier-based detection

In chapter 2.1.1 we raised that RDoS attack detection on the reflector is difficult due to the fact that the incoming RDoS queries are indistinguishable from legitimate traffic. Due to the shared structure between those and Amplification attacks, this is also valid here for the amplifier. Due to that, there have been several approaches to facilitate this:

Rossow [2] published a survey of protocols vulnerable to Amplification attacks, including his findings about the number of vulnerable hosts on the internet. He also installed honeypots for known vulnerabilities to determine their exploitation. Furthermore, he conducted and evaluated measurements to find Amplification attacks. His findings give us a lot of information about actual attacks which we can use as input to determine visualization requirements. However, our goal is a general approach, while Rossow focused on a set of application layer protocols. Still, it gives us valuable input on possible attack vectors, for example TCP, which we will describe in section 3.4.3. In addition to that, his findings inform about the actual flaws which enable an attacker to abuse the mentioned services as amplifiers.

Böttger (et al.) [3, 4] contributed a software solution that is able to detect amplification attacks. Therefore, they extended the Suricata IDS to support detection of amplified packets. The implementation is based on pairflows, which represent the communication between one source within the amplifier network and a target outside of it. A pairflow is defined using the IP addresses of source and destination hosts, the source port and the number of packet payload bytes sent from server to client and vice versa. This data is then used to calculate the amplification factor for corresponding pairflow. Using their implementation, they conducted measurements at the border of the MWN network, which interconnects the Munich research institutions and universities to each other and the internet. Using the captured data, they developed criteria to classify the packets:

Request and Response Packet Size Similarity is used to determine the similarity of packet sizes of request and response. They arguments that an attacker is limited to a small set of queries which lead to an amplified answer. Therefore, the packet size must be very similar. Due to that, also the packet size of the responses is expected to be very similar, as a small set of queries results in a small set of answers. Therefore, they calculate the proportion of lengths for incoming and outgoing packets separately and combine it into this criterion. They evaluated that packets exceeding a threshold of 75% are most likely amplification attacks.

Request and Response Payload Similarity is used to determine the packet content similarity using a zip compression algorithm. They calculate the similarity by zipping the concatenated packet data and then comparing the length of the compressed packets to the length of the uncompressed packets. Finally, the delta between the calculated value and 1 is the result. Again, this is done for queries and responses separately and based on the mentioned argumentation.

Unsolicited Messages: A host that receives unsolicited traffic usually answers with ICMP port unreachable messages and thus tells the sender that it didn't expect the packet. Thus, the number of these ICMP messages is an indicator that there probably is a type of RDoS attack ongoing.

IP Spoofing: They argue that IP spoofing can be measured by comparing TTL values of incoming packets from the attacker to conducted hop count measurements to the victim. If these numbers are not equal, they classify the corresponding pairflow as susceptible. However, outgoing measurements are not always possible due to blocking firewalls or overloaded victims. In addition to that, there is guessing involved to interpret the incoming TTL values. We will outline this in detail later, when we describe the visualization of this criterion.

Within the thesis by Böttger [3], also a scoring system was proposed to use size similarity, unsolicited messages and IP spoofing as indicators. For the subsequent paper, Böttger et al. [4] disregarded the two latter criteria as not precise enough. Therefore, also the scoring system was dropped. Using these described values, they conduct another measurement to give proof of their criteria. They conclude with a list of limitations of their approach, which, however, are mostly of hypothetical nature. For the visualization, we will use all of their proposed criteria, as they give important or at least supporting information.

3.2 Visualization toolsets

Data Visualization is a relatively young research field which is being heavily worked on. We therefore picked up two different approaches that meet our general requirements from chapter 2.4.3 and the scientific questions. We compared their work and goals to each other and evaluated them against our requirements.

The aforementioned D3.js visualization framework is presented by Bostock et al. [15] in a paper describing their goals and starting points. They start by describing a set of disadvantages of other toolsets, including their previous work on ProtoVis which they wanted to overcome by their new approach. The first problem was the breaking of existing standards by toolsets building upon and encapsulating these and thus making them unavailable for the developer. Thus, knowledge about standards is wasted and new users have a long learning phase. This eventually leads to low efficiency. Inferior documentation and restricted debugging possibilities combined with the former argument result in impaired accessibility. Additionally, encapsulation can lead to reduced expressiveness and performance drawbacks by not utilizing other optimized technologies. Due to that, it was their goal to develop a new toolset with compatibility, debugging and performance on their minds to overcome aforementioned problems. Therefore, their approach, D3.js, is rather a visualization framework, as it only interfaces preexisting functions. It depends on existing browser technologies like CSS, SVG and HTML to format the page and its elements and to create a graphical representation. Therefore, it also makes use of existing interaction and animation technologies of modern web browsers. Next to a data import interface which is also capable of aggregation and sorting, it has a predefined set of graph layouts and a set of high-level modules for special visualization. In the remainder of the paper they introduce their design in detail, compare code samples of several approaches and evaluate usability and performance, where their approach scores well and works as expected. As this approach meets our requirements by a preexisting set of graphs and samples we can rely on and also allows for other graphs by its framework design, we will utilize D3.js for our visualization.

A different approach that could have been useful within this thesis was introduced by Wongsuphasawat et al. [19] towards the end of the thesis. They contribute the Voyager visualization browser which utilizes a data-driven design. This means that the data to be visualized is selected first, and only subsequently a set of possible graphs is shown by the system which the user can select. They want to enable users to look at previously disregarded data or views. This is achieved by providing automated graph generation within a browser-based proposal system. In the remainder of the paper, they present their user interface and recommender system and compare their approach to a conventional approach using manual graph definition. They conclude that their approach facilitates and encourages broader exploration of data due to its recommender system and automated graph generation. Voyager does, however, not include all graph types we will define in chapter 4.4. Its approach does for example deny an easy extension by geographical maps or other specialized graph types. Therefore, it is not a candidate for us, although it offers interesting additional functionality. The system looks promising for visualization of an unknown dataset. Therefore, it would be interesting for us to further evaluate the header fields and content of packets belonging to Amplification attacks, as it could facilitate additional evaluation. However, packet data would have to be preprocessed and formatted for Voyager which could only be done by evaluating a large dataset of Amplification attacks and therefore is out of scope for this thesis.

3.3 Visualization approaches

In this section we will outline other visualization approaches and check their ideas for applicability to Amplification visualization.

Yu et al. [20] present a visualization analysis tool focused at detection and analysis of DNS Amplification attacks. Their approach also includes an IDS system for detection and a very basic SIEM-like database for packet storage. The database is evaluated by a GUI using an algorithm to count UDP packets that belong to DNS port 53 which they list within a graph. The time-series representation displays packet count per timeunit and therefore shows a sudden raise in the graph for suspicious events. As an Amplification attack consists of a lot of packets, the GUI alerts the network operator visually or acoustically when a predefined threshold of such packets is exceeded, letting him view the graph and decide on an action. In addition to that, the GUI allows filtering, grouping and sorting of the events as well as displaying packet content. These are noticeable similarities to our approach, as they give additional information for diagnosis of the attack. However, their setup can be overwhelmed by an attack very fast, as there is no limit in packet storage, like in Böttger [3]. Also, they do not differentiate between legitimate and evil traffic, but just count packets. Thus, high usage of a service could be misclassified as a false positive. Also, by just focusing on the known port, attacks to other services can be missed, leading to false negative classifications. However, the approach of displaying the number of packets is useful to analyze the development of an attack and compare events, if based on decent criteria. The operator alerting functions can be improved for different use cases by allowing for messages to be sent to a communication gateway. This would also allow alerts via E-Mail or SMS which are more useful when the system is not under constant supervision. Using a preexisting SIEM system to include the visualization into also allows it to be used for other purposes as well, in contrast to this approach, which developed their own specialized system.

Shiravi et al. [21] contribute a broad review of network security visualization systems. They start off describing the importance and advantages of visualization for network security, as well as the usability requirements. After presenting possible data sources for network security assessment and visualization, they present a list of visualization systems. The systems are chosen according to suitability for network security visualization, contribution of new techniques and are required to be checked for usability. In their opinion, a visualization system should be use-case driven and support in answering specific questions to its domain. Data sources should be utilized according to requirements. They arrange their findings according to a taxonomy of operational area and begin with the two parts of host and network monitoring, which give the most valuable input for us. First, IPs and also Port numbers can be aligned using two axes, which greatly improves informative value due to less aggregation, while saving space. Magnification or zooming can be utilized to drill down into lower levels for deeper analysis, while providing an aggregated complete system view.Also, logarithmic scales or sophisticated grouping can help to emphasize on important details, while saving space. This can be used for both node representation and data alignment. Colors should be used not only for graph styling, but also information encoding in a comprehensible way which means that they should be self-explanatory and sustained within the whole system.Filteringhelps to keep track of the important developments, while suppressing unimportant information. However, automated filtering must make sure that no necessary information is hidden from the network operator. This can also be a problem for learning algorithms, which possibly withhold a slowly developing attack. Correlation of internal to external address data could help to analyze data flows.Interactive charts allow for a deeper understanding of the data, while 3D graphs in most cases give no additional advantage on an information or usability basis over thoroughly designed 2D graphs. The ideas we extracted above could be usable for Amplification attack visualization as well. Later in the paper, they present more specialized visualization types, which are not of interest for us. One example is a visualization of DNS trafficwhich however focuses on the application layer too much to be generalizable. Summing up, they provide a requirements list, developed from detected issues and a lookout to networks development. They see the biggest problems in scalability, as system output should be able to present a high-level alert overview easily understandable by humans. They present a sorted table of their findings for future reference. In the end, they also mention privacy as an issue visualization has to deal with. However, they propose pseudonymization as a solution, which cannot be used in conformance with the German Data Protection Act due to reversibility.

3.4 Vulnerable protocols

In this part, we present a selection of protocols susceptible to Amplification attacks. As we described, the general solution to RDoS attacks would be the implementation of ingress filtering on every edge router of a network. However, the vulnerabilities can also be solved within the affected protocols. Therefore, we present a taxonomy of faults in the following, enriched by a selection of protocols as examples. Thus, taking advantage of our proposed visualization is possible due to better knowledge of the faults. This helps the network operator to reduce weaknesses.

3.4.1 General properties and categorization

Problems within the protocol design can allow Amplification attacks which are often difficult to fix without breaking important properties of the design. The fault can also be a problem within the actual implementation or its configuration, thus only affecting a sub-group of all services of that type, mostly identifiable by a system fingerprint [10]. Depending on the exact problem, the maximum reachable or typical Amplification Factor is limited which can also be used as a categorization.

We will classify the protocols we look at in the following according to these criteria, as this provides important facts for the visualization configuration and evaluation and required reactions to an attack:

3.4.2 Connectionless protocols

Connectionless protocols do not establish a bidirectional connection before sending actual data by design. Therefore, they are not capable of verifying the authenticity of the request which renders them vulnerable to IP spoofing and thus reflection attacks. The impact can be reduced by including state establishment into the application layer protocol. However, this is not always possible easily due to a broadly used specification.

The list of protocols vulnerable to Amplification attacks basically contains all UDP based protocols. In the following, we will present two examples which are subject to the flaws outlined in chapter 3.4.1.

The Domain Name System (DNS) is a service that translates addresses to addresses of another type, e.g. Domain Name to IP Address, and delivers additional information on request. The DNS servers are structured in a tree layout. Each server is responsible for a small subspace of the domain tree and thus stores information about that part. When receiving a query, it answers using information from the database or responds with an upstream DNS server which is a parent in the tree. However, DNS servers can also directly respond to queries not part of their subtree. In this case, they request the information from the upstream servers on their own and respond to the client with the final result. This feature is called recursion [22, 23]. The DNS server can also be configured to cache this information for later usage. If a server answers recursive queries from any host on the internet, it is said to do open recursion. Kambourakis et al. [17] show that DNS is susceptible to Amplification attacks due to several reasons: First of all, it has to be fast with low overhead, as most other services heavily depend on it. This is a design problem due to the fact that the behavior is defined in the DNS RFCs [22, 23]. Apart from this, DNS servers configured to do open recursion on behalf of a querying client allow for no additional data to be exchanged between the forged source address and the server between query and final answer. Combined with the DNS features which allow storage of large amounts of additional data within the database, this allows for relatively small queries for a large dataset to be sent from the spoofed source address of the victim to an open recursive DNS server, resulting in a very large answer in return. Amplification factors for DNS can be up to 40 due to the EDNS0 [24] extension which allows for DNS packets of up to 4KB size. As DNS Amplification is a relatively well researched topic, we used it within the evaluation setup 6 during this thesis. It also supports a high diversity of query and answer packets. For more details on DNS amplification, we refer to the ICANN which described an attack [16] and an advisory [1] to fix the issues by disabling open recursion.

Another UDP example is Network Time Protocol (NTP) which offers the current time to requesters. As shown by Kührer et al. [10], it is susceptible to Amplification attacks due to additional commands that are often available in the default configuration and unprotected from reflective DoS attacks in older versions. NTP allows for very high Amplification factors of up to 4670 and, in contrast to DNS, has a relatively small set of commands and thus different query packets. Due to this, it results in a low diversity within the answer packets, as packets contain identical content. We therefore use it within the evaluation setup 6 during this thesis to assess similarity as an indicator.

