



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

THESIS FOR SCIENTIFIC STUDENTS' ASSOCIATIONS

DEPLOYMENT OF A RESEARCH T-POT SYSTEM IN A CLOUD INFRASTRUCTURE

Author(s): **Bánfi Konrád Koppány**
Computer science Bsc, VII. semester

Internal advisor: **Balázsné Dr. Kail Eszter**
university professor
deputy institute director

Budapest, 2024. 11. 04.

ABSTRACT

Honeypots, intentionally vulnerable systems designed to deceive attackers, offer a unique opportunity to study adversarial behavior within networks. They provide insights into attack techniques, motivations, and evolving threat landscapes. The research will explore honeypot technology, deployment strategies, and their ability to enhance threat intelligence. The goal of this research is to gain in-depth knowledge of honeypots, investigate attacker profiling techniques, and build a honeypot environment to discriminate between human and automated threats. The deployed system will be tested and evaluated for its efficacy in gathering attacker data and supporting behavioral analysis.

The aim of the research is to show why the development of current honeypot technologies and IDS solutions is crucial for further understanding new attacker methodologies. I firmly believe that this topic is relevant because the weaponization of bots has become a major threat in cyberspace, and up-to-date honeypot technologies can help us in bot behavior analysis.

TABLE OF CONTENTS

1	INTRODUCTION	3
1.1	Research aims.....	3
1.2	Research structure	3
2	LITERATURE REVIEW ON HONEYPOT SYSTEMS	5
2.1	The classification of honeypot systems.....	5
2.2	Privacy issues regarding to data collection and data retention	6
2.3	Potential dangers of implementing a honeypot system.....	7
2.4	The comparison of honeypot and IDS technologies	7
2.5	Attacker profiling methodologies	8
3	T-POT IMPLEMENTATION	9
3.1	Specification of T-Pot systems.....	9
3.2	Implementation of T-Pot	9
3.3	Implementation of Suricata IDS.....	14
4	DATA ANALYSIS WITH MACHINE LEARNING TOOLS	16
4.1	Evolution of bot attacks	16
4.2	Classification of Denial of Service attacks	17
4.3	Evaluation of datasets	18
4.4	Classification of machine learning models	20
4.5	Implementation of a supervised machine learning model.....	23
4.5.1	Feature selection and data preprocessing.....	24
4.5.2	Training and evaluation	25
4.5.3	Deployment of the trained model	30
5	THE INFRASTRUCTURE AS CODE (IAC) PARADIGM	33
5.1	Introduction to the IaC paradigm	33
5.1	Implementation of IaC	33
6	RESEARCH GAPS	34

6.1	Short-term aims	37
6.2	Long-term aims	37
7	SUMMARY	36
8	BIBLIOGRAPHY	37
9	LIST OF FIGURES	38

1 INTRODUCTION

1.1 Research aims

In the constantly evolving cyberspace, organizations continuously face new threats. The Intrusion Detection Systems (IDS) and other conventional solutions are still the building block of a company's defense mechanism, but complementary tools become more and more necessary to implement. Among these, honeypot solutions have emerged as powerful tools for further understanding and countering cyber-attacks, offering capabilities that complement the conventional security systems. However, the implementation of these deception-based technologies has its own challenges, from secure deployment and system maintenance to appropriate data collection and storage. The security of modern infrastructures increasingly relies on automation and machine learning tools, from bot detection algorithms to ML based network traffic analysis. The Infrastructure as Code (IaC) paradigm enables rapid deployment and consistent configuration of security systems at scale. This research explores how the currently used security tools can be extended with honeypot implementations such as T-Pot, and examines how these technologies can be effectively deployed and managed. Furthermore, it investigates the role of machine learning in enhancing security against modern attack methodologies. As cyber threats continue to evolve, understanding these technologies and their practical implications becomes crucial for organizations seeking to build robust security systems. This investigation aims to provide insights into both the technical and operational aspects of implementing advanced security solutions while considering legal, ethical, and practical constraints.

1.2 Research structure

The current research has three main pillars. First, the implementation and configuration of an operational T-Pot system (which includes a built-in Suricata IDS). Second, the training and testing of multiple machine learning models (Long short-term memory) based on the input logs provided by the IDS and other T-Pot services. Comparing and manually evaluating the different results of the current IDS configuration, the machine learning model can help us build more sophisticated and efficient tools for classification between bot and human behavior. Either the training and the evaluation of the model can be improved, or the ruleset of the signature-based IDS can be changed. Third, with the help of Infrastructure as code (IaC) paradigm we can deploy, modify and delete the infrastructure at scale. This paradigm enhances automation and reproducibility. When it comes to honeypot deployment, IaC is a crucial component because such a sensitive research area requires leaving out as many human errors as possible.

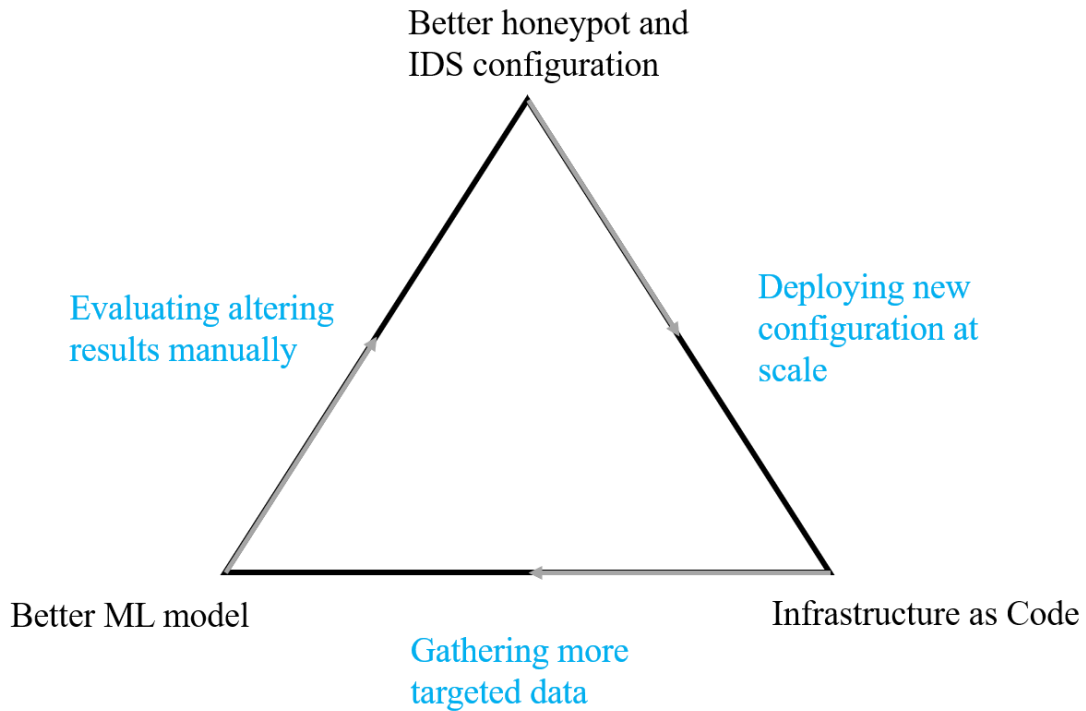


Figure 1.1: Abstract pipeline representation.

After the deployment of multiple honeypot nodes, the quality and diversity of incoming dataflows increase, which can result in the training of more robust ML models. The models evaluate the incoming dataflows in a central node, by which the signature-based reconnaissance and DDoS detection rules can be modified accordingly.

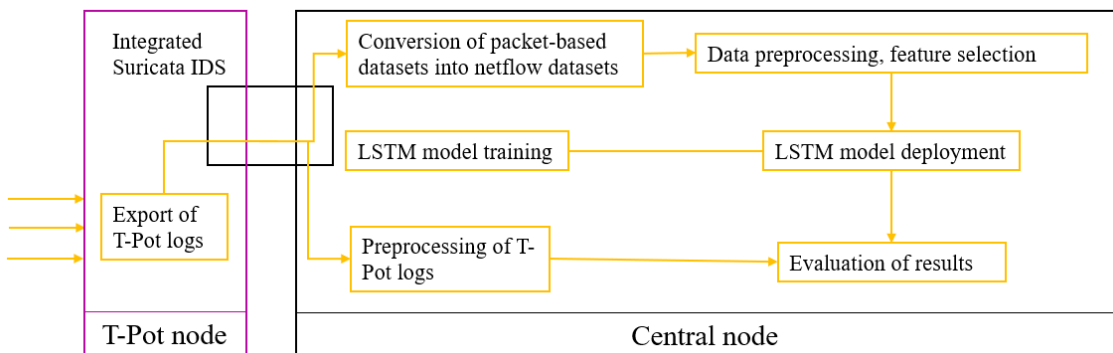


Figure 1.2: Pipeline in case of a single honeypot node.

2 LITERATURE REVIEW ON HONEYPOT SYSTEMS

2.1 The classification of honeypot systems

Attackers are always one step ahead. Honeypots, systems that are deliberately allowed to be compromised, give us an opportunity to interact with the attacker entities and therefore get to know new attack methodologies. In cybersecurity, honeypots play a crucial role in the collection and comprehensive analysis of data. The collected data can later on be evaluated, which helps the continuous development of cybersecurity solutions, or the mitigation of emerging threats, contributing to a more safe and resilient cybersecurity space. However, the maintenance of deception is a constant challenge, new emulated services must be configured in new ways to attract the attacker entities. This research highlighted three main aspects of the classification[1] of honeypot systems. The first aspect is the level of interaction, which can be defined as the range of possibilities a honeypot allows an attacker to have. Low-level interaction honeypots provide emulated services which are best suited for analyzing and understanding reconnaissance from the incoming network flow. On the counterpart, high-level interaction honeypots provide an emulated operating system shell or other system UI, which gives us an opportunity to understand and analyze how attackers take advantage of given exploits or what type of malware they use. Different honeypot interaction types are more suitable for different security aims and resource limitations than others. For instance, a low-interaction honeypot might be an optimal solution for a small network, which must address resource-constrained limitations. These honeypot types simulate services with limited functionality. While they may not provide as much data as high-interaction honeypots, the maintenance and deployment is significantly easier. In contrast, high-interaction honeypots could be deployed in a more complex environment with abundant resources and they provide more detailed and subtle information about the delivery of the attack.

