University of Zurich^UZH

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

Design and Implementation of a Distributed Denial-of-Service Data Simulator

Reto Odoni
Zürich, Switzerland
Student ID: 15-701-246

Supervisor: Muriel Franco, Bruno Rodrigues
Date of Submission: August 22, 2019

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

ifi

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

Zusammenfassung

0.1 Einleitung

DDoS Attacken sind verteilte Netzwerkangriffe auf Internetdienste mit der Beabsichtigung deren Verfügbarkeit zu stören. Durch eine Überflutung von Anfragen durch Angreifer können diese Dienste für die eigentlichen Benutzer sogar gänzlich unerreichbar werden. Um solche Nichtverfügbarkeiten und die damit verbundenen potenziellen finanziellen oder persönlichen Schäden zu vermeiden, suchen Forscher und Unternehmen gleichermassen nach Möglichkeiten, Kriminellen einen Schritt voraus zu sein. Um Abwehrmechanismen gegen verschiedene Formen von DDoS-Angriffen zu schaffen, werden häufig Log Daten von zuvor angegriffenen Dienstleistern herbeigezogen.

0.2 Ziele

Der DDoS Log Simulator ist ein Tool welches Netzwerk Log Daten in Form eines *pcap* Dateiformates (Wireshark) erzeugt. Das Ziel des Programms ist es als eine Plattform zu fungieren auf welcher Netzwerk Log Daten, ähnlich wie bei einem echten DDoS-Angriff, simuliert werden können.

0.3 Resultate

Der DDoS Log Simulator ist ein Werkzeug, welches verschiedene Arten von DDoS-Angriffen, basierend auf einer Konfigurationsdatei, generieren kann. Er ist in der Lage, das Verhalten und die Hauptmerkmale eines typischen DDoS-Angriffs zu demonstrieren und passt sich jeweils an das ihn ausführende System an. Da jede Art von Angriff leicht unterschiedliche Eigenschaften aufweist, wurde der DDoS Log Simulator mit der Absicht erstellt leicht erweiterbar zu sein.

0.4 Weitere Arbeiten

Der DDoS Log Simulator wurde mit der Absicht gebaut, so einfach wie möglich modifizierbar zu sein, somit bringt dieser viel Potential für Erweiterungen mit sich. Dies

könnte unter anderem Folgendes beinhalten: (i) Unterstützung von weiteren Arten von DDoS-Angriffen, (ii) senden der erzeugten Angriffe ins Netz, (iii) Erstellen von visuellen Ausgabemöglichkeiten oder (iv) trainieren von künstlicher Intelligenz basierend auf den erzeugten Log Dateien.

Abstract

By flooding publicly reachable network infrastructure with superfluous requests, attackers can get those infrastructures unreachable for its intended users. Such attacks are known as Distributed Denial-of-Service (DDoS) attacks and are a major concern of online service providers. To prevent such unavailabilities and associated potential financial or personal damage, researchers and corporations alike are searching for ways to be one step ahead of criminals. To create defense mechanisms against different forms of DDoS attacks, logs of previously attacked service providers are consulted. Unfortunately, those records are of limited utility, not anonymized and hard to get. The thesis at hand presents the DDoS Log Simulator which offers an easy way to create anonymous and randomized log files of preconfigured DDoS attack types. The program supports two types of attacks, namely the SYN flood and DNS amplification attack, which were implemented as a proof of concept. Various parameters can be adjusted, such as list of attackers with respective IP addresses and number of packets per second that are being sent. The performance of the DDoS Log Simulator was measured and the generated log files were feed through a commonly used tool for fingerprinting DDoS attacks. Also it was elaborated how the DDoS Log Simulator could be extended with further DDoS attacks and stated that the program can be considered as a foundation for creating a whole range of possible future work.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Muriel Franco, for his support and feedback during the whole creation process of this thesis. I would also like to thank Bruno Rodrigues for listening to my presentation dry-runs and providing me with feedback afterwards. Additionally, I would like to thank Professor Dr. Burkhard Stiller for the opportunity to write my thesis at his chair.

Contents

Chapter 1

Introduction

Distributed denial of service (DDoS) attacks are one of the top security concerns of online service providers [14]. By flooding a publicly reachable service or network resource with superfluous requests, an attacker tries to overload it and make it unavailable to its intended audience. In a time with a growing amount of Internet of Things (IoT) devices with insufficient security mechanisms [16] and businesses moving their infrastructure to the cloud, DDoS attacks can be considered a growing problem, since even short server downtime can result in serious financial or even personal damage.

DDoS attacks are a rapidly growing problem in the modern Internet era. During the last couple of years, the damage caused by DDoS attacks has been constantly on the rise. For example in 2016, some of the biggest Swiss online stores got blackmailed and went offline when they became targets of huge DDoS attacks [9]. In 2018 a DDoS attack reached an all-time high bandwidth while attacking GitHub at an astonishingly 1.3 terabytes per second with a peak of sending 126.9 million packets per seconds according to Cloudflare [2]. There are a lot of reasons and variables which explain this trend and show that this situation will likely continue. At first there is the growing dependency of the economy on the Internet and the society's constant urge to be online. Businesses are moving their infrastructure to the cloud and people are communicating with each other using Instant Messaging and are streaming videos online. As a consequence, the Internet grew accordingly, namely fiber, vectoring and faster cellular network technologies, which paved the way for high-bandwidth attacks, which DDoS attacks are a part of.

In the 1970s, the Internet was designed without security in mind or the possible of a growth to such an extent. The unexpected growth is especially visible in the IPv4 address exhaustion. Today, due to design choices back then, DDoS attacks are simple to execute in an easy and cheap way. A DDoS attack can be deployed by either using premade tools or buying one as a service. As a result of this development, both researchers and companies are searching for solutions. As soon as an attack is targeting a specific application, there is no readily available solution, but there needs to be a customized one. Therefore the market of protection services grows with the amount of different types of attacks, which results in a higher need of expertise.

1.1 Motivation

DDoS attacks appear in different forms, targeting different layers of online service providers. Since it is relatively easy to start a DDoS attack, all service providers should prepare their infrastructure appropriately. This includes, besides developing a defense mechanism for each kind of attack, also to make sure that their created solutions trigger in the right case of an attack. To ensure this, log files from real attacked servers are consulted. However, such records are not easy to get from public databases (*e.g.* from DDoSDB[1]) because they contain sensitive information from the attacked service providers. Furthermore, they contain the source address, where the attack originated, uncovering potentially insecure devices.

Since the attacks tend to grow in size and become more dangerous [11], new ways are needed to be researched to defend publicly reachable infrastructure. For device administrators it is crucial to be aware of what a DDoS attack could look like on a network level. It is important to have data which can be analyzed for mitigation purposes.

At the present time, there are tools available for simulating different kinds of DDoS attacks. However, simply launching a massive attack is of no use if it cannot be logged for later analysis. Instead it would be better to write the attack directly into a log file. However, this still sets restrictions to the possible scenarios capable of generating, especially in terms of computing powers. Mainly, the amount of packets an attacker can generate per second is strongly dependent on the available computing power. Therefore a tool is needed which is capable of generating as much packets as demanded and log those packets.

Furthermore, it can be argued that log files from DDoS attacks, even when simulating a hypothetical scenario, can be helpful for researchers to identify patterns and understand behaviors of DDoS attacks. For example, it would be possible to use simulated log files to validate forensic techniques or evaluate the accuracy of novel DDoS mitigation solutions.

1.2 Description of Work

This thesis presents the development of a DDoS Log Simulator, which is aimed to help cybersecurity researchers and developers to validate and evaluate DDoS solutions. The DDoS Log Simulator is a tool to generate network packet log data in form of a *pcap* file (Wireshark), similar to how it would appear in a real DDoS attack. It tackles the problem of providing anonymized and randomized log files for custom attack scenarios and supports two different attack types (SYN flood and DNS amplification) and a wide variety of configurable parameters.

The generated attack log is parameterizable based on the settings in a configuration JSON file. Each such file may represent a different kind of DDoS attack with varying severities on the given parameters to reflect different attack scenarios.

[1]`https://ddosdb.org/`

1.3 Thesis Outline

The thesis itself is structured as follows: Chapter 1 gives a brief overview about the thesis itself and the problems the DDoS Log Simulator tries to tackle. Chapter 2 gives context and mentions related work. Chapter 3 is about the design, the functionality and the scope of the DDoS Log Simulator itself. Chapter 4 contains the evaluation of the method and discusses its limitations. In Chapter 5 conclusions are taken and future application purposes are discussed.

Chapter 2

Background

The DDoS Log Simulator presented in this thesis can be classified as a supporting tool for creating solutions against DDoS attacks. By providing access to customized Wireshark *pcap* files, researchers can test their DDoS defense mechanisms against these scenarios. At first, the context around the DDoS Log Simulator is explained and any prior knowledge is introduced. Afterwards related work is presented and shortly summarized.

2.1 Internet Protocol

The Internet Protocol (IP) is the communication protocol of the Internet, relaying datagrams between hosts. Its revolutionary feature was that it relayed packets between hosts based only on the IP addresses in the packet headers, making it connectionless. This distinguished it from more reliable connection-oriented protocols. Despite the disadvantages of best-effort delivery such as possible data corruption, packet loss and duplication, IP's scalable architecture eventually led to its worldwide success.

To achieve often desirable connection-oriented services, TCP (Transmission Control Protocol) was added to the Internet Protocol suite. It provides reliable, ordered and error-checked full-duplex communication between two hosts. It contrasts with the connectionless datagram service UDP (User Datagram Protocol) that emphasizes reduced latency over reliability.

The Internet Protocol (IP) was designed primarily with scalability in mind, to support a large number of hosts attached to networks. Security was a secondary consideration, as nobody could have predicted the worldwide success that ensued.

2.2 Classification of DDoS Attacks

A denial of service (DoS) attack is any attack with a single origin where a publicly reachable infrastructure gets inaccessible to its intended users. A DDoS attack is a distributed denial of service attack, which means an attack has several origins.

DDoS attacks can be classified by their degree of automation, their exploited vulnerability, their attack rate dynamics or by their impact on a targeted device [3]. In this thesis the DDoS attacks are classified by their exploited vulnerability since the DDoS Log Simulator supports only the creation of log files from the used vulnerability. Those attacks can be subdivided into flood attacks, amplification attacks, protocol exploit attacks and malformed packet attacks.

DoS attacks are constrained by properties of networks. For example every endpoint only has a limited amount of network bandwidth available to send large amounts of traffic. Due to TCP's three-way handshake, a connection can only be established if both endpoints (defined by its IP addresses) are willing to do so. Sizes of individual packets are constrained by the maximum transmission unit (MTU), that is 1500 bytes over the public Internet.

DoS attacks however benefit from weaknesses of internet protocols, such as the fact that IP source addresses can be spoofed, if this is not prevented by the ISP (Internet Service Provider). This however mainly affects UDP communication, as TCP communication will not continue past the connection establishment phase, due to the fact that SYN, SYN-ACK and ACK packets need to be exchanged.