Two surveys [2, 3] and one extended investigation [25] of susceptible UDP-based protocols give a more complete reference, showing that many important and frequently-used protocols are vulnerable to Amplification attacks.

3.4.3 Faults of connection oriented protocols

Yet, there are other affected protocols beside UDP. An example is TCP, as outlined by Kührer et al. [10] which should not allow for a successful Amplification attack by its connection oriented design. TCP requires a 3-way handshake to succeed on connection establishment before exchanging data. The handshake starts with a SYN packet from the client initiating the connection to the server, containing a random sequence number $s\#_A$. The server then has to answer with a SYN/ACK packet containing the ACK number $s\#_A + 1$ and a random sequence number $s\#_B$. To establish the session, the client has to answer with an ACK packet containing sequence number $s\#_A + 1$ and ACK number $s\#_B + 1$. This protocol design should render Amplification attacks impossible, as the connection is teared down when an unexpected packet is received. However, there are implementations which reply to an incoming SYN request with up to 20 repetitions of the SYN/ACK packet, if these are not answered. This enables an attacker to successfully conduct an attack which has an Amplification factor of about 20.

3.5 Summary

In this chapter, we have presented two sets of approaches: First, we looked at detection approaches. Following that, we justified the selection of our chosen visualization toolset D3.js. As a last point, we outlined other attack and security visualization approaches, partly specific to Amplification attacks. We outlined the advantages and downsides of all systems and highlighted interesting ideas that could be useful for our approach.

In general, visualization and detection approaches are not really comparable. However, detection has specific downsides that can be overcome with visualization building on top of it. This includes human comprehensibleness, as emphasized by Shiravi et al. [21]. This enables the network operator to detect suspicious activity that would have been unrecognized otherwise. However, the evaluation of visualization also is a very time-consuming task. Therefore, the combination with detection is a win-win approach to utilize the advantages of both fields.

We also have to note that standard network visualization usually focuses on performance and link utilization in general and over time. Included statistics that rely on more than size and timestamp of the packets, if any, usually focus on the source and target networks and protocols or depend on predefined groupings. For attack visualization, only some of these graphs are useful, while others have to be extended to be meaningful. Relying on our research questions and the previous chapters about existing visualization methods, we will elaborate on the graphs we will implement.

As a general rule the graphs have to be detailed enough to allow for specific investigation of an attack, while still giving a good overview to easily recognize one. We will solve this by a set of graphs that gives a general overview, while having the option to view additional details using links to other data sources and also have special filters to show detailed information about similarities and affected systems and services.

Chapter 4

System architecture

In this chapter, we will describe the system architecture we developed during this thesis. Therefore, we will give a detailed description of the used software and developed interfaces, but do not talk about the actual implementation, which will be covered in the subsequent chapter.

We have to note that our selection of IDS and SIEM systems, as outlined in chapters 2.3 and 2.4, is only one possible choice of systems. Thus, the system design we outline below can be transferred to any other systems in the field, which support the requirements mentioned before. Of course, other choices therefore can replace the IDMEF or the outlined interfaces, depending on software support or willingness to implement. To ease replacements we will, however, clearly state which data is to be sent along the interfaces using a black-box approach.

Within the following chapters, we will use specific keywords for important terms, which we describe here:

event is used as an abbreviation to any Amplification attack, detected or undetected.

alert is defined as one single message from the Suricata IDS to Prelude SIEM. In our case, one alert only contains information belonging to one event.

set of alerts means all IDMEF messages belonging to one event, in our case one Amplification attack.

4.1 System overview description

During the thesis, we have used Suricata and Prelude to develop Amplification attack visualization based on D3.js, as outlined in chapter 2.

We had to develop and extend several interfaces between the software pieces to achieve that goal. First, the current Amplification attack detection implementation by Böttger [3] was not configurable, as it contained hard-coded values. It also did not fully make use of the IDMEF standard to alert Prelude, due to the fact that he evaluated the data from log files during the former work. Therefore, we designed an extension to the existing interface to fully transfer the knowledge about the detected Amplification attacks within Suricata to Prelude. Additionally, we defined additional Suricata rules to gain information about packets that could be related to Amplification attacks, but were not detected by previous approaches.

For Prelude, we developed extensions to gain additional knowledge about the IP addresses our alerts contain. As such, we defined location information and Autonomous System Numbers as interesting. In addition to that, we made hop count measurements and included ICMP alerts from Suricata.

On top of that, we developed the visualization, making use of all of the above mentioned datasets. Therefore, we developed an interface to extract data from the Prelude database in a compatible input format to the visualization framework D3.js.

The complete structure of the system is shown within the Evaluation in chapter 6.1 on 47.

In the following, we will describe our work in detail, starting from the lowest level and finishing with the visualization.

4.2 IDS: Suricata

4.2.1 Suricata-Prelude IDMEF interface

To send data from Suricata to Prelude, we use the "Intrusion Detection Message Exchange Format" (IDMEF), which is both a data exchange format and a definition of exchange procedures for IDS systems. The work on that project was started with the goal to create a common standard protocol suited for the exchange of datasets between IDS/SIEM products and thus make them interoperable. The development group of the IDMEF aimed at creating an IETF standard, but stopped working on it before reaching that goal. Their work finally led to the experimental RFC4765 and informational RFC4766, which describe the standard and an XML based implementation.

This protocol is supported by Suricata as a sender and Prelude as a receiver as well as the Prelude database, which is designed using the same structure to store the data. We will use this for the exchange of Amplification attack data from Suricata to Prelude. In the following, we first describe the IDMEF standard in general. As there already is a Suricata interface to Prelude, we will use this and build upon it to support additional values important for Amplification attacks, which we will describe. Additionally, we will introduce new values used only within the Prelude database to store additional data that we generate in conjunction with data received from Suricata.

4.2.1.1 IDMEF introduction

The aim of the Intrusion Detection Exchange Format Working Group (IDWG) was to create the above mentioned interoperable standard to enable interaction between different IDS/SIEM systems in order to improve information exchange.

The starting condition was a market of non-interoperable systems from different vendors, often aiming or focused at detection of special intrusions, or just used for a small subset of the systems within a network. Additionally, there are systems focusing on different points within a network, e.g. hypervisor hosts, routers, firewalls, servers and end-user systems. All of these can have different operating systems or protection requirements induced from their network environment or usage, e.g. monitoring a special application, service or runtime environment or the general system. This leads to a variety of Host and Network IDS systems. The advantage is having specialized systems for different use cases. The downside is that it is not possible to exchange information from these systems due to missing interfaces. Thus, a network operator cannot easily correlate, compare and distinguish different attacks, but must manually check for similarities on spec. This leads to a lot of additional work, but can also result in overlooking or misdiagnosing attacks or important details.

To overcome this, the IDWG proposes a standard for message exchange and procedures specialized for use within IDS/SIEM systems by the set of supported data types and its data structure. This enables the exchange of alerts about attacks between the supported systems.

4.2.1.2 Amplification attack extensions

In a former work [3], which we will build upon, the Suricata IDS, used as a network intrusion system sitting at the border router of the amplifier network, has been extended to detect Amplification attacks. However, this happened on a manual and out-of-band approach and thus did require additional steps for evaluation and did not inform the network administrator in real time nor in an easily comprehensible form. Due to this, we developed visualization for the Suricata data to ease understanding and evaluation. As a basis to extract data, we use the IDMEF standard, which is already supported by Suricata.

In its original implementation, Suricata sends every single packet belonging to a detected attack as a separate IDMEF message. In the former work, packet collection has been limited to 100 packets per Amplification attack (pairflow) within 10 minutes, as these give the necessary information about the attack and limit storage to an acceptable amount. This also limits the number of packets sent via IDMEF.

There was another thesis ongoing at the same time, which aims at including on-line detection of Amplification attacks to overcome the usability problems of the implementation by Böttger. Therefore, we worked together to pass all data required for the visualization to Prelude via IDMEF.

In addition to the sending of raw packets as described above, we define a set of advanced IDMEF messages based on it to transfer the important information about Amplification attacks. As the receiving system, we will use Prelude SIEM to collect and store the alerts generated by Suricata, as it fully supports the IDMEF standard and its database design is based on the standard as well. This makes the extension easily comprehensible due to the matched design. Additionally, this extension allows for realtime data transfer and evaluation at runtime because of the database design.

Amplification attack detection consists of three phases in the mentioned Suricata Implementation with the changes applied that are in current development:

1. **Detection of the attack**: Here, Suricata waits for the number of packets from a pairflow to reach a predefined packet and bytes threshold, as well as an amplification factor based on the ratio of incoming and outgoing bytes. If the amplification factor is reached, Suricata classifies the pairflow as suspicious and starts to capture packets in phase two. Otherwise it disregards the pairflow. We defined an IDMEF message which notifies Prelude about detection at the end of phase one.
2. During the **packet capturing phase**, Suricata captures a configurable amount of packets and stores them until the end of the phase. Then, packet similarity is determined, as it is an indicator for an Amplification attack. This is done by a compression routine, which takes the raw packet data as input and outputs the compression ratio, which is then used to calculate the similarity index. As similar data is highly compressible and random data is not compressible at all, a high compression ratio means a high similarity. As the end of the phase, Suricata sends an IDMEF message with the calculated similarity indices of packet length and packet content similarity.
3. During the **statistics phase**, which lasts as long as the remaining lifetime of the pairflow, statistics data is being generated and sent to Prelude in regular intervals and at the end of the pairflow. These contain all data from phase one. The length of a pairflow is also configurable.

We defined a set of four IDMEF messages, which contain the necessary data about Amplification attacks sent from Suricata to Prelude:

End of phase 1: Send an IDMEF alert message, using the values of the (pair)flow and Suricata. Apart from these, set

```
  CreateTime to the detection time
  Classification to "Amplification Attack threshold reached"
  Assessment to
    Impact to
      severity=info
      impacttype=dos
    Action to
      actioncat=notification-sent
    Confidence to
      rating=low
  AdditionalData to
    type=integer meaning=packet_count_in
    type=integer meaning=packet_count_out
    type=integer meaning=packet_size_in
    type=integer meaning=packet_size_out
    type=real    meaning=amplification_factor
```

During phase 2: Send an IDMEF alert message, using the values of the (pair)flow and Suricata. Apart from these, set

```
  CreateTime to the incoming time of the packet
  Classification to "Amplification Attack packet"
  Assessment to
    Impact to
      severity=info
      impacttype=dos
    Confidence to
      rating=low
  AdditionalData:
    send the packet header and content in the default layout of
Suricata (alert_prelude)
```

End of phase 2: Send an IDMEF alert message, using the values of the (pair)flow and Suricata. Apart from these, set

```
  CreateTime to the detection time
  Classification to "Amplification Attack packet similarity"
  Assessment to
    Impact to
      severity=info
      impacttype=dos
    Action to
      actioncat=notification-sent
    Confidence to
      rating=low
  AdditionalData to
    type=integer meaning=packet_count_in
    type=integer meaning=packet_count_out
    type=integer meaning=packet_size_in
    type=integer meaning=packet_size_out
    type=real    meaning=amplification_factor
    type=real    meaning=similarity_size
    type=real    meaning=similarity_content
```

End of phase 3: Send an IDMEF alert message, using the values of the (pair)flow and Suricata. Apart from these, set

```
  CreateTime to the detection time
  Classification to "Amplification Attack statistics"
  Assessment to
    Impact to
      severity=info
      impacttype=dos
    Action to
      actioncat=notification-sent
    Confidence to
      rating=(set to low, medium, high depending on similarity)
  AdditionalData to
    type=integer meaning=packet_count_in
    type=integer meaning=packet_count_out
    type=integer meaning=packet_size_in
    type=integer meaning=packet_size_out
    type=real    meaning=amplification_factor
```

4.2.2 Configuration flexibility

There were hard-coded values in the Amplification attack detection code developed by Böttger, [3], namely a count of exactly 100 packets to be analyzed to classify an Amplification attack, as well as a lifetime for a detection cycle (pairflow) of ten minutes. The number also influences the number of packets sent to Prelude. In order to allow the visualization to be as flexible as possible, we wanted to eliminate such restrictions. Consequently, we designed the interface from Suricata to Prelude to allow for arbitrary values here. For both of the values, we had two choices where to define them:

- Amplification attack detection rule: this way, we could define separate values for each detection rule, allowing for different values e.g. for different IP address ranges or amplification factors. However, the implementation effort could be high, as currently the values are hard-coded in a part of Suricata that is not called on a per-rule basis.

- Suricata main configuration: Here we can define values, which are valid system wide and therefore affect every detection rule. It should also be easy to read the value from that file on start of Suricata due to its included configuration file parser.

As the actual implementation of the Suricata extensions was carried out during another thesis at the same time, we added support for both options to the definitions in the previous chapter by the naming scheme.

4.2.3 Advanced ruleset

The Amplification attack detection ruleset, as developed by Böttger, [3], was extended during this work due to two requirements:

To allow *detection of false negatives*, we added an additional rule with lower thresholds, which helps us by also informing us about events that may represent an attack. To distinguish these from alerts using the default amplification factor of 5, we send a lower severity for these. Thus, that can be used as a filter criterion within Prelude.

