University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

DDoSGrid 2.0: Integrating and Providing Visualizations for the European DDoS Clearing House

Jan von der Assen
Switzerland, Zurich
Student ID: 14-719-132

Supervisor: Muriel Franco, Bruno Rodrigues
Date of Submission: February 1, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

DDoS-Attacken sind seit Jahrzehnten eine reale und häufiger werdende Bedrohung für Internetdienste. Die daraus resultierende Bedrohung der Reputation und des Einkommens von Organisationen motivieren Bemühungen, diese Attacken abzuwenden. Die Post-Mortem Analyse von DDoS-Attacken kann zu wichtigen Erkenntnissen für die langfristige Planung von Abwehrmechanismen dienen. Wichtige Abwehrmechanismen sind unter anderem die Schulung von Personal und der Einsatz von Services. Diesen Prozessen unterliegt die Erkennung von Mustern, welche durch die visuelle Darstellung von Merkmalen des Netzwerktraffics vereinfacht werden kann.
Diese Arbeit analysiert visuelle Post-Mortem Analyse-Applikationen. Entdeckte Schwächen werden im Besonderen von zwei Applikationen überwunden. *DDoSGrid*, eine Visualisierungsplattform für DDoS Attacken, verfügt über eine verbesserte Skalierbarkeit bei der Darstellung grosser Datenmengen. Die modulare Architektur erlaubt zudem die Entwicklung neuer Visualisierungen. Im Rahmen des *DDoS Clearing House* Projekts wurde eine Plattform entwickelt, die Austausch und Entdeckung von Netzwerkaufnahmen ermöglicht. Das Hauptziel dieser Arbeit ist die Verbesserung und Integration beider Plattformen. Die resultierende Analyse-Plattform ermöglicht das Auffinden von Netzwerkaufnahmen und die Analyse auf verschiedenen Abstraktionsschichten. Die im Rahmen der Evaluation durchgeführten Experimente legen nahe, dass dieser Anwendungsfall auch mit grossen Datenmengen funktioniert, ohne die Benutzerfreundlichkeit zu beeinträchtigen.

Abstract

DDoS attacks have posed an increasing threat to infrastructure built on top of the internet for years. The resulting threat to an organization's reputation and financial income motivate efforts in mitigating such attacks. Post-Mortem analysis of DDoS attacks generates insights that can be used for long-term planning of defense mechanisms. This includes the provisioning of services, development of personnel and the detection of novel attack patterns. The underyling task of detecting patterns can be supported by visualizing relevant features of network traffic.
This paper analyzes the current state of visual Post-Mortem analysis tools. Two applications overcome weaknesses exposed in other visualization tools. *DDoSGrid*, a visualization platform for DDoS attacks, overcomes the scalability issues observed in other tools. The modular architecture also supports the implementation of novel visualizations. In a different way, the *DDoS Clearing House* project developed a platform which allows victims and researchers to explore and share attack data. The main outcome of this work is the integration and improvement of these two platforms. This yields an analysis platform where victims can share attack data and analyse it on various abstraction levels. The experiments conducted in our evaluation suggest that this use case can be performed with large datasets, while maintaining appropriate usability.

Acknowledgments

I want to thank my supervisor Muriel Franco for his continuous efforts throughout this thesis. Muriel's guidance in technical and academic matters has made this thesis a rewarding closure of my studies.

I would like to extend my gratitude to the other members of the Communication Systems Group who contributed in some way or another. I thank Bruno Rodrigues for repeatedly providing his insights and Eder Scheid for helping us build the operational infrastructure.

Finally, a thank you to Prof. Dr. Burkhard Stiller for enabling this project and introducing me to computer networking at the beginning of my studies.

Last but not least, I wish to thank my family and especially Alma Gnani for their everlasting support.

Contents

Chapter 1

Introduction

The Internet was originally designed to interconnect only a small number of institutions. This network has become a critical component of today's society. Although resilience against attacks was one of the design goals during the Internet's creation, Distributed Denial-of-Service (DDoS) attacks have become a serious threat against its operation. Although these types of attacks have been known for multiple decades, we find them constantly growing in intensity and frequency. This has led to the development of several mitigation solutions [59], with detection and defense being the most important activities [4]. This thesis aims to improve a DDoS visualization system. Therefore, the following chapter describes the motivation for such a system and the scope and structure of this report.

1.1 Motivation

Over the last years, DDoS attacks have constantly grown in scale regularly achieving new records. Previous predictions on the increase in DDoS attack sizes and frequency as well as an increased adoption rate of commercial DDoS mitigation solutions [5] [4] have turned out to be true. For example, in late June 2020 Akamai was faced with the largest DDoS attack ever recorded in terms of the number of packets per second (PPS). The attack peaked at around 800 million packets per second (Mpps) and grew to a bandwidth of around 400 Gigabits per second (Gbps) within seconds. The mitigation provider deemed the attack to be sophisticated for its nature [8]. A similar observation was made by Cloudflare who was faced with a DDoS attack of a similar scale on the very same day. That attack was also of sophisticated nature. By using a multivector attack it effectively targeted weaknesses in the TCP protocol. Further, the attack was considered to be specifically targeting the DDoS mitigation solution since it tried to deny the services by overwhelming them with a large number of small packets instead of trying to saturate links with high bandwidth traffic [9].

From an economic perspective, DDoS attacks are still a highly relevant threat to businesses' reputations and can lead to financial losses [62]. Quantifying the impact of a

DDoS attack is difficult with different parties suggesting average numbers between tens of thousands of dollars up to millions of dollars lost per attack [6] [7]. Akamai reports that specific industries are faced with DDoS attacks. Financial service and Internet & Telecom companies appear to be frequent targets of such large-scale attacks [8].

Interestingly, the mitigation of both of these exemplary attacks was considered to be successful, with Cloudflare even reporting of a fully automated mitigation [8] [9]. Here, Cloudflare stresses the importance of planning and access to expert resources to understand traffic patterns and volumes to design a proper mitigation scenario. This connection of automatic and human mitigation motivates our need for tools that promote the understanding of attacks and traffic patterns [8]. Here, visualization techniques can support humans in identifying patterns from network logs that would otherwise be hard to understand [11] [63].

1.2 Description of Work

This thesis is concerned with the analysis and improvement of post-mortem DDoS attack analysis applications. Specifically, two projects are examined for the creation of a novel DDoS attack analysis suite. First, the *ddos-clearing-house* which was developed at the University of Twente, covering efforts that resulted in various software components that help researchers analyze and share DDoS attack information [12]. Secondly, components from the *DDoSGrid* application developed at the University of Zurich are considered. *DDoSGrid* is an application focusing solely on extracting and visualizing attack patterns from network traces [33]. Using components from both projects, an integrated post-mortem DDoS attack analysis platform is developed as part of this work. Therefore, the main goals of this work are to establish the current state of DDoS attack visualizations and analysis from literature and to develop the previously described system.

1.3 Thesis Outline

This introductory chapter has motivated the need for integrated DDoS attack analysis and visualization applications given the current trends of DDoS attacks. The remainder of this report is structured as follows: Chapter 2 introduces the reader to three topics required to understand DDoS analysis applications. The following chapter then explores such related work by reviewing the current literature for existing DDoS attack analysis systems. First, a definition and classification of the term ”post-mortem analysis system” is introduced. Then, existing applications are described and compared against the dimensions established in the definition. This yields an understanding of these existing tools and creates awareness for their strengths and weaknesses that motivate the desired integration of the two platforms. Chapter 4 consists of an in-depth analysis of one specific set of software components referenced in the previous chapter on related work. This analysis covers functional and technical aspects with the aim of deriving requirements toward the development of an integrated visualization system. Chapter 5 focuses on the design

and implementation of the system that meets the previously discussed requirements. The main goal of this chapter is to solve the technical aspect of integrating the various software components. Finally, an evaluation of the developed application in chapter 6 and a summary in the subsequent chapter conclude the report.

Chapter 2

Background

This chapter familiarizes the reader with the necessary background knowledge for the literature review on DDoS visualization systems given in the subsequent chapter. First, DDoS attacks are introduced. This provides the background information to understand the features that are being transformed during the visual transformation step. Thus, we first introduce concepts from applying visualization to security-related information and finally, we review the literature for features that can be extracted from network traffic. Here, we focus on DDoS detection mechanisms and especially the features used to classify attacks.

2.1 DDoS Attacks

DDoS attacks are attacks where the provider of a service is overwhelmed with an amount of traffic so high that it leads to a state where legitimate users are unable to access the service. Thus, DDoS attacks do not necessarily exploit specific vulnerabilities that would need to be present on the providers' infrastructure. Instead, a large number of hosts are used to create the necessary traffic volume for the attack [13]. To gain access to a large number of hosts other attacks are used, such as the distribution of malware via e-mail [14]. Design decisions and system administration processes that provide weak security have made compromising a large number of devices even easier [13]. Trends such as the Internet-of-Things (IoT) have led to a rise in the number of devices that are connected to the Internet. Often, such devices have no security mechanisms in place so that manually scanning for openly exposed services is enough to gain access to the devices. The existence of billions of such weakly secured devices has allowed attackers to create even more powerful attacks every year [14].

The results of these trends are an increasing number of attacks that severely impact several industries, as outlined in chapter 1.1. One important trend to reiterate is that DDoS attacks are dynamic. For example, it was observed that different protocol vectors have been targeted and that attacks were tailored to impact DDoS mitigation software. This was achieved by scaling up the attack by creating a higher number of smaller packets

instead of creating a high traffic volume in terms of overall bandwidth [8]. This is relevant for our overarching topic of DDoS analysis since these dynamics have to be discovered to configure DDoS mitigation systems accordingly. To understand the dynamics of these attacks better and to see how visualizations can reveal these patterns we introduce the most common attack vectors.

DDoS attack vectors can be categorized into direct and indirect attacks, where the latter are often referred to as reflector attack. A direct attack is present when the attacker directly sends a packet to the victim. The source address is therefore not relevant for the attack, although it is often spoofed to make the mitigation more complicated. In reflector attacks, a packet is sent to another node which then sends one or more packets to the actual victim. This makes the detection of the actual attacker more complicated and the mitigation even more complex since the attacker only appears as a controller behind the mediating intermediaries [13]. [15] extend this classification by claiming that direct and indirect attacks are considered infrastructure-level attacks. Additionally, [15] consider application-level attack vectors where the attacks exploit application- or implementation-specific properties.

Protocols used for DDoS attacks are usually from the network-, transport- or application-layer. On the network-layer, we can frequently observe ICMP-based attacks, with "Ping-of-Death" and "ICMP-flooding" being the most frequent. Both target the infrastructure-level and are most often carried out directly [15]. Nagpal et al. also describe a reflected application of ICMP flooding [13]. Considering the transport-layer, a frequent attack vector exploits the connection setup procedure used in TCP. One way to achieve this is to directly flood the victim with "SYN" packets indicating the first phase of a connection setup. Leaving many connections "half-open" requires substantial resources which results in service disrupture if those resources can be depleted [15] [13]. In another way, one can exploit the same procedure by sending SYN packets to an intermediary node with a spoofed source address. This causes the intermediary to send an acknowledgment ("SYN-ACK") of that setup packet to the actual victim [13].

With UDP flooding, we can see another popular protocol to be abused for attacks on the transport-layer. With this, one can target both the infrastructure-level as well as application-specific weaknesses. On the infrastructure-level attacks can be direct or reflected [15].On the application-layer attacks frequently target the HTTP protocol. They do so by sending large amounts of HTTP packets. These may also be malformed, which ultimately leads to resource depletion in the webserver used by the provider. Generally, we can group HTTP-based attack strategies into four categories; request flooding, session flooding, asymmetric attacks and, slow request/response attacks [20].

2.2 DDoS Visualizations

The human brain can spend a large number of its resources on visual processing. Computers, on the other hand, can process vast amounts of data in a short time. This is what makes the visualization of large amounts of data, as we would find it with DDoS attacks, a powerful application to discover patterns in large amounts of data. We can therefore

define a DDoS attack visualization system as a system that turns large amounts of log data into something where a human can discover the attack patterns [11].

To turn log data from a DDoS attack into a visualization we can follow the six-step process proposed by [24]. First, we need to understand what we are interested in. For example, if we want to create a visualization focused on visualizing SYN-flood attacks, we need to think about the information needed from the data. In this case, we would probably be interested in the distribution of TCP flags in our dataset. The next step is to look at the data that is available in our dataset. The third step would then be to process this information. Going back to our SYN-flood visualization example, if we have a text file of packets, we could extract the distribution of TCP flags over all packets and save this distribution in another text file. During this step we also need to apply other intermediate processes, for example to aggregate the data or to extend the extracted data with data from another source. The extracted information now needs to be visually transformed. For that, we need to understand the dimensions and scale of the information and map it to some visualization technique. With our previous example, we would now realize that there is only one dimension and that there are just a finite number of items in our distribution since the number of valid flag combinations in a TCP header is limited. This allows us to use a visualization technique like a pie chart to render the proportions. We look at more visualization techniques later in this chapter to understand when to use which chart. During this process of visual transformation, we would also decide on other factors such as, how the axis would be labeled and how color is used. Here, we need to understand more about the information we are looking at. For example, to understand when some value may be out of proportion, we need to have metrics to decide when we color something to attract the human eye. We visit such metrics in the next section on DDoS classification metrics. The fifth step is the view transformation where the previously create visual transformation may be viewed in different ways by manipulating some property of its representation, for example the scale. This should make it easier for the human to perform the final step of looking at the visualization, interpreting it, and deciding on some action [24].

Considering possible data sources for such a visual transformation, we often see traffic flows and packet captures to be used when analyzing DDoS attacks. Some researchers consider PCAP files to be the standardized way of capturing traffic [11]. Packet captures are considered to be more detailed than other data sources such as traffic flows since the packets are captured at the lowest layer of the network stack. This allows that all data that a host receives can be intercepted and analyzed. Unfortunately, one loses the knowledge about what the application did with the packet. For example, using a network capture we can see that a packet was dropped by a router but not the reason for doing so. Given that a packet capture is unfiltered data the size of a given dataset might become problematic.

On the other hand, traffic flows are considered to belong to a higher layer than packet captures, implying that some information is lost. This data is usually captured on the transport layer by network equipment, which still does not provide insight into an application's behavior. It is important to note that there are different implementations of network flows by different vendors. Other difficulties include that these captures are often turned off or sampled due to performance reasons when routing the actual traffic. [24].

Finally, more insight into actual techniques and graphs that can be used during the visual transformation step is necessary to understand the existing visualization implementations in the next chapter. There exists a large number of visualization techniques and graphs that can be applied for the visualization of a DDoS attack. Such charts can range from simple charts, like pie charts with which many people are familiar with, to complex charts, for example link graphs.

Since datasets on DDoS attacks can be large it is important to consider and implement the right chart for a given problem. For example, a pie chart could be used to display the distribution of TCP flags over all packets for several of reasons. First, the number of values would be quite low since there is only a finite number of valid combinations in the TCP header. Consider on the other hand rendering the number of bytes sent by each source address. The number of values would make a pie chart hard to read. Secondly, our extracted information only contains one dimension, which is a strong requirement for a pie chart. The same can be said about the data type of the dimension to be visualized, which in this case would be categorical. Using a line chart would not be a good fit since the values are not continuous. What we have outlined in exemplary form is that each chart requires a dataset with appropriate data dimensions, number of values, compatible data type and, use case [24].

2.3 DDoS Classification Metrics

The first section of this chapter established that vectors, such as ICMP-, UDP-, TCP- and HTTP-based attacks, are the most relevant to focus on. The previous chapter introduced a process for security visualization where multiple properties need to be known to create a visualization. Therefore, it is vital to introduce several metrics on how to detect and classify an attack from network traffic.

Starting with the SYN-flood-based attacks, we can first derive that there is only one categorical dimension per time slice by which we can classify or visualize the attack pattern. To collect data about this attack we would simply measure the flags in the TCP packets header for some given time unit. Using all flags as features we know that in a SYN flood attack the number of packets that has the SYN flag set must be larger than the number of packets that has the FIN flag set. A simple metric could constitute the proportion of SYN flags compared to the proportion of FIN flags. For example, if 20% of all packets carry the SYN flag and 5% have the FIN flag set, then there would be four times as many SYN flags as FIN flags. For all values greater than one we can assume an SYN-flood attack [16]. A simple way to visualize such an attack would be to display the proportion of TCP flags over all packets. Alternatively, one could only show the proportion of SYN flags to FIN flags. A technique that could leverage this would be the use of animation or interaction, where a user would be allowed to disable the display of other flags [24]. In the approach proposed by [16], the TCP flags are taken as features and the anomaly of their distribution is computed for each time slice. The resulting visualization is plotted using two dimensions, time and anomaly, here expressed as the distance between the covariance matrix and mean matrix [16]. This is an effective way

of dealing with many flooding-based attacks. However, the threshold between differing covariance matrices is not defined [17].

There is no inherently normal distribution for other features of network traffic. For example, the number of inbound Ping requests observed for some host may be an attack or normal traffic behavior. [18] proposes a detection mechanism based on parallel processing of PCAP files. Here, multiple predefined thresholds need to be crossed for a packet to be flagged and the source host to be added to a list. For example, the number of hosts and ports a host may ping is previously established. Interestingly, for outbound packets, it is easy to detect whether a host is conducting a flooding attack using ICMP. For a given host it is usually easy to answer whether a given source address belongs to the network or not. Therefore, outbound ICMP packets with a spoofed source address may be part of an ICMP attack [18].

For HTTP-based approaches, we see similar techniques being used for the detection of an attack, such as pattern matching, clustering, and statistical methods. More relevant for this work are the features that are being selected to investigate attacks on the application-layer. Some approaches simply observe the distribution of service protocols used in incoming packets. Multiple algorithms focus on anomaly detection in user behavior, which is modeled in terms of time that passes between requests to the same web page. Other approaches combine features from the HTTP protocol, such as the page visited with the source port and the overall number of source IP addresses [20].

[19] proposed a detection process that does not make statements about specific attack vectors but rather uses 14 different features to create a robust statement about the presence of an attack. The result of the applied classification gives little information to visualize since its output is a simple ”yes” or ”no”. However, many features used internally are important since they focus on the presence of multivector attacks, something we have not considered so far. For example, ”same_srv_rate” indicates the percentage of hosts that send traffic to the same service. Under attack conditions, this metric would be very high.

Other techniques for detecting flooding attacks are based on the computation of a baseline entropy over all features in the network traffic. The distance of the observed data entropy to the baseline entropy is then used to signal the presence of an attack [17].

2.4 DDoSGrid

DDoSGrid is a prototypical implementation of a PCAP-based DDoS analysis and visualization system that was developed as a master project at the University of Zurich. The main objective was the development of a system that can extract relevant information from PCAP files for the analysis of DDoS attacks. Therefore, an important design goal was that both the feature extraction and the visual transformation scale well for large amounts of data. For example, imagine rendering all source IP addresses on the X-axis and their aggregated traffic bandwidth on the Y-axis of a scatter plot chart. This would be fine if the objective is to analyze computers in a home network. However, in a DDoS attack with a large number of hosts this visualization is hard to interpret. Similarly, if

all computations applied in the feature extraction step are not optimized for scalability, the solution is hardly applicable to this domain. For example, consider a situation where we want to analyze the average packet size of all packets. A computation where the size of each packet is stored in memory until the average is computed does not scale well. Therefore, the whole application has to be designed so that a feature extraction component requires only the current value and a new sample from which that value is updated, for example by updating the moving average. Therefore, important questions were if it is possible to extract, enrich, aggregate information so that DDoS attacks across all protocol layers can be visualized in a scalable manner with a high degree of automation.

The prototypical implementation was tested using three use cases. First, a researcher who aims to explore attacks. A second use case covered a similar explorative use case that targeted network operation. This stakeholder has similar objectives to a researcher. However, his perspective differs. For example, a network operator needs to generate a report about an attack and present it in a short time to an audience that is not technically knowledgable. Finally, a third use case covered a professor whose main objective is to teach about DDoS attacks. Therefore he/she wants to precisely communicate patterns and behaviors of attacks.

Internally, DDoSGrid is made up of three components. First, the *miner* module, which allows for the orchestration of feature extraction modules. The output of such a feature analysis can be made available through a JSON-based *web API*. Both modules have been implemented using JavaScript on the Nodejs platform. The client connecting to the web interface is a web-based *frontend* which mainly provides convenient access for the data miner and the visualizations.

This work was considered successful since the application provided several functional features that specifically targeted each use case. Limitations of that solution included missing integration with other tools, lack of visualization techniques and live visualizations [33].

2.5 DDoS Clearing House

The *DDoS Clearing House* (DDoSCH) is a platform made up of several software components with the objective of making it easier to share DDoS attack information. *DDoSDB* is the central component that also provides a web-based interface for the user to interact with. Through this interface, the user can search and explore DDoS attacks. The user can investigate an attack through features that are extracted from its packet capture. This also allows for structured filtering of attacks. For example, the user may search for attacks with a certain number of source IP addresses [28] [29].

The set of features that are extracted from each attack are called fingerprints. The component necessary for the extraction of such attacks is called *ddos_dissector*. The command-line application takes a packet capture or network flow as input and creates a JSON-based fingerprint that can reveal similarities based on a heuristic algorithm. Further, the output can be uploaded to the database provided by DDoSDB in an anonymized form. Finally,

the output can be visualized using a graph generation tool. At the time of writing, there was still active development around the DDoSDB component with the newest version being 3.0 [31].

Finally, the component that can further process the output of the *ddos_dissector* is a script called *converters*. This script transforms the insights from a fingerprint to a set of firewall rules which can be used to configure *iptables* [30].

Chapter 3

Related Work

This chapter presents the analysis of existing post-mortem DDoS analysis systems. First, a definition for that term is given which is then used to conduct a literature review in the next section. Finally, we summarize and contrast the elicited dimensions and derive conclusions about the current state of post-mortem DDoS analysis systems.

3.1 Taxonomy and Definitions

This work focuses on the improvement of a DDoS visualization and analysis system. The existing work is outlined in more detail in the next chapter, but it is important to consider the main use cases of that application. The main use cases were analysis and visualization of DDoS attack data from packet capture files with a target audience of academic teachers, researchers, and forensics personnel from the industry. We consider this application an example for the definition of a *Post-mortem* DDoS Analysis System which we now define. The first and maybe the most important categorization can be done regarding the time when it is applied. Mitigating an attack as early as possible is considered the most effective [21]. However, in our definition, we require that an analysis can be performed after the attack. This is required for the main purpose of detecting new patterns and trends which can then be used to deter future attacks. In that sense, the application requires information from past attacks to improve future attack defense. What follows from this is that information about an attack is stored for analysis. As mentioned in Section 2.2, this is not always given, since many network operators do not collect network traffic for performance reasons [24]. The taxonomy introduced by [21] further proposes a categorization based on the deployment location of the defense application. With our definition, we pose no requirement for the deployment location. Doing so would constrain the application to focus on network-level attacks or application-level attacks. Additionally, this would constrain possible data sources by the location from where they were collected, *i.e.*, the data could only be collected at the source or destination.

Finally, we define the purpose of such a system as one where a human can explore an attack and derive knowledge about it. This definition excludes automated applications

such as an Intrusion-Detection-Systems (IDS) that automatically blocks sources based on an algorithm. A similar taxonomy of DDoS visualization systems has been proposed by [2]. This taxonomy is based on a use-case classification and covers three use cases. First, visualizations that show the overall state of a network as a whole. Secondly, visualizations that describe the traffic flows in the network are considered. The final use case which covers visualizing characteristics of high-dimensional data to reveal patterns fits our definition best. Such a system transforms the attack data into an image that allows for attack type detection by a human.

To summarize our definition, a post-mortem analysis system needs to support processes for attack pattern analysis in a manner so that it can be performed after an attack has ended. It is important to note that this can include tools with different functions, such as databases, and visualization tools.

3.2 Overview

The previous section has introduced the definition of a post-mortem DDoS analysis system. Now, an overview of existing implementations and theoretic approaches that fit our definition is given. [23] have reviewed similar applications, thus we consider the same properties and extend them. An important objective from the DDoSGrid PoC implementation [34] was the idea to provide a platform based on which new visualizations, detection mechanisms, and data sources can be implemented. Therefore, additional attributes such as supported data sources and visualization techniques were investigated. Ideally, additional functional and qualitative requirements can be derived from this review so that an extended implementation can be targeted. Finally, the implementation state of the investigated approaches is of interest since we want to know whether the presented work is a proposed technique or a developed solution. We focus on the following properties and elicit them from existing literature wherever possible:

- An important property to elicit is the main **goal** and **approach** that the application tries to achieve. For example, although forensics and reporting tools would both need to be able to visualize an attack, they would still have wildly differing requirements, since a reporting tool needs to convey the hidden pattern in a shorter time [24].
- **Real-time** visualizations require two functional requirements to be satisfied. First, a continuous data source is present from which the features are continuously extracted. Second, the visualization needs to be able to transform the information in real time.
- **Interactivity** is a property of the visualization process where the visualization can be dynamically changed through user interaction. This is an important requirement for three-dimensional visualizations that are plotted in two-dimensional space. [24] states that only a few open source applications support this.
- A special case is when a tool focuses only on attack vectors related to DDoS attacks. In this case, we want to know if a DDoS **detection algorithm** is employed during

the visual transformation or feature extraction step. As outlined in the previous chapter, this can be used to color values which are supposed to raise the viewers' attention (*e.g.*, colorized malicious traffic).

- An important piece of information are the **data sources** that the application supports. A major reason why we are interested in this property is that we want to find out if a data source needs to be supported.

- Under visual transformation we try to investigate which **visualization techniques** are supported.

- As outlined before, we want to find out whether some approach was implemented or not. If an **implementation** does exist, we are interested in the **technology** used to implement it and the platform the tool can run on.

- Some platforms provide direct access to the extracted features and require the user to manually perform the visual transformation by plotting the features to a chart. Other applications are fully automated, meaning that the visualization is directly created for some given problem. Therefore, we are interested in the **degree of automation** being employed.

3.2.1 Analysis Tools

With the features we are interested in being defined, we investigate existing analysis tools. We narrow our scope to tools that fit the definition proposed in Section 3.1. However, we consider solutions proposed in academic papers as well as tools that are actively used in the industry.

