University of Zurich[UZH]

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

Distributed Analysis of Cyberattacks in a Collaborative Setting

Marion Dübendorfer
Zurich, Switzerland
Student ID: 16-927-501

Supervisor: Jan von der Assen, Muriel Franco
Date of Submission: March 1, 2022

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

ifi

Bachelor 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

Seit Beginn des 21. Jahrhunderts stellen DDoS-Angriffe eine große Bedrohung dar für die Verfügbarkeit von Dienstleistungen, die mit dem Internet verbunden sind, da sie weitreichende Auswirkungen auf Unternehmen, Organisationen und die Gesellschaft als Ganzes haben können. Da die Häufigkeit, das Volumen und der Schweregrad von DDoS-Angriffen kontinuierlich zunehmen, sind sowohl in der Forschung als auch in der Industrie Applikationen zur Analyse von DDoS-Angriffen entstanden. In dieser Arbeit werden gegenwärtige Programme untersucht, die Analyse- und Schutzdienste für DDoS-Angriffe auf der Grundlage von Netzwerkverkehrsanalysen anbieten. Ausserdem wird das Fehlen verteilter, kollaborativer Eigenschaften in diesen Tools diskutiert. Das Hauptziel dieser Arbeit ist es, einen Prototyp zu entwerfen und zu implementieren, der diese Eigenschaften erfüllt. Dies geschieht durch die Erweiterung der Architektur von SecGrid, einer Plattform für die Extraktion, Verarbeitung und Analyse von Cyber-Attacken anhand einer Post-Mortem-Methode. Die im Rahmen der Evaluation des Prototyps durchgeführten Fallstudien legen nahe, dass die vorgestellte Lösung die verteilte und kollaborative Analyse von Cyberangriffen ermöglicht, während die Skalierbarkeit und Nutzbarkeit des SecGrid-Systems erhalten bleibt. Die im Rahmen dieser Arbeit durchgeführte Performance-Analyse deutet darauf hin, dass die Skalierbarkeit in bestimmten Anwendungsfällen sogar erhöht werden kann.

Abstract

Since the early 2000s, DDoS attacks pose a major threat to the availability of services connected to the internet, as they can have far-reaching impacts on businesses, organizations, and society as a whole. As DDoS attacks continue to grow in frequency, volume, and severity, DDoS attack analysis systems have emerged both from research and industry. This thesis examines current tools that provide DDoS attack analysis and protection services based on network traffic analysis, and discusses the lack of distributed, collaborative features present in these tools. The main goal of this thesis is to design and implement a prototype that fulfills these features. This is done by extending the architecture of *SecGrid*, a platform for the extraction, processing, and analysis of cyberattack traffic in a post-mortem fashion. The case studies conducted as part of the evaluation of the prototype suggest that the presented solution enables the distributed and collaborative analysis of cyberattacks, while preserving the scalability and usability of the SecGrid system. The performance evaluation conducted as part of this thesis suggests that in certain use cases, scalability can even be increased.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Jan von der Assen, for the continuous support, expert feedback, and interesting discussions throughout this thesis. I would also like to thank my co-supervisor, Muriel Franco, for his inputs.

I would also like to thank Prof. Dr. Burkhard Stiller for the possibility to complete my Bachelor thesis at the Communication Systems Group (CSG) of the University of Zurich.

Finally, I thank my friends and family for proofreading this thesis, and for their constant support.

Contents

Chapter 1

Introduction

Since the early 2000s, Distributed Denial-of-Service (DDoS) attacks pose a major threat to the Internet availability and are one of the biggest concerns for cybersecurity [1]. DDoS attacks are malicious actors' attempts to prevent legitimate users from accessing information systems, devices, or other network resources [2],[3]. They happen every single day and continue to grow in frequency, volume, and severity, despite the great commercial and research efforts put into mitigation solutions [4]. To mitigate these attacks, the detection and classification of their attack vectors (*i.e.*, the path, method or mechanism used by an attacker to carry out an attack) are vital [5]. For that, visualization systems present an efficient approach to detect patterns in network traffic [6]. The goal of this thesis is to improve an existing DDoS attack analysis and visualization system, enabling the distributed, collaborative analysis of cyberattacks. The following chapter introduces the motivation for such a system, as well as the structure of this report.

1.1 Motivation

The impacts of DDoS attacks are diverse. From a commercial perspective, they can result in large financial losses and damage in reputation for companies [7]. Beyond that, an attack can have a far-reaching impact on society. The telecommunication industry, as well as the financial, educational, and health sectors are frequent targets of DDoS attacks [8]. If a DDoS attack aimed at such a target is successful, it can result in a population losing access to essential infrastructure. For instance, attacks on financial services can potentially disrupt payment transactions, and attacks on healthcare institutions threaten the seamless provision of medical services.

Ideally, a DDoS attack is mitigated as early as possible. It is considered most effective to mitigate the attack before it even happens, or in other words, to prevent the attack from happening at all [2]. Naturally, there is a great amount of cloud-based DDoS protection services offered by third-party providers such as Cloudflare, Akamai, Radware, and Imperva [9],[10]. These protection services mainly focus on preventive measures as well as attack detection and reaction, thus minimizing attack surfaces as well as limiting

damage in the case of an attack. However, in order to keep up with the constant evolution in size, type and complexity of DDoS attacks, research efforts to create DDoS attack analysis tools are required. [7] proposed such an analysis tool with the implementation of DDoSGrid 2.0 as part of the SecGrid project [11]. DDoSGrid 2.0 combines the components of two different platforms to visualize data of DDoS attacks based on network traffic data, and share said data on a collaboratively created database. This provides the capability of sharing the data of DDoS attacks and providing different abstraction levels with visualizations for collaborators [12].

Having the possibility of retrieving multiple data sources from a collaboratively created database opens new directions for further research. DDoS attacks affect various actors and networks. For example, two services may be affected by the same cyberattack at the same time. Similarly, the same cyberattack may be observed by multiple intermediary systems. For instance, an upstream Internet Service Provider (ISP) may see the traffic of the same attack, although the ISP is not the victim. This motivates the visualization of cyberattacks using multiple perspectives from distributed collaborators. However, the implementation of DDoSGrid 2.0 does not support that data is aggregated in an automated manner. Instead, data needs to be merged manually. Furthermore, it does not preserve semantics of having multiple contributors to the analysed data. This motivates the implementation of a tool that enables multiple collaborators to analyze interrelated network traffic of DDoS attacks in a distributed manner.

1.2 Description of Work

The goal of this thesis is to design and develop a prototype that enables multiple collaborators to analyze interrelated network traffic of attacks in a distributed manner. The existing implementation of DDoSGrid will be extended to provide a basis for the distributed analysis of attacks. To achieve this goal, the work in this thesis is divided into multiple activities.

First, an extensive literature review is provided to study the characteristics of DDoS attacks and their related attack vectors, in order to gain thorough knowledge about the theoretical background of this work. As a subsequent task, existing DDoS attack analysis tools, or tools with a more general purpose that can be leveraged as DDoS attack analysis tools, will be investigated. A particular focus is laid on the ability of these tools to provide collaboration and the possibility to employ the tool in a distributed setting. The goal of this step is to establish which tools have emerged from research and industry, and how they differ from the requirements that are established in this thesis.

Next, the current implementation of the DDoSGrid platform is thoroughly investigated, in order to gain the technical knowledge about the tool necessary to be able to extend its functionality. After this goal is fulfilled, a feasible architecture of the extended system is proposed. When the architecture is defined, the technologies that are to be used for the implementation are evaluated, and the prototype is implemented on the basis of these technologies.

Lastly, the implemented prototype is evaluated by demonstrating its effectiveness and provided additional value on the basis of several case studies. This step demonstrates how the prototype can be used as a distributed attack analysis tool in a collaborative setting.

1.3 Thesis Outline

This report is structured as follows. Chapter 2 provides the theoretical basis for this thesis and introduces three topics required to understand DDoS analysis systems based on network traffic data. Chapter 3 introduces related work by reviewing existing network analysis tools that can be leveraged as DDoS attack analysis systems. First, several criteria for the classification of a *post-mortem DDoS attack analysis system* are defined. Then, the tools mentioned above are analysed based on those criteria. In Chapter 4, the requirements established in Chapters 1 and 3 are discussed in more detail and the architecture for the prototype is defined, the development of which will be described in Chapter 5. Finally, the evaluation of the implemented prototype in Chapter 6, and a summary in Chapter 7 conclude the report.

Chapter 2

Background

In this chapter, a comprehensive overview over the underlying concepts of a collaborative DDoS analysis tool is given. First, the concept of DDoS attacks as well as the various possibilities of DDoS attack mitigation are investigated. Second, the concept of network traffic analysis, which can be utilized for attack analysis and mitigation, is introduced. Lastly, the implementation and functionality of SecGrid, the tool whose functionality will be extended in this thesis, is discussed.

2.1 DDoS Attacks

A denial-of-service (DoS) attack is a malicious actors' attempt to prevent legitimate users from accessing information systems, devices, or other network resources [2],[3]. This is done by generating an amount of traffic big enough to overwhelm the victim's infrastructure and rendering the resource inaccessible. The motivation behind such an attack is usually based on ideological or political belief, economic competition, cyberwarfare, or simply an intellectual challenge [2].

Accordingly, in a distributed DoS (DDoS) attack, an attacker compromises a number of network-connected hosts, installs some sort of malware on these hosts and thus utilizes them to build an attack army which executes the actual DoS attack. These attack armies are called *Botnets*. The bigger the attack army is, the easier it is to overwhelm the victim's service. The large number of unsecured devices connected to the Internet and their growing processing capacity allows for attackers to take control of a vast amount of devices, ranging from connected cameras to smart fridges, to launch malicious attacks [2]. Many of these devices are insecure by design and not often impossible to be secured due to their hardware and software constraints. A well-known example of a Botnet is the Mirai Botnet. Starting in September 2016, by exploiting the wide use of default passwords in IoT (Internet of Things) products, the Mirai malware managed to build a Botnet consisting of a stable population of 200'000 to 300'000 infected devices in order to launch massive DDoS attacks [13].

2.1.1 Types of DDoS Attacks

Classification Based on OSI Model

With respect to the OSI model, DDoS attacks can be classified based on the layer on which the attack is targeted. Usually, the protocols of the application, transport or network layer, which represent the 7th, 4th and 3rd layer, respectively, are utilized for DDoS attacks [2]. Application layer attacks usually target the HTTP protocol, using legitimate HTTP requests to overwhelm the victim's resources [14]. In transport layer attacks, either the UDP protocol or TCP protocol is used [15]. In the case of the latter, the three-way handshake process of the TCP protocol is exploited. One way to do so is by spamming the victim with SYN packets, receiving the SYN/ACK packets of the victim, but never sending the ACK packets, resulting in many open port connections [16]. At some point, this prevents the targeted service from responding to requests efficiently, or at all. Similarly, the UDP protocol can be exploited by sending a large amount of spoofed UDP packets to the victim, forcing the victim to check and respond to each packet, potentially exhausting the victim's resources [17]. Lastly, in the case of the network layer, attacks are carried out using the ICMP or IP protocol [18].

Direct and Indirect Attacks

However, DDoS attacks can not only be categorized based on the layer on which they occur. They can also be categorized into direct and indirect attacks. In direct attacks, the attacker, or usually a botnet on their behalf, sends large amounts of packets to the victim directly [19]. Usually, the attacker uses spoofed source addresses to conceal the source of the attack and thus make its mitigation more difficult [20]. A direct DDoS attack can utilize any protocol described above, or a mixture of them [19].

Indirect attacks are also called reflector attacks. As the name suggests, according to [21], a large amount of third-party, legitimate devices are used to act as reflectors. The attacker, or usually a botnet on their behalf, sends spoofed requests to the reflectors. Instead of providing their own source address, they provide the address of the victim. Therefore, the reflectors send their response packets to the victim, generating an overwhelming amount of traffic. This implies that any device that is able to respond to requests is a potential reflector. Moreover, [21] consider amplification attacks as variation of reflector attacks, where the reflectors are selected such that the reply sent to the victim is many times larger with respect to the amount of data being sent, compared to the preceding request.

Multi-Vector DDoS Attacks

A significant development in DDoS attacks is reflected in multi-vector DDoS attacks. Instead of utilizing only one attack vector, like for instance a SYN flood attack, several vectors are utilized, either at the same time, or one shortly after the other. Usually, various layers are targeted. While this kind of DDoS attack is considered to be the new normal [22], they are harder to mitigate than conventional attacks. First, given

the various attack vectors, the volume of attack data is colossal. Second, while attack mitigation systems can nowadays effortlessly mitigate a single-vector attack, they struggle to mitigate multi-vector attacks due to their complex nature and their constant evolution [23].

2.1.2 DDoS Attack Mitigation

DDoS attack mitigation is "the process of successfully protecting a targeted server or network from a DDoS attack", as defined in [24]. In order to mitigate an attack, defense mechanisms are deployed. A suitable way to classify defense mechanisms is by the point in time at which a mechanism is deployed, relative to the time when a DDoS attack occurs [2].