The significance of ICMP unreachable errors, which inform the sender of a packet that it could not be delivered, was also evaluated during former work [3, 4]. However, it could not be concluded that those occur for every Amplification attack, and thus were left out as a criterion in the subsequent article [4]. As they still give additional information to evaluate an attack if they appear, we decided to include them into Prelude and the visualization for evaluation. Therefore, we included a detection rule for ICMP unreachable errors into our ruleset to generate corresponding alerts to be stored within the Prelude database.

4.3 SIEM: Prelude

The Prelude SIEM system has some drawbacks, which we have to overcome: although the database scheme allows storing almost any interesting attack information due to its flexible design, per default it does not store any address information apart from IP addresses, port numbers and protocols.

4.3.1 Location and domain names

In addition to the aforementioned extensions, there is additional data about physical and logical network topology, which can be used to learn about similarities of attacks:

- Domain Names can inform about similarities or connections among the attacked systems also if they are part of different networks and thus have different IP addresses. A possible application could be highly redundant or distributed systems, e.g. Content Delivery Network (CDN) servers.
- Autonomous System [26] (AS) numbers can give additional information, if attacked systems are not within the same IP subnet and do not share parts of the Domain Name, but still belong to the same organization and thus reside within the same AS.
- Location information in the form of GPS coordinates or city and country names can give additional information about attacks that focus on the same physical target area. As the IPv4 address space is very scattered by now [27], it does not necessarily allow for inference about location similarities. Thus, location data can be useful to overcome this. Human readable data like country and city names can be used within the detailed view of Prelude to give additional information or as a search or grouping criterion, while GPS coordinates can be used to generate a map displaying the affected systems.

We will store these datasets within Prelude for the following reasons:

- **Performance** is greatly increased by local, offline storage of the datasets, in contrary to requesting them on demand.
- **Sorting and grouping** is done within the Prelude database. Therefore, the database has to contain the necessary datasets. With data fetched on demand, a two-step process would be necessary, which is not desirable due to performance and system complexity.
- **Availability and correctness** can only be ensured with offline data, as IP address assignment is not static, but subject to change on disposal. Therefore, additional data could not represent the state at the time of packet capture anymore, if requested on demand.

We could also keep track of all these pieces of information within the amplifier network. However, the network documentation is more precise and correct in most cases. Thus, this approach is not tracked further.

In the following chapter, we will propose a scheme extension to the Prelude database to store this information.

4.3.2 Path length

While traveling through the network, each packet passes a number of nodes, called hops, on its path from source to target host. The length of the path is defined as the number of nodes the packet passes, for example routers, therefore called hop count.

Each IP packet contains a value[1], representing how many hops it can travel before it gets discarded. Therefore, each hop on the path from source to target decrements this number. If we know the initial TTL value, we can tell how far the packet traveled. As most systems start with a default value for each packet sent out, we can guess the hop count[2]. Also, we can measure the distance to any IP using *traceroute*. Comparison of the outbound measure to the TTL values of incoming packets can give additional information about anomalies. Therefore, it can be expected that for an Amplification attack, in most cases the guessed hop count of the incoming packet does not match the outgoing measurement value. However, due to technologies like Network Address Translation, asymmetric routes and packet filtering, we may not always get the correct information, if any. Therefore, Böttger et al. [4] left out this rating. However, using additional information about autonomous systems, hop count and location-specific data, this information could be valuable for manual evaluation.

The captured incoming Amplification attack packets contain the TTL or hop count on receipt. Therefore, this information is stored within the Prelude database already. Thus, we need to determine the corresponding hop count from our detection host to the victim by a traceroute measurement and store that information in the database as well. Eventually, we can determine the delta for further investigation.

4.3.3 Anonymization concept

As Germany's Data Protection Act (Datenschutzgesetz) [28] does not allow storing of personalized data without consent of the affected individual. Personal data also contains IP addresses and domain names, as they can be used to identify a person. However, storing is allowed for inevitable reasons for the necessary timeframes. For a system like ours, more than one week cannot be justified. Therefore, that data has to be anonymized in the system to eliminate this possibility after this timeframe. This can be achieved by deleting the last bits of the IP address or the lower-level part of the second level domain name of the stored data. After this, it is not possible anymore to reliably fetch the information mentioned in the last paragraphs using the anonymized data. Because of this, we have store the additional data within the Prelude Database before we anonymize the addresses. Apart from this, we gain an increasing system

[1]The value is called TTL for IPv4 and hop count for IPv6.

[2]The implementation of Böttger [3] states an usual hop count of 20 from source to target, while most operating systems start with initial values of 64, 128 or 255.

performance using this approach, as all relevant data does not have to be queried for on-the-fly when needed, but instead is stored locally. The concept we developed for anonymization will be outlined in chapter 5.2.2.

4.3.4 Search history

Over time, a lot of datasets can be stored within a SIEM system like Prelude. Grouping and filtering the data is one choice to find the necessary information. However, it could be necessary to re-interpret earlier checked datasets of attacks later again to apply new knowledge or compare them to other events. Thus, storing links to datasets which were opened earlier eases future reference. As Prelude does not have such a feature, we incorporate it, allowing re-opening exactly the same view. To facilitate separation, a timestamp of initial opening is stored alongside.

4.4 Visualization

In the first parts of this chapter, we described how to collect the attack data and store it for future usage. Within this chapter, we will describe the visualization we created on basis of our data definition and the interface to the Prelude database. Eventually, the proposals within this chapter will be fundamental to answer the scientific questions. We will outline this in the next chapter.

4.4.1 Interface definition

The data transfer interface, called *"Prelude to Visualization"* (**P2VIS**) in the following, is responsible for data exchange from Prelude to the visualization toolset D3.js. As prerequisites, we have a MySQL database used by Prelude on the server side, which contains the attack data. On the client side, we have JavaScript-based visualization, which can take all kinds of formatted text files as input. However, due to the flexible database layout of Prelude which allows us to store all kinds of datatypes, we cannot easily query for the relevant datasets and pass them to D3.js. In chapter 4.2.1.2, we describe the datasets sent to Prelude via IDMEF and therefore also stored in the database, as it uses an identical layout. Now, we will describe which data we need for the visualization, as we have to aggregate it to be able to pass it to the visualization.

As described in chapter 4.2.1.2, the database stores an amount of full packets belonging to the Amplification attack. Additionally, it contains a fixed set of messages containing statistical information for each event. For the visualization, however, we only need parts of this information for each event, combined from the alerts stored in the database. As the server side contains a database for storage of the event data, while the client side does not, we want to utilize the database performance to process the stored data. This allows us to gain better system performance and minimize the amount of data sent from the server to the client.

To create the visualization graphs we define in the next chapter, we need the following pieces of information for each event, which we can combine from the stored set of alerts:

ID Database ID of the IDMEF alert named "Amplification attack packet similarity". This is the first entry which is unique and contains all information about the attack. Only the information about the total size of the attack is missing, as this is contained in the final statistics message at the end of the pairflow. This data is then joined via SQL queries. The other way round, this value combined with the timestamp allows to link back to the attack within Prelude.

timestamp Timestamp of the classification of the attack, containing date formatted as "YYYY-MM-DD HH:MM:SS" to incorporate it into time series graphs or filter the event data by time.

src_ip contains the IP address of the amplifier within our network. As the original source cannot be determined due to IP address spoofing, this information is not available to us.

dst_ip contains the IP address of the victim, which receives the amplified answers.

src_port contains the port number of the amplifier from where the amplified response packets are originating.

dst_port contains the port number of the victim which receives the answers.

L3_proto contains the ISO/OSI Layer 3 protocol version, as sent within the IP header. For IPv4, this corresponds to 4, while it corresponds to 6 for IPv6. [3]

L4_proto contains the ISO/OSI Layer 4 protocol, as sent within the IP header. For TCP, this corresponds to 6, while it corresponds to 17 for UDP. The full set of numbers is defined by the IANA [4] and the assignment is defined in BCP 37.

AS_num contains the autonomous system, which the destination IP is part of, as determined by querying an external database of whois records.

as_name lists the owner of the AS number which contains the target IP address, usually an internet provider or large network, for fast investigation.

[3]The Prelude SIEM stores ipv?-addr within its database. However, we use the original protocol numbers here to create an interoperable system. Within the next chapter, we convert the values to human readable terms.

[4]http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

loc_lon and loc_lat contain the values required to position a target host on a geographical map based on the location information outlined below.

loc_country is set to the unique country code which the target IP address is inside of according to whois records.

loc_city and loc_region (optional) contain the city and the region (e.g. US State).

dom_name is set to the fully qualified domain name of the target, as determined by a reverse DNS query using the IP address.

amp_factor contains the calculated amplification factor by the IDS system.

sim_size is set to the similarity in packet length as calculated by the IDS system. The type is a decimal number in the range [0,1], where 0 means totally different and 1 means identical.

sim_content is defined like above, but we take the similarity of the packet content this time which is determined by compressing the data and comparing compressed length to original length.

packet_count_in and packet_size_in contain the number or total size of packets incoming from the spoofed source address to the amplifier. As the total packet size is supposed to grow with every statistical alert, we take the most recent one.

packet_count_out and packet_size_out contain the number or size of packets being sent to the victim.

icmp_count is set to the number of ICMP unreachable packets received from the victim during the attack. Therefore, the timestamp value is used to correlate the alerts.

ttl contains the value from the TTL or "hop count" field of the most recent incoming packet from the IP header.

hop_count is set to the number of hops determined by a traceroute call to the victim IP.

We have to implement an interface which takes the alert data as input and generates an output file including this information. The naming scheme within the interface uses the values proposed here in **bold** font.

4.4.2 Graph description

Using the data we outlined above, we build the following set of graphs. For your reference, we included figure 4.1 to display the main graph types. They are sorted by the suggested order of viewing here.

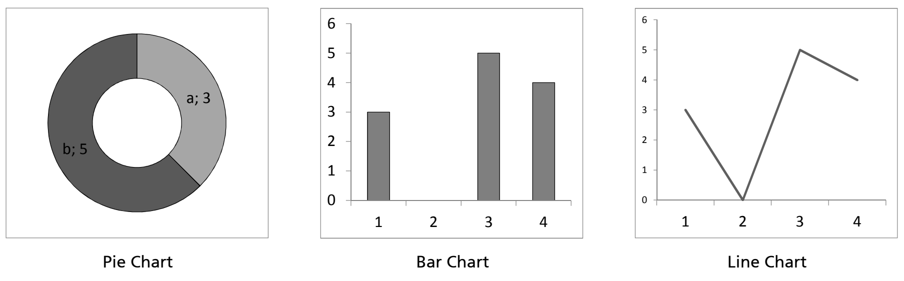


Figure 4.1: Graph types

Events per day shows a time-series line chart where the x axis represents the visualized timeframe, while the y axis illustrates the number of events for each timestamp. Depending on the length of the timeframe, grouping has to be applied to make the graph readable, as we must be able to determinate a unique value for each timestamp. The exact scaling therefore depends on the graph type, the timeframe and the expected values. Because we use a time-series chart which displays a curve representing the corresponding events counter for each timestamp, we need at least two pixels width for every timestamp displayed to allow rising and falling within the graph.

Similarity is displayed by a set of two pie charts for packet length and content similarity. Here, we use the classification suggested by Böttger et al. [4] and map a similarity of smaller than 0.75 to low, while we map the other values to high.

Events by Amplification Factor is a bar graph, listing the number of events for each amplification factor, which is rounded to integer values for each event.

Number of Events by packet count (in,out) is represented by a bar graph which displays a number of events per packet count represents the distribution of attack size in packets. In order to differentiate very small from very large attacks, logarithmic scaling of the x axis which represents the packet count can be useful. The graphs must be separated for incoming and outgoing packet counts, due to the fact that some Amplification attacks work by repeated packets, as outlined in chapter 3.4.3.

Number of Events by packet size (in,out) is a bar graph like above. However, it depends on the overall packet size of an event. This graph is necessary to be able to determine the size of an attack. Both sets of the packet graphs use the identical scales on the x axis and are arranged in one column to be comparable.

ICMP packets count is equal to one of the amplification factor graph and displays the number of ICMP packets received that could be mapped to an event.

TTL+Hop Count is again equal to the previous one. However, only the delta of the outbound measurement and the value read from the incoming packet is displayed here.

IP Layer Protocol refers to the layer 3 protocol used for transmission of the packet. We create a pie chart from the data to represent the distribution of protocols via the slices. Here, we translate the IP version 4 and 6 of the input to readable expressions IPv4 and IPv6 in the graph. If other values are read on the input, they are grouped into an others slice.

Network Layer Protocol depicts the layer 4 protocol and is split into TCP, UDP and others, while the remainder is configured equally to the last described graph.

Destination/Source IP is a set of three pie charts that allow selection and viewing of distribution of events among the first three octets of an IPv4 address.

Destination/Source Port is depicted as one pie chart, where each slice represents a port number.

