



University of
Zurich^{UZH}

GridDB - Enhanced Visualization and Sharing of DDoS Fingerprints

Karim Khamaisi
Zurich, Switzerland
Student ID: 19-740-067

Supervisor: Dr. Bruno Rodrigues, Jan von der Assen,
Prof. Dr. Burkhard Stiller
Date of Submission: 07.09.2022

Zusammenfassung

Mit der zunehmenden Bedeutung von Online-Diensten steigt auch die Verantwortung seitens der Betreiber, ihre angebotenen Leistungen auf digitalen Plattformen sicher und zuverlässig zur Verfügung zu stellen. Während der Covid-19 Pandemie haben wir gesehen, wie wichtig die Digitalisierung in unserem Leben ist. Beispielsweise im Bildungswesen konnten Studierende trotz der Krise Vorlesungen online besuchen und Prüfungen digitalisiert ablegen.

Es ist jedoch riskant, sich bei der Erbringung einer Dienstleistung vollständig von digitalen Geräten abhängig zu machen, da die geforderte Leistung bei einem möglichen Ausfall der digitalen Komponenten nicht offline erfüllt werden kann. In diesem Zusammenhang gibt es viele mögliche Ursachen, warum ein Online-Dienst plötzlich nicht mehr im vollen Umfang funktionieren kann; zum Beispiel ein Stromausfall, ein fehlerhafter Quellcode oder ein Cyberangriff.

GridDB konzentriert sich auf ein bekanntes Problem in Computernetzwerken, den Distributed Denial of Service (DDoS)-Angriffen. DDoS-Angriffe stellen eine grosse Bedrohung für die Verfügbarkeit des Internets dar. Trotz zahlreicher kommerzieller und wissenschaftlicher Bemühungen konnten die Angriffe und deren Konsequenzen nicht eingedämmt werden. Solche DDoS-Angriffe passieren jeden Tag. Da die Gesellschaft einem zunehmenden Digitalisierungstrend folgt, stellen Angriffe dieser Art eine nicht zu vernachlässigende Bedrohung für Unternehmen und Privatpersonen dar.

In früheren Arbeiten wurde das DDoS-Clearing-House-System aufgebaut, um DDoS-Fingerprints auszutauschen. Dies ist eine digitale Signatur von DDoS-Angriffen, die von Partnern in einer kooperativen Allianz zur Bekämpfung von DDoS-Angriffen registriert wird. GridDB bietet ein einfaches Frontend, um die Handhabung von Paket-Erfassungen (PCAPs) und den Austausch von Informationen zu verwalten. Daher sollte das Frontend eine Schnittstelle zum DDoS Clearing-House haben, um dessen Status aktualisieren zu können. Zu diesem Zweck wurden portable Backend- und Frontend-Anwendungen implementiert. Das Hauptergebnis dieser Arbeit ist die Integration der DDoSCH-Komponenten in ein vollautomatisches und portables System, das die terminalbasierte Anwendung des DDoS Clearing-House durch eine benutzerfreundliche Oberfläche ersetzt.

Die implementierte Anwendung ermöglicht es den Nutzern, die analysierten Angriffsdaten in verschiedenen Visualisierungen zu betrachten, um die Wahrnehmung und das Verständnis für die Art des Angriffs zu verbessern, was den Nutzern die Möglichkeit gibt, ähnliche Angriffe in Zukunft zu entschärfen und ihre potenzielle Infrastruktur zu schützen.

Abstract

As the significance of internet services grows, the need to maintain their online presence likewise grows. During Covid-19, we witnessed the significance of digitization in our daily lives. For example, in Education, despite the crisis, students were able to attend online courses and take tests. But also in several other vital disciplines. However, relying only on digital gadgets to provide a service is problematic since we will be unable to complete the intended work if they go down. There are several reasons why an internet service might go down, including a power outage, faulty source code, or a cyber assault.

GridDB focuses on a well-known issue in computer networks, Distributed Denial of Service (DDoS) attacks. DDoS attacks are a significant threat to Internet availability that has not been resolved despite several commercial and academic initiatives. DDoS attacks occur on a daily basis, and as society continues to digitize, these attacks offer several dangers to organizations and people.

In previous work, the DDoS Clearing-house system was built on sharing DDoS fingerprints, a signature of DDoS attacks registered by partners in a cooperative alliance to counter DDoS attacks. GridDB provides a straightforward Frontend to manage handling packet captures (PCAPs) and sharing information. Thus, the Frontend should interface with DDoS Clearing-House to update its status.

Therefore, portable Backend and Frontend applications were implemented. The primary outcome of this work is the integration of the DDoSCH components into one fully automated and portable system, replacing the terminal-based application provided by the DDoS Clearing-House with a user-friendly UI.

The implemented application allows users to view analyzed attack data in various visualizations to increase the perception and understanding of the attack nature, which gives users the ability to mitigate similar attacks in the future and to protect their potential infrastructure.

Acknowledgments

I want to thank everyone who helped to increase the quality of this work in one way or another, especially the people from the communication system group. Also, I would like to thank Prof. Dr. Burkhard Stiller, who allowed me to write my bachelor thesis in his department.

Further, I want to thank my supervisor Dr. Bruno Rodrigues for his helpful insights and instant support during the whole time.

Last, I want to thank my family and friends who helped and stood by me during this work.

Contents

Zusammenfassung	i
Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Thesis Goals	3
1.2 Methodology	3
1.3 Thesis Outline	3
2 Fundamentals	5
2.1 Background	5
2.1.1 System Usability Score	5
2.1.2 Attack types	6
2.1.3 Detection patterns	10
2.1.4 Situational awareness	11
2.2 Related Work	12
3 Design	15
3.1 Requirements	15
3.2 Architecture	16
3.2.1 DDoSCH	16
3.2.2 DDoSGrid 2.0	18

3.2.3	GridDB	20
3.3	Implementation	23
3.3.1	DDoSSCH	23
3.3.2	DDoSGrid V2.2	25
3.3.3	GridDB	27
4	Evaluation	37
4.1	Usability	37
4.2	Portability	44
4.2.1	DDoSGrid V2.2	45
4.2.2	GridDB	46
4.2.3	DDoSDB V2.2	49
4.3	Automation	49
4.4	Discussion	50
5	Final Considerations	53
5.1	Summary	53
5.2	Conclusions	54
5.3	Future Work	55
	Bibliography	56
	Abbreviations	63
	List of Figures	63
	List of Tables	66
	List of Listings	67

Chapter 1

Introduction

No doubt COVID-19 greatly impacted our life negatively. However, the economic downturn enabled many organizations to digitalize their business models and perform internal responsibilities online, *i.e.*, online meetings. Nevertheless, moving forward with a more digitalized world increases the probability of being a victim of a cyber-attack. Countries', organizations', and individuals' dependencies on digital devices make it essential to protect the data; protection against the highly sophisticated cyber-attacks nowadays is a difficult subject [40, 42].

Despite today's cybersecurity evolution, DDoS (Distributed Denial-of-Service) attacks are still included in the most severe and challenging internet issues. DDoS attacks are extremely harmful since they are inexpensive to execute, despite the fact that they may cause enormous financial and reputational harm to their targets[21]. To perform a DDoS attack, one must not be an expert in cyberattacks. One can buy a DDoS attack online using a booter; these websites provide a DDoS attack as a service, and booters are one of the main reasons for the evolution in the number of DDoS attacks [7]. Booters are accessible on the internet and are easy to find; they are also affordable in terms of price, and last but not least, they do not require any technical understanding of the attack itself [7].

For example, a small business that was a victim of a DDoS is estimated to lose up to \$120'000, which increases a lot for larger companies [51]. According to the 2021 DDoS Threat Landscape Report, DDoS attacks evolve each year in frequency, complexity, and volume [30]. Compared to 2020, the number of attacks per month and the number of packets sent over the network aiming to shut down a server or service increased [30].

When there is a conflict between two countries or when performing a military operation, *i.e.*, invasion, such operations come with cyber-attacks; the main target of attacks in cyberspace is to disrupt the communication inside that target country and isolate it from the media [62]. DDoS attacks are a prevalent and powerful tool to render essential websites' access to a country unavailable [62].

Two very recent examples are the cyber-attacks performed against Ukraine and Israel. DDoS attacks against Ukraine's military and financial institutions just before and during

the Russian invasion on February 24 were performed and attributed to Russian hackers [2]. However, the attack was not sufficient to make the websites go offline [2]. In contradiction, on March 14, essential government websites were unavailable for some time in Israel due to a massive DDoS attack [1].

Recent research showed that the top source country generating DDoS attacks during 2021 was the United States generating more than 50 percent of DDoS attacks recorded; the following countries were Europe, India, and East Asia with different activities through the quarters of the year [48]. A common reason is the large number of unsecured devices connected to the Internet and their growing processing capacity, which allows attackers to take control of a vast amount of unsecured devices that range from connected cameras to smart fridges to launch malicious attacks [69]. Many of these devices are insecure by design, and not often they are impossible to be secured due to their hardware and software constraints.

The evolvement in the attack size and complexity made it harder for individuals and small businesses to counter DDoS attacks independently. Thus, the trend toward countering DDoS attacks in a cooperative approach is increasing [55, 57]. In related work, we noticed two art of systems: a centralized and decentralized system for the cooperative defense against DDoS attacks. However, centralized cooperative systems are usually overwhelmed with massive data and face scalability, trust, and fairness issues [55]. As a solution for those issues, the authors of this work [52, 54, 55] suggest a system based on Blockchain and smart contracts to overcome the centralization and trust issues of the system.

While the attackers send many requests over the network (packets), they also leave traces behind. The analyzed traces from an attack pattern are called fingerprints. It is possible to get insights into those traces by investigating the captured exchanged traffic over the network. Such insights can be obtained by the network traffic, which is logged by networking equipment and made available by Packet Captures (PCAPs) or other file types, *e.g.*, flow-based data structure. PCAPs provide more comprehensive information about the traffic and are thus preferable to flow-based data structures when trying to discover new patterns of a DDoS attack [4]. In order to evolve in the mitigation process, we need recent and reliable PCAPs from DDoS attack victims; this will help researchers detect patterns of the DDoS attack's nature and handle them accordingly.

Luckily, such a system to share fingerprints between different partners is implemented and called the DDoS Clearing House (DDoSCH) [22]. DDoSCH is a terminal-based system used to share such fingerprints and facilitate the sharing process of DDoS attacks' information. The DDoSCH is composed of multiple modules available in a GitHub repository and consists of three main components:

1. **Dissector**: is installed on the victim's machine; it is primarily responsible for summarizing the network traffic upon the attack and generating valid fingerprints.
2. **DDoSDB**: is the backend database to share fingerprints.
3. **Converters**: are installed on potential DDoS victims' machines and responsible for translating fingerprints into mitigation rules.

The primary goal of this work is to build a straightforward Frontend to manage the handling of PCAPs and sharing information, taking as a basis the running terminal-based backend, fully integrated within the DDoSGrid system. DDoSGrid is an open platform developed in the scope of a master project at the University of Zurich, aiming to simplify the extraction process of shared PCAPs by providing meaningful visualization allowing a closer understanding of the nature of a DDoS attack [26, 65]. Similar work to the DDoSGrid is the SecGrid system [25], which was also implemented at the University of Zurich. In this work, the authors implemented a system to visualize and analyze attack data, employing a Machine Learning approach to automatically classify the provided traffic type while analyzing the uploaded dataset.

1.1 Thesis Goals

- Implement a Frontend allowing the Web-based interaction with the DDoSCH and integrate it within the DDoSGrid system as an additional tab.
- Integrate the new DDoSDB version within the DDoSGrid.
- Provide a portable version for each of the employed applications.

1.2 Methodology

First, we studied the nature and types of DDoS attacks. Then, after we gained insights into the attack types, we investigated current related work and tried to understand the characteristics of systems that counter DDoS attacks. Finally, according to our studies of the current work in the field of mitigation of DDoS attacks, we derive the requirement of a possible system to protect potential victims against DDoS attacks. We took the functioning terminal-based applications of the DDoSCH components and implemented a straightforward Frontend to manage PCAP files in order to replace the inconvenient CLI with a user-friendly interface.

1.3 Thesis Outline

Chapter 1 introduces the subject of the DDoS attack, points out the followed methodology, and mentions the thesis goals. Chapter 2 presents background knowledge about **DDoS attacks** and describes related work. Chapter 3 describes the requirements to meet the goals and shows the architecture behind the different components that should fulfill those, also it includes information about the hardware and software implementation of **GridDB, DDoSGrid V2.2, and DDoS-DB V2.2**. Chapter 4 explains and evaluates the enhancement of the GridDB system. The last Chapter 5 concludes the thesis in a final consideration including a summary, conclusions, and future work.

Chapter 2

Fundamentals

2.1 Background

One primary goal of this thesis is to develop a frontend application to integrate the new versions of DDoS Clearing House components into one system. Driven by the last goal, the second goal of the thesis is to provide an evaluation of the system using the System Usability Score. Thus, an overview of the System Usability Score is provided in this subsection. The rest of this subsection is structured into three parts. The first part provides an overview of the most in literature and practice of known DDoS attacks types [52]. It gives a constructive characterization of the attack type and a description of examples. The second part provides an overview of DDoS attacks' detection patterns, and the last part covers situational awareness. The related work subsection is the last part of this chapter 2.

2.1.1 System Usability Score

It is Also known as System Usability Scale (SUS). SUS comes in handy once we are interested in examining a product's usability since it provides tools to quickly collect users' subjective ratings of some services [5]. Brooke developed SUS in the late ninetens; the original survey was composed of 10 statements that the survey participant could evaluate on an agreement level by providing a score ranging from 0 to 5 [5]. So naturally, the higher the score, the higher the usability. One can modify the SUS content as needed; an example of the original compared to a modified version of a SUS is shown below [5].

Original SUS Statements	Modified SUS Statements
I think that I would like to use this system frequently	I think that I would like to use this product frequently
I found the system unnecessarily complex	I found the product unnecessarily complex
I thought the system was easy to use	I thought the product was easy to use
I think that I would need the support of a technical person to be able to use this system	I think that I would need the support of a technical person to be able to use this product
I found that the various functions in this system were well integrated	I found that the various functions in this product were well integrated
I thought that there was too much inconsistency in this system	I thought that there was too much inconsistency in this product
I would imagine that most people would learn to use this system very quickly	I would imagine that most people would learn to use this product very quickly
I found the system very cumbersome to use	I found the product very awkward to use
I felt very confident using the system	I felt very confident using the product
I needed to learn a lot of things before I could get going with this system	I needed to learn a lot of things before I could get going with this product

Figure 2.1: Original vs modified SUS [5]

2.1.2 Attack types

Table 2.1 briefly describes the three primary attack categories, with some examples based on a CloudFlare article [16]. The following subsections provide a constructive description of each attack type.

Category	Description	Examples
Application Layer Attacks	Attackers first connect with the target's server and then exhaust its resources, trying to monopolize the transactions in the web application.	HTTP Flood attack
Protocol-Based Attacks	Attackers try to overload critical resources by consuming all their processing capacity, causing disruption of some service.	Syn Flood and Ping of Death
Volumetric Attacks	Attackers send heavy requests to the victim's server to consume its bandwidth. This category is the easiest to generate and thus the most common type of attack.	UDP Flood, TCP Flood, NTP Amplification and DNS Amplification

Table 2.1: Categories of DDoS attacks

2.1.2.1 Botnets, Booters and OCI model

Before defining the different DDoS attack types, it is essential to understand the concept of Botnets and Booters, their role within an attack, and comprehend the difference between them. In addition, this subsection gives an overview of the OSI (open systems interconnection) model.

Botnets: the word **Botnet** is derived from the two words **Robot** and **Network**; an attacker tries to infect as many devices as possible, and each infected device is then a bot and part of a group executing malicious or illegal actions [17]. The hacked devices, also called unwilling victims, are controlled remotely by the BotMaster and execute malicious activities without the acceptance of their owners [9, 41].

Booters: in a brief, straightforward description, Booters are a paid DDoS service offered by cybercriminals to attack a victim's website and render it inaccessible for visitors [11]. The price to hire booters is remarkably cheap; usually not more than \$10 per month, depending on the duration and number of attacks performed specified by the user who purchased the subscription [35].

The main difference between **Booters** and **Botnets** is that Booters are offered as Frontend websites and can be used legally to test one's websites; for the attacks (or the tests), Booters maintain private servers [35]. Conversely, Botnets consist of infected devices controlled by a third party to launch an attack without the devices owners' acceptance. Nevertheless, booters typically use Botnets in their attacks [11].

OSI Model: the International Organization for Standardization created the OSI (open systems interconnection) model to facilitate troubleshooting network problems, helping isolate the source of the issue; it divides the communication systems into seven stacked abstracts layers [20]. Commonly, DDoS attacks occur at the OSI model's 3, 4 and 7 level [19]. The Figure 2.2 illustrates the different OSI layers [20].

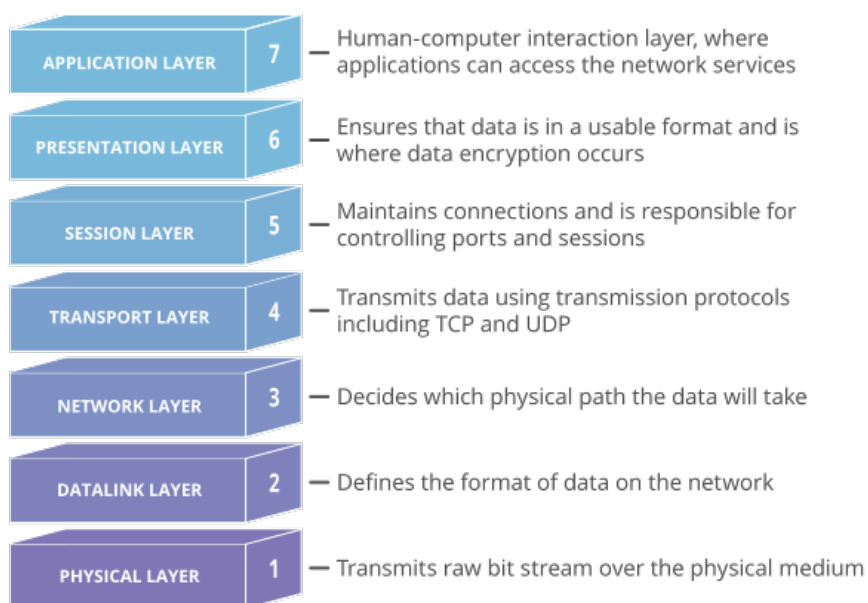


Figure 2.2: 7-layers open systems interconnection (OSI) model [20]

2.1.2.2 Application Layer Attacks

Application layer attacks (ALA), also known as layer seven attacks, where network requests such as HTTP GET, HTTP POST take place, are more sophisticated than other DDoS attacks, *i.e.*, on the network layer, and thus are more challenging to detect [18, 50]. ALA are executed with a few "light" requests to the victim's server, which allows them to overcome volume-based detection mechanisms [50]. ALA are effective not only because they do not require considerable resources from the attackers' devices since they use simple network requests but also it is hard to distinguish legitimate from illegitimate received requests [18]. Few examples of ALA based on [29] are listed below:

- **Request-Flooding Attacks:** attackers send legitimate normal requests, *i.e.*, HTTP GET or HTTP POST, at a more increased rate consuming the server's resources.
 - **HTTP GET attack:** multiple attackers' devices fetch a file or an image from the victim at a coordinated point in time [12].
 - **HTTP POST attack:** multiple attackers' devices send data that is usually saved in the website's database, *i.e.*, submitting some form; queries and database commands to save the pushed data are, in comparison to the sent request, extensively more intensive [12].
- **Asymmetric Attacks:** attackers send high workload requests (not necessarily at a higher rate) to consume the server's resources, for example, the CPU or RAM.
- **Application-Exploit Attacks:** attackers exploit possible vulnerabilities of the operating system where the victim's servers are running. Then, using the sent requests, attackers try to cause errors in the operating systems, such as SQL injection and buffer overflows.