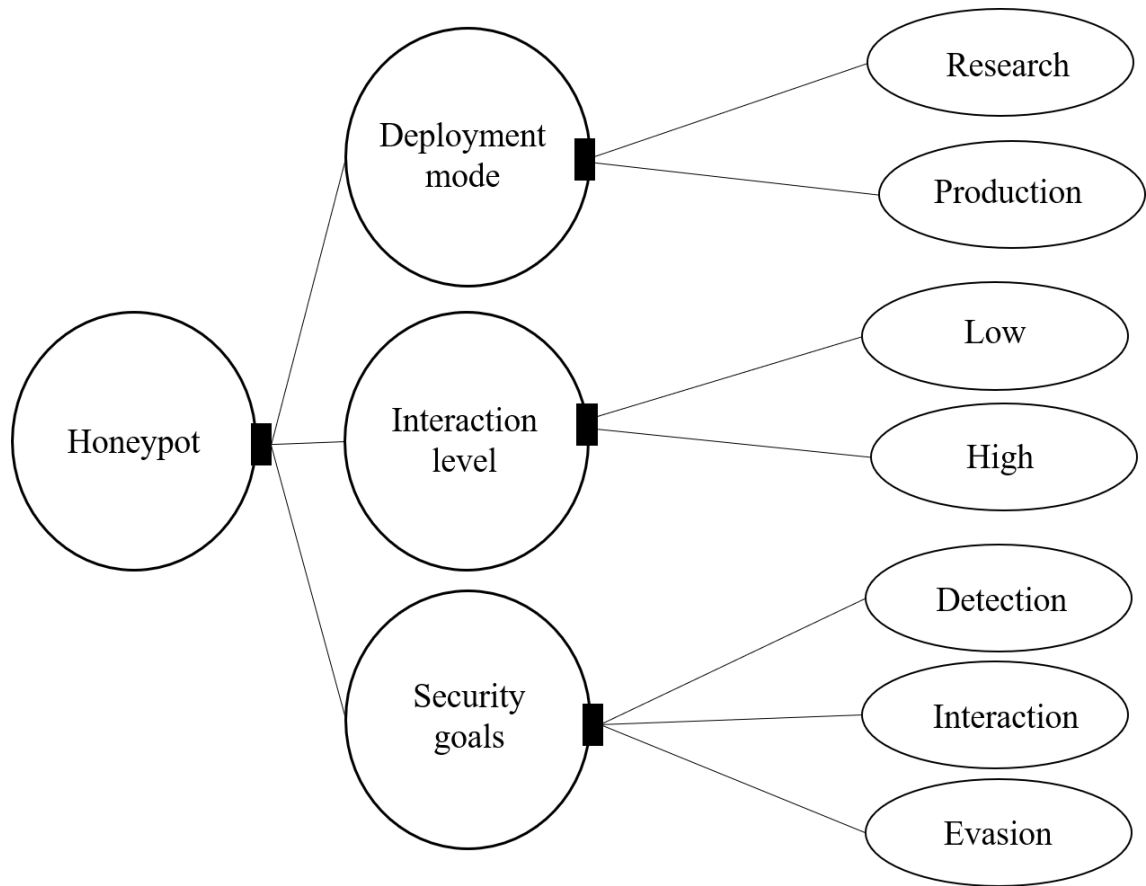


Figure 2.1: Classification of honeypot systems.

The second aspect of the classification is the mode of deployment, which can be either a research or a production honeypot system. Research honeypots are designed for data collection and analysis purposes. These tools are used for gathering information about attackers' techniques, tactics, and procedures. To the contrary, production honeypots are integrated into live, operational networks. Their main function is to actively engage attackers away from critical systems, acting as decoy systems to protect legitimate targets.

The third aspect of the classification can be defined based on the aim of deployment. As a deception tool, honeypots include the capability for intrusion detection, evasion and interaction with the attacker, but based on the customized configuration the focus may shift to one of these three areas.

2.2 Privacy issues regarding to data collection and data retention

Honeypot systems increase our understanding of malicious activity in cyberspace. However, the deployment and the usage of these tools are influenced by a number of technical and legal issues, which need to be carefully considered. We have to keep in mind that there are legal issues regarding the data capturing and the data retention provided by the deployed honeypot system. The most important questions regarding to this topic are:

- What type of data is legally allowed to be collected by honeypots?
- What are the legal conditions for the data collection and data retention?

The legal ground must be chosen according to the interaction level and the deployment mode of the honeypot system. In the case of a production honeypot, it is in the best interest of the enterprise to collect and retain and analyze data for the future protection of their products, services, infrastructure and intellectual property. The possible harm of privacy for the data subject is very little. Therefore, the processing of personal data (for example the public IP address) is acceptable according to article seven of the EU Data Protection Directive. In the case of research honeypots, the situation is more complicated. The researchers' legitimate interest can be the promotion and analysis trends in cybersecurity. These interests must be proportionate with the right of data subjects for privacy protection in the light of the possible harm done by the processing. Since the possible harm is quite low, there is a legal ground for collecting and processing the data as well, referencing the same legal article applied on production honeypots[2].

2.3 Potential dangers of implementing a honeypot system

At the designing phase of a honeypot system deployment, the aim must be to minimize the risks. If the data collected and provided by a honeypot system is inaccurate, then the further development of any cyber security tool will not be effective, future products will detect false positive and false negative intrusions or interactions in the incoming network flow. In other cases, the attacker entity can detect the purpose of the deployed system and use it for discovery and fingerprinting by which our infrastructure can be mapped out due to poor firewall, VLAN or IDS configurations. Furthermore, once entering the shell, the attacker can access the production infrastructure.

2.4 The comparison of honeypot and IDS technologies

In general, intrusion detection systems operate in two ways. Signature-based techniques compare traffic patterns to known attack signatures meanwhile behavior-based techniques detect intrusions via deviations from normal or expected traffic behavior. Both techniques possess limitations: signature-based techniques cannot guard against unknown intrusions, and behavior-based techniques break down under heavy traffic or sudden traffic bursts. Also, signature-based botnet detection methods are considered to be computationally expensive. They are not scalable on high-speed networks. The evolution of machine learning algorithms and their implementations in IDS systems has helped immensely to overcome these challenges and limitations.

If we compare intrusion detection systems with honeypot systems, we can declare that they both aim for intrusion detection and data collection. However, the way they approach it might differ. In brief terms, the sole purpose of honeypot systems is interaction with the attacker. Therefore, the amount of false positive alarms is reduced to almost zero. At the same time, due to these continuous interactions, more data is generated that might be

noisy, hard to gather and filter in a more meaningful way. Due to its stateful qualities, intrusion detection systems might be better for traffic analysis but they might produce more false positive alarms and are not as successful against the detection of slow-scan and stealth scan attacks. The best approach is to apply both systems. Honeypot systems and Intrusion Detection Systems can complete each other if successfully implemented, resulting in low level false negative and false positive alarms.

2.5 Attacker profiling methodologies

There are several attacker profiling methodologies used in cybersecurity to categorize and understand potential threats. The key motivation behind attacker profiling is threat prediction and prevention by understanding the attacker's motives and methodologies. Based on that the defensive parties can create a defense strategy. The following is a brief list of example frameworks by which such profiling [3] can be made:

- capability-based profiling: it classifies attackers based on their skill level and their resources (for example: script kiddies, organized crime groups, nation-state actors)
- motivation-based profiling: it categorizes attackers based on their motivations (for example: financial gain, ideology)
- attack vector profiling: it focuses on the methods and entry points the attackers use (for example: social engineering, malware, network vulnerabilities)
- threat actor groups: it identifies and tracks specific known hacking groups or organizations (APTs)
- MITRE ATT&CK framework: it provides a comprehensive matrix of attacker tactics, techniques, and procedures (TTPs)
- Kill Chain Model: it breaks down the attack process into stages, from reconnaissance to exfiltration.

3 T-POT IMPLEMENTATION

3.1 Specification of T-Pot systems

T-Pot is an open-source, multiarchitecture (amd64, arm64) honeypot platform that uses the Debian 11 "bullseye" base image file. A strong community is developing it continuously, therefore it supports 20+ integrated honeypot services, many visualization options and lots of security tools to further improve the deception experience. T-Pot is lightweight when it comes to resource and implementation requirements due to the containerized, emulated services. All the services are managed via a *docker-compose.yml* file. The utilization of docker services and (autoheal) enhances portability, reproducibility, easier system maintenance and incident response. T-Pot also provides integrated monitoring and visual display of logs. The current research includes the implementation of T-Pot version 24.04.0, the latest stable at the time.

During the research the following integrated T-Pot services were used mainly:

- Cowrie,
- DDoSPot
- Endless,
- Honeytrap,
- Suricata: IDS,
- Elastic Stack,
- T-Pot-Attack-Map.

3.2 Implementation of T-Pot

The research included a demo T-Pot system deployment in the AWS cloud environment, later a deployment in the Hun-Ren OpenStack cloud infrastructure, under the Cybersecurity Research Lab project. In order to make the instance possibly more attractive towards intruders, the random-generated T-Pot hostname was changed to *cs30-db-prod* after the T-Pot installation. Initially the aim was to test how effectively T-Pot can detect scanning and reconnaissance attacks. Studying the current trends in cybersecurity and analyzing the logs provided by T-Pot, it showed that the attackers are using automated scripts and IP rotation techniques to avoid attacker profiling and IP blacklisting. The scope of the research has shifted towards the differentiation of manual and automated bot attacks with the help of machine learning models.

The more nodes are deployed the more diverse dataset can be collected which can be used for future evaluation. Based on the evaluation, changes in IDS configuration, hardening of firewall and other security policies can be made. To achieve this goal, the periodic export of the incoming logs to a central node is crucial for preprocessing and evaluation purposes. During the implementation, a mounted NFS server was used (attached as a separate storage device). With the help of Linux cronjobs, the export shell

script ran periodically. This current implementation did not include the configuration of static routing between the NFS serving interfaces, nor the periodic snapshot creation of the storage device itself. However, before any update or crucial change in configuration, a snapshot was made both from the honeypot and the central node as a security measure. As a security measure, execution of binary files on the mounted volume was prohibited via NFS server policy.

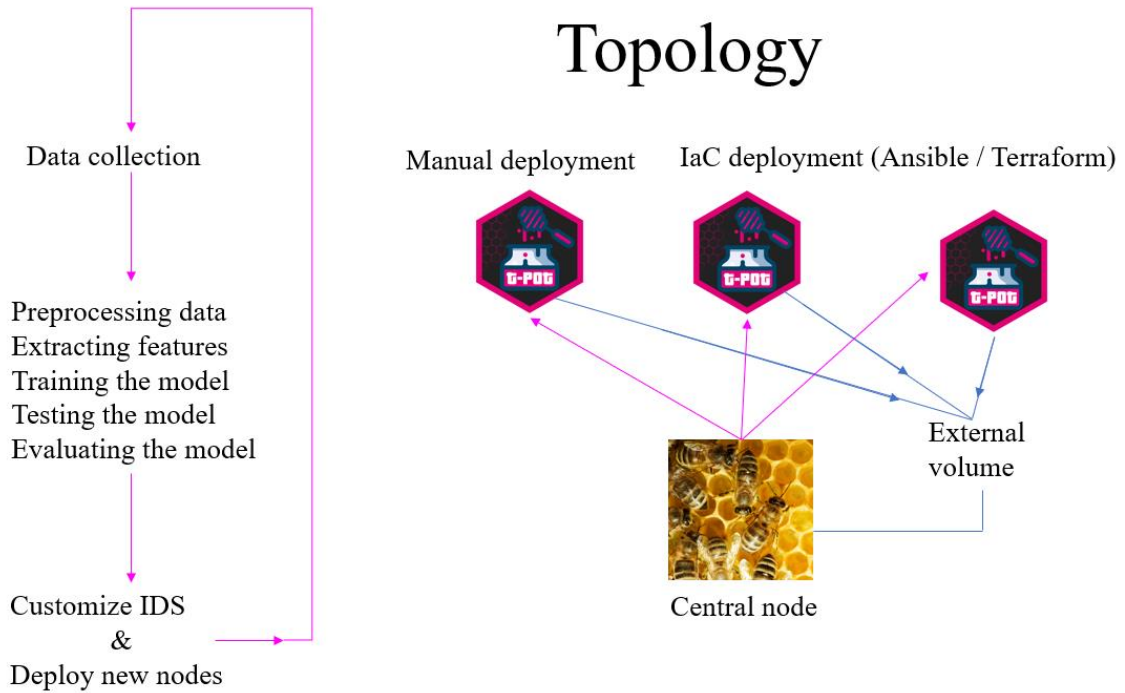


Figure 3.1: Abstract topology representation. All the deployed honeypot nodes can be created, configured and destroyed via a central node. The incoming dataflows are stored on the central node.

When configuring T-Pot, the creation of two separate network security groups was required. The first group is allowing the management services (SSH and Web UI via port 64295 and 64297) to be accessible only from whitelisted public IP addresses. The second group is allowing all the incoming TCP and UDP traffic, and should only be associated with the instance node once T-Pot is operational. By default, all inbound TCP and UDP ports must be open from port 1 to 64000 to 0.0.0.0/0. (SSH is redirected to TCP port 64295.) The outbound rules should not be filtered.