A defense mechanism can be deployed before an attack, in order to prevent it. According to [25], preventive mechanisms attempt to prevent DDoS attacks from happening altogether, or to endure the attack while preventing a denial of service to legitimate clients.

During an attack, attack detection is achieved if attack signs are present on a service and they are picked up either by a monitoring system or by human operators. Available detection techniques include detecting anomalies in traffic data and tracing attack sources [26]. After the attack has been detected, suitable mitigation techniques are deployed in order to minimize downtime of the service under attack. Naturally, these techniques vary depending on the type of DDoS attack vector, and on which OSI-layer the attack occurs (*e.g.*, network layer, transport layer, or application layer). For instance, in the case of an application layer attack, enhancing firewall rules is a valid approach to attack mitigation, whereas attacks on other layers require different traffic filtration techniques [24].

The DDoS mitigation process continues after an attack. While it is not possible to mitigate the current attack after it has finished, the data gathered in the previous mitigation steps can be analysed in order to take countermeasures and derive new preventive mechanisms. This in turn helps mitigating future attacks.

2.2 Network Traffic Analysis

Network traffic analysis is the process of observing and analysing network activity on a given interface and it is applied in various sub-domains of computer science. For instance, possible applications are network problem identification and troubleshooting systems in the area of network engineering, or anomaly detection systems as part of security engineering [27].

Flow export data and packet capture (PCAP) data are often used as data sources for network traffic analysis. Flow export is an approach where packets are aggregated into flows and exported for storage and analysis [28]. NetFlow, a network protocol created by Cisco, defines a flow as follows: A flow is an array of packets with mutual properties like packet header fields such as source and destination IPs, port numbers, and timestamps,

that pass through a network device, where they are collected and exported [28],[29]. Similarly, packet capture is a procedure where complete packets are captured. The packet capture output can be stored in PCAP-formatted files. Since packet capture exports all packets in their entirety, as opposed to flow exports, they provide more detail than flow exports [28]. In contrast, flow exports are a preferable approach when used in high-speed networks, since they significantly reduce the amount of data to be stored [28]. There is a vast amount of network traffic analysis tools, both open-source and commercial, among which WireShark, Arkime, and SolarWinds are popular. In addition, conventional monitoring tools such as the ELK stack can be leveraged for the purpose of network traffic analysis.

2.3 SecGrid

The SecGrid platform is an open-source platform that offers a process for cyberattack traffic extraction, processing and analysis in a post-mortem fashion. It aims to provide insightful data and visualizations for a better understanding of different types of cyber-attacks [12]. DDoSGrid is a runnable instance of the SecGrid platform, and thus the two names are used interchangeably in this work. The platform consists of three main components: the analysis layer, the database layer, and the user interface [12]. The analysis component together with the database component comprise the data layer, whereas the user interface comprises the user layer.

The analysis component takes PCAP data as input. It handles the traffic analysis process, during which the PCAP parser extracts packets based on their protocol and transmits these packets to the miners. The miners of SecGrid are software components that are responsible for the extraction of information from PCAP data. They process individual packets in order to extract features from different protocols. For instance, a miner specific to the HTTP protocol will collect statistics about the distribution of HTTP verbs (*e.g.*, `GET`, `PUT`, `POST`) used in requests. Table 2.1 shows some examples of miners, along with the features they extract and the layer they target according to the OSI model. After the analysis of the input data has finished, each miner runs a post-parsing analysis in order to structure the data and extract features for insights in the form of visualizations or manual data inspection. The analysis process is either invoked via a Command Line Interface (CLI) or the user interface.

The user interface is a web-based interface that allows for the user to upload PCAP data, and after the analysis process is completed, interact with it. The main component of the interface is the visualization module, which enables the user to interact with a subset of the analysed data and transform it into visualizations, such that the user can gain insights about the data in a user-friendly manner. Every analysis is stored in the database. Therefore, the user can access and inspect previous analyses at any point in time, and the analysis only needs to be done once for each dataset.

Due to the open-source nature of the platform and the modular architecture of the analysis layer, the platform shows high extensibility. Additional miners can be implemented and easily integrated into the analysis workflow, along with the existing miners. In addition,

Miner	Extracted Features	OSI Layer
Metrics Analyzer	General metrics (attack duration, number of packets, IPs and ports)	Network Layer
ICMP Messages	Overview of ICMP message types	Network Layer
IP Version	Analysis of IPv4 to IPv6 ratio	Network Layer
Top Source Hosts	Overview of the hosts sending large amounts of requests	Network Layer
UDP vs TCP Ratio	Analysis of ratio between UDP and TCP packets	Transport Layer
TCP States	Analysis of distribution of TCP flags (*e.g.*, ACK, SYN, and FIN)	Transport Layer
Browser and OS Analyzer	Analysis of browser and operation system combinations used for requests	Application Layer
HTTP Verbs	Analysis of distribution of HTTP Verbs (*e.g.*, GET, PUT, POST) in requests	Application Layer
HTTP Endpoints	Overview of endpoints accessed in HTTP requests	Application Layer

Table 2.1: Excerpt of Miners Implemented by SecGrid [11]

the platform is scalable in terms of data handling, since packets are analysed in a stream-based manner [12]. Instead of processing or storing large amounts of packets at a time, the analysis layer inspects single packets and only the resulting statistics are saved in memory. Therefore, the platform can run without specific hardware requirements.

Chapter 3

Related Work

This chapter describes the analysis of existing network analysis tools that can be used as post-mortem DDoS attack analysis tools. First, some criteria based upon which the analysis is conducted are presented. Then, several traffic analysis tools are introduced, reviewed, and compared on the basis of the defined analysis criteria.

3.1 Taxonomy

This thesis focuses on the improvement of an existing DDoS attack visualization and analysis tool. The proposed tool is a "*Post-mortem DDoS Attack Analysis System* with a target audience of academic teachers, researchers, and forensic personnel from the industry", as defined in [7]. In this thesis, the basis for a distributed analysis of DDoS attacks is developed. On the one hand, this allows for better scalability of the analysis procedure. Moreover, it enables a collaborative approach to post-mortem attack analysis, allowing for several collaborators to contribute data to the analysis.

Based on this description of the tool, several criteria can be defined on which the analysis of existing tools is performed:

- The general **purpose** of a tool is analysed. For instance, it is possible that a network analysis tool that is used for real-time network monitoring could be leveraged as a post-mortem DDoS attack analysis tool, even if that is not the main purpose of the tool.
- A tool should consist of **Open-Source Software (OSS)** and be **free of charge**. The first enables the users to comprehend how data is analysed and provides the possibility to extend the functionality of the tool, if necessary. The latter ensures the accessibility of the tool, meaning that anyone can have access to it.
- It is important that the tool provides **visualizations**, such that the targeted audience can interact with the provided data and derive knowledge from it.

- Similarly, to accommodate the technical affinity of the target audience, the tool must exhibit a certain **usability** and the **complexity** of the tool must be low.
- The supported **data sources** need to be investigated. Since the tool must support a post-mortem analysis approach, a tool that only enables real-time packet capture and analysis can not be considered sufficient.
- **Scalability** is an important aspect that the tool should provide, in order to satisfy the requirement of a collaborative analysis with multiple sources of data.
- To allow for a distributed analysis of DDoS attacks, the tool must support the **collaboration** of multiple contributors.

3.2 Existing Network Analysis Tools

Now that the features relevant to our analysis are defined, they are applied on a set of existing traffic analysis tools.

Arkime (formerly Moloch) is an open-source packet capturing system [30]. It consists of three main components: the capture component, the viewer component, and the database and search component. The capture component is an application that monitors network traffic, writes PCAP formatted files, parses the captured packets, and sends metadata to the database component. The viewer component is a Node.js application which handles the web interface and transfer of PCAP files. The viewer supports visualizations of time series in the form of basic line and bar charts. However, more detailed visualizations are not available. Arkime's architecture is designed to be deployed across multiple clustered systems, and is therefore highly scalable [30]. It is possible to share data via the web interface, providing a basis for a collaborative approach. However, it is not possible to aggregate data from different collaborators and visualize the aggregated data.

Wireshark is a similar network analysis tool. It is a popular, open-source network traffic analyzer that allows for live capture and offline analysis [31]. It consists of two components: A graphical user interface implemented in C++ and a component for packet capture and filtering that uses libpcap and npcap. As an alternative to live capture, the user can upload existing PCAP files. The captured or uploaded network data can be inspected and filtered via the user interface. The interface also provides some basic functionality for visualizations. However, the visualization tools are more suitable for monitoring or anomaly detection rather than manual investigation and knowledge derivation. The available statistics include the number of packets captured, the duration of the packet stream, and the average number of packets per second, among others [32]. In terms of scalability, Wireshark is not designed to be deployed across a cluster or across network boundaries, and the application is single-threaded. Furthermore, it is not possible to aggregate data from different PCAP files, which hinders the collaboration of multiple contributors.

Elk Stack (Elastic Stack) is a collection of three open-source products: Elasticsearch, Logstash, and Kibana [33]. Elasticsearch is a search engine, Logstash is a data collection component, and Kibana is a visualization component [34]. Kibana can transform the input

data into any type of visualization such as bar charts and pie charts. Since Logstash is not able to handle PCAP files, an external tool such as Wireshark, or its CLI tshark, is needed to convert PCAP files into JSON format [35]. The Elk Stack is highly scalable and scales vertically and horizontally. Logstash and Kibana do not offer any functionality for aggregating data from multiple sources. Thus, in order to enable a collaborative approach, the data fed into Logstash would already need to be aggregated. Additionally, setting up a traffic analysis system using the Elk Stack is rather complex since communication between the three components has to be ensured and additional tools need to be incorporated into the system.

IVRE is an open-source network recon framework [36]. It is written in Python with a MongoDB backend. It relies on several external tools, such as Nmap, Masscan, Nfdump, and Zeek (Bro), to scan networks and sniff packets. The results can be browsed and filtered using Ivre's CLI tools, Python API, or Web interface. The web interface provides powerful visualizations in the form of bar charts, world maps, and scatter plots, among others. The accepted input file format largely depends on the external tool used for packet capture. For instance, analyzing PCAP files is supported by Zeek and Nfdump, but not by Nmap or Masscan. IVRE is not designed to be deployed across a cluster, and thus scalability is limited. The ability to aggregate traffic data from multiple collaborators is dependent on whether the external tool used for packet capture supports this functionality. Overall, the general purpose of IVRE lies in intrusion detection, monitoring, and site reliability engineering (SRE) [37] rather than traffic analysis in a post-mortem fashion.

SolarWinds Network Performance Monitor (NPM) is a commercial network monitoring and network performance analysis tool [38]. The tool can be used as a real-time network packet capture tool for packet analysis, traffic identification, and network data monitoring [39]. However, it is not possible to use PCAP files or any offline data as a data source for post-mortem analysis. It has a user interface that provides powerful visualizations of the captured data in the form of bar charts, pie charts, and world maps. NPM is highly scalable within network boundaries. It is not possible to aggregate and visualize packet capture data of different sources or applications.

3.3 Comparison and Summary

Having introduced several network analysis tools that can be utilized as post-mortem DDoS analysis tools, it becomes apparent that these tools differ in the fulfillment of our criteria. Table 3.1 highlights the general purpose as well as the accepted data sources. Table 3.2 summarizes to what extent each tool fulfills the criteria. In both tables, SecGrid is listed, showing properties the tool already possesses, as described in chapter 2.3, or properties this work aims to achieve, discussed in chapter 1.2. In the following section, a more detailed description about the fulfillment of the criteria is given.

Considering the general purpose of each tool, it becomes evident that none of the tools are intended specifically for the post-mortem analysis of cyberattacks. Instead, most of them can be described more generally to be network analysis and packet capture tools, or even general-purpose monitoring tools not focused on network analysis, specifically. However,

most tools accept PCAP files as data source, especially in offline mode as opposed to real-time capture. Therefore, these tools are conceptually capable to be used as post-mortem attack analysis tools, even if that is not their main purpose. Only SolarWinds NPM is not suitable to be used for this purpose, since it only allows real-time analysis. A similar argument applies to the Elk Stack. While post-mortem analysis is theoretically possible, Elk Stack is rather intended to be used for real-time monitoring. This also reflects in the fact that NPM and Elk Stack do not accept PCAP files as data sources per default, in contrast to the other tools. An upside of these general-purpose monitoring tools is their scalability, since they are intended do be deployed across networks, as opposed to Wireshark, for instance, which is intended to analyse the traffic of only one machine.

The user interfaces of most of the tools are rather complex and their operation requires certain technical skills. This is probably due to the fact that many tools offer a wide array of functionalities, and the user has the possibility to utilize these functionalities to their own specific needs. Moreover, the visualizations can oftentimes only be generated by interacting with data manually, which also requires specific technical skills and violates the usability criteria.