2.1.2.3 Protocol Based Attacks

Protocol-based attacks (PBA) exploit the weaknesses in layers 3 (Network Layer) and 4 (Transport Layer) of the OSI model [14]. One of the most illustrative attacks to understand PBA is the **SYN (Synchronized) flood**. SYN flood attacks were one of the first attacks to appear and are the most famous attack on the network layer today [13, 53, 59].

SYN flood utilizes the characteristic that is exploitable of the Transmission Control Protocol (TCP); SYN is a type of packet sent between the target and source server to establish a reliable communication and create the connection between the two hosts [59, 61]. According to the TCP protocol specification, the connection creation process follows the "*Three-Way Handshake*" sequence to establish reliable communication between the hosts [31, 61]. Attackers send SYN packets to the "*listening*" ports in the target host; SYN packets contain the IP address of the source host, but in DDoS attacks, the attacker organizes the SYN packets to include spoofed source addresses, which usually do not exist [59]. According to the three-way handshake sequence, after the target host receives the

SYN packet, it responds with an SYN/ACK (acknowledgment) packet [59, 61]. Then, it waits for the source host's ACK packet to respond to finish the connecting process successfully. However, in an SYN flood attack, the attackers send SYN packets with invalid addresses. Therefore, the waiting ports in the target servers will never receive an answer (confirmation ACK messages), thus waiting in vain or until the connection is timed out [31, 59, 61]. With that, the attackers succeed in consuming and occupying the victim's host network resources [15, 59]. As a result, it will no longer be able to establish new connections with legitimate new SYN requests [15, 59]. Figure 2.3 below illustrates the SYN flood attack.

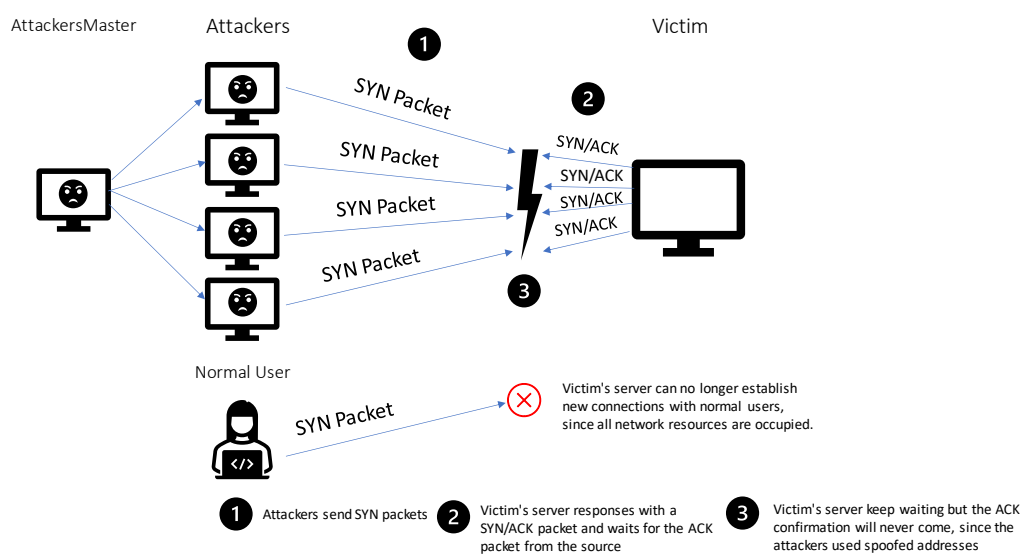


Figure 2.3: SYN Flood attack based on [15, 59]

2.1.2.4 Volumetric Attacks

Like PBA, attackers try to devastate the victim's network, memory, or computing resources and employ imperfections in the network and transport layer of the OCI model [31, 39]. Specifically, attackers aim to consume the bandwidth of the victim's network by sending remarkably high volumes of malicious traffic [46]. To further understand Volumetric attacks, an example of an explanatory known volumetric attack is provided below, namely the Reflection Amplification Attack.

Reflection Amplification Attack: this attack strategy consists of two different attack types combined, 1) reflection attack and 2) amplification attack. Attackers first fake the address of the sent request to be the same as the victim's address (Reflection part), then attackers design requests whose responses are considerably more extensive than the request itself (amplification part) [58]. The two parts are defined individually beneath, and the Figure 2.4 embodies the attack.

- **Reflection Attack:** attackers explore servers that provide open services that reply to any requests without authentication of the source, such as Domain Name Systems (DNS) and Network Time Protocol (NTP) [8]. DNS is a service to translate a domain address from numbers (*e.g.*, an IP address) into a user-friendly domain name; NTP has its goal to synchronize the time among computer systems over the network [8]. Attackers spoof the target's IP address and send requests using the User Datagram Protocol (UDP), which does not require a *"handshake"* (authenticity of the source) to establish a connection [45, 58]. The responses to the requests made to the servers with open services are directed to the target [8, 45].
- **Amplification Attack:** attackers structure their requests to have an amplified response compared to the request size [10, 45]. For example, a DNS response to solve a name are typically larger than the request itself, creating an opportunity to be leveraged by attackers. Using Botnets and thousand of coordinated requests cause damaging issues to the target's bandwidth resulting in a denial of service [45].

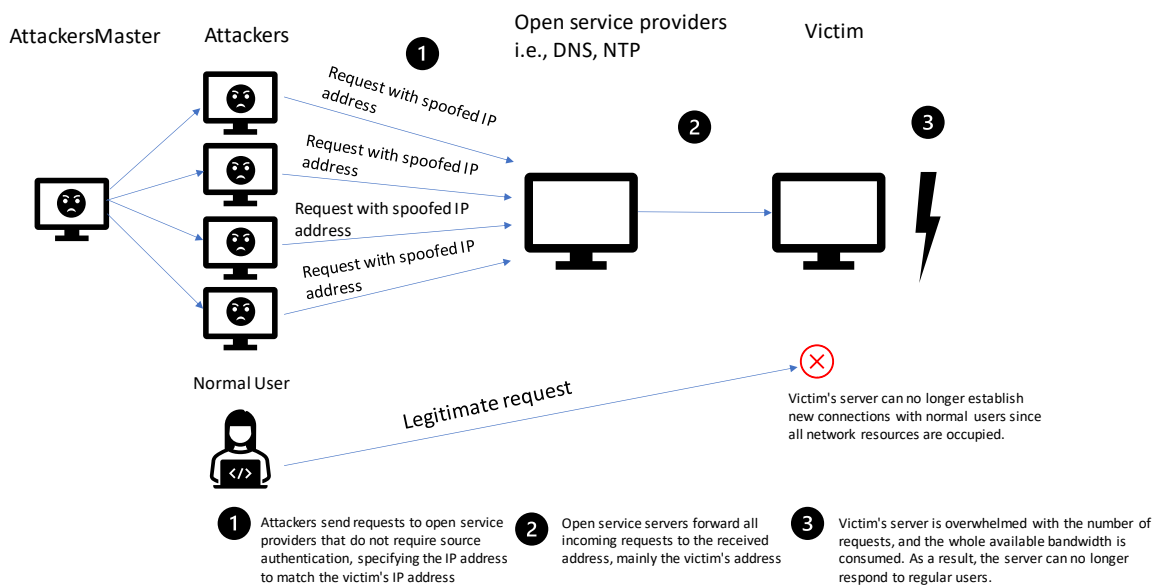


Figure 2.4: Reflection Amplification attack based on [8]

2.1.3 Detection patterns

There are several reasons why DDoS attack detection is a demanding task. Based on [27], a few of many others reasons are listed below.

- The amount of open-source, available malicious tools on the internet.
- Spoofed IP addresses are usually used during a DDoS attack.

- The absence of a benchmark for DDoS protection and detection tools.
- A massive number of new systems rely on the internet.

Nevertheless, many systems propose a detection system for a DDoS attack. For example, once searching for the keywords "*DDoS Detection*" in Google Scholar, almost 88'000 results are displayed. The several works use different approaches and complex mechanisms. The rest of this subsection shows some signs and detection patterns of previously introduced attack types.

Request-Flooding Attacks: this kind of attack employs numerous GET and POST requests coming from the same IP address simultaneously [34].

SYN flood: as stated before, connections that utilize TCP protocol establish the connection in a specific sequence of exchanged messages. A sign indicating an SYN flood attack is the massive number of flow compared to the sent packet per flow [60].

Reflection Amplification Attack: these attacks are evident due to their unusually high bandwidth; furthermore, they are manageable to detect since they usually involve packets that share the same source port with a single target [45, 46, 56].

2.1.4 Situational awareness

In cyberspace, situational awareness aims to encourage decision-makers in different organizations and institutes to analyze surroundings threats and create meaningful visualizations to understand and predict the situation quickly [63]. For this purpose, the Three-layer of situational awareness model by Endsley presented in [KillerDashboard19 SitAwareness3LayMod 63] is beneficial. It states the following:

1. Layer 1 - Perception: according to the state of the surrounding environment, we perceive signs and indicators. Therefore, hands must be analyzed and evaluated.
2. Layer 2 - Comprehension: constructive understanding of the signs from layer 1.
3. Layer 3 - Projection: try to predict the impact of the crucial factors on the situation for the coming future.

According to layer one, victims or targets have to keep an eye on network traffic tools that gather and log network information made available by packet captures (PCAPs) in the context of a DDoS attack. PCAPs provide insightful information to detect a DDoS attack pattern [4]. At the second layer, victims of a DDoS attack have to understand and analyze the gathered PCAPs. One technique to achieve this is to analyze and visualize the data to facilitate the perception and comprehension task. This work aims to provide enhanced and further detailed visualization of PCAPs; this helps with the last and third layer, which is projection. Decision-makers must conclude and develop protection and mitigation mechanisms for future attacks in this layer.

2.2 Related Work

- **DDoS Clearing House:** the DDoSCH (DDoS Clearing House) [22] is an open-source project co-funded by the EU's Horizon 2020 Research and Innovation program. It consists of different modules with the primary goal of collaboratively combatting DDoS attacks. As mentioned in the introduction section 1, the DDoSCH includes three main modules:
 1. **DDoS Dissector:** its direct responsibility to extract DDoS attack's characteristics from the available traffic capture files making those available in a JSON file. The DDoS Dissector module is installed on a DDoS victim's machines.
 2. **DDoSDB:** is a MongoDB Database of the whole system, where the fingerprints generated by the DDoS Dissector are saved.
 3. **Converters:** these Modules are responsible for the protection process of a possible DDoS victim. It interacts directly with DDoSDB and converts fingerprint JSON files into a set of mitigation rules.

The DDoSCH provides the basis milestone for this work since it supplies the running terminal-based backend.

- **DDoSGrid:** DDoSGrid systems were developed to identify DDoS attacks' patterns and characteristics by analyzing captured network flow information and providing meaningful visualization to further increase the situational awareness of different stakeholders [3, 6, 49, 65]. There are three open-source DDoSGrid systems where each new version brings new utilities. The three systems are briefly described below.
 - **DDoSGrid 1.0:** the first version of DDoSGrid systems was implemented in the scope of a master project at the University of Zurich. One of the project goals was to deliver visualizations of analyzed captured data files such as PCAPs. The main objective was to understand the attack characteristics through the different visualizations and help decision-makers and researchers understand and develop new mitigation mechanisms.
 - **DDoSGrid 2.0:** the second version focused on integrating the DDoSCH into the previous DDoSGrid system [3]. It was built as a part of a Master's Thesis at the University of Zurich. DDoSGrid2.0 used an old version of the DDoSDB, mainly DDoSDB 2.0. In the meanwhile, a new version of DDoSDB has been released. This bachelor thesis aims to integrate the new version of DDoSDB3.0 into the DDoSGrid2.0 system.
 - **DDoSGrid 3.0:** the latest DDoSGrid system's version was also developed at the University of Zurich [49]. DDoSGrid3.0 includes the real-time analysis factor to visualize the network and thus detect, react, and, if possible, stop a potential DDoS attack before causing enormous damage.
- **DDoSDB:** the DDoSDB is a part of the DDoSCH and an independent web-based application written in the Django framework [22, 64]. Its purpose is to share DDoS

attack information among different stakeholders and understand and improve mitigation strategies. In addition, the DDoSDB makes it possible to learn about particular characteristics of attacks since it facilitates the sharing of fingerprints after anonymizing those to keep the victim's identity unknown.

- **MISP (Malware Information Sharing Platform):** this work assumes that handling individually recent complex IT threats is a demanding or even an impossible task [66]. That is why the authors introduce a trusted MISP enabling the collaborative-knowledge-sharing about threats in cyberspace. Collaborators can add new entries called *events* with different levels of granularity. It is possible to include attachments and characteristics of relevant information; these are called *attributes*. Once sharing a new *event*, Collaborators can specify different *sharing levels*, *i.e.*, with people in the same organization or with all users on the platform.
- **DDoS Playbooks:** A DDoS playbook is a tool to guide a company or an organization on specific, clear steps during different stages of being under attack [32]. Playbooks can be combined, adjusted, and extended to increase the stakeholders' understanding [32]. An example of a live DDoS Playbook [33] and a guide [47] on creating such playbooks are illustrative for understanding DDoS Playbooks.
- **SOChain:** Like other works, this work assumes that victims or security operation centers (SOCs) are hesitant to share their real-time DDoS data because of trust and fairness issues [28, 68]. SOCchain proposes a decentralized system using blockchain technology to overcome those issues. They further believe that the best mitigation is a blacklist of shared IP addresses and a platform to share relevant attacks' information.
- **Swarm intelligence:** This work proposes a mitigation and detection system based on multi-agent communication [36]. It introduces several types of agents:
 1. Coordination agent: takes decisions and handles intermediate communications among other agents.
 2. Detection agent: as its name, this agent is to detect possible DDoS attacks.
 3. Monitoring agent: this agent is always in live mode, and it mentors the network and available cloud resources.
 4. Recovery agent: this agent is triggered by a sub-agent of the coordination agent (the decision-making agent (DCMA)). This agent is triggered once a DDoS attack is detected; it will take the available resources of the attacker and log essential information about the attacker and provide them to the DCMA to make more precise decisions.

Chapter 3

Design

During the literature review phase, we studied a set of existing applications whose goal is to counter DDoS attacks collaboratively; this helped to derive the needed requirements to develop a system with a similar goal. It was also helpful to understand the crucial traits such a system should have. Thus, in this chapter, we present the high-level requirements, driven by the literature review and further investigation of different existing systems. Second, we will describe the structure of the existing system and the new system. Furthermore, the last subsection will include fundamental implementation decisions.

3.1 Requirements

An initial requirement was to have both DDoSGrid 2.0 and the used version of the DDoSCH up and running. As mentioned in the literature review, there is already a new version of the DDoSGrid system (DDoSGrid 3.0). However, we decided to use the old but more stable version to accomplish one of the thesis goals: integrating the new version of DDoSDB into the DDoSGrid system. In the architecture subsection, we describe the architecture of both systems in depth. An essential requirement with a more technical nature is to provide a fully automated system, demanding from the user only two things:

1. **PCAP file:** this file in the form of *".pcap"* file extension should include a snapshot of the network flow to be analyzed. The user should only upload the file once in one place; all main system functionalities shall be triggered without further intervention.
2. **Authentication token:** a user can acquire such a token by being registered in the DDoSDB user model. The new version of the DDoSDB uses the Django framework that provides a powerful role-based user model. For example, an Administrator can generate tokens for other users, which allows them to upload fingerprints using the DDoS-Dissector module. Thus, a user does not necessarily have to create a new DDoSDB account to use GridDB or DDoSGrid systems.

Our goal is to have a system that integrates all DDoSCH house modules in one platform, where utilizing the token and the uploaded PCAP file:

1. Upload the fingerprint of the analyzed PCAP file to the DDoSDB (**sharing objective**).
2. Envision the most important key metrics of the attack data (**visualization objective** [37, 38]).
3. Examine the attack's data key metrics using the provided visualizations (**investigation objective**).

Another requirement is using the DDoSCH components with a friendly UI instead of having to deal with CLI. A user can operate the GridDB system and has to be able to use the Dissector, DDoSDB, and Converters modules independently of each other (besides the automated process and not instead). When using the automated process, *i.e.*, running all components of the DDoSCH, the endpoints will be protected using the headers, which should include the authorization token. For example, it should be possible for a user to generate a fingerprint without uploading it to the DDoSDB. In summary, we want the GridDB system to bring utility to users who are not members of a DDoSDB.

After running the Dissector module, a summary of statistics of the attack characteristics is generated and is called a fingerprint. Depending on the DDoS attack type, more statistics are included in the fingerprint. From those metrics, we want to provide meaningful visualization to understand the fingerprint and detect patterns more efficiently rather than dealing with the generated JSON file. The dissector's GitHub repository [23] clarifies the included metrics and further information about the DDoS Fingerprint format.

3.2 Architecture

This section provides an overview of the three main components in this thesis: **DDoSSCH**, **DDoSGrid 2.0**, and the new, improved system **GridDB**. As mentioned in the literature review part, a newer version of the DDoSGrid allows real-time analysis and network flow visualization. However, after inspection of both versions, we decided to work with the more stable version: DDoSGrid 2.0. We provide a description of its components, how they interact with each other, and what changes we made to integrate the new version of DDoSCH. DDoSCH's new version has made significant changes to the system. We highlight a couple of those changes in this subsection. For clearness, we use DDoSCH 2.0 and 2.2 as version notation for the different DDoSCH versions used in the DDoSGrid 2.0, though the DDoSCH has not specified such version notation.

3.2.1 DDoSCH

In this subsection, we highlight the three main components of the DDoSCH and explain how the user can interact with the different components at the top level and what type of

files are needed to execute the corresponding module. The different modules expect some files as input to create the output and are not fully functional without a correct input type. However, the DDoSDB module can still be used without the need to upload a file to it. Using the provided UI with the powerful Django role-based user model, a *superuser*, for example, can allocate rights to other users, allowing them to upload fingerprints to the database. Also, a logged-in user can view and query fingerprints (if he/she has the corresponding rights).

- **DDoS Dissector:** this module can be used in two ways (using the command-line interface with changing the passed arguments). It can handle two types of input files: *.pcap* and *.nfdump*. The first way to use this module is to pass one of the network-flow files; the module then will generate a fingerprint in a JSON format. The passed network-flow file and the generated fingerprint can be saved and used externally, *i.e.*, for visualization purposes. Secondly, we can use it to dissect and directly upload the generated fingerprint to the DDoSDB. The only notable change between DDoSDissector 2.0 and 2.2 is the amount of data extracted from the network-flow file. The newer version provides a more comprehensive fingerprint with more attack vector metrics. In addition, the new provided python source code is well refactored and well separated, making it easier to understand than the previous version. For example, in version 2.2, the number of lines in the *main.py* source code file decreased from 1117 to only 67 lines, which helped by decoupling the functionalities in well-separated modules.
- **DDoSDB:** this module serves as the central database of the DDoSCH utilizing a Django framework to manage the UI. There were two primary changes between DDoSDB 2.2 and 2.0. First is the authentication process, *e.g.*, how to authenticate a user who wants to upload a fingerprint to the DB. Using the Dissector module, it was possible to specify the path to the configuration file containing the DDoSDB hostname and the user's username and password. The Dissector then will establish a connection to the DDoSDB server and get the user's permissions, *i.e.*, calling the endpoint */my-permissions*. Finally, it checks if the user is eligible to upload fingerprints. Another way to authenticate the user was utilizing the authentication token created using the OAuth2 provider employing the Django OAuth Toolkit. The Admin of the DB has to create a new application using the OAuth provider, a client secret, and a client id. The admin can then add an access token to be assigned to a specific client id; this access token will be used in the Dissector module to upload a fingerprint. The new version does not use the OAuth2 provider anymore, but Django REST Framework Authentication assigns one or more tokens to a user. Similar to the old version, Dissector 2.2 accepts an authentication token that is directly specified in the configuration file (here without the user's username and password), which allows a user to upload a fingerprint. The second noteworthy change is the actual database search and analytics engine. In the new version, they used *MongoDB* instance to save and query fingerprints, whereas, in the previous version, *ElasticSearch* was used to search and query fingerprints.
- **Converters:** this module accepts a fingerprint file (JSON format) and generates firewall rules. It develops iptables, *i.e.*, a list of IP addresses used by the attacker;

those IP addresses will be blocked. Like Dissector 2.2, where the new version newly supports connection to the MISP system to upload a fingerprint directly, Converters 2.2 accepts fingerprints generated by the MISP system modules to generate iptables.