However, from an intruder's perspective the target can become less interesting if all TCP and UDP ports are open. Therefore, after some experimentation only a selected number of ports were configured to be open to 0.0.0.0/0. These ports are all necessary to be open for well-known services like HTTP, HTTPS, DNS, SMB, NTP and many others.

```
~$ openstack security group rule list tpot_thesis_main_public
```

	IP Protocol	Ethertype	IP Range	Port Range	Direction	Remote Security Group	Remote Address Group
	tcp	IPv4	0.0.0.0/0	23:23	ingress	None	None
	udp	IPv4	0.0.0.0/0	161:162	ingress	None	None
	tcp	IPv4	0.0.0.0/0	25:25	ingress	None	None
	udp	IPv4	0.0.0.0/0	123:123	ingress	None	None
	tcp	IPv4	0.0.0.0/0	8080:8080	ingress	None	None
	udp	IPv4	0.0.0.0/0	53:53	ingress	None	None
	udp	IPv4	0.0.0.0/0	1900:1900	ingress	None	None
	tcp	IPv4	0.0.0.0/0	8888:8888	ingress	None	None
	None	IPv6	::/0		egress	None	None
	tcp	IPv4	0.0.0.0/0	445:445	ingress	None	None
	tcp	IPv4	0.0.0.0/0	80:80	ingress	None	None
	udp	IPv4	0.0.0.0/0	5060:5061	ingress	None	None
	tcp	IPv4	0.0.0.0/0	22:22	ingress	None	None
	tcp	IPv4	0.0.0.0/0	21:21	ingress	None	None
	udp	IPv4	0.0.0.0/0	11211:11211	ingress	None	None
	None	IPv4	0.0.0.0/0		egress	None	None
	udp	IPv4	0.0.0.0/0	27015:27020	ingress	None	None
	tcp	IPv4	0.0.0.0/0	443:443	ingress	None	None
	tcp	IPv4	0.0.0.0/0	8443:8443	ingress	None	None

Figure 3.2: List of open TCP and UDP ports in the OpenStack environment. Opening the ports of only well-known services might attract the intruders more.

Generally speaking, T-Pot is a solution made for detecting TCP related attacks. But with proper configuration it can also monitor UDP traffic, thus providing more comprehensive analysis for different types of attacks, including DDoS attacks. The incoming connectionless traffic was allowed via open UDP ports to see how the interfaces and the bandwidth available becomes more overwhelmed. The Attack Map service in T-POT can provide visualized information about DDoS attacks (Fig. 3.3). This Figure shows DDoSPot detected DNS amplification and NTP amplification attacks several times during the research.

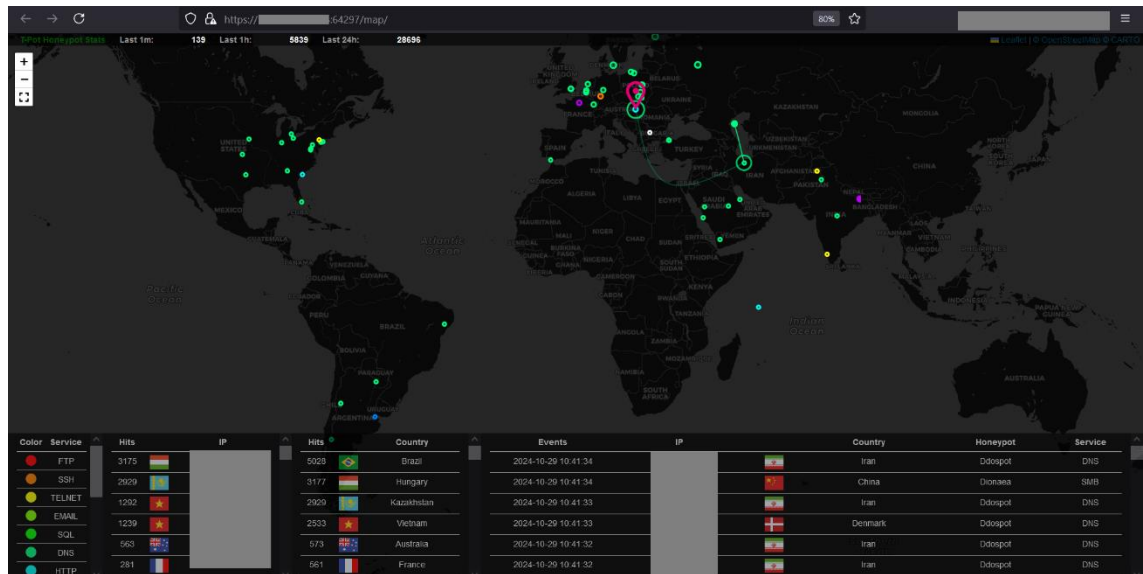


Figure 3.3: DDoS attack visualization via the Attack Map service.

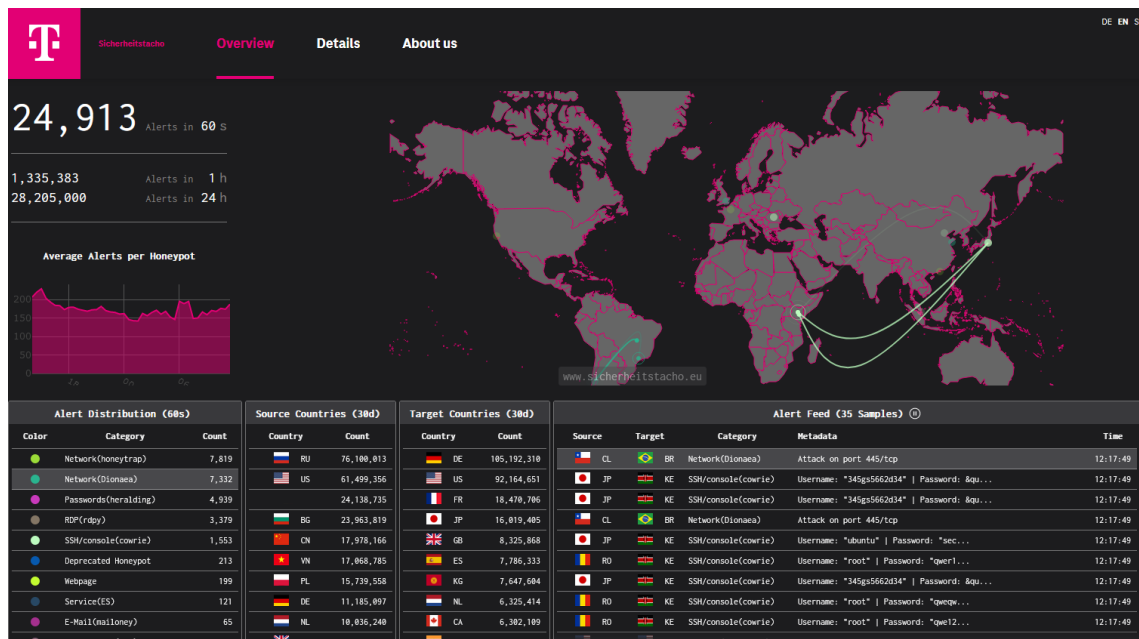


Figure 3.4: Attack map of the main Sicherheitstacho T-Pot.

By default, the data collected by each deployed T-Pot is submitted to Sicherheitstacho, a main T-Pot system hosted by Deutsche Telekom due to the configuration in the `~/tpotce/docker-compose.yml` file. Optionally, this configuration can be altered.

During the research, the following T-Pot services turned out to be the most important:

- Cowrie
- DDoSPot
- Suricata

Cowrie is a high-level interaction SSH and Telnet honeypot, designed to log the brute force attacks and the shell interaction performed by the intruder. T-Pot provides integrated Kibana Cowrie dashboards (tag clouds in this scenario) where the username and password attempts can be seen weighted based on the frequency of trial. Clicking on one particular value inside the tag cloud, we can immediately see the corresponding username or password as well. This is a great tool for building a database which can later be used for device hardening, configuration and deployment of integrated password complexity policies by any vendors whose product is related to information technology. After each login attempt, Cowrie also emulates a shell session which can be used to collect and analyze the most frequently used commands by the intruders.

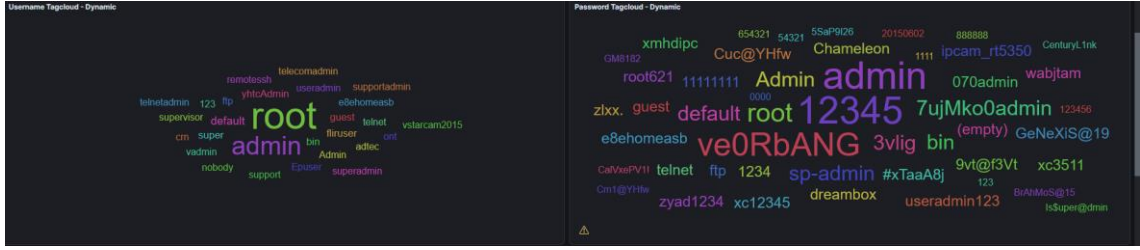


Figure 3.5: Username and password tag cloud, provided by Cowrie.

DDoSPot is a honeypot service which is able to track and monitor UDP-based DDoS attacks. DDoSPot currently supports following services in form of plugins:

- DNS server
- NTP server
- SSDP server
- CHARGEN server
- Random/mock UDP server

DDoSPot tries to emulate a DNS service, by forwarding all requests to a valid recursive resolver and tracking amplification parameters. It also responds to three different NTP modes: client (3), control (6) and private (7). Because they are the ones most utilized in amplification-based DDoS attacks. The amplification parameter can be configured separately at each service. The *packet_threshold*, defines how many identical requests are allowed from the same IP address. If the number of requests exceeds the threshold value, no more responses will be generated by the service. If this parameter is not configured to a low value, the DDoSPot service can potentially be an active DDoS participant. During the research, it was a common pattern that the intruder scanned the honeypot first for open ports and vulnerable services (Dioniea or Honeytrap). Then the first couple DNS requests came from the same (attackers') IP address. After that, the attempt was made to use the vulnerable and open DNS service towards victims at scale.

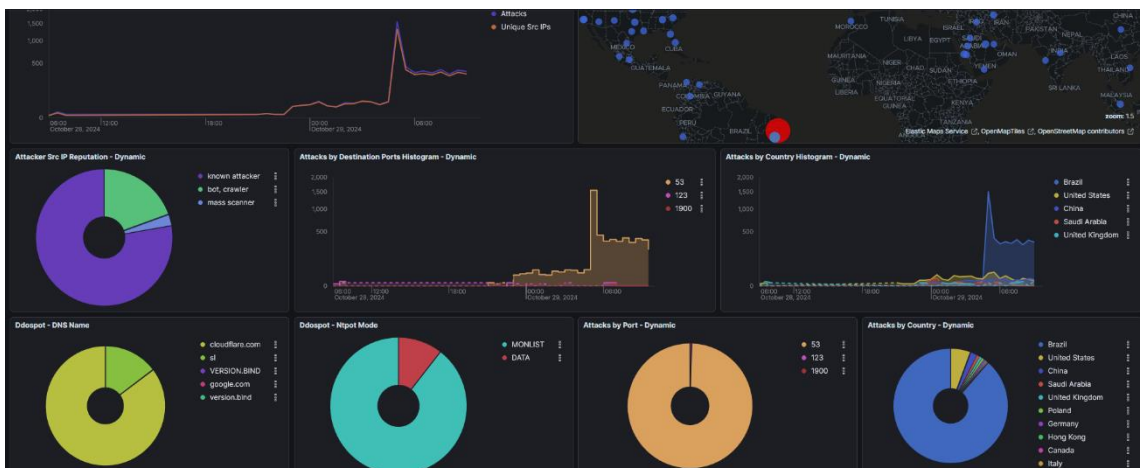


Figure 3.6: Visualization of DDoSPot statistics via Kibana service

3.3 Implementation of Suricata IDS

Suricata IDS [4] is an open-source network security monitoring engine, developed by the Open Information Security Foundation[5]. Among other qualities, its multi-threaded high-performance engine, the IPv6 and modern protocol support makes Suricata popular in the cybersecurity field. Snort rules are compatible with Suricata which gives room for experimentation with both open-source products simultaneously and can be a great source of cross-evaluation.