The graphs presented above provide the important information to detect and classify an Amplification attack and determine amplifier and target. The additional values we added to the P2VIS interface, however, can help us to determine correlations of the events. We implemented several new graphs as follows:

AS Number

Top-level Domain

Country Code

Beside of viewing various data items at once, the network operator also has to be able to narrow down the selection according to relevant criteria. We have two possibilities to achieve that: Within Prelude SIEM we can filter the data according to all criteria, as the Prewikka web interface allows to select fields and values of all types within its database using its integrated filtering engine. Based on these facts, it then only shows the relevant database entries to the user. Finally, we could pass this list to the visualization as input. The other possibility is to export all data via the P2VIS interface to the visualization and build the visualization such that filtering is possible there. The first proposal has the advantage of being able to reuse the filtering engine, while it has the downside of having to reload the visualization every time a newly applied filter changes the data. The second proposal necessitates re-implementing filtering on the client. This offers an improved responsiveness for the client. This means that graphs can be changed on the fly without reloading the page using client-side filtering. Thus, the network operator can directly see changes within the graphs while applying filters, which promises faster exploration of the data. The downside is that all data has to be loaded into the visualization at once. However, this only happens once on opening, while the other approach would send a subset of the data on every change of filters.

We therefore decided for the client-side approach. To reduce the amount of data we have to load and thus increase system performance, we added the possibility to select a timeframe to visualize. This should be the first restriction the network operator sets, as outlined in Q1 in chapter 1.1. We therefore add one more element to the visualization:

Table At the bottom of the graph page, a table is included. It contains the set of selected records by the currently applied filter. The list is sorted by the total size of the attack, with the largest attack on top. It includes the following values:

Alert timestamp

IP Addresses of source and target

Domain Name of target

Similarity indices for content and length

Amplification factor

Total packet count and size

Position mapped to the IP address as city, region and country.

The table contains links to Prewikka via the alert ID and to various other information sources: By clicking on the IP address, a view is opened containing the correlated address data we specified. The domain name link points at a whois service, which can be used to gain information about the owner of the device from the records. The last column, position, opens a map with a marker at the position recorded for the host.

Chapter 5

Implementation

Within the previous chapter we described the system architecture of the visualization system, focusing on a black-box view and giving additional details where necessary. In the following, we will describe the actual implementation. We noted that some parts are implemented as part of another thesis. It was still a work in progress when we submitted this work. Therefore, this chapter will not contain the data exchange interface using the IDMEF from Suricata IDS to Prelude SIEM.

5.1 Suricata IDS advanced ruleset

Here, we describe the configuration changes within Suricata for the described advanced ruleset.

To send alerts from Prelude to Suricata, Suricata has to be built with Prelude support included. Additionally, in the main Suricata configuration alerting has to be enabled and configured to include full packet data for later evaluation:

```
  # alert output to prelude (http://www.prelude-technologies.com/) only
  # available if Suricata has been compiled with --enable-prelude
  - alert-prelude:
      enabled: yes
      profile: suricata
      log-packet-content: yes
      log-packet-header: yes
```

```
# load rule file containing the rules specified below
rule-files:
 - ampl-detect.rules
```

The Suricata detection rule proposed by Böttger [3] only allows capturing of Amplification attacks within Suricata according to fixed thresholds. In chapter 4.2.2, we already proposed an extension which allows for a flexible change of these variables. Additionally, we add another Amplification attack detection rule with lower thresholds to also store these attacks within Prelude DB. As Böttger et al. [4] stated that an attacker has several choices to evade detection by staying under the proposed thresholds of total traffic or amplification factor.

```
alert ip any any -> any any (msg:"Amplification Attack"; amplification:
      byte_ratio 5, byte_threshold 10000000, packet_threshold 25;
    sid:1; rev:1; priority:1;)
alert ip any any -> any any (msg:"Amplification Attack"; amplification:
      byte_ratio 3, byte_threshold 10000000, packet_threshold 25;
    sid:2; rev:1; priority:2;)
```

This additional rule allows the network operator to evaluate those events separately to identify false negatives, which was a goal of *Q5*. Therefore, the network operator is encouraged to add a rule with lower thresholds and evaluate the additional events. In order to avoid being overwhelmed by data, the values of this additional rule can be slowly reduced, starting from the proposed values of the default rule proposed by Böttger [3]. Also, the proposed visualization allows filtering for values of total size and amplification factors, which facilitates evaluation. It is most promising to reduce the amplification factor, as there are ways to undermine this quite easily by sending additional garbage traffic to the amplifier.

Additionally, we include alerts about ICMP destination unreachable packets [29] with another rule:

```
alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"ICMP unreachable";
    icode:3; itype:3; classtype:misc-activity; sid:1; rev:1;)
```

The proposed set of rules allows us to collect all important data from Suricata within Prelude.

5.2 Prelude SIEM

5.2.1 Data storage

In addition to the data outlined above, we have proposed additional useful information in chapter 4.3, which cannot be transferred from Suricata as the data does not exist in there. Therefore we will generate the data directly and pass it to the Prelude database. There are multiple sources[1] of information available on the internet and also for local

[1] http://ipinfo.io/, http://www.infobyip.com/, https://www.whatismyip.com/ip-address-lookup/

downloading that contain all information defined in 4.3.1. As we require current data at the time of request, we decided to query the online sources instead of mirroring a full copy to reduce download overhead while maintaining data freshness. However, we have to generate the traceroute data defined in 4.3.2 ourselves, as it is dependent on the network structure. Then, both sources of data can be included into the databases in the same way.

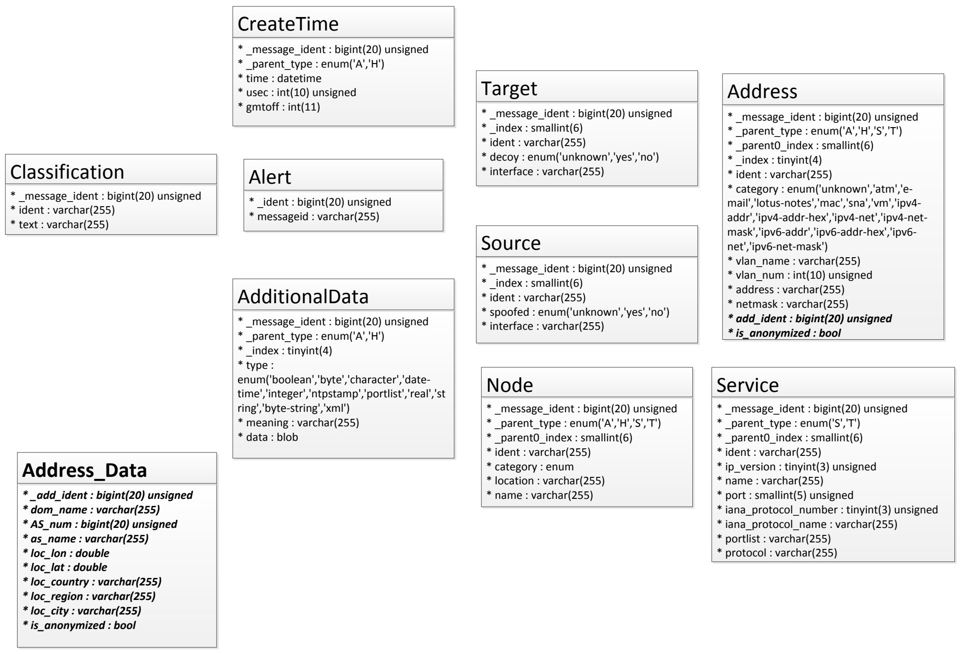
We had two possibilities to extend Prelude: first, we could request and add the information on the fly within Prelude when it receives the alert, which has the downside of slowing down the database insertion process. This could be a problem for a heavily loaded system. Alternatively, we could add the information later, which does not slow down the alert insertion process. If this is done in regular and short intervals, we can also expect to receive current data and allow fast analysis of the data using the visualization, which incorporates these additional datatypes. Therefore, we close the second possibilitiy and implemented a Cronjob which selected all IPs from the database which currently did not have the additional data stored using a MySQL query. For these IPs, we now request the additional information from a web service or generate it locally, depending on the type. Then, we dissect the data into the separate fields to be able to insert them into the database. The Cronjob runs in intervals of 10 minutes, which is the default pairflow lifetime, to have full information within the database at the latest at the end of the pairflow.

Per default, Prelude does store such values within its database. To support storage and also anonymization which is introduced in chapters 4.3.3, 4.3.1 and 4.3.2 while reducing overhead to a minimum, we extended the database scheme. In the following, we will **highlight** our changes in bold font style and include the necessary default Prelude tables for understanding.

In the added table Address_Data, we store the aforementioned additional datatypes. This table is searchable due to the ID field _add_ident is also available in the Address table, which Prelude uses to store address information. You may notice the table AdditionalData, which is able to store almost any datatype. However, searching and linking the requested value always makes use of the "meaning" field, which would slow down performance. Thus, we decided to add another table for our purposes. As the table is only evaluated by P2VIS and not by Prewikka, the table will not interfere with the rest of its functionality.

5.2.2 Anonymization

The anonymization uses the extended database structure we proposed in figure 5.1. There, we added one table Address_Data that depends on the Address table. Within these two tables, we find the fields `address` and `dom_name` that have to be anonymized as described in chapter 4.3.3. Therefore, we use a Cronjob which daily connects to the



```
CreateTime
* _message_ident : bigint(20) unsigned
* _parent_type : enum('A','H')
* time : datetime
* usec : int(10) unsigned
* gmtoff : int(11)

Target
* _message_ident : bigint(20) unsigned
* _index : smallint(6)
* ident : varchar(255)
* decoy : enum('unknown','yes','no')
* interface : varchar(255)

Address
* _message_ident : bigint(20) unsigned
* _parent_type : enum('A','H','S','T')
* _parent0_index : smallint(6)
* _index : tinyint(4)
* ident : varchar(255)
* category : enum('unknown','atm','e-
mail','lotus-notes','mac','sna','vm','ipv4-
addr','ipv4-addr-hex','ipv4-net','ipv4-net-
mask','ipv6-addr','ipv6-addr-hex','ipv6-
net','ipv6-net-mask')
* vlan_name : varchar(255)
* vlan_num : int(10) unsigned
* address : varchar(255)
* netmask : varchar(255)
* add_ident : bigint(20) unsigned
* is_anonymized : bool

Classification
* _message_ident : bigint(20) unsigned
* ident : varchar(255)
* text : varchar(255)

Alert
* _ident : bigint(20) unsigned
* messageid : varchar(255)

Source
* _message_ident : bigint(20) unsigned
* _index : smallint(6)
* ident : varchar(255)
* spoofed : enum('unknown','yes','no')
* interface : varchar(255)

AdditionalData
* _message_ident : bigint(20) unsigned
* _parent_type : enum('A','H')
* _index : tinyint(4)
* type :
enum('boolean','byte','character','date-
time','integer','ntpstamp','portlist','real','st
ring','byte-string','xml')
* meaning : varchar(255)
* data : blob

Node
* _message_ident : bigint(20) unsigned
* _parent_type : enum('A','H','S','T')
* _parent0_index : smallint(6)
* ident : varchar(255)
* category : enum
* location : varchar(255)
* name : varchar(255)

Service
* _message_ident : bigint(20) unsigned
* _parent_type : enum('S','T')
* _parent0_index : smallint(6)
* ident : varchar(255)
* ip_version : tinyint(3) unsigned
* name : varchar(255)
* port : smallint(5) unsigned
* iana_protocol_number : tinyint(3) unsigned
* iana_protocol_name : varchar(255)
* portlist : varchar(255)
* protocol : varchar(255)

Address_Data
* _add_ident : bigint(20) unsigned
* dom_name : varchar(255)
* AS_num : bigint(20) unsigned
* as_name : varchar(255)
* loc_lon : double
* loc_lat : double
* loc_country : varchar(255)
* loc_region : varchar(255)
* loc_city : varchar(255)
* is_anonymized : bool
```

Figure 5.1: Database Scheme (The highlighted values were added)

database and searches for alerts with an age of more than 6 days and the is_anonymized flags not set. This is possible by joining the tables CreateTime, Address and Address_Data. For those, it toggles the flags and anonymizes the datasets, as described in chapter 4.3.3. Thus, we can still retrieve all information we stored before, but do not store personalized data anymore. This facilitates long-term evaluation.

5.2.3 Prewikka filters

Within the last chapters, we added data to the Prelude database. Now, we will focus on how to filter the data to be able to extract the relevant set. Per default, Prewikka groups the data according to the source and target addresses and the alert classification, in our case Amplification attack or ICMP message alerts. The address grouping automatically puts all alerts for one combination of addresses into one group. However, we must notice that Prelude does not know the concept of pairflows, which means that it aggregates multiple alerts. The downside of this is that we cannot differentiate pairflows, while the advantage is that we can easily recognize an accumulation of attacks by the counts of alerts which Prewikka presents in front of the classification, as shown in figure 6.3 on page50 within the evaluation. We therefore present a set of filters that can be configured inside Prewikka:

Single messages for each event To just see one single message for each event, Prewikka can be filtered by the string "Amplification attack packet similarity" within the "text" field in the classification column. Within this alert type, we see every piece of information about Amplification attacks except for the total statistics.

Combined view of Amplification attacks and ICMP alerts To filter for all amplification alerts, the filter from above has to be extended by the other classifications listed in 4.2.1.2.

time filters for selection of graph data in the left bottom corner of the Prewikka interface there is a filter for time values. There, the output can be limited to a number of elements or a timeframe. For the timeframe, the end date and the duration has to be entered. These values then define the starting point.

