TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Computer Science

Ali Ghasempour 194209IVCM

HTTP based Network Intrusion Detection System by Using Machine Learning-Based Classifier

Master's Thesis

Supervisors: Risto Vaarandi, PhD
Senior Researcher
Alejandro Guerra Manzanares, MSc
Junior Researcher

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia Teaduskond
Tarkvarateaduse Instituut

Ali Ghasempour 194209IVCM

HTTP põhine võrgu sissetungi avastamise süsteem masinõppe põhise klassifikaatori abil

Magistritöö

Juhendajad: Risto Vaarandi, PhD
Senior Researcher
Alejandro Guerra Manzanares, MSc
Junior Researcher

Tallinn 2021

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Ali Ghasempour

14.05.2021

Abstract

The constant increase of attacks on web servers poses threats to security, privacy, and service performance. To develop an efficient intrusion detection system and identify the intrusion, discovering the network behavior is necessary.

This thesis proposes behavioral intrusion detection systems based on machine learning techniques to identify attacks by analyzing the HTTP traffic. The approach studied in this works explores the HTTP header to extract the combination of meaningful features, followed by tree-based supervised machine learning algorithms, a creative learning strategy that highlights network intrusion from labeled data. During the learning phase, the algorithm uses labelled data for acquiring a skill to highlight network attacks. This research is based on the data collected from external network perimeter sensors of the Tallinn University of Technology.

The analysis indicates that the proposed framework correctly detects roughly 90% of intrusions. This is an indication that HTTP intrusion can successfully be dealt with using multi-level HTTP header analysis.

This research exposed the proposed framework's suitability to help with the security operation team's analysis procedures by automating intrusion detection.

Annotatsioon

Veebiserverite vastaste rünnakute pidev kasv ohustab turvalisust, privaatsust ja teenuste jõudlust. Tõhusa sissetungi avastamise süsteemi väljatöötamiseks ja sissetungi tuvastamiseks on vaja tunda võrguliikluse iseloomu.

Selles lõputöös pakutakse välja masinõppe tehnikatel põhinevaid käitumuslikke sissetungi avastamise süsteeme rünnakute tuvastamiseks HTTP liikluse analüüsimise teel. Töös uuritud lähenemisviis analüüsib HTTP päiseid oluliste omaduste kombinatsiooni leidmiseks, millele järgnevad puu põhised juhendatud masinõppe algoritmid - õppimise käigus omandab algoritm sildistatud andmete põhjal oskuse võrgu sissetunge esile tuua. Uurimistöö põhineb Tallinna Tehnikaülikooli välisvõrgu perimeetri anduritelt kogutud andmetel.
Analüüs näitab, et välja pakutud lahendus tuvastab õigesti umbes 90% sissetungidest. Seega saab HTTP sissetunge edukalt avastada kasutades mitmetasandilist HTTP päiste analüüsi.

Uurimistöö näitab pakutud raamistiku sobivust turvaoperatsioonide meeskonna analüüsiprotseduuride hõlbustamiseks tänu sissetungi tuvastamise automatiseerimisele.

List of abbreviation and terms

IDS	Intrusion Detection System
CPU	Central Processing Unit
JSON	JavaScript Object Notation
HTTP	Hyper Text Transfer Protocol
XGBoost	Extended Gradient Boost
IP	Internet protocol
TF-IDF	Term Frequency – Inverse Document Frequency
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol

Table of contents

List of Figures

List of Tables

1 Introduction

1.1 Chapter overview

This thesis is about implementing a network intrusion detection system based on a supervised machine learning model on HTTP traffic. The current chapter will discuss the importance of network intrusion detection for any organization. The hypothesis and motivation of this thesis are discussed in this chapter.

1.2 Introduction to network intrusion detection systems

In the last decade, the importance of deep analysis of the network traffic in the Internet and Intranet is growing. This need can be felt so that the Internet and being connected take all daily human life aspects, from the digital government to smart transportation to storing personal photos on clouds [1][2]. As a consequence of current ubiquitous digital data, sensitive and personal information needs to be guarded against being tampered with or damaged, and all systems need to have some degree of resistance against viruses and attacks. The computer network is playing a pivotal role in information exchange and security.

Thanks to recent developments, strong IDS (Intrusion Detection System) technologies are used to detect unusual activities.

Intrusion detection systems come in different shapes. It can be software installed on a computer or physical hardware mounted to a rack. An intrusion detection system's primary purpose is to monitor and analyze networks and hosts' activities and raise the alarm whenever an intrusion is detected. The functionality of IDS depends on its focus area. It can detect malicious activity over the network or find malicious processes on the operating system. Some variety of IDS are also capable of blocking or deterring intrusion.

There are two main types of IDS: host-based IDS (HIDS) and network-based IDS (NIDS). Network IDS is passively intercepting network traffic and inspects packets based on their characteristics and features. On the other hand, host-based IDS is sitting on endpoint computer systems and analyzing operating systems' internal processes. It has a wide variety of functionality like file system integrity checks or ransomware protection. Sometimes anti-viruses are acting as host-based IDS [3]. There is one more major type of IDS: a combination

of host-based and network-based intrusion detection systems. The new design is called hybrid IDS. It has the advantages of two significant IDS types and uses a more effective engine to detect malicious activities. However, it is rare to see two different IDS running and cooperating as a hybrid IDS because of the complexity of such a setup [4].

Suricata and Snort are the most common network-based intrusion detection systems which are both used commercially and non-commercially. However, there are other systems developed by companies like IBM, Fortinet, etc. IBM has an IDS product called Intrusion Detection and Prevention System Management [5]. Also, Fortinet has its own IPS system [6].

Generally, intrusion detection approaches can be divided into two main categories [7]:

- Signature-based: The main goal is to detect the attack by a known signature in the database. It is the most common way of malware and attack traffic detection and involves matching the malicious activity with the pattern that human expert defines.
- Behavioral-based: This model uses profiling techniques to build common traffic characteristics and detect abnormal activity.

Accuracy and performance are vital attributes of the intrusion detection system. Such a system should run as fast as possible without losing the accuracy detection rate.

1.3 Motivation and scope of the study

The signature-based analysis is largely blind to zero-day attacks since such attacks involve malicious network traffic that is usually not yet known to human experts who define signatures. Nowadays, attackers are tweaking their attack payload not to match any signatures before they initiate attacks. It gives them a competitive advantage compared to firewall and intrusion detection systems relying on rulesets. This weakness makes behavioral analysis an important intrusion detection technique. In the past decade, a lot of research has been done on traffic behavior analysis; however, most previously suggested methods were validated using public datasets available from Internet-based repositories. Many publicly available datasets are fairly old and do not reflect the traffic patterns of modern high-speed networks (for example, frequently used KDDCUP'99 dataset originates from the previous century). This raises the following question: are methods developed on old datasets still

valid for deployment in real-time networks? To answer this question, real-time data needed to be collected from a reliable point.

This thesis dataset is based on Tallinn University of Technology's external network perimeter traffic. Since network sensors can collect any data, there is the question of which protocol should be monitored. As the domain of this thesis, HTTP (HyperText Transfer Protocol) is selected due to its widespread use.

HTTP is an application-level protocol for hypermedia transmission information systems. It is a stateless and object-oriented protocol that can establish communication between servers and clients based on the method. Although HTTPS is one of the fastest growing protocol in internet however because of secure characteristics, it is not providing much information about nature of traffic. Moreover, HTTP still has been used by many applications. However if HTTPS traffic is considered, the proposed method can run after decryption process happen by web server or enterprise proxy.

The protocol involves using HTTP headers with several data fields, which allows for the definition of many features for machine learning algorithms.

1.4 Hypothesis and contribution

Intrusion detection systems are dealing with different issues such as performance lag in high-speed networks. Also, matching traffic with signatures can lead to a high false-positive rate, and some significant attacks may be missed. Therefore, employing behavioral analysis can bring more precision. All of the mentioned issues demonstrate a technological gap in this field. This thesis is trying to fill the gap by implementing a supervised machine learning model based on the analysis of HTTP traffic. The following research objectives are listed below:

- Study traffic behaviors for detecting unusual activities by analyzing HTTP packets.
- Develop a machine learning model to have high performance in a real-time network.

The proposed method first extracts raw HTTP header data fields and then derives features from them for machine learning algorithms. The raw HTTP data are provided by the Suricata IDS sensor, configured to produce a JSON record for each HTTP request and response seen

in the network. These raw data were captured from the external network perimeter of Tallinn University of Technology.

1.5 Novelty

Current study is proposing novel multi-layer features' extraction for HTTP traffic. The proposed framework is preparing the data for having a best match with tree-based learning algorithms. In addition, the framework provides a ledger for semi-automating data labeling.

Unlike the other studies using the labeled dataset for detecting malicious activities, this thesis detects outliers without relying on a fully labeled dataset. This approach increases the usability of the proposed framework in real-world scenarios. Furthermore, the current research is analyzing the behavior of the traffic regardless of attack type. This look helps to identify a broader range of attacks than other researches, limited to a list of attacks.

1.6 Outline of thesis

This thesis is organized into five chapters. Chapter one is an introduction to the study, problem statements, and research objectives. Chapter two reviews related research that has been done in the field of intrusion detection systems. Chapter three describes the methodology of this research. The multi-level HTTP features extraction and machine learning models are described in chapter three in a more detailed fashion. Chapter four is discussing machine learning model results as well as demonstrating the performance metrics. Chapter five analyzes the results and provides a discussion on model outputs. Also, possible future work is discussed in this chapter. Chapter six concludes the thesis.

2 Related works

2.1 Chapter overview

Before introducing the proposed model to tackle the research problems, it is wise to review previous methods and studies that have been done in the intrusion detection area. It is a vast area, and researchers address issues from different angles.

2.2 Overview of intrusion detection systems

The expansion of computer software and networks is introducing new attacks to this field. The network is the central part of communication among components. The network should be carefully monitored to detect and deter malicious activities. Network attacks can be categorized into two main parts: Active and passive. Active means intruders interact with the network by sending commands, but in passive, attackers only intercept network traffic. Some of the active attacks are [8]:

a. Spoofing: The attacker presents itself with a fake identity.
b. Modification: Any modification in the message route to cause a delay in communication.
c. Fabrication: Change the content of the message in a way to present false information.
d. Denial of services: malicious node sending massive traffic to saturate bandwidth as well as other node resources.
e. Sinkhole: The compromised node advertises its routing updated to other nodes to attract their network traffic.
f. Sybil: Deploying multiple malicious nodes in the network to increase the chance of attack.

Some of the passive attacks are:

a. Traffic analysis: The attacker measures the amount of data that are communicated between sender and receiver.
b. Eavesdropping: The attacker intercepts network traffic to collect any user information.
c. Monitoring: The attacker reads confidential data without modifying them.

There are some other types of attacks that do not fit the main categories:

a. Blackhole attack: The attacker uses routing protocols to set the best path for interception. This fake route can help the attacker to hide and monitor traffic in most favorite environments.
b. Rushing attack: The attacker captures the packet from sender to receiver, duplicates the message, and keeps sending the same packet to the receiver to exhaust the receiver's resources.
c. Reply attack: Malicious node repeats or delays the data.
d. Byzantine attack: Delay communication by setting up multiple nodes between sender and receiver. It can be done by rerouting or sending by a non-optimal path.

As mentioned in the previous chapter, intrusion detection systems are responsible for finding and alerting unusual activities in the respective layer. These IDS applications can work on many modes and analyze data by different approaches. Some aforementioned types of IDS are network-based, host-based, and hybrid models. In broad categorization, IDS can run into two primary modes: Signature-based or behavioral-based.

A short argument about the pros and cons of the mentioned methods is that the signature-based method generates fewer false alarms since it uses a predefined signature dataset. On the other hand, the behavioral method is more strong against unforeseen or zero-day attacks because of the profiling technique to detect attacks [9]. Speaking more about characteristics of the intrusion detection system, Debar [10] is listing the following properties:

- Accuracy: The system does proper detection without false alarm.
- Performance: An adequate processing time for audit events without missing an intrusion. Real-time processing is ideal.
- Completeness: An intrusion detection system must detect the attack. An incomplete sequence of detections can cause the system to stay useless.
- Fault tolerance: The system itself should be resistant to attack. It is crucial because most intrusion detection systems run above other systems.
- Timeliness: Intrusion detection systems should detect and alert about attacks as quickly as possible to avoid possible damage to other parts of the network.

Current intrusion detection setups rely on deploying multiple network monitoring nodes and capturing every possible packet. After the packet has matched some signature, the IDS

generates an alarm and sends the alarm (usually together with the packet payload) to a central server. Because of the large number of packets that can match signatures, some designs suggest adding Elastic Stack for organizing alarm data.

A lot of research has been conducted to find the most suitable approach for detecting malicious activities. The IDS can run according to the behavioral model. A behavioral model can be implemented in different ways, but in general, it can fit into three main categories: statistical-based, knowledge-based, and machine learning-based. With the statistical approach, statistical methods are used to define the legitimacy of traffic. It is a less efficient method since we do not use any expert system to support our decision. The knowledge-based approach uses all expert knowledge, such as operator feedback and protocol details, to determine the nature of the traffic. However, it is an expensive process, and it is rarely used. Finally machine learning model tries to find the nature of network traffic. Figure 1, is demonstrating the general taxonomy of behavioral approaches [11].

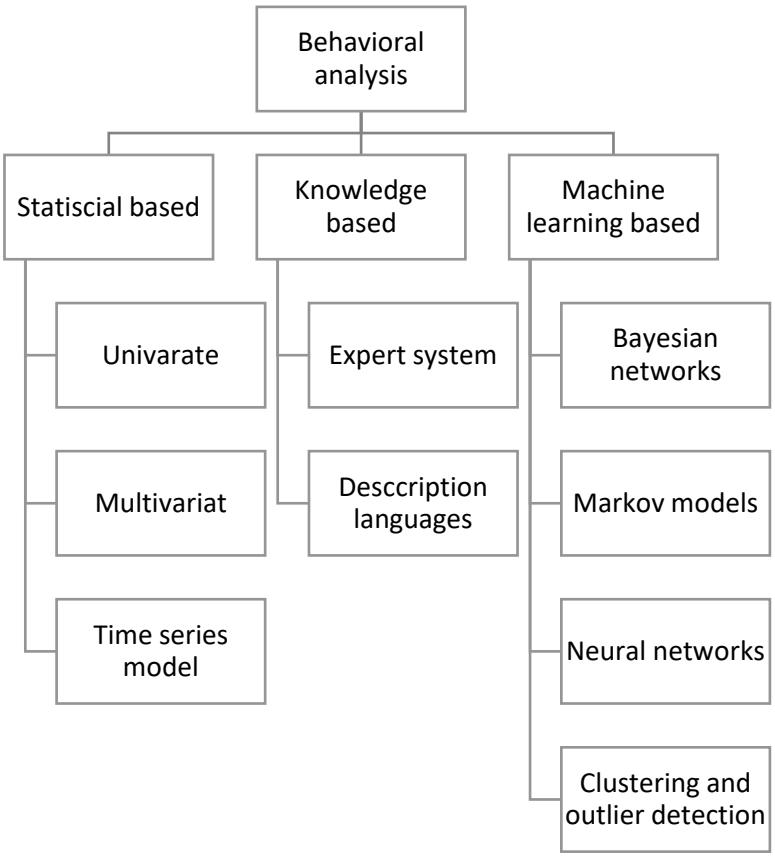

Figure 1 Taxonomy of behavioral analysis approach in intrusion detection systems

Also, behavior-based intrusion detection can be categorized differently. The new taxonomy is, supervised, semi-supervised and unsupervised. The supervised means to label data and build two distinguishable categories like normal and abnormal. Semi-supervised is building model only on routine data. It means we determine the usual data, and any outlier that cannot fit our model is abnormal. In the last technique, unsupervised, the data are categorized into clusters without knowing clusters. It is a less expensive model since it does not need any background information [12].