	General Purpose	Data Sources
Arkime	Packet capture and search system	Network interface, PCAP (real-time, offline)
Wireshark	Packet capture and analysis	Network interface, PCAP (real-time, offline), other capture formats
Elk Stack	Monitoring and data analysis tool	JSON files, log files (real-time)
IVRE	Network recon and analysis	Network interface, PCAP (real-time, offline) (depends on external tool used)
SolarWinds NPM	Network monitoring and performance analysis	Network interface, PCAP (real-time)
SecGrid	Post-mortem cyberattack traffic analysis	PCAP (offline)

Table 3.1: Overview of surveyed network analysis tools

Considering the collaboration feature, it becomes clear that the aggregation of data from multiple collaborators is not considered a necessary functionality for any of the tools, as Arkime is the only tool that provides a data sharing and collaboration feature to some extent. This is most likely due to the fact that the main purpose of these tools is either packet capture or network monitoring in a real-time fashion. That being said, for most tools, a collaboration feature which enables multiple collaborators to aggregate and analyse data in a post-mortem fashion does not lie in the scope of their general purpose.

To summarize, none of the analysed tools defines post-mortem analysis of DDoS attacks as their primary purpose. Most of the tools could theoretically be leveraged as post-mortem analysis tools, but the fact that this is not their intended purpose is reflected in the lack of visualization or collaboration functionality as well as the accepted data sources. As a result, none of the tools fully satisfy the defined analysis criteria.

	OSS	Free	Usability	Visualizations	Scalability	Collaboration
Arkime	✓	✓	✗	◐	✓	◐
Wireshark	✓	✓	✓	◐	✗	✗
Elk Stack	✓	✓	✗	✓	✓	✗
IVRE	✓	✓	?	✓	✗	?
SolarWinds NPM	✗	✗	◐	✓	✓	✗
SecGrid	✓	✓	✓	✓	◐	✓

Table 3.2: Comparison of functionality. ✓= provides functionality, ✗= does not provide functionality, ◐= provides functionality to some extent, ? = unknown.

The tools Arkime and Wireshark focus on real-time packet capture and manual analysis, and therefore do not provide automatically generated visualizations or visualizations that are mainly intended for humans to gain knowledge about attack data. More precisely, one could argue that Arkime and Wireshark can be used to detect anomalies in network data, for instance a spike in traffic, which can be an indicator for a DDoS attack. However, to gain more detailed information about the potential attack, the data has to be investigated manually, which violates the usability criteria

The tools Elk Stack, IVRE and SolarWinds NPM, on the other hand, focus mainly on automated network monitoring. Therefore, they focus on real-time analysis of large amounts of data, rather than offline data analysis. They offer powerful, automatically generated visualizations, but due to the real-time monitoring approach, they can not be leveraged for the analysis of PCAP data in the form of file inputs.

Chapter 4

The Distributed DDoSGrid Approach

The previous chapter has given a basic overview over the requirements of a distributed, collaborative DDoS attack analysis tool. In this chapter, a feasible architecture for this approach, from here on also referred to as *the distributed DDoSGrid approach*, or *D-DDoSGrid*, is presented. In addition, the key features and the added value of this approach are highlighted. This architecture proposition will then be used as a baseline for the implementation of the prototype in Chapter 5.

4.1 Architecture

The current architecture of the SecGrid analysis component is depicted in Figure 4.1. In the scope of one analysis, a single, centralized instance of the analysis component is instantiated. After the analysis has finished, the extracted features are stored on the same device, where they can be used for further processing. The visualization of the current architecture not only highlights the centralized approach, but also the fact that no collaboration is possible within the analysis lifecycle.

The proposed D-DDoSGrid architecture is visualized in Figure 4.2. In this architecture, the distinction is made between a central node and a worker node. Due to the distributed nature of the architecture, multiple worker nodes can be instantiated in the lifecycle of one analysis. Each worker node runs an instance of the analysis component, containing the packet decoder, the protocol parser, and the miners.

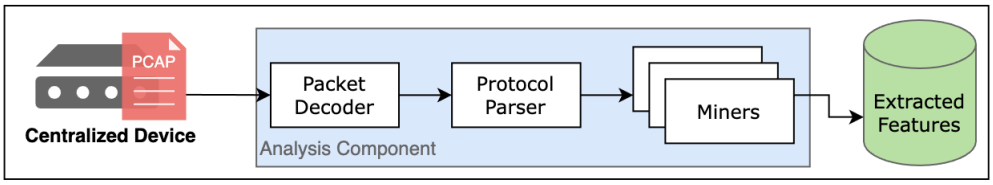

Figure 4.1: Centralized Analysis Component Architecture

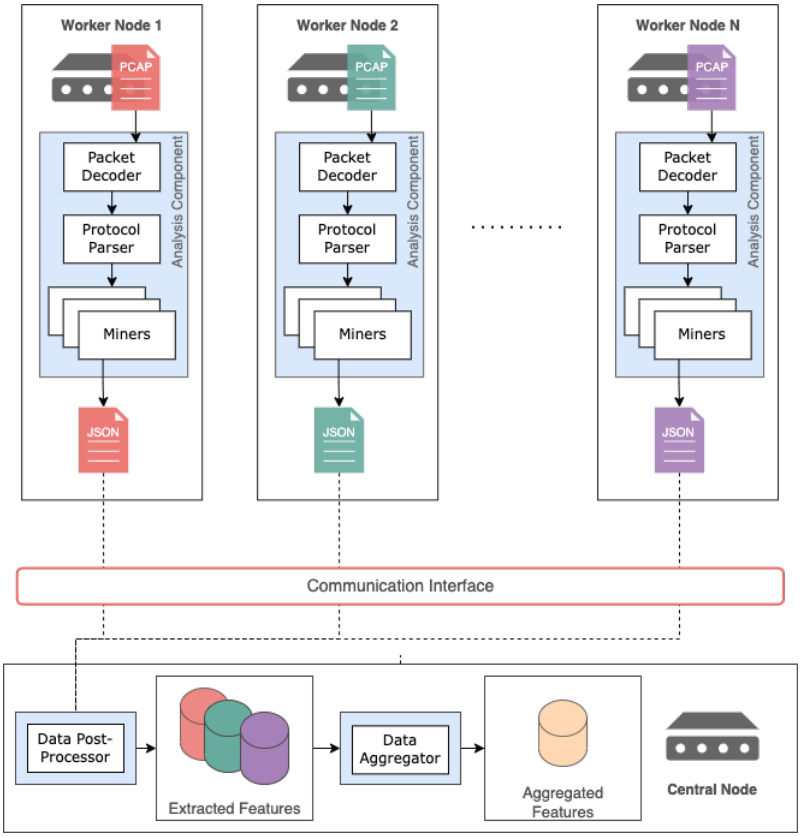

Figure 4.2: D-DDoSGrid Architecture

In contrast, only one central node is launched per analysis cycle. Two data management components on the central node are responsible for managing the data produced by the worker nodes. First, the data processor component runs a post-processing mechanism that prepares the extracted features for further processing. Specifically, it ensures that the extracted data is formatted such that it is compatible with the data formats of SecGrid. This enables integration of the distributed DDoSGrid approach into SecGrid's visualization module. Second, the data aggregation component takes the results of the various analyses as input, aggregates those results, and makes use of the post-processing component to format the aggregated data.

The communication interface enables the exchange of data between worker nodes and the central node. Since the PCAP files are processed on the corresponding worker nodes, only lightweight metadata in JSON format is exchanged. Therefore, the communication interface does not constitute a bottleneck, and thus any number of worker nodes can be instantiated and perform an analysis. After the analysis, the extracted and aggregated features are stored on the central node for inspection and further processing.

4.2 Key Features

The previous section has brought forward a feasible architecture as foundation for the implementation of D-DDoSGrid. Now, the main features and improvements this archi-

tecture entails are discussed. For each feature, a short definition is given first, followed by a paragraph that shows to which extent the current SecGrid system implements the feature. Lastly, it is discussed in more detail how the D-DDoSGrid architecture achieves this feature.

4.2.1 Collaborative Analysis

In the context of D-DDoSGrid, collaboration is defined as the participation of several independent actors in a certain task. More precisely, with respect to SecGrid being a post-mortem cyberattack analysis tool, collaborators need to be able to contribute datasets to an analysis. In this setting, any device that is in possession of a suitable dataset and that is able to run the code necessary to build the SecGrid platform, or any person that is the operator of such a device, can be referred to as a collaborator.

In the current implementation of SecGrid, collaboration is limited. The analysis component is designed such that only one dataset can be analysed in the scope of a single analysis. In the user interface component, it is possible to perform multiple analyses consecutively, and then compare the results of the individual analyses in the form of visualizations. However, this entails high manual effort. Additionally, there is no data aggregation mechanism in place in the current implementation.

In the distributed DDoSGrid approach, collaboration is achieved by allowing multiple independent collaborators to submit their datasets to the same analysis instance. Furthermore, the analysis component of the SecGrid system is able to handle multiple data inputs in the scope of a single analysis. It is essential that no data originating from the datasets is lost, such that the amount of insights the actors can gain from the collaborative approach is maximized. Therefore, the analysis system contains both a mechanism that processes the datasets individually, and a mechanism that aggregates the data originating from the various datasets. After the analysis, the analysis results of each individual dataset as well as the aggregated analysed data is accessible for inspection, or further processing.

4.2.2 Distributed Computing

Distributed DDoSGrid facilitates a distributed architecture for the analysis of PCAP files. That being said, the term *distributed architecture* describes an architecture that allocates software components on multiple instances that are able to do the computing required by the software. This differs from a centralized approach in the sense that the various software components are not run on a single, standalone instance. In a distributed architecture, a communication channel and the corresponding interfaces need to be established in order to enable communication between the distributed instances.

In the current implementation of SecGrid, the analysis component is a centralized instance that handles the entire process of parsing packets, extracting features and post-processing the extracted metadata. The architecture does not allow the launch of multiple analysis

instances, and thus only one dataset can be analysed in every iteration. Hence, with this centralized approach, considering multiple data sources for a single analysis iteration is not possible.

With the distributed DDoSGrid architecture, the analysis component architecture consists of a central management node and multiple distributed worker nodes. The central node is responsible for orchestrating the worker nodes, while the worker nodes perform the PCAP data analysis and feature extraction. In addition, the central node collects metadata (*i.e.*, the extracted features) produced by the worker nodes, aggregates said data, and administers the post-processing of both the non-aggregated and aggregated metadata. Lastly, a communication channel that enables the exchange of data between the central node and the worker nodes is established.

4.2.3 Scalable Feature Extraction

In general, application scalability describes "the ability to handle increased workload" [40]. In the context of SecGrid, this work considers increased workload to be in the form of input datasets increasing in size. Therefore, the more efficient the application is in handling large datasets, the more scalable it is. In addition, since this work aims to develop a collaborative approach, the ability to analyse multiple datasets at the same time is another characterization of scalability.

With respect to the current state of the analysis component, SecGrid is able to parse and analyse large PCAP files, thus providing scalability partly. However, the current architecture offers neither a mechanism to perform multiple analyses simultaneously, nor does it provide the possibility to aggregate analysed data automatically.

Distributed DDoSGrid achieves scalable feature extraction by providing an analysis system that is able to handle large datasets, containing several hundred thousand packets. The approach satisfies the collaborative aspect of scalability by being able to analyse and aggregate multiple large datasets in parallel and in the scope of a single analysis. In addition, the feature extraction mechanism is highly scalable since it enables the distribution of workload onto multiple nodes. Instead of transferring entire datasets via the networking interface, only the extracted features formatted in JSON files are sent to the central node.

4.2.4 Other Features

In order to conserve high usability, the D-DDoSGrid analysis system incorporates a mechanism that aggregates data from multiple sources without manual effort. The analysis component as well as the metadata exchange run fully automated, regardless of the amount of datasets that are provided.

The current SecGrid system consists of open-source software that is accessible on GitHub. It is also free of charge, and thus anyone can have access to it. To conserve free and unrestricted access, the software developed in this work is published on GitHub, along

with the source code comprising the current SecGrid system. This is not the case for some tools discussed in Chapter 3, where the source code is not accessible

The visualization module is an important component of the user interface and provides valuable insights about the analysed data. Therefore, the D-DDoSGrid architecture is designed such that it can be integrated into the visualization component at a later time.

Chapter 5

Implementation

This chapter provides insight into the prototype development. The various technologies that are considered for the extended software components are discussed, and the steps that are taken in order to develop the prototype according to the requirements and architecture elicited in Chapter 4 are reviewed. Lastly, the limitations and challenges that are discovered during the implementation process are discussed, as well as the countermeasures that are undertaken to manage those challenges.

5.1 Implementation of Key Components

In order to implement the key features of the D-DDoSGrid approach proposed in Section 4.2, multiple intermediate implementation steps are extracted. First, the implementation of the analysis component needs to be extended such that it can satisfy the distributed aspect of the DDoS-Grid approach. Second, a suitable communication interface needs to be established to enable collaborators to communicate and exchange data with the central node. Lastly, the data aggregation mechanism needs to be designed such that not only the individual feature extraction mechanisms of the miners, but also the need to maintain a consistent data format for all actors involved are taken into account.