Using these filters, the database can be evaluated easily and the data representation focuses on Amplification attacks.

5.2.4 Search history

For the search history, the problem is split into the parts alert view of Prewikka and visualization view. The implementation of the search history within Prewikka was done with a background script recording the requests sent to the server via HTTP GET and storing them in a link list next to the timestamp of opening. Using this link list, the network operator can re-open the saved view later. Therefore, the original HTTP GET request is re-sent to Prewikka by the user clicking on one of the saved links.

For the Visualization, we were not able to come up with a solution. Therefore, the only current option is to save it on the client side by creating a printout. We tried to implement saving on the server-side using the screenshot feature of html2canvas. Although basically working, it was not usable for our visualization, as the graphs were stored without values and therefore empty. As a future goal, we therefore propose to solve this problem using the following approach: It would be most useful to store the actual filters set by Crossfilter to the server together with the timeframe of the selected data. Thus, on reopening the graph, the filters could be loaded from there and re-applied. In addition to that, it would ease comparison of different timeframes if a filter could be re-used within another data selection.

5.3 Prelude to Visualization interface

Within the previous parts of this chapter, we extended Prelude to include all data we require for visualization. Here, we present our interface P2VIS that reads the required

timeline_unit	timeline_unit_factor
min	60
hour	3600
day	86400
month	2592000
year	31536000

Table 5.1: Time Conversion

datasets from the database and processes them to finally output the aggregated data as described in chapter 4.4.1.

As we outlined in chapter 4.3.3, the Prelude database is split up into a unique table for every kind of information, like source and destination IP addresses, ports, timestamps, classification and many more. Please refer to figure 5.1 on 39 for a detailed view of the tables that are important for us. This means that we have to join several tables to achieve the desired output. To achieve this, we designed a MySQL query that produces the output specified in chapter 4.4.1. As an output format, we use a table-like format, as database queries generate a table as output and thus conversion to formats like XML or JSON[2] would be an additional step. Thus, we decided to use CSV[3] files, as those are easy to generate from database output and supported by our chosen visualization toolset D3.js. Shortly described, P2VIS, which is implemented as a Python script, carries out the following steps:

1. Send a MySQL query to the database, joining the relevant tables using the alert ID as an identifier
2. Store the query result in a Dictionary
3. Output the result to the visualization, formatted as CSV

P2VIS takes three variables to select the timeframe as input, as specified in the end of chapter 4.4.1:

timeline_end is the time of the most recent alerts still to be included, formatted as Unix timestamp.

timeline_unit specifies the unit for the next field and is one of the values noted in table 5.1.

timeline_value specifies the length of the timeframe to be selected, starting from timeline_end and extending into the past.

[2]JavaScript Object Notation
[3]comma-separated value

Therefore, the timeframe is selected as the interval[4]

$$[timeline\_end - (timeline\_unit\_factor * timeline\_value); timeline\_end] \qquad (5.1)$$

if timeline_unit is not unlimited. Otherwise, no timeframe is applied, which means that all alerts are fetched from the database.

The P2VIS script is then called from the visualization web site, which passes the time values to it and uses the output as CSV input for graph generation.

5.4 Visualization

The visualization takes the data described in chapter 4.4.1 as input and creates the graphs presented in chapter 4.4. There, we also argumented to re-implement filtering on the client due to an expected gain in usability. Thus, we had the options of implementing manual entry filter fields like in Prewikka or interactive graphs, which enable the user to filter on specific criteria by clicking on the graph. Due to usability, we decided for the second option. We therefore reuse the JavaScript libraries *Crossfilter.js* [30] and *dc.js* (Dimensional Charting) [31], which offer the required functions. Crossfilter enables us to build graphs with coordinated views, which means that filtering on a criterion in one graph immediately adapts the other graphs part of the same dashboard to the filter. The authors claim that it can easily filter datasets in the range of millions. Thus, we expect very good performance. For the actual graphical representation of the data, Crossfilter depends on a visualization toolset to pass the data to. Here, we use DC.js, which interfaces Crossfilter to get visualization data. It then passes to D3.js for the actual representation, as outlined in figure 5.2. Therefore, DC.js includes a set of standard graphs including the types we show in figure 4.1 on page 33. At the time of writing, the current versions were Crossfilter.js 1.3.11 dated Oct 3, 2014 and dc.js 2.0.0-beta.18 dated Aug 25, 2015.

For reference, we present the graphing environment and one graph that displays the distribution of amplification factors here. The first step is to create the surrounding JavaScript environment using D3.js and Crossfilter, which only has to exist once and can be extended by additional graphs:

```
// load input data from P2VIS interface
d3.csv("p2vis.py?timeline_end=&timeline_unit=&timeline_value=",
  function (data) {
  // input data formatting
  var dtgFormat  = d3.time.format("%Y-%m-%d %H:%M:%S");
```

[4]timeline_unit_factor specifies the number that corresponds to one unit of timeline_unit in seconds and can be read from table 5.1

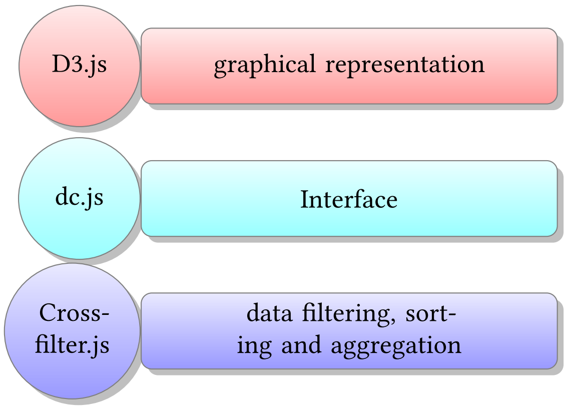


Figure 5.2: Visualization: JavaScript Libraries

```
  var dtgFormat2 = d3.time.format("%a %e %b %H:%M");

  data.forEach(function(d) {
    d.dtg1   = d.timestamp.substr(0,10) + " " + d.timestamp.substr(11,8);
    d.dtg    = dtgFormat.parse(d.timestamp.substr(0,19));

    // Define additional input data here

  });

  // load the data into crossfilter
  var facts = crossfilter(data);
  var all = facts.groupAll();

  // define the graph data values and groupings here

  // count all the facts
  dc.dataCount(".dc-data-count")
    .dimension(facts)
    .group(all);

  // define the graphs here

  // Render the Charts
  dc.renderAll();


});
```

Next, we create an HTML anchor for the graph in the web page the graph should be included. This webpage must also contain the JavaScript code outlined here and load the outlined libraries.

```
<div id="dc-factor-chart">
  <h4> Events by Amplification Factor
    <span>
      <a class="reset"
  href="javascript:factorChart.filterAll();dc.redrawAll();"
style="display: none;"> reset
</a>
    </span>
  </h4>
</div>
```

We then define the graph and map it to the HTML anchor created above.

```
var factorChart = dc.barChart("#dc-factor-chart");
```

Next, we add graph data to the data function defined in the general template:

```
data.forEach(function(d) {
  d.factor = d3.round(+d.amp_factor,0);
// in this case, round input data to integer values
});
```

Finally, we create the graph inside the D3 environment:

```
// input data for factor graph
var factorValue = facts.dimension(function (d) {
  // additional conversion or combination of input data is done here
// for the more advanced graphs
return d.factor;
});
var factorValueGroup = factorValue.group()
  .reduceCount(function(d) { return d.factor; });

// Factor bar graph
factorChart
  // define graph size
.width(480)
  .height(150)
  .margins({top: 10, right: 10, bottom: 20, left: 40})
  // define graph data
.dimension(factorValue)
```

```
  .group(factorValueGroup)
  // graphical formatting
.transitionDuration(500)
  .centerBar(true)
  .gap(1)
  .x(d3.scale.linear()
  .domain(
  d3.extent(data, function(d) { return d.factor; }))
  )
  .elasticY(true)
  .xAxis().tickFormat();
```

We will skip the other graph types here and point to the official documentation. Above, we added comments where the code has to be changed to achieve the desirable outcome.

In the next chapter, we will describe and evaluate the set of described graphs from chapter 4.4 in detail.

Chapter 6

Evaluation

In this chapter, we will evaluate our system design against the scientific questions defined in chapter 1.1. Our black-box approach used in chapter 4 for the system architecture facilitates this as it clearly specifies the outcome.

We mentioned before that we only specified the extension of the interface from Suricata to Prelude, while the implementation was part of another thesis aiming at real-time detection within Suricata. As of writing this, that part was not yet done. Therefore, we cannot use this interface for evaluation. Thus, we are limited to verifying our goal by comparing the definition from chapter 4.2.1.2 to the necessary and existing information about amplification attacks, as described in chapter 3.1.2. For the P2VIS interface and the Visualization, we have to generate data manually due to that. Within this chapter, we will describe where we did that exactly and what we did.

The evaluation therefore is split into three parts: in the first part, we evaluate all data input interfaces and definitions to check data coverage. Following that, we focus on the P2VIS interface, which we check separately due to its importance for the visualization. As a last point, we determine usability of the visualization and propose further additions. We added figure 6.1 to give an overview over all interfaces and flow of data.

For all parts, we will check our specification for feature coverage against the corresponding scientific questions introduced in chapter 1.1.

6.1 Test system

During this thesis, we used a development system consisting of six PCs. Each PC was run by an 8-core CPU and 16GB of RAM. The PCs contained a 4-port gigabit ethernet card and were running a modified version of Grml Linux [32] to suit networkers' needs. Using these PCs, we created a test setup to conduct amplification attacks that consisted of an external network to simulate the internet and an internal network, called amplifier

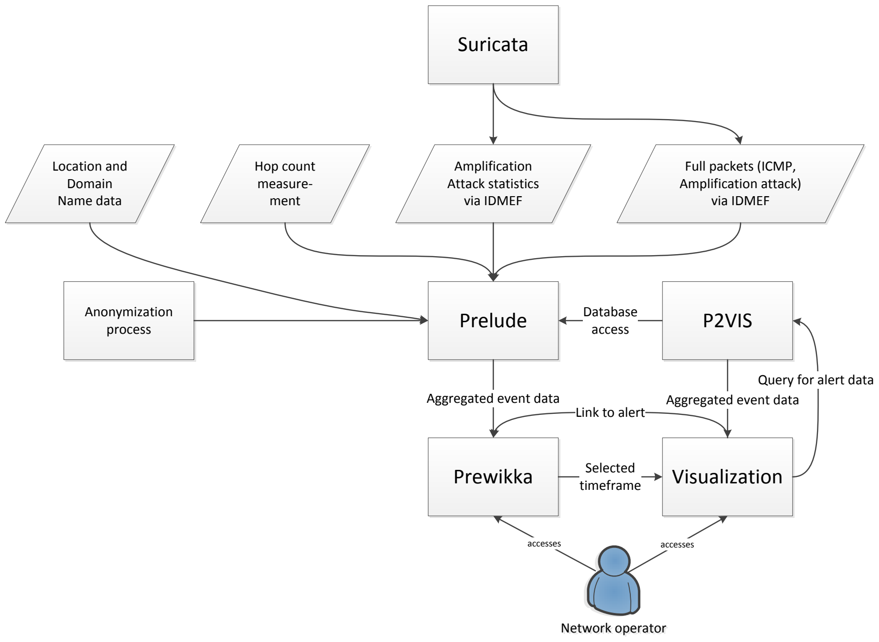


Figure 6.1: System structure

network within this thesis. Within the test setup, each PC had a role, which we describe in the following. A graphical representation of the development system is shown in figure 6.2.

Attacker The attacker PC was set up with an IPTables rule to change the source address of all outgoing requests to the victim's. Therefore, we could use legitimate applications to send requests to the *Amplifier*.

Victim The Victim PC received all amplified answers to the attacker's requests. It was therefore used only passively to analyze incoming traffic using Wireshark [33]. In addition to that, it was used to send out ICMP unreachable messages and answer to hop count measurements.

Border router The border router separated the outside network of *victim* and *attacker* from the internal network, where the *amplifier* sat. It therefore had two network links connected to gigabit ethernet switches for the two networks. On this PC, also Suricata and Prelude were running to scan for attacks.

Amplifier The amplifier was sitting on the other side of the border router compared to the attacker and was running an NTP and DNS server configured to be susceptible to Amplification attacks.

record keeper For DNS amplification, we used an external record keeper for the am-

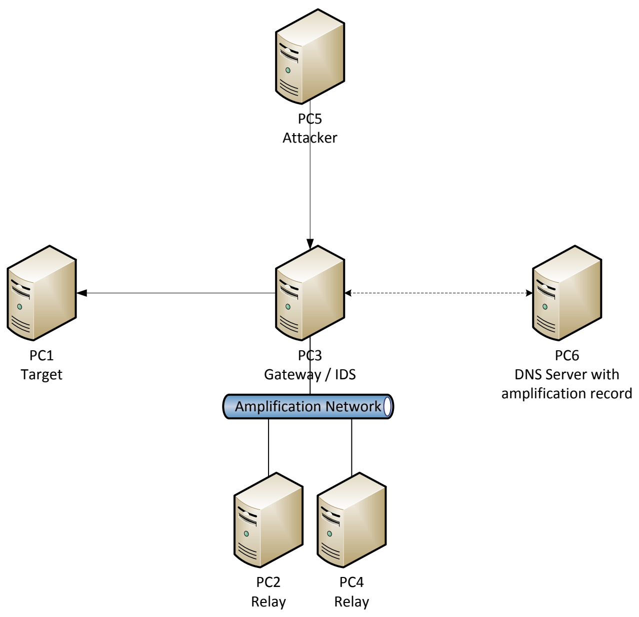


Figure 6.2: Test system setup

plified DNS record. Therefore, on this PC another DNS server was set up to serve requests, in our case open recursive requests from the *Amplifier*.

