BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DDOS ATTACK DETECTION USING NETFLOW CACHE ANALYSIS

DETEKCE DDOS ÚTOKU POMOCÍ ANALÝZY NETFLOW CACHE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JAROSLAV STREIT**
AUTOR PRÁCE

SUPERVISOR **Ing. MATĚJ GRÉGR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

165025

Institut:	Department of Information Systems (DIFS)
Student:	**Streit Jaroslav**
Programme:	Information Technology
Title:	**DDoS attack detection using NetFlow cache analysis**
Category:	Networking
Academic year:	2024/25

Assignment:

1. Get familiar with DDoS attacks and NetFlow technology for monitoring network traffic.
2. Explore the ipt-netflow tool for exporting NetFlow data.
3. Design a system that can search the NetFlow cache when an attack is detected and create an effective filtering rule that describes the pattern of the attack.
4. Implement and deploy the proposed system in a BUT test environment.
5. Evaluate the achieved results.

Literature:

- Claise B., "Cisco Systems NetFlow Services Export Version 9", RFC 3954, DOI 10.17487/RFC3954, October 2004, https://www.rfc-editor.org/info/rfc3954
- Hofstede R. et al., "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," in IEEE Communications Surveys & Tutorials, vol. 16, no. 4, pp. 2037-2064, Fourthquarter 2014, doi: 10.1109/COMST.2014.2321898.
- Mirkovic J., and Reiher P. "A taxonomy of DDoS attack and DDoS defense mechanisms." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 39-53.
- Zargar S. T., Joshi J. and Tipper D. "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks." *IEEE communications surveys & tutorials* 15.4 (2013): 2046-2069.

Requirements for the semestral defence:
1., 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Grégr Matěj, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	22.10.2024

Abstract

This Bachelor's thesis explores the detection and mitigation of Distributed Denial of Service (DDoS) attacks by analyzing NetFlow cache data. The work begins with a thorough investigation of the NetFlow protocol and its components, as well as different protocol version specifics. The thesis extensively overviews DDoS attack types, methods, classifications, and defense strategies. The main contribution is a Python-based tool that analyzes NetFlow cache data during ongoing DDoS attacks in the Brno University of Technology network. By bypassing export timeouts and analyzing the flow cache directly, the tool can achieve faster and potentially more accurate identification of attack patterns. It generates filtering rules that can be immediately deployed for mitigation. The implementation was integrated into the DDoS-Guard system, and its effectiveness was tested in real scenarios, demonstrating promising results in live deployments.

Abstrakt

Tato bakalářská práce se zabývá detekcí a mitigací DDoS útoků pomocí analýzy NetFlow cache dat. Nejprve je podrobně popsána technologie NetFlow a její klíčové komponenty a specifika různých verzí protokolu NetFlow. Práce dále poskytuje rozsáhlý přehled o typech, metodách a klasifikacích DDoS útoků a současně rozebírá existující obranné mechanismy. Hlavním přínosem práce je vývoj nástroje v jazyce Python, který analyzuje data zachycená v NetFlow cache během probíhajícího DDoS útoku v síti Vysokého učení technického v Brně. Díky přímému přístupu k datům z aktivního provozu, bez zpoždění způsobeného časovými limity pro export, dochází k rychlejší a potenciálně přesnější identifikaci útoků. Výstupem nástroje je filtrační pravidlo, které lze okamžitě použít pro mitigaci útoku. Nástroj byl integrován do systému DDoS-Guard a testován na reálných scénářích s pozitivními výsledky.

Keywords

NetFlow, NetFlow cache, ipt-netflow exporter, DDoS attacks, DDoS attack detection, DDoS-Guard, network flow aggregation

Klíčová slova

NetFlow, NetFlow cache, exportér ipt-netflow, DDoS útoky, detekce DDoS útoku, DDoS-Guard, agregace síťových toků

Reference

STREIT, Jaroslav. *DDoS attack detection using NetFlow cache analysis.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Matěj Grégr, Ph.D.

Rozšířený abstrakt

Tato bakalářská práce se věnuje detekci a mitigaci síťových útoků typu DDoS (Distributed Denial of Service) pomocí analýzy NetFlow cache, konkrétně v prostředí páteřní sítě Vysokého učení technického v Brně. Hlavním cílem práce je navrhnout a implementovat nástroj, který je schopen analyzovat síťový provoz zachycený v cache exportéru NetFlow v reálném čase, identifikovat útok na základě vzorů v toku dat a vygenerovat odpovídající filtrační pravidlo pro okamžité nasazení.

Úvodní část práce se zaměřuje na problematiku monitorování síťového provozu, přičemž detailně rozebírá technologii NetFlow — její architekturu, jednotlivé komponenty (exportér, kolektor, protokol), historický vývoj a praktické části implementace. Zvláštní důraz je kladen na open-source projekt ipt-netflow, který funguje jako NetFlow exportér na Linuxových systémech a je přímo použit v testovaném prostředí. Součástí této části je také popis struktury a funkcionality NetFlow cache, ze které jsou v reálném čase získávány data pro detekci útoků.

Následující část práce je věnována rozsáhlému rozboru DDoS útoků. Ty jsou klasifikovány podle různých kritérií — způsob útoku (logický vs. zahlcující), úroveň automatizace, platnost zdrojové IP adresy, cílová vrstva systému apod. Popsány jsou nejčastější typy zahlcujících útoků (např. UDP flood, TCP SYN flood, DNS flood) a jejich specifika. Dále jsou analyzovány současné trendy vývoje DDoS útoků na základě veřejně dostupných dat od společnosti Cloudflare. Jsou diskutovány i různé obranné mechanismy – jak z hlediska jejich rozmístění v síti (zdrojové, cílové, síťové, hybridní), tak i podle doby nasazení (prevence, detekce, reakce).

Stěžejní část práce se zabývá návrhem a implementací detekčního algoritmu, který analyzuje snímky NetFlow cache generované systémem DDoS-Guard v okamžiku detekce podezřelého provozu. Algoritmus je realizován v jazyce Python s využitím knihovny Pandas pro zpracování strukturovaných dat. Postupně agreguje záznamy z cache podle různých atributů (např. IP adresy, porty, protokol, TCP flagy), hierarchicky zužuje množinu relevantních dat a vyhodnocuje dominanci jednotlivých agregátů. Klíčovým prvkem je výpočet přizpůsobené prahové hodnoty (cutoff), která slouží k identifikaci toků odpovědných za nárůst provozu. Výsledkem analýzy je přesně definované pravidlo, které lze okamžitě využít k filtrování škodlivého provozu.

Implementovaný nástroj byl následně integrován do systému DDoS-Guard a nasazen v produkčním prostředí sítě VUT. Pro testování byly využity reálné záznamy DDoS útoků zachycené v průběhu několika měsíců. Výsledky ukázaly, že navržený algoritmus je schopen efektivně identifikovat různé typy útoků, včetně rozsáhlých UDP nebo DNS floodů, a generovat správná filtrační pravidla odpovídající skutečnému průběhu útoku. Mezi výhody tohoto přístupu patří především rychlost detekce (díky vynechání exportního zpoždění) a možnost detekovat dynamicky se měnící vzory útoku.

V závěru práce jsou diskutována zjištěná omezení -– zejména časování exportu cache, které ovlivňuje přesnost analýzy — a navržena možná vylepšení, jako je vícenásobná analýza v průběhu útoku, přesnější synchronizace exportu dat nebo pokročilejší metody agregace pro budoucí rozšíření nástroje.

DDoS attack detection using NetFlow cache analysis

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Matěj Grégr. The supplementary information was provided by Mr. Matěj Grégr. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jaroslav Streit
May 13, 2025

Acknowledgements

I would like to thank my thesis supervisor, Mr. Matěj Grégr, for valuable advice, patience, and time investment during my work on this thesis. I would also like to thank Mr. Peter Nagy for his help with integrating and deploying my tool in the Brno University of Technology network.

Contents

List of Figures

Chapter 1

Introduction

In today's highly interconnected world, the stability and security of network infrastructure are constantly threatened by cyberattacks, mainly Distributed Denial of Service (DDoS) attacks. These attacks can disrupt online services, cause financial loss, and compromise the availability of critical systems. To mitigate such threats, advanced monitoring and detection mechanisms are required.

This thesis investigates NetFlow, a network monitoring protocol, for detecting and identifying DDoS attacks. The emphasis is on analyzing NetFlow cache data collected directly from the ipt-netflow exporter in the Brno University of Technology backbone network. Unlike traditional methods that rely on exported NetFlow records, this work analyzes the flow cache directly, enabling faster detection before data is exported to a collector.

The research includes a comprehensive overview of the NetFlow protocol and the nature of DDoS attacks, including their taxonomy, attack vectors, and defense strategies. The core part of the thesis is the design and implementation of a Python detection tool that analyzes NetFlow cache snapshots captured during active DDoS attacks. This tool generates filtering rules based on identified attack patterns and is integrated with the existing DDoS-Guard detection system.

The thesis is organized as follows: Chapter 2 introduces NetFlow and network monitoring techniques. Chapter 3 details DDoS attacks and current trends. Chapter 4 outlines the workflow of the detection system and the algorithm used. Chapter 5 describes the developed tool's implementation, testing, and deployment. Chapter 6 concludes with an evaluation of results and outlines future improvements.

Chapter 2

Network traffic monitoring and NetFlow

In addition to monitoring the state of the network, its devices, and available services, we also need to look out for bad actors performing malicious activities and attacking the network.

There are two approaches to network monitoring: active and passive. Active, e.g., tools such as Ping or Traceroute, „inject traffic into a network to perform different types of measurements," as stated in [14]. Passive approaches observe traffic at an observation point in the network, for example, an interface of a network device such as a router [7]. Passive monitoring can be viewed as two methods. The first is packet capture, which provides the most information on network traffic by analyzing captured packets, but is challenging to perform in high-speed and high-volume network traffic. The other method is flow monitoring, where packets are aggregated into flows – „a set of IP packets passing an observation point in the network during a certain time interval, where all packets belonging to a particular flow have a set of common properties" as defined in [2]. These common properties are typically a 5-tuple, consisting of the source IP address and port, the destination IP address and port, and protocol, or a 7-tuple with additional IP type of service and SNMP input interface ID fields according to [14], but can also be defined otherwise.

This chapter will focus on the flow export protocol and technology, NetFlow.

2.1 NetFlow

NetFlow is a network monitoring protocol developed by Cisco Systems for Cisco routers, first patented in 1996, that collects and analyzes network traffic data. NetFlow tracks IP traffic to gather statistics about network traffic, identify patterns, help optimize network performance, or detect potential threats. The first widely used version, still in use today, was NetFlow v5, which primarily supports only IPv4 traffic. It became publicly available in 2002 [14]. Later, NetFlow v9, developed by Cisco, was introduced and was followed by the Internet Protocol Flow Information Export (IPFIX) open standard developed by the Internet Engineering Task Force (IETF), which has become the industry standard [15].

2.1.1 NetFlow architecture

NetFlow monitoring technology comprises three main components – flow exporter, flow collector, and the NetFlow export protocol.

- Flow Exporter – Device (observation point) in a network that aggregates network traffic data into flows based on shared characteristics and stores them in a flow cache until export
- Flow Collector – Device that collects exported flow data and prepares it for further analysis
- NetFlow Protocol – Protocol used in the process of exporting flow data, utilizing (in most cases) transport protocol UDP [14]

Network traffic data gathering and communication from exporter to collector is shown in Figure 2.1.