There are many types of systems available [13].

- Suricata: It is an open-source network intrusion detection tool based on the predefined signature set.
- Snort: It is another open-source network intrusion detection system introduced in 1998. It can also be used as a packet sniffer like TCPDUMP.
- Zeek (bro): It is a passive intrusion detection system capable of handling high-speed networks. It provides a scripting language that helps us to write our ruleset.

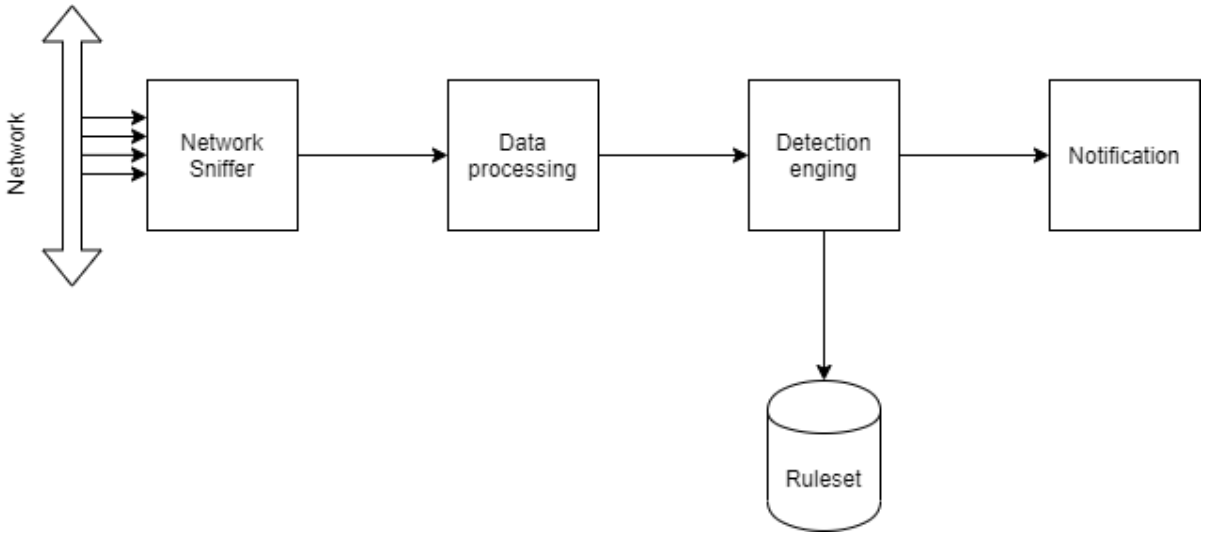

Figure 2 Suricata architecture

One of the main advantages of Suricata over Snort is, it can use advanced HTTP header parsing. Suricata has a better performance based on the measured benchmark than Snort while running in multi-threaded mode [14]. According to Suricata documentation, it can detect the following properties of HTTP requests and responses [15]:

- Stateful HTTP parser and HTTP transaction logger

- Keyword matching on buffers:
    - URI
    - Headers
    - Cookie
    - User-agent
    - Request and response body
    - Method and status code
    - Hostname

Tallinn University of Technology network setup includes Suricata as an intrusion detection component.

Machine learning and neural network solutions have been most frequently suggested in IDS-related research papers in the past decade. Accessing and producing massive data sets and improving hardware acceleration leads researchers to focus more on the machine learning approach [16].

Machine learning and neural network models mainly contain three main stages: data pre-processing, training, and testing. In simple words it means, transform raw data into encoded or pre-processed data. Divide data into training and testing portions, and then train the machine with training data to build the model. Test data can be used to check the built model to figure out the accuracy of the machine learning model [17]. Figure 3 demonstrates an overview of machine learning models.

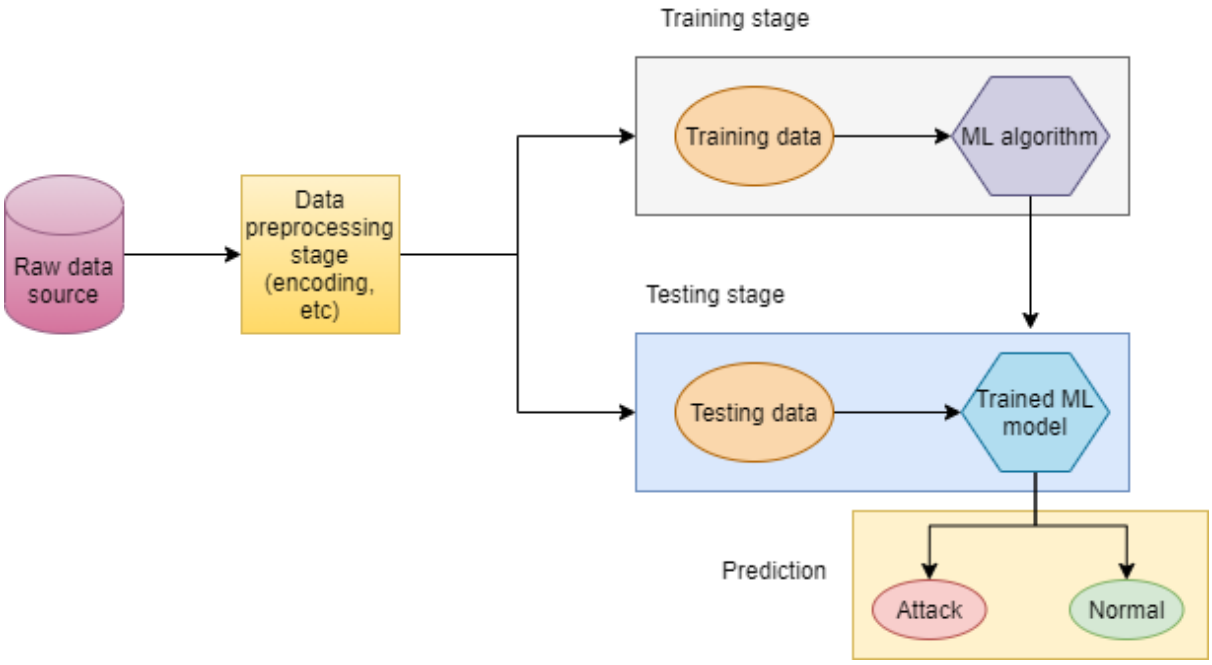

Figure 3 Overview of machine learning-based intrusion detection system

Deep learning is another cutting-edge method for analyzing the behavior of traffic. Deep learning improved machine learning techniques by adding more extra layers and optimized the features selection process. The layers are connected and use each other feedback to improve the model. It is a more efficient technique than a machine learning model since its layers select features. However, it needs more computational power [18]. In short, the difference between deep learning and machine learning are:

a. Deep learning needs a more extensive dataset to analyze compared to machine learning models.
b. Feature selection mostly happens by layers in deep learning; however, machine learning features are introduced to the machine.
c. Deep learning techniques require high computational power and a more prolonged training phase.

2.3 Literature review

For adequately analyzing this research, different angles need to be covered. Because proposed framework contains different parts, and each one has different methods of

implementation. Firstly previous related works are discussed, and after that, the best method for each component is discussed separately.

Recent research papers indicate that anomaly detection methods have featured high detection rates [19]. However, it cannot be seen as a highly developed research field since attacks and attack methods are constantly changing [20]. One of the issues that many current studies have is that they mostly use old publicly available datasets, for example, the KDDCUP'99 dataset [21]. This dataset was released in 1999, and since then, it has widely been used for other network anomaly research. This dataset contains a record with labels for "bad" and "good." One of the issues this dataset is facing is that there are multiple duplicated rows available [22]. Due to this, machine learning training and testing datasets are similar and produce better detection rates. Although a new version of NSL-KDD has been released, the validity of this dataset has been questioned [23], [24], [25].

As mentioned in the previous chapter, selecting the right approach is essential. Classification of data is one of the methods that most of the research is employing. Mustapha et al. [26] made a simple comparison of different classification methods, namely SVM, Naïve Bayes, decision tree, and Random forest, over KDD dataset using Apache spark. Results show that Random forest is more accurate with the lowest prediction time; however, Naïve Bayes was selected as the fastest training model. Also, it is crucial to notice which attack method research is targeting. Pasumpon et al. [27] studied DDOS attacks in the network. The author proposed a combination of neural networks and SVM as detection and classification tools. The research flow elaborates that to label the input data, and they applied quadratic entropy to define a regular traffic threshold. However, involving multiple neural network layers in the model increases the detection time significantly, and accuracy is not promising. Detecting DDOS attack which is coming from distributed sources can be challenging. Botnets are taking a considerable share in cybersecurity threats in 2020. According to the ENISA Botnets C&C server, it increased 71.5% compared to 2018 [28]. These statistics have motivated researchers such as Tong et al. [29] to propose a machine learning model to detect Botnet DDOS attacks. They improved feature selection to increase the robustness of the model. The author picked traditional classifiers like SVM, decision tree, Naïve Bayes, Artificial Neural Network (ANN), and unsupervised learning (USML) to discover insider threats. Overall results based on two different datasets indicate that unsupervised learning had more accuracy among present methods. Other researchers have suggested a semi-

supervised clustering method for detecting DDOS attacks. Muhammad et al. [30] proposed two parallel clustering techniques aggregated by one voting system and feed labeled data to supervised learning. The author proposed a voting method to check and compare different cluster outputs. On one side, Principal Component Analysis (PCA) and K-means clustering and another side Agglomerative clustering have been selected. Their research used simulated data generated by OPNET, but it is questionable if it would produce the same results for real-world traffic or not.

To study network attacks' nature, it is helpful to analyze the features of a specific application layer protocol. This research mainly studies HTTP in the application layer. HTTP header contains different parameters that can be used as features for measurements [31]. HTTP flood attacks can be categorized as one of the DDOS attacks. Indraneel et al. [32] proposed to measure each session time and page count to distinguish DDOS from normal behavior. The author measured the absolute deviation session time frame and clustered them. As a clustering algorithm, the BAT algorithm was selected, representing the behavior of BATs in nature, and results indicated that results are more promising than results reported in other research papers.

URI and request methods are exciting features to analyze [33] to better understand HTTP header and user activity. Pattern analysis is the primary method to decode URI. Yuqi et al. [34] proposed semantic structure detection for HTTP traffic. They assumed that malicious traffic has repeating patterns and divided patterns into multiple states. To improve prediction, the author used Bidirectional Long Short-Term Memory (Bi-LSTM) for profiling the patterns. The results show false positive rate is below 1%. Ke et al. [35] introduced MalHunter, based on malware's statistical characteristics in HTTP protocol. The author extracted character-level features of URLs and applied separate statistical analysis for each feature. Also, each field of the HTTP header is going through the same process.

Sornsuwit et al. [36] use different techniques such as K-NN, C4.5, MLP, SVM, and LDA to boost data classification. The authors are trying to optimize input data as much as possible to improve detection accuracy. The proposed method consists of several phases, and in each phase, the data is applied to multiple classifier algorithms. Also, some researchers are focusing on neural networks. Zhang et al. [37] employed Mind Evolution Algorithm (MEA) and Genetic Algorithm (GA) to reduce input data features and using the classifier method in the following steps. Adem [38] focused on URL and payload parameters as the primary

input for Convolutional Neural Network (CNN). HTTP payload is converted into bag-of-words representation. The results indicate that an almost 97% recall rate can be achieved.

Besides standard classification algorithms such as Random Forest, XGBoost, and precision and recall rates are reporting, XGBoost has better results. Yong et al. [39] mainly focused on IoT devices which are using HTTP traffic for communication. The author proposed a framework that divides HTTP requests into multiple parameters and uses Hidden Markov Model (HMM) to detect attack scores. At the final stage, a voting score is implemented to trigger an alert based on previous scores. John et al. [40] had a deep analysis of near 50,000 HTTP headers and proposed 11 new HTTP header features that are not common among previous research. Surprisingly some header fields such as "cache-control no-cache" demonstrate significant rank in the selection process. The reported features are rarely used in other similar studies. Stefano et al. [41]studied the difference between human and bot activity on different websites by classifying them using unsupervised and supervised learning. Web server access log is the primary input data source of this research, and each session is labeled based on different databases such as the Udger database [42], which contains 43 legit user agent strings. It is closely related to the current research proposal; however, the current study's main aim is to find malicious activities regardless of bot or human. Ashley et al. [43] collected a dataset similar to this research's primary intended data source, coming from university web server logs in 42 days. The author collects both request header and payload and generates a key-value matrix. Labeled features are passed to Random Forest, XGBoost, and Decision tree classification algorithms. Based on importance rank, "accept-encoding" is the most common feature among their dataset, and result-wise, XGBoost has more accuracy in a shorter timeframe.

Suricata and ElasticStack are core tools for this research. Suricata collects all information across the network and stores collected data in elastic search [44][45]. In recent years some extensions have been developed for ElasticStack as the automated detection system. Wazuh is an extension in Kibana which can detect anomaly based on population analysis [46]. Ovidiu et al. [47] proposed another ElasticStack extension to detect anomalous activities and present them in the Kibana dashboard. Z. Chiba et al. [48] have proposed using Suricata as signature-based anomaly detection and use Isolation Forest Algorithm (IFA) as the second layer of detection. The IFA algorithm is detecting anomalies by not using distance or density

measures. This technique is decreasing memory usage since there is no need for creating different trees in memory.

In the following table, selected researches are organized by year and explained for ease of reading.

NO	Autor, Year	Method	Result
1	Mustapha et al. [26], 2018	SVM, Naïve Bayes, decision tree, and Random forest on KDD dataset and Apache spark	Random forest selected the most accurate method. Naïve Bayes was fasted training method.
2	Pasumpon et al. [27], 2019	Complex neural networks model on DDOS attacks.	The proposed method has significant detection time as well as low accuracy.
3	Yong et al. [39], 2019	Parse HTTP header from IoT devices and use hidden Markov model.	The author reported algorithm can detect SQLi and XSS attacks with high accuracy.
4	Muhammad et al. [30], 2019	Cluster data by two parallel voting methods. K-means and Agglomerative clustering.	Results can not be confirmed since it is based on OPNET
5	Indraneel et al. [32], 2019	Detect HTTP DDOS attacks by measure session time and page count.	The author compared his BAT method to other methods, and it shows 94.8% accuracy.
6	Sornsuwit et al. [36], 2019	Heavy data pre-processing for maximum performance. Multi-layer data classification.	The author used K-NN, C4.5, and SVM and compared them with his method. The reported accuracy is 99.98% by their method.
7	Stefano et al. [41], 2020	The author aimed to differentiate bot activities from the human by analyzing web server log.	The data classification has not been done precisely.
8	Tong et al. [29], 2020	SVM, decision tree, Naïve bayes, ANN and USML on Botnet DDOS data.	Based on results, the unsupervised learning had better detection accuracy.
9	Adem [38], 2021	Apply URL and payload to a convolutional neural network (CNN).	Almost 97% recall rate had been achieved by method.
10	Ashley et al. [43], 2021	Analyze HTTP header and payload by Random Forest, XGboost, and decision tree algorithms.	It is one of the most valuable researches since it is based on actual data. XGBoost was selected as the most accurate algorithm.

Table 1 Review of selected researches

3 Design and methodology

3.1 Chapter overview

The following chapter is a discussion on the primary research body. In the general overview of current chapter points, the system's model and architecture have been discussed. Besides, technical requirements and the experiment environment are described in detail. Some more technical improvements have been proposed to make research results more straightforward to be implemented in the real world.

3.2 Infrastructure and environment setup

The setup can be divided into two main components: data collector node and analyzer system.

In the role of the data collector node, we have used a Suricata IDS sensor on the external network perimeter, which raises not only alarms about malicious traffic but also generates records for legitimate network traffic for most common application layer protocols (for example, HTTP, SMTP, DHCP, etc.).