We used the development system to configure Suricata and Prelude to work together and finally collect amplified packets within the Prelude database. Due to the fact that the interface from Suricata to Prelude was not finished when we submitted this thesis, we were unable to incorporate it to do complete measurements. Therefore, we manually generated the additional IDMEF messages using the interface definition from chapter 4.2.1.2 to manually generate these additional alerts within the Prelude database. Based on these, we developed and tested the P2VIS interface.

6.2 Data input

For the data input definition, we relied on the specification of amplification attacks and interesting data by Böttger (et al.) [3, 4], but also added our own ideas, as outlined below the list. Their work defined pairflows, which inform about the addresses of affected systems. In addition to that, they evaluated the following criteria, although they dropped some of them as not usable for automatic detection.

Similarity in size and content of request and response packets

ICMP unreachable messages

hop count measurements

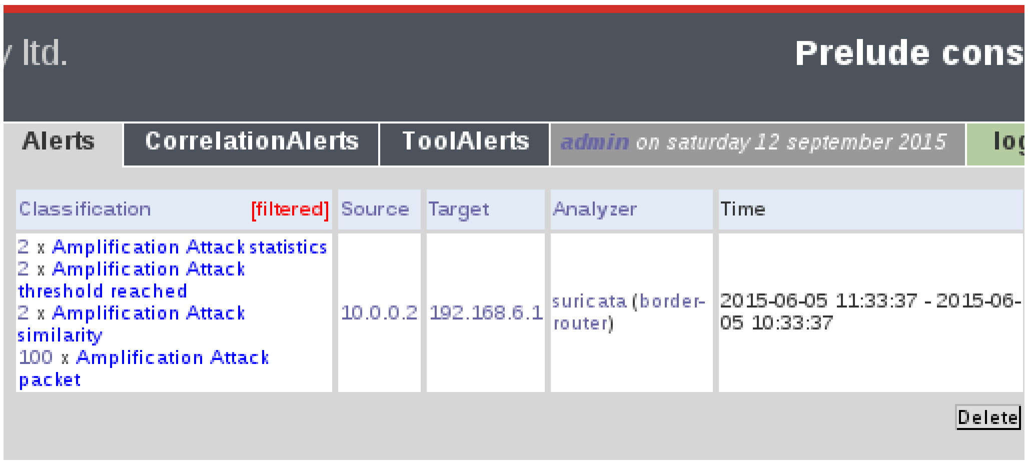
Moreover, we searched for additional identifying data that gives new information. Due to the fact that we focus on visualization and do not aim at extending the actual detection, we were restricted to information we could generate out of the existing data. The only defined datasets within the definition of Böttger (et al.) were IP addresses and port numbers, as the other data originated from measurements. Due to that, we searched for data that can be generated from those pieces of information. As such, we found domain names, autonomous system numbers and location information from *whois* databases can describe connections and similarities between systems we wouldn't know on basis of IP addresses. However, we decided to only maintain these pieces of information from the external network, which the victim is part of. We argued that the internal network documentation should exceed the information we could get. Thus, we do not have any additional identifying information for devices on the internal network. This means that correlation has to be done on a manual basis here, which is possible by using the IP address knowledge about the network structure. To overcome this, we suggest to build an interface to the network documentation similar to what we did with the external network in chapter 4.3.1 and include the data into P2VIS and the visualization.

Out of the presented information, we created the set of graphs described in chapter 4.4, which means that we included all data we had available. This means that the evaluation is limited to the actual usage of the data which will be checked in the following chapter.

6.3 P2VIS interface

The P2VIS interface uses a MySQL query to query the Prelude Database for the necessary data and outputs it to the visualization in an aggregated form, as outlined in chapter 4.4.1. Currently, it only transfers the data used within the graphs. Using this design, we avoid transmission overhead while providing the required information. To provide additional information about detailed packet data we include the alert ID to link back to the alert representation within Prewikka. Using this approach, we combine the advantage of no data overhead within the graph data with the ability to get detailed information on request.

Using the link from the Visualization to Prewikka thus enables us to view detailed information about the attack. Within Prewikka, we are presented with a list of alerts belonging to the event we selected which can be seen in figure 6.3. This contains the specified IDMEF messages from chapter4.2.1.2, as well as the captured packets. Using these, we can evaluate the packet contents and full headers, which can be seen in figure 6.4 and 6.5. This facilitates in-depth analysis of an attack and also allows for manual comparison to other attacks on basis of the packet content. The information within the additionally specified alerts can be used to analyze the chronological development of an attack, by comparing the statistical values among the alerts.



Classification [filtered]	Source	Target	Analyzer	Time
2 x Amplification Attack statistics 2 x Amplification Attack threshold reached 2 x Amplification Attack similarity 100 x Amplification Attack packet	10.0.0.2	192.168.6.1	suricata (border-router)	2015-06-05 11:33:37 - 2015-06-05 10:33:37

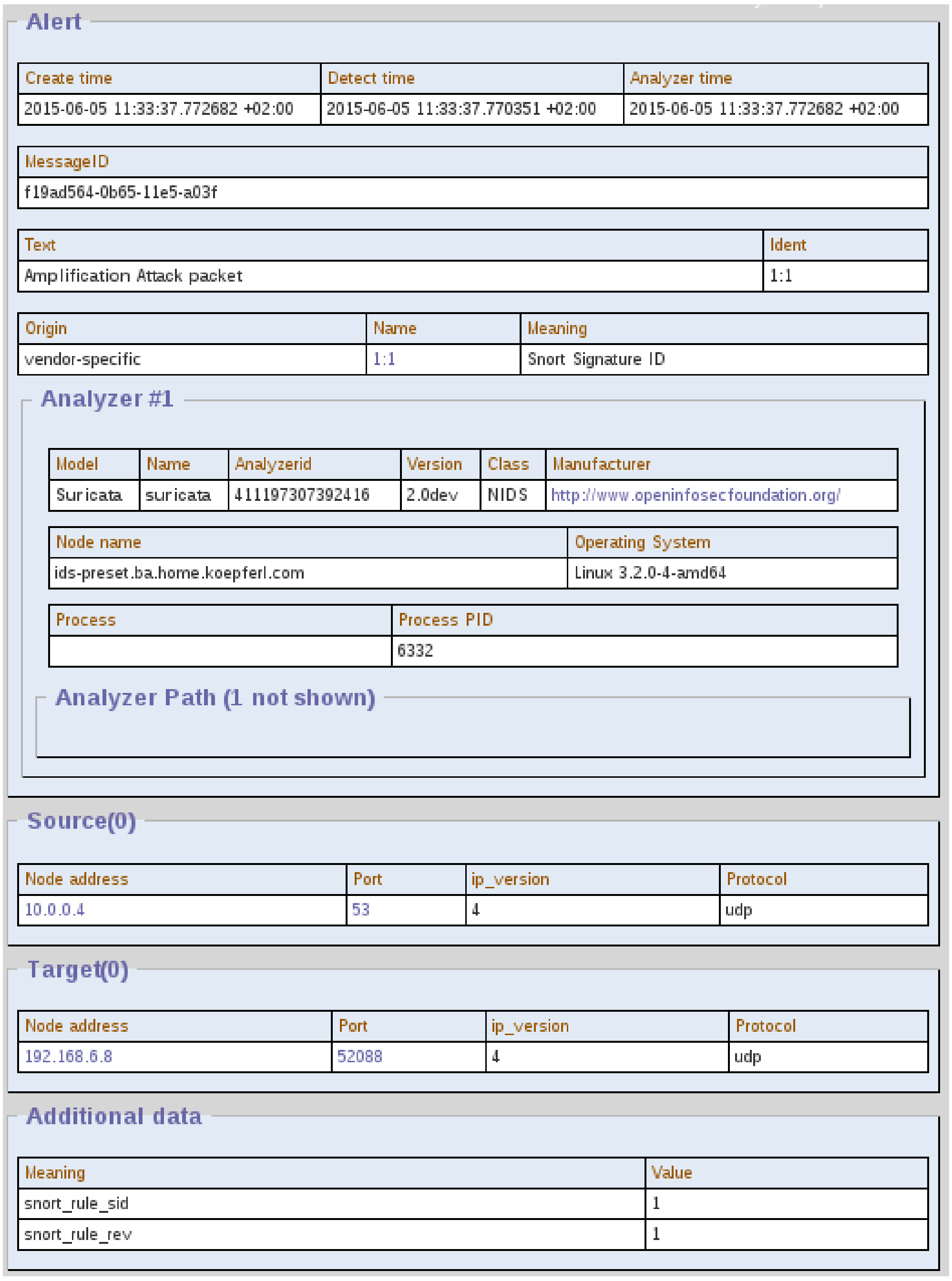
Figure 6.3: Prelude alert overview

This partly answers *Q4*, as the network operator gains in-depth information about all packets belonging to an attack for later analysis. This helps finding the root cause of an attack to finally solve the problem by fixing the responsible configuration or implementation issue. In case of a design problem, it facilitates understanding the weakness so that it can be investigated. In-depth analysis also allows exploring possible false positives or false negatives using the same approach. This was one goal of *Q5*. The determination of false positive or false negative suspects and earlier attacks, however, will be examined within the following chapter.

Also, the interface is limited to Amplification attacks due to fixed comparisons within the MySQL select statements. This was a design choice due to our special use case. For similar attacks, like the mentioned RDoS attacks, the interface could be reused easily, as they basically are of the same structure, but do not have noticeable amplification factors. As general information, Prewikka, if used for the collection of other alerts apart from Amplification attacks, includes basic graphs informing about events on a timescale and the distribution of event types within a timeframe. For more specialized cases, however, the structure of P2VIS could be reused to build more advanced graphs using our set of features. The graphs, however, would have to be built as necessary.

6.4 Visualization

The visualization was designed with the goal in mind to include all available information and presenting it in an understandable way. Therefore, we now have to evaluate if it is suitable to analyze amplification attacks. Thus, we work though the list in chapter 4.4.2, which was sorted by the order of filtering we suggest. Where appropriate, we describe alternative orders of filtering. For all graph types, we include an image showing the



Alert

Create time	Detect time	Analyzer time
2015-06-05 11:33:37.772682 +02:00	2015-06-05 11:33:37.770351 +02:00	2015-06-05 11:33:37.772682 +02:00

MessageID
f19ad564-0b65-11e5-a03f

Text	Ident
Amplification Attack packet	1:1

Origin	Name	Meaning
vendor-specific	1:1	Snort Signature ID

Analyzer #1

Model	Name	Analyzerid	Version	Class	Manufacturer
Suricata	suricata	411197307392416	2.0dev	NIDS	http://www.openinfosecfoundation.org/

Node name	Operating System
ids-preset.ba.home.koepferl.com	Linux 3.2.0-4-amd64

Process	Process PID
	6332

Analyzer Path (1 not shown)

Source(0)

Node address	Port	ip_version	Protocol
10.0.0.4	53	4	udp

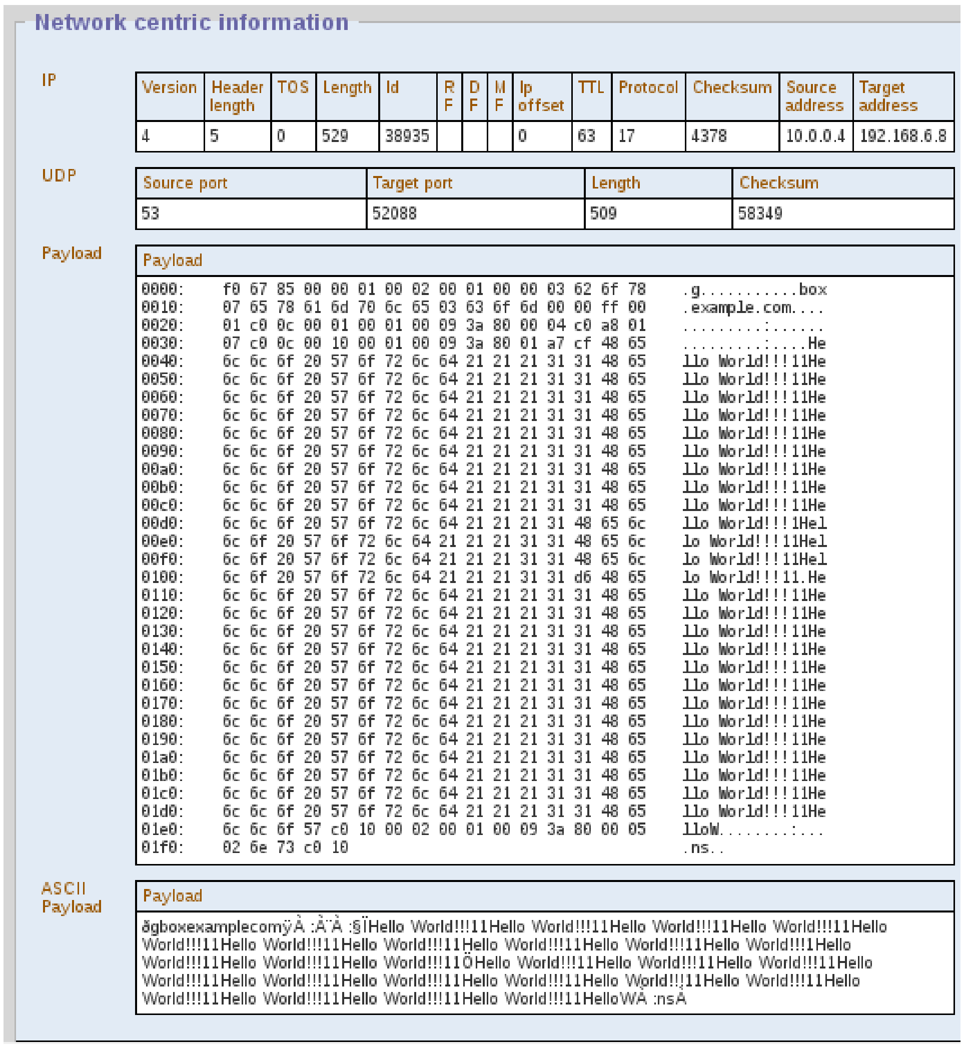
Target(0)