Figure 3.1 shows the three components of the DDoSCH and the environment where the components are supposed to run. First, the Dissector is installed on a DDoS attack victim, where the victim has access to the network flow data. Next, the victim uploads the network flow data to the dissector module, which will anonymize the data and extract the attack key metrics. Finally, a user can choose to upload the fingerprints to the DDoSDB, but the Dissector can be used independently to create the fingerprint without uploading it. For example, using the Dissector, a user can specify the DDoSDB hostname and his/her authorization token and then upload the generated fingerprint to the DDoSDB. If the fingerprint is shareable, every user in the system can view the anonymized fingerprint. Thus, the DDoSDB is a central database to share and view fingerprints among the connected users, which supports encountering DDoS attacks collaboratively. The last component is the converter, which accepts a fingerprint as a JSON format and generates a set of mitigation and firewall rules, *e.g.*, IP-Tables, to block IP addresses used to perform the attack. The converter is installed on a possible DDoS attack victim; having access to the available fingerprints in the DDoSDB, a possible victim can protect himself from similar DDoS attacks.

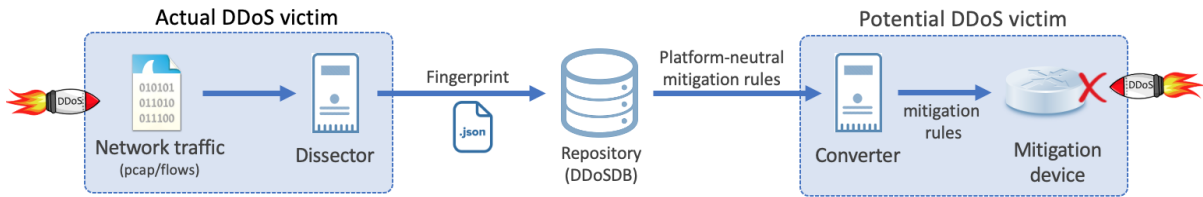


Figure 3.1: Overview of the DDoSCH components based on [22]

3.2.2 DDoSGrid 2.0

Figure 3.2 shows the architecture of the DDoSGrid 2.0 system. Besides the components of the DDoSCH, the DDoSGrid system implements three other main components:

- (i) Miners: to extract visualization metrics from a fingerprint, the output is a JSON file with the needed data to plot a chart.
- (ii) The front-End (Vue.Js) application as the UI to upload network flow files and display the visualizations.
- (iii) The Back-End Node Express server.

The components highlighted in light-red color needed to be adapted and adjusted to integrate the new version of DDoSDB forked from the DDoSCH project. Those were the

Front-End application and the corresponding Back-End. First, we deployed the system for development and had everything up and running using the old version of DDoSDB. The previous version of the DDoSDB is no longer available in the official repository of the DDoSCH, but it was provided in the DDoSGrid repository. After having both systems up and running, we investigated the connection points between the two systems. DDoSGrid 2.0 is bonded to DDoSDB in two ways. First, the authentication process of the user and the web application, and second the upload process of fingerprints.

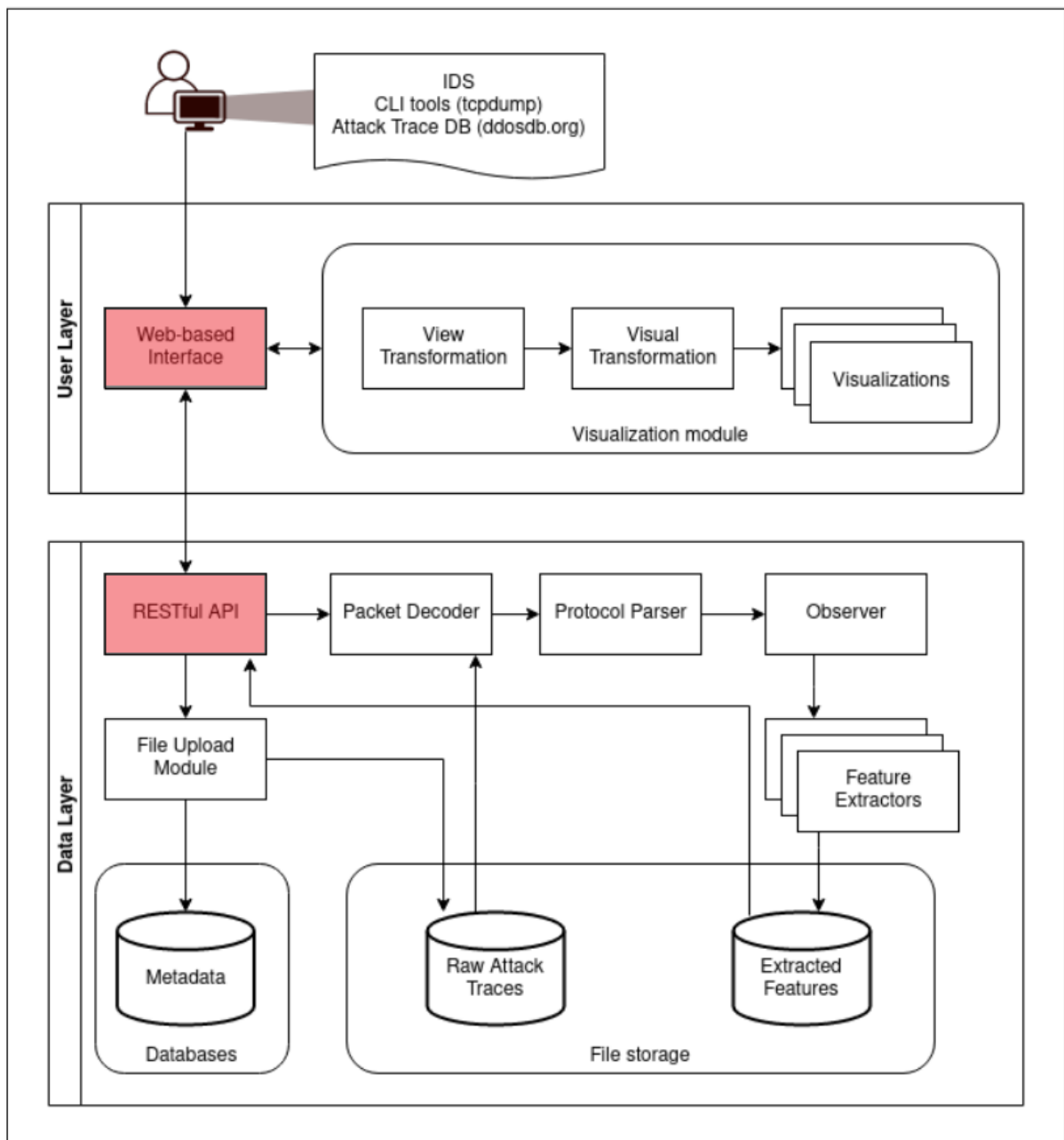


Figure 3.2: Architecture overview of the DDoSGrid components based on [3]

The new version of the DDoSDB authenticates users solely through assigning tokens; further, there is no need to authenticate the application per se that wants to interact

with DDoSDB. Now that we do not need to authenticate the application, we can remove all unnecessary back-end code previously responsible for that. However, we still need to implement a new authentication function to be aligned with the new type of authentication provided in the new version of the DDoSDB. Instead of forcing the user to log in to his/her DDoSDB account, we use the new simplified authentication approach. Now a user can type in his/her token using the UI, the token is being sent to the back-end, and then the back-end verifies if the token is valid by sending a GET request to the DDoSDB Django server. Django answers with a 200 status code if such a token exists.

Concurrently with the API changes in the authorization process, we also adjusted how DDoSCH modules run. The two modules, Converters 2.2 and Dissector 2.2, are changed to accept different parameters. The backend ran those two components as child processes; we also adjusted how the child processes were spawned—for example, Dissector 2.0 needed the DDoSDB username and password to be given as parameters when running the module. In contrast, the updated version 2.2 expected a configuration file with the DDoSDB hostname, user authorization token, and the protocol (HTTP or HTTPS). For that, we implemented a function to create the config file and pass it as a parameter to the dissector.

We also updated the front end by adding an input field for the user to enter the authorization token; this can be obtained from the DDoSDB, where a superuser can create and assign authorization tokens with different permissions to different users. In addition, the corresponding endpoints for the authorization were adjusted, *i.e.*, the way the front end is notified that the user is authenticated and is eligible to interact with the system.

The final adjustment took place in the docker files. Since we want to provide a fully dockerized version of both systems (GridDB and DDoSGrid 2.2), we adjusted those files accordingly. For example, the python version used in the dissector 2.2 changed to 3.8, so for the backend, we searched for a docker image that contains the needed node.js and python versions.

3.2.3 GridDB

We want to create this system as an additional tab to the existing DDoSGrid systems developed at the University of Zurich. The main objective of the new system is to provide a more stable and scalable application where all the DDoSCH components are integrated and can run automatically. Furthermore, as stated in the requirements section, the system should replace the CLI of the DDoSCH modules and run them separately from each other without the need to be authorized. *E.g.*, by using the URI `/dissector` or `/converters`, the user is redirected to the corresponding module and can operate it without needing an authorization token. Unfortunately, in the previous versions of DDoSGrid 2.0 and 2.2, a user can not interact with the system without being authorized (through an authorization token in DDoSGrid 2.2 or a valid login credentials of a DDoSDB account in DDoSGrid 2.0). The new GridDB system allows users that do not have a DDoSDB account or an authorization token to use the modules using a friendly UI to replace the CLI. In addition, we will implement a fully automated upload process, such that the user upload a PCAP file at one place and at one time.

GridDB consists of three main components Frontend, Backend, and DDoSCH components:

1. **Frontend:** is a *react.js* application using the *next.js* framework to render the UI. The Frontend is the starting point for the upload process of a PCAP file. It provides visualizations of the attack characteristics, and it replaces the CLI of the DDoSCH components in an understandable and straightforward web pages. In addition, we decided to utilize the custom server inside the next.js application to authorize the user and thus save code and API calls to the backend.
2. **Backend:** is an express node.js server responsible for running the DDoSCH components as child processes. In addition, it handles data storage, such as saving uploaded PCAP files and saving analysis data.
3. **DDoSCH components:** forked from the DDoSCH GitHub repository, the converters, and dissector modules run inside the Backend as child processes, whereas the DDoSDB runs as a separate web interface.

Figure 3.3 shows how the three systems are connected and how they interact. The dashed lines illustrate a RESTFUL API communication between the system/components.

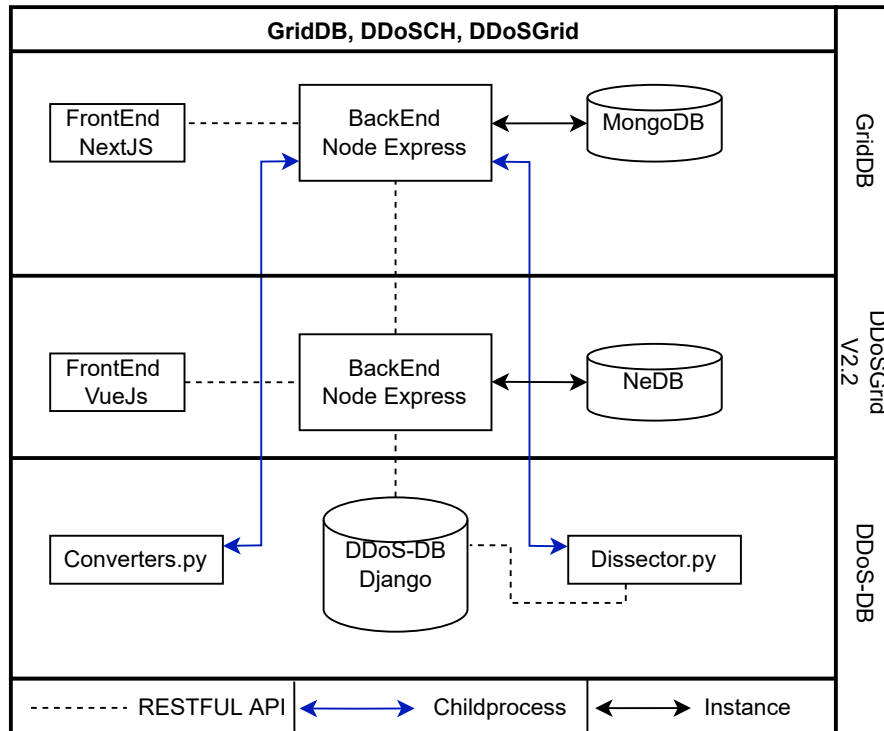


Figure 3.3: High Architecture overview of GridDB, DDoS-DB and DDoSGrid V2.2

We created Figure 3.4 to support the understanding of Figure 3.3. In Figure 3.3. In Figure 3.3, we showed each project and its sub-projects, *e.g.*, DDoSCH includes DDoSDB,

Dissector, and Converters. Figure 3.4 shows the actual environment in which a specific sub-project runs. For instance, the Dissector and Converters modules run inside the GridDB environment as child processes. Similarly, the Miner sub-project runs inside the DDoSGrid environment; we utilize its results through RESTFUL API between the GridDB and DDoSGrid Backends (thus, the dashed line between the two Backends). In contrast to the DDoSGrid, GridDB's Frontend communicates directly with DDoSDB to authorize the user. This is because we used `nextJs` server-side rendering to communicate with the DDoSDB instead of forwarding the request to the Backend, which sends a request to the DDoSDB to check the user authorization token. It is also important to mention that each lane in Figure 3.4 illustrates the actual Docker environment in which the corresponding application with the different sub-projects runs. For example, in the GridDB lane, the Frontend, Backend, and MongoDB are all containers inside the Docker GridDB stack. For instance, the Dissector and Converters sub-projects are part of the Backend Docker container. Likewise, the miner sub-project is a part of the DDoSGrid Backend Docker container.

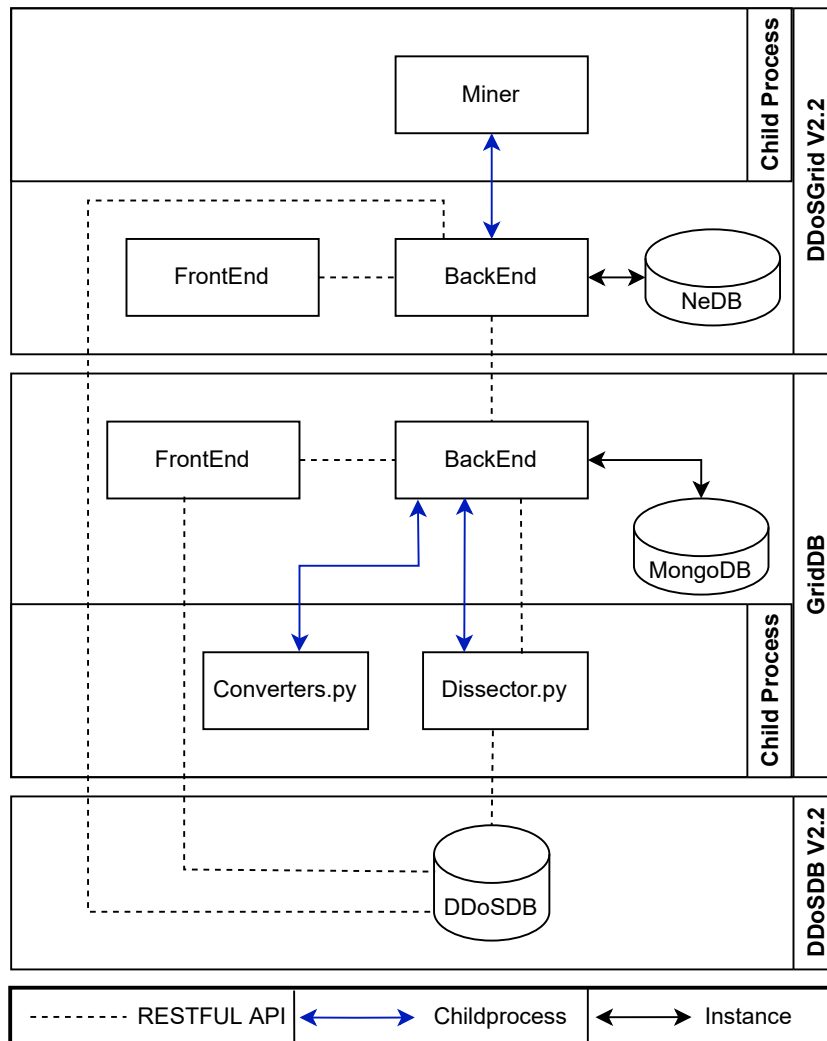


Figure 3.4: Detailed High Architecture overview of GridDB, DDoS-DB and DDoSGrid V2.2

3.3 Implementation

In this section, we describe essential implementation decisions we made during the development process of DDoSCH, DDoSGrid 2.2, and GridDB.

3.3.1 DDoSCH

The first thing we did was get an instance of the DDoSDB up and running. In the DDoSDB GitHub repository, a Linux-based installation guide is provided. We used Virtual-Box to run Ubuntu 20.04 and started with the development. The installation is straightforward; one would fetch the repository and install the needed dependencies. For DDoSDB and Dissector modules, we made no changes to the source code to get everything to work. For the converters module, we made some minor modifications to the source code to make it work. Next, we worked with python virtual environments to create a virtual environment for each module and installed the needed dependencies. An alternative would be to run everything using docker; we used this approach at the end of the development of the DDoSGrid 2.2 by modifying the provided docker files. For example, the updated version of the DDoSCH components required a higher version of python, namely 3.10; thus, we had to update the docker files accordingly.