Wireshark is an open-source packet analyzer that also supports the generation of PCAP files. PCAP files are an example of network capture files as described in Section 2.2. A large number of packets in an attack makes it hard to analyze the traffic that is displayed as a list of packets [11]. However, Wireshark also supports the visualization of selected traffic on a line chart using the "I/O Graph" component. The visual transformation for this has to be created manually by plotting some attribute of a packet to one of the line chart's two dimensions [32] [24]. Differently, **AfterGlow** uses CSV files [22]. AfterGlow is an open-source application that consists of multiple Perl scripts, *e.g.*, one to convert tcpdump output to a CSV file. Another script then takes that CSV file to create a link graph or a boxplot. Regarding visual transformation techniques, we find heavy usage of color-coding but no interactive elements since the output is a simple DOT file [24]. Since the tcpdump output is converted to CSV it seems that real-time visualizations are not supported. This tool is considered to be popular [11]. By manipulating the parameters of the script, we can see it being applied to various use cases.

GGobi [37] is a visualization tool that allows interactive visualization of CSV and XML input using techniques such as bar charts, parallel coordinates, scatter plots, time series and, link graphs. This general-purpose tool is considered powerful due to many visualization techniques being implemented. This also implies that it is not tailored to visualizing

DDoS attack from which we can assume that some manual interaction is required to created sensible visualizations [24]. GGobi is an open-source implementation which runs as a desktop application on all major operating systems.

The previously introduced tool AfterGlow relies on other tools such as GraphViz for the graph rendering. **Tulip** improves this procedure by providing a graphical interface. It takes DOT and GML files as input and provides an interactive visualization in both two- and three-dimensional space [24]. Since the visualization is mainly graph-based it is similar to other applications such as Gephi.

Rumint is similar to Wireshark since it also works on PCAP files. However, the visual transformation capabilities are stronger since it provides several visualizations such as parallel coordinates [11]. Interestingly, this application can capture traffic directly from the network interface which enables a real-time visualization. Interactive visualizations make it easy to interact with the data. The implementation targeting the Windows operating system is known to be limited in scalability since it does not work well with large PCAP files [24].

Similarly, **Etherape** plots connections from PCAP files using a link graph in an automated way. This tool provides some filtering algorithms to only display a limited number of hosts and connections [11]. Visual techniques, such as color coding and the display of statistics, make it a simple tool to understand network traffic [24].

So far, no applications employed a detection algorithm for specific types of attacks. In another way, **NetGrok** uses a detection algorithm provided by an IDS. The resulting insight is transformed by color-coding nodes in a link graph. This application scales especially well using the TreeMap visualization. Additionally, this application can process data from both PCAP files and real-time sources [11].

Similarly, **Time-based Network Visualizer** (TNV) can label malicious activity. Therefore, it must apply some form of a classification algorithm. It uses a two-dimensional visualization showing nodes and their related traffic. Complex visualizations can be created which makes it interesting to researchers and developers of security-related tools rather than for monitoring in real-time. This complexity stems from the interactivity that allows viewing an attack at different levels simultaneously [11].

Similarly, **TVis** is a visualization system providing detection for DDoS attacks. The Java-based implementation works on PCAP files and streams from which it samples traffic to extract features. This makes it similar to tools like Rumint and Wireshark, however, the fact that data is sampled may lead to loss of information. The extracted features are the number of incidents per host over time from which the infected period is computed. Finally, the nodes are visualized using a link graph, and metrics such as the number of unique IP addresses are plotted on a line chart. The same visualization technique is also applied to other metrics such as the computed number of packets per second or the number of unique TCP ports found in some time window [23].

FRuVATS strongly differs from all previously introduced tools in terms of goal, data source, visualization techniques, and implementation. This application makes heavy use of visualizations purely in three dimensions since it plots traffic against firewall rules. Source

and destination address are plotted to the X and Y-axis and the Z-axis represents the order of firewall rules. Color-coding is then used to indicate the associated firewall action. The objective of the system is to make it easier for administrators to create new firewall rules and to verify existing ones. The data processing of the segments was implemented in Java whereas the visualizations are implemented using web technologies such as JavaScript and WebGL. One of the biggest disadvantages is the missing automation when pre-processing the rulesets from firewalls [25].

Another application that specifically targets the visualization of network security is **NetVis**. A three-part process including data processing from a monitoring tool, visualization, and interaction lead to the integration of data from different sources. The Java-based implementation supports multiple two-dimensional visualizations which are grouped by granularity. First, the global view consists of a treemap where the user can explore the data using filters. A more detailed view allows the identification of individual hosts. Here, color coding is used to highlight relevant nodes and traffic [35]. This approach of supporting visualizations that have different abstraction levels to be analyzed at the same time was not used by any other tool we investigated so far.

NetViz is not a novel visualization application but rather a suite of existing software, introduced earlier in this section. The purpose of combining these solutions is to support security analysts. It allows gathering data using an IDS. A fraction of that data is then explored using Wireshark using the hints from the IDS to narrow the search. Finally, the platform includes tools such as Etherape and Netgrok to manually create visualizations. The platform is based on the Linux operating system and can be run in a virtualized machine [11].

On the other hand, **Netviewer** is a novel visualization for network traffic. Here, IP addresses are aggregated into a maximum number of 256 subnets and plotted on a virtual globe in 3D space. Lines then interconnect hosts to visualize traffic patterns. This aggregation allows visualizing all IPv4 addresses. What makes Netviewer a strong visualization is that it is highly interactive. For example, when there are too many items visualized only a subset can be selected for visualization. Similarly, the view can be transformed to look at the visualization from a different perspective. Automated detection is not part of the implementation. Details about the implementation of the data sources used are not available [26].

A tool that uses completely different data sources than any of the previously mentioned ones is **NVisionIP**. This application visualizes network traffic using various techniques such as color mappings, treemaps, and tables. Unfortunately, it is constrained to hosts from a class B network which makes it of limited use for DDoS attacks under current attack conditions [26] [27]. NVisionIP uses Netflow as a data source which allows knowledge from an IDS to be visualized. For example, heavy traffic flows are indicated using a color-coding scheme. The Java implementation of NVisionIP does not support real-time visualizations and analysis [27].

glTail provides animated visualizations and is written in Ruby. The tool first requires manual adaption of a configuration file which it then uses to automatically generate visualizations from a set of databases [38] [24]. All previously defined dimensions are drawn horizontally. Traffic is then visualized by circles flowing horizontally from the source [24].

Visualizations are drawn in real-time which mainly helps to establish a global view on traffic volume and frequency. Given that even for simple visualizations the visualization is considered hard to quantify, we assume its expressiveness may suffer when dealing with highly voluminous traffic.

As its name suggests, **DDoSDB** is an application whose focus is to enable different stakeholders to share DDoS attack information. Such stakeholders may include victims, researchers, and the security industry. The purpose of sharing such information is the stimulus of post-mortem analysis of DDoS attacks [28]. The python-based web application makes it easy to find an attack by searching and filtering extracted information called fingerprints [28] [29]. Internally, multiple scripts can be used to anonymize and upload the data. Here, PCAP files are used as the main data source. Further, one may convert a fingerprint to a set of firewall rules [30]. The search engine provides convenient access to the information in the fingerprints, however, there is little visual support to analyze attacks. However, it is possible to visualize comparison of the source addresses of two or more attacks. The proportion of overlapping source addresses are shown in a table. More information on the components used in this project has been given in Section 2.5.

DDoSGrid is an approach and prototypical implementation of a DDoS visualization platform. The objectives and outcomes of the first project on DDoSGrid have been outlined in Section 2.4. DDoSGrid provides easy access to the complete information of PCAP files for the creation of visualizations. Since packet captures are used as a data source a lot of information can be extracted, however real-time visualizations are not supported. Visualizations for extracted features are grouped by attack type and graph type. Several graphs are available such as Pie Charts, Bar Charts, Scatter Plots, and world maps. Many of the visualizations are interactive. Visualizations can be opened in parallel and freely arranged to allow users to compare between graphs and datasets. The resulting dashboard can be saved to bring it up later. Except for the dashboard configuration, all visualizations are completely automated. Target audiences of the platform include researchers, academic teachers, and the security community. Similar to DDoSDB, the visualization component of the platform is based on web technologies. The feature extraction component and interfaces thereto are written in Node.js [34] [33].

3.3 Summary and Comparison

Having introduced numerous tools that can be used for post-mortem analysis of DDoS attacks we see that these tools differ in several attributes. Since this work focuses on the improvement of the prototypical implementation and approach used within the DDoSGrid project we use this application as a reference and summarise the remaining applications. This allows us to establish a baseline from which we can elicit possible improvements that are used for designing the improved version of DDoSGrid.

Table 3.1 gives an overview of the reviewed applications based on their purpose, visual techniques, degree of automation in information extraction, and visual transformation. Here, it is also considered if these visualizations are interactive. Further, we consider the data sources that are being used and we try to discriminate wherever possible if the

analysis is completely offline or if the information is extracted and visualized continuously. Finally, we consider implementation details such as the platform being used.

Many of the applications are focused on applying visualizations to show network traffic flows. This may explain why link graphs are frequently implemented. Only a few of the tools specifically focus on DDoS attacks. It remains questionable if the visualizations scale well enough so that traffic flows under attack conditions can be correctly analyzed. Some applications, such as FRuVATS are considered to have limited expressiveness under high volume. Tools that specifically focus on network security-related aspects and DDoS attacks often use a combination of visualizations and interactive elements.

Having different visualizations is a critical concept especially when these focus on different metrics and at different scales. For example, NetVis provides both Link graphs and Treemaps so that one can use the latter to explore traffic on a global view and then apply the gained insight into a filter in a more detailed link graph.

Considering tools that are for general-purpose analysis, it is often the case that these require manual interaction for either the feature extraction or the visual transformation of those features. For example, Wiresharks' I/O Graph can be used to plot the distribution of TCP flags so that one can visualize an SYN-flood attack. However, this requires advanced knowledge about attacks and the protocols that are being targeted. Also, the fact that this has to be applied manually creates the possibility for human errors. Another example of a partially automated solution is DDoSDB where a set of scripts are used to upload and share DDoS attack information. However, the orchestration of these tools has to be done on the command-line.

DDoSDB is unique considering its purpose, it focuses solely on the distribution of attack data. For this purpose, it provides strong features such as fingerprinting of attacks and a capable search engine. However, this application is not integrated with other solutions and it only provides a single visualization to compare source addresses across attacks.

Considering data sources we can see a large number of tools to use PCAP files. Tools that provide real-time analysis often directly access the network interface or use another system such as an IDS or a Netflow collector. Considering that the focus in this review is on post-mortem analysis we do not investigate the requirement for real-time visualizations. The last group of tools uses other data types such as CSV and XML files. Such data sources are easier to analyze but they do not contain the same information as a packet capture. It seems as if PCAP files are an important data source since these are widely used, standardized and contain nearly all information known to the host. Also, we can consider that distributing them is easy for offline analysis.

Finally, we look at the technologies used for implementing these tools. Many applications are written as a desktop application or less frequent as command-line tools. The exception here is DDoSDB which is written as a web application since its purpose is to make information available to a large audience. This choice provides multiple benefits that fit the purpose of sharing attack data. Web applications are considered to be supported by most operating systems and they allow multiple people to access the same application without having to install it [36].

Concluding our analysis on existing post-mortem analysis tools by assessing the feature scope and implementation of DDoSGrid we notice a number of differences. First, DDoSGrid is among only a few that analyze attack data specifically for DDoS attacks. Some of the general-purpose tools might not scale well if input data is given that represents attack conditions. DDoSDB is an exception since its purpose is to share actual attack data. This feature is not present in DDoSGrid since the input data can not be retrieved later on and there are no search capabilities in place.

Next, we can see that many tools require some form of manual interaction. This is also the case with DDoSDB where collaborators have to manually orchestrate a set of scripts. An automated solution could hide these technical details from the user and only require him/her to upload a PCAP file.

Many tools provide interactive visualizations with link graphs, box plots, bar charts, parallel coordinates, line charts, and scatter plots being frequently implemented. DDoSGrid implements many of them. However, link graphs and parallel coordinates, which are currently missing, would allow the analysis of traffic flows or replaying an attack. DDoSDB provides insight through their fingerprinting functionality. However, visualizations are not yet used extensively to visualize such information, which could help when discovering the shared attack data.

DDoSGrid and DDoSDB use PCAP files as an input. This seems sufficient for post-mortem analysis since data sources like Netflow would only provide little additional data. Both applications are web-based which would allow for a web-based integration.

The integration of the two tools DDoSGrid and DDoSDB appears to provide a powerful general-purpose, automated, and web-based platform for discovery, analysis, and distribution of DDoS attack data based on PCAP files. Another strong improvement would be the addition of new visualizations.

Table 3.1: Overview over elicited applications

	Application	**Visualization**	**Automation**	**Data Sources**	**Implementation**
Wireshark	Traffic capturing, Deep inspection	Line chart, Bar chart (non-interactive)	Manual analysis and visual transformation	Network interfaces, PCAP (realtime & offline)	Desktop application (C, C++)
AfterGlow	Link Graph generation, Traffic flow analysis	Link Graph, Box Plot (non-interactive)	Manual configuration & generation	CSV file, PCAP conversion possible (offline)	Command-line application (Perl)

GGobi	General purpose network analysis	Bar charts, Parallel coordinates, Scatter plots, Time series, Link graphs (Interactive)	Manual	CSV, XML, R	Desktop application (C, GTk)
Tulip	Graph generation	Link graph (Interactive)	Manual generation through GUI	DOT, GML (offline)	Desktop application (C++)
glTail	Analyzing traffic volume	Custom 2D chart (Non-Interactive)	Manual	Databases	Desktop application (Ruby)
Rumint	Replay & Analyze Network traffic over time	Parallel coordinates, character distribution, time series (Interactive)	Automated	PCAP (offline)	Desktop application (Windows only)
Etherape	Traffic flow analysis	Link graph (Non-Interactive)	Automated	PCAP (offline)	Desktop Application (C, GTk)
NetGrok	Traffic Flow analysis, IDS visualization	Treemap, Link graph	Automated, Detection algorithm	IDS, PCAP (real-time, offline)	Desktop application (Java)
TNV	Traffic Flow analysis	Time-oriented View, multi-level views (Interactive)	Automated	PCAP (offline)	Desktop application (Java)
TVis	DDoS attack detection	Link Graph, Metrics (Interactive)	Automated	PCAP, network interface (offline, real-time)	Desktop application (Java)
FRuVATS	Visualizing firewall rules	Custom cubical chart	Manual preprocessing	Firewall rules (offline)	Java application (data processing) & Web application (visualization)
NetVis	Integrated Analysis, Exploring Network Security	Multi-level visualizations, Treemaps, Link graph (Interactive)	Automated	Snort (real-time)	Desktop application (Java)

NetViz	Access suite of network analysis tools	NetGrok, Etherape	Manual	PCAP (offline)	Linux-based, Virtualized
Netviewer	Traffic flow analysis	3D globe, aggregated addresses (Interactive)	Manual	Unknown	Unknown
NVisionIP	Traffic flow analysis	Treemaps (interactive), scatterplots	Automated	Netflow (offline)	Desktop application (Java)
DDoSDB	Sharing & finding attack data, fingerprinting	Tabular comparison (Non-Interactive)	Partially automated	PCAP, Netflow (offline)	Web application & Command-line (Python)
DDoSGrid	Fully automated platform for DDoS attack visualization & feature extraction	Bar charts, Line charts, Scatter plots, World Maps, Metrics (Interactive)	Automated	PCAP (offline)	Web application & Command-line

Chapter 4

Analysis and Requirements

The previous chapter has given an overview of existing DDoS analysis tools. From that summary, we have been able to understand desirable traits of such an application. In this chapter, we try to make the objectives and requirements of the improved system more explicit. First, we transform the learnings from the literature review into high-level requirements. Next, we investigate the approaches from the *DDoSGrid* and *DDoSCH* in the necessary detail. Finally, we propose an architecture that synthesizes both approaches to meet the requirements.

4.1 Requirements

The most important requirement is the integration of the *DDoSCH* components. This creates a solution that supports the objectives of sharing, exploring, and visualizing attack data in a fully integrated manner. Ideally, this requires that components which are not yet automated are being integrated into an automated process.

Considering data sources we saw that PCAP files are widely used. Therefore, the solution should be able to perform all functions based on the information available in PCAP files. Based on feedback from practitioners from the industry, we found that Netflow is frequently used. Optionally, one could argue that such files can be converted [39]. Considering real-time visualizations we find that tools that support this are often related to network operation and less with post-mortem analysis. An interesting addition, however, would be the replay functionality of certain visualizations which reveals the attack pattern over time.

Certain components from the *DDoSCH* extract specific information about attacks. This can be used as input to the visualization capabilities provided by DDoSGrid. For example, through its *converters* module, *DDoSCH* actively detects malicious traffic and produces a set of firewall rules. These could be visualized in multiple ways. First, one could convert these rules into a format where we could filter traffic during visualization which allows the visualization of pruned traffic. Alternatively, a visualization that focuses on firewall rules, as introduced in the previous chapter, could be used.

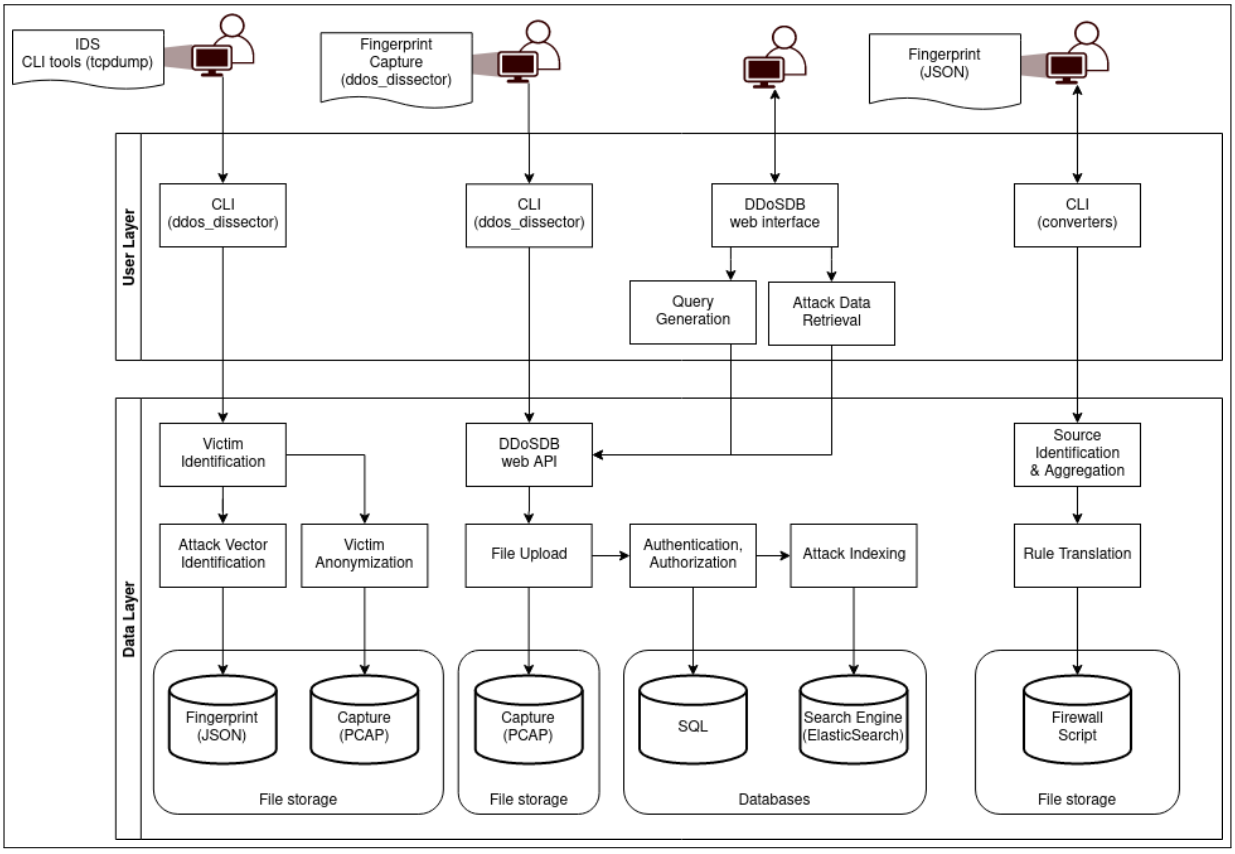


Figure 4.1: Architecture overview of the *DDoSCH* components

4.2 Architectural Analysis

The requirements that were outlined in the previous section can be derived from the literature review. The following subsections further explore the existing functionality of the two projects. By investigating the strengths and weaknesses of both architectures, we derive additional requirements.

4.2.1 DDoSCH

As outlined in Section 2.5, *DDoSCH* provides several components that work together. As we can see in this analysis, these tools can be applied in series and are powerful in themselves, however, they are not integrated into one platform. In Figure 4.1 we can see that there are four major use cases that the *DDoSCH* components support. Each of these interactions is denoted by a human operator that performs some operation with some input for which he/she receives some output.

The first use case deals with the generation of a fingerprint. For this, the operator holds a network capture file like a PCAP file which he/she provides as input. Relevant features about the attack are extracted by the *ddos_dissector* script. This includes attack vector, victim, and attacker. These features make up the pattern of the attack and are then presented as output in the form of a JSON file. The information on the victim is used to

anonymize the input file. By removing information that reveals the victims identity, the packet capture file is pruned and given as a second output.

In a second use case, the anonymized capture file and the JSON fingerprint can be taken as input by uploading them to the web interface provided by the *DDoSDB* component. For that, one uses the *ddos_dissector* component on the command-line. One important thing to note here is that this is the only endpoint of that web interface that does not render a response in HTML format. Another exception here is that permissions of a given user can be queried. That endpoint is used before any file is uploaded. This reveals a strength since *DDoSDB* provides an access model with fine-grained control over users and their roles. For example, some users may later download an attack capture but not upload new capture files. A disadvantageous implementation detail is that some of the information provided by *DDoSDB* can not be accessed through a stable interface, since the HTML pages may change over time. The integration of additional applications using the provided session-based authentication mechanism is also not ideal. We assume that for this very reason the two endpoints used by the script use a stateless authentication mechanism. Integrating other applications requires to abstract some of the endpoints and to extend the stateless authentication capabilities to these endpoints. Additionally, one may consider using the provided access model as a central authorization entity.

Once both fingerprint and capture files are uploaded to DDoSDB, they are stored and indexed. For the fingerprint a search engine is employed and for packet capture files a simple file-storage-based persistence layer is used. The user then accesses *DDoSDB* through a graphical web interface provided by DDoSDB. Here, he/she can search, compare, explore, discuss, and eventually export attack data. Given appropriate permissions, the data model can be administrated through this interface as well. For example, an administrator can extend users privileges so that this other user may upload fingerprints. This graphical interface is implemented as a server-side rendered web application, which simplifies many aspects. For example, a user may compare source IP addresses of two attacks. Such a comparison can easily be computed and rendered on a server. However, this may limit interactivity in the web application, since the presentation of that visualization is not dynamic.

Finally, if the user finds an attack he/she is interested in, that attack can be exported by downloading its fingerprint or capture file. The fingerprint can then be fed into the *converters* module, another command-line based component. This module iterates through all detected attack vectors and attackers. This results in a set of source IP addresses which can then be aggregated into prefixes. Having these prefixes at hand, the script converts the set into *iptables* rules. These rules are then stored as a script on the file system.

Considering the integration of the system we find many powerful components that execute a specific task well. However, there is no automated system that wraps all the components. For example, the *DDoSDB* web interface could provide a file upload form and internally publish the output of the *ddos_dissector* script. Similarly, the mitigation rules could be computed and made available through the same mechanism as the fingerprints. For DDoSDB, there is a lack of interfaces that can be used by other systems. To create an integrated solution these interfaces would first have to be developed. The same applies to the authorization system used by DDoSDB.

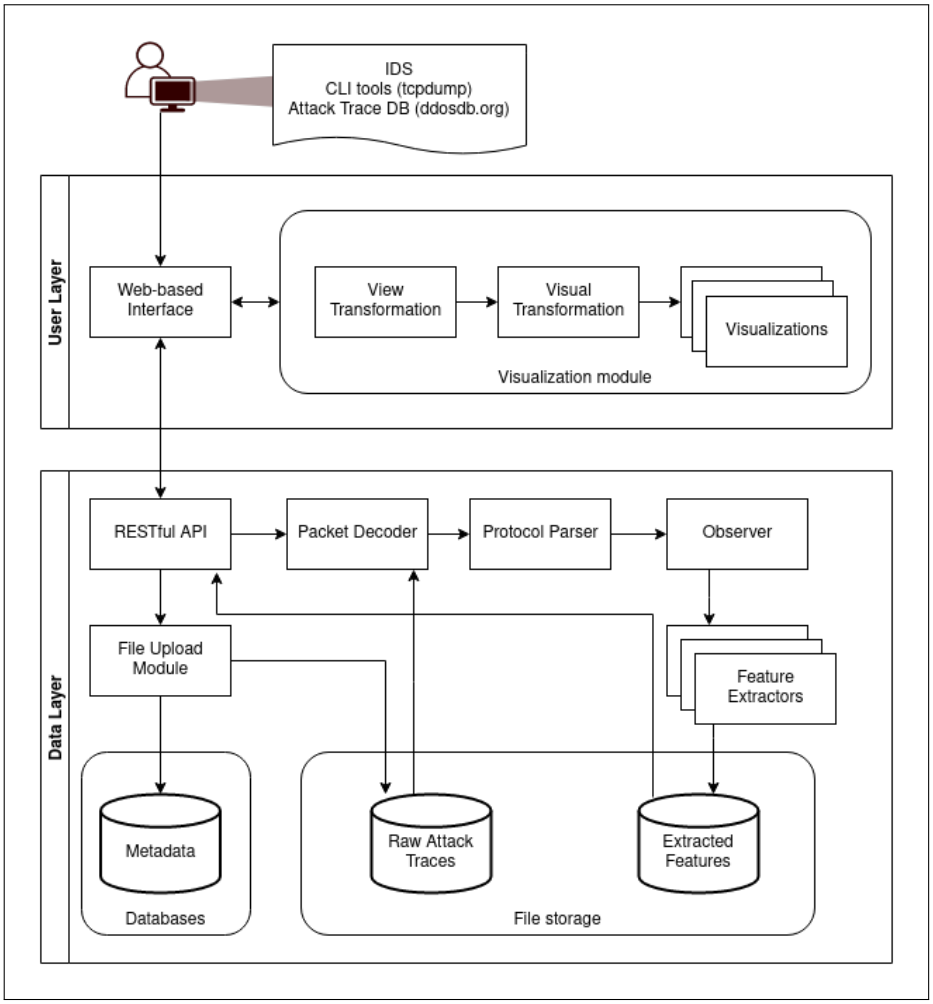


Figure 4.2: Architecture overview of the *DDoSGrid* components

4.2.2 DDoSDB

Figure 4.2 draws an overview of the components in *DDoSGrid* in similar manner as Figure 4.1 in the previous chapter. The biggest difference we can see with this approach is that all interaction between the user and the system can be performed through the graphical web interface. For example, when analyzing a new PCAP file, the user may upload that through a view in the web application. From there, it is stored in the file system and then processed. This processing involves a packet decoder- and parser and manifold software modules that extract and format relevant features. These extraction modules produce output in JSON format which then again is stored on the file system. The web application on the other hand handles all user-related functions, including the visual transformation procedures. This requires a highly interactive client-side application, which is a strength of that interface. It is important to note, however, that all components used in the data layer can be used on the command-line as well.