Node address	Port	ip_version	Protocol
192.168.6.8	52088	4	udp

Additional data

Meaning	Value
snort_rule_sid	1
snort_rule_rev	1

Figure 6.4: Prelude packet detail



IP

Version	Header length	TOS	Length	Id	R F	D F	M F	Ip offset	TTL	Protocol	Checksum	Source address	Target address
4	5	0	529	38935				0	63	17	4378	10.0.0.4	192.168.6.8

UDP

Source port	Target port	Length	Checksum
53	52088	509	58349

Payload

Payload

```
0000:    f0 67 85 00 00 01 00 02 00 01 00 00 03 62 6f 78    .g..........box
0010:    07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 ff 00    .example.com....
0020:    01 c0 0c 00 01 00 01 00 09 3a 80 00 04 c0 a8 01    .........:......
0030:    07 c0 0c 00 10 00 01 00 09 3a 80 01 a7 cf 48 65    .........:....He
0040:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0050:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0060:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0070:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0080:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0090:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
00a0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
00b0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
00c0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
00d0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 48 65 6c    llo World!!!1Hel
00e0:    6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65 6c    lo World!!!11Hel
00f0:    6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65 6c    lo World!!!11Hel
0100:    6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 d6 48 65    lo World!!!11.He
0110:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0120:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0130:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0140:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0150:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0160:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0170:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0180:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
0190:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
01a0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
01b0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
01c0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
01d0:    6c 6c 6f 20 57 6f 72 6c 64 21 21 21 31 31 48 65    llo World!!!11He
01e0:    6c 6c 6f 57 c0 10 00 02 00 01 00 09 3a 80 00 05    lloW........:...
01f0:    02 6e 73 c0 10                                     .ns..
```

ASCII Payload

Payload

```
ðgboxexamplecomÿÀ :À¨À :§ÏHello World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello
World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello World!!!1Hello
World!!!11Hello World!!!11Hello World!!!11ÖHello World!!!11Hello World!!!11Hello World!!!11Hello
World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello World!!!11Hello
World!!!11Hello World!!!11Hello World!!!11Hello World!!!11HelloWÀ :nsÀ
```

Figure 6.5: Prelude packet header and content

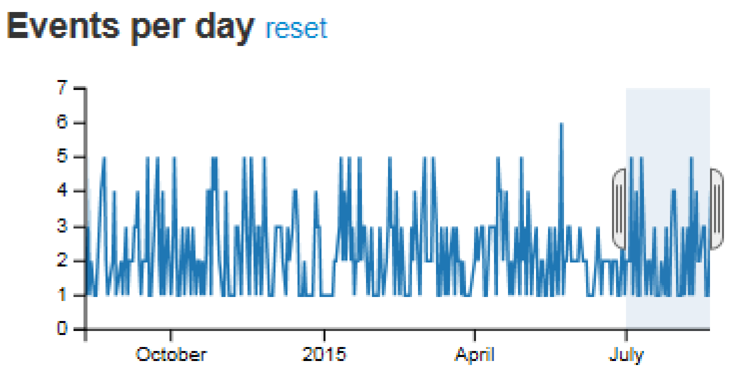


Figure 6.6: Events per day graph

described behavior. If a graph is very similar to another one, we refer to that instead and describe the differences in detail.

Events per day displays one year worth of sampled random data in this example. Due to the long timeframe, the graph has very steep curves, but there is still a separate value recognizable for each day. Due to this, we can find suspicious events quite easily. The selected timeframe limits the data to about 6 weeks. This filter is attached to the other graphs as well, which means that all graphs now only display data that is within these time limits.

The count of events per day as a metric was preferred to count of packets or bytes due to two reasons: First, there is no chance that smaller attacks get out of sight because of a single, large attack. Second, an ongoing attack will be represented as multiple events due to the limited pairflow lifetime. Thus, after timing out, a new event will be recorded.

Alternatively, we offer a possibility to filter for large attacks with the "Events by packet size" graph we will outline below. If the network operator wants to use such filter, it should therefore be the first choice to filter for large attacks using that graph. Including graphs for both metrics, however, would be problematic due to screen size limits, as all main graphs should fit onto one screen.

Similarity is a set of pie charts for size and content similarity. As this is the only critical value Böttger et al. [3] proposed, which is not preselected for us, we select it directly after limiting the timeframe. Within evaluation we recognized that it could have been useful to split up the range of values further. That way, also smaller similarity indices could be selected. It is, however, easy to change the value or add another level within the graph definition.

Events by Amplification Factor displays a bar for each amplification factor. The height describes the number of events in the current selection having this factor. High amplification factors pose a threat to the system no matter of the attack size,

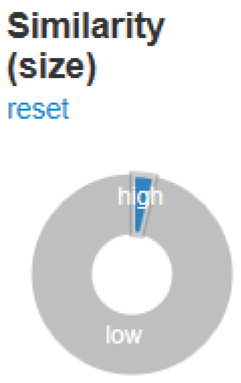


Figure 6.7: Similarity (size) graph

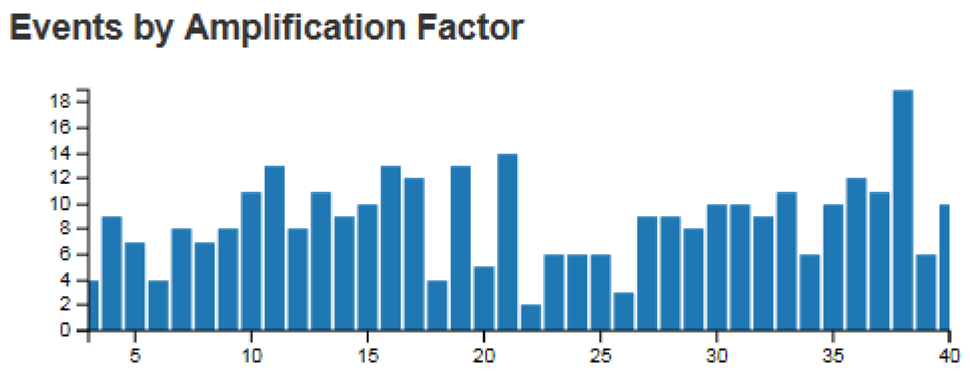


Figure 6.8: Events by amplification factor graph

because an attack can be extended by the attacker easily. She could for example increase the number of devices used within a bot net. Thus, we propose to filter on the highest factors first and evaluate the corresponding attacks.

Number of Events by packet count (in,out) displays a bar for each packet count. In the shown example, we selected the larger half of captured events. This allows us to point at the correlation between the incoming and outgoing packets: we must have huge amount of repeated packets within the selected events, like described in chapter 3.4.3 for the TCP protocol. To allow for such comparisons, we scaled the x axis representing the packet count equally for both graphs, although we do not have any events for most of the values on the incoming graph.

Number of Events by packet size (in,out) is designed equally to the last graph and therefore not printed here. At the "Events per day" graph, we shortly introduced that this graph can be used to select large attacks, which is possible by limiting the graph displaying the outgoing data to the desired values. It could be argued that the graph displaying the inbound values is not necessary, as filtering is possible within the amplification factor graph due to their correlation. However, this way

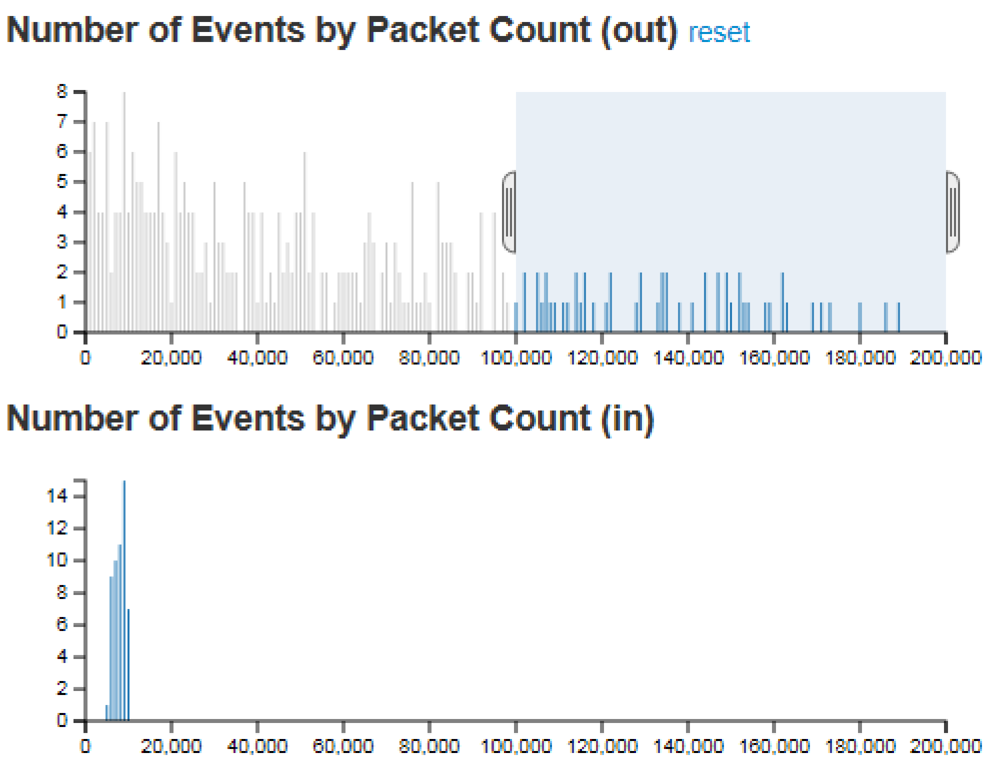


Figure 6.9: Events by packet count graph

we would lose the information about the distribution of values over the events displayed.

ICMP packets count and TTL+Hop Count can be used to further isolate an attack to evaluate the occurrence of abnormalities.

All of the following elements can be used for filtering as well. However, their main purpose is of informational nature: Each shows a distribution of the selected set of events according to their respective values. In the following, we will describe some use cases next to the graphs:

IP Layer Protocol is also displayed as a pie chart equal to the similarity graphs, using the values IPv4 and IPv6. It could be used to filter the overview e.g. to recognize IPv6 configuration flaws that allow for amplification attacks by comparing the graphs with or without the filter.

Network Layer Protocol does the same for TCP and UDP.

Destination/Source IP can be used to zoom in to an IPv4 network using three pie charts representing the first three octets, as described in chapter 4.4. These graphs, as well as the following, depict distribution on basis of slices for the different values which sizes represent the event count. This allows narrowing down on an attack.

Destination/Source Port graphs are bigger than the previously depicted pie charts, sharing the rest of the structure. Using these, it is possible to filter for special protocols on the server side based on the well-known ports allocation.

AS Number, Top-level Domain and Country Code can be used to learn about correlations of attacks. For example, we could chose some value within one graph and evaluate the remaining events shown in the table below and the other graphs to determine correlations. Using these, it is possible to gain knowledge no current automatic detection or logfile analysis offers.

The Table gives the basic information about the filtered events and is the link back to Prewikka. Also, it is filtered according to event size so that the largest attacks measured in bytes are listed on the top. Thus, these can be evaluated first. The exact structure and functions are described in chapter 4.4.

6.4.1 Protocol fault coverage

In chapter 3.4, we looked at a selection of vulnerable protocols and described their special properties during amplification attacks. Now, we check if our visualization can cover those characteristics.