Fortunately, the source code of the DDoSCH is very well-structured, and the modules are well refactored and separated from one another, which makes them extendable and open for further modifications. For the DDoSDB and Dissector, we implemented additional endpoints for authorization and sending a fingerprint to a specific URI.

```

1 def check_token(request, token):
2     """method to check if a token passed in the uri exists or not"""
3     token_exists(token) ? return response(status=200) : return response(
        status=400)

```

Listing 3.1: Check token function in DDoS-DB

We made one modification to the Dissector to enable sending a fingerprint to some URI. For that, we added an extra parameter `--send_fp` to the Dissector's parameters.

```

1 /*other arguemnts*/
2 parser.add_argument("--send_fp", type=Boolean, help="Optional: send the
    generated fingerprint to the URI specified in config files")
3 /*other arguemnts*/

```

Listing 3.2: `--send_fp` tag added to the Dissector

In Addition, it was essential to implement a new `args.parse()` function to extract and map the parameters, once including the new additional `--send_fp` tag.

```

1 if args.send_fp == True:
2     fingerprint.send_generated_fp(**parse_config_griddb(args.config))

```

Listing 3.3: check and parse arguments

Once the GridDB calls the responsible function for spawning the Dissector's child process with the PCAP file hash, it automatically adds the tag `send_fp` as true. It automatically modifies the configuration file by adding a new section `[griddb]`.

```

1  [ddosdb]
2  host = ddosdb:8084
3  token = f270f2cf4e5b993331dd54bba08c72072fa9c3f6a6b21e4d1ee13d40fbc012
4  protocol = http
5  [griddb]
6  token = f270f2cf4e5b993331dd54bba08c72072fa9c3f6a6b21e4d1ee13d40fbc012
7  file_md5 = 9f17044cad86bcf9857f32d5da65b7b8
8  send_fp_uri = http://griddb:8080/analysis/fingerprint/9
   f17044cad86bcf9857f32d5da65b7b8

```

Listing 3.4: configuration file example

The new section should include the URI to which the fingerprint has to be sent. This functionality replaces the `--output` tag provided by the Dissector to save the generated fingerprint to the location delivered after that tag. We wanted to enable sending the fingerprint using `Http.requests` rather than allowing the child process to modify the local files.

```

1  def send_generated_fp(self, token: str, file_md5: str, send_fp_uri: str
2  ):
3      """
4      send fingerprint to specified uri from the config file
5      """
6      LOGGER.info(f"Sending fingerprint to: {send_fp_uri}")
7      //generate json of the fingerprint
8      generate_json_fp(self)
9      //check if token exists
10     if token:
11         add_token_to_request_headers()
12     //send the request
13     try:
14         r = requests.post(send_fp_uri, headers=headers, data=fp_json,
15                             verify=False)
16     except requests.exceptions as e:
17         LOGGER.info(f"Fingerprint NOT sent to {send_fp_uri}")
18         LOGGER.debug(e)

```

Listing 3.5: sending fingerprint function

In contrast to the DDoSDB and Dissector, no Docker files were provided for the Converters subproject; however, implementing such a Docker file is straightforward since it only requires Python V10, and the project's dependencies are provided under `requirements.txt`.

```

1  # select the required python image
2  FROM python:3.8-slim
3  # Install pip requirements
4  COPY requirements.txt .
5  RUN python -m pip install -r requirements.txt
6  # change to the working directory
7  WORKDIR /app/converters
8  COPY . /app/converters
9  # declare the command to start the project

```

```
10 CMD ["python", "simple_iptables_converter.py"]
```

Listing 3.6: Docker file for converters

3.3.2 DDoSGrid V2.2

Since the OAuth2 is no longer supported, we implemented two new restful endpoints. The user now first enters an authentication token on the landing page of the Front-End application. Table 3.1 shows to which endpoint the authentication token is passed. The information flow begins with the front-end input, that is passed to the API endpoint.

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
/auth/token/:token	GET	token<String>	Path	200	Error: reason<String>	Check if token exists

Table 3.1: Specification of the REST Interface: API

Then the API sends a GET request to the DDoSDB Django server, as stated in table 3.2. The DDoSDB server checks if the authentication token is available and returns a 200 status code if it is the case.

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
check-token/:token	GET	token<String>	Path	200	Error: reason<String>	Check if token exists

Table 3.2: Specification of the REST Interface: DDoSDB

In summary, the new authentication process begins with the user entering his/her authentication token, which is sent to the API application and finally to the DDoSDB (for the response the same flow but the other way around); This decreases the complexity of the authentication process a lot. Moreover, We should ensure that this simplification does not come at security costs. The generated authentication token is hard to replicate. It gives the user exclusive access to upload a fingerprint to the DDoSDB (of the curse only if the user has the corresponding permissions related to the token). In addition, both systems do not collect sensitive data or users' data. Again, with the authentication token, the user can not log into DDoSDB.

After integrating the DDoSCH components into the DDoSGrid system, we updated the docker files accordingly. The Docker-Compose file includes only two services: the Front-end and the Backend. Also, we updated the network to enable communication with the GridDB system. The Docker-Compose file for the DDoSGrid V2.2 system looks as follows:

```
1 version: "3"
2 services:
3   ddosgridfrontend:
4     ports:
5     - "3001:8081"
6     build: "./frontend"
7     volumes:
8     - ./usr/src/app
9     environment:
```

```

10     - CHOKIDAR_USEPOLLING=true
11   ddosgridapi:
12     ports:
13     - "3000:3000"
14     build: "."
15     volumes:
16     - ./usr/src/app
17     environment:
18     - PORT=3000
19     - DDOSDB_CHECK_TOKEN=http://ddosdb:8084/check-token/
20     - DDOSDB_HOST=http://ddosdb:8084/
21     - GRIDDB_HOST=http://griddb:8080/
22     #make sure no / at the end of the
23     - CLIENT_APP_ORIGIN= http://localhost:3001
24
25   volumes:
26     userdata:
27
28   networks:
29     default:
30     external:
31     name: griddb

```

Listing 3.7: Docker-Compose DDoSGrid V2.2

Figure 3.5 illustrates the needed stacks in order to run the DDoSGrid.

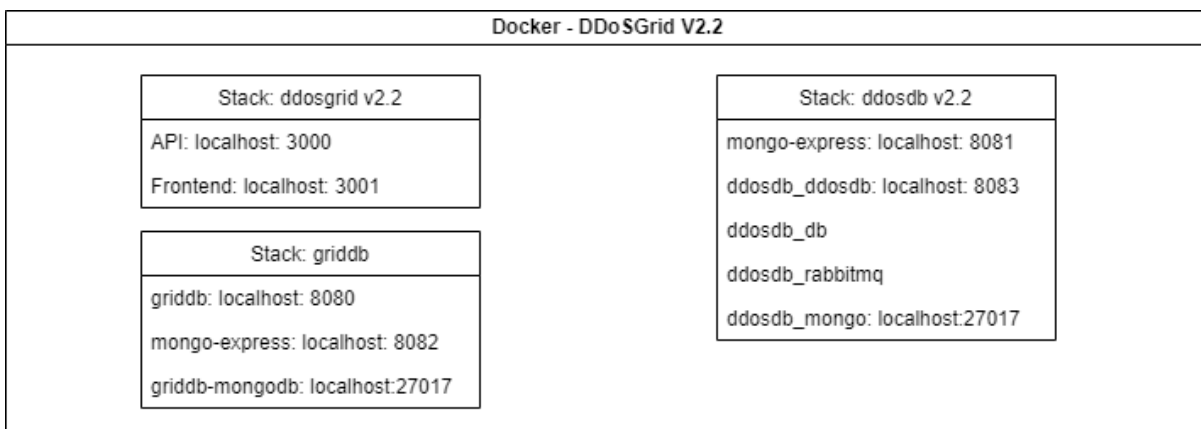


Figure 3.5: Docker overview of the DDoSGrid V2.2 stacks

The API image initially contained the API source code and the subprojects: **Miner**, **Dissector**, and the **Converters**. For that, we need **Node V10** and **Python V3.10**. This was not an issue while we developed the system locally on our virtual machine. Instead, the problem arose as we were dockerizing the application to provide a portable one. We could not find a Docker image that includes **Python V3.10** and **Node V10**. Therefore, after trying various solutions (also mentioned in the evaluation chapter 4), we decided to use the GridDB as a middleware to run the Dissector and the Converters. Furthermore, we do not need any files in return from the Dissector since it is responsible for dissecting and uploading the generated fingerprint to the DDoSDB. For that sake, we implemented new endpoints for running the Converters and Dissector. The Information flow changed

again to go through the GridDB system. First, a user uploads a network flow data file into the DDoSGrid; the file is uploaded and saved to the local files. Then, if the user checks the checkbox for exporting to the DDoSDB, we make two requests to the GridDB to run the Dissector and the Converters. GridDB uses a Docker image which includes Python 3.10 installed. The first request is made to `http://GridDB:8080/analysis/dissector`; we send the PCAP file in the request's data and the token in the headers. The Token and the PCAP file are the only two needed parameters to run the Dissector. Once GridDB receives the request, we save the PCAP file in a different directory for DDoSGrid (`data/public/DDoSGrid`) and run the Dissector; by this, we introduce redundancy to the project because the same PCAP file is saved in two places. In the first place, such redundancy is unavoidable; the Dissector expects the path of the PCAP file as a parameter. One possibility to decrease redundancy is to delete the PCAP file upon successfully exporting the fingerprint to the DDoSDB. Once the Dissector is finished, we send a response back to the DDoSGrid. A similar procedure is used for the Converters (post request to `http://GridDB:8080/analysis/converters`); however, we also return the generated IP-Tables to the DDoSGrid; this can be made available for the users to install.

3.3.3 GridDB

We started the development process by initiating the needed application. First, we initiated a `next.js` web application, and second, the `node express` server. Then, we created a GitHub repository for each application and pushed the code there. Last, we integrated DDoSCH components (`Dissector.py`, `Convertes.py`) into the backend and created the needed docker files for each of the three components (Frontend, Backend, DDoSDB). In order to avoid version confusion, before starting with the development, we tried various ways to create the docker images. First, we searched for a way to have both Python V3.10 and Node V16 on the same docker image to match the subprojects (Converters and Dissector) versions and the used node version for the backend. One possibility is to start with a Node image and then install Python, *e.g.*, using the following commands:

```

1  command FROM node:16-alpine
2  #COPY and install backend code
3  RUN DEBIAN_FRONTEND=noninteractive apt-get install -y python3
4  #COPY and install subprojects files

```

Listing 3.8: Docker Commands GridDB

We need Python in order to spawn the subprojects as child processes. The problem we faced with the multi-stage builds during the DDoSGrid development phase was that we created the executable files based on the right Python image. However, as we want to run them, the available python image is unsuitable; thus, they would fail to run. Alternatively, we searched for a Node image with the required python version already installed; this decreased the build time of the containers dramatically. Building a Docker container might take some time; in our case, building time sometimes exceeds 200 seconds. Figure 3.6 illustrates the four Docker stacks to run the application.

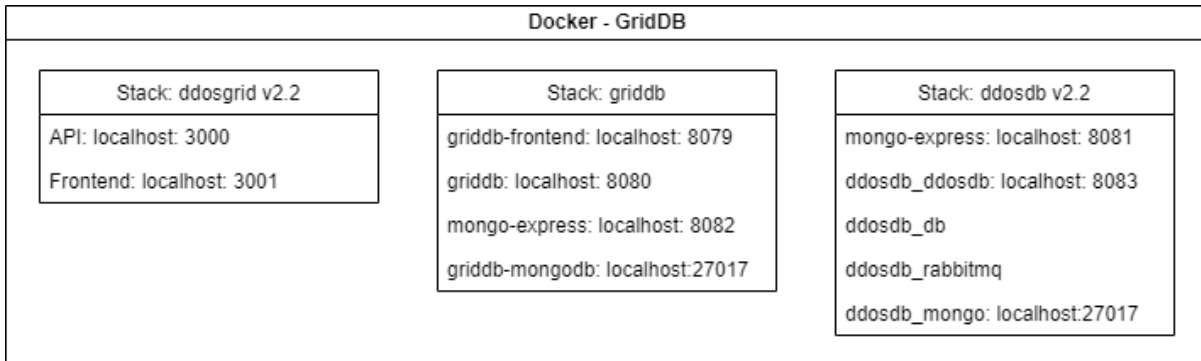


Figure 3.6: Docker overview of the GridDB components

Initially, the DDoSGrid V2.2 stack would not have been needed to run the application. However, since the DDoSGrid is now dependent on the GridDB for running the Dissector and Converters, we also allowed the GridDB to communicate with the DDoSGrid system to utilize the miner subproject. The miner subproject requires Node V10 to run; the GridDB system uses a later version not compatible with the needed one for the miner. Like the GridDB running the Converters and Dissector inside its container and communicate the results to the DDoSGrid, the miner executes inside the DDoSGrid container and communicates its results back to the GridDB using RESTful API. The miner generates visualization metrics in JSON files and saves them locally. We can upload the files into an HTTP request using the `Form-data` module and send them back to GridDB. Since we are still in the development phase, we had to pay attention to the ports exposed by each application. For the development, we used the Docker-Desktop application (utilizing `Portainer` extension) to manage and control containers and Docker images. We developed the system on a Windows machine, on which a Linux virtual machine for docker is installed. Sometimes, we ran into RAM problems because the Virtual machine used more RAM than needed; a good solution is to shut down the virtual machine and re-run it again, *e.g.*, using the command `wsl --shutdown`. In addition, we had to pay attention to the network on which the containers were running. As illustrated in the figure 3.6 above, we use three stacks, each running on a different network assigned by Docker. Therefore, there is a probability that the different containers are not on the same network and thus can not communicate. For that, we created a User-defined bridge network called `griddb` and shared the network among the different stacks, so all containers were able to communicate.

```

1 #create a network called griddb, e.g. using CLI:
2 docker network create griddb
3 #define the network in the different Docker-compose files:
4 networks:
5   default:
6     external:
7       name: griddb

```

Listing 3.9: Docker Commands GridDB - Networking

Usually, after executing a Docker-Compose file for the first time, Docker, by default, auto-generates a network named after the service, *e.g.*, `<service_name>_default`; we can see this in Figure 3.7. Similarly, after executing the following command:

```
1 docker network create griddb
```

and specifying the network name as `griddb`, a network with the name `griddb` will be generated and added to the Docker Network list. The generated network is highlighted in yellow in Figure 3.7. We can inspect available networks inside the Docker environment by typing the command: `docker network ls`

```
PS C:\Users\karim\GridDB> docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f07abf99bde1       bridge              bridge              local
968a36c89313       converters_default  bridge              local
779007858fb8       ddos_dissector_default bridge              local
c987dfabc2e3       ddosdb_default     bridge              local
84c1e104738b       docker_default     bridge              local
5db1ab10944f       griddb              bridge              local
cc8887bfa6f0       host                host                local
cfd30926e5ac       none                null                local
52325ece7107       portainer_portainer-docker-extension-desktop-extension_default bridge              local
```

Figure 3.7: Docker Network list

After assuring that the Docker files work well, we started with the development for the Backend. We implemented the following endpoints following a similar procedure of the DDoSGrid: user first uploads the network flow file to `http://griddb/analysis/upload`, and upon successful uploading, the `analyze` function is called. In the following table, we summarized the specification of the REST interface:

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
<code>/analysis</code>	GET	-	-	200	List of all Analysis	Get all analysis assigned to some token
<code>/analysis</code>	POST	Analysis Schema <JSON>	Body	201	Description: fail or success	Create a new analysis
<code>/analysis/upload</code>	POST	PCAP File <.pcap>	Files	200	Description: fail or success	Upload and save a PCAP file
<code>/analysis/:id/analyse</code>	POST	Analysis id <String>	Path	200	Description: fail or success	Analyze the PCAP file
<code>/analysis/:id</code>	DELETE	Analysis id <String>	Path	200	Description: fail or success	Delete an analysis Document by id
<code>/analysis/fingerprint/:id</code>	POST	Analysis id <String>	Path	200	Description: fail or success	Save fingerprint in the Directory matching the id
<code>/analysis/analysis-files</code>	POST	Analysis Files <JSON>	Files	200	Description: fail or success	Save analysis files from the miner
<code>/analysis/dissector</code>	POST	PCAP File <.pcap>	Files	200	Description: fail or success	Run the Dissector subproject
<code>/analysis/converters</code>	POST	Fingerprint File <JSON>	Files	200	Description: fail or success	Run the converters subprocess

Table 3.3: Specification of the REST Interface: GridDB

Analyse function: the function is supposed to be called after the PCAP file is uploaded to the system. Each file has a unique hash key; we use that key to save the PCAP file in

the directory for the user. For example, in the front end, the user uploads the PCAP file using a form. So, first, we call `/upload` endpoint, then call the `/analysis/:id/analyse` endpoint.

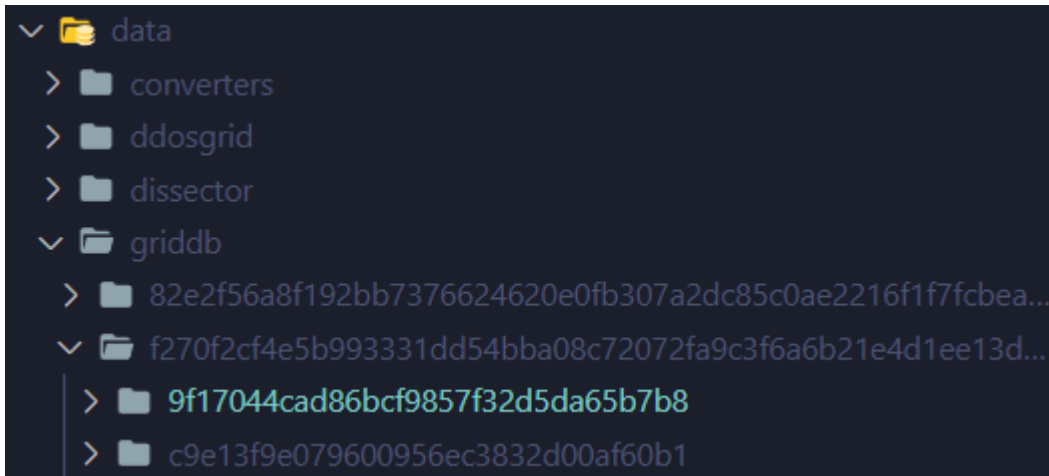


Figure 3.8: File structure of the Data folder - GridDB

The data folder contains four other subfolders. For each project that interacts with GridDB, a subfolder is automatically created to keep every system file separated from the other. *I.e.*, files uploaded using the GridDB frontend are saved in the `griddb` folder. For each user, we create a folder according to his/her authorization token. Then, according to the PCAP unique hash, we create another folder and save the actual PCAP file. Saving the file according to its place is done after the `/upload` endpoint is called.

```

1  const uploadToClient = async (event) => {
2    // first extract name, pcapFile, description from the event
3    const body = extract_and_append_data(event, upload_token)
4    try {
5      // send the request to the backend
6      const response = await fetch(
7        process.env.NEXT_PUBLIC_BACKEND_API + "analysis/upload",
8        {
9          body: body,
10         method: "POST",
11       }
12     );
13     // extract and check the status
14     const { status } = response;
15     if (status === 200) {
16       const re = await response.json();
17       // handle upload success => notify the user
18       // start analysis => send a request to the /analysis/:id={re.
19       id}/analyse endpoint
20       startAnalysis(re.id, 2000);
21     } else { // handle if status !== 200}
22     } catch (err) {
23       // handle request failure}}
24   };

```

Listing 3.10: Handle upload PCAP file - GridDB FrontEnd

The analyze function is mainly responsible for spawning all three subprojects and saving the results to the database. The flow inside the analyze function is as follows: The backend receives a request where the id is a parameter path; again, the id is a unique hash of the PCAP file, which was initially communicated and sent in the response text of the `/upload` endpoint. The function makes two checks:

1. is the uploader token present in the request headers
2. is there a folder with the id passed in the path parameter