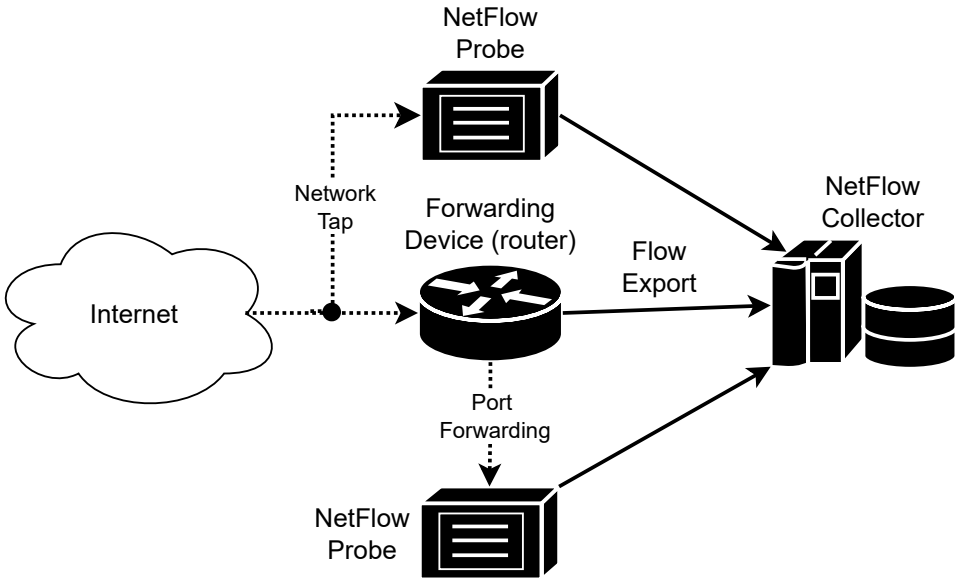

Figure 2.1: Scheme of network traffic monitoring and export of data from exporter to collector (taken from [14], edited)

Monitoring process

The entire monitoring process from packet capture to data analysis can be split into four steps – packet observation, metering and export, data collection, and data analysis, according to [14]. The scheme of these steps is shown in Figure 2.2. In the first step, the packet observation, packets are captured from an observation point and pre-processed [14]. In the metering and export stage, the packets are aggregated into flows with common properties as defined in 2. After a particular flow has ended, the flow record is placed in a NetFlow datagram and exported by the exporting process [14]. During data collection, the data are stored and pre-processed on the collector, aggregated and filtered, compressed, and summaries are generated [14]. The data analysis stage is usually exploratory. Data are commonly correlated, classified, and analyzed for intrusions, other anomalies, or other research purposes [14].

Figure 2.2: Architecture of NetFlow monitoring process (taken from [14], edited)

2.1.2 Exporter

In the NetFlow architecture, the exporter is a network device responsible for generating flow records of network traffic routed through an observation point in a network, typically a router or L3 switch. These records are aggregated, in the case of NetFlow, based on seven key fields (7-tuple) [6] – source IP and port, destination IP and port, layer three protocol type, type of service (ToS), and input logical interface. Aggregated flow data are stored in a local flow cache until they meet certain conditions and then exported to the collector. The conditions are:

- Active timeout – time interval (typically 120 seconds to 30 minutes [14]), after which all flow cache data is periodically exported to the collector
- Idle timeout – if no packets belonging to a certain flow have been registered during this time interval (typically 15 seconds to 5 minutes [14]), the flow is exported
- If the flow ended, e.g., TCP FIN or RST flag has been observed
- Emergency expiration – in case the flow cache becomes full or there has been a significant change to the export device, flow data is exported

The exporter can also be a dedicated device in the network. Such a device is referred to as a flow probe. Probes can duplicate traffic between hosts using network taps or Test Access Ports (TAP) [14]. Another option is port mirroring – the packet forwarding device (router) mirrors network traffic to another port where the probe is listening [14].

Some NetFlow exporters – especially in high-speed and high-volume networks – implement packet sampling, e.g., storing flow data of every 100th, 1000th, or so packet. Such an approach reduces the load on the exporter. A consequence of sampling is the accuracy of stored data – we don't know what happened in the packets between samples. That means, for the purpose of threat detection, it is better not to apply packet sampling.

The exporter used for this work is the open-source project ipt-netflow [1], further described in 4.1.

NetFlow cache

A NetFlow cache, or just a flow cache, is temporary data storage for aggregated traffic data within an exporter. Cache stores flow data until one or more export conditions are met. Each cache entry typically contains details such as the total number of bytes and packets transferred and timestamps of the first and last packet belonging to the flow. The flow cache format can vary slightly based on the specific exporter implementation.

2.1.3 Collector

The NetFlow collector is a dedicated device or application that receives, stores, and manages flow records obtained from the exporter, creating a centralized database of real-time and

past network traffic information. Collector processes flow attributes, such as IP addresses, port numbers, protocols, transferred bytes and packet counts, and timestamps, to create a detailed picture of activity on the monitored network. Thanks to centralizing that data, the collector enables network administrators to analyze traffic patterns, detect potential threats on the network, and optimize network performance with the help of visualization and reporting tools, often integrated within the collector.

2.1.4 NetFlow protocol version formats

NetFlow has been developed in several versions since Cisco patented it in 1996. The first commonly deployed version was NetFlow v5, which supports only IPv4 traffic but is still widely used today. Later, it was followed by NetFlow v9, which implemented templates and, hence, besides other things, IPv6 support. Lastly, based on NetFlow v9, the Internet Engineering Task Force (IETF) developed an open standard IP Flow Information Export (IPFIX). The most widely implemented transport protocol for the export of flow records is UDP, which is easy to implement even in hardware exporting devices with very low overhead, as stated in [14].

NetFlow v5

NetFlow version v5 has static size fields for IP addresses in the flow records, meaning it can only support IPv4 traffic. NetFlow v5 packet comprises two parts – the header and the body. The header of such a packet is shown in Figure 2.3. The header contains information about the protocol version, count of flow records, time of export, information about the exporter (fields engine type and engine ID), and sampling interval, according to [4]. The size of the v5 header is 24 octets.

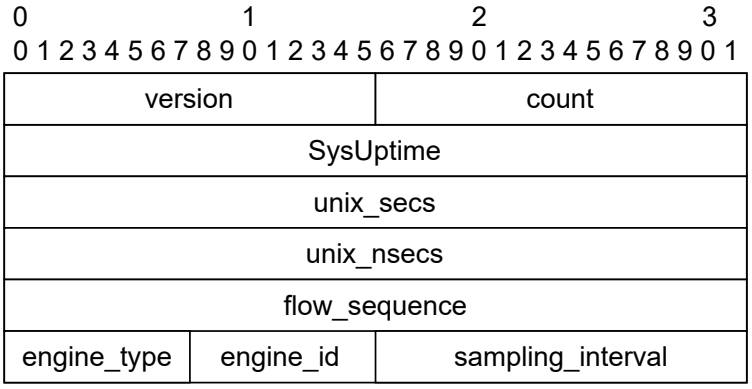

Figure 2.3: NetFlow v5 packet header structure

The body of the packet (flow records) contains information about the specific flows – source and destination IP addresses, ports, SNMP index of input and output interface, number of packets and bytes transferred in the flow, time of first and last packet, protocol type, type of service, and source and destination prefix mask bits as shown in figure 2.3, according to [4]. The size of the v5 flow record is 48 octets.

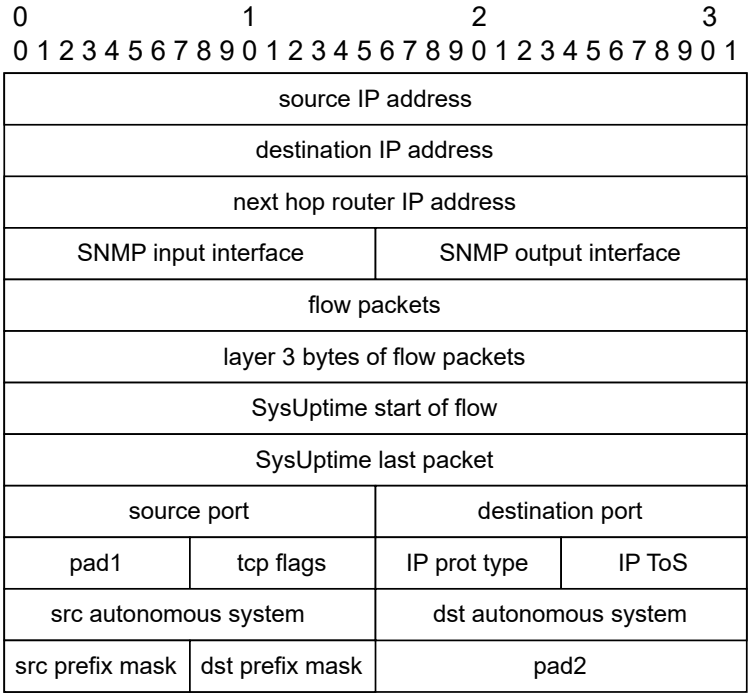

Figure 2.4: NetFlow v5 flow record structure

NetFlow v9

In 2004, NetFlow v5 was obsoleted by its successor NetFlow v9 [14]. NetFlow v9 introduced templates and supports adaptable data formats and IPv6, MPLS, and other features [14]. The NetFlow v9 protocol format is described in [7]. NetFlow v9 is the base for the IETF open standard protocol IPFIX.

A template defines a collection of fields in the NetFlow datagram. It provides multiple advantages [7]:

- New fields can be added without the need to change the export record format
- Collector can still interpret flow records, even if it does not understand the new fields in a template
- Templates allow the export of selected fields from the stored flows

NetFlow v9 uses multiple types of records. These are template records, flow data records, options template records, and options data records. The collection of flow records makes a FlowSet. There are three types of flowsets – template flowset (multiple template records grouped in an export packet), options template flowset (grouped option template records), and data flowset (collection of records of the same type – either flow data or options data record defined by template record or an options template record) [7].

Generally, NetFlow v9 encapsulates export packets in UDP datagrams, but has been designed to be independent of the transport protocol [7].

Version 9 packet and flow record layouts

V9 export packet consists of a header followed by one or more FlowSets. FlowSets are distinguished by FlowSet IDs. The export packet header is shown in Figure 2.5. The packet header contains a protocol version number, the total number of records in the export packet, the time since the device was first booted, the time of export, the sequence number of the exported packet, and the ID of the exporter observation domain.

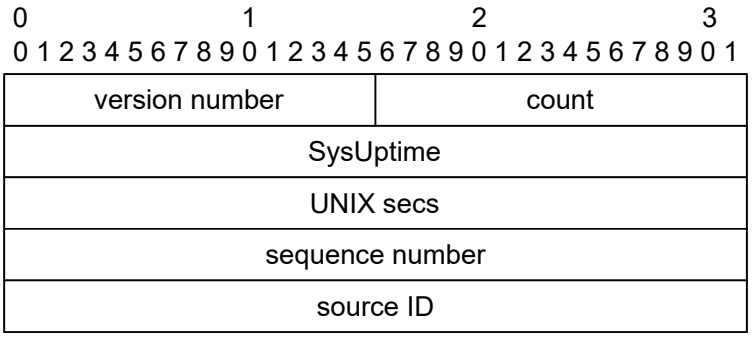

Figure 2.5: NetFlow v9 packet header structure

Individual FlowSet formats are as follows:

- Template FlowSet contains its length, ID, total field count of the FlowSet, and then multiple fields *type* and their corresponding *length* fields. An example format of the template FlowSet is shown in Figure 2.6. Template FlowSets are essential for NetFlow collectors to interpret flow records [7].

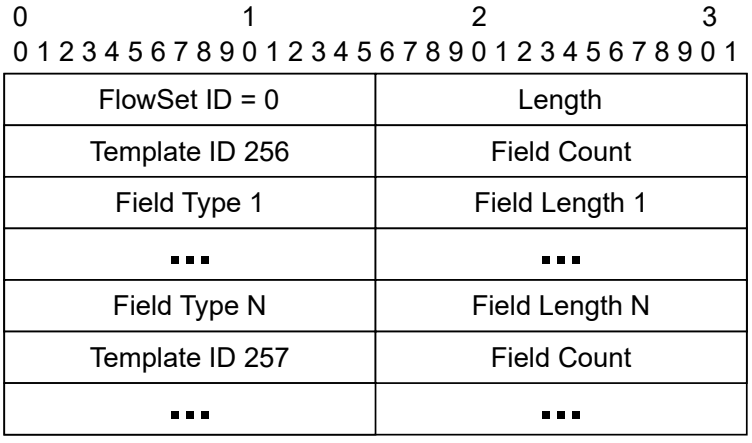

Figure 2.6: Template FlowSet format example

- Data FlowSet contains FlowSet ID identical to the ID of its template, length of FlowSet, and collection of field values previously defined in the template FlowSet. FlowSet is then padded to be 4-byte aligned. Data FlowSet can only be interpreted if its corresponding template is available in the collector [7]. An example of a data FlowSet is shown in 2.7.