The two implemented attacks in this thesis are (i) a flood attack, namely the SYN flood and (ii) the DNS amplification attack which, as the name suggests is a part of the amplification attacks.

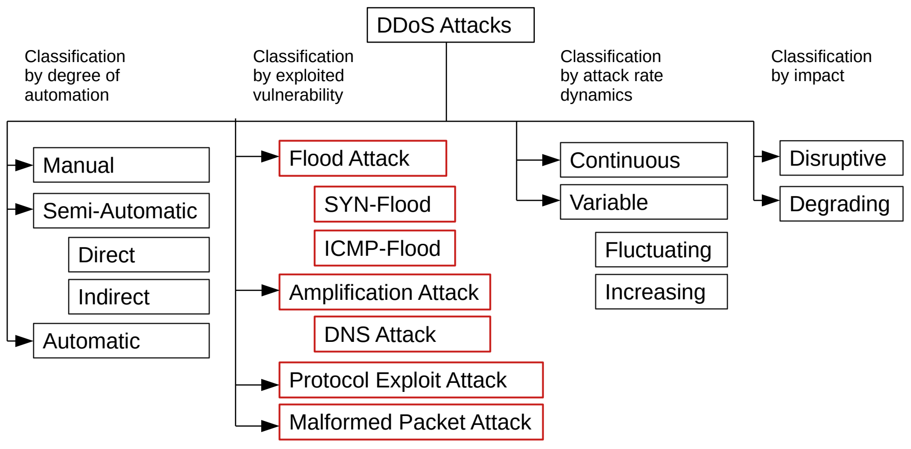

Figure 2.1: Slightly adjusted classification of DDoS attacks from C. Douligeris, A. Mitrokotsa / Computer Networks p. 650

2.2.1 SYN Flood Attack

One of the most powerful flooding methods are SYN flood attacks [1]. The SYN flood attack uses a weak spot in the way a secure connection is established with a server. Namely in the three-way handshake, which is initiated when a TCP connection is established with

Table 2.1: Table of a SYN Flood attack generated by the DDoS Log Simulator.

No.	Time	Source	Destination	Protocol	Length	Info
129	0.012936	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
130	0.013017	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
131	0.013112	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
132	0.013174	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
133	0.013178	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
134	0.013282	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
135	0.013330	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
136	0.013347	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
137	0.013497	230.178.74.14	10.0.0.11	TCP	54	81 → 80 [SYN] Seq=0 Win=8192 Len=0
138	0.013508	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
139	0.013629	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
140	0.013681	10.0.0.13	10.0.0.11	TCP	54	[TCP Retransmission] 81 → 80 [SYN] Seq=0 Win=8192 Len=0
141	0.014070	181.178.56.161	10.0.0.11	TCP	54	81 → 80 [SYN] Seq=0 Win=8192 Len=0
142	0.014629	10.0.0.14	10.0.0.11	TCP	56	[TCP Retransmission] 82 → 80 [SYN] Seq=0 Win=8192 Len=2

a server. During a normal three-way handshake, a client asks for a connection by sending a synchronization message (SYN) to the server. The server acknowledges it by sending a SYN-ACK packet back and if the client also acknowledges, the secure connection is established [18].

Since a server has to wait for a response for a certain threshold, in an attack scenario, an attacker tries to hold open as many SYN requests as possible such that the server has no resources left for its intended users. The attacker cannot receive SYN-ACK responses due to them being delivered to the forged IP address. The server will not receive an ACK back from the alleged client, as the unknowing host with the forged IP address does not recognize the packet and drops it. This way, a single client is capable of overloading an unprotected server without even uncovering its own IP address. During a DDoS attack, multiple compromised clients try to establish a multitude of connections. This may go easily far beyond the boundaries of a single standard host [18]. An example of a typical SYN flood attack log with a low amount of real traffic noise, generated from the DDoS Log Simulator, is visible in the table 2.1.

2.2.2 DNS Amplification Attack

In the DNS amplification attack scenario, high-capacity publicly reachable DNS servers are abused. The attack makes use of the fact that it is possible to turn a small DNS request into a comparatively much larger response (the difference is called amplification factor). For instance, a small request of 60 bytes (e.g. for an ANY resource record) can result in an answer two magnitudes larger. Normally, a DNS server's main task is to answer requests for name resolution purposes and thus is an important service in many IP-based networks. By flooding DNS servers with spoofed DNS requests, it is possible to have several response streams to a single victim's destination. Doing so can eventually result in a denial of service by overloading the target with UDP (User Datagram Protocol) packets. A key part in the whole attack plays the fact that sent UDP packets are not further checked for successful arrival by the sender. The presence of datagrams with these properties may be helpful for delay-sensitive applications such as sending real-time audio and video streams, but prevents the receiver to indicate its unwanted receiving of packets. In an attempt to prevent DDoSers from having a plethora of DNS servers available that

Table 2.2: Table of a generated DNS amplification attack.

No.	Time	Source	Destination	Protocol	Length	Info
137	0.012477	204.70.136.80	131.141.91.199	DNS	1326	Standard query response 0x1f1a ANY sema.cz DS RRSIG NS l...
138	0.012536	204.70.136.80	131.141.91.199	DNS	566	Standard query response 0x4eda ANY irs.gov SOA ns1.irs.g...
139	0.012786	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x4eda ANY irs.gov MX 10 emg1.ir...
140	0.012799	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x76df ANY sema.cz RRSIG MX 1 as...
141	0.012816	204.70.136.80	131.141.91.199	DNS	566	Standard query response 0x4eda ANY irs.gov SOA ns1.irs.g...
142	0.012819	55.215.42.251	131.141.91.199	TCP	54	82 → 80 [SYN] Seq=0 Win=8192 Len=0
143	0.012869	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x4eda ANY irs.gov MX 10 emg1.ir...
144	0.012971	204.70.136.80	131.141.91.199	DNS	566	Standard query response 0x4eda ANY irs.gov SOA ns1.irs.g...
145	0.013018	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x4eda ANY irs.gov MX 10 emg1.ir...
146	0.013021	72.4.104.163	131.141.91.199	TCP	54	82 → 80 [SYN] Seq=0 Win=8192 Len=0
147	0.013043	204.70.136.80	131.141.91.199	DNS	566	Standard query response 0x4eda ANY irs.gov SOA ns1.irs.g...
148	0.013405	204.70.136.80	131.141.91.199	DNS	1326	Standard query response 0x1f1a ANY sema.cz DS RRSIG NS l...
149	0.013450	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x4eda ANY irs.gov MX 10 emg1.ir...
150	0.013623	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x76df ANY sema.cz RRSIG MX 1 as...
151	0.013703	204.70.136.80	131.141.91.199	DNS	2958	Standard query response 0x76df ANY sema.cz RRSIG MX 1 as...
152	0.013716	204.70.136.80	131.141.91.199	DNS	566	Standard query response 0x4eda ANY irs.gov SOA ns1.irs.g...
153	0.013876	179.105.61.70	131.141.91.199	TCP	54	82 → 80 [SYN] Seq=0 Win=8192 Len=0
154	0.013997	204.70.136.80	131.141.91.199	DNS	758	Standard query response 0x1f1a ANY sema.cz RRSIG DS NS d...

they may use for amplification attacks, DNS servers of ISPs should only serve their own customers and not accept requests from outside its own IP ranges.

Those generated DNS answer streams can be up to 20 times larger than the originally sent request [10] and thus an attacker can overwhelm a big target with considerably less available network bandwidth. An attacker forges the IP source address of the packet request such that it is not possible to determine where it actually originated. An easy and effective way to prevent DDoS amplification attacks is accomplished through the support of internet service providers. They should check in an egress filter if the IP addresses of outgoing IP packets from customers match the one associated with the client sending it (see BCP38 for further information). Another effort to reduce DNS amplification attacks can be made by ensuring that DNS answers are not too large (see DNSSEC records, that store bulky public keys in the DNS) with the goal of reducing the amplification factor of attacks. DNSSEC ANY responses may be up to 50 times, in rare cases up to 179 times larger than the size of the query sent. [17]

Table 2.2 represents a snippet from an example DNS amplification generated by the DDoS Log Simulator. It starts at packet number 17113 and represents 3 DNS servers responding to a single destination IP. In between all the DNS responses, there is a single connection that tries to be established.

2.3 Scapy

Scapy[1] is a packet manipulation tool written in Python by Philippe Biondi with support from the Scapy community. Besides forging packets and storing them in *pcap* files, Scapy is also capable of reading, decoding and sending them on a network socket. Another feature its possibility to store packets into *pcap* files which made Scapy very suitable for the DDoS Log Simulator. It supports Python 2.7 as well as Python 3.4 to 3.7 and is designed to be cross platform. It can be used either as a shell utility or as a library [12]. Further more

[1] `https://scapy.net/`

Scapy supports the installation of modules which provide further functionalities such as plotting.

2.4 DDoS Clearing House

The DDoS Clearing House and hence also the DDoSDB are part of the CONCORDIA ecosystem. CONCORDIA, which is short for cyber security competence for research and innovation and states itself as a Cybersecurity Competence Network. The DDoS Clearing House was made to be a bridge between victims, DDoS protection providers, network operators, law enforcement agencies and the academia. Mainly the DDoS Clearing House consists out of the DDoS Dissector, DDoS Fingerprint Converters and the DDoSDB. The DDoS Dissector takes network measurements and based on them produces a fingerprint. Afterwards the DDoS Fingerprint Converters should produce rules and signatures for specific hardware / software solutions. The whole DDoS Clearing House and all its associated parts, especially the DDoSDB and the DDoS Dissector, are Open Source and visible at GitHub[2].

2.4.1 DDoSDB

The DDoSDB[3] is a database which collects and exchanges information about DDoS attacks to help victims, the academic community and security network forensic experts. By sharing logs of attacks the DDoSDB tries to enable a joint effort against DDoS attacks which should provide the user with better understanding of typical DDoS attack characteristics. This approach is supported with the capability of a fingerprint generator from its sister project of the DDoS Clearing House. It extracts properties from network log files and allows users to search for them in the database. During the process of generating the fingerprint of an attack the fingerprint generator anonymizes the identity of the victim and creates a new log file which then can be contributed to the DDoSDB. Unfortunately the DDoSDB is not publicly available and users have to state a purpose during the registration process.

2.4.2 DDoS Dissector

As the DDoSDB, the DDoS Dissector is also a part of the DDoS Clearing House. Its main purpose is to take network log files in form of different input formats for example *pcap*, *net flow*, *ipfix* or *sflow* log data and generate a fingerprint in which only the attacking traces remain. While the program generates the fingerprint it removes all the real traffic noise packets and anonymizes the input network trace. The generated fingerprint shows a variety of different parameters such as the duration of an attack, bytes per second, packets per second, time stamp as well as attack specific details.