When a new file is uploaded through the API, only the metadata such as the filename or its description, is persisted in a database. Compared to DDoSDB, there is no authentication and authorization procedure in place. DDoSDB, however, provides a powerful role-based

user model. Hence, it is desirable to integrate it in a way where the functionality provided in *DDoSDB* is leveraged. Therefore, a user only needs one account, and administrators have a single place to modify the user model.

Other improvements that require a high degree of interaction with the user seem more fitting to be implemented in the web client in DDoSGrid. For example, *DDoSGrid* already provides a file upload view along with status information. It would be beneficial to use this view as a facade and distribute the input file to the *DDoSDB* as well. That would provide a suitable user experience while hiding technical details. Given that it is a rich JavaScript-based client this application could further leverage other APIs. For example, it would be possible to directly associate the *DDoSGrid* web application to PCAP files.

4.3 Synthesis

Having summarized the learnings from the literature review as well as the two specific approaches from the review, we now summarize the requirements for the integrated system. The most important requirement is that both applications can work with the same input data. This data has to be consistently available in both applications without the user having to upload it multiple times. As just described, we strive for a high degree of automation. From this we conclude that technical details shall be hidden from the user as much as possible. A graphical user interface needs to be developed to enable this task.

Once these preliminary requirements are met, new ways of visualizing information extracted by the *DDoSCH* components can be leveraged. However, *DDoSGrid* already visualizes many of the features in a fingerprint, for example the source IP addresses. The result of other components such as the *converters* could be visualized. For example, it would be possible to extract the attack part of a packet capture file and allow the user to switch between different views in the visualization dashboard.

The next subsections define the requirements in more detail, based on an initial classification. We consider three categories for that classification. **Usability** can be explained with the ease of use of the application [45]. For example, if the interfaces between both platforms are compatible, it is possible to manually load the same piece of data into both applications. However, this use case is simplified when that import happens automatically.

Privacy on the other hand can be defined as the degree of freedom from unauthorized intrusion [46]. During our analysis of the *DDoSDB* system, we saw that privacy is an important aspect in this context, since we are dealing with sensitive information. From the two previous categories, we derive **technical** requirements. These may not be directly noticed by an end-user of the system, but these requirements are important towards the usefulness of the platform.

4.3.1 Usability requirements

When using either of both applications, it has to be easy for the user to traverse between both applications. For example, after locating a dataset in *DDoSDB* it must be possible

to easily open that on a dashboard in *DDoSGrid*. In the same way, one can go back to *DDoSDB* to explore an attack focused on *DDoSGrid*. Having both approaches combined presents a powerful visualization since one can explore an attack at different abstraction levels. For that, *DDoSDB* would allow to investigate, explore, and compare different attacks on strongly abstracted levels through its notion of fingerprints. *DDoSGrid* allows the same operations but on a much lower detail. This approach of supporting different perspectives at different levels is known to be effective for volumetric attacks, as we have seen in our literature review.

From a qualitative perspective, this function must be implemented in a way that technical details are hidden from the user resulting in a fully integrated solution. The user should not need to know about specifics such as the internals of data types or interfaces. In our literature review, we have seen the importance of applying color-coding mechanisms during the visual transformation procedure. This allows highlighting important information. One required application of that technique is for *DDoSGrid* to integrate the learnings from the attack vector identification into the visual transformation procedures. For example, if the attackers are known, their source IP addresses can be used for color coding related traffic in a visualization.

DDoSDB currently only publishes the attack traces that were used as input and the fingerprints resulting from the analysis. *DDoSDB* should also make the mitigation rules available in the same way as it publishes the capture file and fingerprint. With that in place, using all three components is simplified since *DDoSDB* makes them accessible through a unified interface. Since *DDoSDB* is the heart of the *DDoSCH* project we have only described efforts towards integrating that application. However, once such integration is established, *DDoSGrid* can be used to visualize information extracted by the remaining modules as well. Therefore, *DDoSGrid* should be able to visualize information extracted by the *converters* module. The visualizations should enable the identification of attack traffic as well as the simulation of the pruned traffic. In that sense, it shall either visualize the blocked or accepted traffic depending on the selected mode.

4.3.2 Privacy requirements

As our analysis of the *DDoSDB* platform revealed, it is vital to provide some degree of anonymity to the victims who share their attack information. After all, network traces may contain sensitive information. To encourage victims to continuously share their attack traces to the integrated platform, *DDoSGrid* must provide the same level of privacy as *DDoSDB*. Therefore, *DDoSGrid* must integrate the anonymization pre-processing step into its feature extraction module and access must be authorized.

4.3.3 Technical requirements

Integrating *DDoSDB* and *DDoSGrid* can only be achieved when imported data is automatically published to both applications. For this, it is necessary to provide a single entry point for the publishing of attack data. To achieve the usability requirement that

DDoSGrid visualizes the information of the *converters* module multiple technical requirements need to be fulfilled. *DDoSGrid* should be able to understand the output of the *converters* module and visualize the information. This also requires that the graphical interface allows the user to switch between views that provide the described filtration.

For the integration of both applications with the previously described goal of improving usability and privacy, new interfaces are required. Although these interfaces should not be visible to the end-user, the development of these interfaces makes the overall platform accessible for future works. In that sense, *DDoSDB* provides the central resource server with *DDoSGrid* being the first third-party application accessing its resources.

Similarly, it must not be possible to use the *DDoSGrid* platform to gain elevated access to *DDoSDB*. Therefore, *DDoSGrid* must implement the authentication and authorization procedures used in *DDoSDB* using the same user model. If necessary, additional roles and permissions are integrated into *DDoSDB*. These rules can then be used to provide fine-grained access to the functionalities provided through *DDoSGrid*.

Chapter 5

Prototype and Implementation

The previous chapter has introduced several requirements that the solution being developed has to satisfy. In this chapter, details on the DDoSGrid prototypical implementation and its extensions of DDoSGrid are provided. An iterative approach was followed for the platform's improvements, in which each iteration focuses on one dimension of the application that is being improved. The different iterations are focusing on: *(a)* the definition of the DDoSGrid models and interfaces, *(b)* the automation of the DDoSCH life cycle, and *(c)* the improvements on usability. Each one of these dimensions is covered along with this chapter.

The first section documents the improvements made to DDoSGrid and DDoSDB in the dimension of models and interfaces. The objective of these improvements was the overall compatibility on both a conceptual and technical level. Examples of these two types of improvements are the addition of an authentication system, respectively, the alignment of a hashing algorithm used in DDoSDB. With the documented changes it is possible to assume a strong compatibility level between both platforms so that data can be easily exchanged and new functionality can be built on top. For example, the next section aims at improving the automation of the analysis process across both systems. The feasibility of that objective is demonstrated by concluding this iteration with an integrated import process.

5.1 Models and Interfaces

A critical point when developing this platform was that it consists of the various components of both projects which can interact. This interaction may still need to be carried out manually. However, all models and interfaces have to be implemented in a way where they do not resemble separate applications. In the rest of this section, we prioritize the requirements towards this dimension of the solution. Using these requirements, we conduct a technical review of both applications. With that insight, we propose several improvements and document how these improvements were implemented.

5.1.1 Requirements Prioritization

To establish a viable development path, the requirements introduced in Chapter 4.3 are now subject to an initial prioritization. The goal of this prioritization is to ensure that the solution can be developed in the given time frame in a way that it becomes useful as soon as possible. That baseline is then used to build on top and provide valuable functionality. Therefore, we define all requirements as a "must-have" if another requirement depends on it. For example, it is not possible to achieve a fully automated process without first establishing the necessary interfaces. This means, that for our first iteration, we try to achieve a platform where both solutions have the necessary interfaces and where the data models are integrated. This goal is considered the most important of this work, because this interoperability is required for all other requirements. Having established these interfaces and data models, we can then automate the process in the next iteration of the development.

This would apply to multiple requirements defined in Chapter 4.3. Thus, an integrated authentication and authorization mechanism should be implemented first. This requires the development of new interfaces. Finally, this integration should be concluded by an integrated import and export feature that distributes input files to both applications using the newly created interfaces.

5.1.2 Analysis and Design

The components of the DDoSCH project were described on an abstraction level that only considered the functional components and their interfaces. For this iteration of the development process, it was required to investigate the internals of the DDoSCH components. The component with the most active development at the time of writing was the DDoSDB component. DDoSDB is a Python application written using Django, a web framework [40]. This web application consists of three parts: The templating system, business logic, and a storage layer. This storage layer is implemented outside of DDoSDB with an SQL-based database and an Elasticsearch search-engine [29]. The remaining components either access the API provided by DDoSDB or work directly on fingerprints. Therefore, we do not need to investigate them in more detail since they have very simple dependencies towards the existing interfaces.

The implementation of DDoSGrid was documented quite extensively in a previous work [33], up to an abstraction level where the reader understands how to implement additional feature extraction modules. Therefore, we only summarize the implementation of the components as it is relevant for the subject of this section, which deals with the integration of both projects. The DDoSGrid platform consists of three modules, all implemented using JavaScript. The *miner* module reads PCAP files and orchestrates numerous feature extraction modules. Once the PCAP is parsed by all extraction modules, it signals completion through the stdout channel or an inter-process channel. This allows the miner module to be run directly through the command-line or as a sub-process of another project. In the default deployment variant, that parent process would be the *api* module. This is a RESTful HTTP API written in the Express.js web framework. It uses a simple file-based

database to store metadata, such as file names and their descriptions. Considering the integration of both tools it is important to note that the prototypical implementation of DDoSGrid does not provide authentication based on a user model. DDoSDB, on the other hand, provides such a user model as well as an administration view to handle access requests. The third component is the *frontend*, which is a Single-Page-Application written using the Vue.js framework.

Considering the backend modules of both systems, the DDoSDB does not provide an API that is decoupled from view logic. This makes it difficult to integrate third-party applications. On the other hand, DDoSDB provides features such as the user model that have to be leveraged by DDoSGrid. In order to use both tools as a platform we propose the following improvements to the existing implementations:

1. DDoSDB receives RESTful API endpoints that provide similar functionality to the existing pages in the Django app. With this, we decouple the business logic provided by DDoSDB from its frontend. This allows third-party apps to consume resources on the server.

2. We implement an OAuth2 provider into the DDoSDB web application. By doing so, DDoSGrid acts as a client and access resources on DDoSDB. Using the OAuth2 terminology, DDoSDB then acts as both the resource- and authorization server.

3. The data model on datasets needs to be made compatible. For example, DDoSDB uses MD5 hashes of the input datasets while DDoSGrid uses SHA256. We need to extend the data model in a way that makes it backward compatible and compatible with both solutions. For example, it should be possible for DDoSDB to request that a dataset is analyzed.

4. Changes in the UI to reflect the achieved integration conclude the implementation. With this, it would be possible to explore datasets in DDoSDB and then request analysis in DDoSGrid with a simple button.

5.1.3 Implementation

The first thing that was aligned were the deployment strategies of both platforms. DDoSDB provides a convenient way of setting up a local instance of DDoSDB with a virtual machine [44]. This is fine for quick deployment scenarios, however, for our development process in conjunction with the DDoSGrid services this was not simple enough. Since all the DDoSGrid components and a few of the DDoSDB dependencies were already ported to the Docker environment, we first ensured that all services can be run as a Docker stack.

Once both platforms could be easily deployed to any Linux-based system, we started to implement the necessary interfaces. First, we focused on DDoSDB, where we added additional views, endpoints, and models. For that, no new dependencies had to be included and no modifications to the existing interfaces were made. All newly created web interfaces return JSON as a response. This led to several components in the controllers of the Django app with redundant functionality. For example, some interfaces exposed the same

functionality with the main difference of returning JSON instead of a rendered template. To keep a clean codebase, some of the functionality used by both types of interfaces was factorized into methods.

Next, an OAuth2 provider was implemented by including a set of routes, controllers, and models provided by the Django OAuth Toolkit. This toolkit allows for the implementation of an OAuth2 authorization and resource server compliant with RFC 5749 [48]. This module was selected since it integrates with the existing user model within DDoSDB. This allows both authorization systems to work on the same user model and that the administration can still be performed using the Django admin page. Now, both authorization systems can be used to protect the web interface; the existing session-based scheme and the newly created token-based authentication. The token-based authentication scheme was then used to protect the endpoints described in the previous paragraph. Compared to the existing DDoSDB configuration where both backend and frontend are served from the same origin, the newly created interfaces are now subject to cross-origin request sharing. Therefore, HTTP headers needed to be set correctly, which was achieved by using the django-cors-middleware package [47].

Now that a fully functional OAuth2 provider was implemented, we started to work on the authentication endpoints in DDoSGrid. We used a session-based authentication scheme with the OAuth2 server from DDoSDB for authorization purposes. This is a common authentication and authorization mechanism, therefore it is not described in detail. To summarize, an URL on the DDoSGrid web interface redirects the browser to the authentication page of the OAuth2 server. For that, it also passes client identification, a secret, and the URL where the user shall be redirected upon successful authorization. All of these parameters need to be registered within DDoSDB first. If the authorization succeeds the redirect URL which points to another endpoint in DDoSGrid contains credentials to obtain an access token from DDoSDB.

This token is then used to query additional information about the user with a new method in the DDoSDB controller. With both the user information and the access token available, we can store or update the users' record that is now authorized in our database. Therefore, a new database using the existing NeDB provider was created. We store information such as the token, username, e-mail address, first- and last name, and then provide a session cookie to the user which initiated the authentication flow. Now, we need to apply two things to the existing codebase. First, all data models need to be extended so that they are personalized towards a given user. For example, the entity that represents an uploaded dataset needs to be extended so that it has a reference to a user ID. This user ID then identifies the uploader of the dataset. After that, all the endpoints in the web API need to check if the request is authenticated. If that is the case, the business logic needs to be adapted so that information privacy between users is preserved. For example, when retrieving a list of available datasets only the datasets where the user is set as an uploader should be returned.

With the web API being fully protected against unauthenticated access, the frontend had to be adapted accordingly. All HTTP requests had to be supplemented with the relevant credentials and additional components for authentication handling were required. These components need to determine the authentication state of the application and prompt

the user with appropriate information. If the user is not authenticated, all actions in the frontend requiring authentication need to be intercepted. For example, when a user enters a page such as the *datasets page* he/she needs to be redirected to the *landing page* where he/she is prompted with information on how to authenticate. Additionally, the action that was intercepted needs to be stored in session storage. Once authentication has succeeded, the user can be redirected in a way that the initial action can be performed. For example, if a user accesses the page to import a dataset from DDoSDB, this action is prevented because he/she is not authenticated. That action and its parameter, such as the dataset, are stored and the user is prompted with information on how to log in. If the user performs the OAuth2 authentication flow, he/she is eventually redirected back to the frontend. Here, the frontend retrieves the initial action, the requested import of a dataset, and proceeds with it.

At this point, the user's data model is aligned between DDoSDB and DDoSGrid. However, further analysis of data models revealed that there is another incompatibility preventing the integration of both platforms. DDoSDB uses an MD5 hash derived from a fingerprint as the main key to identify datasets. DDoSGrid, on the other hand, uses a SHA256 hash computed on the initial input PCAP file as an identifier. Since the SHA256 has a larger digest size, we use that algorithm as a reference and align the models in DDoSDB accordingly. It is important to note that this is the first change that breaks backward compatibility in DDoSDB.

To summarize, so far we have implemented an OAuth2 provider and aligned the interfaces and data models so that all components are compatible and protected. Based on this, it is now possible to implement the final requirement of this iteration, which considers importing datasets from DDoSDB to DDoSGrid. First, we add a new endpoint to the web interface of DDoSDB. Naturally, this endpoint must be authenticated using the token-based OAuth2 server. So far, a similar endpoint allows authentication using username and password supplied as a HTTP header. The business logic of that endpoint is extracted into a new method which only considers the actual delivery of the PCAP file. Therefore, the interface now provides two ways to access the PCAP files.

With that interface in place, DDoSGrid accepts import requests using a new endpoint in the web API. After a successful authentication, the user's authentication token can be used to fetch the PCAP file from DDoSDB. It is important to send the actual user's token so that the role-based authorization system of DDoSDB is respected. Having implemented these interfaces and processes, it is possible to use the API of DDoSGrid to import and analyze a file directly from the DDoSB.

5.2 Automation of DDoSCH Life Cycle

After having improved both platforms in a way where both platforms provide openly accessible models and interfaces, we can leverage these improvements to automate the analysis process. In this iteration, we target a fully integrated and automated analysis process. This process contains pre-processing using the *ddos_dissector*, fingerprint export, filter generation, and export, as well as the unchanged analysis provided by DDoSGrid.

Newly created models and interfaces prove the feasibility of this iteration described in Section 5.1. To hide technical details from the user, the frontend was extended to allow importing from DDoSDB or export thereto with minimal user interaction.

5.2.1 Requirements Prioritization

In the previous iteration, requirements considered as dependencies of other requirements were identified as critical. For example, the development of new interfaces was considered to be critical to implement additional features. In this iteration, we prioritize requirements that improve the analysis process. First, the pre-processing procedure provided in DDoSCH has to be integrated into the analysis process of DDoSGrid. As this module provides two files as output, it is required that both files are exported to DDoSDB. This allows that the process is fully integrated when the user uploads a PCAP file to DDoSGrid. Thereby, we can improve the system's usability since the users do not need to install the ddos_dissector scripts locally. This hides technical details from the user. However, as outlined in the previous iteration, DDoSGrid should still behave like an add-on to DDoSDB. This means that DDoSDB users can continue to publish datasets directly to that tool. If a user wants to analyze the file using DDoSGrid he/she should be able to do so in a later step. Therefore, it must be possible that files which are unavailable in DDoSGrid can be imported directly from DDoSDB without having to download and upload the file.

So far, we have described the requirements to integrate the *ddos_dissector* and DDoSDB components provided in the DDoSCH. Ideally, it would be possible to integrate the filter rule generation as part of this process. The resulting filter rules should be exported and stored to DDoSDB as well.

5.2.2 Analysis, Design, and Implementation

The architectures of DDoSDB and DDoSGrid have been analyzed in-depth in the last iteration. It is important to note that the implementation of that architecture has changed with the previously described improvements. Now, models are compatible and the architecture is aligned so that it is simple to add additional interfaces in the form of HTTP API calls.

To integrate both the *ddos_dissector* and the *converters* modules, we first have to analyze their architecture. Both tools are simple and have a similar architecture. Both components are implemented as simple Python scripts that have a simple and short life cycle. The user has the possibility to provide an input file, as well as a number of options. The *ddos_dissector* script takes as input a PCAP file and produces a fingerprint file and a PCAP file. Optionally, the user can also request that a graph visualization of the source IP addresses is generated. Finally, it is possible to upload the resulting files to DDoSDB by supplying username and password [31]. This is not yet compatible with the approach taken in DDoSGrid, since we only want to store access tokens supplied by the OAuth2 server. With the current implementation, DDoSGrid would have to store the combination

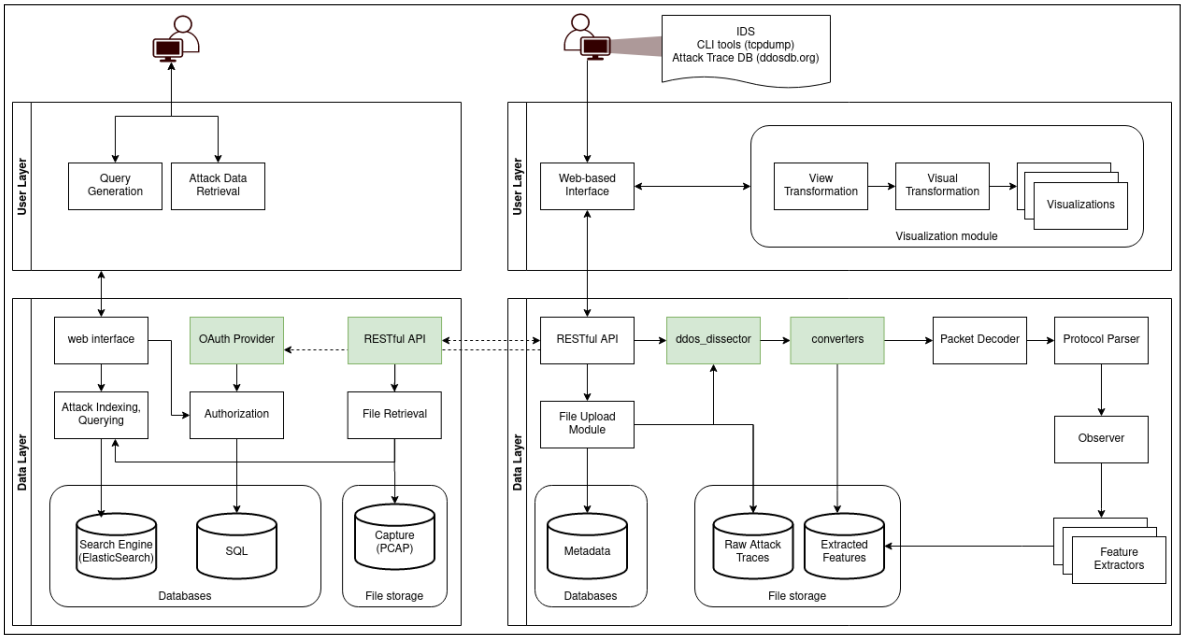


Figure 5.1: Integrated platform architecture

of username and password. Similarly, an interface to upload the files with the new token-based authentication does not exist yet.

To implement the functionality where input files are exported to DDoSDB, we propose the following additions:

- An interface in DDoSDB which accepts fingerprints and PCAP files. This interface needs to be accessible using the token-based authentication.
- An additional option within the *ddos_dissector* script that allows setting the authentication mechanism. Thereby, the username- and password-based authentication can co-exist with the token-based authentication mechanism.
- The *ddos_dissector* script has to be executed on the input PCAP file before the feature extraction of DDoSGrid begins.
- The *converters* script has to be executed on the fingerprint produced by the dissector before the feature extraction of DDoSGrid begins.

These additions were straightforward to implement since the efforts of the previous iteration led to the architecture depicted in Figure 5.1. Then, the script is executed as a child-process of the DDoSGrid API. This approach has the advantage that the application may continue to serve requests in another thread. Depending on the exit code of the script, we can store the status of the pre-processing step.

Differently, the *converters* module does not yet provide an uploading feature and DDoSDB does not yet accept the filtering rules generated by the script. To integrate this script as well, we first create a new interface in the DDoSDB web API. Then, we add an additional script to the *converters* project that allows the filtering rules to be uploaded to that

newly created interface. Similarly to the previous script, an access token can be supplied for authentication with the interface. The client, in this case the DDoSGrid API, runs the script as a child-process and uses its exit code to update the status of the analysis. Finally, these changes have to be made apparent to the user by changing the frontend of DDoSGrid and DDoSDB. This included three things, first that the status is displayed for each part of the process. Secondly, user controls have to be added to the DDoSGrid upload page so that a user can opt-out of the exporting feature. Finally, since this is a novel feature, a download link to the filtering rules has to be integrated into the templates provided by DDoSDB.

Concluding the export functionality, we can revisit the prototypical implementation created in the last iteration. There, the implementation has only been implemented to a degree where one can demonstrate a minimal level of compatibility. To achieve a productively working export functionality several improvements had to be made. The most important improvement is related to the performance of the import process. Here, we needed to adapt the implementation so that all chunks received from the DDoSDB API are written to both a file descriptor and to the SHA256 digestion algorithm. This allows saving both the PCAP file and computing the SHA256 hash string with appropriate memory usage.

Similarly to the *export* feature, the frontend had to be adapted to support the import process. This included an input form where the ID of the DDoSDB dataset as well as name and description may be supplied. The ID field is automatically populated if the URL contains a parameter pointing to the dataset ID in DDoSDB. In doing so, the user leverages the API call created in the previous iteration and import and analyze a dataset. To simplify the import of a dataset we add another link at the bottom of each dataset inside of DDoSDB. This link points to the import endpoint in DDoSGrid, so that all information is automatically passed to the import procedure. The place of such an *import* button was chosen so that it is as close to the existing *export* buttons present in DDoSDB. The components for importing and exporting a dataset can be seen in Figure 5.2.

At this stage, it became apparent that the overall procedure may be unclear to the user since many sub-procedures have to be considered. Therefore, the data model was extended yet again to allow better visualization of the status of the overall import and analysis.

5.3 Usability Improvements

The iteration described in the previous section introduced how an automated and integrated analysis process across both platforms was achieved. Considering a successful implementation, we can improve the user interfaces to reflect the integrated nature of the process. Therefore, we improve the platforms concerning their usability in this iteration.

With this iteration, no additional features were developed. However, the features implemented in the previous iterations were improved from a user interface perspective. This is important since it allows a realistic system test with users. Also, the improved user interface allows a broadened target audience beyond users who are familiar with command-line applications.