By default, Suricata is storing the logs as extensible event format (eve json) files but is also capable of logging the network traffic in .pcap format. Storing network traffic in .pcap is ideal (and much more suitable than the default .json log) for later data processing and evaluation at scale. By default, the log rotation is set to a daily measure but the configuration can be changed. Inside the T-Pot environment, like all the other T-Pot services, Suricata was hosted via a Docker container (with the help of databinding and adequate restart policy). This solution enhances portability, better version control and flexibility from a configuration perspective. Due to its capabilities for DDoS attack detection, the aim of this research was to use Suricata as a cross-evaluation tool (alongside DDoSPot). Inside Suricata, DDoS detection is handled through the following values:

- Threshold configurations
- Stream/flow configurations
- Custom detection rules

The appropriate configuration of thresholds can help prevent alert flooding. It can be set either globally in the *suricata.yml* file or it can be set per rule using the *threshold* keyword. In case of custom rules, four different types of rule actions can be used:

- alert: generates an alert,
- pass: allows the packet or stream to pass,
- drop: drop the packet or stream (inline mode),
- reject: sends RST flags as response

Figure 3.7: Custom Suricata DDoS detection rules

The SYN Flood detection rule logic:

- it watches for TCP packets with SYN flag set
- it counts SYN packets to each destination IP
- if any destination receives more than 5000 SYN packets in 1 second, alert triggers.

The UDP Flood detection rule logic:

- it monitors all UDP traffic
- it counts UDP packets to each destination IP
- if more than 10000 UDP packets hit one destination in 1 second, alert triggers

- the threshold value here is set higher than at SYN Flood because UDP is connectionless

The ICMP Flood detection rule logic:

- it watches for ICMP packets (ping traffic)
- it counts ICMP packets
- if more than 1000 ICMP packets hit one destination in 1 second, alert triggers
- the threshold value here is lower because ICMP traffic is typically light

Stream/flow configuration values can be customized for TCP stream reassembly, connection tracking, memory management and flow timeout handling. The most important parts of the Stream/flow configuration:

- memcap: it sets the memory limit for stream tracking
- checksum-validation: it validates packet integrity
- inline: it enables the IPS mode packet handling
- hash-size: it sets the size of the flow hash table (fine-tuning between memory usage vs lookup speed).

Protocol based rules are still the building block of current security measures, as automated scripts continuously scan all over the internet. Based on the port status, which can be open, filtered or closed, the attacker is able to recon which services are running inside a given infrastructure. TCP connect Scan (-sT) uses the concept of a full three-way handshake to discover port status. A port scanning tool (Nmap for example) sends a SYN flag. If the response is RST (reset), the given port is closed, if SYN/ACK then it is open. If the packet is dropped from the receiving end. Nmap does not get any response back, indicating that the port is filtered. TCP SYN Scans (-sS) scans are called stealth or half-open scans. They operate in a similar way but with open ports, -sS scan sends back an RST package instead of completing the TCP handshake. This functionality results in faster scanning time and it is harder to log by any IDS and firewall. UDP scans do not perform a handshake due to their connectionless qualities. Therefore, these types of scans are harder to implement and analyze for open ports, but for closed port discovery they can be an efficient tool, as the target port will send back an ICMP packet reporting the unreachable status of the given UDP port.

Port scanning tools at scale can be used (specially TCP -sS scan) to deliver Denial of Service and Distributed Denial of Service Attacks. As presented before, netflows can be managed with rule-based solutions. But after a certain point it can be a challenge to follow new netflow trends, write and update rules for all use cases and protocols, not to mention volume-based zero-day attacks. Therefore, as a complementary tool, flow-based detection is needed [6]. Regarding the changing characteristics and increasing volume of network flows, machine learning tools can help to analyze and classify them properly.

4 DATA ANALYSIS WITH MACHINE LEARNING TOOLS

4.1 Evolution of bot attacks

Considering the large scale of incoming network traffic, it was an assumption that the west majority of the incoming traffic is generated by automated scripts, in other terms, by bots. Good bots serve useful functions such as indexing websites for search engines or monitoring website performance. Bad bots run automated tasks with malicious intent, which represents one of the most serious threats to internet security today. They scrape data from sites without permission to reuse it and gain a competitive edge. According to Imperva's Bad Bot Report [7], nearly half of all internet traffic originated from automated scripts, with almost a third from bad bots in 2022. According to this report:

- 47.4% of all internet traffic came from bots in 2022, a 5.1% increase from 2021
- Traffic from bad bots increased for the fourth consecutive year, reaching 30.2% and recording a 2.5% increase from 2021.
- Good bot traffic remained significantly low at just 17.3% but the activity increased from 14.6% in 2021.
- The percentage of human traffic continues its downward trend, from 57.7% in 2021 to 52.6% in 2022 – a 5.1% decrease.

The weaponization of bots has become an emerging threat to websites and APIs. The Open Web Application Security Project[8] (OWASP) has mapped 21 categories of automated threats, with each having a different impact. Organizations face destructive downtime issues when bots are flooding their infrastructure to severely downgrade website response times or even take services offline.

Botnets can be deployed in a centralized or decentralized manner, depending on the C&C design. They use different communication protocols, such as IRC, P2P and HTTP. When communicating with the Command and Control server, the bot generates a unique communication mode. The botnet traffic can be identified by extracting traces of the communication related activity with the C&C server. This is the best phase for botnet detection because it lasts the longest, and instructions are sent automatically by the host.

In the past, bots were simple scripts that could be identified by firewalls based on a combination of IP address and port number rulesets. In other terms, early botnet detection mainly relied on blacklist or whitelist methods. Although the botnet detection accuracy of this method was high, the ability to detect unknown attacks was weak. The next generation of bots used IP rotation tools to bypass these defenses, so a bot came from a different IP address in each request. Bots then began rotating web browsers until web application firewalls started to fingerprint devices to detect and block these types of attacks also. Bots nowadays can mimic user behaviors, such as human-like cursor movements or keystrokes. They often choose a “low and slow” approach, which enables

them to deliver significant attacks using fewer requests and even delay requests, allowing them to not stand out from the normal netflow patterns and therefore, mimic human behavior. Threat actors have also learned that real users type their password characters much slower than bots do, so they adapted their approach. And they figured out that fixed intervals between bot actions are easy to program but also easy to detect, so they randomized the intervals. Some bots now are sophisticated enough to bypass CAPTCHA.

4.2 Classification of Denial of Service attacks

During a DDoS (Distributed Denial-of-Service) attack, the aim of the intrusion is to flood the infrastructure with traffic and to prevent legitimate users from accessing online services and sites. DDoS is a subcategory of the more general DoS (Denial of Service) attack. During DoS, the intrusion is happening via a single internet connection to overwhelm a target with fake requests or to exploit it with the help of vulnerabilities. DDoS represents the same problem but at a much larger scale. It utilizes thousands or millions of connected devices to achieve the goal. The amount of the devices used makes DDoS a lot harder to handle. Primarily, DDoS attacks are delivered via botnets. The intruder will hack into different computers and install malware, which makes a given computer serve as a node inside a botnet infrastructure.

Since different attack types are targeting different parts of the network, they can be classified according to the layer they target inside the OSI (Open Systems Interconnection) protocol stack. Based on this methodology, the different types of DDoS attacks[9] are the following:

- Volume-based or volumetric attacks
- Protocol attacks
- Application-layer attacks

Volume-based attacks aim to control all available bandwidth until it gets completely exhausted. DNS amplification is an example of such an attack type. In this scenario, the intruder continuously sends requests to an open DNS server using the spoofed IP address of the targeted website.

Protocol attacks are aiming for the consumption of all available capacity of web resources (for example servers, firewalls or load balancers). They use layer 3 and layer 4 vulnerabilities of the OSI protocol stack. A SYN flood can be considered an example of a protocol attack, in which scenario the attacker sends an overwhelming number of TCP requests to the target, with spoofed source IP addresses. The targeted servers attempt to respond to each connection request, but the final handshake never occurs, overwhelming the target servers during the process.

Application-layer attacks are referred to as a layer 7 DDoS attacks, referring to Layer 7 of the OSI protocol stack. They target the layer where web pages are generated in

response to HTTP requests. In this scenario, the intruder forces the victim's server to handle more requests than it normally can. An HTTP flood is a type of application-layer attack. It is similar to constantly refreshing a web browser on different computers all at once. The excessive number of HTTP requests overwhelms the server, resulting in a DDoS attack.

Besides the classification methodology shown before, DDoS attacks can be described more in-detail. Some of the most common types according to their behavior and used protocol are the following:

- HTTP Flood
- DNS Flood
- Ping Flood
- SYN Flood
- UDP Flood
- DNS Amplification
- Slowloris

4.3 Evaluation of datasets

In order to choose the appropriate dataset and ML algorithm for the training, a deep understanding of the data qualities is needed first. Generally speaking, there are two types of different datasets that must be made: packet-based datasets and netflow based datasets. Flow-based datasets put more emphasis on flow-level statistics and less on protocols. They capture aggregated information about network connections or "flows" - sequences of packets between the same source and destination over a period of time. The following datasets[10] were used for research purposes:

- KDDCUP1999
 - one of the first well-known packet-based datasets[11] used for research purposes in the context of cybersecurity
 - it was created by processing tcpdump data from a simulated military network environment
 - more focus on host-based and service-based statistics
 - the dataset might not simulate current traffic in the cyberspace
 - It contains about 5 million connection records
 - it has 41 features for each connection
 - It contains redundant records that can affect the variance and bias of the training of the ML algorithm.
- NSL-KDD
 - it was created in 2009 as an evolution of KDDCUP1999
 - contains packet-based statistics
 - it addresses some issues in KDDCUP1999 by removing redundant records

- it contains the same 41 feature characteristics and uses the same attack classification approach as the KDDCUP1999 dataset
- it contains about 150,000 records
- ISCX2012
 - it was created by the Information Security Centre of Excellence (ISCX) at the University of New Brunswick
 - it contains 7 days of network activity, including normal and attack traffic
 - it contains more modern attack scenarios compared to older datasets
 - it Includes full packet payload in pcap format.
- CICIDS2017
 - it was created by the Canadian Institute for Cybersecurity (CIC)
 - it contains 5 days of network activity
 - it includes a wide variety of attacks: Brute Force, Botnet, DoS, DDoS and Web Attacks
 - it provides labeled flows with 80 features
 - it is one of the most comprehensive and recent datasets available for research purposes

The content of packet-based datasets can be separated into three main categories by protocol:

- TCP
- UDP
- DNS

As the aim required to use netflow-based datasets, there was not much emphasis on the deep-level analysis of different protocols and their unique characteristics. All these datasets are labeled which is required for the training phase of a supervised ML algorithm. As a result of the analysis, the CICIDS2017 dataset was chosen as it contains the most information related to netflow statistics, which is a crucial part of botnet and DDoS detection. A complete infrastructure was used to create the dataset and it includes the traffic capturing of the following days:

- Monday: benign (contains about 523000 records)
- Tuesday: SSH and FTP brute force attempts (contains about 445000 records)
- Wednesday: different types of DoS attacks (contains about 692000 records)
- Thursday: XSS and SQL injection attacks (contains about 29000 records)
- Friday: PortScan and LOIT DDoS attacks (contains about 225000 records)

From the perspective of this research, the Friday dataset was the most relevant as it captured a LOIT DDoS attack and many different types of port scan attempts. LOIT (Low Orbit Ion Cannon) is a solution commonly used to launch DoS and DDoS attacks and it

works by flooding a target server with TCP, UDP, or HTTP packets with the goal of disrupting services.