If both conditions are actual, we can start with the analysis. Then, using try-catch blocks, we run each subproject and update the status entry in the database immediately. For instance, this is an example of running the Dissector subproject:

```

1  try {
2    analysis.dissect_status = "in-progress";
3    analysis.export_status = "in-progress";
4    analysis.filter_status = "in-progress";
5    analysis.miner_status = "in-progress";
6    await analysis.save();
7
8    var startTime = new Date();
9    const dissector_result = await pcapDissector.dissectAndUpload(
10     projectPathPCAP,
11     uploader_token,
12     analysis.file_md5
13   );
14   var endTime = new Date();
15   analysis.dissector_duration = (endTime - startTime) / 1000;
16   analysis.ddosdb_key = dissector_result.ddosdb_key;
17   analysis.dissect_status = "done";
18   analysis.export_status = dissector_result.code === 200 ? "done" : "
failed";
19   var projectPathFP = path.resolve(projectPath, `${id}.json`);
20   await analysis.save();
21 } catch (e) {
22   analysis.dissect_status = "failed";
23   await saveAnalysisToTheDB(analysis);
24   console.warn("Error occurred while running the Dissector", e);
25 }

```

Listing 3.11: Try-catch block for the Dissector subproject

First, we update all entries of the subprojects status to in progress and save the changes in the database. Then, we start a timer to measure the time of the subproject to finish. After, we call the actual function for spawning the child process and save its results. The function returns a promise; if rejected, it means the child process failed and the returned code 400; if resolved, we return code 200. The function follows an identical procedure for running the Converters subproject; we save the generated Ip-Tables file to the same directory. However, a difference in the procedure is the case for the Miner since we run it by sending a post request to the DDoSGrid backend; the Miner runs in a different container. Thus the results are communicated using RESTFUL API. In the following Code block, we illustrate how the results of the Miner subproject are received.

```

1  try {
2    analysis.miner_status = "in-progress";
3    await saveAnalysisToTheDB(analysis);
4    var formData = new FormData();
5    formData.append("pcapFile", fs.createReadStream(projectPathPCAP));
6
7    var url = process.env.DDOSGRID_API_HOST + "analysis/miner";
8
9    var response = await axios({
10     method: "post",
11     url: url,
12     data: formData,
13     maxContentLength: Infinity,
14     maxBodyLength: Infinity,
15     headers: {
16       "Content-Type":
17         "multipart/form-data;boundary=" + formData.getBoundary(),
18       "uploader_token": req.headers.uploader_token,
19       "id": id,
20     },
21   });
22   //console.log(response);
23   analysis.analysis_duration = response.data.analysisDurationInSeconds
24   ;
25   analysis.analysisFiles = response.data.analysisFiles;
26   analysis.metrics = response.data.metrics;
27   analysis.miner_status = "done";
28   await saveAnalysisToTheDB(analysis);
29   await requestAnalysisFiles(id, uploader_token);
30 } catch (error) {
31   analysis.miner_status = "failed";
32   await saveAnalysisToTheDB(analysis);
33   console.warn("Miner failed!", error);
34 }

```

Listing 3.12: Try-catch block for the Miner subproject

In order to get the results from the miner, we must make sure that an instance of DDoS-Grid V2.2 is up and running. Therefore, we implemented an additional endpoint in the DDoSGrid V2.2 accessible under `http://ddosgrid/analysis/miner`; the Backend runs the miner as a child process and saves the generated files locally. These files can be obtained by requesting them in another POST request to the DDoSGrid V2.2 under `http://ddosgrid/analysis/analysis-files`. Finally, as the miner successfully finishes analyzing the uploaded PCAP file, we extract three things from the generated results: cleaned results that include only the required attributes for the visualization, miner subproject duration to finish analyzing, and a summary of the key metrics. Those three objects are loaded and stringified in JSON format to be able to load them in the response data. First, we write the Status Code of the header as 200 and specify the returned data content type as JSON. Additionally, we include the uploader token in the response header; the uploader token is needed for the GridDB since the returned data will be saved in our database, and for that, we need to know which user has sent that request.

```

1  res.writeHead(200, {
2    "Content-Type": "application/json",

```

```

3     "uploader_token": req.headers.uploader_token ,
4   });
5   var json = JSON.stringify({
6     analysisFiles: cleanedResults ,
7     metrics: metrics ,
8     analysisDurationInSeconds: analysisDurationInSeconds ,
9   });
10  res.end(json);

```

Listing 3.13: DDoSGrid V2.2 response of the Miner endpoint

The main POST request made by the GridDB to the DDoSGrid backend server to run the miner under `http://ddosgrid/analysis/miner` returns for us three objects:

1. Analysis files: an array that contains supported diagrams and the file name of the JSON file to generate the diagrams. One element of the array can look as follows:

```

analysisFiles: [
  {
    attackCategory: 'Link Layer',
    analysisName: 'Top 5 VLANs',
    file: 'c9e13f9e079600956ec3832d00af60b1/c9e13f9e079600956ec3832d00af60b1.pcap-top-5-vlan-domains-by-eth-traffic.json',
    supportedDiagrams: [
      'PieChart'
    ],
    datasetHash: 'c9e13f9e079600956ec3832d00af60b1'
  },
]

```

Figure 3.9: Element of the analysis files array

For example, the file value in this entry contains the file name, in which the needed data for rendering the top five VLANs exists, and the supported diagram is only a Pie Chart.

2. The duration in seconds of the miner subprocess, we plot and compare the duration with the durations of the other sub-projects, Dissector and Miner. Usually, the Dissector sub-project takes the most time to finish since it analyzes the PCAP file and uploads the generated fingerprint to the DDoSDB.
3. Metrics that include summary statistics of the attack.

These pieces of information are then saved in the database and are ready to be displayed on the front end. Using the `requestAnalysisFiles()` function, we retrieve the actual JSON files from DDoSGrid and save them in the Analysis directory. The figure 3.10 shows what an analysis folder will look like after the function has returned.

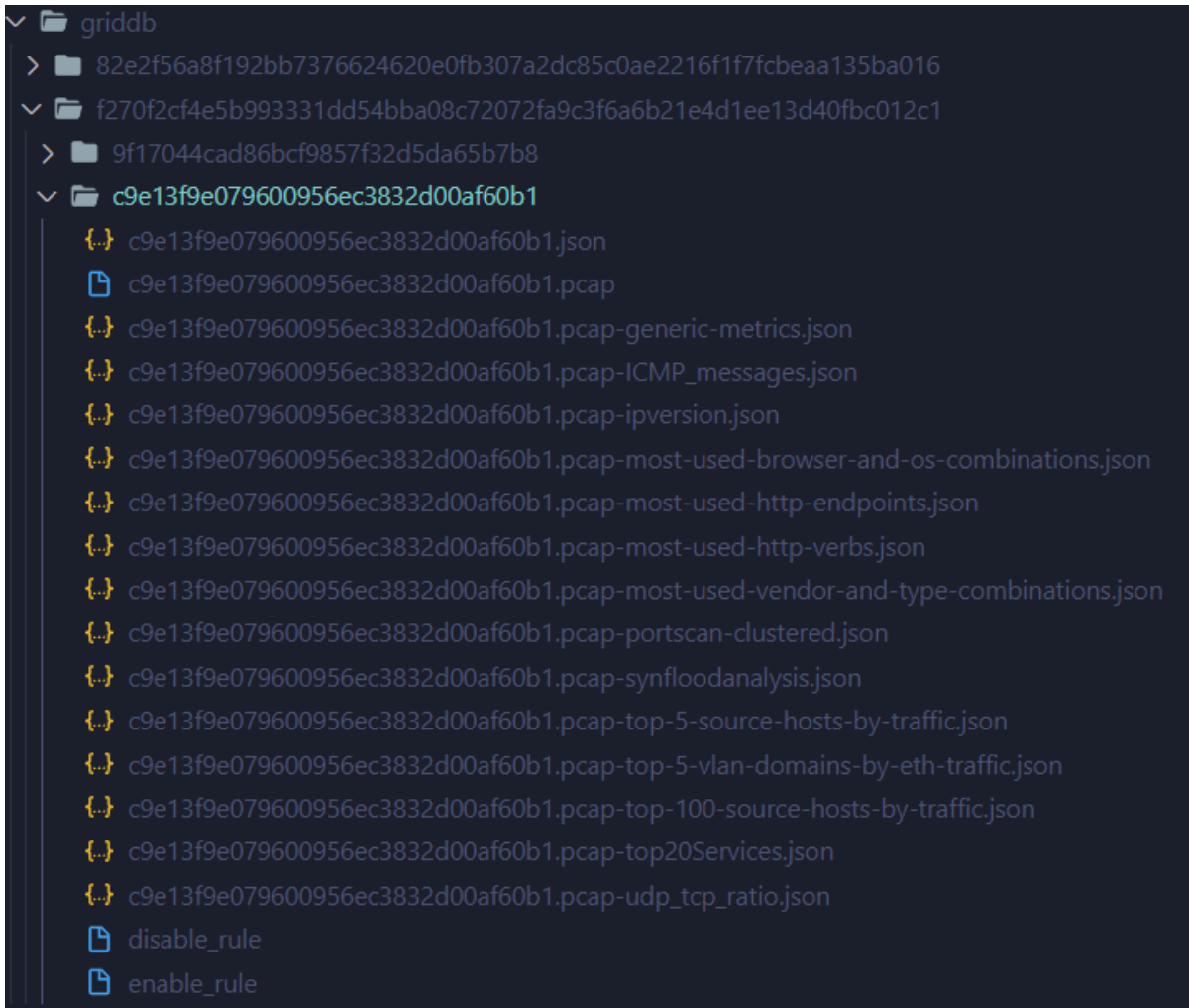


Figure 3.10: Example of an Analysis Directory - GridDB Backend

The Directory is named after the PCAP hash and includes:

- (i) Fingerprint generated using the Dissector
- (ii) PCAP file uploaded by the user
- (iii) Analysis files generated by the Miner
- (iv) Enable and Disable rules of the Ip-Tables generated by the Converters

On the Frontend side, we call the function `start Analysis`, which needs the PCAP file hash as a parameter. Then it sends a post request to the backend with an authorization token as a header. Finally, the status code is extracted from the response and handled according to its value. We Use the local storage to save the interval Id; the `SetInterval` function is responsible for fetching all the analysis from the backend every five seconds.

```
1  const startAnalysis = async (file_md5, timeOutForToast) => {
2    setDoneAnalyzing(false);
```

```

3     try {
4         const id = setInterval(() => {
5             getAllAnalysisWhileAnalyzing(token);
6         }, 5000);
7
8         localStorage.setItem("id", id);
9
10        const response = await fetch(
11            process.env.NEXT_PUBLIC_BACKEND_API + `analysis/${file_md5}/
12        analyse`,
13            {
14                headers: {
15                    uploader_token: token,
16                },
17                method: "POST",
18            }
19        );
20        const { status } = response;
21        const re = await response.text();
22        console.log(status);
23        //alert(re);
24        if (response.status === 200) {
25            setTimeout(() => {
26                toast.success("Analysis should start in a bit !", {
27                    duration: 4000,
28                    icon: <DiGoogleAnalytics />,
29                }, timeOutForToast);
30            } else {
31                toast.error("Failed to start analysis ! ");
32            }
33        } catch (err) {
34            console.warn(err);
35        }
36    };

```

Listing 3.14: Start Analysis Function GridDB - Frontend

For the data storage of the analysis information, we used MongoDB as our primary database. Then, using Docker, we pulled the latest version MongoDB image. Also, we used MongoExpress to have a UI of the database content.

```

1     services:
2         griddb-mongodb:
3             image: mongo:latest # use the latest image.
4             container_name: griddb-mongodb
5             restart: unless-stopped
6             ports:
7                 - 27017:27017
8         griddb-mongo-express:
9             image: mongo-express:latest # latest image
10            container_name: mongo-express
11            restart: unless-stopped
12            ports:
13                - 8083:8081
14            environment:
15                ME_CONFIG_MONGODB_SERVER: griddb-mongodb

```

```

16     ME_CONFIG_BASICAUTH_USERNAME: griddb
17     ME_CONFIG_BASICAUTH_PASSWORD: griddbgriddb

```

Listing 3.15: Docker-compose for MongoDB

In addition, we used the `Mongoose` module to create a Schema for a new analysis document. An analysis document is defined as follows:

```

1  const analysisSchema = new mongoose.Schema({
2    //ddosdb key is used to query the fingerprint in the DDoSDB
3    ddosdb_key: {type: String},
4    // to keep track if an analysis has been deleted
5    deleted: {type: Boolean, default: false,},
6    // to keep track when was the analysis created
7    created: {type: Date, default: new Date(),},
8    // does the system finished exporting the fingerprint to the DDoSDB?
9    export_status: {type: String, default: "planned",},
10   // did the Dissector finish?
11   dissect_status: {type: String,default: "planned",},
12   // did the converters finish generating the filter rules?
13   filter_status: {type: String,default: "planned",},
14   // to keep track of the uploader token
15   uploader_token: {type: String,},
16   // description of the analysis
17   description: {type: String,},
18   // name of the analysis
19   name: {type: String,},
20   // PCAP file size in MB
21   file_size: {type: Number,},
22   // how much it took the miner to finish?
23   analysis_duration: {type: Number,},
24   // how much time it took the Dissector to finish?
25   dissector_duration: {type: Number,},
26   // how much time it took the converters to finish?
27   converters_duration: {type: Number,},
28   // PCAP file hash
29   file_md5: {type: String, },
30   // did the miner subproject finish?
31   miner_status: {type: String, default: "planned",},
32   // the location of the analysis files generated by the miner
33   analysisFiles: {type: Array,},
34   // key metrics generated by the miner
35   metrics: {type: Map, of: String,},});

```

Listing 3.16: Anylsis Schema

Using this approach allows us to be flexible in adding or modifying the current characteristics of an analysis document.

Chapter 4

Evaluation

In this chapter, we discuss and indicate the main contributions of this work. We first show our attempts to increase the system's usability. Then we show how we succeeded in providing a portable version of each used component in this thesis. Then, we describe how we achieved delivering a fully automated system to handle network flow data of PCAP type. And then, we discuss how the new provided system is more scalable and can handle an additional amount of data. In the last section of this chapter, the discussion section, we present and discuss the concrete contribution of this work using a table of contributions and compare it to the previous DDoSGrid system.

4.1 Usability

We made the DDoSCH components more usable by introducing a friendly UI to run the modules. We achieved this by implementing the Frontend using `Next.js`. Also, we made it possible for people who do not have a DDoSDB authorization token to use the different components using the provided UI. Thus, we implemented two pages for the Dissector and Converters sub-projects besides the upload page, which serves as the main page and embodies the entry point for the automated process. Hence, the Dissector and Converters can be used without entering an authorization token; however, if the user clicks on the Upload to DDoSDB Icon, the system asks for the authorization token. The users are greeted on the landing page, and a brief explanation of the project objective is shown. Figures 4.1 and 4.2 show the content of the landing page (in the development environment accessible at `http://localhost:8079/`). The navigation bar at the top of the home page shows the different DDoSCH components the user can use. At the bottom half of the page, we include the components again with a brief description of the usage.

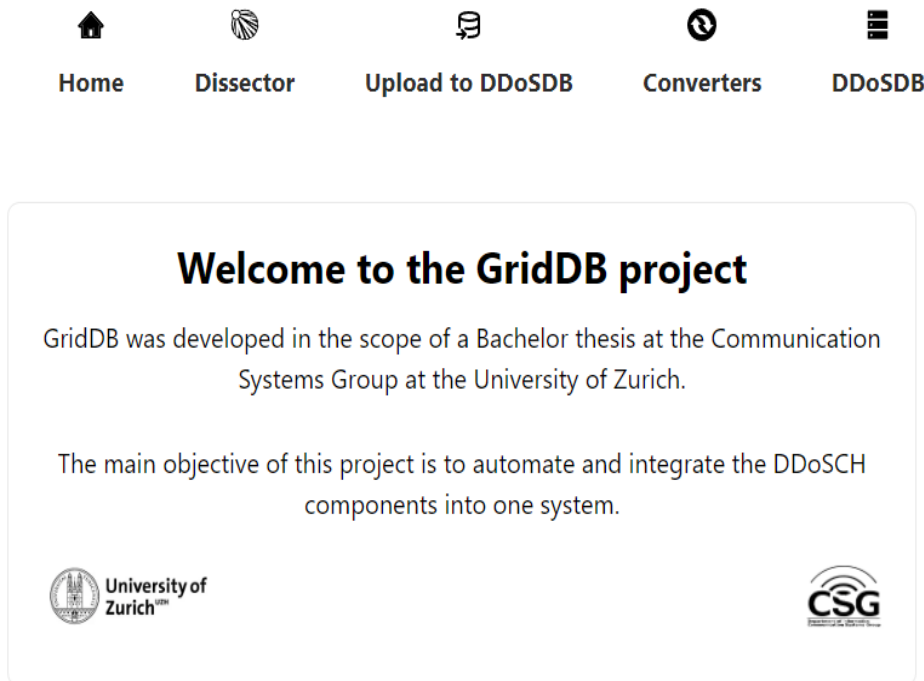


Figure 4.1: Landing Page - GridDB Frontend

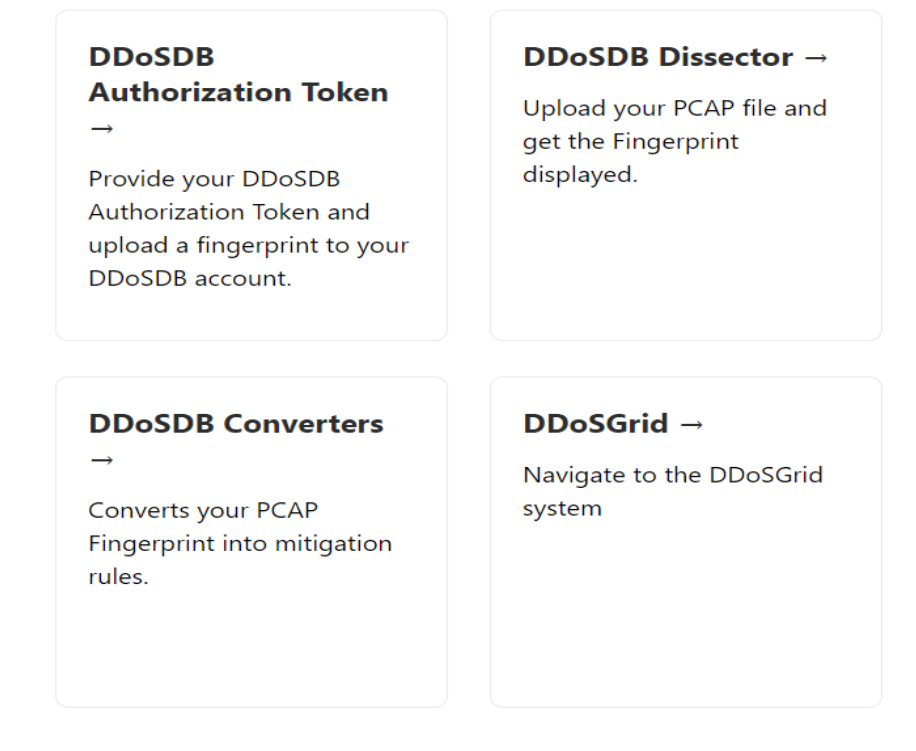
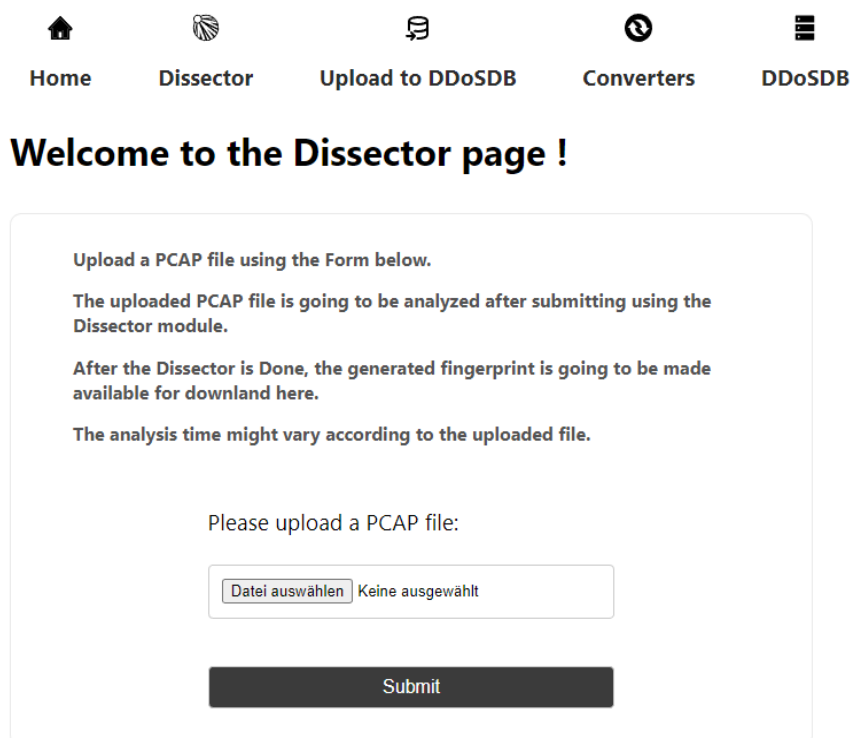


Figure 4.2: Bottom of the Landing Page - GridDB Frontend

Users can now navigate to the desired page. In addition, we introduced two additional pages for users who do not have a DDoSDB authorization token, making it possible to use the DDoSCH components without requiring the CLI to run the projects. For example, figure 4.3 shows how a user can use the Dissector subproject without logging in.