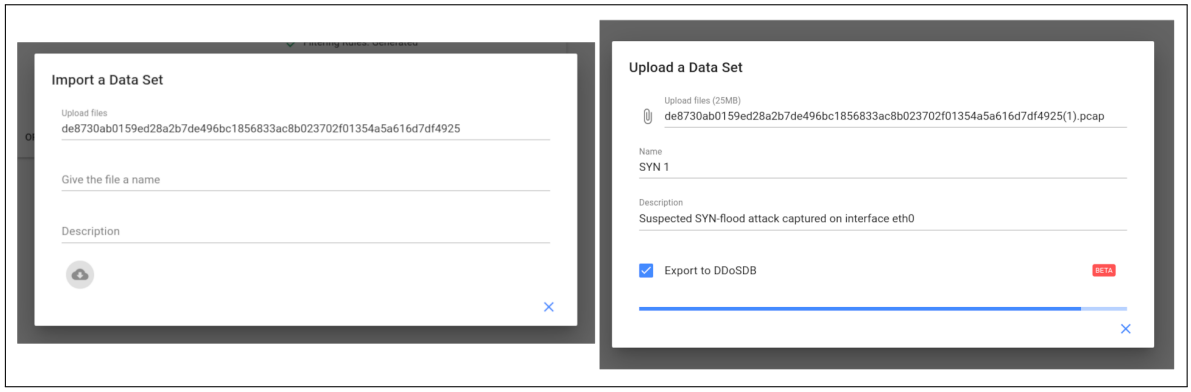


Figure 5.2: DDoSGrid: Importing a dataset (left) and exporting a dataset (right)

5.3.1 Requirements Prioritization

For this iteration the requirements that improve the usability of the overall system were prioritized. We can do so since the previous iterations have led to an automated, integrated process using both systems.

Although the underlying process is well automated and integrated, both platforms are still independent systems. To make it easier to work with these systems, it should be possible to traverse between both applications. For example, it must be possible to overview details of a dataset in DDoSDB even if it was exported using DDoSGrid. It needs to be easy to identify the same dataset on both platforms. Further, the user has to be able to control and inspect the various stages of the analysis process. For example, to see the progress and sub-process of the overall analysis. Also, he/she can decide to opt-out of certain steps to have the ability to continue using both apps independently.

First, the user should easily be able to traverse between both platforms without losing focus. Therefore, it has to be possible to click on a dataset in DDoSGrid and open an appropriate filter in DDoSDB to inspect the dataset. Similarly, opening a dataset from DDoSDB in DDoSGrid can be performed with a single click. After the second iteration, the user can automatically feed a dataset to both applications. However, it is hard to see at a glance if two datasets across both applications are the same. Visual clues have to be added to simplify this understanding.

In the previous iteration, the data model depicting one instance of analysis in DDoSGrid was extended so that there is a separate status for each sub-procedure of the analysis. For example, the sub-process to export to DDoSDB may have the state of *opt-out* while the feature extraction sub-process may be in the state *analyzing*. This has to be displayed visually so that the user understands what is happening.

A similar improvement can be done to the file upload functionality in DDoSGrid. PCAP files have no limiting size, so we need to optimize the case where a very large file is uploaded. At the beginning of this iteration, an in-deterministic spinner was used. Its disadvantage is that one cannot inspect the progress and speed of the file upload procedure. The export component which includes file uploading and the import component are shown in Figure 5.2. The situation could be improved even further: The frontend checks if the

app is currently in the foreground at the time where the analysis completes. If this is not the case, a notification is displayed to the user. If that notification is clicked, the user is taken to the *datasets* page of DDoSGrid.

Finally, the user interface reflecting the current state of authentication should be improved. If the user is not authenticated he/she should be redirected to the landing page and be informed that the action was prevented. However, a user-friendly procedure also considers preserving the prevented action. For example, a user may enter the platform by requesting the import of a dataset on DDoSDB. That action has to be prevented since the user first needs to authenticate. The prevented action is restored once authentication succeeds and the file import components have to be displayed to the user.

Based on the user feedback, another requirement has surfaced: some of the visualizations, such as the *port scan* visualization, which is visually transformed to a scatterplot, are hard to read if there are outliers in the dataset. For this visualization outliers are frequent, since normal traffic often centers around a port. For example, it is normal for hosts to receive most of the traffic on the port dedicated to HTTPS traffic. However, this makes it hard to read horizontal port scan attacks that only require a small number of packets per port. The requested feature is to have the possibility to manually toggle logarithmic scaling of the y-axis in these visualizations.

5.3.2 Design and Implementation

To traverse between both applications and focus on the same dataset, all prerequisites have been met with the previous iteration. That is, each dataset can be addressed using a URI. Now, we focus on visualizing that URI in the frontend. First, to link back to DDoSDB, each exported dataset has a hyperlink pointing to a page in DDoSDB which shows the dataset's details. In the other direction, an export hyperlink in DDoSDB allows opening the dataset in DDoSGrid. There, the web app intercepts the request and loads the dataset page with an import component. The URI identifying the dataset is parsed from the HTTP query parameter and pre-filled. Both hyperlinks pointing to the other platform are shown in Figure 5.3 which renders the representation of the same dataset across both platforms. Additionally, a feature existing in DDoSGrid was ported to DDoSDB. Therein, each dataset ID is visualized using an avatar. This avatar is computed by a JavaScript library directly using the ID. This has the advantage that no other information than the ID needs to be kept. This avatar was added to the templates in DDoSDB rendering the *query* and *overview* pages.

To improve the visibility of the analysis process, each status needs to be displayed in the frontend of the *datasets* page. Therefore, an icon along with a textual description is shown for each sub-process. To improve the visibility of the upload process, we can leverage an existing component of the library used in DDoSGrid. The progress can be computed from the file size and the progress event emitted by the HTTP request. For that, the HTTP module performing the upload had to be rewritten to use the old *XMLHttpRequest*, since the novel fetch API does not emit a progress event. To achieve the total size of the file we leverage the File API web standard [49].

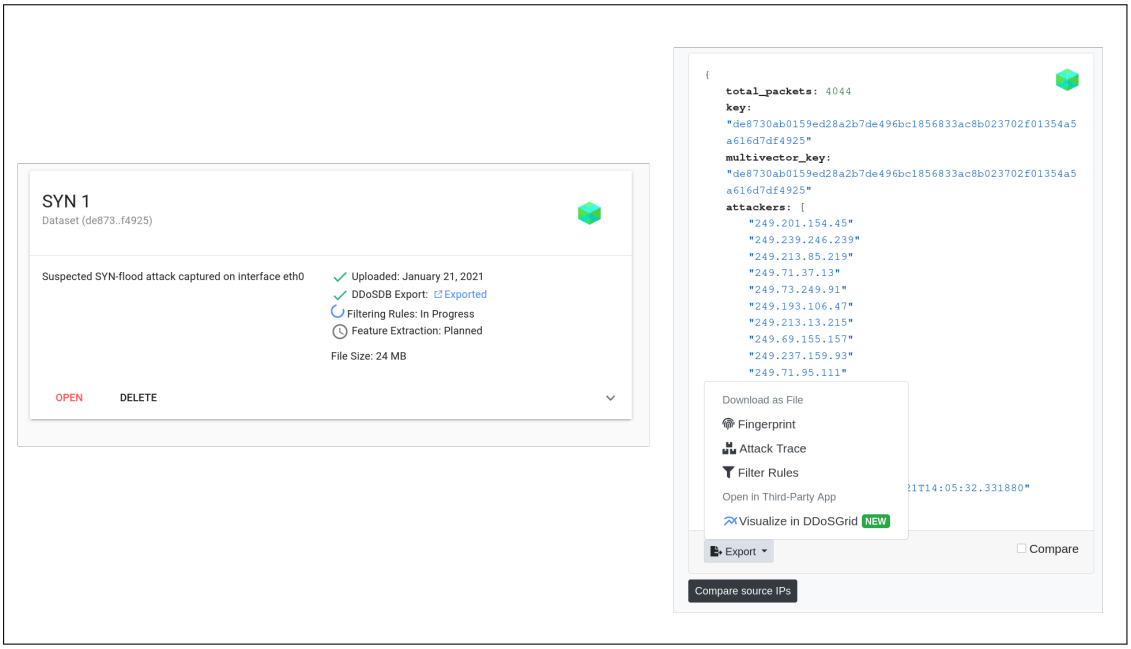


Figure 5.3: Representation of a dataset on DDoSGrid (left) and DDoSDB (right)

An appropriate user interface showing the state of authentication was implemented by creating new components that either show a warning if the action was prevented or display user information. Additionally, DDoSDB as the authentication provider was described. To ensure that the user does not perform actions on the web API without being authenticated, a client-side router library inspects the URLs being loaded. If a forbidden action is executed, a warning is shown and the prevented action is stored in the browser's session storage. Once the OAuth2 flow is completed, the user is redirected back to the web application where the same router library intercepts the page load and replays the action which was prevented by forwarding the user to the appropriate page. The feature to allow visualizations using logarithmic scaled was implemented by integrating a toggle switch using the component library. The actual scaling is performed automatically by the JavaScript library that provides the underlying visualizations, such as the scatter plot chart.

5.4 Discussion on the Dimensions

Using three iterative development cycles, an integrated, automated system consisting of the two platforms DDoSDB and DDoSGrid was built. Three dimensions were targeted for improvement, of which many functional requirements were met using the newly developed platform. In contrast to the previous chapters, all requirements were considered in these three iterations.

An exception are two requirements from the usability dimension. To focus on the platform aspect we refrained from developing additional visualizations. For example, one requirement intended to visualize the information extracted by the *converters* module. However, this information is well integrated into the developed system and available to the feature

extraction module. This allows new approaches to be considered as future work, since with the integration of the fingerprints, we consider that labeled data is currently available. This could be used to further augment novel feature extraction modules which use a supervised learning algorithm.

Similarly, another possibility for future research is to integrate other databases. Considering the first dimension, an OAuth2 authorization server was implemented into DDoSDB and DDoSGrid was refactored to act as an OAuth2 client application. Thereby, additional data sources that provide access by implementing the OAuth2 protocol can be integrated with ease. For example, we can now consider real-time data feeds of productive systems such as a NetFlow collector or an Apache2 Kafka cluster. Reflecting on the integration of such an authentication and authorization protocol we see improved maturity of DDoSGrid with regards to integrating applications used by network operators.

In the same way, we see several improvements in DDoSDB due to the necessity to connect it to DDoSGrid. The outcome is a set of interfaces that can be used to integrate other third-party applications. DDoSGrid has been developed to act as such an application that can be seen as both a consumer and producer of datasets. This shows that DDoSDB is now not only a web application to explore and download datasets, but also a database functioning as a platform. For example, a network operator could enable the automatic publishing of attack data based on the feedback of an Intrusion-Detection-System (IDS).

The privacy aspect of DDoSGrid was improved, since the usage of a dataset is personalized. Additionally, DDoSDB was improved with respect to its usability. Numerous user interface improvements make it easier to identify and work with a dataset. Further, CLI tools such as the *ddos_dissector* and *converter* do not have to be installed, configured, and executed manually. These tools can now be invoked through the user interface provided by DDoSGrid. We consider a broadening of the target audience since technical details and prerequisites are hidden. For example, a network operator may not be familiar with tools commonly used in Software Engineering. Installing Python and multiple CLI tools may deter this audience from publishing attack data. A simple file upload, on the other hand, requires no prior installation or knowledge.

Chapter 6

Evaluation

In this chapter, we establish the actual contributions provided by this work. Therefore, we first investigate the functionality provided by DDoSGrid v2 and the benefits these features provide using case studies. As the synthesized requirements in Section 4.3 revealed, a strong focus of this work lies in usability improvements. The contribution to usability is established using a System Usability Scale (SUS) [56]. Finally, we investigate on a more technical level the scalability of the platform. This provides evidence of the platform's capacity to provide the previously postulated features when dealing with high volume data.

6.1 Case Studies

In this section, we show the contributions of this work based on the provided functionality of the DDoSGrid platform. We do so by the means of six scenarios consisting of initial questions that are then solved using DDoSGrid. First, we show how a cybersecurity trainer finds attack data on DDoSDB to visualize using DDoSGrid. Next, we showcase several visualizations from which the operator can derive information on the current state of the network. In the third scenario we go more into detail and show how information on flows in the attack can be derived and made visible. The next two scenarios require a combination of features provided by DDoSGrid, so that attacks can be detected and identified. In the last scenario we consider a radically different question and show that DDoSGrid can be extended to answer that question. The objective here is to show that DDoSGrid, although specifically developed for DDoS attacks, can be used to efficiently visualize almost any aspect of any protocol and that for very large PCAP files.

6.1.1 #1 Educating DDoS attack patterns

Let us consider how the features provided by the integration of DDoSCH and DDoSGrid can be used in an educational context. This could apply to a range of scenarios found in an academic or industrial context. The goal of such a scenario is to enable the training

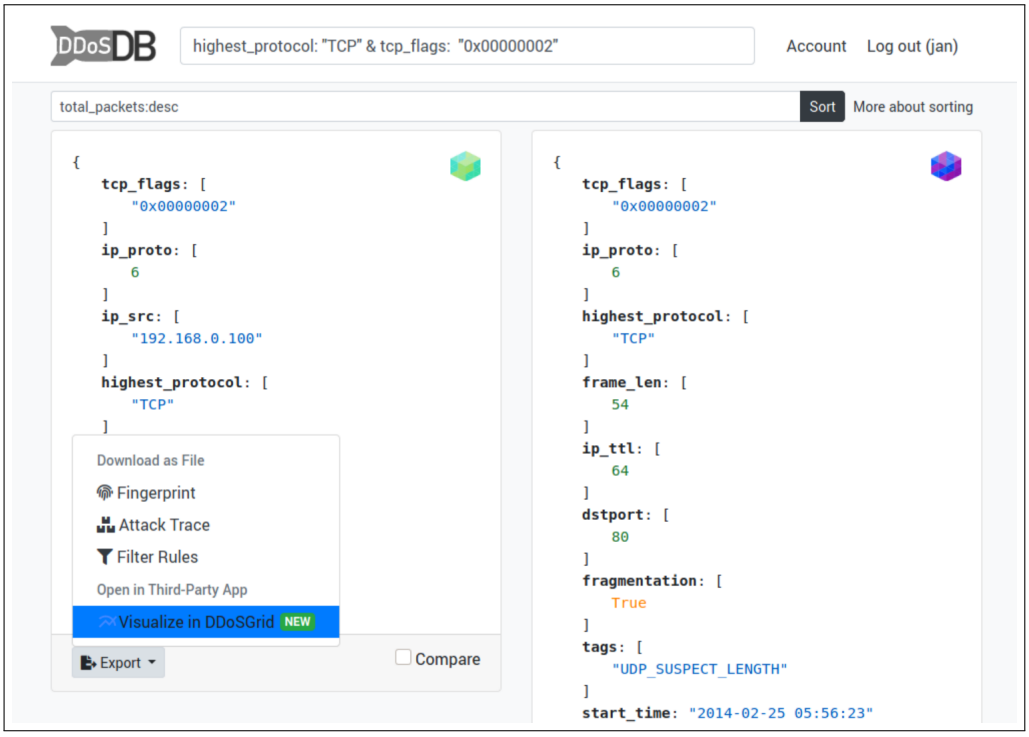


Figure 6.1: Querying and Importing Relevant Attack Data

participant to perform the detection and identification steps of DDoS attack mitigation. For now, we focus on training users to detect patterns. Therefore, we assume a researcher who holds in-depth knowledge on various DDoS attack vectors. He/she has the goal of demonstrating patterns of common attack vectors to network security personnel. Two requirements are important, first, he/she wants to work with traffic collected from real-world attacks. The reason being that he/she knows that for every DDoS attack there is always a certain amount of benign traffic. This makes it harder to spot the patterns pointing towards a specific attack vector. For precisely this reason it is important to him to work with traffic captured in real attacks. He/she wants to show the pattern of an attack vector with varying levels of noise to prepare students for a realistic use case. Secondly, this user considers this preparation a repetitive task that he/she will likely perform at regular intervals. Therefore, he/she strives for the highest degree of automation for as many steps of this work as possible. This includes the two main steps, gathering appropriate datasets and replaying a previously created dashboard.

For the first step of finding real-world attack data, the user is strongly supported by the integrated components. Let us make an example to show how to gather attack data for one of the attack types. In this case, the user is searching for the largest attack available containing an SYN-Flooding attack vector. To find such an attack, we clearly see the contributions provided by the *ddos_dissector* component. Since fingerprints are generated and indexed for each dataset, we can use any property of the fingerprint to formulate a query. A simple Lucene query makes sure that we only select attack types that contain a large number of TCP segments and that a significant amount of segments is in the desired connection state:

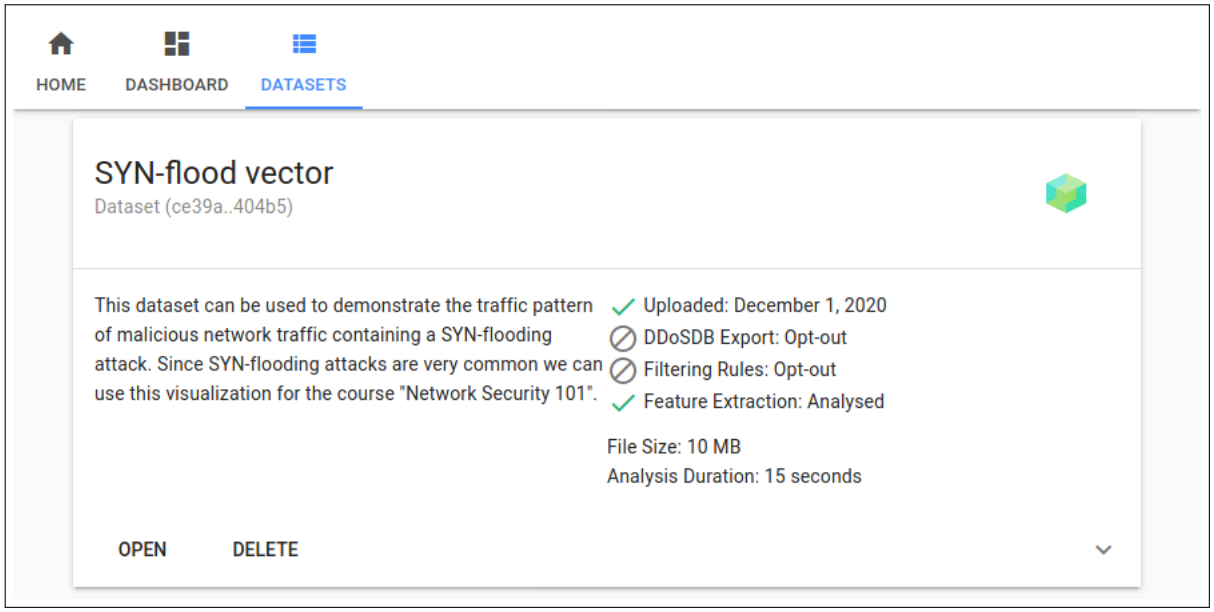


Figure 6.2: Imported Dataset in DDoSGrid

```
highest_protocol: "TCP" \& tcp_flags:  "0x00000002"
```

Now, the user finds a selection of relevant attack datasets and wishes to select the one that contained the largest number of packets. Therefore, he/she adds a parameter to sort all results by the number of packets `total_packets:desc`. With one simple query, the user finds the exact dataset he/she wants to show to his users. As shown in Figure 6.1, the user now investigates the remaining metrics and then opens the first result returned by the query in DDoSGrid. After importing the file into DDoSGrid the user explores the dataset in more depth as provided through the fingerprint of DDoSDB. Once the user is satisfied with the dataset, he/she can store it in one of three ways. First, he/she can simply keep the dataset in the DDoSGrid system. Each user has personalized storage to persist datasets. So, together with the metadata defined by the user, it is easy to identify the correct dataset at a later stage, as shown in Figure 6.2. Since the user can easily navigate back to DDoSDB, he/she could also export the assets and store them in another system manually. A better approach to export and store such files would be to integrate a script into the import process of DDoSGrid. For example, consider a user that plans on maintaining a separate database of certain datasets, such as in this case, datasets appropriate for training. In this case, the user can easily set up an on-premise instance of DDoSDB and reconfigure the export script so that a dataset from DDoSGrid can be exported to that local instance.

6.1.2 #2 Auditing Preventive Measures in an Enterprise

The main objective of DDoSGrid is to provide convenient and efficient access to insights gathered from network traffic. To transform these insights into actual benefits for an insight we need to transform these insights into actions. In this case study, we consider how we use the integrated platform to collect network traffic of an enterprise. The collected

network traffic is then used to audit the enterprise's infrastructure against a set of preventative measures. As an example of such preventative measures, we focus on the ones proposed by the National Cyber Security Centre (NCSC) of the Swiss government. The NCSC proposes 13 steps that should be initiated and verified as a precautionary measure against DDoS attacks. We focus on eight of these items we consider to be technical, since non-technical measures, such as establishing responsibilities among personnel, is unlikely reflected in network traffic [66]. For a number of these, we show how DDoSGrid helps to implement such measures. Additionally, we show how the integrated DDoSDB component is used to record these measures.

Let us first consider how an enterprise would deploy and use the platform. Since we are interested in deploying preventative measures, the platform must be deployed as early as possible. Therefore, an enterprise sets up a local instance of DDoSGrid and DDoSDB to fully guarantee data privacy. After all, even anonymized network traffic reveals certain attributes about the enterprises' infrastructure. Ideally, the enterprise already has samples of network traffic captures as Netflow or PCAP files. For our case study, we assume a service provider that provides hosting services for websites and web applications. The company has already collected traffic samples from situations where both malicious and benign traffic was gathered.

The first step after deploying the platform is to index the available network captures. With the implementation of the API in DDoSDB, it is possible to automate this task with a simple script. This script uploads every PCAP file to DDoSGrid with a description indicating whether it depicts traffic from a "normal" or attack situation. After uploading to DDoSGrid, every PCAP is automatically analyzed, converted to firewall rules, and stored in the database.

At this point, a security analyst analyses the metrics, fingerprints, and visualizations to audit the infrastructure. Since the enterprise does not employ such personnel, they have contracted a company providing security consulting. Through a network tunnel, the enterprise provides secure access to the DDoSGrid platform so that the analyst can analyse the extract information remotely. First, the analyst starts with the following item, where he/she needs to establish how the pattern of normal attack traffic looks like:

> "You know the "normal status" of your network and systems and can detect abnormalities (intrusion detection systems [IDS], centralised log analysis) (..)" [66]

Therefore, he/she opens multiple datasets labeled with "normal" traffic situations. On the "datasets" page he/she first inspects the metrics to have an overview of the traffic. Four important insights are derived from this:

1. Since the traffic capture was gathered over a long time and under high load, it is common that a bandwidth of 37 MB/s needs to be handled.

2. The majority of all packets carry HTTP traffic. This is unsurprising, since the enterprise mainly offers web hosting services.

Figure 6.3: Source Hosts of Benign Traffic Visualized on World Map

3. For the observed duration of 60 minutes, only around 10'000 source IP addresses were extracted.

4. The average packet size is around 5 KB.

This already provides a good overview of the characteristics of benign traffic. To gather more detailed insights, the operator opens several visualizations based on the metrics described above. First, he/she wants to know more about the 10'000 source IP addresses. Since this information is usually carried on the network layer through the IP address, he/she looks at the "Top 100 Sources by Traffic" visualization. As depicted in Figure 6.3, almost all traffic comes from Switzerland. Other origin countries are Norway and the USA. Next, the operator wants to investigate the application-layer traffic in more detail. From the metrics, he/she knows that HTTP is the most important protocol to consider on this layer. He/she investigates the traffic pattern of the HTTP requests by opening the visualization that reveals the distribution of HTTP verbs. There, he/she finds that there are almost equal shares of HTTP "GET"- and "POST"-requests.

Now that the operator knows about the most important services and their characteristics, he/she wants to find out whether other services are running. This is important, since it may influence the stability and robustness of the infrastructure as suggested by the NCSC:

> "Your systems are stable and robust (no unnecessary services, (..) etc.) and have the latest patch level. SYN cookies are activated, etc." [66]

To get an idea of whether other services are being hosted on the same infrastructure, he/she investigates the visualization on the most frequently targeted TCP- and UDP-ports. There, he/she sees that port 80 receives by far the largest number of segments. However, there is also a substantial number of DNS requests being served, since port 53 is the second most frequently observed destination port. Since the analyst does not know about the importance of this publicly exposed DNS server, he/she exports the visualization to discuss the implications of running the service along with the main service. A similar measure can be reviewed from the same visualization. If there were a large number of requests being made to various services, he/she could infer that there are lose firewall restrictions. That would have been something the NCSC discourages:

> "An upstream firewall only lets required protocols through to the system. The firewall has enough system resources to continue to function even in the event of a DDoS attack. (..)" [66]

To look at a broader range of ports, he/she opens the visualization that renders the same information with a different abstraction. The scatterplot visualization groups ports into buckets so that one can see whether there are a large number of ports targeted. Since the operator only finds traffic towards the DNS- and web-services, he/she is not concerned. Continuing with the list of preventative measures he/she investigates whether blocking on the network-layer is a viable option to reduce the attack surface:

> "Check the possibilities offered by Geo-IP blocking. (..)" [66]

The operator has already observed that Switzerland is the most important origin for its customers. Geo-IP blocking can therefore be considered to be a viable measure. So he/she stores the visualization in a dashboard which he/she will restore at a later stage to discuss the findings with the service provider.

Finally, the analyst needs to establish an understanding of the internal deployment of the components. According to [66] an important preventative measure is to separate the traffic created by the organization itself from the traffic generated through the provisioned services:

> "Systems that could potentially fall victim to a DDoS attack (*e.g.*, web-sites) should be connected to a different internet uplink to that used by the organisation's other systems. (..)" [66]

Here, the analyst first runs into the issue that there are no feature extraction module reveals information for this measure. However, since the analyst knows that he/she will repeat this analysis at regular intervals, he/she decides not to analyze this information manually. Instead, he/she implements a feature extraction module so that each PCAP file stored in DDoSDB in the future automatically carries this information. Since VLAN tagging is automatically added to each decoded Ethernet frame in DDoSGrid, one can achieve such automation with just a few lines of code.