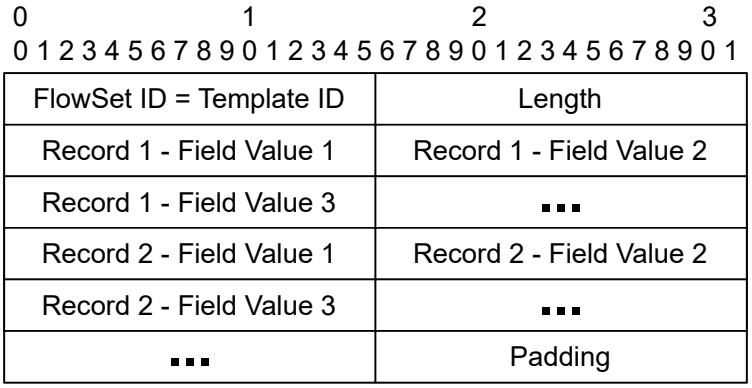

Figure 2.7: Data FlowSet format example

- Options Template FlowSet – FlowSet supplies information about NetFlow process configuration and specific data instead of IP flows [7]. Contains fields ID, which is always 1 (reserved for Template FlowSet), total length, template ID greater than 255, option scope length – length of scope fields in options template record in bytes, and one to many scope field types and their lengths, and option field types and their lengths. An example is shown in 2.8.

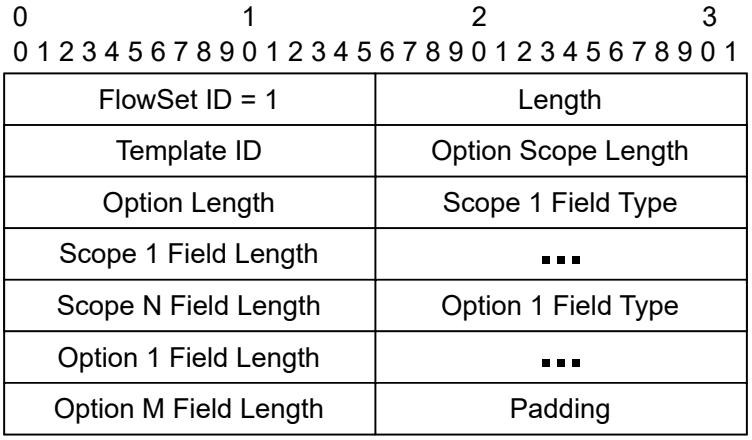

Figure 2.8: Options Template FlowSet example

- Options Data Record – these records are regularly sent in data FlowSets, with configurable frequency [7]. The Options data record contains an ID equal to the template ID and length of the record, and then the rest is a collection of flow records containing scope and field values. The records are padded to 4 bytes [7]. An example of an Options Data record is shown in Figure 2.9.

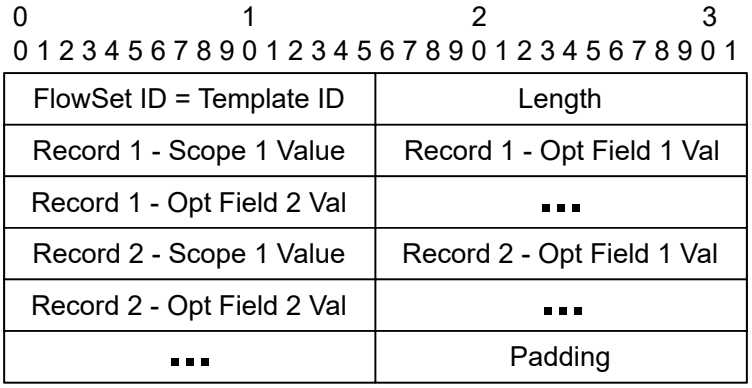

Figure 2.9: Options Data Record example

The collector receives the definitions of templates from the exporter before receiving flow records. Suppose the template definition for a particular flow record was not accepted during record reception. In that case, the collector should store the record for later decoding after receiving the needed template. The collector cannot assume that the data FlowSet and its associated template IDs are exported in the same packet. Similarly, it cannot be assumed that only one template FlowSet exists in an export packet. All template definitions have a limited timeframe. After it expires, the collector cannot decode flow records associated with such a template. Any new template definition received should override existing definitions [5].

Chapter 3

Distributed Denial of Service attacks

A distributed denial of service (DDoS) attack is a network attack that aims to throttle or deny legitimate user access to a service or resource by exhausting the resources of the victim system, network, or website providing the service. Today, DDoS attacks are a growing problem, and effective defenses are still being developed. However, there is no single solution that would solve the entire DDoS problem [27].

Although the goal of such a denial-of-service attack is always the same – to cause damage to the victim – the motives behind the attack can differ. The usual reason to perpetrate a DDoS attack is some form of material (financial) gain by hindering competitors' services or even by blackmailing, according to [18]. Other frequent motives are personal reasons, such as revenge – many attacks are carried out against home computers [18]. Other reasons, as stated in [27], could be ideological beliefs (political reasons) or being part of cyber warfare (usually military groups or terrorist organizations). Cyberwarfare DDoS attacks are generally performed by well-trained specialists. They are aimed at critical infrastructure sections of the opposing side (country), often affecting civilians by targeting financial organizations, mobile service providers, etc.

Typical denial-of-service attacks are carried out by a single device, generating traffic to the victim system, which is usually insufficient to cause any severe disruption to legitimate traffic. In a DDoS attack, being distributed means that the attack is carried out by a large group of so-called zombies, or bots, together forming a botnet [27]. A malicious actor controls this botnet.

3.1 DDoS attack process

DDoS attacks can be split into multiple phases. In the first phase, the attacker must gain control of a large army of machines (bots). That is typically done by exploiting weaknesses, bugs in software, or old machines that are no longer maintained with up-to-date software. Botnet zombies, or bots, are usually recruited to the army through remotely controlled worms or Trojan horses.

The attacker then controls the botnets through handlers [23]. Handlers have mainly been replaced by Internet Relay Chat (IRC) networks, which allow attackers to hide their attacks among legitimate IRC traffic. It allows attackers to control the entire botnet through text messages on IRC centrally. Still, it also makes the commanding server a single point

of failure – once the defender discovers the command server, the entire botnet can be shut down [27]. An alternative to IRC is HTTP-based communication with the bots. That allows the attacker to communicate with the botnet without maintaining a connection to the centralized server; they download attack instructions via web requests [18].

After the bots have been installed on the victim machines (a bot is typically a tiny executable file on the exploited machine or system) and are connected to the desired IRC server using the included remote TCP port and authentication key to join the private attacker's channel, they wait for commands from the attacker [23] – scheme of such a botnet is shown in figure 3.1.

The bots are then commanded to send large amounts of traffic through desired protocols and in desired time intervals to evade attack detection, which is all controlled by the attacker while his identity stays hidden.

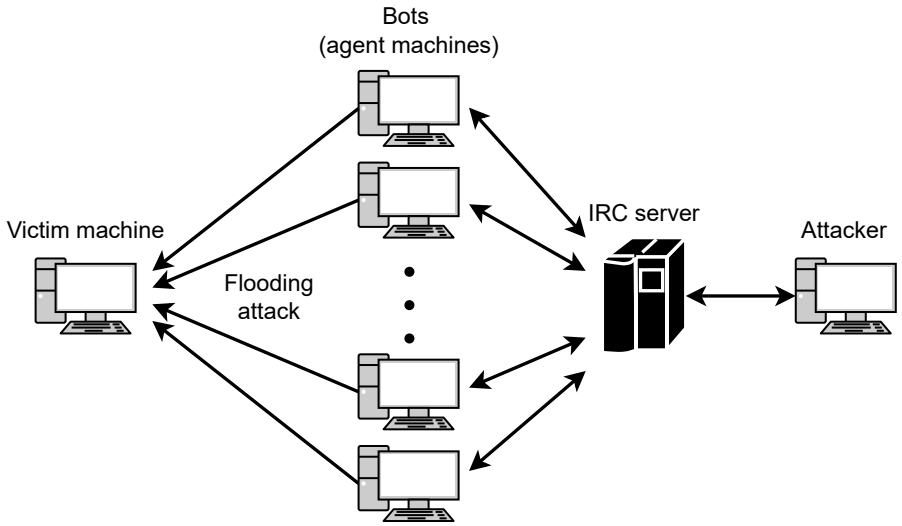

Figure 3.1: Scheme of botnet using IRC communication to perform a DDoS attack

Performing such an attack today is quite easy; many services offer DDoS capabilities that even an amateur can use to perpetrate an effective attack. Some of these tools are, for example:

- IRC based – Trinity [17], using TCP SYN or UDP flooding
- HTTP based – BlackEnergy [16], using HTTP POST for communication and ICMP, TCP SYN, UDP, and HTTP GET flood to attack a victim, or Low Orbit Ion Cannon (LOIC) [10], [22], an open source project for stress testing and DoS attacks, using TCP, UDP, and HTTP flooding

3.2 DDoS attack classifications

DDoS attacks employ various strategies to render the target service unavailable. These attacks can vary significantly in the ways and methods they use to achieve their goal. Thus, it is essential to distinguish and classify them to tailor proper defense mechanisms.

Many possible classifications divide DDoS attacks into categories; some of the main ones are as follows.

Classification by exploited weakness

As mentioned in the previous subsection, logical (semantic) and flooding (brute-force) attacks are the primary ways to divide DDoS attacks.

- Logical attacks – This type of attack exploits a weakness in a certain application or protocol feature or an implementation bug, usually allowing the attacker to exhaust some resource of the targeted machine, application, or service, rendering it unavailable.

  One of the typical examples of such an attack is a TCP SYN attack. A TCP SYN attack exploits the TCP protocol by opening but never completing many TCP connections using SYN requests. That exhausts the available connection queue of the target, making it impossible to accept new legitimate connection requests [18].

  Different examples could be a CGI (common gateway interface) request attack, which consumes the CPU resources of a web server and makes it unavailable to process user requests, or another TCP exploiting attack called NAPTHA [18].

  These attacks are commonly more difficult to detect because they quite accurately mimic legitimate network traffic.

- Brute-force attacks – This DDoS attack is quite simple yet very effective. It is performed by initiating significant network traffic to the target victim. According to [18], these attacks can be pretty similar to the logical attacks in the sense that they both exhaust the victim's resources by sending excess traffic and can exploit a weak feature – in case of brute-force attacks, for example, DNS reflection attack or smurf attack, exploiting Internet Control Message Protocol (ICMP). The main difference is that logical attacks can be mitigated by modifying the exploited feature or fixing a bug. However, there is no „simple“ way to defend against a large-volume brute-force attack. If a logical attack is defended against by modifying the exploited feature, it becomes a brute-force attack [18].

  Typical examples of brute-force attacks are UDP flooding attacks [8], further described in 3.3. Another example can be TCP SYN flooding attacks or ICMP flooding attacks.

  Brute-force attacks are the main focus of this work.

Classification by degree of automation

Another way to divide DDoS attacks into groups is to look at the level of automation implemented in the attack. This classification divides DDoS attacks into manual, semi-automatic, and automatic [18].

- Manual attacks – During a manual attack, no automation is used. The attacker manually scans remote devices for possible exploitable weaknesses, infects them with malicious software, and performs the desired attack. Today, no more DDoS attacks are fully manual.

- Semi-automatic attacks – These attacks utilize a handler and agent machines. The entire process of recruiting a botnet is automated – from scanning and exploiting to infecting with malicious attack software. The attacker then performs the attack

manually by choosing desired parameters such as the attack type, the attack volume, etc.

Semi-automatic attacks can be further divided into attacks with direct and indirect communication, depending on the type of communication between the agent and the handler machines:

- Direct communication – In this type of attack, the agent and handler machines need to know each other's identities. According to [18], the handler's IP address is typically hard-coded in the malicious software installed on the agent machines, and the agents then communicate their addresses to the handlers.

  One main disadvantage of direct communication is that discovering one agent machine exposes the entire attack network.
- Indirect communication – Attacks using indirect communication generally use some legitimate service to synchronize all agents. The most widely used service is Internet Relay Chat (IRC), already described in the section 3.1.