As for the technology used for the implementation of the D-DDoSGrid prototype, all of SecGrid's components are implemented in JavaScript. The analysis component runs with `Node.js`, an asynchronous, event-driven JavaScript runtime [41]. Therefore, the extensions of the analysis component implemented in this work are also developed using `Node.js`. For the communication interface, the `Socket.IO` library is used.

5.1.1 D-DDoSGrid Analysis Component

As defined in the D-DDoSGrid architecture (*cf.*, Figure 4.2), the analysis component consisting of the packet decoder, protocol parser, and the miners are instantiated on every worker node in order to enable distributed computing. To facilitate the handling of interim

results, a public `getInterimResult` getter function specific to each miner is implemented. After the miners have completed the feature extraction, the results are stored as properties on the miner instances. For the client to access these results and transmit them to the central node, the `getInterimResult` function is called on each miner. Depending on the miner, the interim results are formatted before they are returned to the client, in order to maintain a consistent data format across all miners.

In Listing 5.1, the getter method specific to the `HTTP Endpoints` miner with additional data formatting is shown. Initially, the results have the format `{endpoint: "/", count: 40}`. They are then formatted to `{"/": 40}`, and returned to the client, which will transmit the interim results to the central node, using the communication interface. This data format facilitates data aggregation, since from a programmatic perspective, it is easier to parse and compare.

```
1  class HTTPEndpoints extends AbstractPcapAnalyser {
2
3    getInterimResults () {
4      var interimResult = {}
5      for (const result of this.results) {
6        interimResult[result['endpoint']] = result['count']
7      }
8      return InterimResult
9    }
10 }
```

Listing 5.1: Getter Method for Interim Results on HTTP-Endpoints Miner

The process of pre-formatting interim results before returning them to the client ensures that a consistent data format is maintained across all analysis component instances. Therefore, an arbitrary number of distributed worker nodes can be instantiated in the scope of one analysis, and the miners will ensure that the same data format is maintained.

5.1.2 Communication Interface

For the baseline architecture of the communication interface, it is established that a client-server architecture with bi-directional communication is most suitable. The server runs on the central node, listening on a specific port. Each worker node then runs as a client, connecting to the server. The bi-directional communication allows for the client to send metadata to the server, and for the server to orchestrate a set of clients. This provides the necessary communication interface so that the results of the distributed miners can be aggregated. To enable this architecture, two `Node.js` modules are considered, namely `net` and `socket.IO`.

The `net` module provides an asynchronous network API for creating stream-based TCP servers and clients [42]. It provides a `Server` class, which is used to create a TCP server, and a `Socket` class, which is an abstraction of a TCP socket that can be created to communicate with a TCP server. Both classes extend the `Node.JS EventEmitter` class, therefore custom events can be defined for a server or socket to listen on, apart from

the default events (*e.g.*, `data, error, connect, close`). With respect to the prototype implemented in this work, there are a few drawbacks to using TCP sockets. First, the `net` module is a low-level interface that allows for raw TCP connections. This is sufficient for a basic implementation of a communication interface between the central node and worker nodes as part of the analysis component. However, to enable the possibility of integration with SecGrid's user interface and visualization module at a later time, a higher-level networking interface with browser support is preferable. Second, the `net` module's TCP sockets and servers exchange data in the form of either a `String` or `Buffer` object. As defined in the proposed architecture (*cf.*, Figure 4.2), the data should be transmitted in JSON format, which is a language-independent data exchange format. Therefore, using the `net` module, the client would first need to convert the data to be transmitted to a `String` object, and upon receiving data, the server would need to parse the `String` object back into JSON format.

`Socket.IO` is a library that provides bi-directional communication based on the WebSockets protocol [43]. It consists of a server framework implemented in `Node.JS`, and a client library for the browser, which can also be run from inside `Node.JS`. Similar to the `net` module's TCP sockets, the `Socket.IO` API leverages the `EventEmitter` class to emit events on one side and register listeners on the other side. Other than that, the `Socket.IO` library provides several valuable features that TCP sockets do not support. For instance, when a client connects to the server, `Socket.IO` will try to establish a WebSocket connection, and fall back to HTTP long-polling if that's not possible [43]. The client library has reconnection support, thus enabling automatic reconnection of a client to the server. In addition, the `Socket.IO` API supports any serializable datastructure as argument when transmitting data, thus allowing to send data in JSON format without the need to serialize JSON objects beforehand. Given the advantages the `Socket.IO` library presents compared to the `net` module's TCP sockets, it is established that the `Socket.IO` library is used for the implementation of the communication interface.

Listing 5.2 demonstrates the usage of a `Socket.IO` server on the central node. This snippet shows the principle of the `EventEmitter` class that the `Socket.IO` library makes use of. An event listener is registered with the `on` keyword, and its callback is called when the client emits said event with the `emit` keyword. In the code snippet, the `io` variable represents a `Socket.IO` server instance. The `connection` event listener is registered on the server instance, thus the code snippet ranging from Line 5 to 16 is executed every time a client connects. The `connection` event is one of the built-in events that `Socket.IO` automatically emits whenever a client instance connects to a server instance. Upon connection, multiple custom event listeners are registered on the socket to listen on events emitted from the client. For instance, on Line 5, the event called `interimResult` waits for metadata sent by the client, and the post-parsing analysis is run upon receiving the metadata. On Line 9, after handling the received interim results, the connection to the client is closed, since it is not needed anymore. After all the listeners specific to the client have been registered, the socket emits an event called `ack` to the client, acknowledging the connection the client has attempted. Lastly, on Line 19, the server is instructed to listen on port 3000 for incoming connections.

In Listing 5.3, the connection of a worker node to the central node as well as the data transfer between the nodes is highlighted. First, the client connects to the server, which is

implemented in the `createSocketClient` function on Lines 6 to 15. In this code snippet, the `io` variable represents a `Socket.IO` client instance. If the connection attempt succeeds, a socket (in this code snippet called `client`) is initialized. The socket then registers a listener on the `ack` event, which the server emits as soon as the connection is established (*cf.*, Listing 5.2, Line 16). After that, the worker node sets up the miners and runs the analysis, which will produce interim results in the form of extracted features. On Line 4, the interim results are sent to the central node using the `Socket.IO` communication interface. For that, the `EventEmitter` and a custom event called `interimResult` are used, for which an event listener has been registered on the central node.

```
1  async function initializeServer () {
2
3    io.on('connection', (socket) => {
4
5      socket.on('interimResult', async (interimResult) => {
6        // handle interim results
7        runPostParsingAnalysis(interimResult, currentPcapFilePath)
8
9        socket.disconnect()
10     })
11
12     socket.on('disconnect', (reason) => {
13       console.log(reason)
14     })
15
16     socket.emit('ack') // Signal to the worker
17   })
18
19   server.listen(3000, () => {
20     console.log(serverInfo)
21   })
22 }
```

Listing 5.2: Orchestrating Communication on the Central Node

```
1  var client = await createSocketClient()
2  await setUpMiners(miners)
3  var interimResult = await runMiners(miners)
4  client.emit('interimResult', interimResult)
5
6  async function createSocketClient () {
7    return new Promise(function (resolve) {
8
9      var client = io.connect()
10
11     client.on('ack', () => {
12       resolve(client)
13     })
14   })
15 }
```

Listing 5.3: Worker Node Connecting to the Interface Provided by the Central Node

The implementation of the communication interface enables the distributed analysis of datasets. Multiple instances can launch the analysis component as a worker node and

connect to the central node. The interface also provides a first step towards collaboration. Multiple actors can collaborate by spawning a worker node for each actor, and by connecting to a mutual central node. In addition, by leveraging the feature of running an analysis on multiple worker nodes simultaneously, the workload of analysing a large dataset can be split up onto multiple instances instead of running the analysis on a single node, which increases the scalability of the process. However, to enable collaboration and the distribution of workload, a data aggregation mechanism needs to be in place.

5.1.3 Metadata Aggregation

In order to create a clear outline of the aggregation mechanism, the assumption is made that in any case, the data to be aggregated is distinct. In other words, it is assumed that no two datasets that are to be analysed in the same analysis instance contain identical network traffic. This assumption is made to ensure that no distorted perspectives are generated when contemplating the aggregated features.

```
1  class HTTPEndpoints extends AbstractPcapAnalyser {
2    static aggregateResults (resultA, resultB) {
3      for (var key in resultA) {
4        if (resultB.hasOwnProperty(key)) {
5          resultB[key] += resultA[key]
6        }
7        else {
8          resultB[key] = resultA[key]
9        }
10     }
11     return resultB
12   }
13 }
```

Listing 5.4: Sample of a Data Aggregation Function

The data aggregation mechanism is responsible for aggregating the metadata that the central node receives from worker nodes through the communication interface. In essence, when receiving metadata via the communication interface, the central node checks whether it has already received metadata from other worker nodes, and starts the aggregation mechanism if that's the case. Since the feature extraction mechanism is specific to each miner, the mechanism that aggregates the extracted features also needs to be tailored to the miners' data structures. For instance, aggregating statistics about the verbs (*e.g.*, `GET, PUT, POST`) used in HTTP requests requires a different aggregation mechanism than aggregating statistics about the ratio between UDP and TCP packets. This poses a challenge with respect to the implementation of the aggregation methods, since the miners are only instantiated on the worker nodes, and not on the central node, where the aggregation mechanism is run. It is therefore established that the aggregation methods are implemented as static methods in the miner classes. The results to be aggregated are passed to the method as arguments. This way, the server instance on the central node can access the static aggregation method on each miner, without having to instantiate the miners. Listing 5.4 shows an exemplary implementation of an aggregation mechanism within the `HTTP Endpoints` miner. In this case, the method iterates over the entries of

the first result and checks whether the same key is present in the second result. If so, the values are summed up, and otherwise a new entry is created.

Within the scope of the prototype implementation, it is shown that this data aggregation approach is feasible for a variety of miners, covering the link, network, transport, and application layers with respect to the OSI model. If the method is invoked on a miner that does not implement the aggregation method, the method will throw a custom `NotImplemented` Error, which the program flow of the server will catch in order to ensure the continuous operation of the server instance.

The implementation of the data aggregation mechanism is a crucial contribution to enable the analysis of attack data in a collaborative setting. It enables the aggregation of metadata originating from different datasets, which provides valuable insights for all collaborators involved. At the same time, the aggregation mechanism ensures that a consistent data format is maintained, such that the formatting of aggregated data is compatible with non-aggregated features, and both can be further processed accordingly. In addition, the mechanism handles the data aggregation in an automated manner, involving no manual effort.

5.2 The D-DDoSGrid Prototype

After a first implementation of the key components of D-DDoSGrid is completed, the D-DDoSGrid prototype is put together. The interaction of the different features during the analysis workflow is visualized in an activity diagram shown in Figure 5.1. In the swimlane on the left, the diagram shows a worker node connecting to a central node. The central node is represented in the swimlane on the right.

The DDoSGrid analysis workflow starts with the central node creating a socket server, and waiting for a client to connect. When a worker node is initialized, it first prepares the analysis components, *i.e.*, it initializes the packet decoder, the protocol parser, as well as a set of miners. After the set up is complete, the worker node creates a socket client, which attempts to connect to the socket server on the central node. After the connection is established, the central node emits the `startAnalysis` event via the communication interface in order to indicate to the worker node to start the analysis. Upon receiving said event, the worker node performs the analysis by running the analysis component. The result of the analysis is a set of interim results in the form of JSON data. Using the `interimResults` event, this data is then transferred to the central node through the communication interface. Upon receiving the interim results, the central node starts the post-processing mechanism. On the one hand, it starts the post-parsing analysis on the interim results, which will result in the extracted features. These extracted features maintain a consistent data format compatible with the original SecGrid data formats. On the other hand, the central node checks whether it has already received interim results from another worker node. If so, it triggers the data aggregation mechanism, and in addition runs the post-parsing analysis on the aggregated data, in order to produce the aggregated features. These aggregated features can be handled the same way as the non-aggregated features, since they maintain the same data format.