First, the analyst creates a subclass of `AbstractPcapAnalyser`. There, he/she needs to implement the three major life cycle phases of each feature extraction process.

```
async setUp () {
  this.pcapParser.on('ethernetPacket', this.checkEth.bind(this))
}
```

First, the `setUp` phase, where additional libraries may be loaded and the user creates observers for events that correspond to protocols or abstractions thereof. Since the analyst is only interested in the VLAN IDs, with which each Ethernet frame is tagged, he/she just creates an observer for that protocol.

```
checkEth (ethPacket) {
  var vlanID = ethPacket.vlan.id
  var existingEntry = this.results.hasOwnProperty(vlanID)

  if (existingEntry) {
    this.results[vlanID]++
  } else {
    this.results[vlanID] = 1
  }
}
```

Now, each Ethernet packet is passed to the callback function that implements the extraction logic. Simplified, the main statistics that are gathered are a counter for each observed ID.

```
async postParsingAnalysis () {
  var sortedByCount = this.sortEntriesByCount(this.results)
  var topNentries = this.getTopN(sortedByCount, N)

  var fileName = `${this.baseOutPath}-${analysisName}.json`
  var fileContent = {
    // Signal and format to visualize as piechart
    piechart: {
      datasets: [{
        backgroundColor: [ ... ],
        data: this.formatData(topNentries)
      }],
      labels: this.formatLabels(topNentries)
    }
  }
  var summary = {
    fileName: fileName,
    attackCategory: 'Link Layer',
    analysisName: `Top ${N} VLANs`,
    diagrams: ['PieChart']
  }
  return await this.store(fileName, fileContent, summary)
}
```

At some point in time, the last packet is decoded and the third life-cycle phase begins. Here, the analyst needs to compute the final result from the collected statistics, enrich it if applicable, and finally format and signal how the result should be presented. The analyst decides to sort all counted VLAN domains by their frequency. Then, he/she picks the most commonly observed VLAN domains and formats the data so that it can be plotted on a piechart. Since this plot is already available, the analyst can simply signal

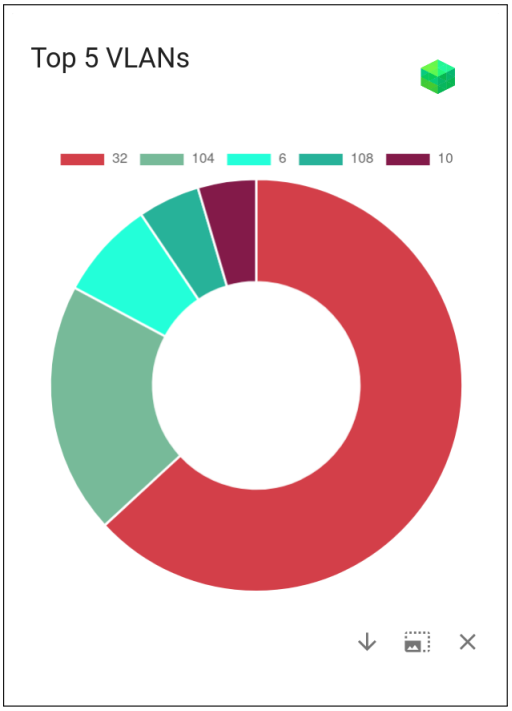


Figure 6.4: Outbound Traffic Distribution Over Available VLAN Domains For One Uplink

the type of plot and define metadata of the module, such as the category or name. After integrating this change into DDoSGrid, he/she can then collect traffic that was captured for each uplink and analyze it. He/she knows that VLAN with ID 104 is used to provide internet access to the employees. The remaining VLAN domains belong to provisioned services. As presented in Figure 6.4, we can already see that both the internal network and the network for the services share the same uplink. According to [66] this is not ideal. Therefore the analyst adds this visualization to the dashboard that he/she will later use for discussion.

At this point, the analyst has reviewed all of the technical measures that [66] proposes to implement and analyze. The analyst was able to derive insights from existing and novel visualizations and has created a dashboard that he/she can later use in the discussion with the service provider. To be able to repeat this analysis in the future, we use DDoSDB to provide centralized storage for network captures. Therewith, we can use DDoSGrid to have instant access to analyze traffic patterns.

6.1.3 #3 Analyzing and Reporting Cyberattacks

In the previous Section 6.1.2, we outlined how DDoSGrid is used to perform auditing of preventative measures in an enterprise. DDoSGrid was used to analyze the implementation of these measures, which was conveniently supplemented by an auditing trail provided by the DDoSDB component. That process was executed before an actual DDoS attack was launched against the enterprise's infrastructure. In this case study, we show how useful the establishment of such measures is by doing an actual post-mortem analysis of a DDoS attack. We consider the same enterprise, however, we focus on only one

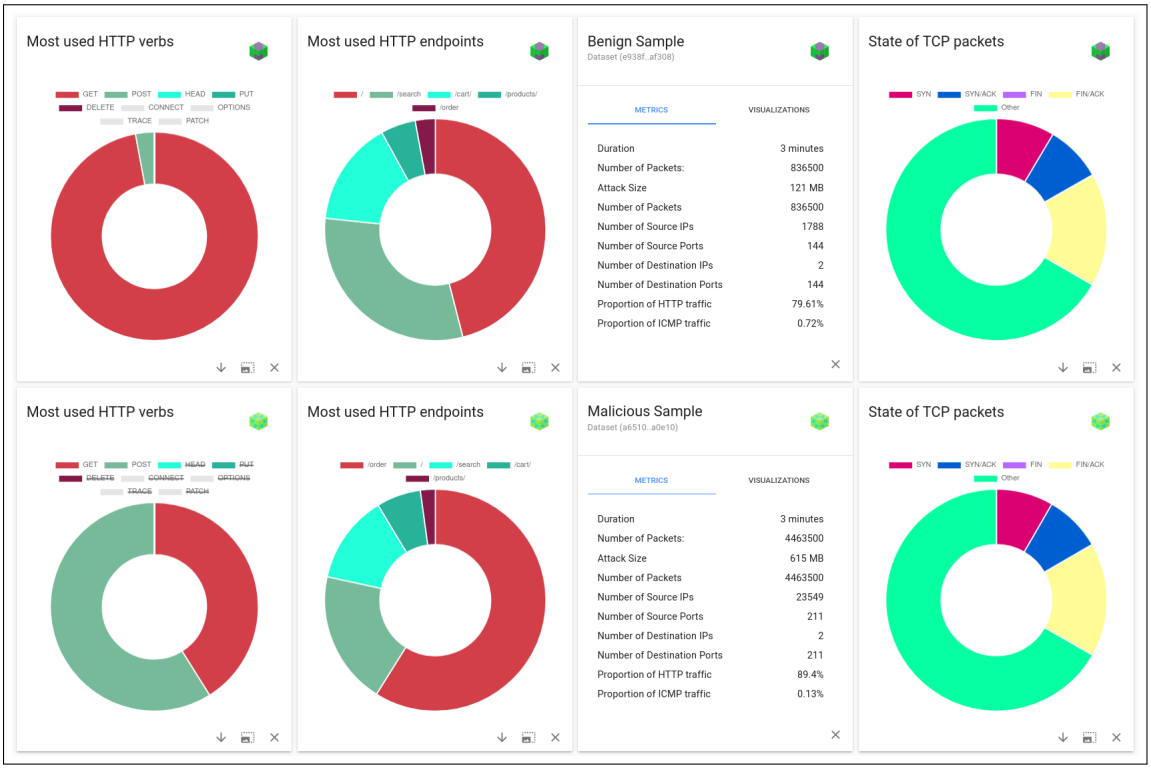


Figure 6.5: Comparing HTTP traffic of two samples: Benign traffic (top) and attack traffic (bottom)

web service provided to a customer who operates a web-shop. During the actual attack, only limited actions were taken due to missing personnel that is knowledgeable of DDoS attacks. However, the network operators in charge of maintaining the services followed the same document that was already used to establish preventative measures defined by [66]. The most crucial step for a post-mortem analysis was adhered to:

> "1. Record the attack (..). These are important for subsequent analysis and for reporting the attack." [66]

This first advice provided by [66] is an apt description of post-mortem analysis. The network samples were therefore collected by capturing the traffic on the interface that routes the traffic towards the service of the web-shop. These samples were then directly exported to the local instance of DDoSDB. At a later stage, the security consultant is requested to perform post-mortem analysis of the attack. Again, the consultant can easily perform this task remotely since he/she has access to DDoSGrid. The integration of DDoSDB proves especially powerful at this stage since the consultant can import samples of benign and malicious traffic directly from DDoSDB. Once imported in DDoSGrid, he/she can open visualizations of both datasets to compare the traffic patterns side by side. To perform the actual post-mortem analysis, the analyst follows the steps outlined in [66] which can be performed after the attack has already passed. The most important insight to create is the actual attack vector that is used. This is important, since it influences other actions to be taken. For example, if the attack vector is an SYN-flooding attack, the analyst can consider that the source IP addresses are forged:

> "The source IP addresses of the attack are probably bogus (..) for SYN, UDP, BGP and SNMP flooding. It makes no sense here to filter the IP addresses and it could even block legitimate users" [66]

Establishing the attack vector is greatly simplified since we have a benign traffic sample available in DDoSDB from the auditing process. None of the metrics and visualizations point towards an attack on the network- or transport-layer. For example, a common attack vector on the network-layer are ICMP-based attacks. However, as we can see in Figure 6.5, only 0.13% of packets carry an ICMP message. Similarly, we can rule out irregularities on the transport-layer as described in the quoted measure. For example, looking at the visualization that shows the TCP connection states we see equal distribution between connection setup and teardown. Further, the distribution is similar to the one found in the benign traffic sample. This indicates that there is no SYN-flooding attack in the attack vector. We also see that there are 0 UDP segments in the network traffic. This is a result of performing the preventative measures in the previous section where unused services were turned off and unwanted protocols are filtered. This proves how DDoSGrid can be used to detect security risks and harden the security of the infrastructure. This does not only improve the security of the system but also eases the attack vector identification since we can ignore UDP-based attacks.

Since no irregular patterns were found, the analyst moves up to the application layer. Since the metrics revealed that almost all packets carry HTTP traffic, we can assume that the attack vector is HTTP-based. Looking at the distribution of frequently used HTTP paths and verbs we find two irregularities. First, the benign traffic sample shows that during normal conditions, most HTTP requests are "GET"-requests. During the attack most requests used the "POST" verb. Additionally, we see that during normal conditions, most requests target the root path and the path to query products. During the attack, most requests targeted the path that is used to submit an order. This is suspicious since it is hard to imagine an e-commerce scenario where users submit more orders than that they look for products. Here, we make use of another powerful feature of DDoSGrid. Since the visualization is interactive, the analyst can filter out the other paths and just compare the "search" and "order" paths. As presented in Figure 6.6, the distribution between both paths is almost inverted during attack conditions. From this visualization, the analyst concludes that the attack vector is an HTTP-POST-flooding attack.

Now that the attack type is known, we can infer more information about the attackers to then discuss various measures. From the metrics defined on each dataset, we see that the number of source hosts is only slightly larger than during normal traffic conditions. From that we can follow that the attackers did not employ a large botnet. Here, the visualizations that were established during the previous analysis help in differentiating real customers from malicious hosts. As highlighted in Figure 6.3, legitimate customers usually stem from Switzerland. However, during the attack a large number of packets was sent from China as shown by the world map. It is important to note that we can only use this information since it is already established that source IP addresses are not spoofed. Now, we correlate this information with the visualization that reveals the most frequently observed sources. As presented in Figure 6.7, the information for each host is supplemented by information derived from a WHOIS database. Therefore, we have

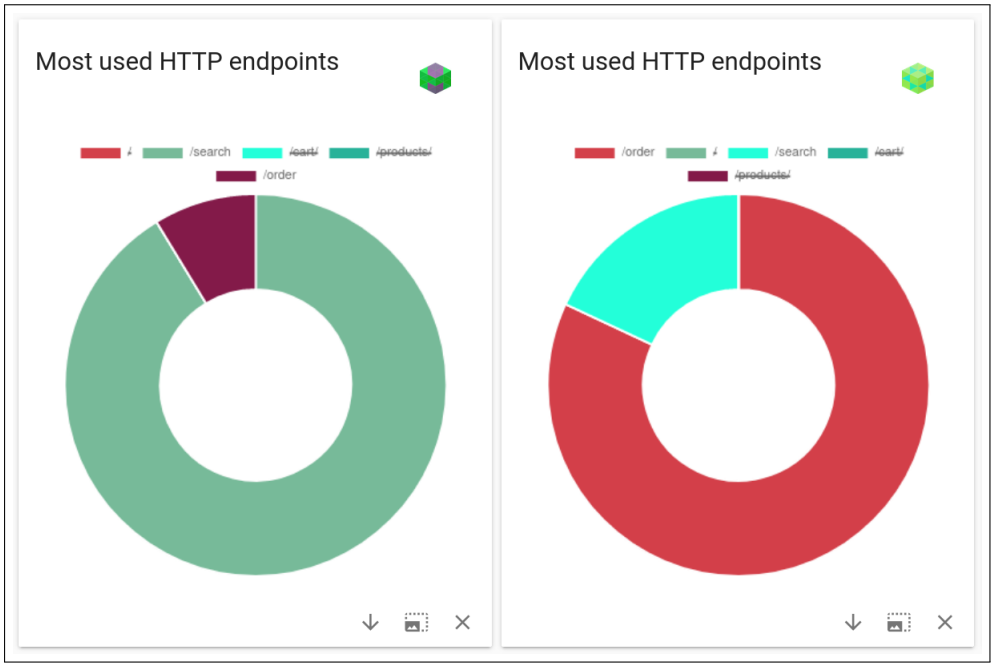


Figure 6.6: Distribution requests towards manually selected HTTP paths

supplemental information such as prefix, autonomous system number, and origin country for each host. Using the country of origin, we can see that four of the hosts could likely be attackers.

At this point, the analyst knows the attack vector and additional information on the attackers. Using his experience and additional steps defined in [66], he/she prepares a dashboard to report the insights to the service provider. Along with these insights, he/she proposes multiple actions to be taken.

First, the infrastructure is already hardened to some degree. This can be seen by the single attack vector that is used. Also, there is little traffic that does not target the services, such as UDP-based protocols. Additionally, he/she proposes to request geo-blocking for the origin country from the ISP that provides access to the services. Alternatively, the ISP or the service provider can blacklist these source IP addresses at the ingress router. It is important to note that this strategy only works until the traffic volume saturates the bandwidth provided by the network. However, in this attack, the computational resources of the web-shop were the main target and the attackers did not manage to saturate the links as shown by the metrics. Therefore, such filtering would have successfully defended against the DDoS attack. A more complex solution would be to install a web-based firewall, which provides filtering on the application layer. Such a firewall can be installed on the premises of the service provider or rented from a DDoS mitigation provider. For example, Cloudflare provides a scrubbing service for these types of attacks, where only legitimate traffic is forwarded to the service provider [58]. Finally, it is important to consider that [66] provides a reporting form for such cyberattacks. At the time of writing, an enterprise is not legally required to report cyberattacks. However, there are voices in the Swiss government requesting to create such a legal obligation [60] as it is already the case in other countries, such as Austria [61]. Here, DDoSGrid can be used as part of a holistic auditing and reporting process. As part of future work it would be possible

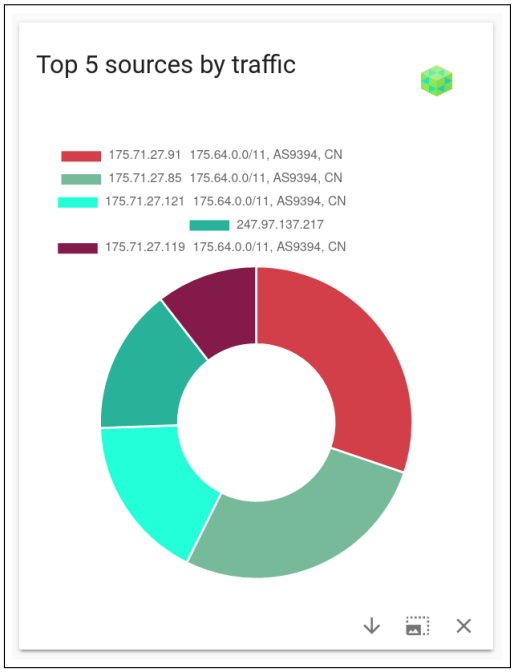


Figure 6.7: Five hosts emitting the largest number of packets

to integrate DDoSGrid with such a reporting body. This would simplify the reporting process and enrich it with information available in a central component. An example of how such integration could look is presented in Figure 6.8.

In this case study, we have shown how users can use the created platform to perform attack vector identification. That information was then used to derive additional information on attackers and was finally used to derive new measures. These measures included technical and structural improvements. For these operations, the integrated nature of DDoSGrid with DDoSDB proved to be a powerful property of the platform that created new capabilities. Finally, we see that with the integration of these components other systems can be integrated as well.

6.1.4 #4 Visualizing Routing and Reachability Information

In this final case study we show how the newly improved DDoSGrid platform can be used to visualize any aspect of network traffic. For this, we use a slightly different setting, where the victim of a DDoS attack deters the traffic by having it routed to a commercial DDoS mitigation provider. The victim, who already has a formal agreement with the mitigation provider, has a DNS- and a BGP-based option to route the traffic to the provider. Due to the victim's infrastructure already operating a BGP speaker that is configured to interact with a BGP speaker in the mitigation providers network, this routing protocol is chosen. The actual DDoS attack was detected automatically based on the configuration of an IDS using insights from DDoSGrid. DDoSGrid was used in a post-mortem analysis of a previous attack. Therefore, the victim already maintains a private instance of the platform. Now, he/she intends to use the same platform to visualize the effects of doing a route advertisement. To be specific, they wish to visualize the messages exchanged with

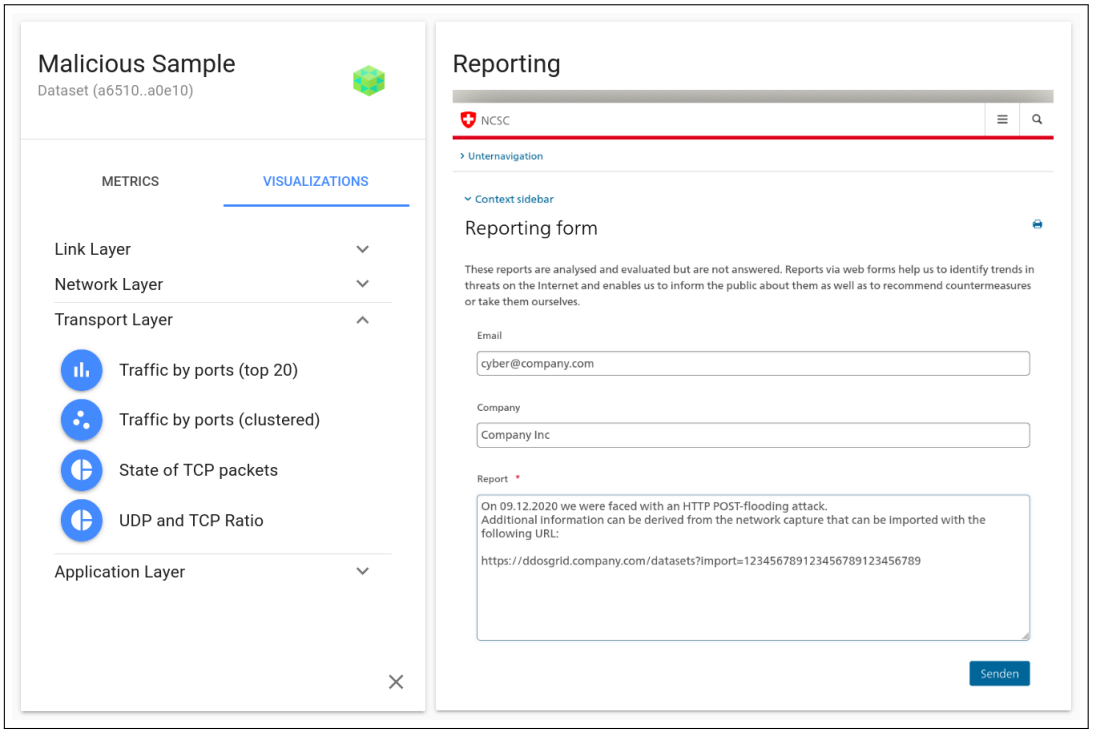


Figure 6.8: Mockup of an integrated reporting cyberattack database

the other BGP speaker. This use case differs from visualizing actual DDoS attacks in terms of traffic characteristics and protocols. DDoSGrid lacks three things to visualize that traffic. First, a decoder for BGP messages, then a feature extraction unit, and finally a fitting visualization. Now, we will see that it is still possible to visualize the messages by providing these missing components. The result is an efficient and scalable visualization of BGP messages which demonstrates that almost any aspect of any protocol can be visualized efficiently using the platform.

To implement the decoder we create a new JavaScript class `BGPDecoder` to hold the logic for decoding the various BGP messages. Regarding the existing codebase, we only need to extend it so that it invokes the decoder if we can assume that we are holding a BGP message as a payload. The DDoSGrid parser already decodes the service name based on the source- or destination port for all ports in the well-known port range. Therefore, we add a hook to invoke the decoder:

```
1  applicationPacket (port, packet) {
2      var serviceName = getService(port)
3      // ..
4      if (serviceName === "bgp") {
5          emit("bgpPacket", decodeBGP(packet))
6      }
7      // .. Other application-layer protocols
8  }
```

At this point, our empty `BGPDecoder` class is invoked with a Buffer that holds the binary payload of the TCP segment. The class provides a simple interface, a single `decode` accepts a Buffer and an optional integer holding the offset as a number of bytes. This

method then returns a JavaScript Object that represents the parsed message or an error. To parse the BGP protocol, the developer uses the Buffer class to extract the information as defined in RFC 4271 [57]. To give an idea of how this works we consider how the common message header is decoded. First, we check if the Buffer is empty and then we check the first 16 bytes. In BGP, these marker bytes should all be set to the hexadecimal equivalent of FF. At this point, we can be fairly certain that the Buffer represents a BGP message. Therefore, we can parse the two remaining properties from the fixed-size header. First, the length of the BGP message, as shown in line 15. Secondly, we parse the message type on the next line. We store both values and then look up the message type, which was represented as a one-byte integer. Then, based on the message type, we invoke the appropriate function that decodes the remainder of the message.

```
1
2  BGP.prototype.decode = function (raw_packet, offset=0) {
3    if(raw_packet === null) { return }
4    var marker = new Buffer('FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF', 'hex')
5    if(!raw_packet.slice(offset, 16).equals(marker)) {
6      throw new Error('Marker malformed')
7    }
8
9    var messageTypes = {
10     1: 'Open',
11     2: 'Update',
12     3: 'Notification',
13     4: 'Keepalive'
14   }
15   this.length = raw_packet.readUInt16BE(16)
16   var messageTypeCode = raw_packet.readUInt8(18)
17   this.messageType = messageTypes[messageTypeCode]
18   this.payload = raw_packet.slice(19)
19
20   if (messageType === 'Open') {
21     this.message = new OpenMessage(this.payload)
22   } else if (messageType === 'Update') {
23     this.message = new UpdateMessage(this.payload)
24   } else if (messageType === 'Notification') {
25     this.message = new NotificationMessage(this.payload)
26   } else if (messageType === 'Keepalive') {
27     this.message = new KeepaliveMessage(this.payload)
28   } else {
29     console.warn('Unsupported Message Type')
30   }
31
32   return this
33 }
```

As we can see, it is easy to work with Buffers and the protocol can be easily decoded. The remainder of the BGP decoder's complexity is influenced by the actual protocol. Since we have already demonstrated how the protocol can be parsed from the Buffer by parsing the header, we are not showing how the remaining message is decoded. In reality, this requires around 200 lines of code after which we can see how information can be mined from the decoded messages.

With the `BGPDecoder` class providing access to decoded BGP messages, we can now

implement a feature miner that aggregates information from these packets. For that, we register a new feature miner class `BGPMessages`. As shown before, this class needs to implement the lifecycle methods `setUp`, `extract`, and `postParsing`. The `setUp` phase consists of simply registering for BGP messages from the previously created decoder:

```
1   async setUp () {
2     this.pcapParser.on('bgpPacket', this.extract)
3   }
```

Now, we implement the actual extraction method that gathers the information. As we will see once we implement the final component, we try to gather the information so that it can be visualized on a timeline. That is, we try to gather information in two dimensions. The first dimension is the source address of the BGP speaker and the second dimension is the index denoting the chronological order of the messages. To hold the index, we simply use a list as a data structure so that the order is preserved. For the first dimension we use the source address of the packet. Additionally, we gather information such as the message type and a summary of the message content.

```
1   extract (message) {
2     this.results.push([
3       message.srcAddr,
4       message.messageType,
5       message.message.summary
6     ])
7   }
```

Once the last packet is decoded, we have a list of objects holding the summarized information. Now, we need to signal to the visualization system how the mined information can be visually transformed. Since there is no fitting chart that can be reused, we define a new diagram type to plot the data. Then, we define the metadata, such as the related OSI layer to categorize the visualization:

```
1   async postParsingAnalysis () {
2     var fileName = `${this.baseOutPath}-bgpMessages.json`
3     var fileContent = { timeline: this.results }
4     var summary = {
5       fileName: fileName,
6       attackCategory: 'Application Layer',
7       analysisName: this.getName(),
8       supportedDiagrams: 'Timeline'
9     }
10    return await this.storeAndReturnResult(fileName, fileContent,
        summary)
11  }
12 }
```

After the implementation of the final life cycle method we are finished with the backend functionality. Now, we define a new visualization module to render the data. We encapsulate this into a new component called `Timeline` which matches the property `supportedDiagrams` in the result of the miner. This automatically renders that component once the user clicks on the BGP visualization. Note, that we do not need to define any other logic to link the miner to the component. The component receives the URL of the JSON file written by the feature miner. Therefore, the component needs to fetch that

information from the API and plot it. First, we use that URL to retrieve the content of the JSON file and then extract the messages from the response. These are then appended to the internal state. That state is predefined with the schema of our visualization. As we saw before, there are three properties for each message. The source address is plotted as the entity of the timeline. The message type is then used as a label. Finally, the summary of the message is passed to the body of the timeline item. This is only shown when hovering over the message in the timeline.

```
1  import { GChart } from 'vue-google-charts'
2
3  export default {
4    name: 'Timeline',
5    props: [ 'url' ],
6    mounted: async function () {
7        var { timeline } = await fetch(this.url)
8        this.$data.chartData = this.$data.chartData.concat(timeline)
9    },
10   components: {
11     GChart
12   },
13   data () {
14     chartData: [
15       [
16         { type: 'string', id: 'BGP Speaker' },
17         { type: 'string', id: 'Label' },
18         { type: 'string', role: 'tooltip' },
19       ]
20     ]
21   }
22 }
23
24 </script>
```

To plot the actual timeline we use a Vue.js wrapper of the Google Charts plot library. We can simply pass the previously described schema and data points to the component which renders a visualization as shown in Figure 6.9. Therewith, the task of visualizing BGP messages is complete. Now, the same visualization can be created for every dataset holding BGP traffic in a fully automated and integrated way. As we can see, even in a worst-case situation where we cannot reuse a decoder, miner, or visualization module it is still feasible to derive insightful visualizations with little effort. In traditional analysis procedures, such a process might be repeated for every analysis. With DDoSGrid, we only need to invest these efforts once after which we can reuse the modules.