Home Dissector Upload to DDoSDB Converters DDoSDB

Welcome to the Dissector page !

Upload a PCAP file using the Form below.

The uploaded PCAP file is going to be analyzed after submitting using the Dissector module.

After the Dissector is Done, the generated fingerprint is going to be made available for download here.

The analysis time might vary according to the uploaded file.

Please upload a PCAP file:

Keine ausgewählt

Figure 4.3: Dissector Subproject - GridDB Frontend

Once the Dissector sub-project is done, the generated Fingerprint is transmitted to the Frontend and is ready to be downloaded as a JSON file. Figure 4.4 shows the results after the Backend has communicated the results to the Frontend. We follow a similar procedure for the Converters subproject. First, information about the usage of the Converters is presented, then there is a form to upload the Fingerprint file. As soon as the Backend generates the Filter rules, they are communicated to the Frontend. Finally, the system writes them in a downloadable file, and the user can download the filter rules file with a mouse click. With that, we replaced the CLI interface of the two projects with a user-friendly interface. Additionally, after the Dissector is done, we present to the user the target IP address of the attack, start and end dates, duration of the attack, and the number of packets in the attack. Also, we visualize using a Bar-Chart the attack volume in MB, the total number of unique IP addresses that were used in the attack, and the number of bytes per packet. These metrics are shared among every generated fingerprint file, and thus it makes sense to plot them. However, other metrics might not be included in the Fingerprint and thus are hard to visualize in a chart. We used the `react-hot-toast` library to notify and alert the user using notifications displayed on the project's page. With that, we give the user feedback and keep him/her updated conveniently. The top

part of Figure 4.4 n shows an example of such notifications, *e.g.*, feedback is given to the user once the Dissector has successfully finished.

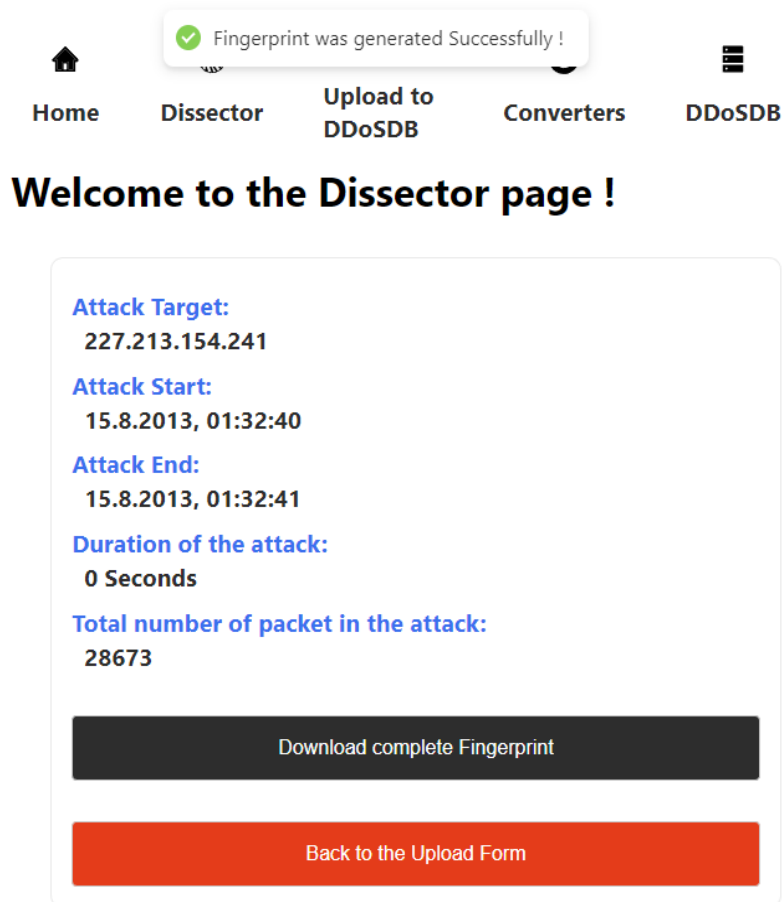


Figure 4.4: Dissector Subproject Results - GridDB Frontend

Besides the two pages to run the sub-projects differently, we implemented the fully automated process to trigger all components of the DDoSCH and the Miner sub-project from the DDoSGrid V2.2. Figure 4.5 shows the input field shown to the user once clicking on the Upload to DDoSDB icon from the navbar. After submitting the Authorization token, we use the NextJs server API to communicate with the DDoSDB server to check if the authorization token exists. Upon successful authentication, the user is redirected to the analysis page, where all previously uploaded analysis associated with that authorization token is shown. Also, a button to upload a new analysis is shown, and the upload form is displayed once the user clicks it. The users are asked to provide the PCAP file, the name, and a description of the Data set. In addition, users can find a brief explanation of the automated process in the upload form. Then, the users can upload the file and receive feedback accordingly (success or failure). Upon successfully uploading the PCAP file to the GridDB system, the analyze function is triggered, and the automated process of spawning the subprojects starts.



Welcome to the GridDB !

Please Enter your DDoSDB
Authorization Token:




 


Figure 4.5: Login Page - GridDB Frontend


Figure 4.6 shows what an analysis card looks like while the analysis is running.


DataSet Name:
Data Set number 4 

Description: Data set number 2
Created on: 25 Aug 2022, 10:10

Export status: in-progress 

Dissect status: in-progress 

Filter status: in-progress 

Miner status: in-progress 

File size: 23.842 MB




Figure 4.6: Analysis in Progress - GridDB Frontend

The uploaded analyses are shown to the user in a Grid manner. In addition, each analysis is displayed on a card with some information about it. Users can click on the analysis card to display more information. When the user uploads a new PCAP file, analyzing the PCAP file starts. Once the user has successfully logged in, feedback is given, and an additional button, "log out," is shown on the navbar. Also, feedback is given when the analysis starts and finishes.

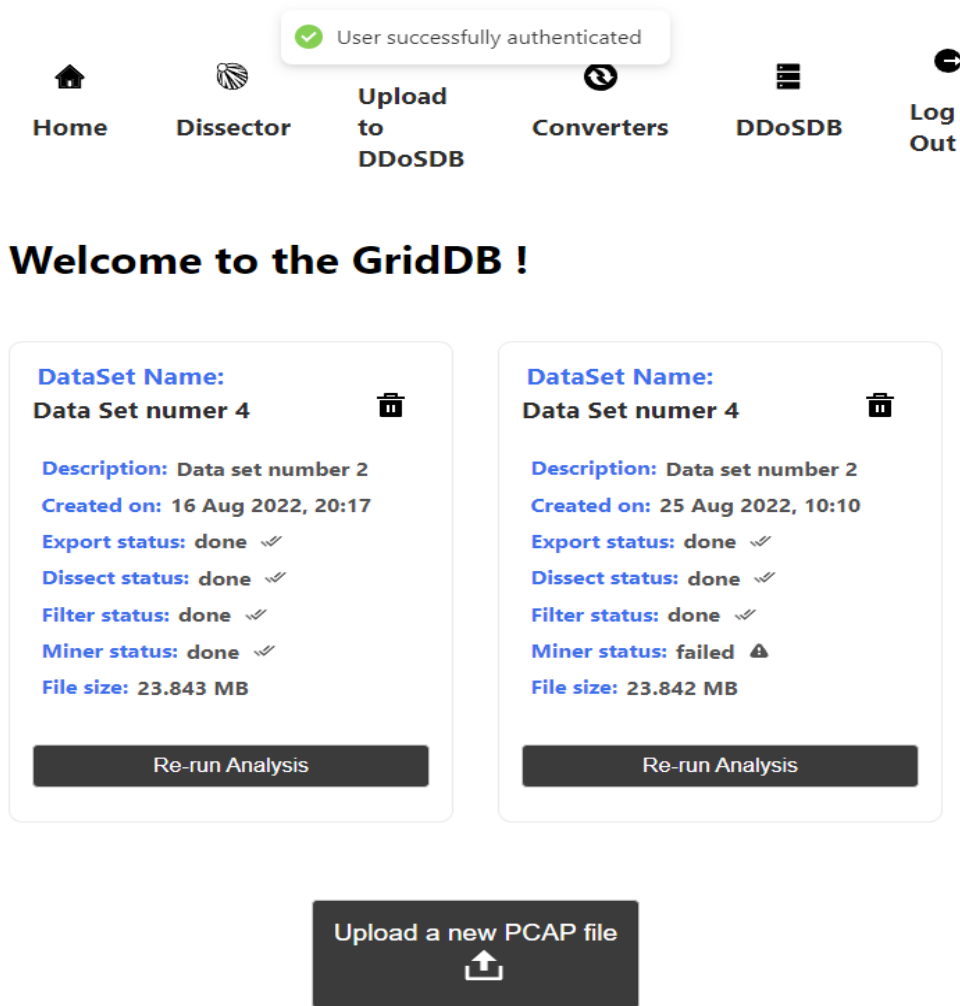


Figure 4.7: Analysis Page - GridDB Frontend

The data set on the right side indicates that the miner failed to run; this was done for illustration purposes, while for the analysis, we turned off the DDoSGrid container in Docker. Users can click on the card to further inspect the analysis. Similarly, the system displays various visualization and separated information about the sub-projects.

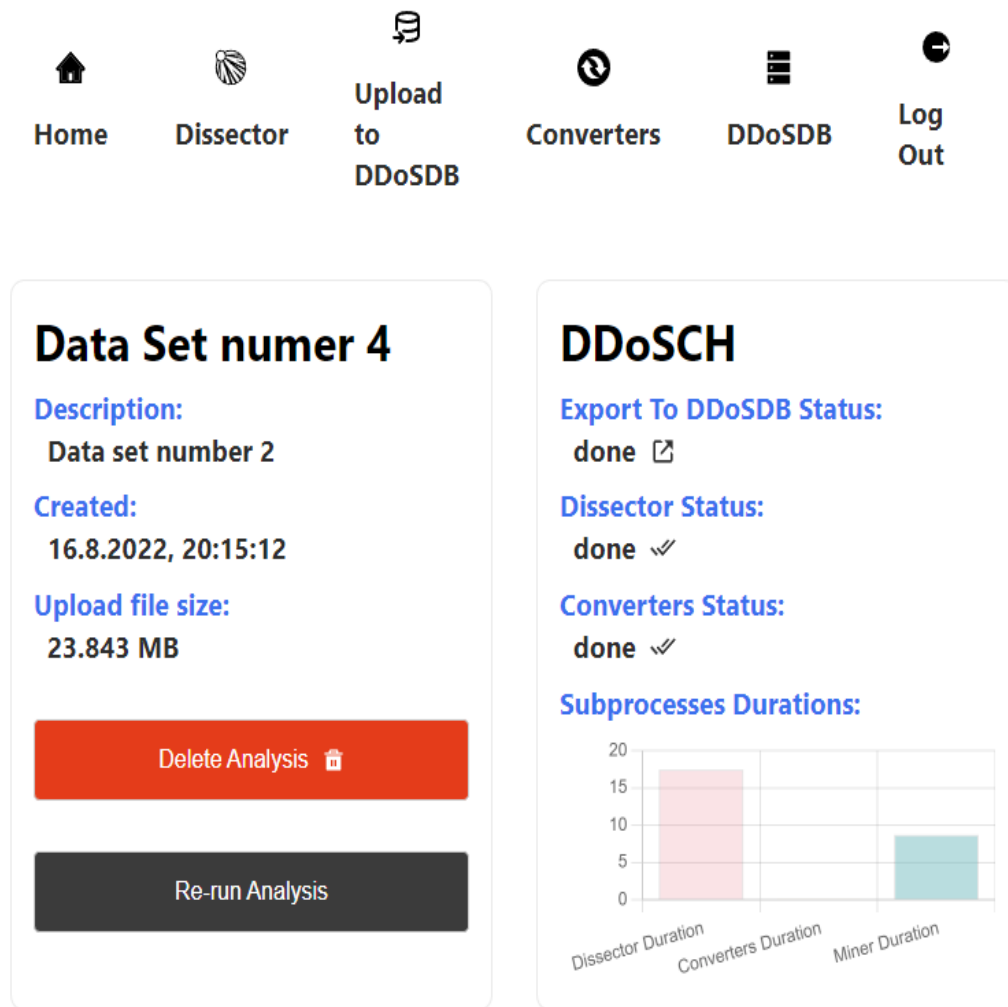


Figure 4.8: Analysis Information Page (top) - GridDB Frontend

Figures 4.8 and 4.9 show the analysis information page of a particular data set. Users can delete or re-run the analysis, *e.g.*, if the analysis failed. Also, the duration of the subprojects is plotted. Usually, the Dissector sub-project takes the most time to finish. On the DDoSCH card, we linked the analysis to the DDoSDB fingerprint page, such that users are re-directed to the fingerprint in their DDoSDB account. The fingerprint in the DDoSDB is displayed as a prettified JSON entry, where all the fingerprint information is shown. For the visualization, we used the `react-chartjs-2` library to display interactive visualization, supporting hovering and excluding or including elements of the charts. For the visualized data, we mainly relied on the generated data of the miner sub-project. Hence, the visualizations are displayed beside the DDoSGrid card.

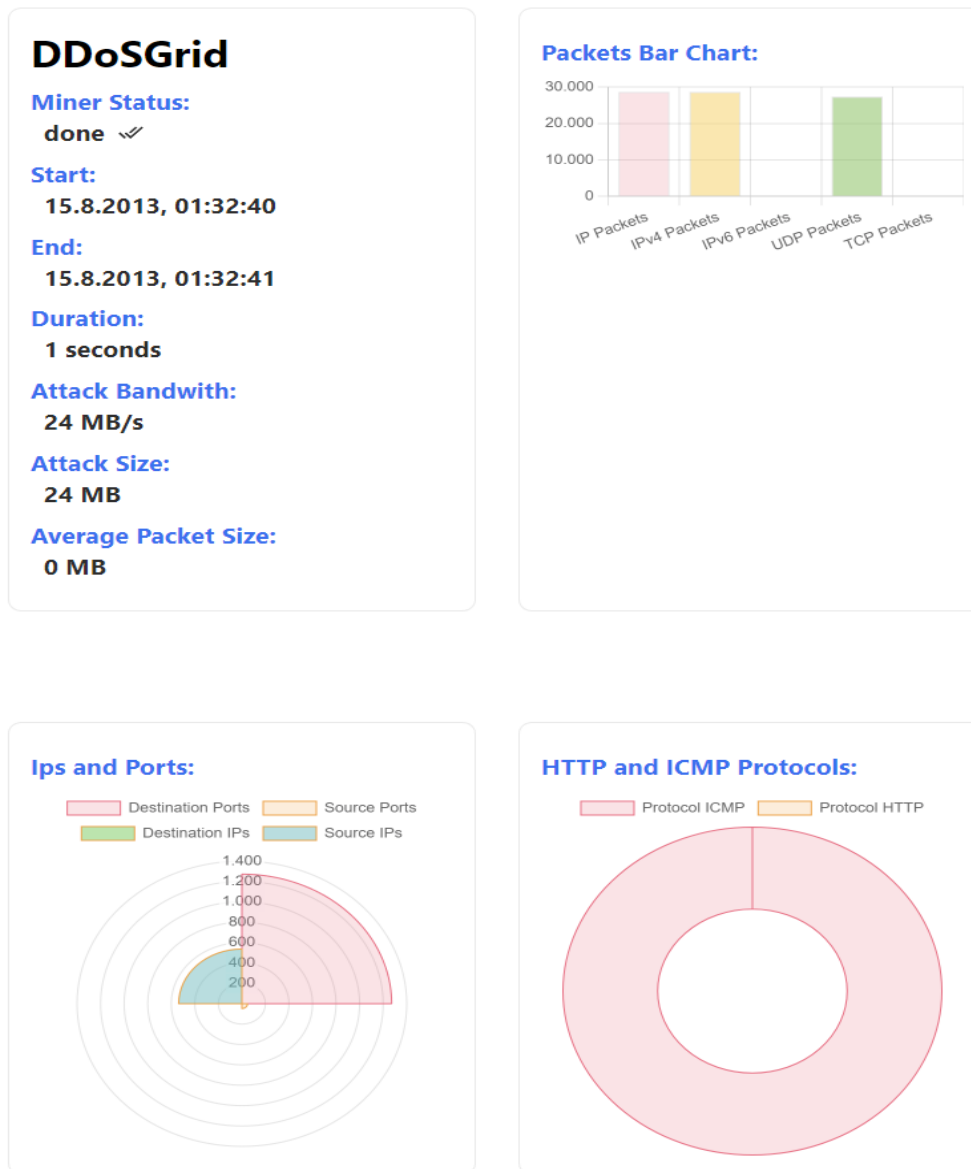


Figure 4.9: Analysis Information Page (bottom) - GridDB Frontend

4.2 Portability

The evolvement of hardware components is rapid, increasing the need to provide portable systems [43]. Furthermore, portable systems are easy to transfer between different environments [43]. At the beginning of this work, we used a virtual machine operating on Virtual-Box software as our primary system for development. In addition, we used some screen-sharing software during our weekly virtual meetings to discuss and inspect the development process. However, we faced an issue once we wanted to share a functioning copy of the files. One solution was to export the whole virtual machine and re-install it on another device using the Virtual-Box software. Another solution was to use a virtual machine in the cloud. The last and most elegant solution is to use Docker to dockerize

the applications; we created an organization on GitHub to push the code with the needed Docker-compose files. Docker is an application that helps developers to decouple source code from the used infrastructure during development; this facilitates testing, developing, and deploying code [24]. Hence, we provided a Docker file for each subproject in this thesis.

4.2.1 DDoSGrid V2.2

After integrating the new DDoSCH components into the DDoSGrid V2.2 system, we updated the corresponding Docker files. However, we faced some challenges since the Node image was bygone, namely `Node V10`. There were a lot of deprecated functionalities if we had used a recent node image. In addition, if we would use `Node V10` as our base image for the Docker file, the required Python version (`Python V3.10`) was not supported in that image. We tried to overcome this problem by using a multi-stage builds Docker container. First, we pull the latest Python image using:

```

1  # start the first stage called: builder
2  FROM Python: latest AS builder
3  # COPY and Install requirements of Dissector and Converters
   subprocesses
4  # start the last stage
5  # use Node V10 for the last stage
6  FROM node:10.22.1-buster
7  # COPY and install miner and API dependencies
8  # COPY executable files from the builder stage
9  COPY --from=builder /usr/src/app/api/ .
10 COPY --from=builder /usr/src/app/miner .