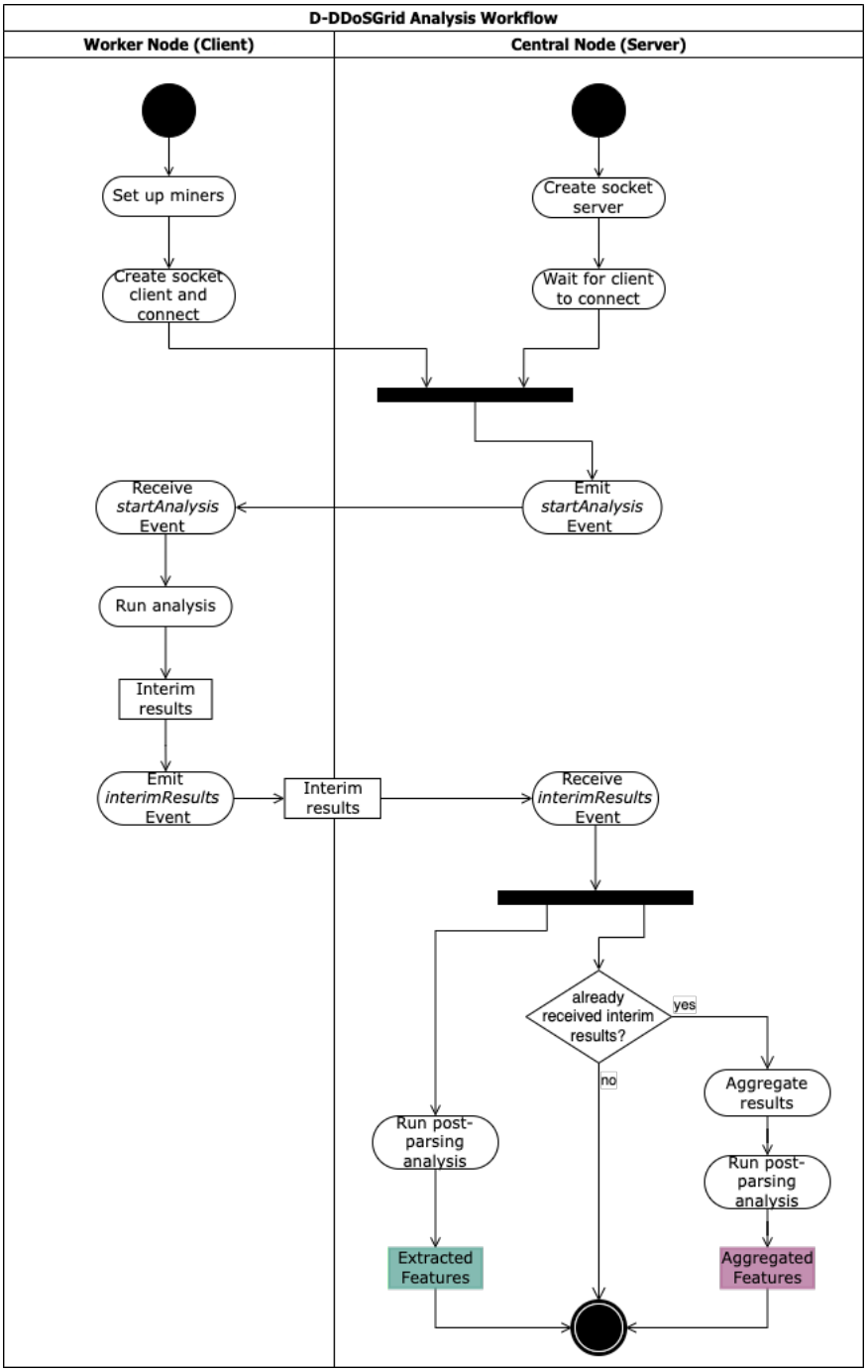

Figure 5.1: Initial Implementation of D-DDoSGrid Analysis Workflow

Thanks to its distributed nature, the analysis workflow is the same for each worker node. Therefore, multiple clients can connect to the server, either simultaneously or consecutively. The connection interface on the side of the server is kept open unless the execution of the `Node.js` script is stopped. Thus, an analysis can be performed in a collaborative setting even if the points in time where the distributed worker nodes are instantiated are stretched over a longer period of time.

5.3 Implementation Challenges

In the course of the prototype implementation, some challenges are revealed with respect to the `Socket.IO` communication interface. When analysing large PCAP files (*i.e.*, with a magnitude of several hundred thousand packets), the packet decoder as part of the analysis component causes high resource consumption in the form of high CPU load. In addition, the analysis of large PCAP files results in long runtimes of the packet decoder. While testing the first D-DDoSGrid prototype, it is discovered that the client-side socket connection on the worker node can be rendered faulty after a large dataset has been analysed.

A first potential cause is found in the built-in `ping-pong` mechanism of `Socket.IO`, which works as follows: The server sends a `PING` packet at a given interval, and the client sends a `PONG` packet back. If the server does not receive a `PONG` packet back after a given amount of time has passed, it will consider the connection closed [44].
In D-DDoSGrid, the packet decoder component is built on top of the `Node.JS EventEmitter` class. Listing 5.5 gives a simplified overview over how the correct running order is ensured. In the analysis workflow, the packet decoder needs to finish (*i.e.*, emit the `complete` event) before the analysis workflow can continue with the protocol parser and running the miners. To ensure this, the `await` keyword is used. This will ensure that the protocol parser and miners are only run after the decoding has finished. However, this keyword will pause the execution of any code dependent on the result of this await statement, not allowing that code to be executed until the awaited function has completed.

```
1  await runPcapDecoder(emitter, target)
2
3  // Only run Protocol Parser and Miners after decoding has finished
4  runProtocolParser()
5  runMiners()
6
7  async function runPcapDecoder (emitter, target) {
8
9    emitter.startPcapSession(target)
10
11   emitter.on('complete', () => {
12     console.log(`Decoding has finished.`)
13   }
14 }
```

Listing 5.5: Packet Decoder built on `EventEmitter`

It is therefore initially assumed that the socket client is not able to respond to the `PING` packet it receives from the server while awaiting the packet decoder to finish. This can cause issues when large datasets are analysed and the packet decoder requires a long enough time to finish. In fact, it is assumed that if the runtime of the packet decoder is longer than the amount of time the socket server waits for the `PONG` packet of the socket client, the connection is closed. This is derived from the assumption that the client-side `Node.JS` runtime is paused during the packet decoding and is therefore not able to respond to the server-side `PING` packet.
To solve this problem, the default values of the `pingInterval` and `pingTimeout` values of the socket server are increased. The first value determines how often the `PING` packet is emitted, and the latter value defines how long the server will wait for the `PONG` packet to be emitted by the client. By increasing these values, it is expected that the server will not close the connection prematurely.

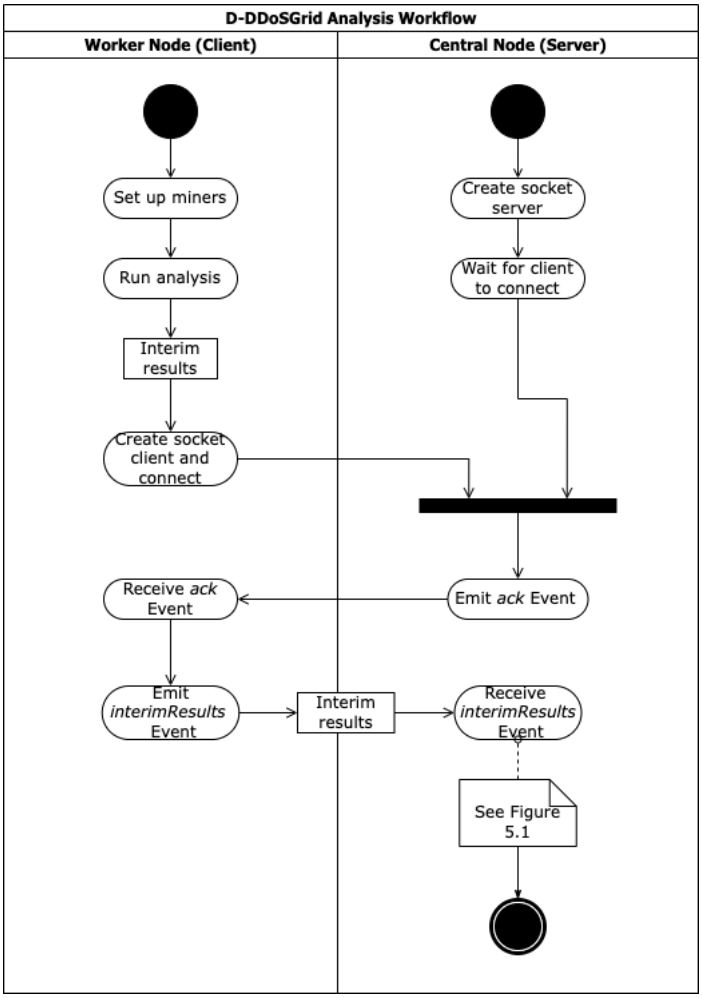

Figure 5.2: Adapted Implementation of D-DDoSGrid Analysis Workflow

However, the issue persists after these countermeasures are taken. After further careful inspection, it is established that the issue arises only in cases where the packet decoding process takes up `99%` or more of the available CPU capacities. Therefore, it is assumed that what renders the socket connection faulty is the high CPU load the packet decoder causes, rather than the potentially long runtime. It is important to note that the size of

a dataset is not necessarily the sole indicator of how much computing power the packet decoder requires, since this could also be dependent on the type and composition of the packets to be analysed, or other metrics.

The issue is bypassed by adapting the analysis workflow of the worker node. The updated D-DDoSGrid analysis workflow is depicted in Figure 5.2. On the adapted worker node, the decoding of the packets occurs before the socket client establishes a connection to the server on the central node. While this mitigates the underlying problem, this is not an ideal solution since it limits the orchestration possibilities of the central node over the worker node. As part of future work, the architecture of the worker node could be split up such that the analysis component and the socket client run in independent processes. That way, the high consumption of computing power of the packet decoder might not affect the connection stability of the socket client.

Chapter 6

Evaluation

In this chapter, the implementation of D-DDoSGrid and its added value are evaluated using case studies. The fulfillment of the system requirements defined in chapter 4 is investigated on the basis of these case studies. In addition, the scalability and performance of the D-DDoSGrid architecture is evaluated by observing the process of analysing a large PCAP file.

6.1 Case Studies

For this section, three case studies have been designed in order to cover multiple possible use cases that validate the D-DDoSGrid architecture and its main features. The first case study involves the usage of the D-DDoSGrid prototype as a post-mortem analysis tool for multi-vector DDoS attacks. In the second case study, it serves as a tool for comparing attack data of multiple independent collaborators. Lastly, D-DDoSGrid enables the approach of federated learning in order to classify DDoS attacks in the third case study.

6.1.1 Analyzing a Multi-Vector DDoS Attack

In this case study, the employment of the D-DDoSGrid prototype is considered in the context of analysing DDoS attacks in a post-mortem fashion. The goal of this scenario is to illustrate how attack data from multiple data sources are aggregated in order to gain additional perspectives. The specific scenario looks as follows: It is assumed that multiple components of a service hosted on the web are affected by a multi-vector DDoS attack. An application layer attack on the web server and a transport layer attack are observed. The attack vector executed against the web server is a HTTP POST flood attack, and the transport layer attack vector is a SYN flood attack. The operators of the service are in possession of packet captures from both servers, which were recorded during the attack. To illustrate this scenario, sample captures of both a HTTP POST flooding attack and a SYN flood attack are used. Both are packet captures of real-world DDoS attacks, retrieved from an open-source repository on GitHub [45]. Some of the attack

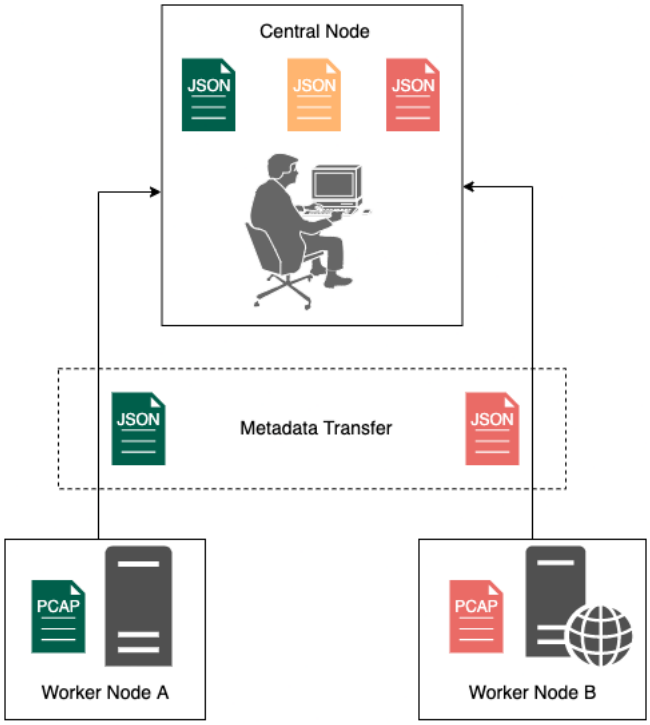

Figure 6.1: Data Sharing and Data Aggregation Workflow

data, such as source or destination IP addresses, has been anonymized, and therefore, the data displayed should be considered for illustration purposes only.

In this scenario, the service operators wish to analyse the secured attack data and derive knowledge from it. First, examining the captures separately provides information about characteristics specific to each attack. For instance, the different API endpoints targeted as well as the distribution of HTTP verbs used during the attack are of importance when classifying the application layer attack. Similarly, the ratio of UDP to TCP packets as well as the frequency of the various TCP flags (SYN, ACK, FIN) are relevant when analysing the transport layer attack. Secondly, the prototype developed in this work can be utilized to aggregate the attack data and gain additional insights. For instance, an aggregated view of the most common source hosts IP addresses provides insights on the distribution of the attack sources, and on whether the attacks are correlated. Alternatively, an aggregated view on the combinations of browsers and operating systems (OS) gives an insight on whether the attacks were launched using a botnet.