- Automatic attacks – Automated attacks entirely automate both the recruit and attack phases; hence, there is no need for communication between attacker and agent machines. That reduces any exposure to the attacker, who only has to issue a single command to start the recruitment phase, and the attack then performs on its own.

Classification by validity of source address

Here, we can divide DDoS attacks into two groups: those with a valid source IP address and those with a spoofed one. DDoS attacks often use spoofed source IP addresses, which means they replace the original source IP address in the IP packets. This technique is used primarily to perform reflector attacks: the attacker replaces (spoofs) their source IP address with the IP address of a victim and then exploits some protocol or service, such as ICMP or DNS, to reflect a large amount of traffic to the victim. Spoofing also makes the attacker less likely to be discovered and makes the attack more difficult to detect.

Multiple spoofing techniques are commonly used [26]:

- Random spoofed address – Some attacks use randomly generated 32-bit numbers as spoofed addresses. This technique is simple and can be prevented using some defense mechanism, such as ingress filtering.
- Subnet spoofed address – Attacker chooses a different IP address from the agent machine's subnet address space as the source IP address. This method can bypass ingress filtering.
- Fixed spoofed address – Technique used in reflector DDoS attacks. The attacker spoofs the victim's IP address as the source IP address of the malicious traffic he generates to redirect responses and overwhelm the victim.

Classification by victim type

DDoS attacks can target various systems, from individual applications to potentially entire networks or infrastructure. Understanding this kind of classification is important for the design of attack-specific defense mechanisms. DDoS attacks in this category are as follows [18]:

- Application-level attacks – This attack targets an individual application on the victim's system. A typical example is using HTTP requests to flood web servers or authentication systems to render a particular service unavailable.
- Host-level attacks – Host-level attacks aim at completely disabling the victim's machine. Methods used for this can be, for example, TCP SYN floods, exhausting connection queues on the machine, or malformed packets, potentially causing system crashes. Such an impairment usually requires human intervention, such as a manual machine reboot.
- Resource attacks – Attacks in this category aim at disrupting operations on a network by targeting its critical resources, such as specific firewalls, routers, DNS servers, or bottleneck links in a network.
- Network attacks – Network attacks are large-volume flooding attacks that simply attempt to consume the incoming bandwidth of a network by generating traffic targeted at the victim network address space. The large volume of these attacks makes them easier to detect.
- Infrastructure attacks – These attacks target services crucial for global Internet operations. An example of this type of attack is attacks on root DNS servers or routing protocols.

3.3 Common flooding DDoS attacks

The main goal of this work is to detect and identify volumetric DDoS attacks. The most commonly known flooding attacks are:

- TCP SYN flood attack – This attack exploits the 3-way handshake of a TCP connection. The attacker sends a large number of SYN packets, to which the victim responds with SYN/ACK packets, and creates half-open TCP connections with the victim that never finish. That exhausts the victim's connection resources and makes the machine or server unable to provide more connections and services until the victim can drop the unfinished connections [12].
- ICMP flood attack – Also known as the Smurf or ping flood attack. This attack utilizes the ICMP protocol to flood the victim with ICMP *echo request* (pings). The attacker spoofs the packet's source IP request to the victim's address and sends many echo requests to a multicast or broadcast address of the victim's network. Everyone listening at that multicast address then sends an echo response to the victim, which overwhelms them [12]. This type of attack is called a reflector attack [19], but can also be performed as a direct attack.
- DNS flood attack – DNS flooding targets a particular DNS server to disrupt a specific domain's DNS resolution. That can make services such as websites or APIs on such domains unavailable [9].
- UDP flood attack – UDP flood attack sends a large amount of UDP datagrams to random ports on the victim's machine. The victim machine attempts to find an application that listens on each port that received a datagram; if there is none, it

is forced to reply with an ICMP *destination unreachable* message. That causes the victim system to exhaust its computational resources [12].

- ARP flood attack – This kind of attack floods the victim by spoofed ARP (Address Resolution Protocol) requests, which causes exhaustion of computational resources on the victim side. It is typically used in a local network. Another option is the periodical sending of spoofed ARP responses with the IP address of the attacker, enabling man-in-the-middle (MITM) attacks [12].
- TCP RST flood attack – Another type of attack exploiting the TCP transport protocol. A large number of received TCP RST packets can cause existing connections on the victim side to be reset [12].

3.4 Defense mechanisms against DDoS flooding attacks

Effective defense against the evolving threat of Distributed Denial of Service (DDoS) attacks necessitates a comprehensive understanding of current mitigation strategies and tools. Modern defense mechanisms can be categorized based on their deployment location and the timing of their implementation [27].

3.4.1 Deployment location

Defense mechanisms are typically deployed at four different points within a network infrastructure [27]:

- Source-based mechanisms: Implemented near the origin of malicious traffic to prevent outbound attacks.
- Destination-based mechanisms: Deployed at the target to protect against incoming malicious traffic.
- Network-based mechanisms: Situated within the network to monitor and filter traffic.
- Hybrid mechanisms: Combine multiple deployment points for coordinated defense.

Source-Based Mechanisms

While traditionally less common due to limited incentives for source networks, source-based defenses are gaining traction with the advent of collaborative security practices. Modern implementations include:

- Reverse Firewalls: Reverse firewalls protect the outside from DDoS attacks within the firewall's network. These systems monitor and control outbound traffic, preventing internal hosts from participating in DDoS attacks [27].
- Rate Limiting: Applies thresholds to outbound traffic to prevent excessive flows that could contribute to DDoS attacks [27].
- Ingress/Egress filtering: This mechanism implements traffic filtering at the edge routers of the source network. It can prevent packets with spoofed IP addresses from leaving a local network by filtering them out at the edge router based on the

IP range of the network's address space. However, it cannot filter out spoofed IP addresses within the correct IP range [27].

Source-based defense mechanisms are rarely deployed because they bring no benefit to the source network while being costly and consuming computational power.

Destination-based mechanisms

Destination-based defense mechanisms are deployed at the target end. The commonly used destination-based defense mechanisms are the following:

- IP traceback: Traceback is a process of tracing back the route of a packet to its source. There are two approaches to IP traceback – packet marking and link testing [27].

  In packet marking, routers mark each packet routed with their identification signature. That allows the victim to trace each packet to its origin, even with a spoofed IP source address, so that the victim can distinguish malicious from legitimate traffic.

  Link testing is a method that starts at the router closest to the victim. It then iteratively tests its upstream links until it finds the link carrying the malicious traffic.

- Cloudflare's Anycast Network: Utilizes Anycast routing to distribute traffic across multiple data centers, mitigating the impact of DDoS attacks by absorbing and dispersing malicious traffic globally [11].

- Hop-count filtering – On the destination side, source IP addresses and hop-counts are stored in a table during routine traffic. According to the table, once a DDoS attack has been detected, the victim can inspect the incoming packets and filter out spoofed IP addresses. Unfortunately, this method can often be quite inaccurate [27].

- Radware DefensePro: Employs behavior-based detection and real-time signature creation to identify and mitigate sophisticated DDoS attacks while minimizing false positives, even protecting against zero-day attacks [25].

Network-based mechanisms

Network-based mechanisms are deployed within network infrastructures. Some of the currently used network-based defense mechanisms are:

- FastNetMon: An open-source DDoS detection tool that analyzes traffic using various protocols (e.g., NetFlow, sFlow) and can trigger mitigation actions such as BGP blackholing [13], [21].

- nScrub: A high-performance DDoS mitigation system capable of operating at 10 Gbps line-rate, offering features like anomaly detection and rate limiting based on traffic behavior [20].

Hybrid mechanisms

The mechanisms mentioned above were centralized, meaning that each deployment point carried out both detection and reaction without mutual cooperation. Hybrid defense mechanisms employ cooperation between deployment points, e.g., attack detection can be done at the victim side, and the response can be distributed to other locations.

Some examples of hybrid defense mechanisms are:

- Arbor DDoS Protection: Netscout Arbor provides adaptive DDoS defense through a hybrid approach, combining on-premises mitigation with Arbor Cloud, a global scrubbing service [3].
- Radware's Cloud DDoS Protection Service: Radware's cloud services offer multi-layered protection against DDoS attacks using advanced real-time behavioral analysis. A globally distributed scrubbing network supports the service. It can mitigate large-scale attacks close to their origin, leveraging anycast-based routing to ensure high availability and minimal latency during mitigation. [24].

3.4.2 Time of deployment

In this category, defense mechanisms are divided into three groups: before the attack (prevention), during the attack (detection), and after the attack (identification and response) [27].

Attack prevention

Attack prevention is the most effective way to defend against DDoS attacks. Prevention mechanisms can be deployed at all locations – source, network, or destination. Most prevention mechanisms try to fix existing vulnerabilities in protocols or computer systems. There are many possible prevention mechanisms; some general examples could be [27]:

- System and protocol security – Bug fixing, vulnerability mitigation, updating outdated software, and preventing unauthorized access to machines.
- Reconfiguration mechanisms – Option to modify the victim's network topology to, for example, provide more bandwidth to tolerate the DDoS attack.
- Installation of firewalls – Firewalls and Intrusion Detection & Prevention systems (IDPSs) prevent unauthorized access to machines, which attackers could potentially try to recruit for an attack.
- Employment of filters, such as ingress/egress filtering, hop-count filtering, etc.
- Load balancing – Improvement of system/network performance.

Attack detection

Detection mechanisms are deployed when an attack occurs. Once again, these mechanisms are suitable for use in all locations mentioned above. Different mechanisms use different strategies to detect ongoing DDoS attacks. Some mechanisms use flow-based congestion levels, while others look for anomalies in traffic patterns or deploy artificial intelligence models to detect attacks. Such systems study the routine behavior of network traffic and then detect changes in the monitored patterns.

Detection mechanisms are best deployed at the source or intermediate networks, where they are less vulnerable to DDoS attacks and less transparent for the attacker [27].

Attack identification and response

These systems are used once a DDoS attack has been confirmed and deployed at the destination of the attack. These systems aim to minimize the impact of the attack and maximize the availability of services.

Such mechanisms have two parts: source identification and response to the attack.

- Source identification – At first, it is essential to identify the source of an attack, for example, by using a traceback of a spoofed source IP address to the real one. That usually requires many routers between the attack source and destination to support packet marking or similar methods [27].
- Response – Once the attacker has been identified, the victim has to initiate an adequate response. Most DDoS defense mechanisms typically use rate limiting / throttle (DEFCOM) and packet filtering (history-based IP filtering) at upstream routers to minimize the impact of attack flows [27].

3.5 Current trends of DDoS attacks

DDoS attacks are still a rapidly growing problem, both in volume and quantity. According to Cloudflare's DDoS reports [29], in 2023, the most significant DDoS attack registered by Cloudflare was over 2.6 terabits per second (Tbps). In 2023 Q3, Cloudflare mitigated, to that day, the most significant HTTP attack of 201 million requests per second, which was almost eight times larger than in 2022. In Q4 2023, they also registered 175 % year-over-year and 25 % quarter-over-quarter increases in network-layer attacks, the most significant amount of attacks being DNS and SYN flood, and the largest increase of 1161 % was recorded in TCP ACK-RST flood attacks. The distribution of network-layer attacks in Q4 2023 is shown in Figure 3.2.