As the domain of this thesis, only HTTP traffic data in JSON format is considered. All incoming HTTP traffic from external sources to TalTech hosted services is in this research analysis scope to be more precise.

The second component is the analyzer machine. The machine is hosted in Tallinn University of Technology, Computer science department. It is the main computer for the rest of the analysis, which has been done for this research. The server technical hardware configuration is:

- CPU Intel® Xeon® CPU E5-2630L v2, 2.40GHz, 15 MiB L3 cache
    - 24 CPU cores.
- 64 GB (8x 8GB) DIMM DDR3 1500 MHz Kingston memory
- RAID1 – 2x Samsung SSD 860 250GB

- 2x Intel Gigabit Network connection, 1Gbit/s

The server's chosen operating system is CentOS Linux version 8, running on kernel 4.18.0 x86_64 Linux. Python is the primary language for analysis. Python 3.6.8, in addition to extra libraries such as Numpy, Pandas, etc., are used.

The server is configured to collect HTTP traffic from Tallinn University of Technology's external perimeter is transferred into the server by rsyslog tool. The data are stored in a single file per day. The information flow is started on 23 November 2020.

Figure 4, is demonstrating the desired setup.

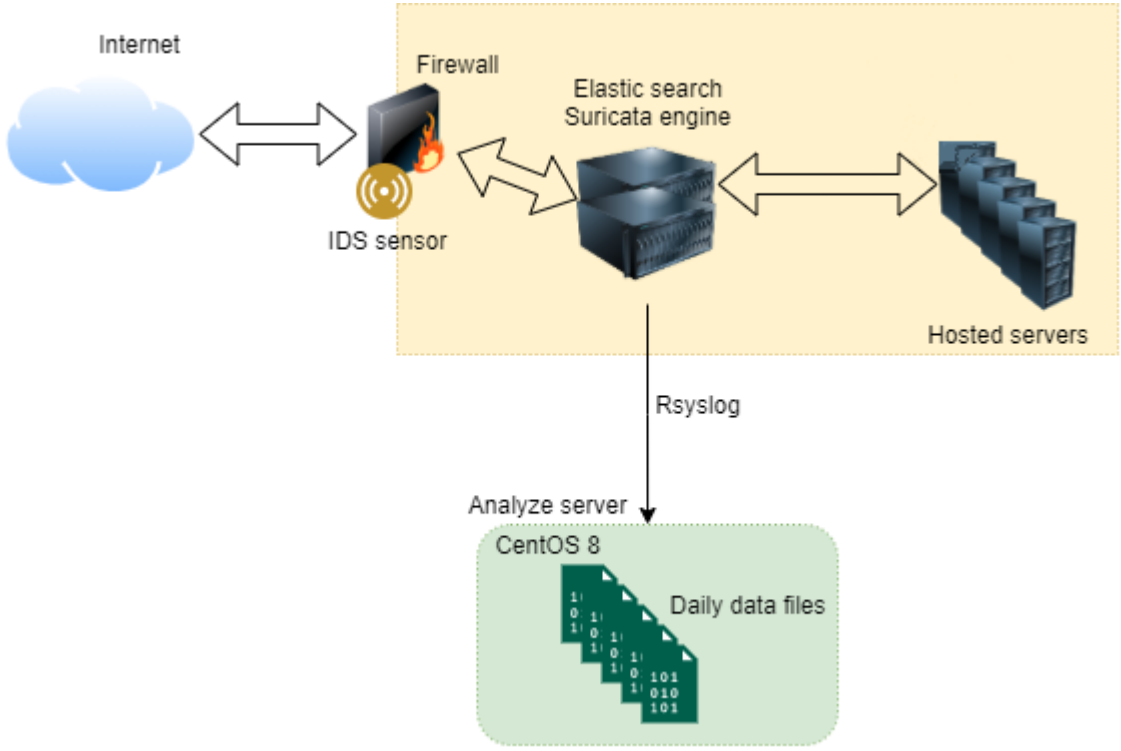

Figure 4 Infrastructure and Environment setup

3.3 Overview of dataset

According to described architecture, the data are collected from the IDS sensor and transferred into a server by syslog protocol. Data are stored in flat files, and each file represents one day. The files contain JSON records; however, a timestamp is also added at each record's beginning. Following is the sample record from 1 April.

```
Apr 1 00:00:03 extids-ict suricata @cee: {"timestamp":"2021-04-
01T00:00:03.401478+0300","flow_id":514485052991264,"in_iface":"eno4","ev
ent_type":"http","src_ip":"AA.BB.CC.DD","src_port":XXXX,"dest_ip":"AA.BB
.CC.DD","dest_port":XXXX,"proto":"TCP","tx_id":0,"http":{"hostname":"onl
ine.msi.ttu.ee","url":"\/tallinn\/graafik.php?_jpg_csimd=1&hoovus","http
_user_agent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/89.0.4389.82
Safari\/537.36",
http_content_type":"image\/png","redirect":"http:\/\/on-
line.msi.ttu.ee\/tallinn\/","http_method":"GET","protocol":"HTTP\/1.1","
status":301,"length":4433}}
```


Each record contains captured HTTP transaction which the IDS sensor has detected. The record represents HTTP requests to the server and the server response to the client. However, if the server does not provide any response, therefore it only represents the request.

The provide JSON can contain multiple values. These values are not fixed and can be seen based on the availability of them. In general, most records have the following attributes:

timestamp		Transaction timestamp
flow_id		Suricata bidirectional flow tracker number
in_iface		Captured interface name
event_type		Indicate logging protocol type
src_ip		Source IP address
src_port		Source port
dest_ip		Destination IP address
dest_port		Destination Port
proto		IP protocol in the packet header
tx_id		Suricata transaction ID
HTTP	hostname	The hostname of the HTTP event
	URL	Visited URL at hostname
	http_user_agent	User-agent software
	http_content_type	The datatype of the HTTP record
	http_method	HTTP method such as GET, POST, etc
	protocol	The version of the HTTP protocol
	status	HTTP status code
	redirect	HTTP redirect
	length	Content size of the HTTP body

Table 2 Selected attributes from dataset

During the experiment period, daily log files had the size of up to 1.3 GB per day. However, to save disk space, older files were compressed.

3.4 System architecture

This research proposes multi-layer feature extraction, semi labeling data, and machine learning models to detect and predict malicious activities. Figure 5 is demonstrating the general overview of system architecture.

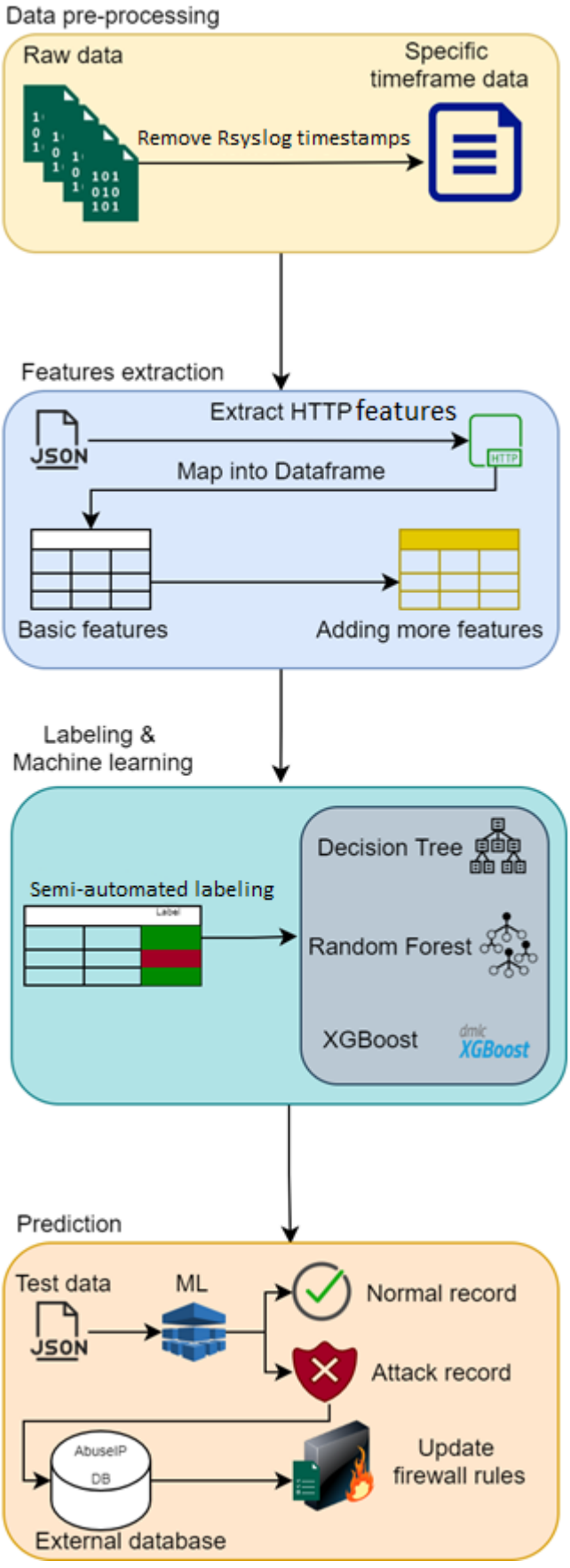

Figure 5 Proposed system architecture

3.5 Data pre-processing and native features extraction

The original data files need to be pruned in such a way that the additional timestamps should be removed. This process has been implemented with UNIX tools. Also, to have a better analysis, it is necessary to aggregate a few days' files into a single file. It would give a more realistic view of data instead of looking at a specific timeframe. Also, mixing data from working days (Monday – Friday) and weekends helps create a more realistic data set that describes traffic nature for different days of the week. According to the analysis, based on computational and storage limitations, selecting three days is optimal. A new data file has a size of around 4GB and 5 million records.

One of the main components of this research is extracting every possible feature from the dataset. It means that alongside provided Suricata HTTP features, other features need to be extracted.

The process is starting with mapping JSON data to Pandas DataFrame. Panda DataFrame is a two-dimensional data structure [49]. It contains rows as records and columns as features. The table has heterogeneous characters and can store most of the datatype in it.

To build DataFrame from 5 million data records, it is necessary to use multi-threaded processing to increase data mapping pace. By nature, Python is not running on multi-thread scheduling and can utilize only one CPU simultaneously. Therefore, the dataset is divided into chunks of multiple lines, and a Python thread is executed per each chunk to utilize maximum CPU power. Also, this method can be used during real-time analysis to increase the data processing speed. After mapping all Suricata data into DataFrame, it would look like the following figure:

new_timestamp	src_ip	src_port	dest_ip	http	...	protocol	status	redirect	length
2020-12-06 00:00:00.211430+02:00				{'hostname': 'idp.ttu.ee', 'url': '\\Vsimplesa...	...	HTTP\/1.1	302.0	http:\/\/idp.ttu.ee\/simplesaml\/saml2\/idp\/S...	1501.0
2020-12-06 00:00:00.271667+02:00				{'hostname': 'deepthought.ttu.ee', 'url': '\\V...	...	HTTP\/1.1	200.0	NaN	3832.0
2020-12-06 00:00:00.629373+02:00				{'hostname': 'on-line.msi.ttu.ee', 'url': '\\V...	...	HTTP\/1.1	200.0	NaN	1237.0
2020-12-06 00:00:00.742654+02:00				{'hostname': 'on-line.msi.ttu.ee', 'url': '\\V...	...	HTTP\/1.1	200.0	NaN	154.0
2020-12-06 00:00:00.857020+02:00				{'hostname': 'on-line.msi.ttu.ee', 'url': '\\V...	...	HTTP\/1.1	200.0	NaN	1236.0

Figure 6 Sample DataFrame

3.6 Advanced features extraction

Extracting meaningful features out of currently available parameters requires more profound analysis.

3.6.1 HTTP methods feature

To start, http_method is a favorable feature to analyze. HTTP method shows the desired action for specific resources. According to Mozilla, there are nine most common methods. These methods are: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, and PATCH [50].

Mentioned methods are also most common in our dataset. Figure 7 demonstrates the distribution of the HTTP method over the selected dataset. In this context, the selected dataset is December 6 2020 captured data. According to the graph, most HTTP method occurrences on the selected dataset belong to common HTTP methods introduced by Mozilla.

There are many different HTTP methods in selected datasets; therefore, to plot them, the encoding technique has been used. It means each HTTP method assigns to one number

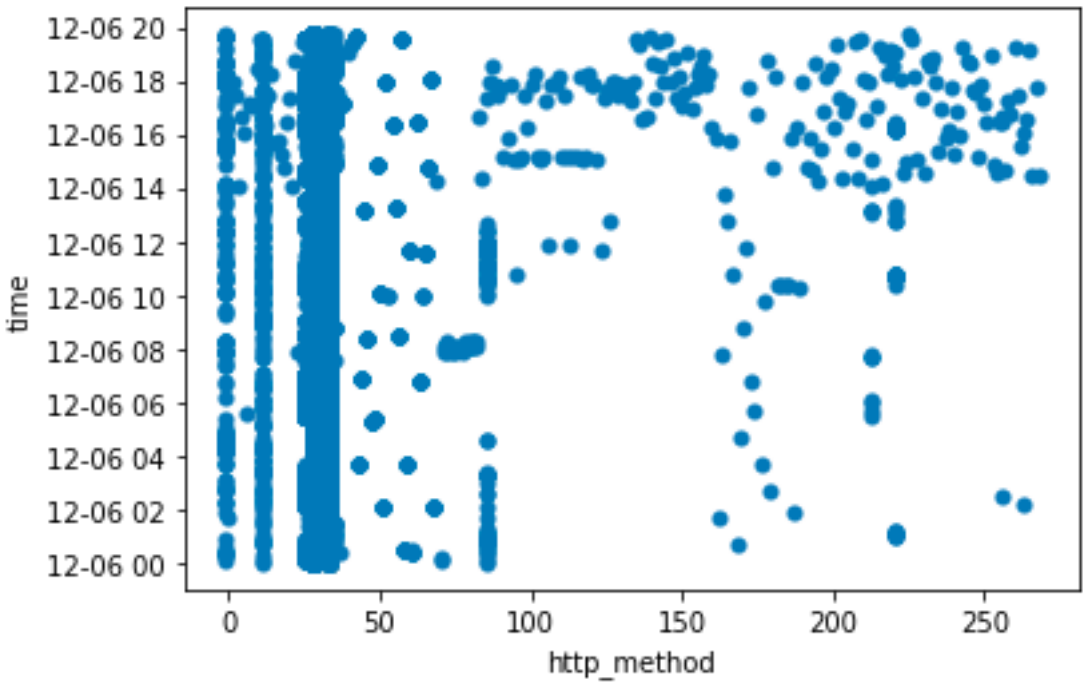

Figure 7 Distribution of HTTP methods. 26 < GET,POST, HEAD, DELETE, PUT < 39

Also, among common methods, GET is the most common one. Figure 8 shows, GET has the most requested method following by POST as the second one.