6.2 Usability Evaluation

With the usability evaluation we collect information on the learnability of the platform and the usability when solving actual tasks. To assess the usability, real users were asked to use the platform and rate its usability using a ten-item questionnaire. This evaluation was conducted using a methodology similar to the one presented in [63].

Figure 6.9: Timeline visualization of the mined BGP messages

6.2.1 Participants and Design

Researchers and students with an academic background were defined as a target for the usability evaluation. This target group is likely to hold the necessary background knowledge. The usability testing begins with a tutorial video. To gather profile information about the participants, we started the evaluation with a series of questions. This profile information included the users' age and the last obtained academic degree. Next, we wanted to know what the user considers his main area of expertise. Finally, we established the background knowledge of the user concerning cyberattacks with two questions. First, we presented a set of high-level cyberattacks such as Ransomware, Cross-Site Scripting, and DDoS attacks. The user had to check each item he/she was familiar with. For the second question, we applied the same procedure with a selection of DDoS attack types. This background information enabled us to interpret the test results. For example, if only a few participants consider themselves knowledgeable of DDoS attacks it may explain why they were unable to fulfill the tasks related to pattern detection.

Once the profile of the target audience was established the actual usability evaluation was conducted. This consisted of eight tasks and multiple setup tasks. To solve these tasks, the user first had to authenticate and open a dataset. Then, he/she had to use the metrics and visualizations to solve the tasks described below. Two datasets were created for the test by taking two datasets from *ddosdb.org* and adding noise. This was achieved by simulating traffic, capturing it, and concatenating the two files using a CLI tool called *mergecap* [55]. For the first dataset, we used a SYN-flood attack executed against a service listening on port 80. Some legitimate traffic sent to the same port was included to provide a level of noise. For the second dataset, we used an ICMP-flooding attack, to which we added a small number of HTTP requests.

After authenticating, the users were instructed on how to open a dataset called "Dataset 1". As apparent, the focus of this evaluation is whether the user can navigate the platform and detect the appropriate patterns indicating a cyberattack. Therefore, we left out technical

features such as importing a dataset from DDoSDB. Using "Dataset 1" the user had to solve the following tasks:

T1 **How many packets were present in the attack?**
This requires the user to look at the metrics that are displayed outside of the visualizations. The answer can be found on both the *datasets* and *dashboard* page. The correct answer is 1640892.

T2 **How many hosts participated in the attack?**
This information can be derived from the metric on the total number of source IP addresses. The correct answer is 2678.

T3 **From which country were most packets sent, considering only the 100 hosts that sent the most packets?**
This investigates whether the user can derive information on a DDoS attack on the network layer. The user is required to look at the "Top 100 sources by traffic" visualization where a world map is used to plot the countries. The correct answer is China.

T4 **Which destination port received the largest number of segments?**
This requires the user to look at the "Traffic by ports (top 20)" visualization. Therewith, we assess whether the user can derive information from the transport-layer. The correct answer is port 80.

T5 **There were only packets directed to ports in the well-known port range (1 to 1024), correct?**
This requires that the Scatterplot chart is correctly interpreted. This interpretation is supported by optional, interactive logarithmic scaling of that visualization. Alternatively, this answer can also be inferred from simply looking at the metric that defines the number of targeted destination ports. Since this number is larger than 1024, the correct answer is "no".

T6 **Which of the following attack vectors describes the attack in dataset "Dataset 1" best?**

(a) UDP-Flooding

(b) ARP-Flooding

(c) ICMP-Flooding

(d) XML-RPC Code Injection

(e) SYN-Flooding

(f) HTTP-Flooding

This question requires the user to use and combine all the previous clues to detect the pattern that indicates the attack type. From the six presented DDoS attack types only "SYN-Flooding" is correct.

After the initial six questions, the user was instructed to clear the current dashboard, open the remaining dataset "Dataset 2" and navigate back to the dashboard page. With this dataset, the user had to solve the following tasks:

T7 **Regarding the HTTP traffic in this dataset, do you consider this traffic as being part of the attack?**
With this, we want to verify two things. First, whether the user understands the visual clues which might indicate that a piece of data is just noise. Additionally, we want to see whether it is possible for the user to derive information from the application-layer. The correct answer to this question is "no" since only 0.16% of all packets contained HTTP traffic.

T8 **Looking at the metrics, which of the following attack vectors describes the attack in dataset "Dataset 2" best?**

(a) UDP-Flooding

(b) ARP-Flooding

(c) ICMP-Flooding

(d) XML-RPC Code Injection

(e) SYN-Flooding

(f) HTTP-Flooding

This question can be answered by looking at the metrics. Since 99.58% of all packets carry ICMP messages, the user can assume that ICMP-Flooding is likely the appropriate answer.

The third phase of the evaluation consisted of establishing the perceived usability on a System Usability Scale (SUS). Although this scale is not standardized, it is frequently used to establish the usefulness of software. This scale was employed for the same reason why it gained its popularity, with its ten questions it is short and simple. Also, it is said to be appropriate even with small sample sizes. To establish such a SUS the following ten questions were asked to each participant. The user expressed his/her approval on a Likert-scale: [56]

S1 I think that I would like to use this system frequently.

S2 I found the system unnecessarily complex.

S3 I thought the system was easy to use.

S4 I think that I would need the support of a technical person to be able to use this system.

S5 I found the various functions in this system were well integrated.

S6 I thought there was too much inconsistency in this system.

S7 I would imagine that most people would learn to use this system very quickly.

S8 I found the system very cumbersome to use.

S9 I felt very confident using the system.

S10 I needed to learn a lot of things before I could get going with this system.

Finally, the user could express any feedback. With that, we collected suggestions and feature requests which can be used to have an overview of possible future work of this paper. Additionally, this unstructured channel provided a possibility for the user to report eventual system errors.

6.2.2 Results

After a duration of seven days, the survey described in the previous section was evaluated. For the remainder of this section, we discuss the collected information on participants, task completion, and usability feedback.

In total, 23 participants conducted the evaluation in the course of seven days. From open feedback that users were able to submit after the evaluation and from the tasks that were solved, we found that only one participant faced technical issues using the platform. Therefore, we consider that 96% of all participants did not face any technical issues during the evaluation. To describe the audience that participated, we see that the majority of participants (74%) is between 20 and 29 years old. The second largest group (17%) were between 30 and 39 years old. Most of these users currently hold a master's degree (44%) or a bachelor's degree (39%). The remaining users (17%) have obtained a doctoral degree. Participants have described various topics as their main area of expertise. We subsume these answers into three areas. Under *Computer Science* experts we consider all participants who have not further stated knowledge of cybersecurity or network engineering. Further, we considered *Network Engineering* and *Cybersecurity* as areas of expertise. The first category was the most prominent with 52% of participants considering this their main area. Almost all of the participants currently hold a bachelor's degree, which might explain that they have not defined a more specialized field. The second category made up 38% of participants and the remaining 10% consider themselves knowledgeable about *cybersecurity*. However, when asking about specific types of cybersecurity attacks, we found that the participants consider themselves knowledgeable of most of the presented options. *Cross-Site scripting* was known by only 65% of participants. The remaining cybersecurity attacks were known by 90% or more of the participants. This includes DDoS attacks, which were known by 100% of participants. Looking at specific DDoS attacks, we see a different picture. DDoS attack vectors that are frequently observed in real life were also known by most of the participants. UDP-, ICMP-, SYN-, and DNS-flooding were known by more than 80% of participants. The fact that only 70% of participants knew about HTTP-Flooding attacks may be problematic for the interpretation of task T7, where participants were required to determine whether the apparent HTTP traffic was malicious or benign.

ID	Statement	Success Rate
T1	How many packets are present in the data set?	95.7%
T2	How many hosts participated in the attack?	91.3%
T3	From which country were most packets sent?	91.3%
T4	Which destination port received the largest number of segments?	95.7%
T5	Traffic was only sent to ports in the 'well-known port range' (1 - 1024), correct?	69.6%
T6	Which attack vector describes the attack in dataset "Dataset 1" best?	60.9%
T7	Regarding the HTTP traffic in this dataset, would you consider this traffic as being part of the attack?	78.3%
T8	Looking at the metrics, which attack vector describes the attack in dataset "Dataset 2" best?	69.6%

Table 6.1: Success rates of the evaluated tasks including test runs with system errors

To determine if the tasks were solved successfully we first consider the overall rate of correct answers. On average, tasks were solved correctly by 81% of participants. If we consider the participants who did not face technical errors, the success rate is 85%. However, there are a few notable outliers to discuss. Task T2 was criticized for how the question was formulated. Multiple participants were confused about whether destination IP addresses should be included in this result.

Task T6 and T8 had the lowest rate of being successfully answered, as shown in Table 6.1. A possible answer for this is that these questions can be considered the most difficult ones since these tasks require the user to combine multiple insights to detect the correct pattern. For task T6, 64% provided the correct answer, proving that the majority of users are able to infer the attack type using DDoSGrid. Among these users without system errors, 36% misinterpreted the HTTP traffic as malicious traffic. This traffic was incorporated into the dataset on purpose to see whether users are able to spot noise, that is benign traffic, in the attack traffic. Similarly, 73% of users managed to detect the correct attack in task T8 with the remaining 27% considering a range of other attack types.

To summarize these findings we see that DDoSGrid has a high success rate when monitoring network state and traffic insights. When it comes to detecting and identifying traffic patterns we still see that the great majority of users can use DDoSGrid for that purpose. However, as part of future work it would be beneficial to further incorporate visual clues to the visualizations. For example, a user might be looking at the Pie Chart visualization showing the distribution of HTTP verbs in HTTP traffic. If the user sees that there are millions of packets for each verb it may appear as malicious traffic to him/her. However, if one were to incorporate contextual information such as the percentage of this group of packets relative to the whole capture, it may be easier to detect that the visualized traffic is noise.

Finally, we consider how the usability of DDoSGrid was perceived by the users. Therefore, we first compute the overall score of the SUS as described in [56]. If we include the

ID	Statement	Disagree	Neutral	Agree
S1	I think that I would like to use this system frequently	4.4%	17.4% 0	78.3%
S2	I found the system unnecessarily complex.	82.6%	8.7%	8.7%
S3	I thought the system was easy to use.	4.3%	4.3%	91.3%
S4	I think that I would need the support of a technical person to be able to use this system.	78.3%	13%	8.6%
S5	I found the various functions in this system were well integrated.	0%	8.7%	91.3%
S6	I thought there was too much inconsistency in this system.	82.6%	17.4%	0%
S7	I would imagine that most people would learn to use this system very quickly.	0%	4.3%	95.6%
S8	I found the system very cumbersome to use.	78.4%	17.4%	4.3%
S9	I felt very confident using the system.	0%	17.4%	82.6%
S10	I needed to learn a lot of things before I could get going with this system.	69.5%	17.4%	13%

Table 6.2: System Usability Scale Results

participant facing a system error we obtain a score of 80,65. Considering only successful usages of the system the score would be 81,59. It is important to note that this is not a percentage and that this score needs to be normalized. In both cases, the SUS score is above 80,3 which is considered an excellent score [56].

As shown in Table 6.2, the participants responded positively to most questions. Question S10 received the lowest approval rate, for which we have multiple possible explanations. First, as we can see from the profiling of the participants, we require in-depth knowledge in areas such as computer networks, cybersecurity, and DDoS attacks specifically. For example, to correctly answer tasks T6 and T8 the user needs to have familiarity with all of the former areas. Therefore, it could be possible that users consider both the effort of learning the platform and DDoS attack patterns. Another person who strongly agreed with this question was participating on a mobile device. This may be problematic since the application was not designed for mobile usage and participants were encouraged to use a large monitor at the beginning of the questionnaire. This may also explain why this participant provided answers for some of the tasks that varied strongly from the remaining participants. Overall, we can conclude that users are satisfied with the usability of the platform. We can infer that from the high SUS score and the high success rate for the tasks, even though substantial prior knowledge is required.

Although participants were content with DDoSGrid, few suggestions were made at the end of the survey. We conclude the usability test by summarizing the main ideas. The most frequent suggestion refers to the feature to freely create a dashboard using the two

main tiles, dataset- and visualization-tiles. This feature was implemented so that users can create reports and export these at a later stage. With that, one has a presentation at hand to quickly present results and insights. However, in this survey, the tasks mainly dealt with creating insights. During that, many participants were confused by the fact that datasets were opening at the bottom. Further, many users did not like the feature to mix visualizations of different datasets. We still consider it an important feature to compare patterns across multiple datasets. Users wished in this case, for it to be clearer to which dataset a visualization belongs. As part of future work, one could implement a stronger color coding of these tiles or a feature to automatically arrange the tiles by the underyling dataset.

Another request was to present more contextual information. One suggestion was to provide for each visualization the share of the corresponding packets in relation to the whole capture. Such a feature was discussed at the beginning of this chapter. Other users wanted to have even more guidance when creating a dashboard. They want an opinionated presentation of the possible visualizations by suggesting or dissuading certain types of visualizations. For example, if there is no or little traffic corresponding to the visualization on ICMP messages, that visualization should not be accessible. For this to work, one would have to implement a classifier that decides which attack type and which type of network traffic is sensible to show. At the time of writing this paper, similar work is already in progress [64].

Finally, users suggested completely novel features. These suggestions do not aim at changing the existing functionality, but at building on-top of it. One user reasoned about the possibility to have visualizations that show the development of patterns over a series of time slices. Information for such a visualization could be easily extracted using DDoSGrid. An approach where one creates interim results based on time slices could be leveraged once the current implementation runs into performance issues if one were to analyze a larger file.

6.3 Performance Testing

In the previous sections, we have seen how DDoSGrid was evaluated with respect to the provided functionality and the perceived usability. In this section, we want to evaluate the scalability of the platform. This is necessary since we are working with volumetric data. The input data can be characterized as such since recording DDoS attacks generates large files. Therefore, we try to establish the required resources and the time it takes to provide the analysis for all input files. Therewith, we can provide system requirements and make statements about the scalability of the system. To be more specific, we try to gather data to then answer the following seven questions:

R1 What is the maximum size of a PCAP file that can be analyzed using the *miner*?

R2 Does the time required to analyse a PCAP file grow exponentially with the input file size?

R3 Does the memory consumption grow exponentially compared to the size of the input PCAP file?

R4 Is the size of a PCAP file the main aspect to consider when estimating system requirements?

R5 What are the system requirements of the *miner* package?

R6 How resource-intensive are the currently available feature extraction units in the *miner* module?

R7 We consider mainly the *miner* module, since it's the core of the platform. Considering the integrated components from the *DDoSCH*, which component can be considered a bottleneck of the integrated platform?

In the remainder of this section, we describe two things. First, the design of the testbed and how the tests are executed. Finally, we investigate the gathered results and discuss the questions that were just formulated.

6.3.1 Design and Execution

To measure the performance of DDoSGrid we perform load testing, from which we gather several metrics. Here, we focus only on the *miner* component of DDoSGrid since it is the most computationally intensive unit of the platform. Hence, this would likely be the bottleneck of the overall platform. Therefore, to increase the number of transactions, we need to look at increasing the resources based on the miner's system requirements.

To establish the system constraints and response time for the available datasets we first define two parameters frequently considered in performance testing. We define the number of transactions to be considered and the number of users who are using the system concurrently. [50] defines *Response Time* as the amount of time that passes between request and response of a transaction. For our testing, we consider the invocation of the *miner* module as a transaction. This includes providing it the PCAP file as input and expecting a set of output files. Each output file corresponds to a feature extraction module inside the *miner* module. This is important to note since decoding and analysis happen concurrently. Therefore, we run multiple iterations to determine the metrics for each feature mining module independently, in addition to a system test where all modules are executed concurrently. From our experience in testing the system with real users, we already know that our transactions provide response times in the range of multiple seconds to minutes. Considering [50] it is already clear that this is not classified as an interactive visualization, we implemented features that allow the user to return to the application once it has finished. We therefore only consider one user executing one transaction.

To execute our tests we observe multiple parameters and provide a set of different input files. First, the main result we want to derive is the response time. This would allow benchmarking the platform against similar tools. Next, we want to know about the required resources and how they scale depending on the size of the input file. If two of

such parameters are exponentially increasing we run into problems when analyzing large files. From the development of the prototype, we know that memory usage is the critical resource to observe.

Multiple input files were used to repeat the measurements. These datasets mainly differ in their size. Internally, the datasets differ strongly since they were collected by different parties for various motives. The following datasets were considered:

Anon Booter This dataset stems from a real DDoS attack that was launched against an educational institution in the Netherlands in 2015 [53]. Interestingly, this 127 GB dataset contains many large PCAP files, with the largest one being 47 GB [52].

Aposemat IoT-23 is a 97 GB dataset containing traffic captured from real IoT devices. The traffic contains both benign and malicious traffic of various cyberattack types. It is important to note that this dataset is not pruned or modified to contain DDoS attacks specifically [51].

DDoSDB.org which is a publicly accessible instance of the *DDoSDB* component provides access to many PCAP files that are labeled automatically using the *ddos_dissector* tool. Using the search engine available on DDoSDB.org we composed a 14 GB dataset containing 225 PCAP files. 67 PCAP files target the TCP protocol and 158 focus on the UDP protocol.

IoT network intrusion dataset was created for academic purposes and is largely based on simulated traffic. A range of cyberattacks from DDoS attacks to ARP spoofing is available [54]. The overall dataset is 1.5 GB in size and contains 42 PCAP files.

Considering the actual execution we use 2 GB of DDR3 memory running at a clock speed of 2.133 GHz. For processing, an Intel® Core™ i7-8650U running at a base frequency of 1.9 GHz was used. To compute the response time we analyze each input file ten times and then compute the average over all measured responses. This is done with a simple script that measures the response times and stores them. A more sophisticated load testing application seemed to provide only few benefits since we can consider very long and sequential transactions. Therefore, features such as distinguishing warm-up execution or multi-threaded request generation were not necessary.

Due to the high number of rounds and a large number of input files this first test iteration takes a long time. Once this is completed we consider the sixth question defined in section 6.3. We gather data for this question by running a complete decoding session for each feature extraction module. Essentially, we are disabling all extraction modules but one and running the analysis process in series for one PCAP file. The selection of this PCAP file is based on the results of the previous test execution. Effectively, we select a large PCAP file that has led to the longest analysis duration. The same dimensions are gathered, namely analysis duration and memory usage.

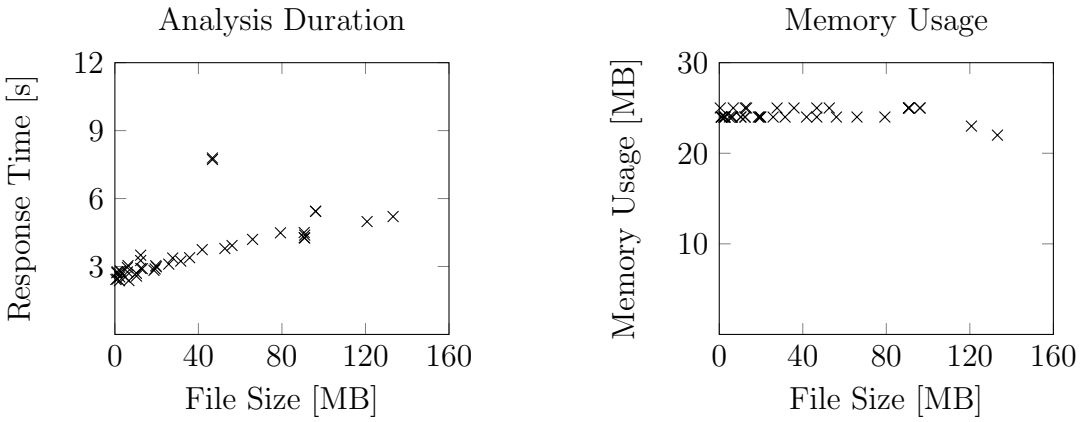


Figure 6.10: **IoT network intrusion:** Performance Testing Results

6.3.2 Results

The first dataset we considered was the **IoT network intrusion dataset**. All 42 PCAP files were successfully processed in a reasonable time. As shown in the previous section, this dataset is smaller than others. This was important so that we were able to optimize the test execution script before analyzing larger datasets. After fixing the test script and several bugs that led to memory leaks in the code we were able to analyze the remaining datasets.

The first notable insight from running the performance testing is, that once we eliminated the issues related to memory leaking, there were no system errors. As system errors we consider failures of the analysis process or an analysis duration over 30 minutes. As we will see when discussing question seven, the first issue is not uncommon for files of this size. Therefore, to revisit and answer question **R1** we have not found a file size limit that constrains the input of the application, even under constrained resources the tool was able to provide the feature extraction. We can therefore assume reasonable stability for PCAP files up to 50 GB or even larger. Combined with the fact that this was the largest PCAP file we have found references to in academic contexts, we assume that DDoSGrid is a stable tool for feature extraction, and thus, suitable for research purposes.

Let us now consider the second question **R2**, which investigated how the analysis duration correlates to the input file size of a PCAP file. This question needs to be considered within the context of a post-mortem analysis system, as defined in section 3.1. As stated before, the response time of such a system is far beyond what one could consider a real-time system. What we are trying to establish here is if DDoSGrid is scalable. For example, if the analysis duration grows exponentially with the file size of the input domain we can assume that there is a limit to the file size that can be reasonably analyzed. Regarding interactivity, it is also important to consider that we are solely testing the feature extraction module. DDoSGrid follows a principle of extracting features once and then using these features to provide an interactive visualization frontend. This is different from other applications such as Wireshark. Here, changing a display filter leads to an additional analysis process that may take a long time. Regarding the response time of the *miner* module, we see that for smaller files, such as the ones from the IoT network

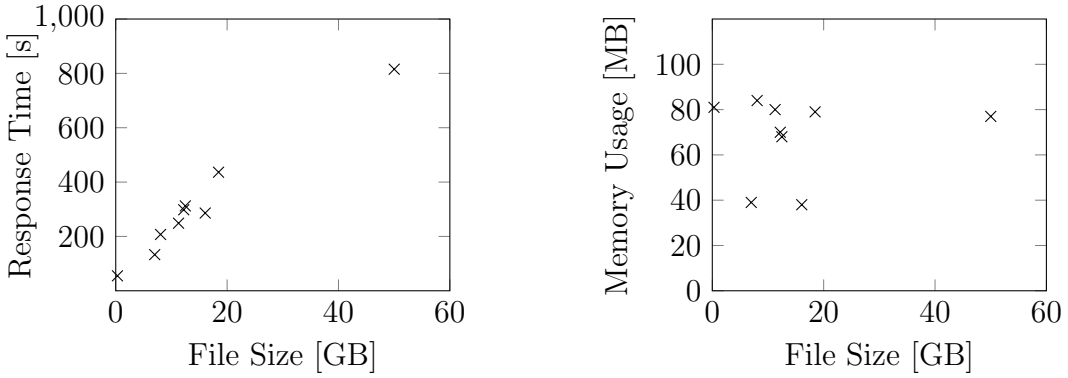


Figure 6.11: **Anon Booter:** Performance Testing Results

intrusion dataset, the response time seems to increase linearly. This can be seen in the first chart of Figure 6.10, where the input files are between 2 MB and 120 MB. Interestingly, a similar picture is visible when considering the "Anon Booter" dataset. The PCAP files in this dataset are larger, ranging from 300 MB to 50 GB. Still, we can see that the response time increases linearly, as shown in Figure 6.11. The remaining datasets, depicted in Figures 6.12 and 6.13 require a careful interpretation. In these datasets, there is a larger range in terms of file sizes. For example, in the "DDoSDB.org" dataset, the input files range from a couple of kilobytes to 1.3 GB. When looking at Figure 6.13 it appears as if response time would start to grow exponentially after the 10 MB mark. However, we have to consider that our testing does not measure purely the feature extraction process but the overall *miner* process. This includes the operations that are performed at start and end, such as input file verification or writing the results to a file. Therefore, for small PCAP files, the duration of these operations makes the overall processing time appear to be fixed. This might explain why the duration starts to increase after 10 MB. Therefore, we conclude **R2** by considering the response time to grow in sub-exponential time. In the context of a post-mortem analysis system, this shows the stability and viability of the approach even for large file sizes.

Another critical resource to consider when discussing the systems scalability and stability is memory consumption. First, let us consider the worst case, where a large file is analyzed as in the "Anon Booter" dataset. In the second chart of 6.11 we can see that memory consumption remains stable even for very large datasets of 50 GB. This is already a strong indication that the tool can be used to extract features from large files with little resources, since none of the analysis required more heap memory than 90 MB. Making a more precise statement about the correlation between input file size and memory consumption is difficult, however. For example, let us consider the results from the "Aposemat IoT23" dataset shown in Figure 6.12. Although we do not see a critically high memory consumption we do see that memory consumption is not static. We observe that files of the same size often have differing memory consumption. A possible explanation here is that the file sizes are not the critical factor, but rather the entropy of the contents of the packets in these files. For example, let us consider a feature extraction miner that collects the set of source IPv4 addresses in a PCAP file. Due to the stream-processing architecture of DDoSGrid, we only have to consider the buffering of the current packet that needs to be analyzed and the (interim-)results that are stored. Since the size of the

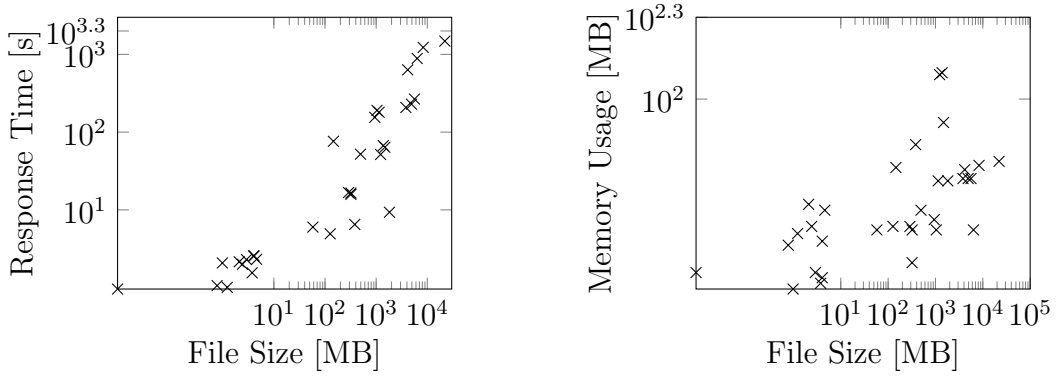


Figure 6.12: **Aposemat IoT23:** Performance Testing Results

current packet is roughly static, we need to consider the data that is being gathered from the stream. If all packets are sent from the same IPv4 address, the memory usage is a single 32-bit value and therefore very small. However, if we consider that every host of the IPv4 address space is present, at least 16 GB of memory would be required. As we can see, the entropy of the packet contents has a strong effect on the memory consumption of the miner. It is also important to consider that it would be hard to optimize such a system, since, it is theoretically not possible to collect all IPv4 addresses with less memory space. However, this does not seem to be an issue since all of the 310 PCAP files were successfully analyzed with the highest recorded memory consumption being around 1 GB. Since most of these PCAP files are from real traffic we can assume that for question **R3** memory consumption should remain stable and in many cases even static.