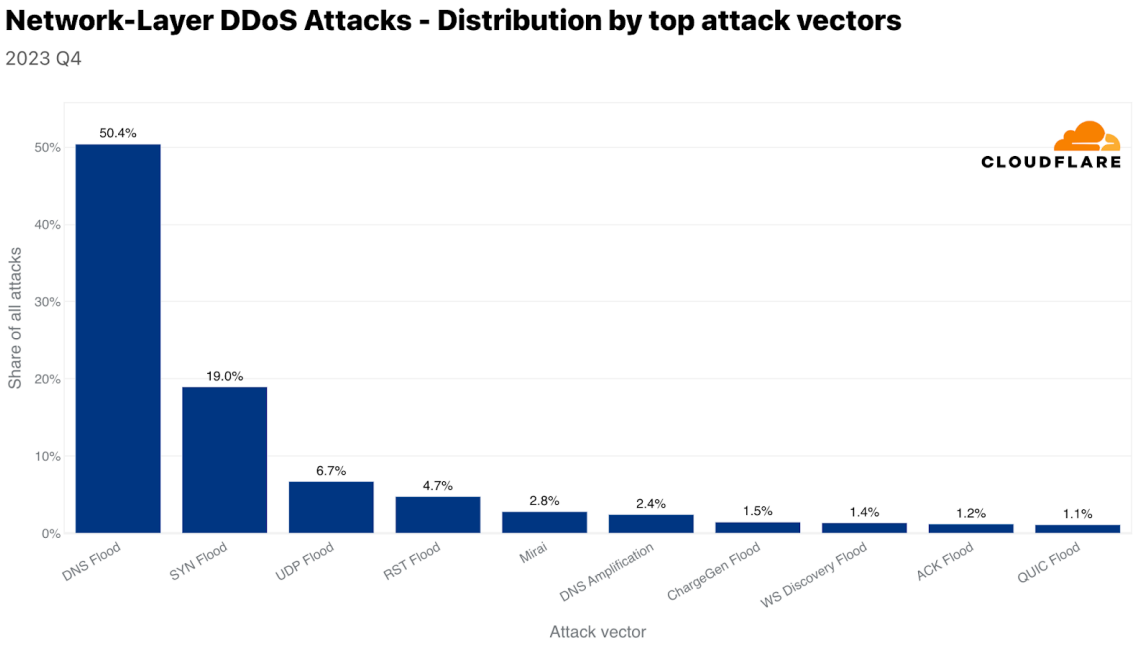

Figure 3.2: Network-layer attacks distribution by attack vectors (taken from Cloudflare's DDoS report of Q4 2023 [29])

In Q1 of 2024 [29], Cloudflare's defense systems mitigated 4.5 million DDoS attacks, which was a 50% increase compared to the previous year. The distribution of attack vectors was similar to the last quarter, with a strong representation of DNS flood in the lead (52.1 %), followed by SYN flood, RST flood, and UDP flood.

In Q2 of 2024 [29], Cloudflare registered a slight (11%) quarter-over-quarter decrease in the quantity of DDoS attacks compared to Q1, but a 20% increase compared to the previous year, mitigating 4 million attacks. The distribution of types of DDoS attacks in Q2 of 2024 is shown in Figure 3.3.

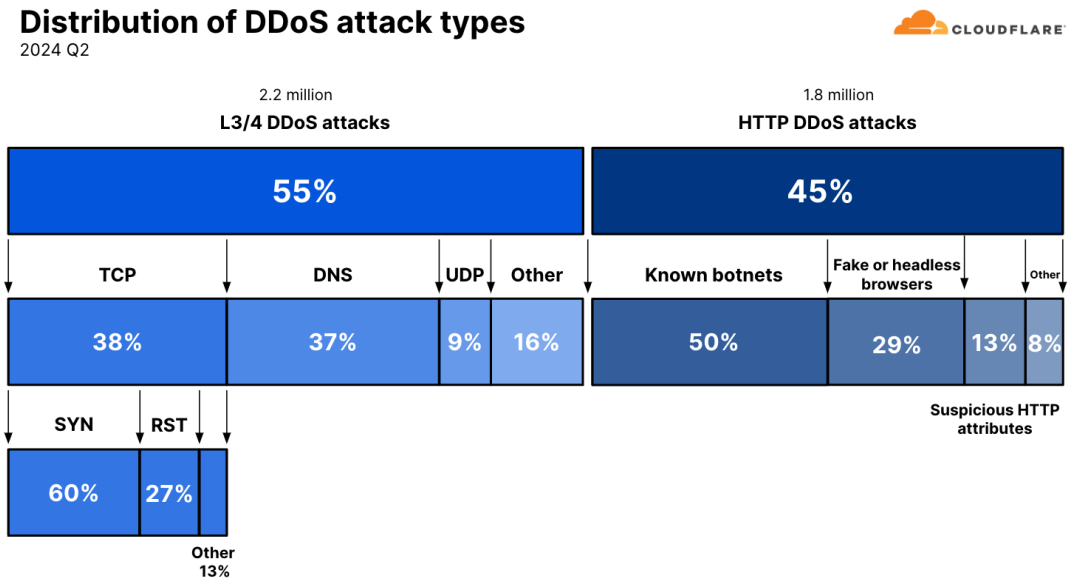

Figure 3.3: DDoS attack distribution by types and vectors (taken from Cloudflare's Q2 2024 report [29])

According to Cloudflare's DDoS report of Q3 2024 [29], there were almost 6 million mitigated attacks, representing an increase of nearly 50 % compared to Q2. More than 200 were hypervolumetric – exceeding three Tbps and 2 billion packets per second, with the most significant attack of 4.2 Tbps, breaking a new record on October 21, 2024. About half were network-layer attacks and half were HTTP attacks. The distribution is shown in Figure 3.4.

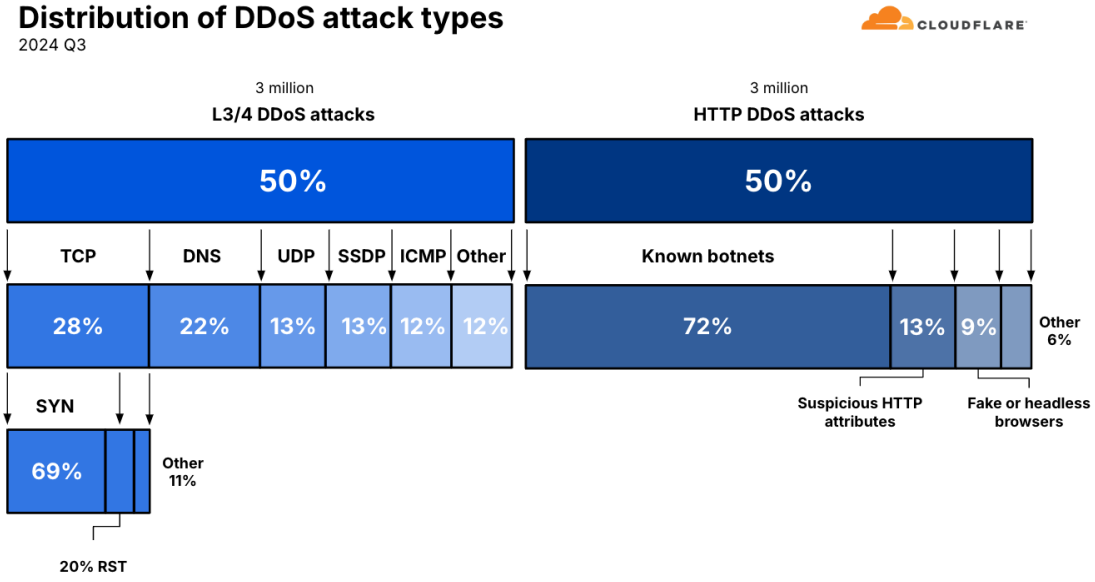

Figure 3.4: DDoS attack distribution by types and vectors (taken from Cloudflare's Q3 2024 report [29])

Q3 brought a 4000 % increase in Simple Service Discovery Protocol (SSDP) amplification attacks. SSDP attacks exploit the Universal Plug and Play (UPnP) protocol, sending SSDP requests to vulnerable UPnP devices such as routers, printers, etc., with spoofed source IP addresses as the victim's. These devices respond to the victim with overwhelming and amplified traffic [28].

Chapter 4

Application workflow and design

This chapter describes the design of a tool that will be able to recognize a DDoS attack pattern in a snapshot of a flow cache and create a sufficient filtering rule that can further be used to mitigate the attack. The existing tool, DDoS-Guard, in the Brno University of Technology network will be used to detect an ongoing attack and create a flow cache snapshot that contains the attack flows among regular traffic. In this way, passive and active timers on the exporter will be avoided, leading to faster attack pattern recognition and potentially higher accuracy thanks to direct access to the flow cache.

The following section also briefly describes the NetFlow exporter used in the network traffic data collection in the Brno University of Technology network and the flow cache it uses.

4.1 Ipt-netflow exporter

The ipt-netflow exporter is the NetFlow exporter used in the Brno University of Technology network. It is a high-performance kernel-level NetFlow implementation designed for Linux systems. It operates as a kernel module integrated with the iptables framework, enabling the capture and export of network traffic data in NetFlow format. The exporter supports multiple versions of NetFlow, including NetFlow v5 and v9[1].

4.1.1 NetFlow cache

A NetFlow cache is a temporary storage mechanism used to maintain records of active network flows observed by a NetFlow device. It collects and aggregates flow information, such as source and destination IP addresses, ports, protocols, and timestamps. Once flows expire (based on active or inactive timeout) or meet specific criteria, the cache exports the data to a collector for further processing and monitoring. The structure of the particular cache used by the ipt-netflow exporter is visualized in Figure 4.1.

[1] https://github.com/aabc/ipt-netflow

```
hash a dev:i,o mac:src,dst vlan type proto src:ip,port dst:ip,port nexthop tos,tcpflags,options,tcpoptions packets bytes ts:first,last
3700 0 4,-1 44:a9:2c:41:02:51,58:8d:09:bf:5e:80 1700 0800 6 113.98.60.58,53739 37.46.209.74,80 0.0.0.0 0,2,0,0 1 40 14921,14921
46b00 2 4,-1 44:a9:2c:41:02:51,b8:83:03:62:20:38 112 0800 1 194.180.110.1,0 185.73.23.133,769 0.0.0.0 c0,0,0,0 7 539 38690,14946
7dc00 0 5,-1 44:a9:2c:41:04:41,44:a9:2c:41:01:61 639 0800 1 147.229.252.91,0 35.203.211.249,769 0.0.0.0 0,0,0,0 1 72 14811,14811
77300 0 4,-1 44:a9:2c:41:02:51,d0:7e:28:75:b6:88 1700 0800 1 37.19.216.1,0 147.229.255.100,2048 0.0.0.0 0,0,0,0 1 84 14620,14620
```

Figure 4.1: Example of flow data stored in the ipt-netflow cache

4.2 Workflow description

This section explains the overall workflow of the entire process from attack detection to creation of the filtering rule. The scheme of this workflow is depicted in Figure 4.2 below.

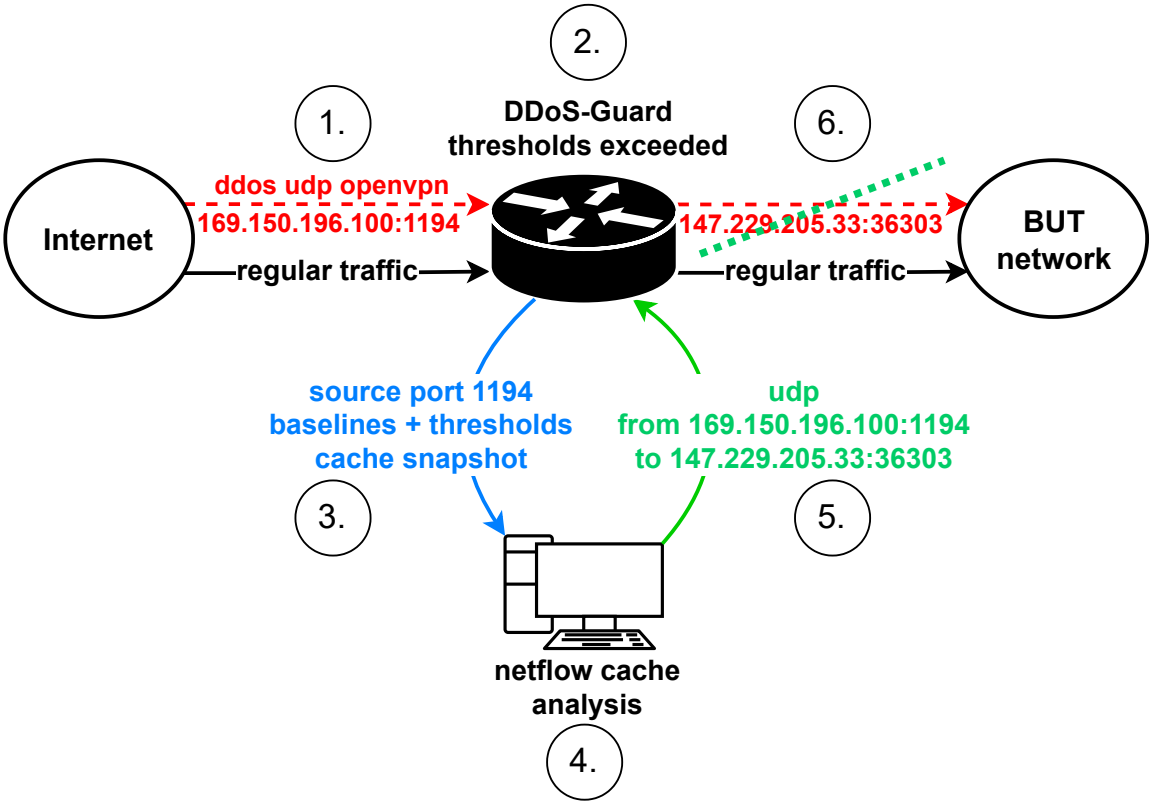

Figure 4.2: The complete workflow scheme during an ongoing DDoS attack

1. At first, the DDoS-Guard analyzes traffic flowing through the ipt-netflow exporter, aggregates it into flows, and saves the flows to its flow cache.