[2]`https://github.com/ddos-clearing-house`

[3]`https://ddosdb.org/`

2.5 Wireshark

Wireshark[4] is an open-source packet analyzer published under the GNU General Public License (free software) which is widely used for network troubleshooting and analysis purposes. It is a cross-platform software and works under Unix, Linux, Solaris, Mac, Windows and different versions of BSD. The program is developed by a community of networking experts across the globe and was originally released in 1998 by Gerald Combs [19].

Besides the capability to inspect hundreds of protocols, one of Wireshark's main features is capturing live network traffic from a wired/wireless connection. This can be used for analyzing observed network traffic, but also as a general education tool to understand communication protocols. Wireshark can capture packets and store them in *pcap* files. These allow reproducing the flow of packets as observed.

In the context of DDoS attacks, Wireshark allows network admins to record ongoing attacks against a specific infrastructure for later inspection and analysis. It would be beneficial for the network security community to have access to such log files, such that they can learn from them and improve their defence mechanisms. However, the presence of identifying information such as IP address policies complicate the publication of such log data. Anonymization of such log data is important, but not always possible due to company policy or there is the risk of losing essential information that were necessary to properly understand the attack.

As a small addition, while the records from DDoSDB are not publicly visible, Wireshark shares a quite large list of sample captures in their wiki [5]. Also it is possible with Wireshark to modify different *pcap* files with a variety of functions and hence to create interesting log files.

2.6 Related Work

Since DDoS attacks are capable of impacting economy and society in a tremendous way, a lot of research is done in the field. However, at the time of writing, there was no tool found with the same features as the DDoS Log Simulator presented in this thesis. The most related work will be presented in this section.

During the process of creating the program as well as writing the thesis, it was necessary to dig into two different topic fields. Firstly, there are already similar programs compared to the DDoS Log Simulator available which are capable of creating DDoS attacks *e.g* on peer-to-peer (P2P) networks or on the session initiation protocol (SIP). Secondly knowledge regarding DDoS attacks in general had to be acquired as well as for specific types of attacks (*e.g.* SYN flood attack).

[4]`https://www.wireshark.org/`

[5]`https://wiki.wireshark.org/SampleCaptures`

For example in [13] Nidal Qwasmi *et al.* created a DDoS attack simulator on P2P networks for demonstration purposes. They stated that their outcome in the P2P network was mostly in line with what can be expected in a normal DDoS attack. The most interesting difference they noticed was an impact on not only the target but the whole P2P network by slowing down performance as well as dynamism and flexibility. In [15] Jan Stanek *et al.* implemented a tool to generate Session Initiation Protocol (SIP) DDoS flood attacks and compared the generated attack with real data by modifying a popular SIPp call generator. In 2011 Andrzej Kosowski *et al.* [7] analyzed the different possibilities and technique to perform cyberattacks focusing mainly on DDoS attacks. In their thesis they stated the network security simulator nessi2 as extremely helpful to determine the impact of potential DDoS attacks as well as identifying weak points and best countermeasures.

Since it is critically important to understand DDoS attacks a lot of studies focus on those aspects. As an example in [8] there is the thesis from Z. Morley Mao *et al.* where they analyze large DDoS attacks based on multiple data sources. As another example in [6] Jiahui Jiao *et al.* created an approach to detect DDoS attacks with a rate higher than 99% which was implemented on the Baidu Cloud Computing Data Centers. A statistical approach to detect and response to DDoS attacks was presented by Laura Feinstein *et al.* in [4]. They created methods based on computing entropy and frequency sorted distributions of selected packet attributes.

Chapter 3

DDoS Log Simulator

The DDoS Log Simulator is a tool that is able to create different types of DDoS attacks based on a predefined configuration. It is capable of demonstrating the behaviour and main characteristics of a typical DDoS attack. Since each type of attack has slightly different characteristics the whole DDoS Log Simulator is meant to be easily extendable allowing users to implement further attack types and provide a memory light scalable platform to generate network logs from.

The DDoS Log Simulator is started in the terminal via the command in listing 3.1. The only parameter which has to be stated for a execution to start is the configuration file. Note, the configuration file declared in this example is named `syn.json` but could also have any other name. A program under Linux which came in handy during the development process was `nice`. It allows the user to execute a program in reduced priorty mode. An example how to execute the DDoS Log Simulator with nice is found in listing 3.2. For further information consult the manual of nice[1].

```
$ ./log_gen.py syn.json
```

Listing 3.1: Command to start the DDoS Log Simulator via terminal and `syn.json` as configuration file.

```
$ nice -n 19 ./log_gen.py syn.json
```

Listing 3.2: Example command to start the DDoS Log Simulator with decreased priority.

Besides generating *pcap* log files it is also possible to merge two previously generated files via the argument `--merge` respectively `-m`. Since it is possible to create two separate attacks with the same timestamp this feature comes in handy especially when creating multi-vector attacks. An exmple is stated in 3.3.

[1]https://linux.die.net/man/1/nice

```
$ ./log_gen.py -m file1 file2 outputname
```

Listing 3.3: Example command to merge two *pcap* files .

3.1 Overview

The following sections explain the behaviour of the DDoS Log Simulator based on the flow diagram in figure 3.1. The flow diagram is labelled with numbers from 1 to 11 to make it easier to reference the different execution stages. On the bottom left there is a legend in grey describing the different elements.

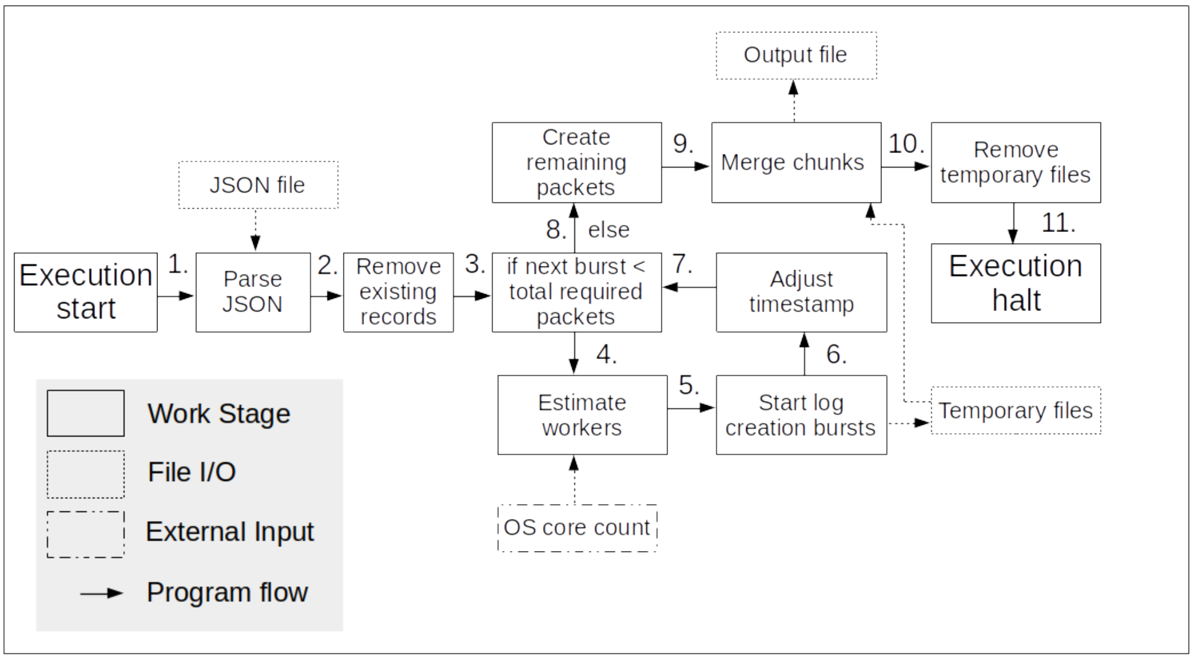

Figure 3.1: A visual representation of the execution flow of the DDoS Log Simulator.

After the program is started, in stage 1 the DDoS Log Simulator begins parsing the JSON configuration file which is referenced in the console command. See reference point 1 in the flow diagram. In stage 2 the DDoS Log Simulator then checks for previously created log files. This procedure is necessary since remaining existing log files would cause trouble while merging them together in stage 9.

One of the main features of the DDoS Log Simulator is the capability to generate arbitrarily long log data independent of the available computational power by making use of multiple threads. This is achieved by the DDoS Log Simulator by iterating over a loop until the requested amount of packets is reached. The iteration process starts at stage 3 and is then reevaluated at 7. Before a new log creation burst starts at stage 5, the DDoS Log Simulator estimates the amount of workers (threads) which have to be deployed at stage 4. This procedure is necessary since the program should generate as many packets as possible while avoiding packets being dropped (see [12] under “Known bugs”) and also prevents an overproduction of packets. To prevent such an overproduction the amount of spawned workers decreases towards the end. After the packet creation process has terminated on every worker, the DDoS Log Simulator starts adjusting the timestamps on all created packets in stage 6. For the first iteration this is achieved by subtracting the difference between the start of the script and the desired start timestamp. For all following iterations the difference between the start of the iteration and the script needs to be subtracted as well. Before a new iteration round starts the DDoS Log Simulator starts counting all already created packets in stage 7 and calculates the overall progress. Based on the progress made, the DDoS Log Simulator determines the amount of workers to be created in the next iteration round.

After the fast and multi-threaded creation of mostly 92% to 99% of the demanded packets a single process creates the remaining packets in stage 8. While not being in a multi threaded environment it is easier to count the amount of packets already generated and thus easy to prevent any overproduction. When stage 8 terminates the DDoS Log Simulator merges all *pcap* chunks to a single file together in stage 9. This produces the final output *pcap* file. Before the execution halts the DDoS Log Simulator removes all temporary created files during stages 5 and 8.

3.2 Design and Challenges

The creation of the DDoS Log Simulator involved various design decisions that had to be made. They often originated from challenges during the development process. This section discusses the most interesting ones and presents them in depth.

3.2.1 Performance

During the development of the DDoS Log Simulator different approaches to tackle the actual log generation process were tested and implemented. The first problem occurred upon realizing that the generation process was way slower than expected and quite CPU intense. In an attempt to solve this problem the decision was made to implement multi-threading support. While doing so it turned out that the available computing power still wasn’t fast enough to create massive amounts of packets per second. Therefore the DDoS Log Simulator sometimes took longer to create the demanded amount of packets than what was stated in the duration parameter. As a consequence the whole architecture had to be revised. The attempt was made to just let the spawned threads write into a single file and count upwards until the demanded amount of packets were reached. Unfortunately this would lead to a corrupt *pcap* file.

In the final solution attempt the generation process was done by constantly creating small chunks of packets, adjusting their timestamps and merging them together. This is possible since Wireshark provides a variety of tools to manipulate individually created *pcap* files. The solution approach also resulted in constant low memory usage regardless of the total size or density of the generated packets.