Considering **R4**, we do find that the input file size is an important factor when assessing the duration of the analysis. However, for memory consumption other factors such as the entropy of the packets have to be considered as well. As part of future work, one might analyze the system by providing PCAP files that depict a worst-case scenario considering the packet entropy. Then, one could investigate whether heuristic approaches can be used to improve such scenarios.

As discussed before, it is difficult to determine the resources required for a new dataset since multiple factors need to be considered. However, to provide some approximated values for question **R5** we consider that the worst-case scenario would be comparable to the PCAP file with the highest resource usage of the previously tested datasets. Since all of the files were executed in an environment limited to 2 GB of memory, we use this value as a guideline. This, together with storage space to hold the PCAP file, is the only critical requirement. The remaining resources affect the duration of the analysis but not the success of that operation. For computational resources, strong single-thread performance is important since the *miner* module runs in a single thread. Regarding I/O, the medium from which the PCAP file is consumed, it is recommended to be able to provide read speeds of 50 MB/s, otherwise, it might become the bottleneck of the application. This value was derived from the bandwidth found when analyzing the files from the "Anon Booter" dataset.

So far, we have analyzed the *miner* module as a whole. After analyzing the results from testing, we used the file "Booter 4" from the DDoS Booter attack described earlier, as the

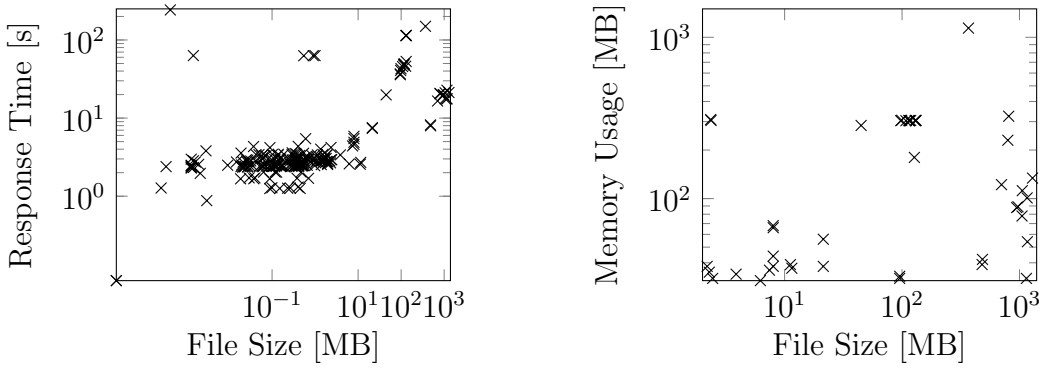


Figure 6.13: **DDoSDB.org:** Performance Testing Result

input file. The analysis of that file required almost 14 minutes in the previous experiment. With a file size of 50 GB it can also be considered the largest we have ever found online. Now, we want to find out whether there are strong differences between individual feature extraction modules. To discuss the feature extraction modules we first denote them with an identifier that is based on the OSI-layer which was targeted:

[**L1**] Top 5 VLANs by Ethernet traffic

[**N1**] Miscellaneous Metrics

[**N2**] Analysis of IPv4 vs IPv6 traffic (based on packets)

[**N3**] Distribution of ICMP Message Types

[**N4**] Top 5 source hosts (IPv4)

[**N5**] Top 100 source hosts (IPv4)

[**T1**] Top 20 UDP/TCP ports by number of segments

[**T2**] Number of segments received over all TCP/UDP ports

[**T3**] Connection states of TCP segments

[**T4**] Ratio between UDP and TCP segments

[**A1**] Most used HTTP verbs

[**A2**] Top 5 most used HTTP endpoints

[**A3**] Top 10 most used Browser and OS Combinations

[**A4**] Top 10 most used Devices

Then, we executed the whole decoding and analysis process for each feature extraction module in series. During the analysis, we gathered the same statistics as in the previous execution. That is the maximum amount of heap memory used during the decoding

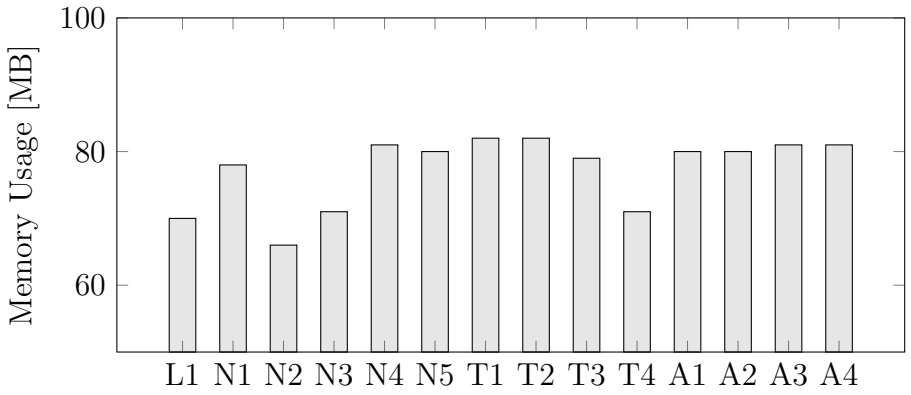


Figure 6.14: Memory Usage over all Extraction Modules

process, as well as the duration of analyzing the input file. After collecting these two metrics over all feature extraction modules we see that there are only minor differences, as visible in Figures 6.14 and 6.15. This applies to both memory consumption and analysis duration. However, there are a few subtle differences we try to explain. Let us take, for example, *N2* and *N1*. We can see that *N1* is more complex. Therefore, it requires more time and heap memory to provide the desired results. Two factors are considered here, first, the data that is extracted. *N2* simply tries to collect two counters, one for the number of IPv4 and IPv6 packets encountered. This can be done with very little memory, because a number requires only 64 bits of memory since all numbers are effectively 64-bit floating-point doubles. Compared to *N1*, this is simple since *N1* provides a number of metrics, such as the number of distinct source IPv4 addresses. To count this, it first has to gather all these addresses and store them in memory until the decoding is finished. The second factor to be considered is the data structure that is used. This has strong implications on the computational complexity and therefore on response time. In *N2* we simply increment a counter, which we can consider to be trivial from a computational standpoint. Since *N1*, on the other hand, has to store all addresses, it needs to do a lookup for each IPv4 address to increment the respective counter. This is clearly more complex than incrementing a counter in *N2*.

After observing that there are only subtle differences to be found for **R6**, we investigated whether the analysis duration can be improved by running parts of a feature miner in a different process. The desired outcome here would be to offload some of the just described computational load, such as doing lookups, to a different process. This was tested by modifying *N1*, so that the actual feature miner collects a set of relevant packets. In that case, we found that around 100 IPv4 packets must be gathered before communicating with the other process. As soon as these packets are available they are serialized into a buffer and sent to a sub-process we call *worker*. The *worker* then performs the complex functionality of doing a lookup of the addresses in the table and updating the counters. At the end of the decoding, the *worker* process receives a serialized event. Upon this event, it aggregates the metrics, which involves counting the addresses with the highest N occurrence.

Then, the *worker* communicates the result back to the feature extraction module from which it was spawned. Once it was confirmed that this implementation provides the same

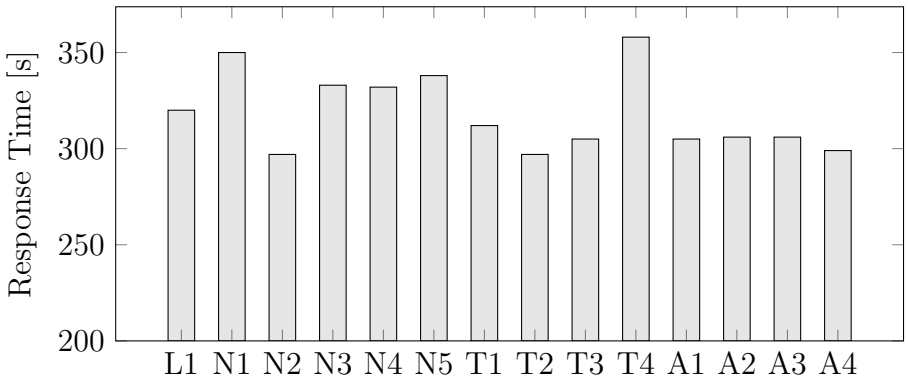


Figure 6.15: Analysis Duration over all Extraction Modules

results as running the analysis in a single thread, we used the same input file to collect the performance metrics. Regarding memory consumption, we found no notable difference. However, regarding the analysis duration, we observed that the multi-threaded solution has a duration of 539 seconds. Compared to the 350 seconds required with the single-threaded this is an increase of **54%**. Similar results were confirmed when using smaller PCAP files. With these files we were able to do more in-depth profiling of the application. We saw that it is computationally expensive to serialize data before it is sent to the *worker*. The time lost on serializing does not seem to be offset by the computational load that gets offloaded to the *worker*. However, we can still consider it beneficial to have proven that such an architecture is well possible in DDoSGrid. As part of future work, developers may write more complex analysis modules or work with PCAP files that require the lookups to become a substantial workload so that this approach becomes viable.

Finally, we consider DDoSGrid as a whole for **R7**, including the integrated components from the *DDoSCH*. To locate bottlenecks, we do not build a complex testbed to investigate the individual components. From using DDoSGrid during the development and evaluation, we already have plenty of observations to interpret. Thereform, we know that both *DDoSDB* and the *miner* modules can easily handle files that are a couple of gigabytes in size. However, there is one limiting component, the *ddos_dissector* script provided as part of the *DDoSCH*. Internally, this script uses Tshark to decode PCAP files and extract features from them. We have experienced failures from running the platform in a VM constrained to 1 GB of memory due to insufficient resources being available when using the script. The same was observed during the development of the *miner* module. Here, we searched for an alternative tool that can be used to cross-verify the correctness of the extracted features. Clear weaknesses were found when working with this tool. For example, considering the datasets from the "Anon Booter" dataset, we could not analyze one of them using the constrained memory limit of 2 GB. It is important to note that we also simplified the feature extraction using Tshark to determine only one metric at a time, namely the set of source IPv4 addresses involved. By increasing the memory limit to 16 GB, we provided results for three of the nine PCAP files in the "Anon Booter" dataset. Given that this is a bottleneck introduced into the system, we propose an additional step to rewrite the insights provided by the *ddos_dissector* using a novel feature extraction module.

Chapter 7

Summary, Conclusions and Future Work

This thesis was concerned with the research, design and integration of a platform for DDoS visualizations with the pilot of the European DDoS Clearing House. As part of that, new visualizations as well as improvements in usability, performance and security were realized. This enables the generation of insights on different types of attacks for a broadened range of users (*e.g.*, network operators, instructors and even business decision-makers).

At the beginning of this thesis, we considered two post-mortem solutions for improvement and integration. First, the DDoSCH provides features to extract DDoS attack information and also focuses strongly on sharing attack information. Secondly, we considered DDoSGrid, which provides detailed visualizations on DDoS attacks at various abstraction levels.

Exploring the current state of DDoS attacks as well as visualization and classification techniques thereof provided the basis to review a broad range of post-mortem analysis tools. We provided a literature review of 17 applications that we consider relevant for the defined classification of post-mortem analysis. These tools were then contrasted considering dimensions such as objectives, visualization techniques, automation, data sources, and state of implementation. The findings from that comparison indicate that both DDoSGrid and DDoSCH are valid approaches which improve on aspects such as the insights provided, performance, constraints, and usability. Additionally, we found that DDoSCH is the only approach that promotes information sharing between actors. This provided the necessary motivation to integrate both applications to further improve the usability and performance of the components.

That motivation and an architectural analysis were then consolidated into a set of requirements. We focused on three groups of requirements relating either to the privacy, technical implementation, or usability of the system. These requirements were implemented using three iterations. First, new interfaces were built into both systems and the data models were aligned to ensure compatibility between them. This included the adoption of an additional authorization mechanism for DDoSCH. After establishing that level of compatibility the DDoSCH components were integrated into DDoSGrid and fully automated.

At this stage, a fully automated and integrated process overarching both solutions was made available. Finally, we improved numerous weaknesses and implemented new visualizations. Improving the user interface to support the previously described integrated process concluded the usability improvements.

After implementing the integration along with the previously described improvements we considered that our prototype can be used in a context where real users can derive insights from attack data gathered in the real world. This prototype was evaluated by assessing the available features as well as their usability and performance dimensions. Our main findings suggest that the integrated platform provides a strong basis for sharing and analyzing attack data for various use cases. Examples of such use cases include auditing, teaching, investigating, and reporting attack data. Additionally, we showed that any aspect of network traffic can be decoded, extracted, and visualized. All of these use cases can be met with appropriate usability and performance making the platform a realistic application for the user-friendly analysis of large attack data. This is backed by the fact that the majority of users solved the investigative tasks and doing so found our system easy to learn and use. Similarly, no performance constraints were discovered that made the analysis of large PCAP files unfeasible.

With the summarized findings we find that the integrated platform contributes to the mitigation of DDoS attacks by providing analysis and sharing of DDoS attack data. Our evaluation suggests that this can be performed in a scalable manner. From user feedback and the success rate of real users solving tasks, we deem that the application is user friendly and that users can infer and share insights about DDoS attacks. As highlighted by Section 7.1, a powerful outcome of this work is therefore, that this platform can be used as a basis to implement other insightful processes on top.

7.1 Future Work

We outline future work using two main categories; improvements, and extensions. The former are weaknesses that were discovered during the evaluation described in Section 6. Under extensions we summarize the suggested efforts that are now feasible due to the advancements of DDoSGrid as a platform.

Regarding current limitations we can consider two sources, the findings from the performance evaluation and feedback from conducting the usability test. Focusing on performance, we clearly saw that the feature extraction and interface component provided by DDoSGrid scale well for files up to 50 GB. However, by integrating components from DDoSDB we have introduced a dependency to tshark. We found that this component is likely a bottleneck for large files. However, it is now feasible to refactor the *ddos_dissector* component to use the stream-based approach provided by DDoSGrid. Since DDoSGrid already computes a subset of these metrics we are convinced that it is possible to port the *ddos_dissector* to DDoSGrid.

A small number of users suggested changing the arrangement of visualizations. Some suggested a fixed dashboard and others requested to receive more guidance over how a

dashboard is created and arranged. The investigation of such usability related issues could be implemented during a work that aims at extending the visualization capabilities of the system.

As outlined during our discussion of requirements in Section 5.4, we enabled that the information extracted by the *converters* and *ddos_dissector* components can be used. For example, consider the *converters* component that transforms fingerprints to firewall rules. We could simply transform these rules into a syntax that is recognized by *libpcap*. Since that library is used by the decoder of DDoSGrid, we could then create a visualization where only pruned traffic is considered. Therewith, each visualization would receive a switch at the bottom giving the user a preview of how network traffic would look if the filtering rules were installed. The feasability of such a translation was demonstrated with a simple prototype of the *converters* module [67]. As future work, one would need to integrate the visual transformation of that output.

Similarly, we could use the insights contained in a fingerprint to further classify the features extracted by each feature miner. For example, color coding could be used to further separate each bar of the visualization that shows traffic towards a port based on the source address. This would allow the user to see malicious traffic. Additionally, this could be used as a recommendation engine to create dashboards according to attack types likely to be present. At the time of writing, a similar approach is taken by [64] where a machine-learning-based classifier is developed on top of this work.

Finally, we consider it vital to use this platform to build additional processes on top. An example of such an integration where DDoSGrid was used to establish an internal auditing trail in an enterprise environment was described in Section 6.1.2. A comparable approach that implements a process on top of this work is being developed by [65] at the time of writing. That work focuses on economic aspects in a collaborative setting, which is supported by the features implemented as part of this thesis.

Bibliography

[1] U. Saxena, J. Sodhi, Y. Singh: "An Analysis of DDoS Attacks in a Smart Home Networks" (2020)

[2] C. Wu, S. Sheng, X. Dong: "Research on Visualization Systems for DDoS Attack Detection," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan (2018)

[3] S. T. Zargar, J. Joshi, D. Tipper: "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks" (2013)

[4] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, A. Pras: "Measuring the Adoption of DDoS Protection Services" 2016. In Proceedings of the 2016 Internet Measurement Conference (IMC '16). Association for Computing Machinery, New York (2016)

[5] Cisco Systems, Inc.: "Cisco Annual Internet Report (2018–2023) White Paper, Updated 9.3.2020", `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html/`, Accessed: 14. August 2020

[6] G. Somani, M. Singh Gaur, D. Sanghi: "DDoS/EDoS attack in cloud: affecting everyone out there!" In Proceedings of the 8th International Conference on Security of Information and Networks (SIN '15). Association for Computing Machinery, New York, NY, USA (2015)

[7] S. Patton: "What is the real cost of a DDoS attack?", `https://www.iottechexpo.com/2018/10/iot/what-is-the-real-cost-of-a-ddos-attack/`, Accessed: 14. August 2020

[8] T. Emmons: "Largest Ever Recorded Packet Per Second-Based DDoS Attack Mitigated by Akamai", `https://blogs.akamai.com/2020/06/largest-ever-recorded-packet-per-secondbased-ddos-attack-mitigated-by-akamai.html`, Accessed: 14. August 2020

[9] O. Yoachimik: "Mitigating a 754 Million PPS DDoS Attack Automatically", `https://blog.cloudflare.com/mitigating-a-754-million-pps-ddos-attack-automatically/`, Accessed: 14. August 2020

[10] D. Freet, R. Agrawal: "A virtual machine platform and methodology for network data analysis with IDS and security visualization" SoutheastCon 2017, Charlotte, NC (2017)

[11] D. Freet, R. Agrawal: "A Statistical Comparison of Security Visualization Efficiency Compared to Manual Analysis of IDS Log Data" SoutheastCon 2018, St. Petersburg, FL (2018)

[12] DDoS Clearing House: "DDoSDB", `https://ddosdb.org/`, Accessed: 26. August 2020,

[13] B. Nagpal, P. Sharma, N. Chauhan, A. Panesar, "DDoS tools: Classification, analysis and comparison" 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi (2015)

[14] S. Triantopoulou, D. Papanikas, P. Kotzanikolaou: "An Experimental Analysis of Current DDoS attacks Based on a Provider Edge Router Honeynet" 10th International Conference on Information, Intelligence, Systems and Applications (IISA), PATRAS, Greece (2019)

[15] A. Bhardwaj, G. V. B. Subrahmanyam, V. Avasthi, H. Sastry, S. Goundar: "DDoS attacks, new DDoS taxonomy and mitigation solutions — A survey" International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), Paralakhemundi (2016)

[16] S. Jin, D.S. Yeung: "A covariance analysis model for DDoS attack detection". IEEE International Communication Conference (ICC04). 4. 1882 - 1886 Vol.4. (2004)

[17] A. Girma, M. Garuba, J. Li, C. Liu: "Analysis of DDoS Attacks and an Introduction of a Hybrid Statistical Model to Detect DDoS Attacks on Cloud Computing Environment" 2015 12th International Conference on Information Technology - New Generations, Las Vegas, NV (2015)

[18] R. Khattak, S. Bano, S. Hussain, Z. Anwar: "DOFUR: DDoS Forensics Using MapReduce" 2011 Frontiers of Information Technology, Islamabad (2011)

[19] S. Nandi, S. Phadikar, K. Majumder: "Detection of DDoS Attack and Classification Using a Hybrid Approach" Third ISEA Conference on Security and Privacy (ISEA-ISAP), Guwahati, India (2020)

[20] G. Jaafar, S. A. Ismail: "Review of Recent Detection Methods for HTTP DDoS Attack" Journal of Computer Networks and Communications (2019)

[21] S. T. Zargar, J. Joshi, D. Tipper: "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks" in IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2046-2069 (2013)

[22] R. Marty: "AfterGlow | AfterGlow | Link Graph Visualization | Project Home", `http://afterglow.sourceforge.net/`, Accessed 02.09.2020

[23] A. Kalwar, M. H. Bhuyan, D. K. Bhattacharyya, Y. Kadobayashi, E. Elmroth, J. K. Kalita: "TVis: A Light-weight Traffic Visualization System for DDoS Detection" 14th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP), Chiang Mai, Thailand, pp. 1-6 (2019)

[24] R. Marty: Applied Security Visualization, First Edition, Addison-Wesley Professional (2008)

[25] H. Kim, S. Ko, D. S. Kim, H. K. Kim: "Firewall ruleset visualization analysis tool based on segmentation" 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), Phoenix, AZ, 2017, pp. 1-8 (2017)

[26] Z. Jiawan, Y. Peng, L. Liangfu, C. Lei: "NetViewer: A Visualization Tool for Network Security Events" 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, Hubei, pp. 434-437 (2009)

[27] K. Lakkaraju, W. Yurcik, A. Lee: "NVisionIP: netflow visualizations of system state for security situational awareness" VizSEC/DMSEC (2004)

[28] DDoS Clearing House: "DDoSDB", `https://ddosdb.org/about`, Accessed 03.09.2020

[29] R. Poortinga - van Wijnen, J. M. Ceron, J. Santanna: "ddos-clearing-house/ddosdb: Software responsible for extracting DDoS characteristics", `https://github.com/ddos-clearing-house/ddosdb`, Accessed 03.09.2020

[30] J. M. Ceron: "ddos-clearing-house/converters: Converters: fingerprints to mitigation tools rules", `https://github.com/ddos-clearing-house/converters`, Accessed 03.09.2020,

[31] J. Santanna, J. M. Ceron, K. van Hove, J. van der Ham, N. Shah: "ddos-clearing-house/ddos_dissector Software responsible for extracting DDoS characteristics", `https://github.com/ddos-clearing-house/ddos_dissector`, Accessed 09.09.2020

[32] Wireshark Foundation: "8.8. The I/O Graphs Window", `https://www.wireshark.org/docs/wsug_html_chunked/ChStatIOGraphs.html`, Accessed 03.09.2020

[33] L. Boillat, J. von der Assen, M. Franco, C. Killer: "A Tool for Visualization and Analysis of Distributed Denial-of-Service (DDoS) Attacks", Communication Systems Group, Department of Informatics, Universität Zürich (2020)

[34] L. Boillat, J. von der Assen: "ddosgrid/ddos-visualization: Dashboard visualization system for DDoS mining", `https://github.com/ddosgrid/ddos-visualization`, Accessed 18.09.2020

[35] Z. Kan, C. Hu, Z. Wang, G. Wang and X. Huang: "NetVis: A network security management visualization tool based on treemap" 2010 2nd International Conference on Advanced Computer Control, Shenyang, pp. 18-21 (2010)

[36] M. Rouse: "What is Web Application (Web Apps) and its Benefits", https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app, Accessed 04.09.2020

[37] The GGobi Foundation, Inc.: "Documentation. GGobi data visualization system", http://ggobi.org/docs/, Accessed 04.09.2020

[38] E. Simonsen: "glTail.rb - realtime logfile visualization", http://www.fudgie.org/, Accessed 04.09.2020

[39] J.G. Conrads: "DDoS Attack fingerprint extraction tool: making a flow-based approach as precise as a packet-based" (2019)

[40] Django Software Foundation: "The Web framework for perfectionists with deadlines", https://www.djangoproject.com/, Accessed 18.09.2020

[41] OpenJS Foundation: "Express - Node.js web application framework", https://expressjs.com/, Accessed 18.09.2020

[42] E. You: "Vue.js", https://vuejs.org/, Accessed 18.09.2020

[43] D. Hardt: "The OAuth 2.0 Authorization Framework", https://tools.ietf.org/html/rfc6749, Accessed 18.09.2020

[44] J. M. Ceron, R. Poortinga - van Wijnen: "ddos-clearing-house/dddosdb-in-a-box: DDoSDB system embedded in a virtual machine", https://github.com/ddos-clearing-house/dddosdb-in-a-box, Accessed 18.09.2020

[45] Merriam-Webster.com Dictionary: "Usability | Definition of Usability by Merriam-Webster", https://www.merriam-webster.com/dictionary/usability, Accessed 23.09.2020

[46] Merriam-Webster.com Dictionary: "Privacy | Definition of Privacy by Merriam-Webster", https://www.merriam-webster.com/dictionary/privacy, Accessed 23.09.2020

[47] A. Johnson, O. Yiu: "zestedesavoir/django-cors-middleware: Django app for handling the server headers required for Cross-Origin Resource Sharing (CORS)", https://github.com/zestedesavoir/django-cors-middleware, Accessed 14.10.2020u

[48] Jazzband: "Welcome to Django OAuth Toolkit Documentation — Django OAuth Toolkit 1.3.2 documentation", https://django-oauth-toolkit.readthedocs.io/en/latest/index.html, Accessed 14.10.2020

[49] Mozilla Contributors: "File - Web APIs | MDN" https://developer.mozilla.org/en-US/docs/Web/API/File, Accessed 22.10.2020

[50] I. Molyneaux: "The Art of Application Performance Testing", O'Reilly Media (2009)

[51] A. Parmisano, S. Garcia, M. J. Erquiaga: "Aposemat IoT-23: A labeled dataset with malicious and benign IoT network traffic", https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/, Accessed 18.11.2020