For NTP, we described the very high similarity and amplification factor. If we would not suspect an amplification attack on that service - would we still be able to detect it using the visualization? First of all, we can see a bar at the calculated amplification factor, which is outstanding due to long distance to the other bars which represent the lower amplification factors. We then filter on the similarity first as suggested above. Still, the bar is there, as NTP amplification attacks are very similar due to the small set of queries. Now, we are really interested and filter on the amplification graph. The port chart now shows only one remaining element, which is the NTP port. Thus, we have identified the protocol. Depending on the number of abused NTP servers within our network and the timeframe, we get a number of alerts. Within the table, we can now jump to the corresponding alert within the Prewikka web interface to view details like packet content. Thus, we can evaluate the attack in detail and eventually prevent future abuse.

For the described TCP attacks, however, the procedure is a different, as we do not expect a very high amplification factor, but repeated packets. We can determine that from the packet count graphs. Also, we can there filter on the area of the outgoing graph where the incoming graph is empty, as shown in figure 6.9. This way, we only see attacks that rely on repeated answers. Using the other filters, we can now localize the problem.

6.4.2 False positives and false negatives coverage

Analysis of these errors cannot be automatic, as otherwise it would have been included into the detection, thus eliminating them. Thus, we want to check whether identification is possible using the visualization.

We first focus on false positive interpretation of normal usage as Amplification attack. Therefore, we need to understand the regular usage scenario of the network, which is information the network operator can contribute. Using this, we can look at the filtered graphs which show possible suspicious action. We can now evaluate the IP address and port graphs and filter out the protocols we know to generate attack-like traffic. Now, we evaluate the graphs once more. If they are now inconspicuous, we have determined a false positive. We can now proof our findings by doing in-depth packet inspection as described above.

For the detection of false negative traffic, we have to adapt the Suricata ruleset to change the thresholds, as described in chapter 5.1. Then, we can evaluate the additional traffic by filtering the lower values of the Amplification factor and Packet size graphs. Thus, we can especially look at the attacks using the general approach described in the beginning of this chapter and take advantage of the packet inspection features of Prewikka to evaluate the alerts. Finally, we can decide whether we want to further limit, leave as is or increase the thresholds in the Suricata ruleset.

For both approaches, it could be useful to watch the Time of day and Day of week graphs, as these can inform about repeating patterns, which are most probably a backup job or other automated traffic.

6.5 Recap on scientific questions

We now evaluated the possibilities of the visualization thoroughly. Thus, we want to reconsider on our scientific questions and point to the answers.

The warning message, as defined in *Q1*, is part of the default features of any SIEM system, including Prelude. Within the tool prelude-manager, e-mail alerts can be configured, which solves our problem. The network operator should then be able to immediately make use of the alert, which is possible by opening the visualization with a short timeframe. Thus, he is informed about the core data describing the attack.

Q2 demands an overview which informs about the affected systems of an amplification attack and its extent. This is possible by using the visualization as well, as it can output the total volume of attacks for a timeframe, distributed into the graphs determining trends and accumulations. Therefore, the distribution of attacks among systems, ports and protocols can be determined from the graphs. We also added the values of domain name, autonomous system and location to allow for additional correlation. The table allows to determine the mapping between amplifier and victim.

Correct determination of short-term actions was the goal of *Q3*. Here, we point at our answer to *Q2*, as this is exactly points out the required information. Therefore, the visualization output can be used to determine the input for firewall blocking or rate limiting rules or to shut down systems.

Q4 is based on *Q2* as well. However, it aims at solving the problems on the long run by eliminating amplifiers. For this, we point at the packet inspection view of Prewikka, which can be opened from within the visualization. This should help narrowing down the fault to finally fix it. Therefore, we proposed a taxonomy of possible problems within chapter 3.4.

Q5 was tackled in a separate part with the conclusion that possibilities exist.

Chapter 7

Conclusion

Within today's interconnected world, we risk economical or even security problems in case of network outage. Nowadays, new security breaches are announced almost on a daily basis. Some of them pose a hazard to the core of our infrastructure. We focused on Amplification attacks, which are a variation of Denial of Service attacks, increasing the impact. Related work researched many aspects of Amplification attacks: a broad field of detection attempts on the victim side exists, using different approaches of detection and also visualization. Additionally, the research about susceptible protocols and attack patterns shows the significance of the hazard. There were previous attempts to mitigate the issue broadly by correcting faults. In addition to that, approaches to detect the attacks within amplifier networks exist.

However, none of the related work examined the way of visually evaluating these attacks in that position. Existing visualization approaches were not suitable due to the singularity of the threat. Thus, we researched possibilities overcoming this to learn about the threat in detail the visual way. We therefore introduced a set of research questions with the goal in mind to create a usable solution for a network operator which enables him to detect attacks immediately, but also offers detailed information for evaluation.

7.1 Summary

In the beginning of this thesis, we introduced the attack pattern of amplification attacks. We therefore described the development from simple DoS attacks to distributed and reflected ones and provided a taxonomy to confine our domain. Additionally, we covered the impact and remediation possibilities from different points of view. The description of general properties of software relevant was followed by a presentation of our used software. We then focused on existing approaches to the threat from a visualizing amplifier's point of view, but also included other ideas where relevant. We then proposed a taxonomy of vulnerable protocols which we reused in the end to evaluate our visualization for problem coverage. In addition to that, we evaluated existing visualization approaches and toolsets for usability within our approach and justified our selection of D3.js with to the set of available features and the usability. Subsequently, we proposed a system architecture enabling visualization and extended evaluation of Amplification attacks that built upon an existing detection approach using Prelude as a SIEM system and D3.js for visualization. Therefore, we proposed an extended interface for Suricata to transfer all data relevant for analysis. Within the SIEM system, we added additional interfaces to enrich the visualization. In addition to that, we added anonymization and a search history to allow for and facilitate system usage. Following that, we described the main parts of our development to allow additional insight into the functionality of our approach. This contained the interfaces described above and a data transporting tool for the visualization to import data from the Prelude database. Finally, we evaluated the implementation against our defined goals. The visualization appeared to be useful, compared to our scientific questions. However, future deployment within a production system has to prove usability in a daily usage scenario.

We concluded that visual information greatly increases perceivability, if designed adequately. A user-friendly solution facilitates regular evaluation of attacks and thus helps to understand the issues to the system in more detail. This is a first step towards a solution of the problem.

7.2 Outlook

We were unable to test our solution with real data, as the interface to Suricata was not finished when submitting this thesis. Therefore, we are looking forward to future usage within an amplifier network. There, it would be possible to compare the information retrieved from the visualization to the logfile output of detection approaches. From user reports, improvement suggestions or additional necessary information could be retrieved. For example, it could be necessary to adapt some of the graphs to use different scales or types of input data. For a first solution, however, we kept to user expected defaults and only proposed other choices where appropriate.

Using the visualization, it is easy to detect correlations within the set of attacks. It is, for example, possible to see that two servers of the same type were abused for an amplification attack on the same port. In this case, we suspect that it was the same underlying attack scheme. We can also compare the metrics included in the visualization. However, if we want to proof the correlation, we have to check the packet contents and compare them manually. The visualization, however, facilitates finding such similarities. Future work thus could make use of this and propose how to do inter-pairflow similarity checking based on the outlined facts.

We also included a search history into our approach to facilitate reinterpretation of data for future reference and comparison. However, we were unable to store the selection within the visualization due to client-based design. Our approaches to save this data on the server were not successful. Thus, future work could add this feature. This, of course, could also be interesting for other approaches based on client-side visualization or handling of large datasets.

We presented another visualization approach that appeared towards the end of the thesis. It focused on a data-driven design that enables the user to view graphs based on proposals the program gives due to the data structure. Future work could build upon this to dissect the packet data and full header information and pass it to the toolset. This way, even more information could be evaluated to determine if it contains criteria useful to classify amplification attacks.

Bibliography

[1] ICANN, “SSAC Advisory SAC008 DNS Distributed Denial of Service (DDoS) Attacks,” ICANN, Tech. Rep. March, 2006. [Online]. Available: https://www.icann.org/en/system/files/files/dns-ddos-advisory-31mar06-en.pdf

[2] C. Rossow, “Amplification hell: Revisiting network protocols for DDoS abuse,” *Symposium on Network and Distributed System . . .*, no. February, pp. 23–26, 2014. [Online]. Available: http://www.internetsociety.org/sites/default/files/01_5.pdf

[3] T. Böttger, “Detection of Amplification Attacks in Amplifier Networks,” Master’s Thesis, Technische Universität München, 2014.

[4] T. Böttger, L. Braun, O. Gasser, F. von Eye, H. Reiser, and G. Carle, “Dos amplification attacks – protocol-agnostic detection of service abuse in amplifier networks,” in *Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, M. Steiner, P. Barlet-Ros, and O. Bonaventure, Eds. Springer International Publishing, 2015, vol. 9053, pp. 205–218. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17172-2_14

[5] S. Hansman, “A taxonomy of network and computer attack methodologies,” *Engineering*, pp. 1–48, 2003.

[6] S. Specht and R. Lee, “Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures,” *nternational Conference on Parallel and Distributed Computing Systems*, vol. 3, no. 17, pp. 543–550, 2004. [Online]. Available: http://palms.ee.princeton.edu/PALMSopen/DDoSFinalPDCSPaper.pdf

[7] P. Ferguson and D. Senie, “BCP 38: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing,” 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2827

[8] S. Gibson, “The Distributed Reflection DoS Attack,” Gibson Research Corporation, Tech. Rep., 2002. [Online]. Available: http://homes.cs.washington.edu/~arvind/cs425/doc/drdos.pdf

[9] J. Mirkovic and P. Reiher, “A taxonomy of ddos attack and ddos defense mechanisms,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004. [Online]. Available: http://doi.acm.org/10.1145/997150.997156

[10] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, “Exit from Hell ? Reducing the Impact of Amplification DDoS Attacks,” in *USENIX Security 2014*, 2014.

[11] J. Viinikka and H. Debar, “Intrusion Detection : Introduction to Intrusion Detection and Security Information Management,” *Foundations of Security Analysis and Design III*, vol. 3655, pp. 207–236, 2005.

[12] Open Information Security Foundation, “Suricata | Open Source IDS / IPS / NSM engine:,” 2015. [Online]. Available: http://suricata-ids.org/

[13] CS, Systèmes d’Information, “WikiStart - Prelude-SIEM - Prelude SIEM,” 2015. [Online]. Available: https://www.prelude-siem.org/

[14] M. Bostock, “D3.js - Data-Driven Documents,” 2015. [Online]. Available: http://d3js.org/

[15] M. Bostock, V. Ogievetsky, and J. Heer, “D 3 : Data-Driven Documents,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[16] ICANN, “Root server attack on 6 February 2007,” ICANN, Tech. Rep., 2007. [Online]. Available: https://www.icann.org/en/system/files/files/factsheet-dns-attack-08mar07-en.pdf

[17] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, “Detecting DNS amplification attacks,” *Critical Information Infrastructures Security*, vol. 5141, no. 2008, p. 11, 2008. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-89173-4_16

[18] C. Sun, B. Liu, and L. Shi, “Efficient and low-cost hardware defense against dns amplification attacks,” in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE.* IEEE, 2008, pp. 1–5.

[19] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer, “Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations,” *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2015. [Online]. Available: http://idl.cs.washington.edu/papers/voyager

[20] H. Yu, X. Dai, T. Baxliey, X. Yuan, and T. Bassett, “A visualization analysis tool for DNS amplification attack,” *2010 3rd International Conference on Biomedical Engineering and Informatics*, no. Bmei, pp. 2834–2838, Oct. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5639324

[21] H. Shiravi, A. Shiravi, and A. a. Ghorbani, “A survey of visualization systems for network security.” *IEEE transactions on visualization and computer graphics*, vol. 18, no. 8, pp. 1313–29, Aug. 2012. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/21876227

[22] P. Mockapetris, “RFC 1034: DOMAIN NAMES - CONCEPTS AND FACILITIES,” 1987. [Online]. Available: https://www.ietf.org/rfc/rfc1034.txt

[23] P. Mockapetris, “RFC 1035: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION.” [Online]. Available: https://www.ietf.org/rfc/rfc1035.txt

[24] J. Damas, M. Graff, and P. Vixie, “RFC 6891: Extension Mechanisms for DNS (EDNS(0)),” 2013. [Online]. Available: https://tools.ietf.org/html/rfc6891

[25] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, “Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks,” in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/woot14/workshop-program/presentation/kuhrer

[26] J. Hawkinson and T. Bates, “Guidelines for creation, selection, and registration of an Autonomous System (AS),” 1996. [Online]. Available: https://tools.ietf.org/html/rfc1930

[27] Internet Assigned Numbers Authority, “IANA IPv4 Address Space Registry,” 2015. [Online]. Available: http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

[28] Bundesrepublik Deutschland, “Bundesdatenschutzgesetz,” 2003. [Online]. Available: http://dejure.org/gesetze/BDSG

[29] J. Postel, “RFC 792: INTERNET CONTROL MESSAGE PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION,” 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc792.txt

[30] Square Inc., “Crossfilter,” 2015. [Online]. Available: http://square.github.io/crossfilter/

[31] Woodhull, Gordon, “dc.js - Dimensional Charting Javascript Library,” 2015. [Online]. Available: http://dc-js.github.io/dc.js/

[32] Grml Live Linux, “Grml Live Linux,” 2015. [Online]. Available: https://grml.org/

[33] Wireshark Foundation, “Wireshark - Go Deep.” 2015. [Online]. Available: https://www.wireshark.org/