Since the used processor differs from system to system the DDoS Log Simulator has to estimate approximately the amount of threads it uses in an attempt not to create too many packets. This can be considered a downside with this solution's approach. A different approach that would not need any predictions would have been a global counter. This counter could just stop all threads if the total amount of packets would have been generated. It turned out that this would lead to an uneven distributed packet distribution in some scenarios with only a few packets. Therefore the estimation is based on starting not too fast and detecting the end of a generation process and reducing the amount of spawned threads.

3.2.2 Output File

An important decision is the choice of output file format the program should produce. The most obvious choice was the *pcap* file format which is used by tcpdump/WinDump and Wireshark, namely the log file output format of the two most popular packet sniffing tools. [5]. But besides the *pcap* file format, Wireshark also supports the "*pcap* Next Generation Capture File Format" also known as *pcap*-ng. Due to more resources about *pcap* and better compatibility especially with Scapy, the final decision was to use the old *pcap* format.

3.2.3 User Interface

The DDoS Log Simulator consists of a single script where the log simulation can be started via a console application. All configuration is done in a single file which has to be referenced for the program to execute successfully. At first each parameter was stated on the command line while execution but while implementing more and more features into the DDoS Log Simulator the parameters had to move into a separate file. This approach to separate the configuration file provides a way to store the numerous different parameters smoothly. Also it becomes possible to define and store and share newly created attacks in a convenient way. An intuitive and clear way to store data can be found in the JavaScript Object Notation file format also known as JSON file format. While the JSON file format was initially designed for JavaScript, it is otherwise independent from its origin and parsers exist in most common programming languages.

3.3 Prototype and Implementation

In this section, technical details about the prototype's implementation and a discussion regarding technologies used will be provided. Besides, details about the parameters being used in the configuration file are outlined.

Python 3.7 was picked as the main programming language for the implementation of the prototype, mainly because of its simplicity and capacity to integrate with different libraries to deal with computer networks (*e.g.,* IPv4 packets and *pcap* files). The library used for this purpose was Scapy 2.4.2. Python in combination with Scapy provided all the required features needed to implement the most essential features. This includes (i) spawning different processes, (ii) forging packets, (iii) writing the generated packets into files while (iv) iterating over different *pcap* file chunks. To modify already created packets, Wireshark 2.6.8 was used. Except for a Wireshark-Python wrapper, there was no Python library available for Wireshark and hence the decision was made to execute Wireshark over the OS module.

3.3.1 Configuration JSON

The DDoS Log Simulator offers a wide variety of configuration possibilities in the form of a JSON file. The choice of the parameters was based on the ability of a parameter to reproduce real DDoS attacks as closely as possible. Those parameters and the structure of the JSON file will be explained in detail in this section.

The various parameters in the JSON file are split into general and attacker specific settings. Every parameter expect the packets per second have a default value which will be consulted if nothing else is defined. The packets per second must be defined since the program otherwise cannot know how may attackers it should spawn (see listing 3.5). This should help keep the configuration process as intuitive as possible. The general settings are listed in table 3.1 and define attack specific properties. In the JSON template below the general parameters are the ones only enclosed in the first curly brackets. To support multiple attackers, the attacker specific configuration is placed in a JSON list of multiple such objects.

```
{
  "attack_type": ... ,
  "duration_sec": ... ,
  "real_traffic_noise": ... ,
  "dst_IPv4": ... ,
  "dst_IPv6": ... ,
  "dst_port": ... ,
  "dst_MAC": ...
  "attackers": [
    {
      "IPv4": ... ,
      "IPv6": ... ,
      "MAC": ... ,
      "port": ... ,
      "bandwidth_Bps": ... ,
      "packets_per_sec": ...
    },
    ...
    {
      ...
    }
  ]
}
```

Listing 3.4: JSON configuration file with all parameters shown.

The general configuration parameters contain everything which needs to be configured beforehand. The main parameter is the ”attack_type”. It defines which kind of attack the DDoS Log Simulator should create and thus determines the types of packets generated, like the type of the packet’s protocol (such as TCP/UDP), of its payload or, when available, which flags are set. The default attack type chosen by the DDoS Log Simulator is the SYN Flood attack but only if the program does not recognize the attack (typo). The DDoS Log Simulator uses at least a minimal configuration as shown in listing 3.5. Another parameter is the duration in seconds stated in the JSON file as ”duration_sec”. The duration determines the amount of time during which an attack should take place and can be specified as either an integer or a floating-point number.

Internet Protocol (IP) addresses are essential parts of log files, as they designate purported sources and destination of the attack. IP addresses are assigned to each device which communicates over the internet protocol and are available in either IPv4 or IPv6. While the IPv4 address struggles with its depletion of unassigned addresses, IPv6 should provide a solution for this problem. Both versions are supported. The IPv4 address can be specified as a string via `dst_IPv4` and analogously the IPv6 via `dst_IPv6`. If both addresses are stated, the DDoS Log Simulator favours the IPv6 address. If no address is stated, it generates a random IPv4 address.

Besides the destination address, an attack also needs a communication endpoint, namely a port. The destination port can be set via `dst_port` and has to be an integer. The range of the port has to be between 0 and 65535 which corresponds to the valid range of ports (see RFC6335). The media access control (MAC) address is used in the data link layer of the OSI model and is a unique identifier for a machine. The destination MAC address can be adjusted via `dst_MAC` and has to be a string of characters. If no MAC address is stated, the DDoS Log Simulator creates a random one.

The starting time of an execution can be declared in `start_time` as a Unix timestamp. By defining the timestamp in Unix time format a trade-off was made between usability and time format confusion. If no time is declared the DDoS Log Simulator takes the starting time of the generation as a default value.

```
{
  "attack_type": ... ,
  "attackers": [
    {
      "packets_per_sec": ...
    }
  ]
}
```

Listing 3.5: Minimal JSON configuration to start an execution successfully.

Another characteristic of most DDoS attacks is the occurrence of real people trying to reach their desired server. This parameter is represented in the configuration as `real_traffic_noise` and indicates the amount of legitimate packets besides the DDoS attacker's packets. Each noise packet has a unique destination IP address and sends a SYN packet to the destination IP. The amount of noise can be declared in categories such as `none` (0), `low` (0.04), `medium` (0.08) or `high` (0.16). They describe the percentage of real noise compared to the total amount of attacking packets. Alternatively, the amount of traffic noise can be declared as a floating point number in a range from 0 to 1. The total amount of packets is defined by all attacking packets multiplied by 1 + the numerical representation of `real_traffic_noise`.

DDoS Log Simulator - General Config			
Parameter	Information	Range	Default Value
attack_type	Attack type.	syn_flood, dns$_amplification$	"syn_flood"
duration_sec	Duration time in seconds.	Float	5
dst_IPv4	IPv4 for attacked device.	IPv4 range	random
dst_IPv6	IPv6 for attacked device.	IPv6 range	random
dst_port	Port on which device gets attacked.	[0, 65535]	80
dst_MAC	Machine address code of attacked device	MAC range	random
start_time	Time in unix time stamp.	Integer	current time
real_traffic_noise	Step wise increase of real traffic noise.	["none", "low", "medium", "high"] or [0, 1]	"low"

Table 3.1: List of configurable general parameters for the DDoS Log Simulator.

The lower part of the JSON file contains the attacker's configuration. The different attackers can be listed in an array and may have the following parameters shown in table 3.2. A generation has to have at least one attacker with one parameter defined to execute successfully.

During the creation process of the packet it is possible to add a certain payload to each packet. This payload is represented in form of the `bandwidth_Bps` parameter and is defined in bytes per second. By default no additional payload is added. This is also the case if the stated amount is smaller than the size that the packet would already be even without the payload. Furthermore it is possible to define an IP address for each attacker either in `IPv4` or `IPv6`. As in the destination IP the DDoS Log Simulator prefers the IPv6 if stated and creates an IPv4 address if both entries were left empty. Via `MAC` a media access control address can be configured. If left blank, the program creates an address randomly.

Each attacker has to be configured with an amount of attacking packets per second via `packets_per_sec`. Since the packets per second are defined in the attacker's section, it is possible to have a different amount of packets per seconds for each attacker. Therefore the total packets per second has to be added up over all attackers. The amount of packets per second per attacker is required to state since it also defines the amount of attackers when no other parameters are defined. The last parameter which can be adjusted is the attacker's outgoing port via `port`. By default port 80 is preconfigured. The parameters `IPv4`, `IPv6` and `MAC` have to be defined as character strings, `bandwidth_Bps`, `packets_per_sec` and `port` as integers.

DDoS Log Simulator - Attacker Config			
Parameter	Information	Valide range	Default Value
bandwidth_Bps	HTTP load from attacker in bit per second.	Integer	0
IPv4	IPv4 for attacking device.	IPv4 range	random
IPv6	IPv6 for attacking device.	IPv6 range	random
MAC	Machine address code of attacking device	MAC range	random
packets_per_sec	Packets per second from attacker.	Integer	-
port	Port on which device gets attacked.	[0, 65535]	80

Table 3.2: List of configurable attacker parameters for the DDoS Log Simulator.

3.3.2 Packet Generation

The following sections should provide the reader with a deeper insight into the main functionality of the DDoS Log Simulator by explaining parts of the source code. In this section the actual process of generating packets is described. For that, it contains parts of the `create_log()` function. For a better understanding note that variables, parameters and functions in the text are formatted in `teletype font`.

The generation process can distinguish between two different packet types. The upper part from line 1 to 19 handles the creation of real traffic noise while the second part from line 20 to 28 handles the creation of the actual attacking packets. The trigger for real traffic noise creation depends on the `noise_counter` variable. When this variable surpasses 100 percent, a noise packet will be generated. The `noise_counter` is incremented by `noise` after an attacking packet is created and corresponds to the percentage of real traffic, as specified in the configuration. Since the `noise_counter` variable is initialized randomly between zero and one, an even real noise distribution should be ensured from the very beginning. This initialization is not visible in the presented code.

When the program enters the noise packet generation at line 2, it first determines the version of the internet protocol that encapsulates the SYN packet. Every 10th random noise packet is an IPv6 based SYN request. After the decision process, Scapy creates either an IPv6 or otherwise an IPv4 network layer. In a second step Scapy creates the transport layer which consists out of the source and the destination port and sets the

SYN flag. Afterwards the actual packet gets forged by Scapy by merging the previously created parts and a randomly created MAC address. To finalize the process creation of the real noise packet, the timestamp for the packet is set explicitly. Not doing so would lead to errors in execution environments. At this point the packet is ready to be written into a temporary *pcap* file based on the `filenumber`. The `filenumber` is determined by the iteration of the generation process and which thread the process spawned. To finalize the writing of the noise packet, the `noise_counter` is decreased by one. This prevents an endless spawn of noise packets and will be increased only by created attacking packets. Now the thread sleeps for a specified amount of time to prevent creating too many packets. This is especially necessary when creating just a very few packets. Since the DDoS Log Simulator should provide a base to generate a wide variety of DDoS attacks, this sleeping time was implemented even though it affects performance negatively.