Using the prototype of this work, the two servers on which the attacks occurred act as worker nodes that send critical metadata to a central node, which in this scenario is the personal device of one of the service operators. The worker nodes analyse the attack data using the miners implemented by SecGrid, which results in metadata formatted in JSON data. This data is then sent to the central node for an initial analysis and further processing. In contrast, other approaches require to send the full attack data in the form of packet captures to the central entity, where that data is aggregated manually, which results in lower scalability and higher manual effort. In Figure 6.1, an overview of this constellation and the transfer of metadata is given. The component labeled `Worker Node`

A represents the server targeted by a transport layer attack vector, along with the packet capture containing attack data. The component labeled Worker Node B represents the web server containing attack data from the application layer attack vector, respectively. The Central Node component represents the service operator receiving the metadata, in JSON format, originating from the worker nodes, as well as the aggregated metadata. With respect to attack data related to the application layer attack, the relevant metadata is shown in Listing 6.1. This data shows that the majority of the HTTP verbs used in requests were of the type POST, and all requests were made on the same endpoint. Therefore, the operators determine that the attack was probably of the type HTTP POST flooding.

```
{
  GET: 94,
  POST: 125,
  HEAD: 0,
  PUT: 0,
  DELETE: 0,
  PATCH: 0,
},
{
  "/": 219,
}
```

Listing 6.1: Traffic Metrics Indicating HTTP Verbs and Endpoints

Similarly, the operators derive knowledge from the metadata analysed on the server with respect to transport layer traffic, shown in Listing 6.2. The data shows that only TCP packets and no UDP packets were sent. Furthermore, all TCP packets hold the SYN flag. Therefore, it appears likely that the attack can be categorized as a SYN flooding attack vector.

```
{
  nrOfUDP: 0,
  nrOfTCP: 37841,
},
{
  nrOfPacketsInSynState: 37841,
  nrOfPacketsInSynAckState: 0,
  nrOfPacketsInFinState: 0,
  nrOfPacketsInFinAckState: 0,
  nrOfPacketsInAckState: 0,
  nrOfPacketsInRemainingStates: 0,
  nrOfTransportPackets: 37841,
}
```

Listing 6.2: Traffic Metrics Indicating UDP vs TCP Ratio and TCP States

The post-processing procedure of metadata includes formatting the metadata such that it can be fed into the visualization mechanism of SecGrid, as well as aggregating metadata from multiple independent worker nodes. The aggregated data provides an overview of the attacks and potentially opens a new perspective with respect to the correlation of the attacks. The service operators look at the aggregated source host IP addresses to get an overview of the various attack sources, the amount of packets originating from these

sources, and possibly their geographic location. In Listing 6.3, the top 15 source hosts, along with the number of packets sent from these sources are shown. The expression `top N source hosts` is defined as the N source hosts that sent the largest amount of IP packets. From this data, the service operators derive that some hosts were sending large numbers of packets, implicating that these requests were made automatically as opposed to manual requests made by legitimate users. Additionally, the non-aggregated data is compared to check if some of the source hosts were involved in both attacks.

```
{
  "8.8.8.8": 6430,
  "10.0.0.2": 1429,
  "10.128.0.2": 1002,
  "8.8.4.4": 847,
  "212.8.51.69": 750,
  "212.8.51.71": 555,
  "212.8.51.72": 414,
  "212.8.50.179": 414,
  "212.8.51.99": 408,
  "146.120.160.133": 324,
  "212.8.51.140": 315,
  "212.8.50.158": 312,
  "1.1.1.1": 250,
  "212.8.51.120": 240,
  "212.8.50.188": 210,
}
```

Listing 6.3: Aggregated Traffic Metrics Indicating Top 15 Source Hosts

Finally, since both the non-aggregated as well as the aggregated metadata is stored on the central node, the service operators can further explore the provided data according to their specific needs to get as much insight as possible about the attack. In contrast, deriving the same knowledge with related approaches would entail copying large amounts of data and processing it manually.

6.1.2 Comparing Attack Data of Multiple Independent Devices

This case study investigates an approach that uses D-DDoSGrid to analyse and compare cyberattacks of the same type, originating from multiple independent hosts. Specifically, it is assumed that multiple independent devices store packet captures recorded during SSH brute force attacks. The SSH brute force attack is a common attack vector based on which the attacker attempts to gain access to a remote device by guessing username and password combinations on the SSH interface [46]. SSH brute forcing is a prevalent attack vector since systems can easily be compromised if lax security measures are in place [47], especially due to the increase of IoT devices, where default username and password combination (*e.g.*, `admin/admin` or `root/root`) are not changed.

In this scenario, multiple independent device administrators wish to compare the attack data they collected on the affected devices. This administers not only the process of deriving knowledge about the attack their devices are affected by, but also gaining insights on other SSH brute force attacks. As a consequence, additional perspectives on the attack

vector are gained. This in turn enables deriving suitable defense mechanisms against SSH brute force attacks, such as traffic filtering or limiting connection rates [47]. Concretely, the administrators of these devices wish to investigate the source hosts involved in SSH brute forcing, not only on their own devices, but across all devices that are part of this scenario. This enables multiple perspectives on the geographic distribution of the source hosts.

Two real-world datasets, in the form of packet captures, have been generated to use for the visualizations of this case study. Both datasets were recorded using Virtual Machines (VM) running Ubuntu, located in Helsinki. Both datasets were recorded over a time span of approximately 18 hours. To capture the traffic, the `tcpdump` [48] software was used with the following command:

```
1 $ tcpdump -i enp3s0 port 22  and src net not 65.108.0.0/16
```

With this command, only port 22 and inbound packets were considered (*i.e.*, traffic originating from the management network is excluded), everything else was dropped. Since no other SSH traffic was moving to or from this host, it can be assumed that all packets were malicious and targeting the SSH brute force vector specifically. The command output was written into PCAP files. For the sake of simplicity, this case study assumes that there are two worker nodes connecting to the central node. However, in practice, this scenario works for any number of distributed worker nodes.

Using the D-DDoSGrid prototype, the devices in possession of the packet captures act as distributed worker nodes, and use the miners to analyse the PCAP files. The worker nodes then connect to a central node and transmit the metadata resulting from the analysis to the central node. On the central node, the metadata is further processed, and the results originating from the separate worker nodes are aggregated. In this scenario, the non-aggregated data is more relevant than the aggregated data, however the aggregated data is still considered. To enhance the ability to compare the extracted features, the data stored on the central node is fed into the visualization system of SecGrid. For instance, using a pie chart simplifies the analysis of the most common source hosts, *i.e.*, the source hosts that sent the largest number of packets, and a world map highlights their geographic location. While the same conclusions about the data would be made if the data is analysed in `JSON` format, the visualizations support human knowledge derivation.

The attack metrics and visualizations resulting from the analysis of the two datasets are depicted in Figure 6.2. Each row represents attack data from one of the attacks (*i.e.*, datasets), labelled `Node A` and `Node B`, respectively. From the metrics tab on the left it becomes apparent that a large amount of packets were sent to both hosts, originating from a variety of source IP addresses. The following two visualization tabs represent the `top N source hosts`, meaning the N source hosts that sent the largest amount of IP packets. The world map visualizations of the top 100 source hosts highlight the fact that the geographic locations of the source hosts are similar in both attacks, with the majority of packets originating from India, Russia, and China. Additionally, the pie chart visualizations reveal that the most common source hosts are different. With respect to the first attack (`Node A`), the majority of packets originate from a host in Russia, whereas the attack sources of the second attack (`Node B`) are more distributed, with three hosts in China and Hong Kong combined, and one each in India and Russia.

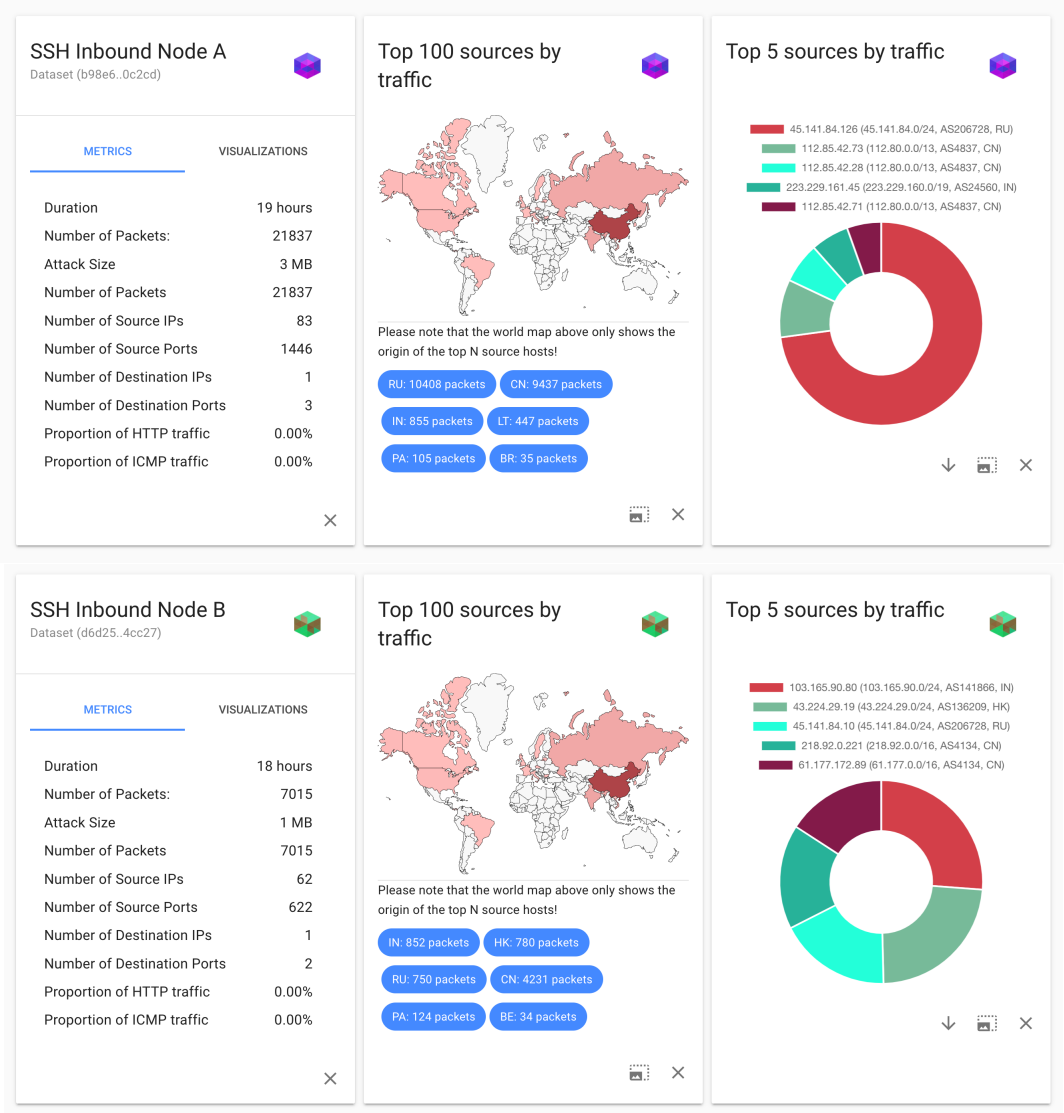

Figure 6.2: Attack Metrics and Source Hosts Visualizations

In Figure 6.3, the aggregated data of the two datasets is shown. When looking at the world map visualization of the aggregated top 100 source hosts, the large amount of malicious packets originating in Russia and China is highlighted, and the perspective on the geographical distribution of the overall traffic is maintained. However, some insights get lost. For instance, one of the top 5 source hosts of node A and one of node B originate from the same autonomous system (AS) (*i.e.*, they hold the same AS number), implying that these source hosts are interrelated. When looking at the aggregated top 5 source hosts visualization, this insight is lost. In order to maintain this perspective, a larger subset of the aggregated data needs to be considered (*i.e.*, consider the top 10 or top 20 source hosts instead). In conclusion, whether or not considering the aggregated data visualizations in addition to the separate visualizations offers additional insights on the data is dependent on the type of visualization and on the size of the subset of aggregated data that is considered.