```

Listing 4.1: Docker Commands DDoSGrid V2.2

telling Docker to start a new stage called `builder`. Next, we install the dependencies of both DDoSCH subprocesses (Dissector and Converters), and then we start a new stage using `Node V10` to install the API module dependencies (since it is the last stage, there is no need to name it). In the end, we copy the files, mainly the executable files of the builder stage, to the last stage so that we have everything up and running in a lightweight container. Unfortunately, this solution did not solve our problem since Docker will use the image from the last build as its central image, which does not contain the required Python version. We tried to update and install Python on the node image, but it was impossible to use the latest and needed image. As a result, we decided to use the backend code of GridDB as our reference to run the DDoSCH components through RESTful API. Achieving this provides us with two fully portable and separate Docker containers.

The Dockerfiles of the Frontend were not changed since they work fine. Nevertheless, we changed the Dockerfile of the DDoSGrid Backend only to include the miner subproject and to install the required NPM modules of the node Express server. The previous Dockerfile included the DDoSCH subprojects (Dissector and Converters), yet we removed them from the Dockerfile since they are no longer needed. The new docker files are structured as follows:

```

1 FROM node:10.22.1-buster
2
3 RUN apt-get update; DEBIAN_FRONTEND=noninteractive apt-get install -y
   libpcap-dev tshark;
4
5 COPY miner /usr/src/app/miner
6 WORKDIR /usr/src/app/miner
7 RUN npm install
8
9 WORKDIR /usr/src/app/api
10 COPY api /usr/src/app/api/
11 RUN rm -rf node_modules/ ; npm i; mkdir -p tmp
12 EXPOSE 8080
13
14 CMD [ "npm", "start" ]

```

Listing 4.2: Final Docker Commands DDoSGrid V2.2

4.2.2 GridDB

For the GridDB, we assured before starting the development that we provide portable Docker images. Thus, we dedicated enough time for that matter. First, we created a Docker File for each component: Backend and Frontend. After that, we created a Docker-Compose file to fire up both images and create the containers. Ultimately, we integrated the DDoSCH components (Dissector and Converters) and defined the network GridDB, on which both Stacks: GridDB and DDoSDB are supposed to work to enable communication between the different containers. The GridDB Docker stack includes four containers (frontend, backend, MongoDB, and MongoExpress). We employed recent versions of the used frameworks for the GridDB system to increase the system's quality and make it extendable for future work and improvements.

During the Development phase, we ran the application only within the Docker environment; this helped us by specifying the URI of the applications that needed to communicate with each other. For example, on the Backend, by server start, we establish the connection to our MongoDB; instead of hard-coding the actual URI of the DB, we can let docker handle this for us by specifying the service name.

```

1 //connect to the mongoDB
2 console.log("connecting to MongoDB...");
3 const url = "mongodb://griddb-mongodb:27017/test";
4 mongoose.connect(url, { useNewUrlParser: true });
5 const db = mongoose.connection;
6
7 db.on("error", console.error.bind(console, "failed to connect to
   MongoDB"));
8 db.once("open", () => {
9   console.log("====connected to MongoDB====");
10 });

```

Listing 4.3: Docker URI handling example 1

Docker also came in handy for the environment variables of the Frontend application. For example, for the connection with DDoSDB to check the authorization token, instead of using localhost, we can use the service name of the DDoSDB container; this will work because, for the authorization, we used `nextJs` server-side rendering to send the check request, thus running inside the Docker environment. For other environment variables, we had to specify the actual URI of the application since they are exposed outside the docker containers, namely running on the browser, *e.g.*, DDoSDB UI.

```
1 NEXT_PUBLIC_BACKEND_API="http://localhost:8080/"
2 NEXT_PUBLIC_DDOSGRID_HOST="http://localhost:3001/ddosgrid"
3 NEXT_PUBLIC_DDOSDB_HOST="http://localhost:8084/"
4 NEXT_PUBLIC_DDOSDB_CHECK_TOKEN="http://ddosdb:8084/check-token/"
```

Listing 4.4: Docker URI handling example 2

Usually, the build time for the GridDB is not less than 100 seconds. To identify the top five commands, we plotted the duration in seconds in figure 4.10. As we can see, running `NPM install` and `NPM build` are the commands with the highest running time. Fortunately, building the Docker images is not a recurring task after everything is set correctly, and we can utilize different libraries to refresh the files upon changes, *e.g.*, we used the `Nodemon` library to re-run the backend image after changes automatically. Rebuilding the Docker containers was a problem for us while creating the Docker files for the DDoSGrid system with the version issue. Whenever we tried a different solution, we had to rebuild the files since the changes were made directly to the Dockerfile and not the Docker-compose file.

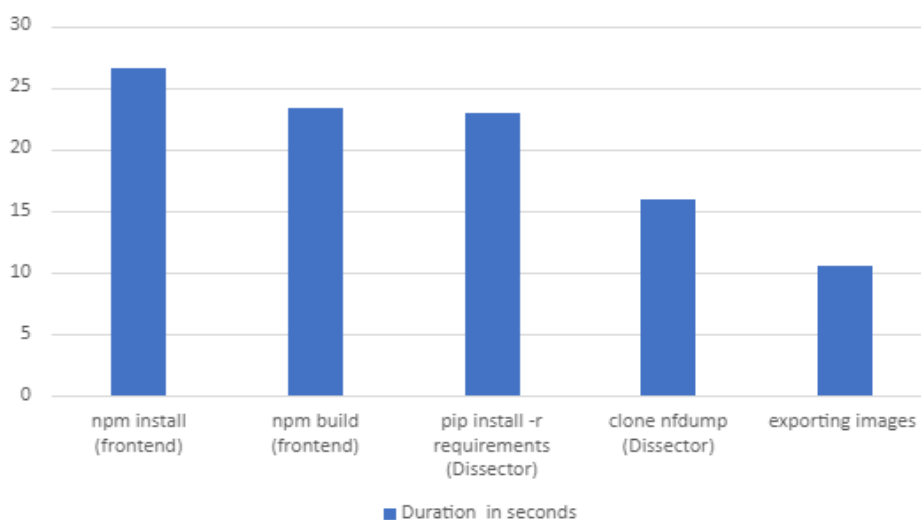


Figure 4.10: Docker Build Time Top 5 Commands - GridDB

To run the GridDB using Docker, we need four services to work together. However, if we want to utilize the miner results, *e.g.*, to display them on the GridDB Frontend, we should ensure that an instance or a docker container for DDoSGrid V2.2 is also running. Similarly, if we upload the fingerprint to the DDoSDB, we should ensure that an instance of the DDoSDB is up and running. Therefore, we used Docker-compose to fire up all

four services to run the GridDB while utilizing the converters and Dissector (without the upload possibility) sub-projects. The services are:

1. GridDB Frontend
2. GridDB Backend
3. MongoDB
4. MongoExpress

```
1 version: "3.4"
2 services:
3   # GridDB backend
4   griddb:
5     container_name: griddb
6     image: griddb
7     build:
8       context: .
9       dockerfile: ./Dockerfile
10    environment:
11      NODE_ENV: production
12      DDOSDB_HOST: http://ddosdb:8084/
13      DDOSDB_HOST_DISSECTOR: ddosdb:8084
14      FRONTEND_HOST: http://griddb-frontend:8079/
15      SEND_FP_HOST: http://griddb:8080/analysis/fingerprint/
16      DDOSGRID_API_HOST: http://ddosgridapi:3000/
17      # for CORS we use localhost:
18      DDOSGRID_API_HOST_CORS: http://localhost:3000/
19      FRONTEND_HOST_CORS: http://localhost:8079/
20    ports:
21      - 8080:8080
22    volumes:
23      - ./usr/src/app
24    links:
25      - "griddb-mongodb"
26  # Mongo Express Service
27  griddb-mongo-express:
28    # MongoExpress config
29  griddb-mongodb:
30    # MongoDB config
31  griddb-frontend:
32    build: "./frontend"
33    ports:
34      - 8079:3000
35    volumes:
36      - "../usr/src/app"
37    env_file:
38      - ./frontend/.env.local
39  networks:
40    default:
41      external:
42        name: griddb
```

Listing 4.5: Docker services for GridDB

4.2.3 DDoSDB V2.2

A Docker File for the DDoSDB image was already provided in the DDoSDB GitHub repository [22]. However, we still needed to modify the Docker files provided by the DDoSCH. For example, change the default network of the DDoSDB containers to enable communication with DDoSGrid and GridDB.

```
1 networks :
2   default :
3     external :
4       name : griddb
5
```

Listing 4.6: Docker Network for DDoSDB

We also created a Docker file (included in the chapter design 3 under implementation) for the Converters module, which was mainly used in the GridDB backend container.

4.3 Automation

Driven by the requirements, we wanted to provide a fully automated, scalable system that integrates all new DDoSCH components. We also wanted the user to upload the files in one place and at one time. With the help of the Backend functionalities, we achieved that in the following way:

The user first enters the DDoSDB authorization token provided to him/her. After that, the user is supposed to upload a network flow data file (PCAP file); we utilize the `http://griddb/analysis/upload` endpoint to upload the file. Finally, we call the `http://griddb/analysis/:id/analyze` endpoint upon successful upload to start the analysis. By starting the analysis, we mean the following:

1. The system runs the Dissector subproject to dissect the uploaded PCAP file and to generate the attack fingerprint.
2. The system sends back the generated fingerprint to the server and displays critical metrics using visualizations.
3. The system uploads the fingerprint to the DDoSDB and sends the URI of the fingerprint back to the user such that the user can view it using his/her DDoSDB account.
4. The system runs the Converters sub-project to generate mitigation rules according to the generated fingerprint by the dissector. An IP-Tables file is generated and is made available for the user to download and enable.

Figure 4.11 illustrates the automated process; it starts with the user entering the authorization token and then uploading the PCAP file. Finally, the Analyze function is triggered, and the sub-projects start.

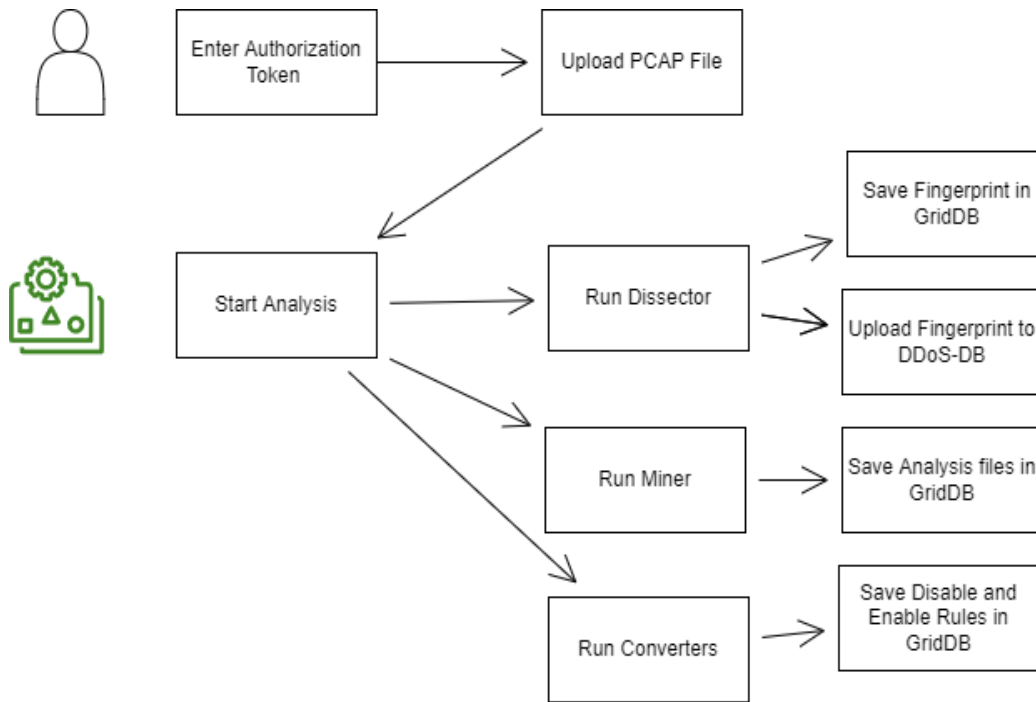


Figure 4.11: Automation Process - GridDB

4.4 Discussion

In this section, we discuss how the goals of this thesis were achieved and which were not. For that matter, we created a Table of contributions. Also, we start by briefly explain how we tried to increase the scalability of the data storage for the analysis information.

A scalable system is a system that can handle a more significant amount of data without the need to add extra resources to the system [67]. We increase the system scalability by scaling the data storage of the analysis information. The previous DDoSGrid V2 used NeDB, an in-memory database, which is, according to the Developers, the DB is no longer maintained, and it may have bugs and security issues [44]. In this regard, we decided to use a more reliable DB. Thus, we introduced MongoDB as our primary database to save the generated analysis documents. We utilize the powerful indexing of documents and the fast querying provided by MongoDB. The analysis data from the DB are mainly requested by the Frontend and displayed to the users. We used the latest MongoDB Docker image to install the DB. Furthermore, we pulled and connected MongoExpress to the DB for the visual representation of the DB content (MongoExpress is not an essential component of the GridDB system but serves as an additional feature besides the actual DB and can be turned off if not needed).

Initially, we had three main goals. The first is to integrate the newer version of the DDoSCH components into the DDoSGrid system. The other two goals are to provide a UI to employ the DDoSCH components and evaluate it according to the SUS. Due to a time limitation and in common agreement with the advisor, we opt not to realize the evaluation experiment according to the SUS. Instead, the focus was on the other

“evaluation dimensions” of the prototype. Thus, during the development, we focused more on the portability of the operating systems and on the automation process of running the DDoSCH components per click. Although we made changes to the UI compared to the DDoSGrid, *e.g.*, each analysis has its page and includes all related information and visualizations to that analysis, we were of the view that the changes in the UI are insufficient to evaluate the system according to the SUS. In contrast, in the DDoSGrid, the visualizations of further analysis are displayed on the same page, namely the `/grid`.

In Table 4.1, we point out the goals achieved by this thesis and describe them.

Goal	Description	How Achieved
Integration DDoSCH	DDoSCH provides three separate components, operated using a CLI. We aim to integrate the components into one system and replace the CLI with a user-friendly interface.	We built a Backend server to run two of the DDoSCH components (Dissector and Converters) as child processes. The third component DDoSDB is a Django server with UI, which due to implementation limits, can not be integrated within our Backend server. However, we allowed communication between our backend server and Django server using HTTP requests.
Integration DDoSGrid	The DDoSGrid V2 employed an older version of the DDoSCH components. In the meanwhile, the components have been updated and improved. Thus, we aimed to integrate the newer version of the DDoSCH components into the DDoSGrid V2 system.	With the help of the GridDB system, we run the sub-projects of the DDoSCH (Dissector and Converters) inside the GridDB docker environment and communicate the results to the DDoSGrid system. We also modified the required endpoints to allow a successful connection to the DDoSDB.
Automation	The DDoSCH components are separate from each other. Therefore, each component can be used as a one-decoupled system. We aimed to automate the usage of the three components.	We built a Frontend application to allow a user to upload a PCAP file; upon successful upload, we sent a request to the Backend to start the analysis and automate the usage of the DDoSCH components. Next, the Backend runs the Dissector, uploads the Fingerprint to the DDoSDB, and finally runs the Converters module. This automated process requires the user only to provide his/her authorization token and a PCAP file.
Portability	After implementing the different systems in this thesis, we aimed to provide a portable version for all applications. A portable version facilitates the team’s development and decouples the development environment from the actual code basis.	Using Docker, we built a Dockerfile and a Dockercompose file for all the applications we newly implemented. For the DDoSGrid and DDoSDB, we modified the provided Dockerfiles to allow successful communication with the GridDB system.

Table 4.1: Table of contributions

Further, GridDB uses the latest versions of the employed frameworks compared to the DDoSGrid. For example, the Backend of the DDoSGrid uses `Node V10` since it depends on the `libpcap` library, which requires that version. Thus, besides the changes to the UI, we increased the code quality by employing the latest versions to avoid deprecated functionalities of the dependencies.

Chapter 5

Final Considerations

5.1 Summary

Systems that encounter DDoS attacks collaboratively are more likely to protect possible victims from the same attack type. In addition, providing meaningful visualizations facilitates the objective of understanding the attack type. Such systems support the different stakeholders and decision-makers in understanding the attacking nature and help in mitigating similar attacks in the future. Systems like GridDB, DDoSGrid, and DDoSCH provide users with information about the attack characteristics and visualize critical metrics to ease the perception and understanding of the attack type.

We started this thesis by studying the nature of DDoS attacks and how they evolved. Our focus was on systems that counter DDoS attacks collaboratively. We noticed that most available works to counter DDoS attacks by sharing the network data provide reliable systems to protect possible DDoS attack victims from similar attack characteristics. However, this approach has its downside, such as hesitating crowds to share their information, fairness issues, and anonymization of the shared data. DDoSCH is an open-source project that tries to protect possible DDoS attack victims by studying network flow data of devices that have already been victims of a DDoS attack. DDoSCH consists of three separate modules. The first module is the DDoSDB, a Python Django server serving as the primary database. It is responsible for viewing, uploading, and inspecting generated fingerprints of the attack data. The other two modules are the Dissector for generating the fingerprint and the Converters to generate filter rules according to the attack fingerprint. DDoSDB employs the Django user-based model to allocate permissions to users and groups; with that, we can define who is eligible to share and view attack data. The Dissector module is also responsible for anonymizing the attack data.

Furthermore, the converters generate IP-Tables without showing the actual victim's information, resulting in a system that correspondingly counters a set of issues of similar systems that use an equivalent approach to protect against DDoS attacks. Those are anonymization of the data (achieved by the Dissector), and the DDoSDB achieves fairness by allocating permissions to users who are allowed to view the attack data. Additionally, the Converters module defines a set of rules beneficial for everyone by blocking the

network traffic for IP-Addresses that were a part of launching a DDoS attack. The main objective of this work was to integrate all DDoSCH components into one fully automated system and replace the inconvenient way of running the DDoSCH modules using the CLI with a user-friendly interface. Furthermore, another main objective of this thesis is to replace the actual DDoSCH components running in the DDoSGrid system with the new updated versions of these components.

For that sake, we followed an incremental development process. The first step was to run both applications (DDoSGrid and DDoSDB), study their architecture, and understand how they interact. After achieving this, we started by integrating the new DDoSCH components into the current DDoSGrid code. The first achievement was implementing an endpoint to authorize users and establish the connection between the two applications. Allowing the Frontend component to be notified once the user is authenticated, which enables him/her to interact with the DDoSGrid system. Then, we started by implementing the Dissector and the Converters to be spawned as child processes of the DDoSGrid. We faced only minor issues during that, for example, allocating root permission to the child process to access the system files. The critical issues appeared as we tried to dockerize the application and provide a portable version of the system. Namely, we could not find a Docker image that satisfies all sub-projects versions. In addition, building time for the Docker containers was too long, and we tried various ways to solve this problem. The objective of integrating the DDoSCH components into the DDoSGrid system exceeded the time we originally dedicated to that matter, since we spent too much time targeting the versions issue. Parallely, we started working on the new GridDB system using recent versions of the employed frameworks. To avoid facing the same issue with portability, we created and tested the Docker files before starting with the actual code. Last, we operated GridDB as a middleware to run the Dissector and Converters sub-projects. We achieved both objectives in parallel, without wasting too much time trying another solution to run the sub-projects within the DDoSGrid environment.