Regardless of whether a real noise packet was generated or not, the DDoS Log Simulator tries to write an attacking packet from line 19 to line 26. The process of generating the attacking packet differs from that of a noise packet in that the attacking packet was already created previously. Since an attacking packet just needs to be forged once, it offers an effective possibility to improve execution speeds. The only modification on the pre-forged packet is adjusting its timestamp to the specified or current time. Since the packet writing process is exception handled, the program would pass through the generated errors from Scapy in case of an invalid configuration. Those exceptions are in natural language and point out the mistake made in the configuration concisely.

```
1  # Generate and write real noise packet
2  if noise_counter >= 1:
3      if random.randrange(100) < 10:
4          network_layer = IPv6(src=random_IPv(6), dst=random_IPv(6))
5      else:
6          network_layer = IP(src=random_IPv(4), dst=random_IPv(4))
7      transport_layer_new = TCP(sport=src_port, dport=dst_port, flags="S")
8      packet_real_traffic = Ether(dst=rand_mac(),
           src=rand_mac())/network_layer/transport_layer_new
9      packet_real_traffic.time = time.time()
10     try:
11         wrpcap('flood_trace_tmp%d.pcap' % (filenumber), packet_real_traffic,
               append=True)
12     except Exception as e:
13         log.critical(f"[-] Something went terribly wrong: {e}")
14         sys.exit()
15     noise_counter -= 1
16     time.sleep(sleep_time)
17
18 # Write the attacking packet
19 try:
20     packet.time = time.time()
21     wrpcap('flood_trace_tmp%d.pcap' % (filenumber), packet, append=True)
22     time.sleep(sleep_time)
23 except Exception as e:
24     log.critical(f"[-] Something went terribly wrong: {e}")
```

```
    sys.exit()
noise_counter += noise
```

Listing 3.6: Part of the function which generates the actual packets.

3.3.3 Expandability

This section describes the packet spawning function and shows how it is possible to extend the DDoS Log Simulator in an easy manner. The displayed part of source code in listing 3.7 shows on one hand the critical part of the creation of the SYN flood packet (line 3), and on the other hand the creation of the DNS amplification attack (line 11). In the SYN flood packet crafting part, the program creates a TCP packet via `TCP()`, sets the source and destination port, and sets the SYN flag. The packet is stored in the `transport_layer` variable.

In the DNS amplification packet forging process, at first sample DNS server responses are loaded into `payloads` on line 14. Afterwards `Big_DNS_Responses` are selected and stored into `DNS_Responses`. A hash generated from the IPv4 and IPv6 address then determines the sample that is selected. Doing so ensures that each server has its own response. Since the JSON file format does not natively support binary data, the raw DNS responses are Base64-encoded before being added as the payload. In a final step, a UDP packet is forged with the destination port as well as the previously selected and encoded payload. Since both functions write into `transport_layer`, no changes are needed in the actual loop of writing packets, which will start soon after this part.

```
1  # Attack specific configurations here
2  # Syn Flood
3  if attack_type == "syn_flood":
4      transport_layer = TCP(sport=src_port, dport=dst_port, flags="S")
5      if bpp > 40:
6          payload = "s" * (bpp-40)
7      else:
8          payload = "s" * bpp
9
10 # DNS Amplification Attack
11 if attack_type == "dns_amplification":
12     try:
13         with open("./data/DNS_server_list.json") as payloadsFile:
14             payloads = json.load(payloadsFile)
15     except Exception as e:
16         log.critical(f"[!] Data directory is missing. Payloads could not be
               loaded..\n")
17     DNS_Responses = payloads.get("Big_DNS_Response")
18     payload = base64.b64decode(DNS_Responses[(hash(src_ip) + hash(src_IPv6)) %
           len(DNS_Responses)-1]["Sample"])
19     transport_layer = UDP(dport=dst_port)/Raw(load=payload)
20
```

```
# Next Attack type
# ...
```

Listing 3.7: Section of packet forging which allows easy extendability.

3.3.4 Multi Thread Support

The following code section in listing 3.8 describes the process spawning starting at line 18 based on the thread estimation starting at line 2. During the development process the thread estimation was redesigned several times. The whole challenge is based on the difficulty to predict the amount of packets a single packet generating process can create while keeping the program as fast as possible. When comparing different systems there were single threaded packet generations measured from 2 500 up to 8 500 packets per second. While the packet generation initially was too defensive, it was too aggressive in some revision. The final solution is a mixture between idling time (`sleep()`) in each packet generation as well as thread prediction.

When the program has parsed all inputs the actual packet generation can start. At first the program decides if the packet generation should be in bursts or handed over to the exact single threaded generation process. The burst execution starts when the total amount of packets from the current attacker is smaller than the already generated packets added up with the amount of packets at least expected to be generated by one process.

After each generation burst the program stores the average amount of generated packets per process in a variable called `packets_1process`. Since in the first iteration no previously estimated generation speed is available, the program spawns the same amount of processes as the system has threads. This approach does not create any packets too much since the sleeping time during a single packet creation is equivalent to the total amount of packets divided by the amount of threads a system has. In other words, spawning all treads could maximally result in the exact amount of packets demanded.

Presumed the program is already in its second loop and has a certain value set to `packets_1process`, the elif in line 7 is invoked. The elif checks if the new `coreCount` will be smaller than the first one and executes if so. This should prevent the specific scenario in which the program assumes too much program processes and since each program process then works even slower than expected it expands the amount of program processes even more. It is not possible to set the amount of maximal threads spawned to a certain value like the doubled amount of threads available since the sleeping time between each created packet gives the CPU a lot of room to handle other tasks. When the elif in line 7 is entered the amount of program processes is determined and saved into `coreCount` and if demanded a debug message is printed out. The last check done on line 12 is not to let the `coreCount` be zero since then the program would stuck in a loop.

In a final step the actual packet crafting processes are spawned in form of `create_log()` functions in line 19. All parsed arguments are passed to the current program process while always incrementing the current file number stored in the variable `fileNu`. The current file number indicates a process in which temporary file it should write into. Such an approach had to be implemented since letting multiple processes writing into a single file would result in a corrupt file. Finally the created process is appended to the list of processes and started afterwards. Since all of this is in a loop starting at line 15, the program keeps creating processes until the predicted amount of predicted threads beforehand is reached.

```
 1 # Multi Thread support
 2 while total_packets_per_attacker >
       logged_length_per_attacker+(pps_per_attacker*duration*1/os.cpu_count()):
 3
 4     if packets_1process == 0:
 5         log.debug(coreCount)
 6         core_count = os.cpu_count()
 7     elif coreCount > total_packets_per_attacker -
           logged_length_per_attacker)/packets_1process)-1:
 8         coreCount = round(((total_packets_per_attacker -
               logged_length_per_attacker)/packets_1process)-1)
 9         log.debug(coreCount)
10
11     if coreCount < 1:
12         coreCount = 1
13
14    for x in range(coreCount):
15         fileNu = fileNu + 1
16
17         p = Process(
18                 target=create_log,
19                 args=(attack_type, src_net4, src_net6, dst_IPv4, dst_IPv6,
                     dst_port, pps_per_attacker/os.cpu_count(), attackers,
                     duration, port, fileNu, bytes_per_packet, dst_MAC, src_MAC,
                     noise, isIPv6), daemon=True)
20         processes.append(p)
21         p.start()
```

Listing 3.8: Section which spawns the processes based on the thread estimation.

Chapter 4

Evaluation

This section is an attempt to quantify the functioning of the DDoS Log Simulator and afterwards discuss the received results. Such an approach to process and discuss real measurements provides the reader with further insight into how the program works.

In the first part, the setup for the different tests is presented. This includes the configuration file used for the different tests as well as a description of the systems the DDoS Log Simulator was executed on. In the second part, the performance measurements are explained in detail. Plots are included for the reader to get a clear picture of how performance is impacted by different factors. After commenting about performance, the next section details the verification of the results generated with the specified configuration file. The verification process was conducted by using the fingerprint generator from DDoS Clearing House. Their fingerprint generator is used for quantifying real DDoS attacks, based on the contents of a network traffic log file. It provides an excellent opportunity to compare the generated fingerprint with its configuration file, mainly due to the fact that its generated fingerprints are very similar to the DDoS Log Generator's configuration file.

4.1 Initial Setup

To have comparable test results, it is crucial to have the same configurations running on all machines. This section describes at first the JSON configuration file and its different parameters. Afterwards the two platforms, on which the tests were executed, are presented. The JSON file, as seen in the listing below, represents a typical SYN flood attack and all possible parameters are set except for the IPv4 address. The program only takes into consideration either the IPv4 or IPv6 address. In this case the IPv6 address is specified, so any IPv4 address may be omitted.

The configured attack has the characteristics of a typical SYN flood attack. It is inspired by SYN flood attacks observed in Clearing House's DDoSDB. The start time is in the Unix timestamp format (an integer representing the number of seconds elapsed since midnight on Thursday, 1 January 1970) and converts to the ISO 8601 representation 2019-05-09T13:29:01. The attack has a specified duration of 20 seconds and a medium amount of real traffic noise. Since a medium level of noise corresponds to 0.08%, there is a total of 240 000 noise packets in the generated attack.

An IPv6 address from the 2001:0db8::/32 subnet was used because they are meant for documentation and demonstration purposes (see RFC 3849 from IETF). This prevents confusion originating from the generated log file. The MAC address is randomly chosen, so it could also be left blank. Port 80 (HTTP) is chosen as the destination port. Besides port 443 (HTTPS), port 80 is commonly seen as it is the port used by web servers to serve content to clients. For this purpose, it fits nicely into the SYN flood attack scenario.

The attack itself originates from two attackers. Both have an IPv6 address from the 2001:0db8::/32 subnet and a random MAC address. The ports of the attackers are 81 and 82, which provide a better overview when looking at the final generated logs. The bandwidth is set to 25 000 Bytes per second on both of the attackers. Furthermore, the configuration states an attack rate of 100 000 packets per second for the first attacker and 50 000 packets per second for the second one. This results in 150 000 packets per second and 3 000 000 attacking packets in total. In addition to the real noise traffic, the generator has to create around 3 240 000 packets. The tests were executed using the configuration file as presented in Listing 4.1.

```json
{
  "attack_type": "syn_flood",
  "start_time": 1557408541,
  "duration_sec": 5,
  "real_traffic_noise": "medium",
  "dst_IPv6": "2001:0db8:0a0b:12f0:0000:0000:0000:0001",
  "dst_port": 80,
  "dst_MAC": "8c:7f:41:b0:5d:99",
  "attackers": [
    {
      "IPv6": "2001:0db8:85a3:0000:0000:8a2e:0370:7334",
      "MAC": "00:0a:95:9d:68:16",
      "port": 81,
      "bandwidth_Bps": 2500,
      "packets_per_sec": 10000
    },
    {
      "IPv6": "2001:0db8:85a3:0000:0000:8a2e:0370:7335",
      "MAC": "40:22:12:8d:32:1a",
      "port": 82,
      "bandwidth_Bps": 2500,
      "packets_per_sec": 50000
    }
  ]
}
```

Listing 4.1: JSON configuration file based on which the whole performance analysis is based on.