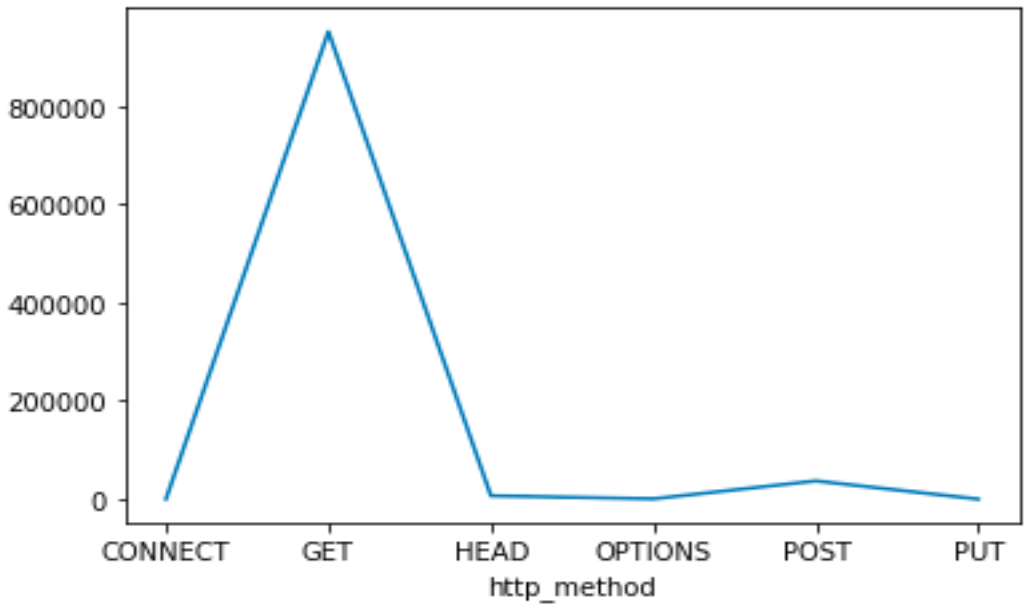

Figure 8 Count of common methods over the dataset

Based on measurements, odd HTTP methods can be counted as not standard methods or containing malicious requests. By odd methods it means, it is not part of Mozilla top methods. Therefore a new feature called "uncommon_http_method" is added to the dataset. Further validation on the dataset revealed that most odd HTTP methods carry a malicious payload.

3.6.2 HTTP status code feature

HTTP status code is the second feature that can be analyzed directly. The status code shows the HTTP response to a specific request. Status codes can be categorized into five main ranges [51].

1. Information responses (100 – 199)
2. Successful responses (200-299)
3. Redirects (300-399)
4. Client errors (400-499)
5. Server errors (500-599)

Each range contains several status codes which have a specific meaning. For example, status code 301 is about the requested URL moved permanently, and 404 manifests the fact that the requested page does not exist.

Typically, security analysts consider status codes above 400 as malicious or needing a closer review. Most manually crafted scanning, bot scanning, and brute force attacks produce client errors (codes 400-499) from servers. Although the 500-599 range belongs to server errors such as application failure or web server faults, attackers might inject some arbitrary codes into the application, which causes it to crash and return server errors [52]. Therefore in most cases, it is a valid assumption to consider codes above 400 as unusual activity. However, the above statement does not imply that all of the other status codes belong to the normal range. If the attacker successfully requests or posts payload to the server, the webserver would reply with code 200. Also, in some cases, the attacker might request an old URL or the URL redirected elsewhere. In this situation, the server responds with 300 range. The 100 range is most common among all connections since it belongs to a web socket connection.

Figure 9 is showing the most common status codes in are 100, 200, and 300 ranges.

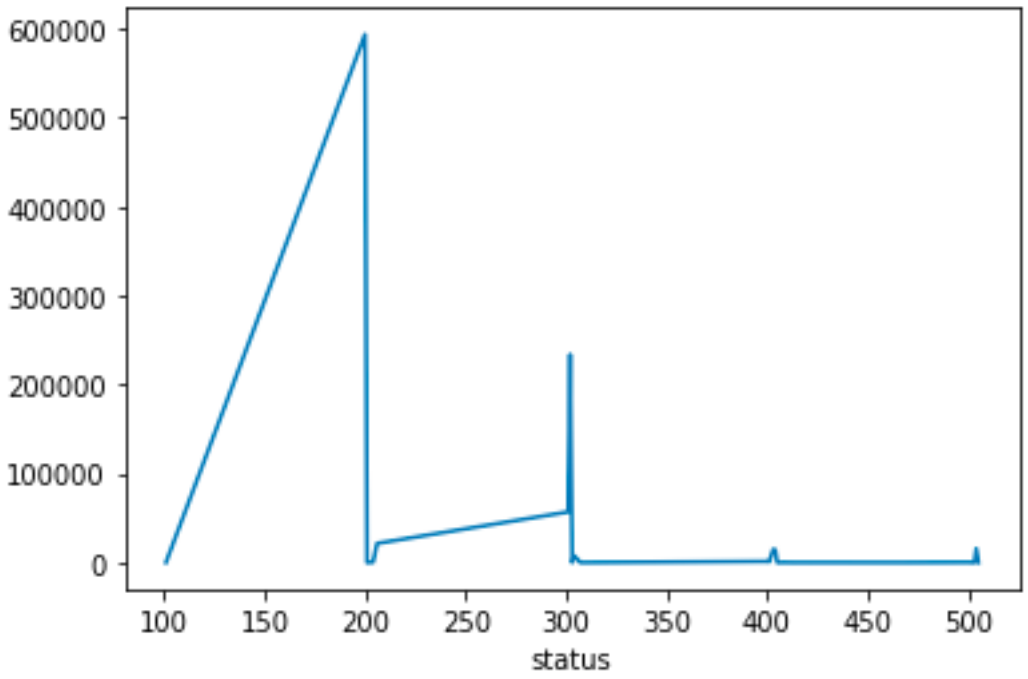

Figure 9 HTTP response code distribution

After manually checking the selected dataset, the following status codes which are below 400, are most common ones:

- 101: It is a response to the upgrade header request from the client.
- 200: Means OK. However, the code has different meaning for different methods :
    - GET: Request fetched successfully.
    - HEAD: The header is inside the message.
    - POST/PUT: Request submitted successfully.
- 206: The client is requesting part of the resource.
- 301: URL has been moved permanently, and new URL provided.
- 302: Requested URI has been changed temporarily.
- 304: It tells a client that the information is the same and can use its cache.
- 307: The same action as what 302 does, however the HTTP method cannot be changed.

After excluding mentioned status codes, the distribution of the status codes over the dataset is looking like below:

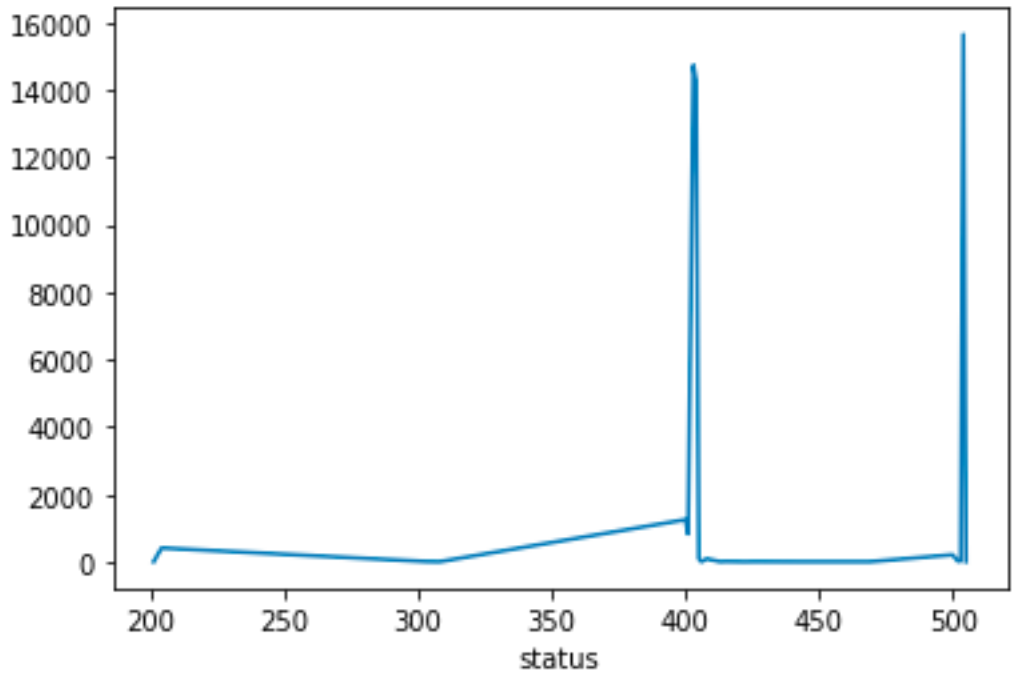

Figure 10 Updated HTTP status code distribution

The new feature is called "odd-status-code". Every transaction which does not have mentioned status codes (the commonly recognized status codes) has been marked.

3.6.3 HTTP content length feature

The third extracted feature is related to HTTP content length. Content-Length is showing the size of the body which has been sent to the recipient. Applications are using the content-length parameter to indicate the transfer length of the message body. This feature does not have any meaning by itself. However, if we measure it by the time, a normal distribution pattern appears. The Kolmogorov Smirnov test has been conducted on data to prove the normal distribution [52]. The test shows the data is normally distributed on most of the different timestamps. However, there are cases that the data is not normally distributed.

Figure 11 shows over random rows on dataset content length have shown the normal distribution pattern.

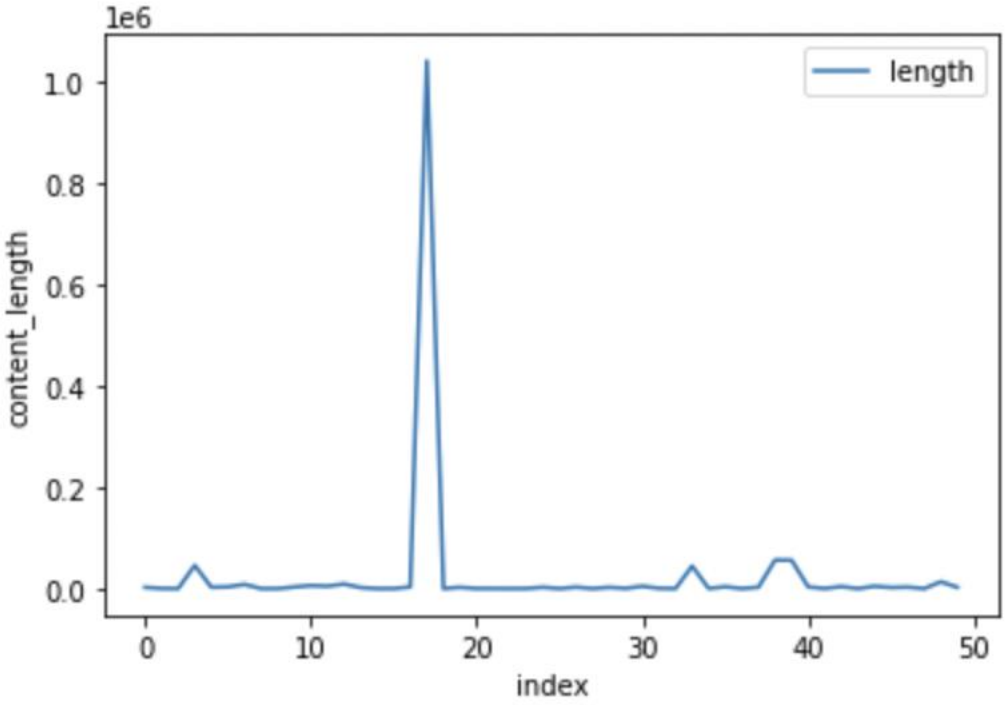

Figure 11 Content-length value on random records in a dataset. The index represents the index of random records in the dataset.

One approach calculates the mean for data and assigns the data above the mean as more significant data. It would provide us the possibility to detect large data transfers which have been conducted over the server. Since our domain is limited to external IPs approaching our internal IPs, content length can be interpreted as the length of data requested by an external host. In some cases, if an attack is successful, the length is considerably larger than average.

Therefore, after aggregating several days of data, the total mean can be used to find records above the mean. The new feature is called "content-length-above-mean."

3.6.4 HTTP user agent feature

The next following exciting feature which significant in the dataset is HTTP User-Agent. It is information from the client that reports the application, operating system, vendor, and version of the requesting agent [53]. The provided information is varied and depends on the client software. The User-agent field has a general pattern:

```
User-Agent: <product> / <product-version> <comment>
```

And in a common format from browsers:

```
User-Agent: Mozilla/5.0 (<system-information>) <platform> (<platform-details>) <extensions>
```

User-agent is the pervasive field. There are many different agents (different in software, browser, operating system, version, etc.) that are used to visit university webservers. Also, in our dataset, we have roughly around 1 million records for each day. Therefore it would be impossible to find meaningful information by only looking at the user-agent feature. There are some standard user-agent that can be seen in our dataset.

Firefox user agent string:

```
Mozilla/5.0 (platform; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
```

"`geckoversion`" is the firefox version, `and "geckotrail`" indicates a browser-based Gecko.

Example of Chrome user agent string:

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
```

Example of Safari user agent string:

```
Mozilla/5.0 (iPhone; CPU iPhone OS 13_5_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1.1 Mobile/15E148 Safari/604.1
```

Example of Internet Explorer user agent string:

```
Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0;
IEMobile/9.0)
```

Example of Crawler or bot user agent string:

```
Mozilla/5.0          (compatible;          YandexAccessibilityBot/3.0;
+http://yandex.com/bots)
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
```

Furthermore, lastly, an example of the library or net tool in a collected dataset.