Due to the direct comparison of attack data this approach enables, administrators gain

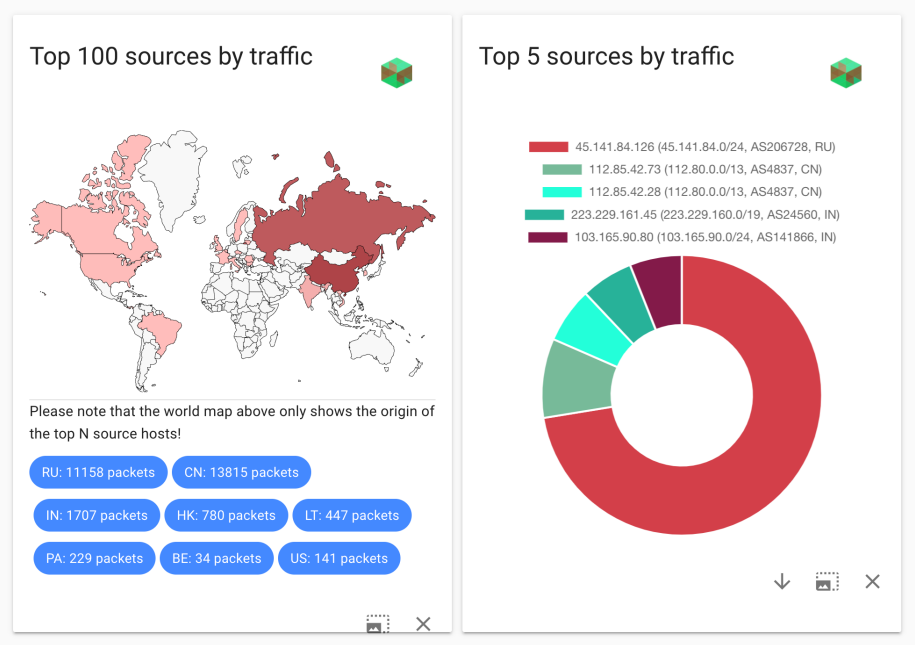

Figure 6.3: Aggregated Source Hosts Visualizations

enhanced insights about the characteristics of SSH brute force attacks. For instance, multiple perspectives on the source locations of malicious traffic facilitate the process of traffic filtering. Additionally, comparing the number of malicious packets sent to the device brings to light approaches on how to limit connection rates. Both measures constitute effective defense mechanisms against SSH brute force attacks. In order to achieve a direct comparison with other approaches, administrators first need to manually exchange the datasets. The various datasets then need to be analysed separately, and a mechanism needs to be in place that can contrast the analysis results with each other. Since most approaches don't offer functionality that allows for the direct comparison of multiple datasets, this would also need to be done manually in most cases. In addition, to get insights on the aggregated data as well, the data would need to be aggregated manually.

6.1.3 Federated Learning

Federated learning is a machine learning technique that trains a model across multiple decentralized nodes with local data samples, without exchanging them between the nodes [49]. This concept provides a distributed alternative to centralized machine learning approaches where multiple datasets are uploaded to one central instance [49]. Additionally, potentially sensitive data is contained on the original nodes, and not distributed.

In this use case, the D-DDoSGrid prototype is used to implement the *centralized* federated learning approach. This means that a central server is responsible for coordinating the distributed nodes and aggregating the updated models, in contrast to the decentralized approach, where no central node is needed and the nodes are able to coordinate themselves [49]. Specifically, instead of gathering a lot of network data on the central server, the central server distributes a model to the nodes. Then the nodes use that model to

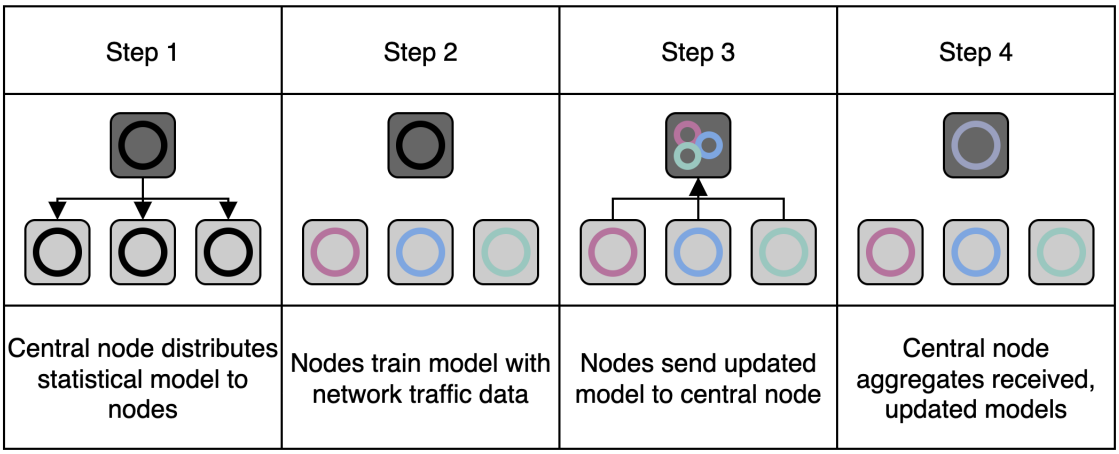

Figure 6.4: Centralized Federated Learning Workflow Using Network Traffic Data [49]

learn from their observed network traffic. Finally, just the updated models need to be sent to the central server where they can be aggregated. The workflow of this technique is illustrated in Figure 6.4.

In this scenario, the model to be trained is a model that detects whether a DDoS attack has taken place, *i.e.*, it classifies whether the input data bears attack signs. The central node is in possession of a statistical model, and the decentralized nodes are in possession of packet captures, recorded during DDoS attacks. These packet captures are now used to train the statistical model. The prototype developed in this work is leveraged for several tasks of this scenario. The distributed architecture and the communication functionality implemented in this work are used for the orchestration of the central node and the distributed nodes, as well as the communication between the nodes. This corresponds to step 1 and step 3 in Figure 6.4. Furthermore, the data mining process which generates input data for the machine learning model is done by using the miners implemented in SecGrid. This corresponds partly to step 4 in Figure 6.4. In order to generate input vectors for the model, the machine learning feature extraction miner implemented in SecGrid is used. The miner extracts features from the packet captures on the nodes and stores them in `CSV` format, in contrast to the conventional miners of SecGrid that store extracted data in `JSON` format. Using the networking implemented in this work, the updated models are sent back to the central node, and the distributed nodes disconnect automatically. After the central node has aggregated the updated models, further iterations of the workflow can follow. For this, the workflow restarts at the first step, and the central node distributes the updated model to different nodes, containing different network traffic data. With the implemented prototype, this is simply done by connecting a new set of distributed nodes to the central node using the networking interface.

In conclusion, the approach of using D-DDoSGrid for the centralized federated learning techniques generates added value in comparison to conventional centralized machine learning techniques in several aspects. For instance, data privacy is improved in situations where sensitive data, *e.g.*, network traffic data that contains IP addresses, is used to train statistical models. Only the updated model is transmitted to the central node instead of the dataset containing the sensitive data, therefore maintaining privacy. Another advan-

tage is the limited demand of bandwidth with respect to the network connectivity between the central node and the distributed nodes. Transmitting only the updated model and not the entire dataset to the central node implies transmitting smaller amounts of data at a time.

6.2 Scalability and Performance Evaluation

In this assessment, the scalability of analysing a large PCAP file using the D-DDoSGrid prototype is evaluated. To do so, it is assessed whether the distributed aspect of the D-DDoSGrid architecture can increase performance in terms of time needed to perform the analysis, compared to the centralized SecGrid architecture.

For this evaluation, a large PCAP file is first analysed using the current SecGrid analysis component. Then, the same file is analysed using the distributed D-DDoSGrid analysis component. The PCAP file used is a packet capture of a SYN-flood attack, containing `1'390'995` packets, with a file size of `97.4 MB`. The evaluation is performed on a machine with `6 CPU` cores and `16GB RAM`. To measure the runtime of the analysis, the point in time at which the analysis is started is measured. After the packet decoder, the protocol parser and the miners have finished, another timestamp is measured. The two timestamps are then subtracted to obtain the analysis duration in seconds, and the results are written to the console.

```
1 Analysis started
2 Setup of the following miners has completed:
3     - Miscellaneous Metrics
4     - Connection states of TCP segments
5     - Ratio between UDP and TCP segments
6 Decoding has started...
7 Decoding has finished (101.596s), starting post-parsing analysis
```

Listing 6.4: Performance with SecGrid

The output of the analysis iteration with SecGrid is depicted in Listing 6.4. For this analysis, three miners that are relevant to the SYN-flooding based attack are initialized. The output on Line 7 indicates that the analysis runtime amounts to `101.596 seconds`.

Now, the dataset is analysed using the D-DDoSGrid prototype. The dataset is split into two separate datasets of equal size, in order to distribute the workload onto two worker nodes. To do so, the following command is used:

```
1 $ editcap -c 695498 SYN-flooding.pcap SYN-flooding-split.pcap
```

With this command, the packet output of the file `SYN-flooding.pcap` is split to different files with a maximum of `695498` packets each. This command will result in two PCAP files, one containing `695498` packets, and the other containing `695497` packets. Both files have a size of `61.2 MB`. The worker nodes run the analysis simultaneously and connect to the same central node, where the metadata produced by the worker nodes is aggregated. In addition, with the `time` command, the runtime of the operation is measured:

```
1 $ time editcap -c 695498 SYN-flooding.pcap SYN-flooding-split.pcap
2 0.64s user 0.36s system 86% cpu 1.149 total
```

The output of this command on Line 2 indicates that the runtime of splitting the given dataset into two equally large datasets equals `1.149 seconds`.

```
1 Setup of the following miners
     has completed:
2      - Miscellaneous Metrics
3      - Connection states of TCP
         segments
4      - Ratio between UDP and TCP
         segments
5 Analysis has started...
6 Decoding has started...
7 Decoding has finished (51.09s),
     sending interim results to
     central node...
```

```
1 Setup of the following miners
     has completed:
2      - Miscellaneous Metrics
3      - Connection states of TCP
         segments
4      - Ratio between UDP and TCP
         segments
5 Analysis has started...
6 Decoding has started...
7 Decoding has finished (50.699s),
     sending interim results to
     central node...
```

Figure 6.5: Performance with Two D-DDoSGrid Worker Nodes

The output of the analysis iterations with D-DDoSGrid is depicted in Figure 6.5. First, it is observed that both worker nodes produce a very similar analysis runtime. Second, a linear runtime development with regard to the SecGrid analysis iteration is observed, *i.e.*, analysing half of the number of packets with D-DDoSGrid results in halt of the runtime. However, the distributed worker nodes are run simultaneously, and the aggregation mechanism on the central node ensures that the metadata of the individual worker nodes is aggregated. Therefore, the complete analysis workflow runtime is approximately halved when utilizing the D-DDoSGrid architecture using 2 worker nodes.

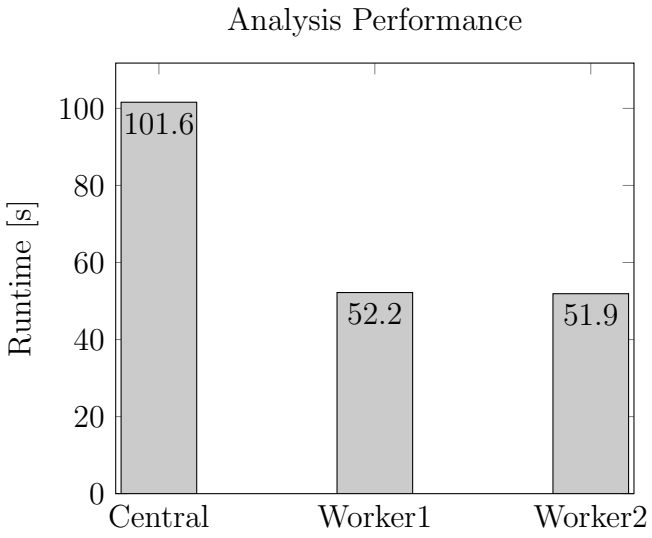

Figure 6.6: Summary of Analysis Runtimes

A summary of the runtimes given the various analysis iterations is depicted in Figure 6.6. The runtimes depicted for the two D-DDoSGrid worker nodes (*i.e.*, Worker1 and Worker2)

consist of the sum of the analysis runtime and the `editcap` runtime. In conclusion, these initial tests suggest that the distributed aspect of D-DDoSGrid may improve the performance of analysing large datasets by providing the possibility to distribute workload onto multiple worker nodes. This is especially useful in use cases where multiple datasets are analysed. Using the D-DDoSGrid architecture, a worker node can be initialized for each datasets, providing a scalable approach for collaboration. In contrast, other approaches require to manually collect and aggregate multiple datasets on a central device, which can result in large amounts of data, and then perform an analysis using the aggregated dataset. That being said, it is important to note that more extensive testing needs to be done for a more accurate picture of the D-DDoSGrid performance. For instance, the overhead of the communication between nodes, as well as the aggregation mechanism needs to be considered. In addition, experiments can be performed in order to determine whether increasing the number of worker nodes further decreases the analysis runtime.

Chapter 7

Conclusion and Future Work

The main objective of this thesis was the improvement of an existing post-mortem DDoS attack visualization and analysis tool. By enabling the distributed and collaborative analysis of cyberattacks, this thesis addresses the lack of collaborative network traffic processing tools for DDoS attack analysis and protection services present in research and industry.

The theoretical background necessary to define the key features of such a tool was established by analysing the topics of DDoS attacks and their mitigation, as well as the process of using network traffic analysis as an approach for the analysis of cyberattacks. Furthermore, the implementation and components of SecGrid, whose functionality was extended in this thesis, was thoroughly investigated. With the necessary theoretical background at hand, a review of five existing network analysis tools that can be leveraged as post-mortem DDoS attack analysis tools was performed, and these tools were then analysed based on a set of criteria such as usability, scalability, and support for collaboration. Based on this analysis, it was established that none of these tools provided the option for multiple actors to perform cyberattack analyses on the basis of network traffic in a collaborative setting. This confirmed the motivation to develop such a tool in a distributed setting.