To provide a better comparison on how the DDoS Log Simulator behaves, two different systems were used for benchmarking. The first device is a mobile notebook computer with a mid-range 15 watts TDP Intel i5-8250U processor. This CPU has a base clock speed of 1.6 GHz and a boost clock speed up to 3.4 GHz. Due to the fairly weak thermal design a notebook is equipped with, the boost clock speed is only reached in the initial seconds of the programs execution and the CPU is throttled afterwards. The notebook has 8 GB DDR4 RAM as well as a fast 256 NVMe SSD storage device. As an operating system Ubuntu Linux is installed and Python 3.7.3 is used to execute the DDoS Log Simulator.

The second device is a desktop computer with an AMD Ryzen 7 1700 processor. The CPU has 8 cores with 16 threads and a base clock of 3GHz. Due to great cooling the expected core boost frequencies are 2x3.7GHz and 6x3.1GHz without any throttling during the execution. The computer is equipped with 48 GB DDR4 RAM and also a NVMe storage device. Its operating system is CentOS Linux and it also runs Python 3.7.3 to execute the DDoS Log Simulator.

1. Platform	**Mobile Notebook**
Operation System	Ubuntu Linux
Processor	i5-8250U @ 1.60GHz (4/8)
Memory	8 GB DDR4
Storage Device	256 GB NVMe

2. Platform	**Desktop**
Operation System	CentOS Linux
Processor	AMD Ryzen 7 1700 @ 3GHz (8/16)
Memory	48 GB DDR4
Storage Device	256 GB NVMe

4.2 Performance Analysis

When measuring the execution time from the previously presented configuration file with the Linux `time` command, it reports a time from start to finish of 2m31.314s on the server and 4m51.666s on the notebook. Even though the desktop system is at least twice as fast as the notebook, this does not manifest itself in the execution times entirely. A reason for this behaviour can be found in the fact that the packet counting process is single threaded only and also needs a certain amount of time to finish.

In Figure 4.1 the amount of spawned threads from the DDoS Log Simulator are compared in a graph. The red line indicates the desktop system while the blue line stands for the mobile device. At first the DDoS Log Simulator spawns as much processes as if it could generate its load in the very first iteration. This is nicely visible by the fact that the desktop system only uses 16 processes at the beginning which is equivalent to the amount of threads the system has. On the other hand the mobile device spawns 8 threads which is also equivalent to its amount of threads. When the DDoS Log Simulator realizes the systems are not capable of generating all packages in the first iteration it starts to spawn more processes.

This approach can be considered as efficient since the program sets a sleeping time between each generated packet such that the CPU can process other processes in the meantime. The amount of spawned process corresponds to the average amount of packets the processor could generate in the previous generation round divided by the amount of total amount of packets. The estimation process had to be implemented since computer processing power differs greatly from system to system. The exact process is further explained in the second part of Chapter 3.

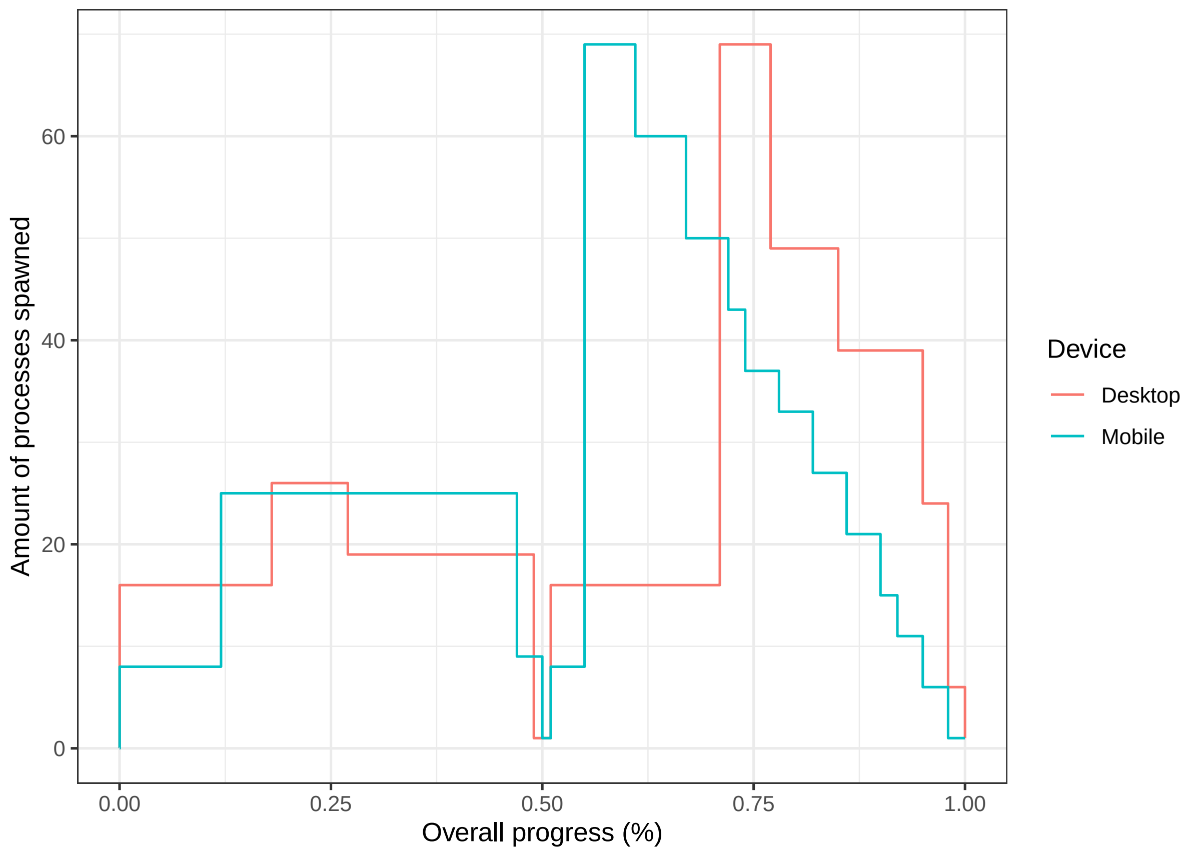

Figure 4.1: An overview how many threads the DDoS Log Simulator allocates during the generating process.

When the DDoS Log Simulator expects the generation to finish, it starts decreasing the number of processes towards the end of the current attackers packet generation. Since there are two attackers defined in the configuration file, the whole process is repeated once again. It is also visible that the DDoS Log Simulator did not spawn as much process in the desktop packet generation as it does in the mobile system. This is due to the fact that the DDoS Log Simulator expects faster packet generation per process in the desktop than in the notebook.

In the second half both systems spawn at first approximately the same amount of processes.

In figure 4.2 there is a graph which shows the amount of used memory during the execution. Since the graph had the same shape and dimensions on both tested systems, only one is shown in the graphic for an easier overview. Note that the program initially allocated 95 megabytes and hence this is where the graph starts.

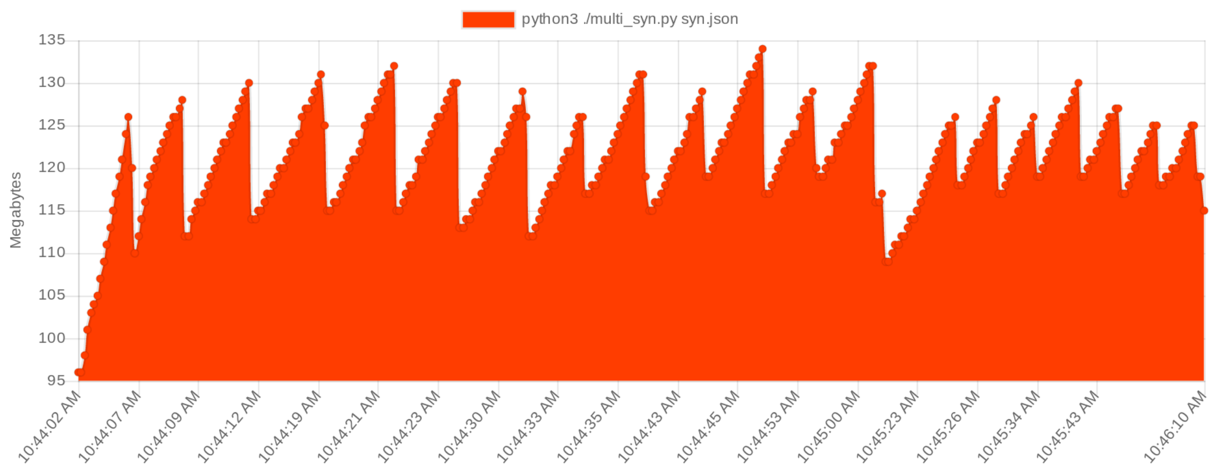

Figure 4.2: An overview of the amount of memory the DDoS Log Simulator allocates during the generation process using the configuration presented in Listing 4.1.

Each peak represents a packet generation burst and each drop the merging process afterwards. In average the process uses about 120 megabytes of memory nearly uninfluenced from the total amount of packets which had to be generated for each attacking instance. In two thirds of the graph there is a double drop to 100 megabytes which indicates the start of the generation from the second attacker. By always merging small chunks of generated log data, the DDoS Log Simulator is capable of keeping its used memory to a minimum.

The figure 4.3 represents the memory consumption for a very dense log file where the generation time took more than 30 minutes. Even tough a small increase of the allocated memory is visible during the whole generation process, the average amount of allocated memory stays at a reasonable level of about 125 megabytes in total. The visible memory increase, after the initial drop, until the end is about 40 megabytes. Which is compared to modern computer systems a very reasonable amount.

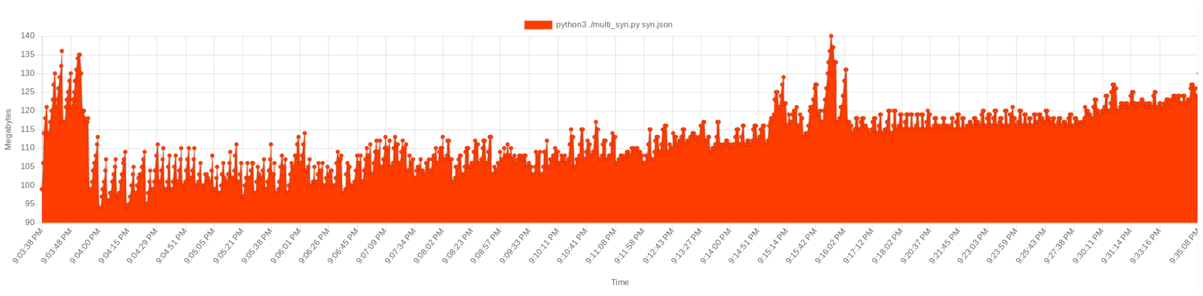

Figure 4.3: An overview of the amount of memory the DDoS Log Simulator allocates during a long generation time of over 30 minuts.