4.4 Classification of machine learning models

To choose the appropriate machine learning algorithm for the task, they also must be classified and compared. Machine learning is a type of artificial intelligence that enables algorithms to uncover hidden patterns within datasets. This quality allows them to make predictions on new, similar data without explicit programming for each task.

When it comes to classification of ML models, there are three main types: supervised, unsupervised, and semi-supervised. Supervised ML is using labeled datasets to train algorithms to classify data or predict outcomes accurately. Input data is fed into the model, the model then adjusts its weights until it has been fitted optimally. Unsupervised ML is using clustering of unlabeled datasets. This type's ability to discover similarities and differences in information make it ideal for exploratory data analysis, image and pattern recognition. It is also used to reduce the number of features in a model through the process of dimensionality reduction. Semi-supervised learning offers a medium between supervised and unsupervised learning. During the training phase, it uses a smaller labeled data set to help classification and feature extraction from a larger, unlabeled data set.

Besides the main classification, the differentiation of ML algorithm types also can be made. The main algorithm types are the following:

- Decision trees
- Random forests
- Linear regression
- Logistic regression
- Neural networks
- Clustering

Many of these algorithms can be implemented either as a classification or as a regression model. In the classification problem statement, the target variables are discrete. Evaluation metrics like Precision, Recall, and F1-Score can be used. In the regression problem statement, the target variables are continuous. In this case evaluation metrics like Mean Squared Error, R2-Score, can be used. To choose the appropriate algorithm for the task, the following algorithms were studied and analyzed:

Decision Tree: DTs can be used for both predicting numerical values (regression) and classifying data into categories (classification). These models use a branching sequence of linked decisions that can be represented with a tree diagram. DTs use methods like the Gini impurity to build the tree hierarchy from the root node, based on the priority of the input characteristics. One of the advantages of DTs is that they are relatively simple

models. They are easy to build, interpret and audit, unlike the neural networks. However, from a variance (difference in fits between data sets) and bias (inability for a ML method to capture the true relationship between data) context, DTs are prone to overfitting (relatively low variance ability). DTs are not flexible when it comes to classifying new samples.

Random Forest Classifier: they combine the simplicity of DTs with flexibility, resulting in a vast improvement in accuracy. In RFs, the machine learning algorithm predicts a value or a category by combining the results from a number of decision trees. First, the algorithm creates a bootstrapped dataset, a subset of the original dataset. Each tree inside the forest randomly selects samples and features from the bootstrapped dataset (the tree can select the same sample more than once) and builds a hierarchical tree from the root node. The hierarchy of the selected features is also random. Then, the out-of-bag subset is given to the trees and based on majority voting of the trees, we can evaluate our model. To a certain extent, the more trees we use, the out-of-bag (OOB) error rate should converge towards zero. However, after a certain point, the OOB rate will not decrease, despite adding more trees to the model. Due to its experimental and trial-error behavior, RFC excels both in bias and variance metrics against the DT. However, it might not be the best choice for DDoS detection and evaluation. Random Forest models can capture relationships between different feature characteristics inside a single netflow, but they have limitations when it comes to understanding complex, time-based relationships between a series of netflows. For such a complex problem some type of sliding window approach is required. Nevertheless, for identifying packed-based or protocol-based attacks, RFC can be a relatively simple and resource friendly approach.

Neural networks: they are computational models composed of interconnected layers of nodes. These models are capable of recognizing patterns and they play an important role in applications including language translation, image and speech recognition. The basic components include input vectors, weights, and biases, which are processed through activation functions to produce outputs. The activation function transforms the weighted sum of inputs into meaningful signals that propagate through the network. Deep neural networks extend this architecture by adding hidden layers between input and output. The main advantage is the ability to learn and discover complex relations. However, while this increased complexity allows them to solve more complex problems and recognize new patterns, it comes with multiple tradeoffs: lower interpretability compared to simpler models like Decision Trees or Random Forests, and higher computational resource requirements.

In order to work with time series, Recurrent Neural Networks[12] can be used. These models are able to capture and update their inner state regarding the new inputs, with the help of integrated feedback loops. Contrary to DNNs, Recurrent Neural Networks do not have hidden layers. Instead, they use a feedback loop which makes RNNs ideal for sequential data analysis. At first glance RNN seemed like an optimal choice for the aim

of this research. However, since the weights and biases are shared across the unrolled pipelines, vanilla RNNs are prone to vanishing /exploding gradient problems (Vanishing Gradient if the weight value is less than one, exploding gradient if the model is greater than one). Either the model only finds a local optimal value or oversteps and global optimal gradient (overshooting). In case of DNNs, each hidden layer has its own weights and biases so weight values are able to cancel each other out which makes them less prone to overfitting. For these reasons, RNNs only work on fairly short sequences.

To solve the problem mentioned above, Long-short term memory algorithms [13] were used. Contrary to RNNs, these models are better at filtering out contextually irrelevant data. LSTMs are designed to capture and learn from time-based dependencies in sequential data flows (netflows in this scenario). This is crucial for understanding the progression of attacks over time. The long-term memory component of LSTMs allows the model to remember important information over long sequences. This is important for detecting patterns that develop over time, such as the gradual buildup of a DDoS attack. LSTM uses Sigmoid and Tanh activation functions. The Sigmoid function assigns input values to a range between 0 and 1. The Hyperbolic Tangent function assigns input values to a range between -1 and 1.

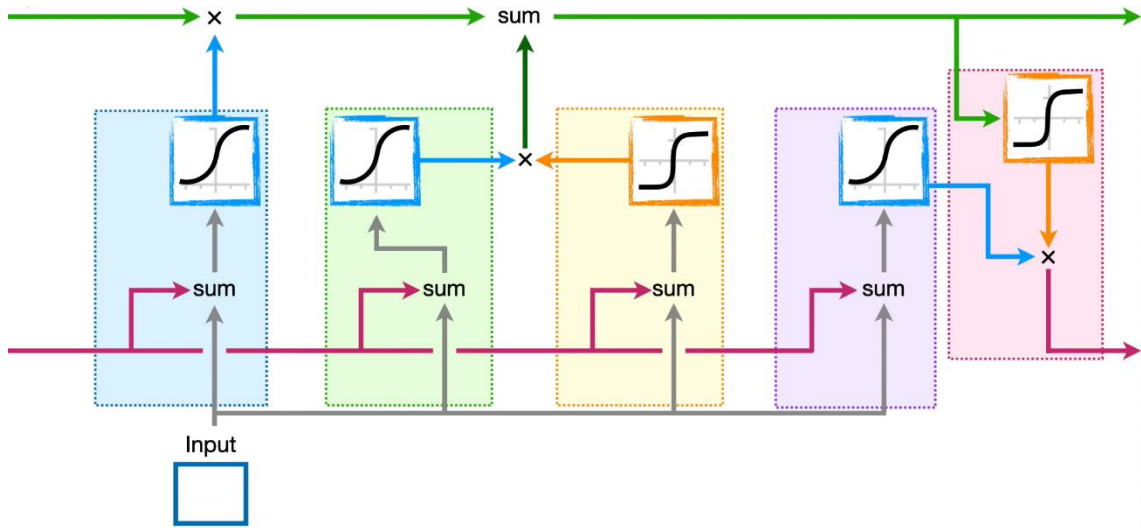


Figure 4.1: Abstract representation of the LSTM model.

In the illustration (4.1), the blue rectangles represent the Sigmoid function, the orange ones the Tanh function. The biases are illustrated as sum signs and the weights as arrows. The cell state (illustrated as a green line that runs all the way across the top) represents long-term memory. No weights and biases can modify the cell state directly. This lack of weights allows LSTM to flow through a series of unrolled units without causing the gradient to explode or vanish. The hidden state (illustrated as a pink line) represents short-term memories. LSTM can be broken down into three different stages or gates. The first stage (highlighted as blue) determines what percentage of the long-term memory is remembered, with the help of the Sigmoid function. Therefore, this stage is usually called

the forget gate. The second stage has two parts. The second part (highlighted as orange) will create a potential long-term memory. The first part (highlighted as green) decides the percentage of the input to be added to the cell state (identical to the forget gate of the previous state). This stage is often referred to as the input gate. The third stage (highlighted as purple) updates the short-term memory. This stage is often called the output gate. Despite all the benefits, LSTMs have some drawbacks as well. Since it is a much more complex model, training LSTMs requires more computational power and resources overall. Cross-validation might still be required to make sure the model is not overfitting during training.

4.5 Implementation of a supervised machine learning model

For the implementation, a supervised ML model was used. Since labeled datasets were available for the training phase, it was easier to evaluate how well the model performs. There are two types of labels ('DDoS' and 'Benign') in the original Friday subset. Initially, the aim was to train a model which can differentiate three different netflow types: normal traffic (software updates, NFS export, SSH logins from allowed IP addresses), port scans (nmap, rust etc.) and DDoS attacks. The main goal was to differentiate seemingly separate attacks and port scans as part of a DDoS attack. TCP -sS scans can be utilized for launch of DDoS attacks as it leaves half-open connections by not completing the TCP handshake. It forces the target to allocate resources for each pending connection and fills up the target's TCP connection queue. Unfortunately, this idea had to be temporarily discarded for several reasons. First of all, a dataset which included both port scan and DDoS attacks was not found. Since the CICIDS2017 contains netflows labeled as 'PortScan' (in a separate dataset file), it was an option to compare the record values. However, in case of sequential data the order of events is crucial and merging two separate datasets would cause the representation of unrealistic netflows sequences. Therefore, a more simple, binary classification model (DDoS=1, Other=0) was implemented which is considered a sequence-to-vector architecture.

The pipeline of the process was the following:

- Data collection
 - o data collection from Suricata IDS, Honeytrap, Cowrie, Dionea and other T-Pot services, .pcap file generation
- Data preprocessing
 - o utilization of Argus, a network analyzing tool to process packet data and generate the network flow dataset
 - o utilization of Argus for feature selection
- Training
 - o selection of the seemingly appropriate subset of feature characteristics for training
 - o training of the LSTM model on the CICIDS2017 Friday subset dataset

- utilization of early stopping tools (stop the training when validation performance stops improving)
- Testing and evaluation
 - utilization of cross-evaluation techniques
 - evaluation of the LSTM model against CICIDS2017 testing subset folds

4.5.1 Feature selection and data preprocessing

When choosing the appropriate feature characteristics for training, the following questions were considered the most important:

- Which parameter change can indicate potential intrusion attempt, in this case, a DDoS attack?
- Which feature characteristics will be available after data collection and preprocessing?

As a start, the record values were compared to see if any pattern is detectable. Sudden bursts in the Flow/Bytes value (compared to recent history) and drop in Flow Duration can indicate scanning activities. To the contrary, sudden bursts in Flow Duration and drop in Flow/Bytes values can indicate network congestion which is often a result of DDoS attacks. To train the model, the following feature characteristics were used from the Friday subset of the CICIDS2017 dataset (9 in total) as these characteristics can be extracted from the incoming network logs with the help of Argus:

- Destination Port
- Flow Duration: duration of the flow in microseconds,
- Total Forward Packets: total number of packets in the forward direction,
- Total Backward Packets: total number of packets in the backward direction,
- Total Length of Forward Packets: total size of packet in forward direction,
- Total Length of Backward Packets: total size of packet in backward direction,
- Forward Packet Length Mean: mean value of packet in forward direction
- Backward Packet Length Mean: mean value of packet in backward direction
- Flow Bytes / second: number of flow bytes per second.