```
Curl1/7.64.1
PostmanRuntime/7.26.5
```

More information about user-agent types can be found in RFC 7231 [54].

Figure 12 is showing all of the user agents which contain "bot" in the name.

0	Mozilla\/5.0 (Linux; Android 6.0.1; Nexus 5X B...	1
1	Mozilla\/5.0 (Macintosh; Intel Mac OS X 10_10_...	1
2	Mozilla\/5.0 (compatible; AhrefsBot\/7.0; SA; ...	1
3	Mozilla\/5.0 (compatible; Gowikibot\/1.0; +htt...	1
4	Mozilla\/5.0 (compatible; YaK\/1.0; http:\/\/l...	1
..	...	...
75	Mozilla\/5.0 (compatible; MJ12bot\/v1.4.8; htt...	7909
76	Mozilla\/5.0 (compatible; AhrefsBot\/7.0; +htt...	16303
77	Mozilla\/5.0 (compatible; DotBot\/1.1; http:\/...	20418
78	Mozilla\/5.0 (compatible; SemrushBot\/7~bl; +h...	33060
79	Mozilla\/5.0 (compatible; YandexBot\/3.0; +htt...	222457

Figure 12 HTTP user agent occurrence with bot keyword

After manual dataset reviewing, it turns out that, based on current features, there is a pattern for malicious HTTP user agents. The pattern is based on newly introduced features. It is trying to identify most of the odd behavior that has been observed. The rule for determining if the user agent is malicious is the following:

$$(\text{odd} - \text{status} - \text{code} == 1 \text{ OR uncommon} - \text{HTTP} - \text{method} == 1\,)\ AND\ (\ \text{content} - \text{length} - \text{above} - \text{mean}\ == 1 \text{ OR dest} - \text{port}\,! = 80)$$

If an HTTP transaction matches the above rule, the user agent from that transaction is regarded suspicious, and the "suspicious-user-agent" feature is set to 1 for all HTTP transactions with the given user agent.

3.6.5 HTTP URL features

So far, most of the selected features did not need extensive and deep analysis over specific built-in features. However, the "URL" feature can provide us much information. This feature is combined with "hostname" to build the full requested domain and address. The URL features contain much information to extract. It mostly reveals the nature of the request. There have been many methods for extracting meaningful information. As the first step, keyword scanning has been used. In addition to regular expression-based matching, most of the current signature-based intrusion detection systems are often also relying on substring matching for URLs. It is useful to extract the top common substrings (henceforth called *keywords*) among the attacks based on previous studies. Also, keyword selection is based on Tallinn University of Technology's Computer Emergency Response Team (CERT) database for attacks.

Keywords are the specific command, file extension, or method that have been used in the URLs. For example, there should not be any system or application command execution in most legitimate requests. The following table is demonstrating all of the keywords and respective categories which have been used as features.

Compressed file	.tar	PHP	.php
	.gz		.phtml
	.xz		.php3
	.bz2		.php4
	.7z		.php5
	.zip		.phps
			phpmyadmin
	.rar	Python	.py
Interpreters	bash		.pyc
	perl		.pyo
	ruby		.pyw
	exec		.pyd
	.rb	Connection tool	nc
	.pl		wget
	.sh		curl
	.exec	JS functions	eval()
	.cgi		link()
Others	bot		unescape()
	sql		search()

Table 3 Select keywords on URL field

Every mentioned keyword is added as a feature to the dataset, and it has a binary value. If the keyword is present in the URL, the value would be 1 otherwise 0.

The second step for URL analysis is length. The URL length can be interpreted in different ways. Firstly, the number of characters inside the URL has been considered. It does not have any meaning by itself; however, it is adding more details to the analysis. Figure 13 shows most URLs have lengths up to 100 characters, and only a few are longer.

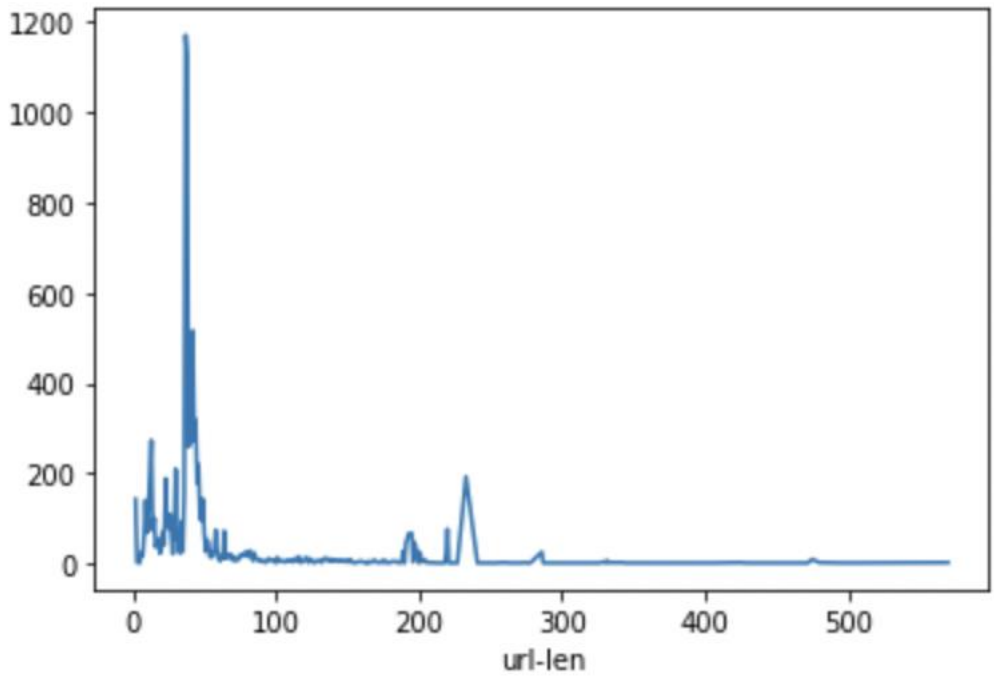

Figure 13 URL length on 10K sample

The number of words is also another feature. It means dividing the URL into parts (words) by considering special characters which are used in the URL as delimiters between words. Special characters which are used as delimiters to extract words from the URL are: `% & = / \ : // \\ ? .`

For each record, a new numeric feature is created, set to the total number of words used in the URL.

One technique to detect outliers based on URLs is to identify less frequent URLs among the dataset. It means most common URLs are repeated over the dataset, and probably they represent legitimate traffic; however, a specially crafted request that includes a non-generic function to perform malicious activities can be seen rarely on the dataset. The proposed technique is calculating the coefficient for each record to describe how rare is the URL. This technique utilizes currently collected information such as the number of words and uses statistical methods to calculate the weight.

The algorithm is divided into 6 steps. In steps one and two, the words and occurrence of them in each records are counted. The steps 3 is counting all of words for each records. Then count total number of words inside the dataset in step 4 and in the step 5 calculate the occurrence of each word per dataset. Lastly, in step 6, if the occurrence of word is below

50% therefore the word counted as less common word and affect the total coefficient number. To adjust most effective threshold, algorithm has been run multiple times. Also since 50% is relatively high number to be threshold therefore it omits only most common words.

Following Algorithm 1 is explaining the steps. The author has proposed the following algorithm.

Algorithm 1: Finding uncommon URL in the dataset

Input : Dataset D

Output : Occurrence coefficient for each row in D

Step 1 : Dataset has m transaction, and each transaction has n words. Every word can be repeated C times in a transaction.

Step 2 : //Extract words as *W* from each row of the dataset

$$C_1 \times W_{1,1}, C_2 \times W_{2,1}, C_3 \times W_{3,1} \cdots C_n \times W_{n,m}$$

// Where C_n represents the number of repeats for a word $W_{n,m}$. $W_{n,m}$ Represents the word *W* in record *m* and with *n* position.

Step 3 : //A total number of each word in the dataset as $Total_W$. The W_n is the *n*th word.

For all m in D **do**

$$Total_W = Total_W + C_n \times W_n$$

End for

Step 4 : //Total number of words in the dataset as $Total$

$$Total = Total_{W1} + Total_{W2} + Total_{W3} + \cdots + Total_{Wn}$$

Step 5 : //Calculate occurrence of each word $Total_W$ per the dataset D

$$Occurrence_W = \frac{Total_W}{Total}$$

Step 6 : //Adding new feature called "URL-odd-words-coefficient"

For all m in D **do**

If $Occurrence_W < 50\%$, **then**

$$Coefficient_m = Coefficient_m + Occurrence_W$$

End if

If there is any $Coefficient > 1$, **then**

$$Coefficient_m = \frac{Coefficient_m - \min(Coefficient)}{\max(Coefficient) - \min(Coefficient)}$$

End if

End for

In short, the technique is going through each row and count the occurrence of less common words in the URL. Then normalize data to range zero to one. Therefore, every row with a higher number would have a higher coefficient of being a unique URL. Figure 14, is demonstrating the weight of the URL according to the proposed method for 40000 samples. Most of the URLs are fitted into 0.2 to 0.4.

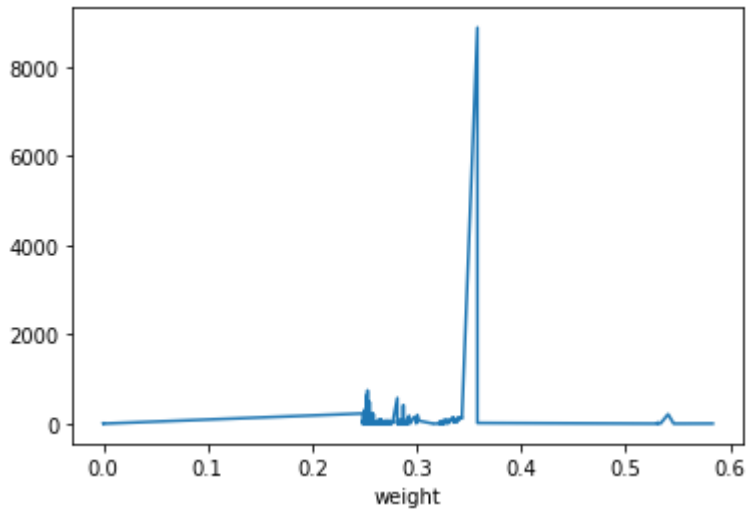

Figure 14 Weight distribution according to the proposed weighting scheme

The other method for detecting outlier from URL features is using textual analysis techniques. The idea is still following the previously proposed method to underline those URLs containing a less frequent word than others. In other words, identify URL which has less common items compared to rest of dataset. One of the standard methods in natural language processing (NLP) is using TF-IDF weight. TF-IDF, instance on Term Frequency, Inverse Document Frequency. Heuristic intuition indicates that if a term occurs many times in a document, it is not a good discriminator between normal and anomalous HTTP transactions, and more weight should be given to words that are less frequent in the data. In general, the TF-IDF formula is divided into two main parts: term frequency, how important the term is in a document, and inverse document frequency is how frequent it is, word over all of the documents [55].

$$w = tf \times idf$$

$$tf_{i,j} = \frac{f_{i,j}}{\sum_{t' \epsilon d} f_{t',d}}$$

$$idf_i = \log\left(\frac{N}{df_i}\right)$$

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Where $w_{i,j}$, is the weight of term i in document j. $tf_{i,j}$, is determining how frequent is the term i in document j. The N represents the total number of documents and df_i, is how many time term i repeated all over the dataset.

In this research, Scikit-learn, Python library is the central part of the implementation. For practical implementation of TF-IDF, the *TfidfTransformer* method has been used. This method is part of the Scikit-learn library [56]. Some tuning and structural changes to the standard textbook notation of TF-IDF have been implemented in the mentioned library. One of the changes is adding one to the IDF equation. It means the new equation is:

$$\mathrm{IDF(t)} = \log [\, \mathrm{N} / \mathrm{df(t)} \,] + 1$$

If the term is repeated in all of the data, then the inverse document value would be zero. Therefore IDF value would totally be ignored. For avoiding this issue, 1 is added to the IDF formula.

Also, to prevent zero divisions, the constant of one is adding to the IDF fraction. It will avoid error if the document contains all of the terms in the dataset. The final equation is,

$$\mathrm{IDF(t)} = \log [\, (1 + \mathrm{N}) / (1 + \mathrm{df(t)}) \,] + 1$$

Moreover, the results are normalized by Cosine Normalization to be bounded into zero and one. It means that sum of squares of vector elements is 1. The vector represents the TFIDF value. The one interprets as data is less frequent; therefore, it has a higher value. On the other hand, zero means the data is pervasive in dataset and transactions.

The TF-IDF score is calculating for each term in each URL for all of the datasets. To build the single feature which is representing the TF-IDF score of URL, the average TF-IDF of record counts as the total value of TF-IDF.

Figure 15 is showing the distribution of TF-IDF scores over 40000 samples. Most of the TF-IDF weights are belong to the 0.3 to 0.7 range. It means that most of the terms are repeated equally over the dataset.

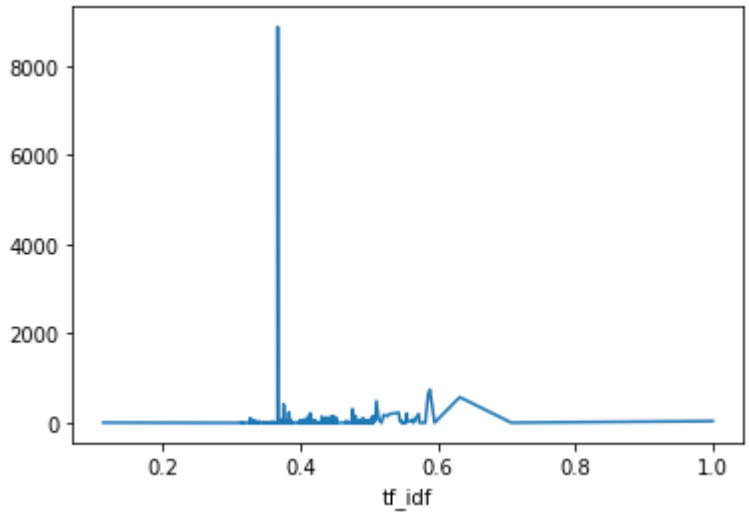

Figure 15 Distribution of TF-IDF weight over 40K samples.

One of the issues related to TF-IDF methods is that it is computationally heavy to perform. It means that the size of features is equal to the size of the term in documents. Therefore, if documents contain an extensive vocabulary, it is costly to calculate each term's weight [56].

3.7 Limit IP range and dividing the data set into sessions

The current architecture is capturing both incoming and outgoing HTTP traffic to the TalTech network. It means that information about all transmitted HTTP packets is included in the dataset. However, interpreting this information is relatively complicated. It covers a wide variety of HTTP transactions with different natures – HTTP transactions initiated by clients in TalTech network and going to external web servers, and HTTP transactions from external clients to TalTech web servers. Since the purpose of this work is not to profile the HTTP clients in the TalTech network but rather detect attacks against TalTech web servers, the current research is only looking at HTTP transactions that are originating from external clients to TalTech webservers. HTTP transactions that are initiated by clients in the TalTech network are excluded. The method is shrinking the size of data to almost half size.

3.7.1 IP selection

The process of IP selection is based on the following rules:

1. Based on the SRC_IP field, exclude all transactions for the IPs belong to TalTech network.

2. Based on the DEST_IP field, exclude all transactions for IPs which are not belonging to TalTech servers.

After selecting desired IPs, the datasets are built again with a limited IP range.

3.7.2 Arranging transactions into sessions

The communication process between servers and clients is not limited to one transaction. For example, when a page is requested from a web server, the page can contain many elements with different URLs, such as JavaScript functions and pictures. Therefore server responds to every URL we are requesting. So, looking at a single transaction between server and client might bring this question, is it enough to narrow the scope to a single transaction or looking at data in a more extensive timeframe. Arranging transactions into sessions helps to find trends over data. It means that to analyze data for specific intervals to find the correlation of observations. The traffic needs to be followed from a specific source or port any related features to find meaningful information. It is also applied to HTTP traffic. For example, the sender can send much traffic over a small time frame. By looking at each of those traffics, we can not conclude any relevant information. However, when counting several requests in a specific timeframe, it would categorize as a DDOS attack.

To arrange sessions, every HTTP session is considered as 5 minutes in the current study. The 5 minutes assumption is the approximate length of each connection for one entire session. It would increase our view over traffic and let to analysis connection based on time frame. Our dataset contains the Timestamp feature, which is letting to follow up at 5 minutes interval. According to more analysis, every 3500 records can be counted as one timeframe. Due to the complexity of converting timestamp fields to meaningful attributes for the programming language, chunking datasets into specific rows is faster.

Finally dataset is divided into 5 minutes (3500 records) to have better understanding of traffic behavior.