4.3 Verification

To verify the results produced by the DDoS Log Simulator, the initial configuration JSON is compared to its corresponding fingerprints generated by the DDoS Dissector from the final output *pcap* file. The DDoS Dissector is part of the DDoS Clearing House and creates a JSON file with a summary of the network trace characteristics. During the generation process, the Fingerprint generator filters and anonymizes the input file such that only the attacking instances remain. In other words, all random traffic noise will be filtered out. In this section, all parameters created in the fingerprint JSON (see listing 4.2) are compared to the original configuration file in listing 4.1. A desired result from this comparison would be to have two files matching as much as possible. This would indicate a very accurate output of what was configured in the JSON file.

4.3.1 SYN Flood Attack

Since the Fingerprint generator is capable of analyzing different kinds of file types (*e.g.*, *pcap*, pcapng, nfdump, netflow, and ipfix), it firstly states the kind of file produced in `file_type`. As intended by the DDoS Log Simulator the reported file type is in the *pcap* format. The protocol of the attack is TCP which stands for Transmission Control Protocol and is part of the transport layer. Since the vulnerability exploited in the SYN flood attack is also part of TCP Three-Way-Handshake the stated protocol appers to be correct. The next extracted information from the generated log file are the flags set in the attack. Since Python can't write some unicode character the string `"\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7S\u00b7"` looks encoded like this `"··········S·"` which represents the SYN flag which was set during the packet creation.

The fingerprint generator also recognizes the two defined attacking IPv6 correctly in `src_ips`. Even though the IPv4 addresses were stated as well in the configuration file in listing 4.1 the DDoS Log Simulator didn't take them in account due to it's policy to prefer IPv6 over IPv4. Also the amount of attacking IP's in `total_src_ips` and their ports in `src_ports` were detected properly from the DDoS Log Simulator. The MAC addresses stated in the configuration file aren't listed in the generated fingerprint. This is may due to the fact, that in a real life scenario the MAC address corresponds to an in-house device anyways which makes stating them in the fingerprint superfluous.

```
{
  "file_type": "pcap",
  "protocol": "TCP",
  "additional":
    {
      "tcp_flag":
        "\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7S\u00b7"
    },
  "src_ips": [
    "2001:db8:85a3::8a2e:370:7335",
    "2001:db8:85a3::8a2e:370:7334"
  ],
  "total_src_ips": 2,
  "src_ports": [82, 81],
  "total_src_ports": 2,
  "dst_ports": [80],
  "total_dst_ports": 1,
  "start_timestamp": 1557408541.000279,
  "key": "41f2173dabe30f52d99015e4bdc610db",
  "start_time": "2019-05-09 15:29:01",
  "duration_sec": 5.004808187484741,
  "total_packets": 20002,
  "avg_pps": 3996.5567611597867,
  "avg_bps": 6129252.680793878,
  "vector_filter": "
    (['_ws.col.Protocol']=='TCP')&
    (['dstport']==80)&
    (['tcp.flags.str']==
      '\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7\u00b7S\u00b7')",
    "multivector_key": "41f2173dabe30f52d99015e4bdc610db"
  }
}
```

Listing 4.2: Fingerprint generated from DDoS Dissector based on the JSON configuration in Listing 4.1

What is stated in the fingerprint is the starting time in the Unix time format and additionally in ISO 8601. The time the execution started in `start_time` is recognized in the fingerprint and compared to the one specified in the configuration file 0.000279 seconds off. Those 0.000279 may be in the margin of error while the program is calculating the duration between starting time and the actual time. The duration in the fingerprint is stated in seconds. While the desired amount specified in the configuration file were 5 seconds, the fingerprint generator reported a duration of 5.004808187484741 seconds. Those 0.004808187484741 seconds too much then impacted the average packets per seconds significantly. In total the DDoS Log Simulator had to create 20 000 packets. The fingerprint generator reported an amount of 20 002 attacking packets created. Since the duration of the whole attack was measured 0.00480818748474 seconds longer than it should be, the average packets per second, defined in `avg_pps` corresponds to 3996.556761. One could infer from that, the DDoS Log Simulator to have created 18 packets (3.5 multiplied by duration of 5 seconds) too little even tough it generated 2 packets too much.

The average bytes per seconds are 6 129 252 and stored in `avg_bps`. Defined in the configuration in listing 4.2 were 100 000 from the first attacker and 500 000 from the second attacker. Therefore the DDoS Log Generator has created 129 252 bytes per second to much. This are 32 bytes per packet too much which can be considered a rather large excess. In relation to the desired amount of bytes per minute it is 1.0215 the demanded quantity. Hence the bytes per minute creation is 2.15% off.

4.3.2 DNS Amplification Attack

This section compares the implemented DNS amplification attack's output with its corresponding fingerprint generated by the DDoS Dissector. While in the sections above referred to the configuration file in 4.1, this section will discuss the results from the DNS amplification attack generated by the configuration file provided in listing 4.4. While the configuration above represented a scenario not possible in the real world due to its chosen IPv6 addresses, the DNS amplification attack log in this sections aims to match a real world attack as nicely as possible.

For example the DNS requests are usually sent to the name server via UDP port 53 and therefore the destination port in `dst_port` is set to 53. Also the MAC addresses on all attackers are matching since in a real life attack those MAC addresses represent the local packet entry point. Further more the attackers sending port is set to a big number since the lower numbers are reserved by the system. One characteristics which does not match a real world scenario is the duration of only 10 seconds. Real DNS amplification attacks can have a length of several days or even weeks.

```json
{
  "attack_type": "dns_amplification",
  "duration_sec": 10,
  "dst_port": 53,
  "attackers": [
    {
      "MAC": "00:0a:95:9d:68:16",
      "port": 46068,
      "packets_per_sec": 2000
    },
    {
      "MAC": "00:0a:95:9d:68:16",
      "port": 41401,
      "packets_per_sec": 2500
    },
    {
      "MAC": "00:0a:95:9d:68:16",
      "port": 46211,
      "packets_per_sec": 1500
    }

  ]
}
```

Listing 4.3: Configuration of a DNS Amplification attack trying to be as close to a real world scenario as possible.

While generating the fingerprint of the attack configured in listing 4.4 the fingerprint generator from DDoS Dissector often had trouble determining the attacker. This resulted in the fact that it often stated one attacker too little and could be due to the fact that the DNS responses were extracted from real log data and often stated by Wireshark as too big to be fully captured. Possibly those records were in some way corrupt and the fingerprint generator could not determine it correctly. Strangely this happened not in all generations.

Therefore also the total / average amount of packets per seconds do not match with what was stated in the configuration file even thought the file itself contained the right amount of packets. Considering the missing of one attackers, the fingerprint shows the same divergences as the previously generated.

```

{
"file_type": "pcap",
"protocol": "DNS",
"additional": {"dns_query": "sema.cz", "dns_type": 255.0},
"src_ips": ["164.63.16.134", "150.51.199.215"],
"total_src_ips": 2,
"src_ports": [41401, 46068],
"total_src_ports": 2,
"dst_ports": [53],
"total_dst_ports": 1,
"start_timestamp": 1566412233.4858382,
"key": "eaf0f734f08b6e7be331ab53a5ad86d2",
"start_time": "2019-08-21 20:30:33",
"duration_sec": 10.027663946151733,
"total_packets": 45003,
"avg_pps": 4487.884739822237,
"avg_bps": 10019842.162596507,
"vector_filter":
    "(['_ws.col.Protocol']=='DNS')&(['dstport']==53)&(['dns.qry.name']=='sema.cz')",
    "
multivector_key": "eaf0f734f08b6e7be331ab53a5ad86d2"
}
```

Listing 4.4: Configuration of a DNS Amplification attack trying to be as close to a real world scenario as possible.

4.4 Discussion and Limitations

This section critically analyses and discusses some of the characteristics of the DDoS Log Simulator regarding its performance, its expandability as well as its implemented parameters. Also the verification process is discussed in detail.

The DDoS Log Simulator can generate arbitrarily long and dense log files. A downside of this approach is the relatively slow execution time, which is at least as long as the duration of the attack itself multiplied with the amount of attackers defined in the configuration file. This design decision was made since the logs are not meant to be generated in real-time as well as for reliability purposes. Since there is always a certain sleeping time between each single packet creation, the system on which the generation happens keeps responsive unless the packet density gets too high. A workaround for this problem can be found with the program nice which can adjust process priorities under Linux.

Regarding the execution time, there is still some room for improvement available. Especially in the way the program counts the already created packets. Right now the total package counting is done by opening the created files after a creation burst. Efforts were made to find another solution to speed up said process but there was no reliable solution found for inter-multi-process communication. Fortunately the packet counting is not the main time consummating part of the whole generation process.

Much more frequent the program iterates in the packet generation process. The packet generation process depends on a correct thread estimation and the attempt has been made to let the amount of spawned process drop just in the very last moment. The Listing 4.5 shows a pre-final but very mature version of the DDoS Log Simulator estimating the wrong amount of processes. Even tough the estimation process was revised several time sometimes the DDoS Log Simulator decreases the amount of spawned program processes not strongly enough. As visible, the output reports the amount of packets generated in percentage and how many program processes were spawned. The last two lines indicated the switch into the single threaded final exact packet generation. Unfortunately the DDoS Log Simulator had already created 165 620 packets out of 162 000 during the burst generation. Those 5 620 packets too much corresponds to either one or two processes too much.

```
Current Amount of processes spawned: 8
[~] Amount of packets generated: 58%
[~] Amount of packets generated: 59%
[~] Amount of packets generated: 60%
[~] Amount of packets generated: 61%
[~] Amount of packets generated: 61%
[~] Amount of packets generated: 62%
[~] Amount of packets generated: 63%
[~] Amount of packets generated: 64%
Current Amount of processes spawned: 5
[~] Amount of packets generated: 64%
[~] Amount of packets generated: 65%
[~] Amount of packets generated: 66%
[~] Amount of packets generated: 67%
[~] Amount of packets generated: 67%
162000.0
165620
```

Listing 4.5: Extract of a log showing the DDoS Log Simulator not decreasing the amount of spawned processes and over-creating packets in a non-final version.

The amount of different configurable parameters should provide a general way to determine all fundamental characteristics of DDoS attacks. They are influenced by the different parameters stated in the DDoSDB attack fingerprints. This approach allows a maximum compatibility to a wide variety of different DDoS attacks.