In the future, a more scientific approach can be used for feature selection, for example, the implementation of encoders. They transform input data into a compressed representation and learn how to extract and represent the most important features of the input data in a more compact form.

CICIDS characteristics	Argus characteristics	DDoS	PortScan	Benign
Destination Port	Dport	80	80	58607
Flow Duration	Dur	1293792	5021059	15
Total Fwd Packets	SrcPkts	3	6	1
Total Backward Packets	DstPkts	7	5	1
Total Length of Fwd Packets	SrcBytes	26	703	6
Total Length of Bwd Packets	DstBytes	11607	1414	6
Fwd Packet Length Max	sMaxPktSz	20	356	6
Fwd Packet Length Min	sMinPktSz	0	0	6
Fwd Packet Length Mean	sMeanPktSz	8.666666667	117.1666667,	6
Fwd Packet Length Std	N/A	10.26320288	181.5361305,	0
Bwd Packet Length Max	dMaxPktSz	5840	1050	6
Bwd Packet Length Min	dMinPktSz	0	0	6
Bwd Packet Length Mean	dMeanPktSz	1658.142857	282.8,	6
Bwd Packet Length Std	N/A	2137.29708	456.923626,	0
Flow Bytes/s	Rate	8991.398927	421.6242032,	800000

Figure 4.2: Content of the CICIDS2017 feature characteristics. The chosen “DDoS” and “PortScan” records are the first ones from an intrusion netflow. Before that, the incoming netflow was labeled as “Benign”.

4.5.2 Training and evaluation

For the implementation, a popular open-source development environment, JupyterLab was chosen. It provides interactive notebooks, Python consoles (with ipykernel support) for quick code execution. It also provides a variety of libraries for data visualization. The development environment was hosted via a docker container with data binding to the host, providing persistent storage. The notebooks were saved to the host machine’s filesystem. For the implementation, the following Python libraries were used:

- Pandas: data loading and preprocessing
- Numpy: sequence creation and other numerical operations
- Sklearn: feature normalization, cross-validation splitting
- Tensorflow.keras: deep learning framework, model architecture
- Matplotlib & Seaborn: visualization libraries

In the implementation, the following parameters were used and customized:

- data preprocessing parameters,
- sequence creation parameters,
- model architecture parameters,
- training parameters,
- cross-validation parameters
- early stopping parameters.

During preprocessing, the features (9 in total) were selected from the original dataset. The missing or invalid values were handled by the *numpy.nan* function to ensure the data is clean for training. For normalization of the features, the MinMaxScaler method was used. It transformed each feature value to a fixed value between range 0-1.

Next, the sequence creation parameters were set to define how the temporal data is structured for the LSTM model. The *time_steps* value is used to define the length of sequences, in other words, the number of previous consecutive netflow records to make a prediction. This value defines the size of the sliding window inside the LSTM model. The *stride* value controls how much the sequences overlap inside the window. If the *stride* value equals to the *time_steps* value then there is no overlap at all. If the *stride* value is set to 1, then the overlap is almost complete. After some experimentation, the *time_steps* value was set to 20 and the *stride* value was set to 5. The model architecture became quite complex as three hidden LSTM layers were used in one sequence. The number of LSTM units (neurons) were set to 64, 32 and 16. The incoming netflow, containing 9 features was broken down into 64, 32 and 16 vector values in the corresponding hidden layers. This architecture made sure that each LSTM layer can learn temporal patterns which increases model capacity to learn complex relationships between incoming network flows. The *dropout* rate was set to 0.3 across all hidden LSTM layers, which randomly deactivated 30% of neurons during training. This mechanism helped prevent overfitting. The final dense layers were used for classification. They use Rectified Linear Unit and SoftMax activation functions. The role of dense layers is to reduce dimensionality, convert the features to classification probabilities and to map these probabilities to output vectors (Normal or DDoS in this case). Finally, the Categorical cross-entropy function is used as a loss function to measure the difference between predicted probabilities (SoftMax output) and the one-hot encoded true labels.

Continuing the process, the training parameters had to be configured. The most important training parameters were the following:

- *batch_size*: the number of sequences processed before model update
- *epochs*: the maximum number of training iterations
- *learning_rate*: it controls how much the model adjusts its weights

After some experimentation, the *batch_size* value was set to 64, the number of *epochs* to 20 and the *learning_rate* to 0.001. For defining the learning rate, the module called Adam was used from the *tensorflow.keras* library.

To evaluate the efficiency of training, a k-fold cross-evaluation method was implemented. The most important parameters for cross-evaluation were the following:

- *n_splits*: number of folds for validation
- *time_steps*: size of the sliding windows during evaluation

The steps of the algorithm are the following

- split the data into k equal parts (5 in this case)
- for each training iteration:
 - use one part as the training set, use the other part as the evaluation set
 - train the model and evaluate performance

During the cross-evaluation process, the displayed class weight values indicated that the dataset is slightly imbalanced, with fewer class 0 (Normal) samples than class 1 (DDoS) samples across all folds. After cross-evaluation, the average emulated training accuracy and loss validation results could be displayed.

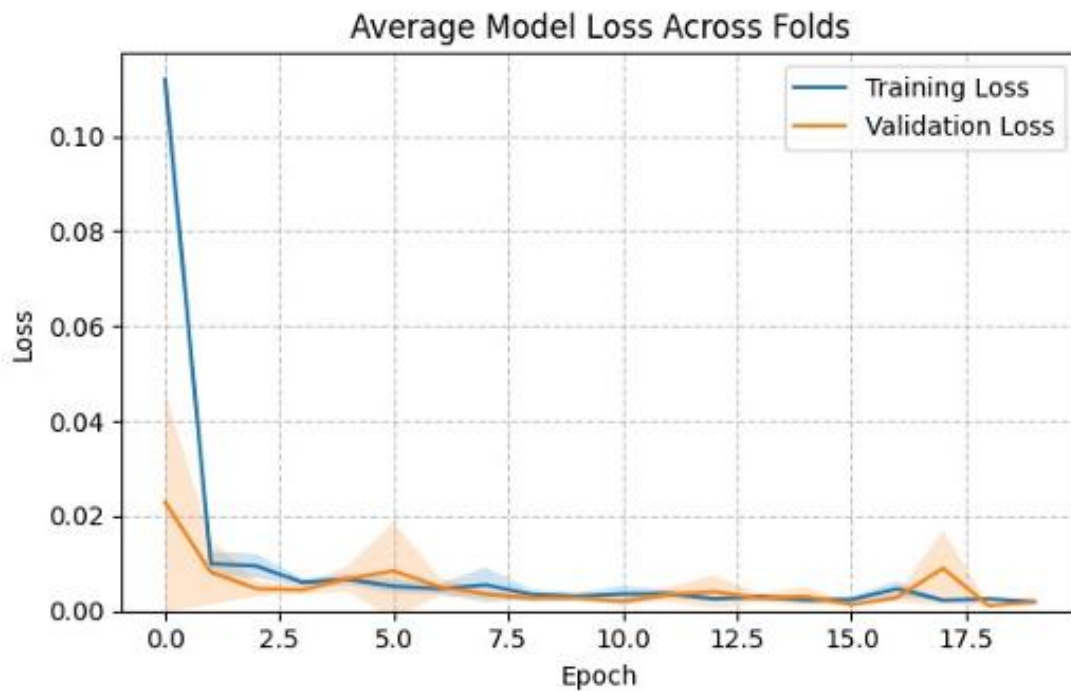


Figure 4.3: Average validation loss values across all cross-evaluation folds.

As training continues across all epochs, the value of the loss function changes in proportion to the actual step size. The step size can be determined as the product of learning rate and the gradient value. Ideally, the loss function converges towards the global optimum value, which is zero. Since early stopping was implemented, the training stops once the near-optimum global value is found and no further improvement can be achieved. In the code, *the patience* value was set to 5. The training process stopped if the validation loss did not improve across 5 consecutive epochs.

Early stopping statistics:
 Stopping epochs: min=8, max=20
 Mean stopping epoch: 15.6 ± 4.5
 Individual stopping epochs: [16, 14, 20, 8, 20]

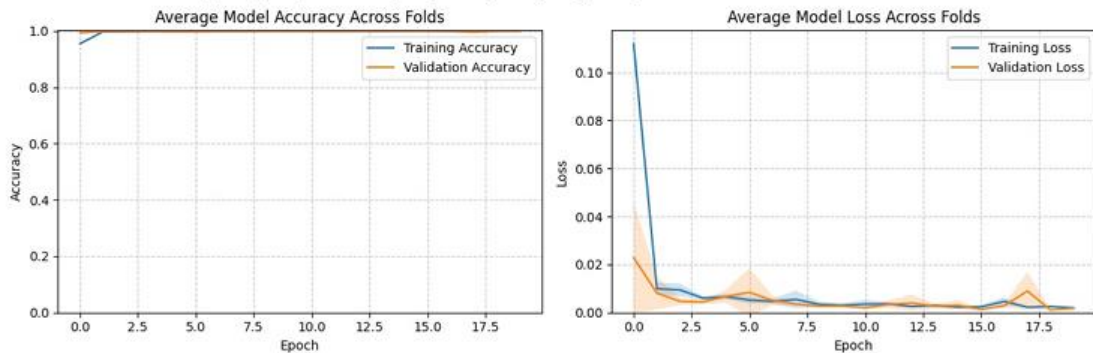


Figure 4.4: Average prediction accuracy of the model across all cross-evaluation folds.

The displayed results present the average relation between training and validation metrics across all five folds. In Figure 4.6, training accuracy shows how good the prediction of the model is on the training dataset, and training loss shows how confident the model is on the prediction. In each epoch, the model:

- makes predictions on training data,
- calculates the error (loss),
- updates its weights to improve.

Validation accuracy shows how good the prediction is on unseen, testing data and validation loss shows how confident the model is on the prediction. In each epoch, the model:

- makes predictions on validation data
- calculates metrics
- does not update weights based on these results.

For further evaluation, a Confusion Matrix can be used. In case of a binary classification problem, it is a 2x2 table, containing the following values:

- TP: Correctly predicted positive
- TN: Correctly predicted negative
- FP: Incorrectly predicted positive
- FN: Incorrectly predicted negative

The displayed CF represents only the validation results which can be calculated the following way:

- the total number of netflows in the original dataset was approximately 225000 records, out of which 45000 sequences could be created (due to the *time_steps* value being equal to 20 and the *stride value* being equal to 5),
- 20% of the sequences were used as a validation subset, resulting in about 9000 sequences as the sum of true positive and true negative values.