The review of tools together with the technical knowledge about SecGrid allowed for the synthesis of a feasible architecture for the development of the prototype, called D-DDoSGrid. The key properties of this architecture include distributed computing, collaborative cyberattack analysis, and scalable feature extraction. With the D-DDoSGrid architecture proposed, a prototype was then implemented, taking into consideration the previously mentioned key properties. The prototype was evaluated by considering its features in the context of multiple case studies. These case studies involved the usage of the D-DDoSGrid prototype as a post-mortem analysis tool for multi-vector DDoS attacks, as a tool for comparing attack data of multiple independent collaborators, and as a tool enabling the approach of federated learning in order to classify DDoS attacks.

In conclusion, the evaluation suggests that the implementation of the prototype fulfills the key features defined in the D-DDoSGrid architecture. The case studies showed that the prototype can be used for the distributed and collaborative analysis of cyberattacks, while preserving the scalability of the SecGrid system, thus validating the overall approach of

this thesis. During the implementation and evaluation phase, domains were established in which the provided prototype can be further developed as future work. For instance, as of now, the prototype is only accessible via CLI. Therefore, the integration of the D-DDoSGrid architecture into the user layer of the SecGrid system would provide an increase in usability, and further encourage collaboration. A web interface could be envisioned on which multiple collaborators can interact with the analyses provided by the D-DDoSGrid prototype. Furthermore, the data aggregation component could be extended with a mechanism that is able to detect whether the attack data present in multiple datasets is related or not, and then aggregate the data accordingly. As of now, the aggregation mechanism is implemented under the assumption that the data to be aggregated is always distinct, thus excluding some use cases. Additional future work with respect to collaboration could be visualizations specific to aggregated data. For instance, visualizations that take into account the individual sources of the datasets could reveal new perspectives as opposed to visualizations that only show the aggregated data as a whole.

Bibliography

[1] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity", *Journal of Computer and System Sciences*, vol. 80, pp. 973–993, August 2014. Special Issue on Dependable and Secure Computing.

[2] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks", *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, March 2013.

[3] US-CERT, "Understanding Denial-of-Service Attacks". `https://us-cert.cisa.gov/ncas/tips/ST04-015`, November 2009. Accessed 18.10.2021.

[4] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS Attack Detection and Mitigation Using SDN: Methods, Practices, and Solutions", *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, January 2017.

[5] C. Roberts, "Biometric attack vectors and defences", *Computers & Security*, vol. 26, pp. 14–25, February 2007.

[6] D. Freet and R. Agrawal, "A Statistical Comparison of Security Visualization Efficiency Compared to Manual Analysis of IDS Log Data", in *SoutheastCon 2018*, pp. 1–5, 2018.

[7] J. von der Assen, "DDoSGrid 2.0: Integrating and Providing Visualizations for the European DDoS Clearing House", Master's thesis, Communication Systems Group, Department of Informatics,, University of Zurich, Zurich, Switzerland, February 2021.

[8] D. Wartburton, "DDoS Attack Trends for 2020". `https://www.f5.com/labs/articles/threat-intelligence/ddos-attack-trends-for-2020`, May 2021. Accessed 16.10.2021.

[9] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras, "Measuring the Adoption of DDoS Protection Services", in *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, (New York, NY, USA), p. 279–285, Association for Computing Machinery, 2016.

[10] D. Holmes, J. Blankenship, A. Bouffard, and P. Dostie, "The Forrester Wave™: DDoS Mitigation Solutions, Q1 2021". `https://www.forrester.com/report/The-Forrester-Wave-DDoS-Mitigation-Solutions-Q1-2021/RES159092`, March 2021. Accessed 15.02.2022.

[11] M. Franco, J. Von der Assen, L. Boillat, C. Killer, B. Rodrigues, E. J. Scheid, L. Granville, and B. Stiller, "SecGrid: a Visual System for the Analysis and ML-based Classification of Cyberattack Traffic", in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pp. 140–147, 2021.

[12] J. von der Assen, M. Franco, B. Rodrigues, and B. Stiller, "Analysis and Classification of Cyberattack Traffic using the SecGrid Platform", tech. rep., Communication Systems Group, Department of Informatics, University of Zurich, 2021.

[13] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet", in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1093–1110, USENIX Association, August 2017.

[14] Y. Xie and S.-Z. Yu, "Monitoring the Application-Layer DDoS Attacks for Popular Websites", *IEEE/ACM Transactions on Networking*, vol. 17, pp. 15–25, 2009.

[15] US-CERT, "DDoS Quick Guide". `https://us-cert.cisa.gov/sites/default/files/publications/DDoS%20Quick%20Guide.pdf`, October 2020. Accessed 18.10.2021.

[16] Cloudflare Inc., "Syn flood attack". `https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/`. Accessed 18.10.2021.

[17] A. Serrano Mamolar, P. Salvá-García, E. Chirivella-Perez, Z. Pervez, J. M. Alcaraz Calero, and Q. Wang, "Autonomic Protection of Multi-Tenant 5G Mobile Networks Against UDP Flooding DDoS Attacks", *Journal of Network and Computer Applications*, vol. 145, p. 102416, 2019.

[18] Cloudflare Inc., "How do layer 3 DDoS attacks work? | L3 DDoS". `https://www.cloudflare.com/learning/ddos/layer-3-ddos-attacks/`. Accessed 22.10.2021.

[19] B. Nagpal, P. Sharma, N. Chauhan, and A. Panesar, "DDoS tools: Classification, analysis and comparison", in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 342–346, 2015.

[20] H. Wang, C. Jin, and K. G. Shin, "Defense Against Spoofed IP Traffic Using Hop-Count Filtering", *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 40–53, 2007.

[21] S. Arukonda and S. Sinha, "The Innocent Perpetrators: Reflectors and Reflection Attacks", *Advanced Computer Science*, vol. 4, pp. 94–98, 2015.

[22] R. Lemos, "Multi-Vector DOS Attacks Turning into the New Normal". `https://symantec-enterprise-blogs.security.com/blogs/expert-perspectives/multi-vector-dos-attacks-turning-new-normal`. Accessed 22.10.2021.

[23] J. Malik, A. Akhunzada, I. Bibi, M. Imran, A. Musaddiq, and S. W. Kim, "Hybrid Deep Learning: An Efficient Reconnaissance and Surveillance Detection Mechanism in SDN", *IEEE Access*, vol. 8, pp. 134695–134706, 2020.

[24] Cloudflare Inc., "What is DDoS mitigation?". `https://www.cloudflare.com/learning/ddos/ddos-mitigation/`. Accessed 09.02.2022.

[25] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms", *SIGCOMM Comput. Commun. Rev.*, vol. 34, p. 39–53, apr 2004.

[26] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS attacks in cloud computing: Issues, taxonomy, and future directions", *Computer Communications*, vol. 107, pp. 30–48, 2017.

[27] Rapid7, "Network Traffic Analysis". `https://www.rapid7.com/fundamentals/network-traffic-analysis/`. Accessed 27.01.2022.

[28] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX", *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[29] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Cisco Systems Netflow Services Export Version 9", 2004.

[30] A. Wick, "Arkime | Full Packet Capture". `https://arkime.com/`. Accessed 08.10.2021.

[31] G. Combs, "About Wireshark". `https://www.wireshark.org/`. Accessed 09.10.2021.

[32] M. Pollock, "What is Wireshark?". `https://www.liquidweb.com/kb/what-is-wireshark/`. Accessed 19.11.2021.

[33] Elasticsearch, "The Elastic Stack". `https://www.elastic.co/elastic-stack/`. Accessed 16.10.2021.

[34] S. J. Son and Y. Kwon, "Performance of ELK stack and commercial system in security log analysis", in *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, pp. 187–190, 2017.

[35] C. Wurm, "Analyzing Network Packets with Wireshark, Elasticsearch, and Kibana". `https://www.elastic.co/blog/analyzing-network-packets-with-wireshark-elasticsearch-and-kibana`. Accessed 16.10.2021.

[36] IVRE, "IVRE - Network recon framework". `https://ivre.rocks/`. Accessed 16.10.2021.

[37] A. Tundis, W. Mazurczyk, and M. Mühlhäuser, "A Review of Network Vulnerabilities Scanning Tools: Types, Capabilities and Functioning", in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, (New York, NY, USA), Association for Computing Machinery, 2018.

[38] SolarWinds Worldwide, "Network Performance Monitor". `https://www.solarwinds.com/network-performance-monitor`. Accessed 10.01.2022.

[39] SolarWinds Worldwide, "Network Packet Capture (PCAP) Tool". https://www.solarwinds.com/network-performance-monitor/use-cases/packet-capture. Accessed 10.01.2022.

[40] C. B. Weinstock and J. B. Goodenough, "On System Scalability", tech. rep., Carnegie Mellon Software Engineering Institute, 2006.

[41] OpenJS Foundation, "About Node.js". https://nodejs.org/en/about/. Accessed 21.02.2022.

[42] OpenJS Foundation, "Node.js v17.5.0 documentation | Net". https://nodejs.org/api/net.html#net. Accessed 22.02.2022.

[43] Socket.IO, "Socket.IO | Introduction". https://socket.io/docs/v4/. Accessed 22.02.2022.

[44] Socket.IO, "Socket.IO | How it works". https://socket.io/docs/v4/how-it-works/. Accessed 27.02.2022.

[45] L. Ridder, "DDoS Packet Capture Collection". https://github.com/StopDDoS/packet-captures, 2021. Accessed 27.01.2022.

[46] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert, and C. Kemp, "Detection of SSH Brute Force Attacks Using Aggregated Netflow Data", in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 283–288, 2015.

[47] Carnegie Mellon University, "Protect Against Brute-force/Dictionary SSH Attacks". https://www.cmu.edu/iso/aware/be-aware/brute-force_ssh_attack.html, 2022. Accessed 11.02.2022.

[48] The Tcpdump Group, "MAN PAGE OF TCPDUMP". https://www.tcpdump.org/manpages/tcpdump.1.html, January 2022. Accessed 14.02.2022.

[49] Wikipedia contributors, "Federated learning — Wikipedia, the free encyclopedia". https://en.wikipedia.org/wiki/Federated_learning, 2022. Accessed 17.01.2022.

Abbreviations

ACK	Acknowledgement Flag (TCP)
API	Application Programming Interface
AS	Autonomous System
CLI	Command Line Interface
CSV	Comma-Separated Values
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
FIN	Connection Teardown Flag (TCP)
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
MB	Megabyte
OS	Operating System
OSS	Open-Source Software
PCAP	Packet Capture
SRE	Site Reliability Engineering
SYN	Synchronize Flag (TCP)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
VM	Virtual Machine

List of Figures

List of Tables

Appendix A

Installation Guidelines

This appendix provides information necessary to run the D-DDoSGrid prototype. The following instructions have been tested in a MacOS environment as well as on a Debian VM, and should be compatible with other distributions.

Prerequisites

1. The source code of this project, available on GitHub:
   `https://github.com/demaerl/ddosgrid-v2`
2. The necessary libraries and tools to build and run this project:
   - `Node.js`
   - `npm`
   - `libpcap`

Build and Run Prototype

Enter the miner subproject and install the necessary dependencies:

```
1 cd miner
2 npm i
```

After that, the central instance can be run through a shell:

```
1 node index.js
```

The server will then wait for nodes to connect:

```
1 Server listening on *:3000. Waiting for client to connect...
```

To connect a worker node, the path to a PCAP file needs to be passed as argument.

```
1 node worker.js pcap_path='/path/to/pcap_file_A'
```

This will run the miners and transmit the results to the central instance.
The output of the worker node will look as follows:

```
Checking input values...
Setup of the following miners has completed (0.001s):
    - Miscellaneous Metrics
    - Top 20 UDP/TCP ports by number of segments
    - Ratio between UDP and TCP segments
    - Analysis of IPv4 vs IPv6 traffic (based on packets)
    - Top 5 source hosts (IPv4)
    - Most used HTTP verbs
    - Top 10 most used Browser and OS Combinations
Analysis has started...
Decoding has started...
    0.001 * 10^6 PCAP packets analysed. Current Heap Memory usage: 20MB
Decoding has finished (0.768s), sending interim results to server...
```

The output of the central node will look as follows:

```
A client connected. ID: 9ULJO3Vc-C6J2uKUAAAB
Received interim result from client (ID: 9ULJO3Vc-C6J2uKUAAAB).
Starting post-parsing analysis of interim result...
Post-parsing analysis has completed. The results are available at '/path
   /to/pcap_file_directory/'
A client disconnected. Reason: server namespace disconnect. ID: 9ULJO3Vc
   -C6J2uKUAAAB
```

To connect additional worker nodes, use the same command as above:

```
node worker.js pcap_path='/path/to/pcap_file_B'
```

Appendix B

Contents of the CD

1. This thesis as PDF
2. This thesis as LaTeX source in a `.zip` file
3. Midterm presentation slides as PDF
4. The source code of this thesis
5. The datasets used for the evaluation, in a directory called `datasets`