Unfortunately, not each defined parameter makes as much sense to be stated in different attack types. For example the bandwidth defined in the configuration file as `bandwidth_Bsp` may be considered as superfluous in the SYN flood attack scenario since it adds unnecessary payload to each sent packet which would never be seen in a real world attacks. But since it is possible to define the amount of bytes per seconds a way need be defined for the DDoS Log Simulator to handle it as well as possible. Also in the DNS amplification attack the bytes per second caused trouble while implementing the attack. Since the decision was made to add real big DNS responses to the payload the DDoS Log Simulator ignores the defined value in the configuration file completely. A better but much more complex approach would have been to implement a function which returns a DNS response dependent on the `bandwidth_Bsp` variable.

Another problem regarding the different parameters which can be set in the JSON configuration file concerns the real noise traffic. Right now the real noise traffic represents random either IPv4 or IPv6 addresses sending a SYN packet to the configured destination IP address. This implementation was rather simple and there is much capability for improvement available. For example real traffic noise could consist out of different SYN, SYN-ACK, ACK requests or even HTTP GET, POST, PUT or DELETE and so on. Another important feature of real noise which is not implemented yet would be the servers response. Right now the DDoS Log Generator does not create any response coming from the server side. The decision no to implement a sophisticated answer was due to the fact, that the responding server should have been further defined *e.g.* as a webserver, FTP-Server, DNS-Server or even a backend-server from a specific application.

What can be considered as a clever approach is the verification process of comparing the configuration file with the output generated by the DDoS Dissector's fingerprint generator. With this verification method even smallest divergences between the actual log and what was configured are shown. What is missing in this kind of verification process is the analysis of the nature of the attack itself. There is no parameter stating the accurateness of an attack or if the generated log file even could have been captured in real life.

Chapter 5

Summary and Conclusion

DDoS attacks are a major concern for the network community. They cause expensive disruptions of internet services and are an important threat to defend against. To study them, it is vital to have numerous logs of recorded attacks. It is impractical for web infrastructure companies to regularly publish such logs because of difficulties to anonymize this data. Therefore it would be convenient to have a tool that could simulate such attacks with a close resemblance to real attacks.

This paper introduces the DDoS Log Simulator that creates logs of DDoS attacks in the form of *pcap* files based on a configuration file. It supports the implementation of different attack types. Implemented in the program are the SYN flood attack and the DNS amplification attack. A variety of parameters including the attack duration, list of attackers with respective bandwidth, and ratio of real traffic noise can be adjusted.

Due to its design choices, the DDoS Log Simulator is capable of generating arbitrarily long and dense log files. It uses multiple threads depending on the number of CPU cores available. The performance was measured on two different devices, a mobile notebook and desktop computer.It was concluded that the DDoS Log Simulator is capable of efficiently utilizing multiple threads in parallel.

The DDoS Dissector was used as a tool for dissecting network traffic from *pcap* files and identifying and generating DDoS attack fingerprints, to evaluate the log files generated by our DDoS Log Simulator. It was able to detect the typical characteristics of the two implemented attacks, namely SYN flood and DNS amplification attacks. It is possible to extend it with other attacks.

Generated logs can afterwards be used in different upstream applications. These include for example teaching, visualization or penetration testing purposes, since it is capable of creating logs of attacks which in practice could not have happened. An example is the visualization of a log file, which can be used for teaching or illustrative purposes. This may help with better exploring and understanding DDoS attacks by simplifying them. Other future work may be found in the penetration testing field. An interesting application would be the conversion from the logs to real attacks. With such an approach real infrastructure could be tested for specific attack scenarios.

5.1 Future Work

The DDoS Log Simulator is designed as a platform to generate DDoS log files. It was built with the intention to be as easy to modify as possible and therefore holds a lot of potential for further expansions. This may includes: (i) Support for other types of DDoS attacks, (ii) sending the generated attacks on wire, (iii) the creation of visual output, and (iv) to train artificial intelligence based on the generated log files. Those possible future works may be used as support for security teams (*e.g.,* universities, police departments, and governments) and to conduct researches in direction to develop and test novel solutions for cybersecurity.

An obvious and valuable extension of the program would be the creation of more attacks. So far two attacks were implemented as a proof of concept. The SYN flood attack and the DNS amplification attack demonstrate the DDoS Log Simulator's capabilities. After having created a new attack, the corresponding configuration file may be shared for others to reproduce the same type of attack with customized parameters.

Another interesting application scenario could be found in the generation of real attacks from log data. This would allow the user to test their created log files on wire. Right now there is a tool called tcp replay which is capable of sending pcap files out of an interface. Since tcpreplay is free Open Source software a connection between the two program could be possible. Due to the capability of the DDoS Log Simulator to create preconfigured log files the program would be well suited as a content creator tool. Those content could be used by a variety of different applications. For example for visualization purposes. For example some functionalities from CapAnalysis could be implemented into the DDoS Log Simulator. CapAnalysis is a tool to present .pcap files in many forms. It is Open Source and released under the GNU General Public License version 2 and therefore should provide good preconditions to implement code into the DDoS Log Simulator.

Also, further solutions can be developed to use the log files generated by this thesis in order to train artificial intelligence for network operators to identify behaviors of DDoS attacks in an efficient way. Thus, this can help to improve the cybersecurity by having another way to detect attacks.

Bibliography

[1] Esraa Alomari, Selvakumar Manickam, BB Gupta, Shankar Karuppayah, and Rafeef Alfaris. Botnet-based distributed denial of service (DDoS) attacks on web servers: classification and art. *arXiv preprint arXiv:1208.0403*, 2012.

[2] Cloudflare. Famous DDoS Attacks | The Largest DDoS Attacks Of All Time. `https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/`. Accessed: 2019-08-18.

[3] Christos Douligeris and Aikaterini Mitrokotsa. DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art. *Computer Networks*, 44(5):643–666, 2004.

[4] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical approaches to DDoS attack detection and response. In *DARPA information survivability conference and exposition*, volume 1, pages 303–314, Anaheim, USA, 2003.

[5] Piyush Goyal and Anurag Goyal. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In *9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 77–81, Grine, Cyprus, 2017.

[6] Jiahui Jiao, Benjun Ye, Yue Zhao, Rebecca J Stones, Gang Wang, Xiaoguang Liu, Shaoyan Wang, and Guangjun Xie. Detecting tcp-based ddos attacks in baidu cloud computing data centers. In *IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 256–258, Hong Kong, China, 2017.

[7] Andrzej Kosowski and Volodymyr Mosorov. Nessi2 simulator for large-scale DDoS attack analysis. In *Perspective Technologies and Methods in MEMS Design*, pages 157–159, Polyana, Ukraine, 2011.

[8] Z Morley Mao, Vyas Sekar, Oliver Spatscheck, Jacobus Van Der Merwe, and Rangarajan Vasudevan. Analyzing large DDoS attacks using multiple data sources. In *SIGCOMM workshop on Large-scale attack defense*, pages 161–168. ACM, 2006.

[9] NZZ am Sonntag Marco Metzler. Kriminelle erpressen Schweizer Online-Shops. `https://www.nzz.ch/nzzas/nzz-am-sonntag/cyber-attacken-kriminelle-erpressen-schweizer-online-shops-ld.9083/`, 2016. Accessed: 2019-08-18.

[10] Jose Nazario. Ddos attack evolution. *Network Security*, 2008(7):7–10, 2008.

[11] K Nishizuka, L Xia, J Xia, D Zhang, L Fang, and C Gray. Interorganization Cooperative DDoS Protection Mechanism. *Internet-Draft, Draft*, 2016.

[12] Scapy Project. Scapy - Packet Crafting for Python2 and Python3, 2019. https://scapy.net/, last visit August, 2019.

[13] Nidal Qwasmi, Fayyaz Ahmed, and Ramiro Liscano. Simulation of DDoS attacks on P2P networks. In *IEEE International Conference on High Performance Computing and Communications*, pages 610–614, Alberta, Canada, 2011.

[14] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. Multi-domain ddos mitigation based on blockchains. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 185–190, Zurich, Switzerland, 2017.

[15] Jan Stanek and Lukas Kencl. SIPp-DD: SIP DDoS Flood-attack Simulation Tool. In *20th International Conference on Computer Communications and Networks (ICCCN 2011)*, pages 1–7, Hawaii, USA, 2011.

[16] Madiha H Syed, Eduardo B Fernandez, and Julio Moreno. A misuse Pattern for DDoS in the IoT. In *23rd European Conference on Pattern Languages of Programs*, page 34, Bavaria, Germany, 2018.

[17] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. Dnssec and its potential for ddos attacks: a comprehensive measurement study. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 449–460. ACM, 2014.

[18] Haining Wang, Danlu Zhang, and Kang G Shin. Detecting SYN flooding attacks. In *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1530–1539, New York, USA, 2002.

[19] Wireshark. About Wireshark. `https://www.wireshark.org/about.html`. Accessed: 2019-06-08.

Abbreviations

ACK	Acknowledgement
AI	Artificial Intelligence
CPU	Central Processing Unit
DoS	Denial-of-Service attack
DDR	Double Data Rate
DDoS	Distributed-Denial-of-Service attack
DNS	Domain Name System
FTP	File Transfer Protocol
GB	Gigabyte
GHz	Gigahertz
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
MAC	Media access control
MTU	Maximum Transmission Unit
NVMe	Non-Volatile Memory Express
OS	Operating System
P2P	Peer-to-Peer
PCAP	Packet capture
PCAP-NG	PCAP Next Generation
RAM	Random-Access Memory
SSD	Solid-state drive
SIP	Session Initiation Protocol
SYN	Synchronize
TCP	Transmission Control Protocol
TDP	Thermal Design Power
UDP	User Datagram Protocol

List of Figures

List of Tables

Appendix A

Installation Guidelines

1. Install Wireshark (from `https://www.wireshark.org/`)
   to ensure that tools `editcap` and `mergecap` are available in PATH

2. Install Python 3.7 (from `https://www.python.org/`)

3. Confirm correct Python version:
   `python --version`

4. Switch to DDoS Log Simulator's folder

5. Create virtual environment (recommended):
   `python -m venv venv` (in folder `venv`)

6. Activate virtual environment (if applicable):
   `source venv/bin/activate`

7. Install requirements:
   `pip install -r requirements.txt`

The installations in step 1 to 3 may be performed via the packet manager of the distribution used.

A.1 Usage

- Show DDoS Log Simulator's help message:
  `python log_gen.py --help`

- Run DDoS Log Simulator with configuration file `syn.json`:
  `python log_gen.py syn.json`

- Run DDoS Log Simulator with configuration file `syn.json` in verbose mode:
  `python log_gen.py syn.json --verbose`

- Merge two files named `file1` and `file2` to a file named `output`:
  ```
  python log_gen.py -m file1 file2 outputfile
  ```
- Show DDoS Log Simulator's version:
  ```
  python log_gen.py --version
  ```

Appendix B

Contents of the CD

DDoS_log_sim
DDoS Log Simulator source code

Bachelorarbeit.pdf
Bachelor Thesis (PDF)

Zsfsg.txt
Abstract German (TXT)

Abstract.txt
Abstract English (TXT)