2. Once the DDoS-Guard detects an abnormal increase in traffic volume that exceeds certain baseline-based thresholds, it creates a snapshot of the current contents of the exporter's cache.

3. Then, it exports the snapshot together with the base filtering rule and current baseline values, which will then be used to filter the cache and narrow down the search area for the attack traffic.

Rule	openvpn	
	packet/s	**bit/s**
Baseline	4.5k	46.4M
Threshold	50.0k	500.0M

Table 4.1: Input parameters provided to the developed tool by DDoS-Guard

4. After the tool receives the flow cache snapshot and the required parameters, it analyzes the cache, checking for any significant traffic contributors that are most likely to be causing the abnormal increase in traffic volume, causing the DDoS attack. The tool analyzes combinations of source and destination IP addresses and ports, protocols, and tcpflags in the case of TCP traffic.
5. Once the cache is fully analyzed and the flow data cannot be aggregated anymore, the tool creates a filtering rule based on the largest aggregate. The rule is then given back to the DDoS-Guard.
6. The DDoS-Guard can then deploy the received filtering rule and cut out the DDoS attack traffic that goes to the Brno University of Technology network.

The final filtering rule produced by the tool developed in this thesis is compared to the result in the DDoS-Guard attack report, shown in Figure 4.3.

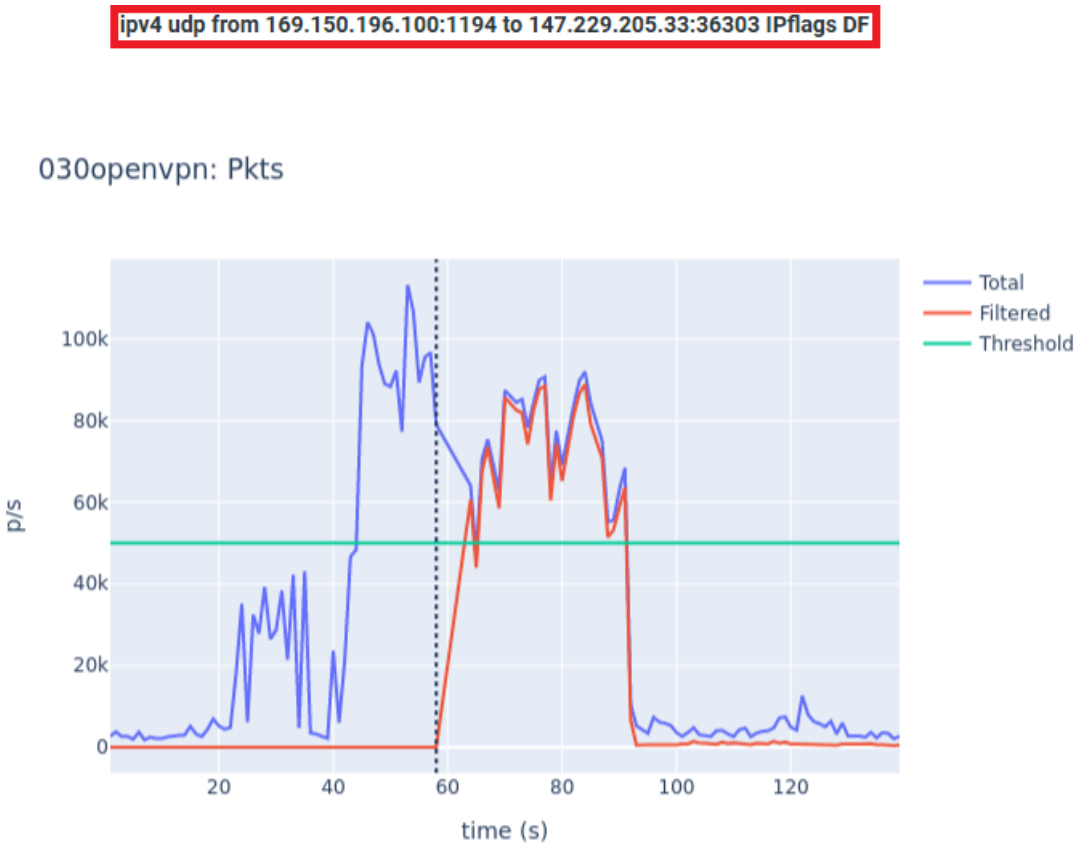

Figure 4.3: Final attack evaluation by DDoS-Guard, used to compare results with the developed tool

4.3 Application design

The tool was first designed as a console application in Python, later modified as a Python module for the purpose of integration into the DDoS-Guard tool and deployment in the Brno University of Technology network for testing on real-time data.

The following section describes the design of the algorithm that was later implemented and used to analyze the NetFlow cache.

4.3.1 Detection algorithm

The detection algorithm is designed to isolate flows or flow groups responsible for anomalous spikes in traffic. The overall process follows a hierarchical narrowing approach and proceeds in several main steps:

1. Input and pre-filtering

First, the algorithm receives three essential components from DDoS-Guard:

- a flow cache snapshot of traffic,
- baseline and threshold values,
- a base filtering rule.

The base rule is then used to pre-filter the cache file to reduce its volume significantly. This initial pre-filtering is needed because the goal is to analyze only the traffic that has exceeded preset thresholds, specified by the base rule.

2. Calculating cutoff thresholds

After the cache is filtered and prepared for further processing, the algorithm calculates the total bit and packet rates it contains and a cutoff value, a new type of threshold, based off the baseline and threshold received in the initial step, which the algorithm uses to determine whether a sufficient volume of attack traffic has been identified using a custom formula:

$$\text{cutoff} = (T - B) \cdot \left(\frac{T/B}{(T/B) + 8} \right) + B \tag{4.1}$$

where T is the threshold and B is the baseline. The constant ensures that the desired cutoff will be exactly in the middle between the baseline and the threshold when the threshold is eight times larger than the baseline. This formula and constant eight have proven to yield the most consistent and accurate results during early testing of the tool, along with its implementation, but are subject to further refinement while the tool is deployed in the Brno University of Technology network and more DDoS data are being collected.

3. Flow aggregation

The algorithm then begins flow aggregation. It checks whether there is a large enough flow such that the cache volume would decrease below the cutoff value if the flow were removed. If no single flow aggregate meets this condition, the algorithm proceeds to a step-by-step evaluation of the traffic characteristics. It prioritizes a list of specified traffic attributes, such as source and destination IP addresses, ports, protocol types, or TCP flags. These are assessed in a defined order, starting with required attributes.

4. Attribute-based narrowing

The evaluation proceeds by analyzing traffic grouped by attributes such as:

- Source/destination IP,
- Source/destination port,
- Protocol type,
- TCP flags.

At each step, traffic is aggregated by a set of attributes from the specified attribute list, starting with the first attribute. Once the flows are aggregated, the contribution of each aggregate is measured in terms of packet and bit rates. If any aggregate stands out as the dominant contributor, the latest attribute is also included in the next iteration of the data aggregation.

The graphic representation of this iterative part of the algorithm is shown in Figure 4.4 below.

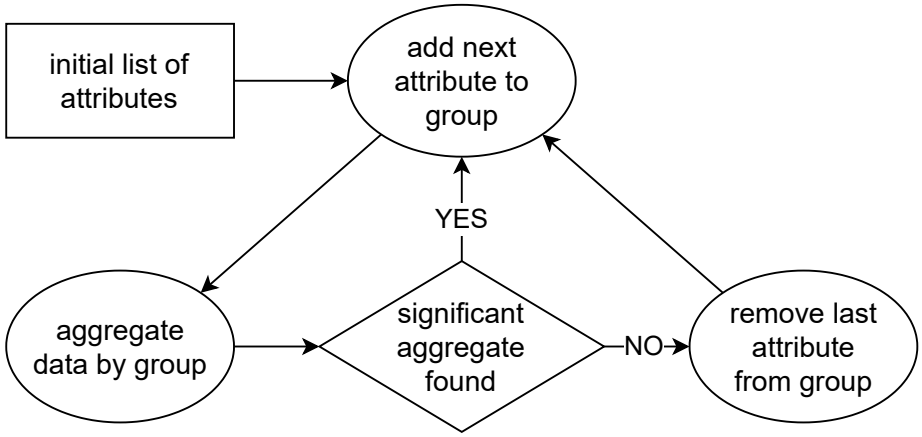

Figure 4.4: Scheme of iterative narrowing of flow cache

Suppose aggregation after adding a particular attribute to the grouping set does not reveal a significant contributor. In that case, the algorithm checks whether grouping traffic by broader patterns, such as subnet-level aggregation for IP addresses, can reveal any significant contributors. However, if this broader approach still does not reveal any dominant group, the attribute is removed from the grouping set of attributes and is left out of the following iterations. This narrowing continues attribute by attribute, gradually isolating the most specific source of abnormal traffic, which is still large enough to satisfy the cutoff threshold.

5. Finalization of the analysis

Once all possible attributes have been exhausted, the algorithm selects the top group responsible for the traffic spike, returning its defining attributes (source IP, destination IP, used ports, protocol, and, in case of TCP traffic, also tcpflags) as the final filtering rule back to DDoS-Guard. This output can then be used directly for attack mitigation or further investigation.

The complete algorithm is visualized in Figure 4.5.