[52] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, A. Pras: "Booters - An analysis of DDoS-as-a-Service Attacks", `https://www.simpleweb.org/wiki/index.php/Traces#Booters_-_An_analysis_of_DDoS-as-a-Service_Attacks`, Accessed 18.11.2020

[53] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, A. Pras: "Booters — An analysis of DDoS-as-a-service attacks" 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa (2015)

[54] H. Kang, D. H. Ahn, G. Min Lee, J. Do Yoo, K. Ho Park, H. Kang Kim: "IoT network intrusion dataset" IEEE Dataport (2017)

[55] S. Renfro, B. Guyton: "mergecap(1) - Linux man page", `https://linux.die.net/man/1/mergecap`, Accessed 20.11.2020

[56] usabiliTEST: "System Usability Scale online with analytics | usabiliTEST", `https://www.usabilitest.com/system-usability-scale`, Accessed 20.11.2020

[57] Y. Rekhter, T. Li, S. Hares: "A Border Gateway Protocol 4 (BGP-4)", Request for Comments: 4271 (2006)

[58] Cloudflare, Inc.: "HTTP Flood DDoS Attack | Cloudflare" `https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/`, Accessed 09.12.2020

[59] M. Franco, E. Sula, B. Rodrigues, E. Scheid, B. Stiller: ProtectDDoS: "A Platform for Trustworthy Offering and Recommendation of Protections", International Conference on Economics of Grids, Clouds, Software and Services (GECON 2020), Izola, Slovenia, (2020)

[60] Computerworld.ch: "Delegierter für Cybersicherheit will Meldepflicht - computerworld.ch", `https://www.computerworld.ch/security/netzpolitik/delegierter-cybersicherheit-meldepflicht-2546187.html`, Accessed 09.12.2020

[61] I. Steiner-Gashi: "Meldepflicht für Opfer von Hacker-Angriffen | kurier.at", `https://kurier.at/wirtschaft/meldepflicht-fuer-opfer-von-hacker-angriffen/400032193`, Accessed 09.12.2020

[62] B. Rodrigues, M. Franco, G. Parangi, B. Stiller: "SEConomy: A Framework for the Economic Assessment of Cybersecurity", 16th Conference on the Economics of Grids, Clouds, Systems, and Services (GECON 2019), Leeds, UK (2019)

[63] M. Franco, R. L. dos Santos, A. Schaeffer-Filho, L. Granville: "VISION – Interactive and Selective Visualization for Management of NFV-Enabled Networks", IEEE 30th International Conference on Advanced Information Networking and Applications (AINA 2018), Crans-Montana, Switzerland (2018)

[64] L. Boillat: "DDoSGrid-Mining: Analyzing and Classifying DDoS Attack Traffic" (In preparation, 2021)

[65] F. Chao, W. Qiaowen, X. Xianxiao: ”Design and Implementation of a Collaborative System for Sharing Insights and Economic Impacts of Cyberattacks” (In preparation, 2021)

[66] MELANI/GovCERT.ch: ”Measures to counter DDoS attacks”, https://www.melani.admin.ch/dam/melani/en/dokumente/2015/04/Massnahmen_gegen_DDoS_Attacken.pdf.download.pdf/Measures_to_counter_DDoS_attacks.pdf (2015)

[67] J. Ceron, J. von der Assen: ”ddosgrid/converters-translators”, https://github.com/ddosgrid/converters-translators, Accessed 18.12.2020

Abbreviations

AAA	Authentication, Authorization, Accounting
ACK	Acknowledgement Flag (TCP)
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
CD	Compact Disc
CLI	Command Line Interface
CORS	Cross-Origin Resource Sharing
CSV	Comma-Separated Values
DDoSCH	DDoS Clearing House
DDoS	Distributed Denial-of-Service (Attack)
DNS	Domain Name System
FIN	Connection Teardown Flag (TCP)
GB	Gigabyte
GHz	Gigahertz
GML	Graph Modelling Language
GTk	GIMP Tool Kit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
ICMP	Internet Control Message Protocol
ID	Identifier
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JS	JavaScript
JSON	JavaScript Object Notation
MB	Megabyte
MD5	Message Digest Algorithm Five
NCSC	National Cyber Security Centre
NFV	Network Function Virtualization
NLP	Natural Language Processing

OAuth	Open Authorization
OSI	Open Systems Interconnection
OS	Operating System
PCAP	Packet Capture
PoC	Proof of Concept
PPS	Packets Per Second
RFC	Request For Comments
RPC	Remote Procedure Call
SHA	Secure Hash Algorithm
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SUS	System Usability Scale
SYN	Synchronize; Connection Setup Flag (TCP)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
WebGL	Web Graphics Library
XML	eXtensible Markup Language

Glossary

Throughout this work we assumed a target audience that is familiar with computer networking and cybersecurity-related concepts. Therefore, the following glossary only focuses on terms that may be unclear in this context or that have an ambiguous meaning. Additionally, we restate terminology that is used to refer to components in the DDoSGrid prototype.

Anonymization Removing data that reveals information on one of the communicating hosts from a packet.

Buffer Fixed-length sequence of bytes represented by the built-in Node.js Buffer class.

Callback Function defined by the client that is invoked once an event is triggered by the decoder in DDoSGrid.

Consumer Software component that takes a dataset as an input to produce output from its content.

Controller Software component implementing the flow of a software by modifying a model.

Dataset In this work, a dataset considers a PCAP or a series of PCAP files stemming from the same continuous packet sniffing procedure.

Dissector Software component that extracts a set of metrics from a dataset.

Django Python-based web framework that follows an architectural pattern similar to the MVC pattern.

Feature miner Software component in DDoSGrid that observes packets emitted by the packet decoder according to some specification. This component needs to implement all DDoSGrid extraction life cycle methods.

Fingerprint Set of metrics that provide identification and classification of a (DDoS) attack.

Code Injection We refer to code injection in a cyber security context instead of a software engineering context. Here, malicious software is executed on the victim's premises.

Intelligence The process of collecting information to provide value for economic, military or political aspects.

Multivector Attack A complex cyber attack that compromises more than one attack type.

Packet Decoder DDoSGrid software component of the *miner* package that reads a Buffer and produces data in a well-defined data structure that reveals the semantics of the packet.

PCAP, Packet Capture A packet capture may refer to the act of sniffing packets or the output thereof. We consider the latter, producing a PCAP file as part of an offline session established using the *libpcap* library.

Upstream Referring to the ISP of a service for an enterprise or another (smaller) ISP.

Visualization In DDoSGrid, we use this term to refer to a Vue.js component that implements the transformation of extracted features to a visualization.

List of Figures

List of Tables

Appendix A

Installation Guidelines

In these installation guidelines we differentiate between an infrastructure for development and for production use. We first describe the procedures that need to be done when setting up DDoSGrid and DDoSDB locally. Then, we outline how both systems need to be set up in a realistic scenario where we assume a VM running the Debian distribution.

A.1 Development environment: DDoSGrid

As a prerequisite for all the components, the user needs to have the repository that home the three major components. This can be done using the contents of the CD or by cloning from our GitHub repository:

```
1 git clone https://github.com/ddosgrid/ddosgrid-v2.git
```

Additionally, the user needs to have the libraries and tools installed for which we refer the user to the offical documentation:

1. Node.js
2. npm
3. git
4. SSH agent
5. Python 3.6 or higher
6. libpcap

A.1.1 Miner

To develop using the *miner* package we suggest to use the CLI interface during development. For that, one needs to navigate into the *miner* directory and fetch the dependencies:

```
npm i
```

If one wants to use the experimental BGP packet decoders, one needs to use a special fork of the *node_pcap* library. This can be installed directly from GitHub:

```
npm i git+ssh://github.com:ddosgrid/node_pcap.git
```

At this point, one may invoke the *miner* by invoking the CLI as follows. Additonally, one may add the `--inspect` flag to debug the Node.js process.

```
node index.js pcap_path=/path/to/pcapfile
```

During the analysis of a PCAP file, the *miner* process emits information about the current stage of the analysis and the feature extraction modules that are enabled. Additionally, basic performance information is displayed by regularly showing the current heap memory being used and by displaying the response time of the stages:

```
Input check completed
Analysis started
Set up of the following miners has completed (0.5s):
 - Top 5 VLANs by Ethernet traffic
 - Miscellaneous Metrics
 - Top 20 UDP/TCP ports by number of segments
 - Number of segments received over all TCP/UDP ports
 - Connection states of TCP segments
 - Ratio between UDP and TCP segments
 - Analysis of IPv4 vs IPv6 traffic (based on packets)
 - Distribution of ICMP Message Types
 - Top 5 source hosts (IPv4)
 - Top 100 source hosts (IPv4)
 - BGP Messages
 - Most used HTTP verbs
 - Top 5 most used HTTP endpoints
 - Top 10 most used Browser and OS Combinations
 - Top 10 most used Devices
Decoding has started...
     6 * 10^6 PCAP packets analysed. Current Heap Memory usage: 24MB
Decoding has finished (38s), starting post-parsing analysis
Post-parsing analysis of the following miners has completed:
 - (0.0002s)   Top 5 VLANs by Ethernet traffic
 - (0.0001s)   Miscellaneous Metrics
 - (0s)  Top 20 UDP/TCP ports by number of segments
 - (0.0001s)   Number of segments received over all TCP/UDP ports
 - (0s)  Connection states of TCP segments
 - (0.0001s)   Ratio between UDP and TCP segments
 - (0s)  Analysis of IPv4 vs IPv6 traffic (based on packets)
 - (0s)  Distribution of ICMP Message Types
 - (0.0767s)   Top 5 source hosts (IPv4)
 - (0.0583s)   Top 100 source hosts (IPv4)
 - (0.0001s)   BGP Messages
```

```
  - (0.0002s)   Most used HTTP verbs
  - (0.0002s)   Top 5 most used HTTP endpoints
  - (0.0003s)   Top 10 most used Browser and OS Combinations
  - (0.0002s)   Top 10 most used Devices
All miners have finished.
```

To perform a more in-depth performance testing of the *miner*, one should have a look at the *performance-testing* directory, which contains a number of scripts to automate this process. The `evaluation.sh` script accepts three parameters. First, a descriptor of directories to be used, for which all the PCAP files will be analysed in series. Additionally, one needs to supply the name of this analysis and the number of iterations for each PCAP file. For example, let us consider a dataset directory that has many subdirectories with many PCAP files. We want to smooth the performance testing results by using two iterations. In that case we recommend to run the script as follows:

```
./evaluation.sh '/path/to/extracted/dataset/**/*.pcap' analysisname 2 >
    stdout.log 2> stderr.log &
```

As we can see, standard output and error are redirected to logfiles and the process is forked to the background. This then creates a CSV file containing all the results in a folder called *results/analysisname/output.csv*. The schema of that file is as follows:

```
File,    Size, Res Time 1, Res Time 2, Avg Response, Memory Usage

xy.pcap, 100,  20,         22,         21,           25MB
yz.pcap, 200,  40,         42,         41,           26MB
.. 23 hidden
```

As we can see, there is one row for each PCAP file found, along with its name, file size, response times and memory usage. For large files this may take a long time since each PCAP file is analysed sequentially for the number of iterations. To inspect the current state, one may look at the stdout of that script:

```
tail -f stdout.log

25 PCAP files were found in your dataset
Evaluation started.
analyzing /path/to/extracted/dataset/**/*.pcap as analysisname and
   analysing each file for 2 rounds
Starting..  xy.pcap
============================================================> (2/2)
Starting..  yz.pcap
============================================================> (2/2)
Starting..  za.pcap
============================================================> (1/2)
```

Once completed, the CSV can be analysed to investigate the scalability of novel features such as feature miners. If desired one can transform these findings directly into LaTeX diagrams by using the following script that requires the path of the CSV and the number of iterations used during the analysis:

```
node create_Latex_Charts.js ./results/analysisname/output.csv 2
```

A.1.2 web API

The *miner* is integrated with the DDoSCH components by the web API found in the *api* directory. Before starting the server, the *miner* needs to be set up. After that, we can set up the API which consists of three steps. First, we fetch the direct dependencies of the web API:

```
npm i
```

Then, we need to make sure that both the *converters* and *ddos_dissector* scripts can be executed as a child process by the API. We clone these into the top-level folder of the project and fetch their dependencies:

```
cd ddosgrid-v2

# Fetch and set up converters
git clone git@github.com:ddosgrid/converters.git
cd converters
pip3 install -r requirements.txt

# Fetch and set up dissector
git clone git@github.com:ddosgrid/ddos_dissector.git
cd ddos_dissector
pip3 install -r requirements.txt
```

At this point, we have all static resources to run the API. However, we need to define the parameters so that the API is configured to talk with a DDoSDB instance for authentication and information sharing. We'll ignore all the available environment variables for now, since these are described in more detail in the production deployment. For development purposes we recommend to reuse the variables contained in a development script which can be run with `npm run dev`. These use an OAuth2 application ID and token of the publicly deployed DDoSDB instance on the CSG VM:

```
#!/bin/bash
export PORT=8080
export CLIENT_APP_ORIGIN=http://localhost:8081
export OAUTH2_AUTHORIZE=https://www.csg.uzh.ch/ddosgrid/ddosdb/o/
    authorize/
export OAUTH2_TOKEN=https://www.csg.uzh.ch/ddosgrid/ddosdb/o/token/
export OAUTH2_CLIENTID=NnOKS0...mGmDHk
export OAUTH2_CLIENTSECRET=bHAsk2lpd...2TKsrJ4qx4IUqVkT
export OAUTH2_CALLBACK=http://localhost:8080/auth/provider/callback/
export OAUTH2_SUCCESS_FORWARD=https://localhost:8081/ddosgrid/
export DDOSDB_PROFILEINFO=https://www.csg.uzh.ch/ddosgrid/ddosdb/api/
    profileinfo
export DDOSDB_HOST=www.csg.uzh.ch
export DDOSDB_PCAPEXPORT=https://www.csg.uzh.ch/ddosgrid/ddosdb/
export DDOSDB_FILTEREXPORT=https://www.csg.uzh.ch/ddosgrid/ddosdb/api/
    upload-filter_rules
export DDOSDB_ATTACKTRACE_PATH=/ddosgrid/ddosdb/api/attack-trace


node index.js
```

If one wants to use a different DDoSDB instance as an authorization server, the appropriate URLs need to be changed. Additionally, one needs to register the authorization endpoint of the API in that DDoSDB instance and use the resulting secret and ID appropriately. Normally, a developer can simply run that script which spawns the server listening on port 8080:

```
# Runs ./scripts/start_dev_server.sh
npm run dev

App is listening on port 8080
```

A.1.3 Frontend

Finally, we set up the DDoSGrid frontend. For that, we navigate to *frontend* and fetch the dependencies using `npm i`. After that, we can run the development web server using `npm run serve` which brings up a server that automatically restarts when the codebase changes:

```
DONE  Compiled successfully in 4313ms
  App running at:
  - Local:   http://localhost:8081/ddosgrid/
  - Network: http://192.168.88.225:8081/ddosgrid/

  Note that the development build is not optimized.
  To create a production build, run yarn build.
```

As we can see, the server is listening on port 8081 to match the `CLIENT_APP_ORIGIN` parameter of the server.

A.2 Development Environment: DDoSDB

We already described how to set up the *ddos_dissector* and *converters* modules as part of the web API. One can follow these instructions and simply run the scripts in the CLI to work on them. Similarly, we advise the user to follow the documentation which can be found in the *README.md* file of the DDoSDB directory. If one wants to use DDoSDB together with the web API, he/she needs to set up an OAuth2 application for the web API. This is described in detail in the section on production deployment.

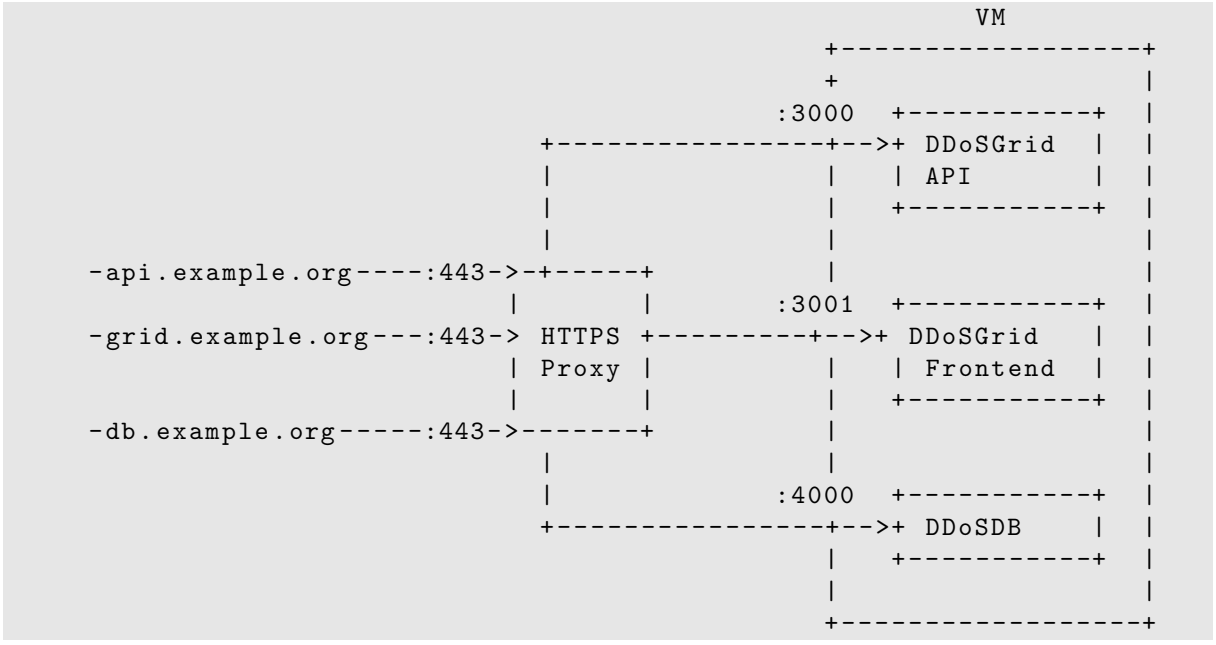
A.3 Production Deployment (Automated)

To provide a more convenient way to deploy and manage a productively running instance we want to point the reader to the *deployment* folder. This folder allows to deploy all components in an automated way using Ansible and Vagrant. This script fetches all sources directly from the version control systems. For this reason, we do not consider this

deployment version representative of this work, since it is not deterministic. For example, as part of future work, other developers may push features to the repository. Therefore, the outcome of the deployment script depends on the state of the (remote) repository. However, this confiruation still presents a "working version" of this documentation. For more information on how the scripts can be used we direct the user to the *README* file which can be found in that folder. Another contribution of this deployment variant is that it is idempotent so that one can use it for the provisioning of a CI/CD pipeline. That would involve triggering the playbook whenever the remote repository receives changes.

A.4 Production Deployment (Manual)

Let us now see how we can deploy DDoSGrid and DDoSDB as an integrated platform to a virtual machine. Regarding system requirements, we suggest considering the findings of section 6.3. The scenario described below was successfully used on a VM running Debian 10 with 1GB of memory and 1 virtual CPU. Three ports are required by the VM to which an HTTP reverse proxy forwards requests from three domains as shown here:



Keep in mind that some proxies set a file size limit for HTTP requests. This parameter needs to be set appropriately, depending on the size of PCAP files that you are working with. Additionally, a prerequisite of this deployment is that Docker, including docker-compose and git, are set up beforehand. Once this is given we can set up the three stacks, which may cover multiple services. Setting up DDoSDB along with the dependent services like the database can be done in just a few statements since all components are ported to Docker.

A.4.1 DDoSDB

If we do not use HTTP paths in our reverse-proxy we first need to remove these paths from `ddosdb/ddosdb/website/settings.py`.

After that, we can simple run `docker-compose up` to build and deploy images of the Django app, housekeeping scripts and ElasticSearch search engine. If the DDoSDB instance should listen on a different port one needs to modify the `command` property of the `docker-compose.yml` file. During the first start, or, whenever the search engine should be cleared we can use the Docker service holding the housekeeping scripts to set it up:

```
1 docker exec -it ddosdb_housekeeping bash
2 /app/ddosdb-docker.db
```

In a similar way, we can use the DJango scripts to create a new admin user:

```
1 docker exec -it ddosdb bash
2 /app/manage.py createsuperuser
```

Now, DDoSDB is running on proxied URL `db.example.org` and we can log in using the newly created user. To set up the remaining services, we also need to create an OAuth2 secret and ID for the web API of DDoSGrid. This can be done in the admin interface located at `https://db.example.org/admin/oauth2_provider/application/`. Here, we need to define the user, a public client type and code-based authorization type. Additionally, we need to define acceptable redirect URLs for the web API. For our example, we should use `https://api.example.org/auth/provider/callback/`. Save the registration and note the client secret and ID. These are required for the set up of the web API.

A.4.2 DDoSGrid API

To deploy the API of DDoSGrid, we first have to clone the DDoSGrid project as already described before. Additionally, we need to fetch the two modules provided by the DDoSCH to the top-level folder of the repository. This was already described in the development set up. Therefore, it makes sense to first set up that and then continue with the production deployment. Doing so can be done by parametrizing the application using the `docker-compose.yml` file. We explain the main environment variables that need to be set for this example.

```
1  - DDOSDB_PROFILEINFO=https://db.example.org/api/profileinfo
2  - DDOSDB_PCAPEXPORT=https://db.example.org/ddosdb/
3  - DDOSDB_ATTACKTRACE_PATH=https://db.example.org/api/attack-trace
4  - DDOSDB_FILTEREXPORT=https://db.example.org/ddosgrid/ddosdb/api/upload-
       filter_rules
5  - DDOSDB_HOST=db.example.org
6  - DDOSDB_ATTACKTRACE_PATH=/api/attack-trace
7  - OAUTH2_AUTHORIZE=https://db.example.org/o/authorize/
8  - OAUTH2_TOKEN=https://db.example.org/o/token/
9  - OAUTH2_CLIENTID=123456789
10 - OAUTH2_CLIENTSECRET=abcdefgh123456789
```

```
11
12 - OAUTH2_CALLBACK=https://api.example.org/auth/provider/callback/
13
14 - PORT=3000
15 - CLIENT_APP_ORIGIN=https://grid.example.org/
16 - OAUTH2_SUCCESS_FORWARD=https://grid.example.org/
```

As we can see in the snippet above, the first five variables tell the API how to reach various services of the DDoSDB API. Make sure that URLs pointing to and endpoint within DDoSDB end with a slash. Next are the two OAuth2 parameters that were established in the previous section when the DDoSGrid API instance was registered as a client application in DDoSDB.

The `OAUTH2_CALLBACK` tells the API how it can be reached externally. This is necessary since the API needs to ask the DDoSDB authorization server to forward back to this URL later. As we can see, this needs to match the redirect URL previously registered. Finally, we tell the API on which port to listen and from which Origin it may accept CORS requests. We should specifically consider the Origin of the frontend, a wildcard would make most browsers hide cookies from the API or aborting CORS requests altogether.

Finally, we wish to configure the frontend sub-directory at the URL previously defined in the API as an allowed origin. However, we first have to define the single variable `VUE_APP_APIBASEURL` in `.env.production`.

Also, we need to tell it how it can link to datasets on DDoSDB, therefore we set the `VUE_APP_DDOSDBBASEURL` to `https://db.example.org/ddosdb` This has to be done at compile-time, since the frontend eventually runs in the users browser, where we cannot set any environment variables. In this example, we simply set it to `https://api.example.org/`. After compilation, the static frontend is served by NGINX. Therefore, we modify the NGINX config in `default.conf` and set `server_name` to the host of our URL, which is `grid.example.org`.

If one wants to repeat the usability evaluation in the future one may also set the variable `VUE_APP_DEMOUSER` to the e-mail address of a user that exists within DDoSDB. If that user logs in, he only has the option to open visualizations and create a dashboard. Actions such as uploading or deleting a dataset are not available. The specified variable needs to be placed in `.env.production` of the `frontend` project.

Once these parameters are defined for the API and frontend, a simple `docker-compose up` builds all components and deploy them according to the diagram.

Appendix B

Contents of the CD

A number of resources are supplied along with this report. As part of the **DDoSGrid** project/folder, we provide the following sub-folders:

Artifact	**Description**
ddosgrid-v2/api	DDoSGrid web API
ddosgrid-v2/frontend	Vue.js app
ddosgrid-v2/miner	Feature extraction app
node_pcap	The fork of the underlying libpcap binding. This contains the extended decoder that implements the BGP protocol
Performance_Testing	This holds the scripts and results to evaluate the *miner* package and the results that were used in the performance testing chapter.
git.zip	Git archive of the repository. This holds experiments in their respective branches, *e.g.*, *bgp-messages* for BGP Message analysis and *worker* for multithreaded analysis.

Table B.1: DDoSGrid artifacts

As the goal of this thesis was the integration with DDoSCH components, we also provide extended components based on that project as part of a folder called **DDoSCH**:

Artifact	Description
DDoSDB	fork that implements OAuth2 and a new RESTful API
converters	script including filter upload
converters_experiment	script including libpcap filter translation experiment/PoC
ddos_dissector	aligned to the new stateless web API

Table B.2: DDoSCH artifacts

Additionally, we provide the following items related to the **usability testing**, so that the experiment could be repeated:

Artifact	Description
Datasets	Two PCAP files that were used during the evaluation
Questions	Questions and Instructions given to participants. Since Google Forms does not provide an official export functionality we provide a JSON of the questions and a printed PDF of the page
Tutorial	Video given to each participant to explain usage of the platform
Results	CSV file containing the full answers given by the participants

Table B.3: Artifacts from the Usability Test

Further, we provide a folder **deployment** which covers a number of scripts and resources to create a VM running a productive version of the platform. This includes an HTTPS reverse proxy and an Ansible-based automted deployment script. Such an automated deployment provides an almost guaranteed success of deploying the system, thereby eliminating issues when developing future work. Additionally, it serves as a working documentation of the installation instructions.

Artifact	Description
HTTPS reverse proxy	Reverse proxy that takes connects the various services which expose an HTTP endpoint
Configuration Management	Ansible-based deployment script. Includes a Vagrantfile to setup a VM which is then provisioned using Ansible. A README.md documents the prerequisites and how the playbook may be executed.

Table B.4: Artifacts for Deployment and Configuration Management

In the **report** folder we hold the LaTeXsources and images used to compile this report and presentations.

Finally, we provide the sources and PDF files of the midterm and final presentation. This can be found in the **presentation** folder of the CD.