Confusion Matrix Analysis:
Average predictions per fold:
True Normal: 3890
True DDoS: 5103
Predicted Normal: 3888
Predicted DDoS: 5105

Percentages:
True Normal correctly classified: 99.95%
True DDoS correctly classified: 100.00%
False DDoS rate: 0.05%
False Normal rate: 0.00%

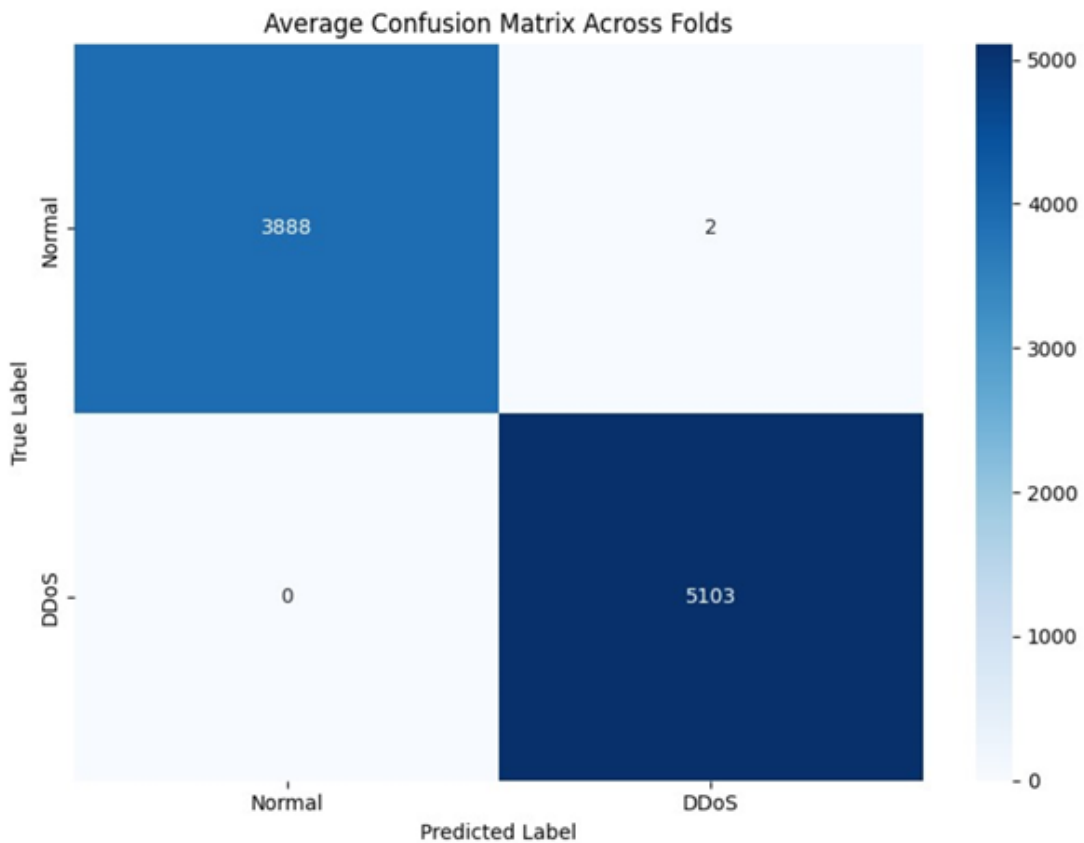


Figure 4.5: Average Confusion Matrix values across all cross-evaluation folds.

From the TP, TN, FN and FP values, the following metrics can be calculated:

- Accuracy = 99.98% (the proportion of total correct predictions)

- Precision = 99.96% (the proportion of correct positive predictions among all positive predictions)
- Recall = 100% (the proportion of actual positive cases that were correctly identified)
- F1 Score = 99.98% (the harmonic mean of precision and recall, providing a single score that balances both metrics)

```

Cross-validation results:
Fold 1: 0.9999
Fold 2: 0.9998
Fold 3: 1.0000
Fold 4: 0.9999
Fold 5: 0.9999
Average accuracy: 0.9999 (±0.0000)

Training final model on full dataset...
Epoch 1/20
598/598 ————— 47s 15ms/step - accuracy: 0.9615 - loss: 0.0859 - val_accuracy: 0.9952 - val_loss: 0.0310
Epoch 2/20
598/598 ————— 37s 12ms/step - accuracy: 0.9985 - loss: 0.0067 - val_accuracy: 0.9977 - val_loss: 0.0151
Epoch 3/20
598/598 ————— 53s 16ms/step - accuracy: 0.9995 - loss: 0.0023 - val_accuracy: 0.9993 - val_loss: 0.0033
Epoch 4/20
598/598 ————— 36s 12ms/step - accuracy: 0.9995 - loss: 0.0023 - val_accuracy: 0.9988 - val_loss: 0.0094
Epoch 5/20
598/598 ————— 45s 15ms/step - accuracy: 0.9984 - loss: 0.0061 - val_accuracy: 0.9988 - val_loss: 0.0040
Epoch 6/20
598/598 ————— 83s 15ms/step - accuracy: 0.9997 - loss: 0.0014 - val_accuracy: 0.9996 - val_loss: 0.0015
Epoch 7/20
598/598 ————— 46s 15ms/step - accuracy: 0.9997 - loss: 0.0012 - val_accuracy: 0.9982 - val_loss: 0.0160
Epoch 8/20
598/598 ————— 46s 15ms/step - accuracy: 0.9997 - loss: 0.0011 - val_accuracy: 0.9989 - val_loss: 0.0089
Epoch 9/20
598/598 ————— 81s 15ms/step - accuracy: 0.9997 - loss: 0.0011 - val_accuracy: 0.9984 - val_loss: 0.0093
Epoch 10/20
598/598 ————— 47s 16ms/step - accuracy: 0.9999 - loss: 5.9135e-04 - val_accuracy: 0.9989 - val_loss: 0.0056
Epoch 11/20
598/98  ————— 27s 14ms/step - accuracy: 0.9998 - loss: 9.4444e-04

```

Figure 4.6: Cross-evaluation results and training of LSTM.

During the final training phase, 85% of the sequences were used as a training subset, resulting in 598 batches in total (as batch size was set to 64). There is always room for experimentation with different model architectures and training values but given the results from the cross-evaluation, the used parameters seemed optimal. Additionally, other fine-tuning strategies can be used, for example using multiple datasets that contain similar feature characteristics but their values are slightly different. This would ensure that the weights and biases are adjusted accordingly inside the model architecture.

4.5.3 Deployment of the trained model

The current ML model only works as a complementary solution to the honeypot system. T-Pot was not an optimal choice as a data collection tool since all the services are configured to deflect potential exploitations. This behavior results in differing datasets from the one used during training (CICIDS2017 - Friday). When capturing the original dataset, the researchers emulated a network traffic of a working environment. To the contrary, T-Pot does not allow the incoming netflow to overwhelm the interfaces, rather its aim is to detect protocol-based anomalies. If poorly configured, many tools can be used by the intruders to deliver real-life attacks against victims. (DDoSPot is one example

as it can be part of a volume-based DNS or NTP amplification attack towards victims). Therefore, LSTM only detected the incoming traffic as “benign” when T-Pot services were enabled and running. Flow duration, forward packet count, backward packet count and flow bytes /sec values were all similar to a conventional netflow. When all T-Pot services were stopped, the model was able to detect incoming DDoS attacks. However, it is difficult to evaluate the model performance if no signature-based tools are running simultaneously with the capturing process. To evaluate the model performance, one dataset (similar to CICIDS2017) of the CIC-DDoS2019[14] datasets was used.

```
Detailed Summary:
2024-11-04 19:42:39,281 - INFO - total_samples: 692703
2024-11-04 19:42:39,281 - INFO - ddos_alerts: 188800
2024-11-04 19:42:39,282 - INFO - normal_traffic: 503903

Sample of predictions:
  predicted_class  ddos_probability
0                0.0              0.268959
1                0.0              0.268959
2                0.0              0.268959
3                0.0              0.268959
4                0.0              0.268959
```

Figure 4.7:DDoS detection report on incoming netflows.

In the deployment code, the *time_steps* value was set to 20, but the *stride* value was also set to 20 as in the detection phase the aim is making a 1:1 relation between netflows and labels and overlap should be avoided. Based on the results, the deployed LSTM model shows the ability of detecting DDoS attacks but there is room for improvement:

- Accuracy value indicates that the model correctly classifies about 89% of all cases,
- the high precision means that when the model detects a DDoS attack, it is right about 97% of the time,
- the recall value indicates that the model captures about 72% of all actual DDoS attacks,
- the F1 Score value was 82.95%

Confusion Matrix Metrics:
True Positives (TP): 183095
True Negatives (TN): 434326
False Positives (FP): 5705
False Negatives (FN): 69577

Performance Metrics:
Accuracy: 0.8913
Precision: 0.9698
Recall: 0.7246
F1 Score: 0.8295

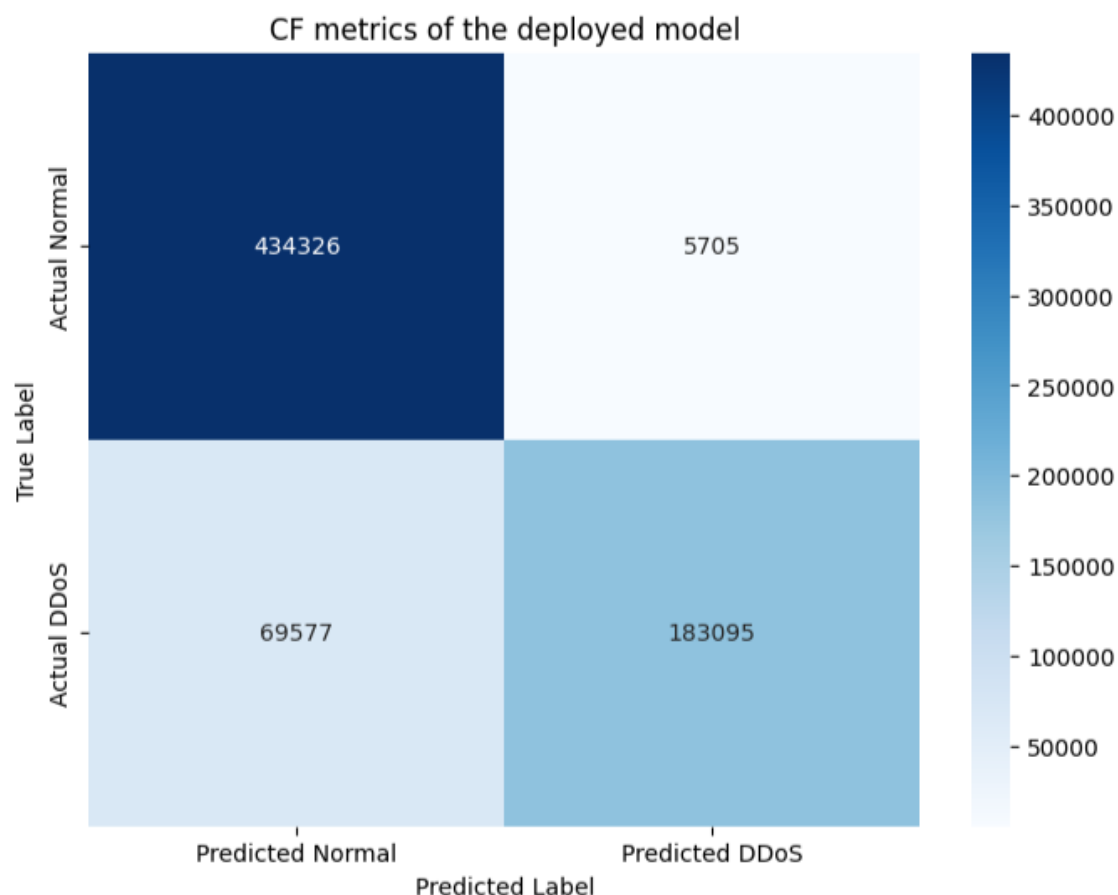


Figure 4.8: CF metrics of the deployed model.

As a future research aim, the recall value should be improved to detect more actual DDoS attacks. There's often a trade-off between precision and recall as increasing precision often decreases recall and increasing recall often decreases precision values. The presented result confirms that the current model should indeed only be used as an extension of already existing conventional cybersecurity tools until further development. Also, as a better data gathering strategy, a simple demo webserver will be used to allow incoming traffic. With the help of integrated monitoring (running node exporter service on the honeypot node and running Prometheus server on a monitoring node), any network congestion related activity could be detected.