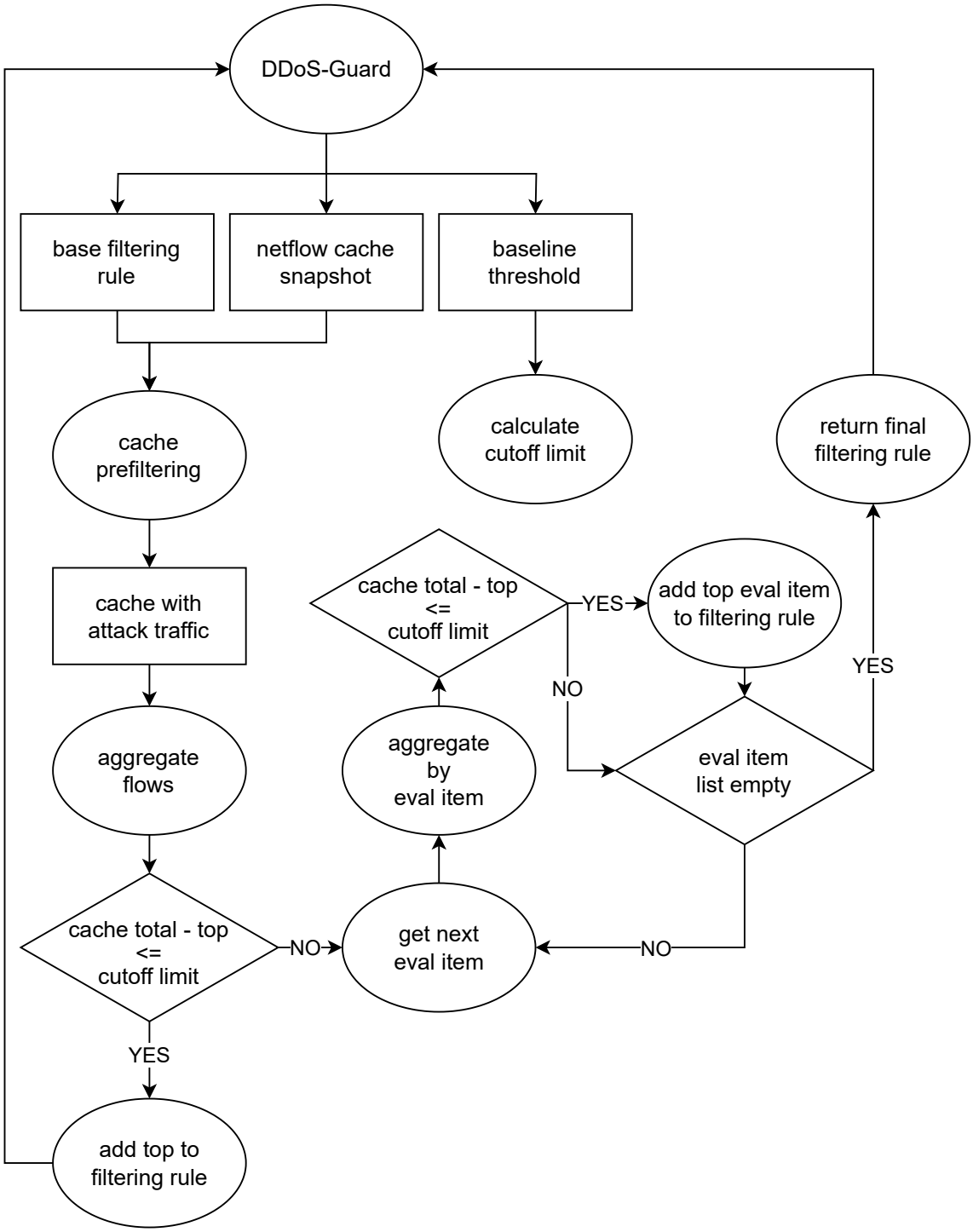

Figure 4.5: Step by step scheme of the detection algorithm

Chapter 5

Implementation

This chapter focuses on how the detection algorithm was developed and tested. The section *Realization of the Detection Algorithm* goes over the main parts of the implementation, including what tools were used and how the logic was translated into code. The following section, *Testing*, describes how the implementation was tested, what data were used, and how the results were evaluated. Section *Deployment* mentions the current state of the tool and its integration and deployment in the university network.

5.1 Realization of the Detection Algorithm

The algorithm is implemented in Python and built around the Pandas library for data manipulation. Flow data is initially extracted from a cache file using AWK-based filter conditions corresponding to the base filtering rule provided by DDoS-Guard. These conditions are translated into an AWK expression, allowing efficient pre-filtering of large cache files before further processing. The AWK expression is then executed on the cache file using the Python module Subprocess, returning the filtered cache contents.

Algorithm 1: AWK-based Pre-filtering

Input: Base filtering rule, cache file
Output: Filtered cache contents

```
awk_conditions ← {};
foreach flow parameter in base rule do
    condition ← convert parameter to AWK expression;
    Append condition to awk_conditions;
end
awk_command ← join awk_conditions into AWK command;
filtered_cache_contents ← execute awk_command on cache file;
return filtered_cache_contents;
```

Initially, pre-filtering was performed after the data had been loaded into a DataFrame structure, using Pandas DataFrame filtering capabilities. Switching to pre-filtering using AWK before creating a DataFrame resulted in a significant improvement in script runtime, outlined in Figure 5.1. This optimization has led to an average of approximately 50 % decrease in script runtime among all cache files tested.

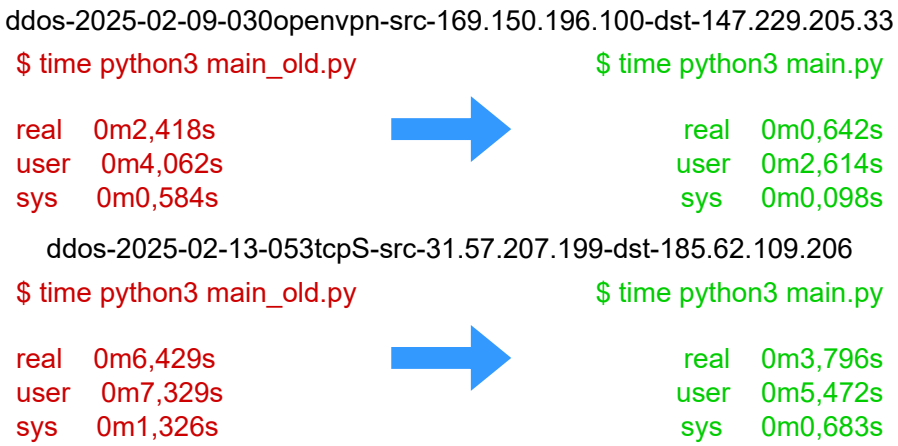

Figure 5.1: Script runtime difference before and after AWK pre-filtering optimization

Once the relevant data is filtered, they are converted into a structured DataFrame. The script extracts specific fields such as source and destination IPs, ports, protocols, TCP flags, flow timestamps, and flow metrics such as packet and byte counts. These values are parsed and cast into appropriate data types for efficient memory use.

Next, the script computes the filtered cache's packet and bit rates, taking the total cache volume and dividing it by the time frame from the first to the last recorded flow. Then, it calculates custom cutoff limits using the provided baseline and threshold values. These limits are derived through a formula that balances sensitivity and tolerance to traffic spikes mentioned in section 4.3.1.

Once cutoff limits are calculated, flows with identical characteristics are grouped by pandas.DataFrame.groupby and their packet and bit rates are summed up. The top of the DataFrame is then checked to see whether it is large enough to be flagged as the cause of the abnormal traffic increase, and if so, it is returned as the filtering rule to the DDoS-Guard.

If no such flow aggregate is found, the script initiates a step-by-step analysis, outlined in the algorithm 2. All data manipulation is performed using Pandas.DataFrame methods and filtering capabilities. The IP address truncation and prefix aggregation are encapsulated in a standalone method, but for easier understanding, I decided to break it down in the algorithm.

Algorithm 2: Step-by-step Flow Attribute Evaluation

Input: Filtered flow DataFrame `df`, cutoff values, list of traffic attributes for evaluation `eval_items`

Output: Aggregated DataFrame isolating the most significant traffic contributor

```
grouping_list ← [ ];
foreach item in eval_items do
    append item to grouping_list;
    df_temp ← pandas.df.groupby(grouping_list), summing packet and bit rates;
    foreach metric in [packet rate, bit rate] do
        Sort df_temp by metric and check for significance;
        if significant top of df_temp then
            Narrow df to flows matching the top attributes;
            continue outer loop;
        end
    end
    if item is srcip or dstip then
        for prefix_length in [24, 16, 8] do
            Truncate IP addresses to fit prefix_length;
            Group traffic by IP prefixes and check for significance;
            if significant top of df_temp then
                replace item with prefix, extended by asterisks to fit 4 octets
            end
        end
        continue;
    end
    remove item from grouping_list;
end
if grouping_list is not empty then
    Compute unique counts for remaining items not in grouping_list;
    df ← pandas.df.groupby(grouping_list), summing packet and bit rates, and
     adding unique counts;
    return df;
end
return None;                                  // No significant aggregation found
```

After the analysis concludes, the final aggregated DataFrame is checked to see if it contains all attributes marked as „required“, and all the attributes are then returned in a structured format for DDoS-Guard to put in use.

The implementation also includes verbose logging using the Python module logging. Every step of data manipulation and any significance test or intermediate result is logged into a text file with additional information. The logging helps to track the analysis progress – which attributes are considered, which are discarded, traffic volume contained in the cache, what cutoff values were used, and any information that might help with understanding the algorithm's decisions regarding the final filtering rule it has created.

5.2 Testing

This section outlines the test results of the developed tool and some obstacles encountered along the way. The tool has been tested on several NetFlow cache records, retrieved by DDoS-Guard from live traffic during an ongoing DDoS attack on the Brno University of Technology network. The testing was performed locally on a Linux Ubuntu 22.04 machine. Further testing will occur while the tool is deployed in live traffic.

5.2.1 Testing attack scenarios

This section presents and evaluates the results of the testing of the developed tool on selected cache files. I have chosen three of the most distinct recorded attacks with notable results.

1. Attack scenario

The first selected attack was an OpenVPN UDP flood. The attack was quite simple, with just one source, one destination IP address, and static source and destination ports. The attack parameters, configuration, and evaluation are outlined in Table 5.1 and its progress in Figure 5.2.

Rule	`openvpn`	
	packet/s	**bit/s**
Baseline	4.5k	46.4M
Threshold	50.0k	500.0M
Eval items	proto srcip dstip tcpflags srcport dstport	
Required items	proto dstip	
DDoS-Guard result	udp from 169.150.196.100:1194 to 147.229.205.33:36303	
My result[1]	udp from 169.150.196.100:1194 to 147.229.205.33:36303	

Table 5.1: DDoS attack parameters and evaluation

In this case, the cache aggregation revealed a single, clearly dominant flow, making the analysis straightforward and fast. However, the cache may have been created slightly late – missing a portion of the attack due to export timeouts, and thus not exceeding the predefined threshold for its analysis. For better understanding, the exact values retrieved from logging were: `INFO - cache total (pps/bps): 35652.08 409296519.03`. This means the cache contained roughly 28.3 % lower packet rate than the thresholds set by DDoS-Guard. The difference is even more significant compared to the actual peak of the attack in Figure 5.2. However, the volume in the cache was still sufficient to identify the malicious pattern correctly.

[1]The result of my tool is a Python dictionary, such as `{'srcip':'169.150.196.100',...}`. The protocol is represented by the IP protocol number (in this case, 17 for UDP). For better readability and comparison, it is displayed in the same format as the DDoS-Guard result in the table.

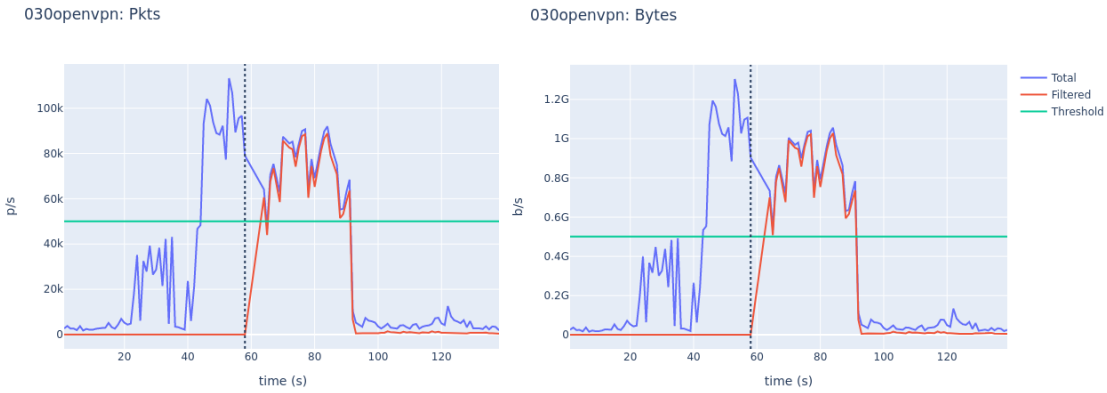

Figure 5.2: DDoS-Guard record of OpenVPN (source port 1194) attack progression

2. Attack scenario

The second selected attack was a TCP SYN flood. Multiple attacks were detected in quick succession at the time of this attack, which likely resulted in the discrepancy between DDoS-Guard's and my results, shown in Table 5.2 below.

Rule	`tcp syn`	
	packet/s	**bit/s**
Baseline	23.1k	9.7M
Threshold	69.4k	50.0M
Eval items	srcip dstip tcpflags srcport dstport	
Required items	srcip	
DDoS-Guard result	from 31.57.207.199 to 185.62.109.206:443 TCPflags SYN	
My result	from 5.39.202.68 to 185.62.109.206:443 TCPflags SYN	

Table 5.2: DDoS attack parameters and evaluation

The table shows that DDoS-Guard recognized a different IP address than my tool as the attack source. There were two TCP SYN attacks shortly after each other. The DDoS-Guard report showed that both of these attacks shared the same source and destination IP addresses. However, the cache analysis of the first attack revealed two significant contributors with the same destination IP address:

dstip	dstport	pps	bps	unique_srcip	unique_srcport
185.62.109.206	443	84744.575	2.712206e+07	2	65536

Table 5.3: Cache analysis log showing additional data about the identified attack

The number of unique source ports suggests that the traffic was flagged correctly as a DDoS attack. However, two separate, simultaneous attacks targeting the same destination IP address seem rather coincidental. This could mean that DDoS-Guard has detected an attack from the IP address `31.57.207.199` , and at some point, the attacker changed the

source IP address of the attack to `5.39.202.68` . The delayed timing of the creation of the cache snapshot then allowed me to reveal the other attack source, while DDoS-Guard stuck with the early identification of the source IP address for both detected attacks.