As per the beginning of the thesis, we intended to focus also on the UI aspects to increase the frontend code's usability. Nevertheless, our focus was more on automating the DDoSCH components and delivering a portable version of the developed systems. Thus, we did not see the need to evaluate the system usability according to the system usability score but more on providing an evaluation of how we achieved supplying a portable, automated, and scalable system.

5.2 Conclusions

DDoS attacks are a significant threat to individuals, businesses, and governments. They are increasing in complexity, volume, and duration each year. Thus, it is worth spending efforts developing systems to encounter this cyber assault. However, due to the sophistication and the enormous amount of data used in the attack, dealing alone against such threats became nearly an impossible task. Therefore, systems that handle DDoS attacks in cooperative alliances have gained more interest. In this thesis, we integrated the components of the DDoSCH into one portable system, GridDB. DDoSDB, one of the components of the DDoSCH, provides a database to share analyzed network traces, *i.e.*, fingerprints

among registered users. Thus, users can share attack data and cooperatively comprehend and protect their potential infrastructure against similar attack characteristics.

Systems employing the cooperative approach come along with high technical requirements. For example, a centralized system often has to deal with vast data due to the nature of the attack itself. Consequently, scalability issues may occur. Employing Blockchain technology solves various issues that come along with systems that utilize a cooperative approach against DDoS attacks. In addition, blockchain technology is helpful since it targets decentralization, anonymization, and privacy concerns. However, other trust and fairness issues may persist.

In conclusion, cooperatively combating DDoS attacks is a novel approach for protecting potential DDoS attack victims and understanding the attack nature and characteristics from already available attack data of other DDoS attack victims. In addition, providing meaningful visualizations facilitates the perception and understanding of the underlying data and supports decision-makers. Potential issues on those systems like Fairness, Privacy, and Scalability introduce additional requirements to the implementation and must be adequately targeted.

5.3 Future Work

First, we consider the one unreached initial goal of this thesis as a priority for the future work; namely, the goal of evaluating the GridDB Frontend according to the SUS.

Second, we consider the performance testing of the system's capabilities as the second priority for future work. Performance testing is essential to identify bottlenecks and determine the system's weaknesses and strengths.

During the development of the GridDB system, our primary focus went more on automation, integration, and portability. In addition, we increased the Usability and discoverability of the employed UI by providing a brief explanation of usage and giving proper feedback to the users upon actions.

Last, we point out the rest of possibilities for future work under extensions and improvements.

Extensions and Improvements:

- Provide more visualizations according to the attack type: currently, the visualizations illustrate data shared among every generated fingerprint apart from the attack type. For example, attacks on a specific layer include more sophisticated data; this data can also be visualized to the users.
- Integrate the generated visualization of the miner project into the GridDB: Currently, we only visualize part of the summary data generated by the Miner project. However, the Miner provides more advanced data, which are also saved in the GridDB system as JSON files in the attack folder. Using these data, we can include more visualization.

- Let the GridDB activate the mitigation rules generated by the Converters with a button click: Currently, the GridDB allows the user to download the generated IP-Tables file; the file is in text format, which includes shell commands to enable the rules. A possible extension is to let the GridDB system enable the rules automatically on the actual device.
- Responsiveness of the UI: the system was developed on a screen resolution of 2560x1440 and thus optimized for this size. The system is still usable but might not be optimal on a smaller screen, *e.g.*, on mobile devices. Thus, further improvement in UI design responsiveness can be made.
- Deploy the system: currently, the implemented systems in this work function in a docker environment, another priority for future work is to deploy the system to be available online.

Bibliography

- [1] VentureBeat: Kyle Alspach. *Israeli government websites temporarily knocked offline by ‘massive’ cyber-attack*. 2022. URL: <https://portswigger.net/daily-swig/israeli-government-websites-temporarily-knocked-offline-by-massive-cyber-attack> (visited on Mar. 17, 2022).
- [2] VentureBeat: Kyle Alspach. *Ukraine: We’ve repelled ‘nonstop’ DDoS attacks from Russia*. 2022. URL: <https://venturebeat.com/2022/03/07/ukraine-weve-repelled-nonstop-ddos-attacks-from-russia/> (visited on Mar. 17, 2022).
- [3] Jan von der Assen. “DDoSGrid 2.0: Integrating and Providing Visualizations for the European DDoS Clearing House”. In: *University of Zurich* (2021).
- [4] Jan von der Assen et al. “Analysis and Classification of Cyberattack Traffic using the SecGrid Platform”. In: LCN. 2021.
- [5] Aaron Bangor, Philip T Kortum, and James T Miller. “An empirical evaluation of the system usability scale”. In: *Intl. Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594.
- [6] Luc Boillat et al. “A Tool for Visualization and Analysis of Distributed Denial-of-Service (DDoS) Attacks”. In: *Communication Systems Group, Department of Informatics, Universität Zürich* (2020).
- [7] José Jair Cardoso de Santanna. “DDoS-as-a-Service: Investigating Booter Websites”. PhD thesis. University of Twente, Nov. 2017.
- [8] Chih-Chieh Chen et al. “Detecting amplification attacks with Software Defined Networking”. In: *2017 IEEE Conference on Dependable and Secure Computing*. 2017, pp. 195–201.
- [9] Hyunsang Choi and Heejo Lee. “Identifying botnets by capturing group activities in DNS traffic”. In: *Computer Networks* 56.1 (2012), pp. 20–33.
- [10] Cloudflare. *DNS amplification attack*. 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/dns-amplification-ddos-attack/> (visited on Apr. 3, 2022).
- [11] Cloudflare. *How are IP booters different from botnets?* 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/ddos-attack-tools/ddos-booter-ip-stresser/> (visited on Mar. 31, 2022).
- [12] Cloudflare. *HTTP flood attack*. 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/http-flood-ddos-attack/> (visited on Mar. 31, 2022).
- [13] Cloudflare. *Network-layer DDoS attacks*. 2022. URL: <https://blog.cloudflare.com/ddos-attack-trends-for-2021-q4/> (visited on Apr. 2, 2022).
- [14] Cloudflare. *Protocol attacks*. 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/> (visited on Apr. 2, 2022).

- [15] Cloudflare. *SYN flood attack*. 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/syn-flood-ddos-attack/> (visited on Apr. 2, 2022).
- [16] Cloudflare. *What are the categories of denial-of-service attacks?* 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/ddos-attack-tools/ddos-booter-ip-stresser/> (visited on Mar. 24, 2022).
- [17] Cloudflare. *What is a Botnet?* 2022. URL: <https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/> (visited on Mar. 24, 2022).
- [18] Cloudflare. *What is an Application Layer DDoS attack?* 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/application-layer-ddos-attack/> (visited on Mar. 31, 2022).
- [19] Cloudflare. *What is layer 7?* 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/what-is-layer-7/> (visited on Mar. 31, 2022).
- [20] Cloudflare. *What is the OSI Model?* 2022. URL: <https://www.cloudflare.com/en-gb/learning/ddos/glossary/open-systems-interconnection-model-osi/> (visited on Mar. 31, 2022).
- [21] Amrita Dahiya and Brij B Gupta. “A reputation score policy and Bayesian game theory based incentivized mechanism for DDoS attacks mitigation and cyber defense”. In: *Future Generation Computer Systems* 117 (2021), pp. 193–204.
- [22] *DDoS Clearing House*. 2020. URL: <https://github.com/ddos-clearing-house> (visited on Mar. 17, 2022).
- [23] *DDoS Clearing House – Dissector fingerprint format*. 2022. URL: https://github.com/ddos-clearing-house/ddos_dissector/blob/main/fingerprint_format.md (visited on July 26, 2022).
- [24] *Docker overview*. 2022. URL: <https://docs.docker.com/get-started/overview/> (visited on Aug. 7, 2022).
- [25] Muriel Franco et al. “SecGrid: a Visual System for the Analysis and ML-based Classification of Cyberattack Traffic”. In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE. 2021, pp. 140–147.
- [26] Getoar Gallopeni. *Botnet Command-and-Control Traffic Analysis*. Communication Systems Group, Department of Informatics, 2020. URL: <https://bit.ly/2Z0HgoY>.
- [27] Akshat Gaurav, Brij B. Gupta, and Prabin Kumar Panigrahi. “A novel approach for DDoS attacks detection in COVID-19 scenario for small entrepreneurs”. In: *Technological Forecasting and Social Change* 177 (2022), p. 121554.
- [28] A. Gruhler, B. Rodrigues, and B. Stiller. “A Reputation Scheme for a Blockchain-based Network Cooperative Defense”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*. 2019, pp. 71–79.
- [29] Nazrul Hoque, Dhruba K. Bhattacharyya, and Jugal K. Kalita. “Botnet in DDoS Attacks: Trends and Challenges”. In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2242–2270.
- [30] Imperva. *2021 Global DDoS Threat Landscape Report*. 2021. URL: https://www.imperva.com/resources/reports/Imperva_2021-DDoS-Report.pdf (visited on Mar. 17, 2022).
- [31] Imperva. *The Top 10 DDoS Attack Trends*. 2015. URL: https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf (visited on Apr. 2, 2022).
- [32] Flexible IR. *Playbook for DDoS*. 2020. URL: <https://playbooks.flexibleir.com/ddos-playbook/> (visited on Apr. 22, 2022).

- [33] Flexible IR. *Playbook for DDoS*. 2020. URL: <https://board.flexibleir.com/b/Tg9w9eR6pULq8XRZ2/1> (visited on Apr. 22, 2022).
- [34] Ghafar A Jaafar, Shahidan M Abdullah, and Saifuladli Ismail. “Review of recent detection methods for HTTP DDoS attack”. In: *Journal of Computer Networks and Communications* 2019 (2019).
- [35] Mohammad Karami, Youngsam Park, and Damon McCoy. “Stress Testing the Booters: Understanding and Undermining the Business of DDoS Services”. In: *Proceedings of the 25th International Conference on World Wide Web. WWW '16*. 2016, 1033–1043.
- [36] R Kesavamoorthy and K Ruba Soundar. “Swarm intelligence based autonomous DDoS attack detection and defense using multi agent system”. In: *Cluster Computing* 22.4 (2019), pp. 9469–9476.
- [37] Christian Killer, Bruno Rodrigues, and Burkhard Stiller. “Security Management and Visualization in a Blockchain-based Collaborative Defense”. In: *ICBC 2019*. 2019, pp. 108–111.
- [38] Christian Killer, Bruno Rodrigues, and Burkhard Stiller. “Threat Management Dashboard for a Blockchain Collaborative Defense”. In: *The IEEE GLOBECOM Workshop 27th on Blockchain in Telecommunications: Emerging Technologies for the Next Decade and Beyond*. 2019, pp. 1–6.
- [39] Jiabin Li et al. “RTVD: A Real-Time Volumetric Detection Scheme for DDoS in the Internet of Things”. In: *IEEE Access* 8 (2020), pp. 36191–36201.
- [40] Yuchong Li and Qinghui Liu. “A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments”. In: *Energy Reports* 7 (2021), pp. 8176–8186.
- [41] Ahmed M. Manasrah, Thair Khmour, and Raeda Freehat. “DGA-based botnets detection using DNS traffic mining”. In: *Journal of King Saud University - Computer and Information Sciences* (2022).
- [42] Stephan Mannhart. “Mitigation as a Service in a Cooperative Network Defense”. MA thesis. Universität Zürich, 2018.
- [43] Michael Mattsson, Hakan Grahn, and Frans Mårtensson. “Software architecture evaluation methods for performance, maintainability, testability, and portability”. In: *Second International Conference on the Quality of Software Architectures*. Cite-seer. 2006, p. 18.
- [44] *NeDB GitHub*. 2022. URL: <https://github.com/louischatriot/nedb> (visited on Aug. 29, 2022).
- [45] NETSCOUT. *What is a Reflection Amplification Attack?* 2022. URL: <https://www.netscout.com/what-is-ddos/what-is-reflection-amplification-attack> (visited on Apr. 3, 2022).
- [46] NETSCOUT. *What is a Volumetric DDoS Attack?* 2022. URL: <https://www.netscout.com/what-is-ddos/volumetric-attacks> (visited on Apr. 3, 2022).
- [47] F5 Network. *F5 DDoS Playbook: A Procedural Survival Guide to Combating DDoS Attacks*. 2015. URL: <https://www.oar.net/sites/default/files/page-files/F5%20DDOS%20PLAYBOOK%20092915%20FINAL%20%281%29.pdf> (visited on Apr. 22, 2022).
- [48] Azure Networking. *Azure DDoS Protection—2021 Q3 and Q4 DDoS attack trends*. 2022. URL: <https://azure.microsoft.com/en-us/blog/azure-ddos-protection-2021-q1-and-q2-ddos-attack-trends/> (visited on Mar. 17, 2022).

- [49] Sandro Padovan et al. *DDoSGrid 3.0: Enabling the Real-time Processing and Analysis of Cyber Attacks Traffic*. 2022. URL: <https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mp-padovan-nadig-birchler.pdf> (visited on Apr. 7, 2022).
- [50] Amit Praseed and P Santhi Thilagam. “HTTP request pattern based signatures for early application layer DDoS detection: A firewall agnostic approach”. In: *Journal of Information Security and Applications* 65 (2022), p. 103090.
- [51] Bullet Proof. *Bulletproof annual cyber security report 2019*. 2019. URL: <https://www.bulletproof.co.uk/industry-reports/2019.pdf> (visited on Mar. 17, 2022).
- [52] B. Rodrigues et al. “Evaluating a Blockchain-based Cooperative Defense”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*. 2019, pp. 533–538.
- [53] Bruno Rodrigues and Burkhard Stiller. “The Cooperative DDoS Signaling based on a Blockchain-based System”. In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2021, pp. 760–765.
- [54] Bruno Rodrigues et al. “A blockchain-based architecture for collaborative DDoS mitigation with smart contracts”. In: *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, Cham. 2017, pp. 16–29.
- [55] Bruno Rodrigues et al. “Blockchain signaling system (bloss): Cooperative signaling of distributed denial-of-service attacks”. In: *Journal of Network and Systems Management* 28.4 (2020), pp. 953–989.
- [56] Bruno Rodrigues et al. “Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks”. In: *Journal of Network and Systems Management* 28.3 (2020), pp. 1–27.
- [57] Bruno Rodrigues et al. “SC-FLARE: Cooperative DDoS Signalingbased on Smart Contracts”. In: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2020)*. 2020, pp. 1–3.
- [58] Thijs Rozekrans, Matthijs Mekking, and Javy de Koning. “Defending against DNS reflection amplification attacks”. In: *University of Amsterdam System & Network Engineering RP1* (2013).
- [59] Ahmad Sanmorino and Setiadi Yazid. “DDoS Attack detection method and mitigation using pattern of the flow”. In: *2013 International Conference of Information and Communication Technology (ICoICT)*. 2013, pp. 12–16.
- [60] Ahmad Sanmorino and Setiadi Yazid. “DDoS Attack detection method and mitigation using pattern of the flow”. In: *2013 International Conference of Information and Communication Technology (ICoICT)*. 2013, pp. 12–16.
- [61] C.L. Schuba et al. “Analysis of a denial of service attack on TCP”. In: *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*. 1997, pp. 208–223.
- [62] Paulo Shakarian, Jana Shakarian, and Andrew Ruef. *Introduction to cyber-warfare: A multidisciplinary approach*. 2013.
- [63] Huaglory Tianfield. “Cyber Security Situational Awareness”. In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2016, pp. 782–787.

- [64] Communication Systems Group UZH. *DDoSDB*. 2022. URL: <https://www.csg.uzh.ch/ddosgrid/ddosdb/> (visited on Apr. 7, 2022).
- [65] Communication Systems Group UZH. *DDoSGrid*. 2020. URL: <https://www.csg.uzh.ch/ddosgrid/> (visited on Mar. 17, 2022).
- [66] Cynthia Wagner et al. “Misp: The design and implementation of a collaborative threat intelligence sharing platform”. In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. 2016, pp. 49–56.
- [67] Charles B Weinstock and John B Goodenough. *On system scalability*. Tech. rep. carnegie-mellon univ pittsburgh pa software engineering inst, 2006.
- [68] Lo-Yao Yeh et al. “SOChain: a privacy-preserving DDoS data exchange service over SOC consortium blockchain”. In: *IEEE Transactions on Engineering Management* 67.4 (2020), pp. 1487–1500.
- [69] Saman Taghavi Zargar, James Joshi, and David Tipper. “A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks”. In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2046–2069.

Abbreviations

API	Application Programming Interface
CLI	Command Line Interface
DDoS	Distributed Denial of Service
DDoSCH	DDoS Clearing House
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
OAuth	Open Authorization
PCAP	Packet Capture
SUS	System Usability Scale
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

List of Figures

2.1	Original vs modified SUS [5]	6
2.2	7-layers open systems interconnection (OSI) model [20]	7
2.3	SYN Flood attack based on [15, 59]	9
2.4	Reflection Amplification attack based on [8]	10
3.1	Overview of the DDoSCH components based on [22]	18
3.2	Architecture overview of the DDoSGrid components based on [3]	19
3.3	High Architecture overview of GridDB, DDoS-DB and DDoSGrid V2.2	21
3.4	Detailed High Architecture overview of GridDB, DDoS-DB and DDoSGrid V2.2	22
3.5	Docker overview of the DDoSGrid V2.2 stacks	26
3.6	Docker overview of the GridDB components	28
3.7	Docker Network list	29
3.8	File structure of the Data folder - GridDB	30
3.9	Element of the analysis files array	33
3.10	Example of an Analysis Directory - GridDB Backend	34
4.1	Landing Page - GridDB Frontend	38
4.2	Bottom of the Landing Page - GridDB Frontend	38
4.3	Dissector Subproject - GridDB Frontend	39
4.4	Dissector Subproject Results - GridDB Frontend	40
4.5	Login Page - GridDB Frontend	41
4.6	Analysis in Progress - GridDB Frontend	41

4.7	Analysis Page - GridDB Frontend	42
4.8	Analysis Information Page (top) - GridDB Frontend	43
4.9	Analysis Information Page (bottom) - GridDB Frontend	44
4.10	Docker Build Time Top 5 Commands - GridDB	47
4.11	Automation Process - GridDB	50

List of Tables

2.1	Categories of DDoS attacks	6
3.1	Specification of the REST Interface: API	25
3.2	Specification of the REST Interface: DDoSDB	25
3.3	Specification of the REST Interface: GridDB	29
4.1	Table of contributions	51

Listings

3.1	Check token function in DDoS-DB	23
3.2	-send_fp tag added to the Dissector	23
3.3	check and parse arguments	23
3.4	configuration file example	24
3.5	sending fingerprint function	24
3.6	Docker file for converters	24
3.7	Docker-Compose DDoSGrid V2.2	25
3.8	Docker Commands GridDB	27
3.9	Docker Commands GridDB - Networking	28
3.10	Handle upload PCAP file - GridDB FrontEnd	30
3.11	Try-catch block for the Dissector subproject	31
3.12	Try-catch block for the Miner subproject	32
3.13	DDoSGrid V2.2 response of the Miner endpoint	32
3.14	Start Analysis Function GridDB - Frontend	34
3.15	Docker-compose for MongoDB	35
3.16	Anylsis Schema	36
4.1	Docker Commands DDoSGrid V2.2	45
4.2	Final Docker Commands DDoSGrid V2.2	46
4.3	Docker URI handling example 1	46
4.4	Docker URI handling example 2	47
4.5	Docker services for GridDB	48
4.6	Docker Network for DDoSDB	49