5 THE INFRASTRUCTURE AS CODE (IaC) PARADIGM

5.1 Introduction to the IaC paradigm

Infrastructure as code (IaC) allows us to define our infrastructure with a declarative approach, ensuring that the state of the environment is consistent across different stages and platforms.

Infrastructure as code (IaC) is becoming a crucial concept when it comes to infrastructure deployment and maintenance. It enhances consistency, reproducibility, version control, automation, scalability, cost efficiency, faster deployments and improved security. With the help of IaC, the quick and dynamic allocation of hardware and software resources is possible. As one particular approach, the steps of such an operation consist the following:

- authentication towards any cloud provider that supports IaC
- configuration and deployment of hardware resources and network security groups via Terraform code
- any operating system or other software-based configuration, deployment and deletion can be made via Ansible.

Due to the idempotent qualities of Ansible (which registers and modifies the state of any configuration, based on the declared parameters) accidental repetition of operations and misconfiguration can be avoided. At the same time, security patches, updates, firewall configurations can be made at large scale with a declarative, automated approach. This quality of Ansible makes it possible to modify the configuration of any honeypot node in case of an unexpected and poorly handled security incident. This way, further escalation of such incidents can be mitigated. With Terraform, any node can be destroyed in a matter of seconds. Since logs are exported periodically to a central node, ideally the given incident would not result in significant data loss.

5.2 Implementation of IaC

Before using Terraform to deploy any resources, authentication towards a cloud provider is needed. In general, the authentication is resolved with the help of a source file. After a successful authentication, the creation and deletion of cloud resources is possible. To use Ansible, port 22 must be open on the client nodes. Ansible configurations are written in `.yaml` files and the configurations can be deployed to nodes that are part of the inventory file.

6 RESEARCH GAPS

6.1 Short-term aims

For the future of this research, setting up static routing between the interfaces serving the NFS share is a must from a maintenance and a security perspective. Although IPv6 related traffic was collected, the scope of this research did not include the analysis of any IPv6 specific attacks. The currently trained LSTM model cannot be used as a functional extension of a real-time IDS service since the incoming logs must be preprocessed first in order to be evaluated. Solving such a crucial challenge can help make the model extend existing signature and behavior-based firewalls, IDS and IPS systems.

There are many different packet-based and flow-based datasets (for example, the CICIDS2023 flow-based dataset provided by the University of New Brunswick) which can be analyzed and by which different machine learning models and specific configurations can be customized in accordance with the quality of the chosen, new dataset. The design of the LSTM model is complex, there is always room for experimentation with different activation functions, evaluation methodologies and training optimization using techniques like hyperparameter tuning. Using encoding methods would be a useful extension of the already existing project, as it would show the most relevant feature characteristics of the incoming netflows.

6.2 Long-term aims

Differentiating bot attacks (for example, UDP flood vs TCP SYN attack) during the training and evaluation phases would be beneficial to get a deeper insight into the structure of a particular attack. To implement that, a future LSTM model would be trained on a packet-based dataset. Such an implementation raises challenges training wise, for selecting the appropriate feature characteristics, epoch, slide sizes and many other parameters. It might also require the implementation of a more complex, Bidirectional LSTM model which has an architecture similar to Gated Recurrent Networks and Transformers [15].

Honeypot nodes could be deployed at scale by which the diversity of the incoming data, consisting of netflows and different protocols can vary greatly. The diverse incoming data can be used to train better models with the help of distributed ML solutions. This method would enhance the efficient usage and better allocation of resources, the training time and effectiveness. Extending the already existing IaC solution, and applying it to the configuration of the distributed ML development environment, or to the deployment of the trained model itself would be an important part of the implementation. The infrastructure could further be extended with integrated monitoring to make sure the resources are allocated adequately when it comes to the distributed ML nodes, and also the state of the honeypot nodes is healthy and operational.

7 SUMMARY

The detection and mitigation of bot attacks will become a more and more difficult challenge to solve. As the complexity of bot attacks continues to grow, the netflow content of attacks will vary less and less from a benign netflow, or even from a typical service issue (for example: slow-loading of web pages). Some companies may welcome uncommon spikes in their bandwidth usage due to a release of a new product. Situations like this make it very difficult to find relations between seemingly unrelated netflows and act accordingly to avoid service downtime.

The aim of this research was to represent and study an alternative solution. With the help of adequately trained and evaluated ML algorithms, the detection of bot attacks is possible. Due to the evolution of these attacks and change in netflow characteristics, both the training and evaluation methodologies must evolve continuously, following the new trends in cyberspace. The currently trained LSTM model is able to detect DDoS attacks in real-life scenarios with high precision and relatively low recall values, leaving room for improvement.

7.1 List of tools used during the research

T-Pot	https://github.com/telekom-security/tpotce
Suricata IDS	https://suricata.io/
List of datasets	https://www.unb.ca/cic/datasets/index.html
Argus	https://openargus.org/using-argus
Jupyterlab	https://jupyter.org/
Numpy library	https://numpy.org/
Tensorflow library	https://www.tensorflow.org/
Pandas library	https://pandas.pydata.org/
Matplotlib library	https://matplotlib.org/
Hashicorp Terraform	https://www.terraform.io/
Ansible	https://www.ansible.com/

8 BIBLIOGRAPHY

- [1] Dubey, Anubha. ‘A Review on Honeypot Deployment’. *London Journal Press*, 1 January 2020. https://www.academia.edu/43747094/A_Review_on_Honeypot_Deployment.
- [2] Sokol, Pavol, Jakub Míšek, and Martin Husák. ‘Honeypots and Honeynets: Issues of Privacy’. *EURASIP Journal on Information Security* 2017, no. 1 (28 February 2017): 4. <https://doi.org/10.1186/s13635-017-0057-4>.
- [3] Kotenko, Igor, Elena Fedorchenko, Evgenia Novikova, and Ashish Jha. ‘Cyber Attacker Profiling for Risk Analysis Based on Machine Learning’. *Sensors* 23, no. 4 (January 2023): 2028. <https://doi.org/10.3390/s23042028>
- [4] Waleed, Abdul, Abdul Fareed Jamali, and Ammar Masood. ‘Which Open-Source IDS? Snort, Suricata or Zeek’. *Computer Networks* 213 (4 August 2022): 109116. <https://doi.org/10.1016/j.comnet.2022.109116>.
- [5] OISF. ‘OISF’. Accessed 6 November 2024. <https://oisf.net/>.
- [6] Mittal, Meenakshi, Krishan Kumar, and Sunny Behal. ‘Deep Learning Approaches for Detecting DDoS Attacks: A Systematic Review’. *Soft Computing* 27, no. 18 (1 September 2023): 13039–75. <https://doi.org/10.1007/s00500-021-06608-1>.
- [7] OWASP Top Ten | OWASP Foundation’. Accessed 6 November 2024. <https://owasp.org/www-project-top-ten/>.
- [8] Resource Library. ‘2023 Imperva Bad Bot Report’. Accessed 26 October 2024. <https://www.imperva.com/resources/resource-library/reports/2023-imperva-bad-bot-report/>.
- [9] Fortinet. ‘What Is a DDoS Attack? DDoS Meaning, Definition & Types’. Accessed 27 October 2024. <https://www.fortinet.com/resources/cyberglossary/ddos-attack>.
- [10] ‘Datasets | Research | Canadian Institute for Cybersecurity | UNB’. Accessed 5 November 2024. <https://www.unb.ca/cic/datasets/index.html>.
- [11] ‘KDD Cup 1999 Data’. n.d. Accessed 6 November 2024. <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.
- [12] ‘What Is a Recurrent Neural Network (RNN)? | IBM’, 6 October 2021. <https://www.ibm.com/topics/recurrent-neural-networks>.
- [13] Hochreiter, Sepp, and Jürgen Schmidhuber. ‘Long Short-Term Memory’. *Neural Comput.* 9, no. 8 (1 0 1997): 1735–80. <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [14] ‘DDoS 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB’. n.d. Accessed 6 November 2024. <https://www.unb.ca/cic/datasets/ddos-2019.html>.

- [15] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 'Attention Is All You Need'. arXiv, 2 August 2023. <https://doi.org/10.48550/arXiv.1706.03762>.
- [16] Pramilarani, K., and P. Vasanthi Kumari. 'Cost Based Random Forest Classifier for Intrusion Detection System in Internet of Things'. *Applied Soft Computing* 151 (1 January 2024): 111125. <https://doi.org/10.1016/j.asoc.2023.111125>.
- [17] Chaolong, Jia, Wang Hanning, and Wei Lili. 'Research on Visualization of Multi-Dimensional Real-Time Traffic Data Stream Based on Cloud Computing'. *Procedia Engineering*, Green Intelligent Transportation System and Safety, 137 (1 January 2016): 709–18. <https://doi.org/10.1016/j.proeng.2016.01.308>.
- [18] Singh, Sunil Kumar, and Pradeep Kumar Roy. 'Detecting Malicious DNS over HTTPS Traffic Using Machine Learning'. In *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, 1–6, 2020. <https://doi.org/10.1109/3ICT51146.2020.9312004>.
- [19] Najafimehr, Mohammad, Sajjad Zarifzadeh, and Seyedakbar Mostafavi. 'DDoS Attacks and Machine-Learning-Based Detection Methods: A Survey and Taxonomy'. *Engineering Reports* 5, no. 12 (2023): e12697. <https://doi.org/10.1002/eng2.12697>.
- [20] Zilberman, Ben. 'Bot or Not? Fortinet Enhances Application Security with Advanced Bot Protection Service'. Fortinet Blog, 30 January 2024. <https://www.fortinet.com/blog/business-and-technology/fortinet-enhances-application-security-with-advanced-bot-protection-service>.
- [21] Le, Duc C., A. Nur Zincir-Heywood, and Malcolm I. Heywood. 'Data Analytics on Network Traffic Flows for Botnet Behaviour Detection'. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–7, 2016. <https://doi.org/10.1109/SSCI.2016.7850078>.
- [22] GeeksforGeeks. 'Port Scanning Techniques By Using Nmap', 6 July 2022. <https://www.geeksforgeeks.org/port-scanning-techniques-by-using-nmap/>.
- [23] 'What Is the Low Orbit Ion Cannon (LOIC)?' Accessed 27 October 2024. <https://www.cloudflare.com/learning/ddos/ddos-attack-tools/low-orbit-ion-cannon-loic/>.
- [24] Stevanovic, Matija, and Jens Myrup Pedersen. 'An Analysis of Network Traffic Classification for Botnet Detection'. In *2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 1–8, 2015. <https://doi.org/10.1109/CyberSA.2015.7361120>

9 LIST OF FIGURES

Figure 1.1: Abstract pipeline representation	4
Figure 1.2: Pipeline in case of a single honeypot node	4
Figure 2.1: Classification of honeypot systems	6
Figure 3.1: Abstract topology representation.	10
Figure 3.2: List of open TCP and UDP ports in the OpenStack environment.	11
Figure 3.3: DDoS attack visualization via the Attack Map service	11
Figure 3.4: Attack map of the main Sicherheitstacho T-Pot.	12
Figure 3.5: Username and password tag cloud, provided by Cowrie.	13
Figure 3.6: Visualization of DDoSPot statistics via Kibana service	13
Figure 4.1: Abstract representation of the LSTM model.	24
Figure 4.2: Content of the CICIDS2017 feature characteristics	27
Figure 4.3: Average validation loss values across all cross-evaluation folds	29
Figure 4.4: Average prediction accuracy across all cross-evaluation folds.	30
Figure 4.5: Average Confusion Matrix values across all cross-evaluation folds.	31
Figure 4.6: Cross-evaluation results and training of LSTM	32
Figure 4.7: DDoS detection report on incoming netflows.	33
Figure 4.8: CF metrics of the deployed model	34