However, the second traffic spike in Figure 5.3 indicates that the latter revealed source IP address was in addition to the former, rather than replacing it, suggesting that there were indeed two simultaneous attacks on the same destination IP address.

In either case, this suggests that the cache analysis could be performed at multiple points during an ongoing DDoS attack to adjust the filtering rule in response to potential changes in the attack pattern.

Figure 5.3: DDoS-Guard record of TCP syn attack progression

3. Attack scenario

The third DDoS attack was a DNS flood. This was a large-scale DDoS attack, likely involving a botnet sending traffic from a large number of IP addresses to just one destination address.

Rule	`dns`	
	packet/s	**bit/s**
Baseline	3.2k	15.4M
Threshold	50.0k	500.0M
Eval items	proto srcip dstip tcpflags srcport dstport	
Required items	proto dstip	
DDoS-Guard result	udp from port 53 to 185.62.109.206:53	
My result	udp from port 53 to 185.62.109.206:53	

Table 5.4: DDoS attack parameters and evaluation

This attack was the most significant DDoS attack on which the tool was tested. The attack volume reached about 800,000 packets per second and more than 8 gigabits per second, shown in Figure 5.4. The log records again show that the exported cache did not contain the full attack capacity, as outlined in Table 5.5.

proto	dstip	srcport	dstport	pps	bps	unique_srcip
17	185.62.109.206	53	53	290018.975	3.213456e+09	140349

Table 5.5: Cache analysis log showing additional data about the identified attack

This again suggests that the timing of the cache export might not have been perfect and that it was probably done earlier or later than ideal. However, the attack was so large that it did not affect the analysis. The log data also shows that the attack originated from more than 140,000 unique source IP addresses, pointing towards the use of a botnet.

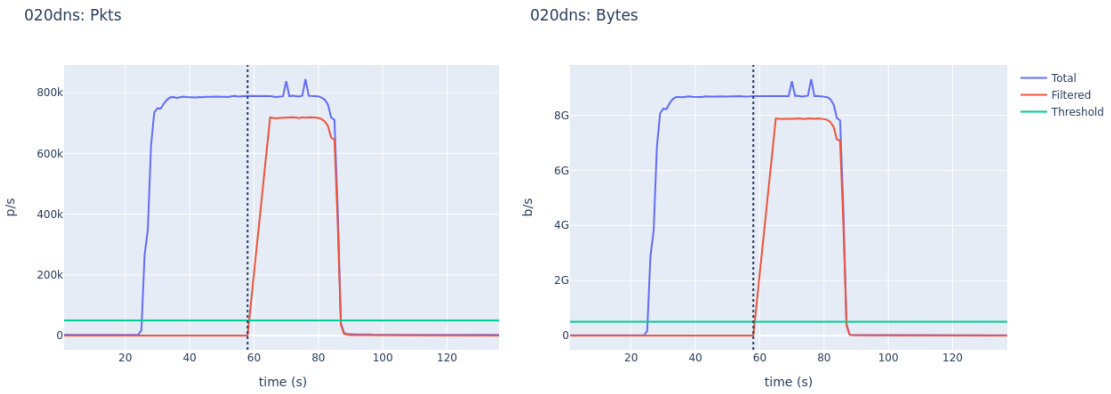

Figure 5.4: DDoS-Guard record of DNS (port 53) attack progression

5.3 Testing obstacles

During the early testing, I was dealing with some difficulties. One of the main ones was the timing of cache export. The very first cache files were exported too late, after all of the attack traffic was fully exported to the collector. This issue was later improved, but still persisted as mentioned in the previous section, resulting in the exported cache files not having the full capacity of attacks recorded by DDoS-Guard. That may or may not impact the final results of future cache analyses, depending on the total scale of the recorded attack.

This issue might potentially be eliminated entirely when the tool is fully integrated into the DDoS-Guard system, resulting in better timings for exporting the cache directly into the tool.

5.4 Deployment

The tool has been integrated into the DDoS-Guard and is currently actively deployed in the Brno University of Technology backbone network. Some issues have arisen during the integration and are steadily being worked on and fixed as new issues appear with the variety of captured traffic.

Currently, there are no notable DDoS data from deployed cache analysis, and we are waiting for more DDoS traffic to come.

Chapter 6

Conclusion

This thesis presented a comprehensive approach to detecting DDoS attacks using NetFlow cache analysis, specifically within the context of the Brno University of Technology network. The work began with an in-depth examination of the NetFlow protocol and its role in modern network traffic monitoring. A wide taxonomy of DDoS attacks and their associated defense mechanisms was also outlined, with an emphasis on current threat trends and the limitations of traditional mitigation strategies.

The main contribution of the thesis is the development of a Python-based tool that analyzes NetFlow cache directly from the ipt-netflow exporter, allowing the detection of malicious traffic without relying on delayed exported flow data. The implemented algorithm applies a step-by-step narrowing approach to aggregate traffic by key attributes, such as destination IP, port, and protocol, until dominant components of a potential attack are isolated. The resulting filtering rules are constructed dynamically and returned in a form suitable for immediate deployment within the defense system.

The developed tool was tested on real-world DDoS attack data retrieved from the Brno University of Technology network. These datasets included volumetric UDP floods, TCP SYN floods, and other common flooding patterns. The testing confirmed the tool's ability to accurately identify dominant traffic components responsible for attack behavior and to generate filtering rules based on live NetFlow cache snapshots. Following successful testing, the tool was integrated into the DDoS-Guard infrastructure, allowing it to interface with the existing mitigation system.

While full-scale evaluation within the production environment is still ongoing, initial testing indicates that the tool operates reliably and efficiently under realistic conditions. Its lightweight design and real-time analysis capabilities suggest strong potential for deployment in active defense workflows, complementing existing detection and mitigation mechanisms.

Future improvements may focus on better control of cache export timing, adaptive rule generation, and potentially more advanced traffic analysis using statistical or machine learning methods. Testing the tool in larger environments would help validate its scalability and effectiveness beyond the university setting.

Bibliography

[1] AABC. *Ipt-netflow* online. August 2020. Available at: https://github.com/aabc/ipt-netflow. [cit. 2024-10-30].

[2] AITKEN, P.; CLAISE, B. and TRAMMELL, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* RFC 7011. RFC Editor, september 2013. Available at: https://doi.org/10.17487/RFC7011.

[3] ARBOR, N. *Arbor DDoS Protection Solutions* https://www.netscout.com/arbor. 2024. [cit. 2025-05-12].

[4] CISCO. *NetFlow Export Datagram Format* online. September 2007. Available at: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html. [cit. 2024-10-30].

[5] CISCO. *NetFlow Version 9 Flow-Record Format* online. Cisco, may 2011. Available at: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html. [cit. 2024-11-5].

[6] CISCO. *NetFlow Configuration Guide, Cisco IOS Release 15M&T* online. 2016. Available at: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/netflow/configuration/15-mt/nf-15-mt-book.pdf. [cit. 2024-10-28].

[7] CLAISE, B. *Cisco Systems NetFlow Services Export Version 9* RFC 3954. RFC Editor, october 2004. Available at: https://doi.org/10.17487/RFC3954.

[8] CLOUDFLARE. *UDP flood attack* online. Cloudflare. Available at: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/. [cit. 2024-12-06].

[9] CLOUDFLARE. *What is a DNS flood? | DNS flood DDoS attack* online. Cloudflare. Available at: https://www.cloudflare.com/learning/ddos/dns-flood-ddos-attack/. [cit. 2024-12-13].

[10] CLOUDFLARE. *What is the low orbit ion cannon (LOIC)?* online. Cloudflare. Available at: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/low-orbit-ion-cannon-loic/. [cit. 2024-11-13].

[11] CLOUDFLARE. *Rate Limiting and DDoS Protection* https://www.cloudflare.com/resources/assets/slt3lc6tev37/2hIapovmEBdhDq0DLCuwDR/3691243ee090906900ba1a8dce7ddd45/Two_Pager_Rate_Limiting_Letter_EN-US.pdf. 2024. [cit. 2025-05-12].

[12] Dzurenda, P.; Martinasek, Z. and Malina, L. Network Protection Against DDoS Attacks. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, march 2015, vol. 4.

[13] FastNetMon. *FastNetMon – DDoS Detection* https://fastnetmon.com/. 2024. [cit. 2025-05-12].

[14] Hofstede, R.; Čeleda, P.; Trammell, B.; Idilio Drago, R. S.; Sperotto, A. et al. *Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX* online. IEEE Commun. Surveys Tuts., may 2014. Available at: https://is.muni.cz/publication/1181098/flow-monitoring-explained-paper.pdf. [cit. 2024-10-27].

[15] IBM. *What is NetFlow?* online. IBM. Available at: https://www.ibm.com/topics/netflow. [cit. 2024-11-5].

[16] Khan, R.; Maynard, P.; McLaughlin, K.; Laverty, D. and Sezer, S. Threat Analysis of BlackEnergy Malware for Synchrophasor based Real-time Control and Monitoring in Smart Grid. *Electronic Workshops in Computing*. BCS Learning Development, 2016, p. 53 – 63. ISSN 1477-9358. Available at: http://dx.doi.org/10.14236/ewic/ICS2016.7.

[17] Marchesseau, M. *"Trinity" – distributed denial-of-service attack tool* online. SANS Institute, september 2000. Available at: https://www.giac.org/paper/gsec/123/trinity-distributed-denial-service-attack-tool/100510. [cit. 2024-11-13].

[18] Mirkovic, J. and Reiher, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.* New York, NY, USA: Association for Computing Machinery, april 2004, vol. 34, no. 2, p. 39–53. ISSN 0146-4833. Available at: https://doi.org/10.1145/997150.997156.

[19] Mölsä, J. Mitigating denial of service attacks: A tutorial. *Journal of Computer Security*. IOS Press, 2005, vol. 13, no. 6, p. 807–837. Available at: https://aaltodoc.aalto.fi/server/api/core/bitstreams/b024083c-6402-4e84-99e1-2ddaa3c88ad0/content.

[20] ntop. *NScrub – DDoS Mitigation at Line Rate* https://www.ntop.org/products/ddos-mitigation/nscrub/. 2024. [cit. 2025-05-12].

[21] Odintsov, P. *FastNetMon* https://github.com/pavel-odintsov/fastnetmon. 2024. [cit. 2025-05-12].

[22] Praetox Technologies. *Low Orbit Ion Cannon* online. Available at: https://github.com/NewEraCracker/LOIC. [cit. 2024-11-13].

[23] Puri, R. *Bots and Botnet: An Overview* online. SANS Institute, august 2003. Available at: https://sansorg.egnyte.com/dl/s9RHzYWkNe. [cit. 2024-11-12].

[24] Radware. *Cloud DDoS Protection Services* https://www.radware.com/products/cloud-ddos-services/. 2024. [cit. 2025-05-12].

[25] Radware. *DefensePro – Real-Time Network and Application Attack Mitigation* https://www.radware.com/products/defensepro/. 2024. [cit. 2025-05-12].

[26] Senie, D. and Ferguson, P. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing* RFC 2827. RFC Editor, may 2000. Available at: https://doi.org/10.17487/RFC2827.

[27] Taghavi Zargar, S.; Joshi, J. and Tipper, D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys amp Tutorials*, november 2013, vol. 15, p. 2046 – 2069.

[28] Yoachimik, O. and Pacheco, J. *4.2 Tbps of bad packets and a whole lot more: Cloudflare's Q3 DDoS report* online. Cloudflare. Available at: https://blog.cloudflare.com/ddos-threat-report-for-2024-q3/. [cit. 2024-12-13].

[29] Yoachimik, O. and Pacheco, J. *DDoS Reports* online. Cloudflare. Available at: https://blog.cloudflare.com/tag/ddos-reports/. [cit. 2024-12-13].