```
0       2021-03-12T00:00:00.002614+0200
1       2021-03-12T00:00:00.211255+0200
2       2021-03-12T00:00:00.342562+0200
3       2021-03-12T00:00:00.466720+0200
4       2021-03-12T00:00:00.509843+0200
                      ...
3495    2021-03-12T00:04:58.703022+0200
3496    2021-03-12T00:04:58.761778+0200
3497    2021-03-12T00:04:58.793755+0200
3498    2021-03-12T00:04:58.802591+0200
3499    2021-03-12T00:04:58.806558+0200
```

Figure 16 Sample 3500 records timestamp field

3.8 Finalizing dataset

All of the new features that have been discussed are added to the dataset one by one, as described in previous subsections. Following is the list of all new features that have been derived from basic HTTP header features:

Feature name		Description
uncommon_http_method		Set the feature value to 1 if the HTTP method is uncommon, otherwise set to 0
odd-status-code		Set the feature value to 1 if the HTTP status code is not belonging to common status codes list.
content-length-above-mean		Calculate the mean of the content-length header. Set the feature value to 1 if content-length is greater than the mean, otherwise set to 0
suspicious-user-agent		By using a crafted rule to detect suspicious HTTP, user agents. Set the feature value to 1 if the HTTP user agent is suspicious, otherwise set to 0
HTTP URL features	keywords	Find the list of suspicious keywords. Set the feature value to 1 if the URL contains one (or more) suspicious keywords
	Number of characters	Count the number of characters inside the URL as the length of the URL, and set the feature value accordingly.

	Number of words	Count number of words in URL based on % & = / \ : // \\ ? and set the feature value accordingly
	URL weight score	Calculate less common URL by Algorithm 1, and set the feature value accordingly
	URL TF-IDF score	Calculate each URL TF-IDF total score, and set the feature value accordingly
Limiting IP range		Limiting IP range to all incoming traffic to TalTech servers.
Session arranging		Chunking dataset into 5 minutes intervals.

Table 4 Added features to initial HTTP features

3.9 Labeling and machine learning

After creating the final dataset, the learning phase is stepping in. The learning stage is conducted with machine learning models. Machine learning is an algorithm that gets trained on a training dataset and is evaluated on a test dataset. Machine learning is being fed by dataset as input, and algorithms learn patterns amongst data based on features and characteristics.

The provided dataset does not have any labels. It means that none of the records are defined as malicious or benign. So, there would be arguments on how to train our machine learning model. According to what has been discussed in the literature review chapter, many machine learning models can be used. Each of them represents a specific approach. Like un-supervised learning is mainly associated with clustering machine learning algorithms or supervised algorithms, such a decision tree requires a labeled dataset. Any of approaches have their costs and benefits.

The K-means method has been applied to the dataset to demonstrate the clustering algorithm's weakness in current research. This algorithm is analyzing different data points and tries to fit similar data points into the same cluster. The "K" in K-means comes from several used clusters, and "means" demonstrates finding the nearest mean of each data point to the cluster center [56]. This algorithm is regarded as an unsupervised classification since it not looking at any labels in data. As mentioned above, due to our dataset's nature, not all

machine learning models can handle it. In general, K-means is using the Euclidean distance function to calculate the distance from centers. However, most of our dataset's features are binary. The binary data is discrete and does not allow for using a traditional distance function. The results of the K-means methods are discussed in chapter 4.

As the scope of the current thesis supervised machine learning has been selected. The reason is based on nature of dataset. Some of un-supervised machine learning techniques such as k-mean are not matching the binary characteristics of dataset. Also, the semi-automated labeling process is part of proposed method, therefore it is more efficient to use supervised learning. The other techniques, such as isolation algorithms or intrusion detection algorithms, match current HTTP analysis; however, according to the proposed feature selection, these algorithms are incapable of handling data. They are mostly missing to classify the data.

The tree-based algorithms have been selected for current research. It provides high accuracy predictive model over labeled datasets while it is easy to interpret the information and it provides more human readable representation. It also works well for non-linear information. The provided dataset is mainly based on categorical information, therefore the nature of dataset is closely matching to tree-based methods.

Selected methods are decision tree, random forest, and XGBoost.

The supervised machine learning methods require a labeled dataset. Therefore manual and automated labeling should be done on the dataset. The details around labeling are primarily discussed in chapter 4.

3.9.1 Decision tree

The decision tree is one of the most common decision-making algorithms. It is an upside-down tree in which the internal non-leaf nodes represent a class on the feature, and each leaf node represents a class label.

The decision tree can be used for classification and regression problems. The classification tree, which is primarily applicable to statistics and probability, is based on discrete information, while the regression tree predicts real numbers. There are different types of decision tree algorithms [57].

- ID3: Iterative Dichotomiser 3 is a top-down greedy model. It creates the tree, and at each layer, select the most fitted feature.
- C4.5: is the successor to the ID3 algorithm. It can handle non-categorical data. At each level, it selects the feature which split the samples into new enrich subsets.
- CART: Classification and Regression Tree is very similar to C4.5; however, it supports the numerical parameters. The Scikit-learn library is using the CART approach.

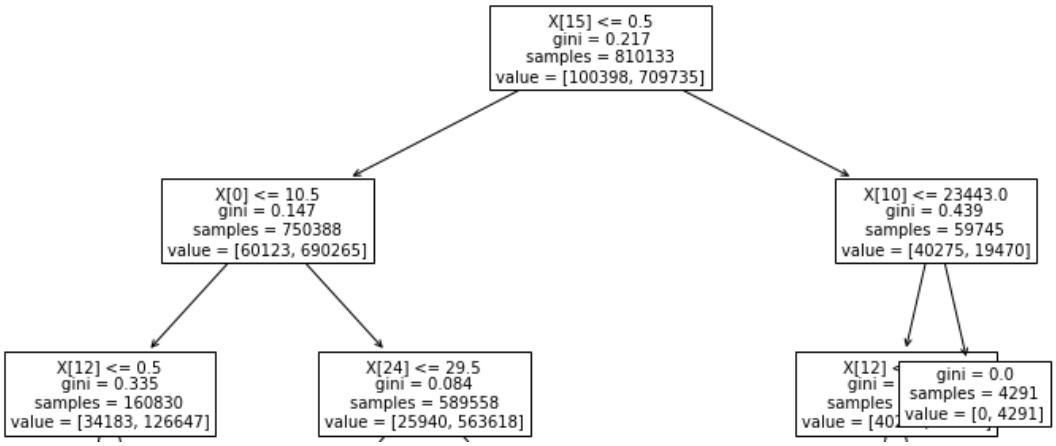

Figure 17 Sample decision tree using CART algorithm visual representation

To ensure the samples are finely selected, there are two main impurity measurements for decision tree samples. Entropy and Gini index. Entropy determines disorder in samples [58]. Entropy is defined as below:

$$Entropy(S) = \sum_{i=1}^{c} p_i \times \log_2 p_i$$

Where p_i is frequentist probability of class i.

The Gini index measure inequality in samples. If the sample is entirely homogeneous and all of the features are the same close class, then the Gini index is 0.

$$Gini - index = 1 - \sum_{i=1}^{n} p_i^2$$

Where p_i is frequentist probability of class i.

The CART technique is using the Gini index as a cost function to split features selection.

3.9.2 Random forest

The random forest algorithm is another type of decision tree-based model. It builds multiple decision trees and aggregates those trees for achieving better results. In other words, it builds an ensemble of decision trees using the bagging method to increase prediction results. Also, it creates trees based on selected features to maximize randomness. The random forest algorithm is computationally efficient [58]. Following characteristics can be named for this method:

- It supports both regression and classification data.
- Relatively faster learning and prediction phases.
- Few optimization hyper-parameters.
- Control over-fitting error.

The random forest algorithm is described below:

Algorithm 2: Random Forests

Input	:	Dataset D
Output	:	Prediction of new point
Step 1	:	Let D = $\{(x_1, y_1), \dots, (x_N, y_N)\}$ as training dataset and $x_i = (x_{i,1}, \dots, x_{i,p})^T$ Where p is predictors.
Step 2	:	Take sample S from D with size N
Step 3	:	Using S as training data and fit tree using binary recursive partitioning.
Step 4	:	a. Start all samples in a single node b. Repeat for each unsplit node until: i. Randomly select m predictors from all predictors. ii. Find best split among all splits.

Step 5 : Prediction of new point x in classification:

$$f(x) = max_y \sum_{j=1}^{J} (h_j(x) = y)$$

Where $h_j(x)$ is the prediction of response variable at x using the *j*th tree.

The random forest can be validated with an Out Of Bag (OOB) score. Out of bag is measuring the samples that did not belong to the bag while forming subset samples. It means that the tree is missing some samples. Out of bag is calculating as follow:

$$Error_{oob} = \frac{1}{N} \sum_{i=1}^{N} (y_i \neq f_{oob}(x_i))$$

Where $f_{oob}(x_i)$ is out of bag prediction for sample *i*.

3.9.3 XGBoost

The XGBoost is a machine learning algorithm based on the decision tree method. The University of Washington developed the algorithm in 2016 [58]. XGBoost is hiring effective characteristics of other methods. It is mainly based on a gradient boosting framework which is a gradient descent algorithm that minimizes errors. The XGBoost algorithm improved current frameworks by system optimization. The system optimization is including the following aspects:

1. Parallelization: The algorithm can build a sequential tree by using parallelized implementation. This method is increasing performance.
2. Tree pruning: XGBoost uses the maximum depth hyper-parameter and down to top pruning method for improving performance.
3. Hardware optimization: The algorithm is developed to allocate an internal buffer for each thread for gradient calculation. Also, for improving disk utilization, out-of-core computation using block compression and block sharding has been used.

The XGBoost algorithm is offering many optimization hyer-parameters. Tuning hyper-parameters can improve results. The hyper-parameters are mainly for tree booster and linear booster models.

3.10 Prediction and validation

The last step in the proposed methodology is prediction. After training the machine with the provided dataset, the built model can predict the unseen situation. In the prediction phase, the test data can be selected from the available dataset by removing the label or selecting unseen data. The machine learning model is assessing the test data based on its model.

The machine learning techniques have their measurement parameters. These parameters represent the correctness of the decision made by the machine learning model. There are many different parameters available; however, four are common among most machine learning techniques. These parameters are precision, recall, F-measure, and accuracy.

Precision is measuring ratio of true positive to the sum of true positive and false positive.

$$Precision = \frac{True\ Positive\ Rate}{True\ Positive\ Rate + False\ Positive\ Rate}$$

The recall is demonstrating ratio of true positive to the sum of true positive and false negative.

$$Recall = \frac{True\ Positive\ Rate}{True\ Positve\ Rate + False\ Negative\ Rate}$$

F-measure or F-score is a harmonic ratio of precision and recall. This measurement says how accurate the classification is.

$$F - Measure = \frac{2Precision * Recall}{Precission + Recall}$$

Accuracy is the ratio of the total number of correct predictions over all of the predictions.

$$Accuracy = \frac{True\ Positve\ Rate + True\ Negative\ Rate}{True\ Positive\ +\ False\ positive + True\ Negative\ + False\ Negative}$$

Validation is an essential part of machine learning techniques. Generally, validation means the results can describe by hypothesis and original data. The proposed validation stage is divided into two main parts. The Cross-validation and external source validation.

Hold-out is a trivial and most common technique. It divides data into two big chunks for testing and training. Most of the time, testing is 20%, and training is 80% of the dataset. The training dataset trains the machine, and the test is done by the test dataset. If the dataset is not entirely distributed, the random selection might cause training and test datasets to become very similar.

Cross-validation is splitting data once or several times to measure the performance of the algorithm. Cross-validation is avoiding the overfitting issue [52]. There are some cross-validation methods; however, only two of them have been used in this research. Hold-out and K-fold techniques.

The second technique is K-Fold. This method is adding randomness to data selection and shuffles the train and test for optimizing results [52]. The method procedure is explained below:

1. Select random number of K. K can be maximum the length of the dataset.
2. Split dataset into K equal chunks.
3. Pick one of the chunks as training and the rest as a test dataset.
4. Repeat selecting chunks K times until all of the chunks are used as a test one time.
5. Calculate the average score of K times repeats as the final result.

K-fold is more reliable and stable than Hold-out since different parts of the dataset have been used for training and testing. The selection of K should be in a way to not put an extra computational cost on the system.

For adding extra value to current research, external validation is also considered. It means that the IP addresses which are identified as belonging to attackers are compared to internet sources for extra validation (for example, AbuseIPDB database). Furthermore, the framework feedbacks can be used as firewall rules.

After daily records are fed into the learning model and malicious records are identified, The IP field will be checked with external databases such as AbuseIPDB for validation. If the IP has malicious activity reports, the system correctly identified the input dataset's record.

The results, performance metrics, and implementation are discussed in chapter 4.

3.11 Summary

This chapter was dedicated to the proposed structure and system. The desired system is receiving HTTP data from Suricata sensors. The multi-layer feature extraction is proposed to identify the most valuable features from HTTP traffic information. The framework is looking for extraordinary characteristics in each record and identifying odd behaviors compared to the rest of the dataset. Some of the extracted features are directly concluded from native provided features such as HTTP status code; however, some of them, like HTTP user agent, rely on other features. Extensive analysis on URLs provides extra value for current research.

The native HTTP features provided by Suricata are not providing diverse characteristics to build the efficient detection model. Building the new features which are directly extracted from native features, the wider range of features for each transaction can be defined. It helps to improve the learning process by deeper analysis for each transaction. Combining all native and newly added features provides stronger dataset for learning algorithms.

After forming a new dataset based on extracted features, the decision-tree-based machine learning techniques are hired to classify data. Each algorithm is tuned to find the most optimal accuracy in detection. Lastly, validating methods and performance measurements are conduct on machine learning output.

4 Framework implementation, results

4.1 Chapter overview

The following chapter is discussing the implementation of proposed framework and results. The different hypotheses are measured to compare best fitting algorithm for current research. The labeling method is discussed in detail.

4.2 Data labeling

One of the essential phases in the current study is having a structured dataset. The structured dataset is data that adheres to the pre-defined data model. It is mostly in tabular format with the relationship between rows and columns. The collected data are entirely unlabeled, and it is not fit supervised learning, which requires a labeled dataset. This issue has been solved by labeling the dataset by the author.

The labeling process is happening in two phases. The first phase is automated labeling, while the second phase involves the review and confirming the labeling. Automated labeling is using some predefined rules to label the dataset. After that, the labels are reviewed by a specialist to confirm or reject the labels. The process is quite time-consuming and requires a lot of manual work to prepare a fully labeled dataset. In most cases, automated labeling marked the malicious transaction correctly. Without this approach analyzing every transaction requires checking the IPs, method, and URL to conclude the transaction is legit or malicious.

As mentioned above, automated labeling is based on predefined rules. Using predefined rules, brings this question that is it valid to train machine by data which is labeled by another rule? If automated labeling is enough, there is no need for an extra machine learning layer for prediction. The answer is that by automated labeling, there will be many records that have been mislabeled. It means that some malicious records are identified as normal and vice versa. The rule for initial automated labeling is as described in algorithm 3:

Algorithm 3: Automated labeling

Input : Dataset D

Output : Labeled dataset D

Step 1 : // Labeling the dataset D with m rows.

For all m in D **do**

If [(uncommon-http-method == 1) AND ((odd-status-code == 1 OR suspicious-user-agent == 1) AND URL-keywords(Compressed file OR Interpreters OR Connection tool OR Others))], **then**

$label_m$ = malicious

else

$label_m$ = normal

End if

End for

By applying the aforementioned rule, most malicious records are correctly labeled. However, the label is narrow, and there would be other records with malicious activities that are not fitting to the above rule. Therefore a specialist reviewed the dataset to check if the data are correctly labeled or not. Both normal and malicious records are checked. The author is specialist who confirm the labeling process. The knowledge is based on more than one year of Tallinn University of Technology SOC team's information collection about ongoing attacks on university servers by 5 specialists.

This process was taking place on a specific timeframe (two working days in university) to limit the required time and resources for labeling. It means that not all of available datasets are labeled and only some of them contain label.

4.3 Results

In the previous chapter, different machine learning models are introduced. In this research, supervised machine learning techniques have been selected since they fit the collected

dataset. The other methods, such as some of the clustering algorithms, could not provide meaningful information for detecting malicious activities. However, it should not be interpreted that all other learning methods are invalid for such research. The domain of the current thesis is focusing on supervised learning methods.

A simple analysis of the K-means method has been conducted to demonstrate one of the clustering machine learning models' incapability on the dataset.

The K-means algorithm forms K clusters and uses the Euclidean distance function to calculate the distance from centers. For calculating the optimal number of clusters or namely K, there are different methods. The Elbow method runs clustering for K's range of values and computes all clusters' average scores. It uses the sum of squared distances to determine optimal K. Figure 18 demonstrates the Elbow algorithm over our dataset. From K=7, the diagram slope is decreasing, and 7 is the optimal selection of clusters.

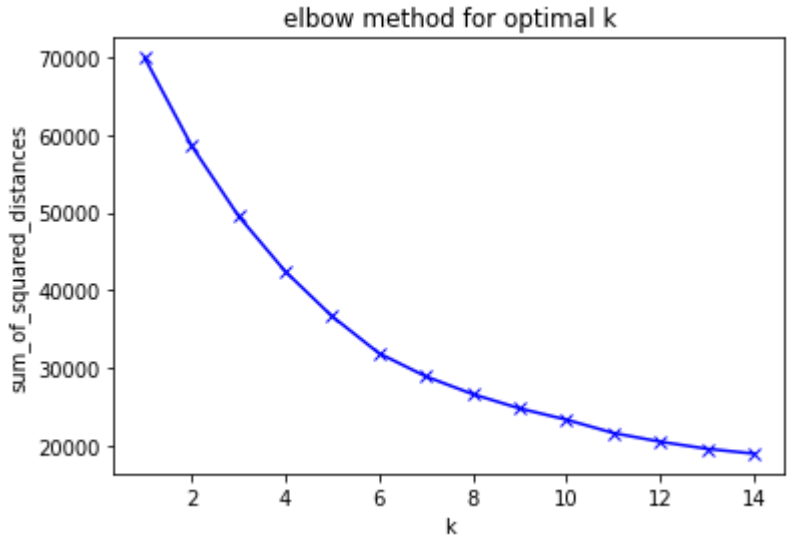

Figure 18 Elbow method for finding optimal K

After selecting optimal K, data are transformed and fit into K-means algorithms. The algorithm forms clusters centers based on input data. One of the issues of applying the K-means algorithm on our data is that most of our dataset is binary, and K-means use the mean function to calculate distances. Therefore it would cause the algorithm not to work correctly. The second reason is to divide data into two main categories of malicious and benign, requiring K values to be 2. However, as elbow calculation shows, the optimal K for the current dataset is around 7. It means that if data are only separated into two significant

clusters, many data are misclassified and would not represent correct separation. Nevertheless, it is worth mentioning that clustering algorithms do not need a labeling process. Therefore they are very efficient if dataset characteristics match it.

Besides unsupervised machine learning models, in supervised techniques, the selected methods were decision tree, random forest, and XGBoost. All of them are tree-based techniques. These methods require labeled datasets as input to start building a machine learning model.

4.3.1 Decision tree's results

The decision tree technique is forming leaves and branches to classify data. By default, the tree can grow until every feature is classified. For the current study, the Scikit-learn version of the decision tree has been used. This version is using the CART method. The input data needs to be split into two training and testing chunks. The training part is using to train the machine, and testing is using for validating the model. As mentioned before, there are two main techniques to split data. The hold-out and K-fold. In the normal splitting, the data are divided into random subsets. Following is settings for the "train_test_split" function in Scikit-learn:

```
train_test_split(X, y, test_size=0.2, random_state=20, shuffle=True)
```

Where 20% of data belongs to the test and 80% is used to train the model. The data are randomly shuffled to maximize randomness in subset selection.

Our dataset is unbalanced. It means that the proportion of data is 12% data are attacks, and the rest are normal. Therefore it is needed to calculate each label's performance metrics separately to keep the importance of metrics equally. The macro average and weighted average describe the model's effectiveness regardless of each sample's proportion in the dataset.

Running the decision tree classifier algorithm with default settings is not optimal. By default, the maximum depths of the tree are infinite. It means trees grow to classify every feature. The following results are describing model performance metrics.

Label	Precision	Recall	F1-score	support	Macro average F1-score	Weighted average F1-score	Accuracy
Attack	0.86	0.87	0.87	25228	0.92	0.97	0.97
Normal	0.98	0.98	0.98	177306			

Table 5 Decision tree performance metrics on default settings

For optimizing tree setting, the maximum depth hyper-parameter is considered. The performance metrics are measured in different depths for finding the most optimal tree depth. Following figure 19 is demonstrating Precision and Recall rate for attack data on different depth.

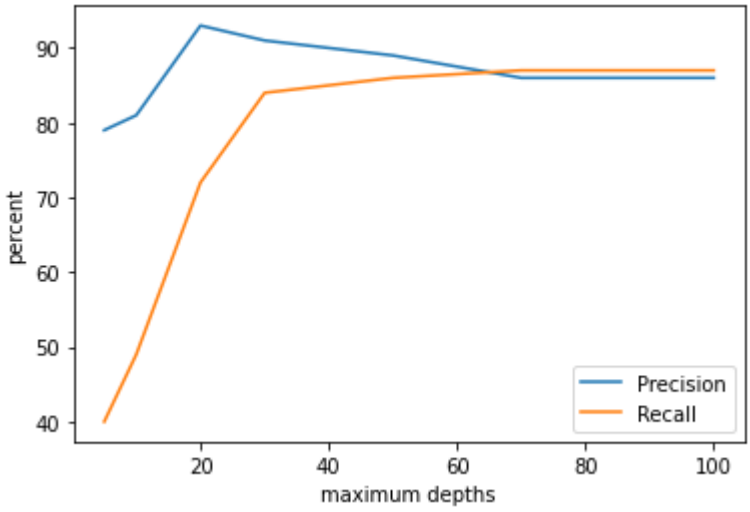

Figure 19 Performance metrics based on tree depth for attack data

Based on Figure 19, the tree has the most optimal hyper-parameters in the depth of 30; therefore, the tree's maximum depth is allowed to grow to 30.

The following results are describing decision tree performance metrics in the depth of 30.

Label	Precision	Recall	F1-score	support	Macro average F1-score	Weighted average F1-score	Accuracy
Attack	0.91	0.84	0.87	25228	0.93	0.97	0.97
Normal	0.98	0.99	0.98	177306			

Table 6 Decision tree performance metrics on non-default settings

Another method is using K-fold cross-validation techniques. To briefly remind the method, in cross-validation, the test and training data keep changing to find the optimal results and avoid overfitting issues. The dataset is dividing into random subsets. Then decision tree algorithm runs over each of the subsets, and the best accuracy would be selected. In this research, the dataset is divided into ten folds. The following figure is showing the accuracy rate per iteration.

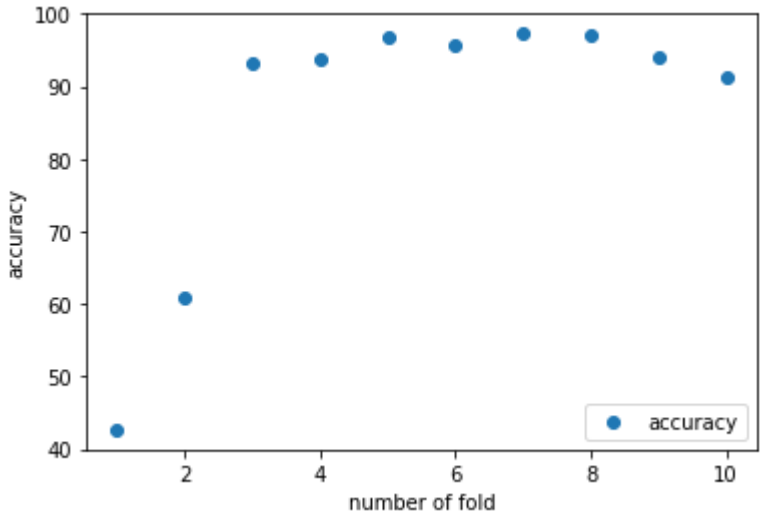

Figure 20 Accuracy of method on 10 K-fold cross-validation

The average accuracy for the model is 96.78% percent, and it is happening at a maximum depth of 29.

4.3.2 Random forest's results

The second algorithm is random forests. To briefly remind the method, the random forests algorithm is based on the decision tree model. It builds an ensemble of decision trees and computes the results for each tree. Furthermore, it uses majority voting to classify data. Besides standard performance metrics, random forest is measured by out-of-bag score as well. The out-of-bag score indicates the error rate of random forest. It means how many samples are never selected in any subset, or in other words; they are out of selected bags for classification.

The Scikit-learn implementation of the random forest has been used in this research. The algorithm is running by default settings. In general settings, the algorithm uses the whole dataset to train each tree. Also, 100 trees would grow in the forest. Like a decision tree, the

maximum depth for a tree is infinite, which means the algorithm runs until all of the leaves are classified.

The data split into two trains and test dataset by using the same "train_test_split" function in Scikit-learn:

```
train_test_split(X, y, test_size=0.2, random_state=20, shuffle=True)
```

Where 20% of data belongs to the test and 80% is used to train the model. The data are randomly shuffled to maximize randomness in subset selection.

The following table is demonstrating the random forest algorithm's results on default settings.

Label	Precision	Recall	F1-score	support	Macro average F1-score	Weighted average F1-score	Accuracy	Out-of-bag score
Attack	0.88	0.84	0.86	25228	0.9	0.96	0.97	0.966
Normal	0.98	0.98	0.98	177306				

Table 7 Random forest performance metrics on default settings

Multiple hyper-parameters can improve random forest algorithms. One of the tuning hyper-parameters is the number of estimators, grows in the number of trees. Figure 21 shows by increasing the tree numbers, and the results are similar after some level.

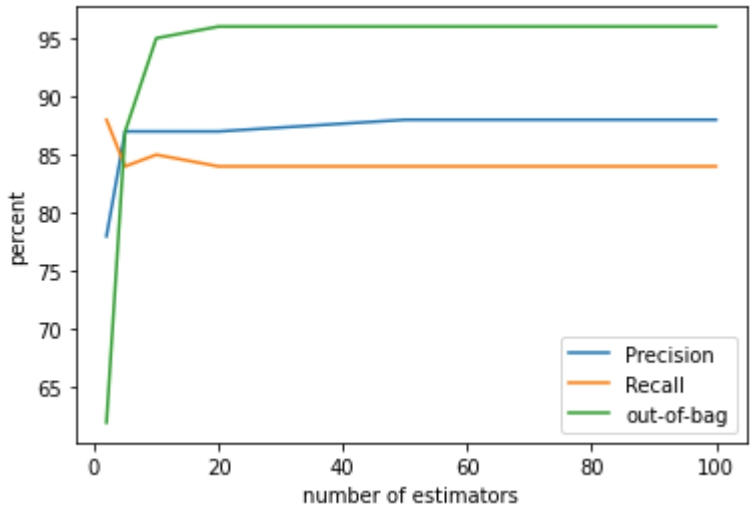

Figure 21 Performance metrics correlation with the number of estimators in random forest algorithm in Attack samples

The other tuning hyper-parameter is the maximum allowed depths for each tree to grow. Based on the results in Figure 21, the optimal number of estimators is around 10. Figure 22 is showing different maximum depth effects with ten estimators. According to graph 50, is the optimal depth that a tree can grow.

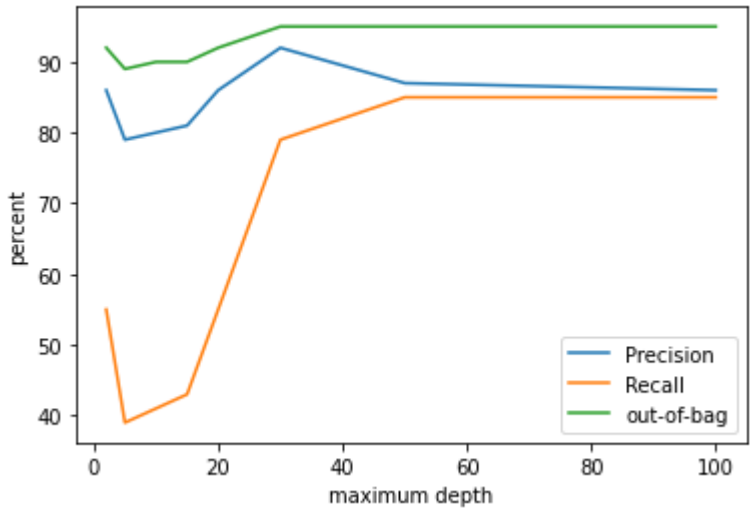

Figure 22 Performance metrics correlation with increasing of maximum depths in attack data

Another method for sub-setting data is using K-fold cross-validation. The data divided into ten subsets and run random forest algorithm ten times with different train and test sets. Following figure 23 is showing the algorithm accuracy results based on different training and test sets.

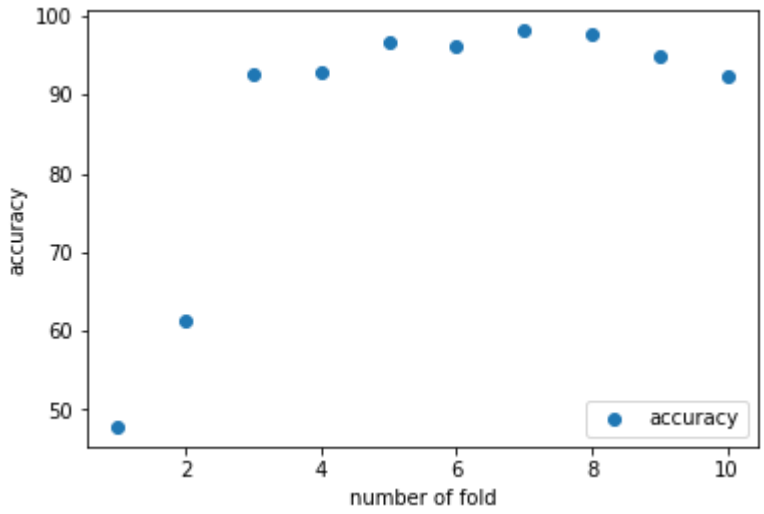

Figure 23 Accuracy of method on 10 K-fold cross-validations for random forest

4.3.3 XGBoost's results

The third machine learning method is XGBoost. To briefly remind the method, the XGBoost is based on the ensemble of different tree-based algorithms. The algorithm is trying to cover most over-fitting issues. The XGBoost algorithm contains many tuning hyper-parameters. Most of the settings from tree-based and gradient boosting algorithms are exist in the XGBoost library. For implementation, Scikit-learn API for XGBoost random forest classifier has been used. The data split into two trains and test dataset by using the same "train_test_split" function in Scikit-learn:

```
train_test_split(X, y, test_size=0.2, random_state=20, shuffle=True)
```

Where 20% of data belongs to the test and 80% is used to train the model. The data are randomly shuffled to maximize randomness in subset selection.

To run the algorithm following hyper-parameters have been selected. The list is added to the rest of the default hyper-parameters.

Hyper-parameter	Value	Description
Colsample_bytree	0.3	Subsample ration of column for each tree.
Learning_rate	2	Boosting learning rate
Max_depth	40	Maximum tree depth for each learner
Reg_alpha	10	Regulation term on weight
N_estimators	20	Number of tree in random forest
Num_parallel_tree	100	Number of the parallel tree for boosting algorithm

Table 8 XGBoost tuning hyper-parameters

After running the XGBoost algorithm, the following table describes the performance metrics for the algorithm.

Label	Precision	Recall	F1-score	support	Accuracy	Macro average F1-score	Weighted average F1-score
Attack	0.87	0.87	0.87	25228	0.97	0.92	0.97
Normal	0.98	0.98	0.98	177306			

Table 9 XGBoost performance metrics on non-default settings

4.4 Discussion

In this chapter, the implementation details and results of machine learning models are discussed. The labeling process for supervised learning has been detailed out. The decision tree, random forest, and XGBoost are the three machine learning models that have been used in this thesis. Each algorithm's performance metrics are measured. Also, the cross-validation method has been used to find the most optimal results. The discussion on results is provided in chapter 5.

5 Discussions and future works

5.1 Chapter overview

The following chapter is discussing results from machine learning models. The results are compared based on different metrics. Also, external validation results and firewall rules are discussed in this chapter. Moreover, future research for improving the framework is described.

5.2 Discussion on machine learning algorithm's results

So far, the dataset formation, features extraction, and labeling dataset has been done. In chapter four, the results for three selected machine learning models are demonstrated. The question is, which model has better accuracy and less training and testing time? For answering the question, a comparison of results is needed.

Firstly, the performance metrics are compared. Each algorithm runs on some tuning hyper-parameters to optimize the results. The same dataset has been used for each model to have proper grounding for comparison. Figure 24 compares the precision, recall, and F1-score results for the decision tree, random forest, and XGBoost algorithms on the attack sample. Figure 25 is comparing the same information on the benign dataset.

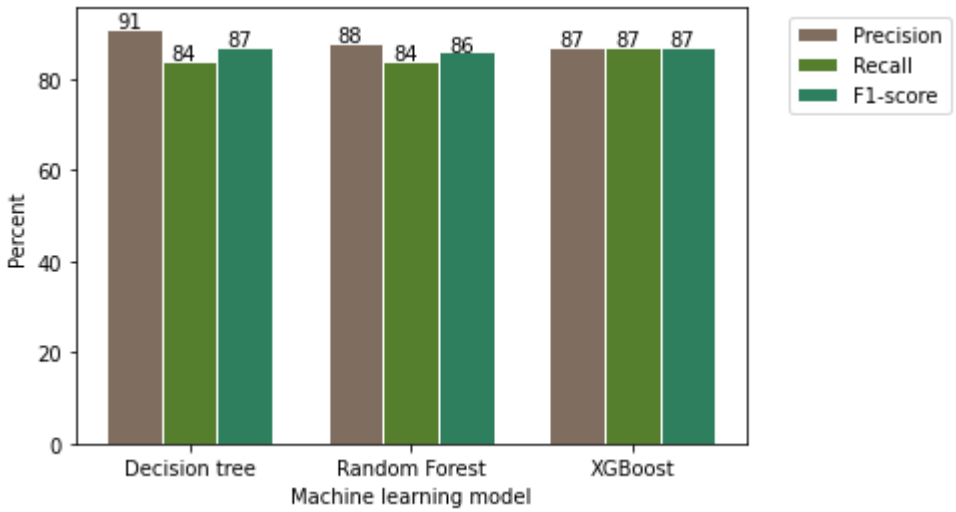

Figure 24 Comparison of machine learning algorithms' performance metrics on attack data

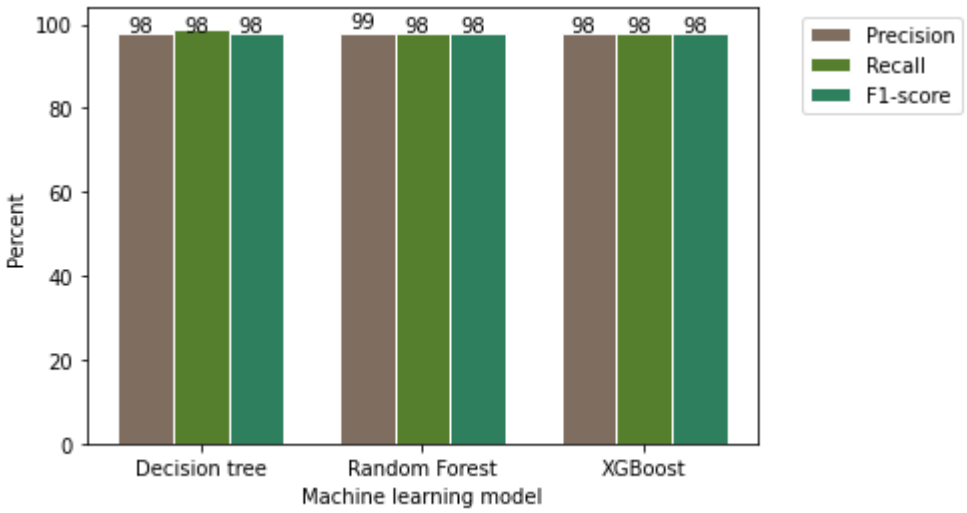

Figure 25 Comparison of machine learning algorithms' performance metrics on benign data

Among all of the methods, the decision tree has more promising results with the highest classification of attack samples. However, both the XGBoost and random forests also have relatively high detection rates. The significant difference in the algorithms appears in CPU time measurements.

The second parameter for measurements is CPU time. The CPU time is showing how long the training stage has lasted. Time is the critical factor for the measurement of the real-time capability of the algorithm. If the learning time is extended, therefore, the model does not apply to run live. Figure 26 is demonstrating the CPU time for each algorithm.

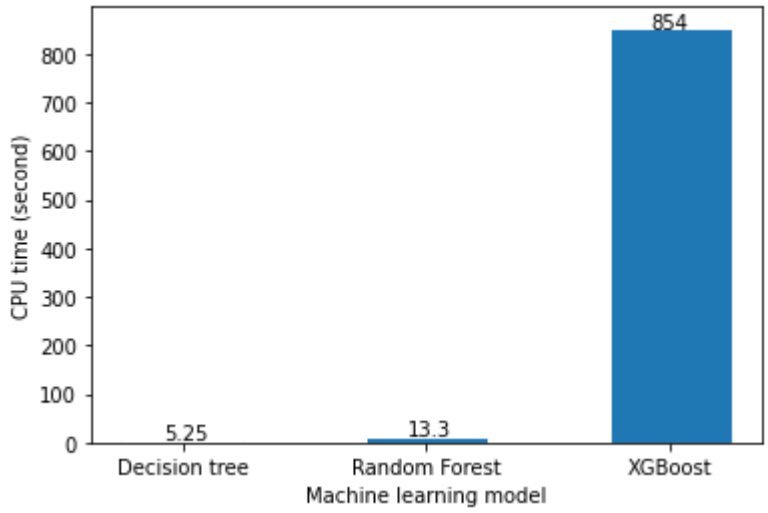

Figure 26 CPU time comparison for training phase of machine learning algorithms

As figure clearly showing the XGBoost involves significant CPU time consumption when compared to two other algorithms. It takes 170 times longer to run XGBoost compared to the decision tree. Also, the decision tree and random forest have close CPU time.

When using cross-validation, the training time is increasing. Figure 27 is showing the decision tree and random forest CPU time using the K-fold cross-validation technique.

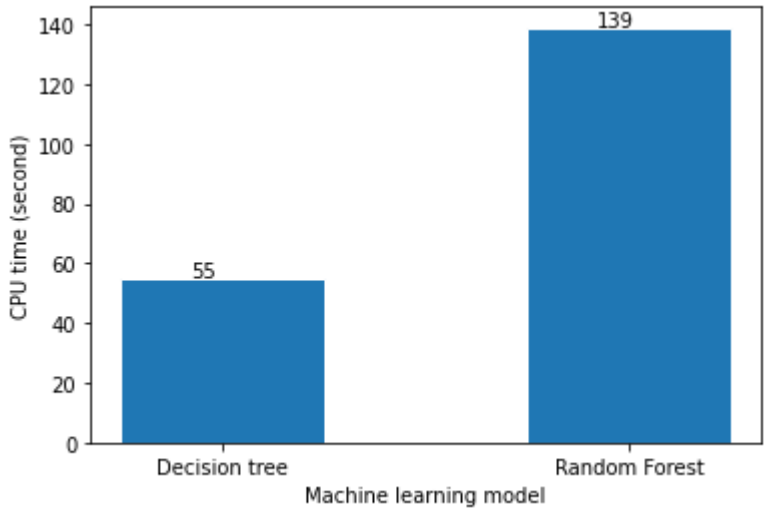

Figure 27 CPU time comparison for decision tree and random forest algorithms in K-fold validation

The CPU time determines the applicability of the method in real-time. According to the above graphs, the decision tree consumes the least amount of CPU time. However, the random forest consumes a moderate amount of CPU time as well. It is important to note that the reported CPU times belong to current hardware and can be different on other servers.

Not every machine learning algorithms can predict with fully assurance. In current studies some of the attacks are misclassified by learning model. To understand, what types of attacks are missed from proposed framework, the results have been analyzed. Most of attacks that are not detected, interact with server in a way that are against the expected trained behavior. It means they contains characteristics such as common verbs in URL, legit user-agents and 200 status code. These characteristics are tricking the model by providing the features which are classified as normal traffic before. For avoiding such problems, more extensive feature extraction is required. It helps the model to meet more conditions before final classification happens.

5.3 External validation

The current research aims to build an application to ease detecting malicious activity for the security operation team. It means that the application should run at a specific time and report malicious IPs to analyzers. The following proposal has not been implemented during this research.

Due to the machine learning process's nature, there is no complete confidence that the final reported results are entirely correct. Therefore adding extra checks to provide a list of IPs is adding value to the result. The IPs are checked with the AbuseIP database. The AbuseIP database is a public database for malicious identified IPs. The IPs in that database have a confidence level. If IP is reported many times as malicious IP, then the confidence level is high. The application can check IPs from machine learning with AbuseIP, and if the IP is reported, it has a significant chance to be blocked as malicious IP.

The blocking procedure is varies based on company security policy and technological architecture. However, the suggested system is based on IP pools. If IP is identified as malicious IP, then IP is added to the suspicious pool with an expiration time of 24 hours. The firewall is blocking suspicious list IPs for one day. If the IP is not repeating the other days, it would automatically be removed from the list and unblocked. However, if the IP exists more than 48 hours in the suspicious list, then IP adds to a new list called the blacklist. The firewall is blocking the blacklist forever.

5.4 Future works

In this thesis, the proposed framework is using selective feature extraction and several machine learning approaches. However, some topics still are missing from this research.

5.4.1 More extensive feature extraction

For this research, several features have been extracted and introduced to make the dataset richer and add value. However, in URL and user-agent HTTP fields, more research is required.

The HTTP user-agent can be very different based on the client browser, operating system, and hardware. Therefore finding the correct pattern to extract malicious user-agent is helpful.

The HTTP URL field contains a lot of information. In the current research, some textual and statistical methods are implemented to find more meaningful features. The analyzing process can extend to find unusual activity by using natural language processing techniques.

5.4.2 Unsupervised machine learning

The current research is using supervised learning as the primary technique. However, using supervised learning needs a labeled dataset, and it is a costly task. By using unsupervised methods such as clustering or density-based algorithms, pre-processing time can be reduced.

6 Conclusion

The intrusion detection systems are the main component of any network architecture these days. These software or hardware are required to run as fast as possible to keep up with rapidly increasing network traffic size. Also, attacks and their signature are constantly changing. Therefore, intrusion detection systems need to update their detection system to avoid missing the zero-days attack.

This research is proposing a behavioral network intrusion detection system on HTTP data by using machine learning techniques. The framework is detecting malicious activity on HTTP protocol from Suricata sensors' data. These sensors are installed on Tallinn University of Technology network perimeters. The proposed research is running on tree-based supervised machine learning techniques to maximize performance and accuracy.

The proposed system aims to detect malicious activities by analyzing attacks' patterns and behavior instead of relying on attack signatures.

In this thesis, extensive HTTP feature extraction has been used. Besides, native features collected by Suricata, multi-level feature extraction, have been used to collect new features from different parts of the HTTP header. The new features are describing the behaviors and characteristics of each transaction. Also, for having better tracking of the HTTP sessions, session arranging has been used.

The supervised tree-based machine learning algorithms have been selected for the current research's prediction phase. The decision tree, random forest, and XGBoost are selected algorithms. Each method is optimized to have maximum performance. The cross-validation techniques are also have been used to ensure the results are not over-fitted.

The proposed system's results are promising. With a one-day timeframe collected information, which is almost 1 million records, the system can detect malicious activities with almost 90% confidence depending on algorithms. Besides, the performance metrics from each algorithm is showing that the decision tree has the best overall score compared with other techniques with 91% precision. In addition to classic performance metrics, each algorithm's running time shows that the decision tree is the fastest training and detection speed.

On the other hand, the XGBoost is 170 times slower than the decision tree or random forest. The CPU time is essential for running the algorithm in a live production server. The extra validation with external databased has been proposed for increasing the assurance of algorithm outputs.

All in all, this thesis is contributing to security operation teams by automating the attack detection process. The proposed application can act as added incident detection and alerting. The proposed model still needs improvements for identifying more unseen situations.

7 References

[1] A. Ghasempour, Z. Mohd.hanapi, M. Salehi, and Z. Vahdati, "Using traffic control scheme in intelligent transportation system," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 1.4 S1, pp. 165–172, 2019.

[2] M. Park, J. Han, H. Oh, and K. Lee, "Threat Assessment for Android Environment with Connectivity to IoT Devices from the Perspective of Situational Awareness," *Wirel. Commun. Mob. Comput.*, vol. 2019, pp. 1–14, Apr. 2019.

[3] A. Patel, M. Taghavi, K. Bakhtiyari, and J. Celestino Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review," *Journal of Network and Computer Applications*, vol. 36, no. 1. pp. 25–41, Jan-2013.

[4] M. A. Aydin, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Comput. Electr. Eng.*, vol. 35, no. 3, pp. 517–526, May 2009.

[5] "Intrusion Detection and Prevention System Management | IBM," *IBM*. [Online]. Available: https://www.ibm.com/security/services/intrusion-detection-and-prevention-system-management. [Accessed: 23-Mar-2021].

[6] "FortiGate Intrusion Prevention System (IPS)," *Fortinet*. [Online]. Available: https://www.fortinet.com/products/ips. [Accessed: 23-Mar-2021].

[7] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to Snort system," *Futur. Gener. Comput. Syst.*, vol. 80, pp. 157–170, Mar. 2018.

[8] M. V. Pawar and J. Anuradha, "Network security and types of attacks in network," in *Procedia Computer Science*, 2015, vol. 48, no. C, pp. 503–506.

[9] Aaron, "Snort vs Suricata," *Tactical Flex, Inc.*, 2019. [Online]. Available: https://tacticalflex.zendesk.com/hc/en-us/articles/360010678893-Snort-vs-Suricata. [Accessed: 04-Oct-2020].

[10] H. Debar and J. Viinikka, "Intrusion Detection: Introduction to Intrusion Detection and Security Information Management," Springer, Berlin, Heidelberg, 2005, pp. 207–236.

[11] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1–2, pp. 18–28, Feb. 2009.

[12] R. Samrin and D. Vasumathi, "Review on anomaly based network intrusion detection system," in *International Conference on Electrical, Electronics, Communication Computer Technologies and Optimization Techniques, ICEECCOT 2017*, 2018, vol. 2018-January, pp. 141–147.

[13] V. Nagadevara, "Evaluation of Intrusion Detection Systems under Denial of Service Attack in virtual Environment Venkatesh Nagadevara," Blekinge Institute of Technology, Karlskrona, 2017.

[14] W. Park and S. Ahn, "Performance Comparison and Detection Analysis in Snort and Suricata Environment," *Wirel. Pers. Commun.*, vol. 94, no. 2, pp. 241–252, May 2017.

[15] "All features | Suricata," *Suricata*. [Online]. Available: https://suricata-ids.org/features/all-features/. [Accessed: 23-Mar-2021].

[16] R. Prasad and V. Rohokale, "Artificial Intelligence and Machine Learning in Cyber Security," Springer, Cham, 2020, pp. 231–247.

[17] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, p. e4150, Jan. 2021.

[18] G. Karatas, O. Demir, and O. K. Sahingoz, "Deep Learning in Intrusion Detection Systems," in *International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism, IBIGDELFT 2018 - Proceedings*, 2019, pp. 113–116.

[19] S. Zavrak and M. Iskefiyeli, "Anomaly-Based Intrusion Detection from Network Flow Features Using Variational Autoencoder," *IEEE Access*, vol. 8, pp. 108346–108358, 2020.

[20] G. A. Jaafar, S. M. Abdullah, and S. Ismail, "Review of Recent Detection Methods for HTTP DDoS Attack," *Journal of Computer Networks and Communications*, vol. 2019. Hindawi Limited, 2019.

[21] "KDD Cup 1999 Data." [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed: 05-Nov-2020].

[22] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009.

[23] N. Paulauskas and J. Auskalnis, "Analysis of data pre-processing influence on intrusion detection using NSL-KDD dataset," in *2017 Open Conference of Electrical, Electronic and Information Sciences, eStream 2017 - Proceedings of the Conference*, 2017.

[24] G. Meena and R. R. Choudhary, "A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA," in *2017 International Conference on Computer, Communications and Electronics, COMPTELIX 2017*, 2017, pp. 553–558.

[25] "NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB." [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html. [Accessed: 05-Nov-2020].

[26] M. Belouch, S. El Hadaj, and M. Idlianmiad, "Performance evaluation of intrusion detection based on machine learning using apache spark," in *Procedia Computer Science*, 2018, vol. 127, pp. 1–6.

[27] A. Pasumpon pandian and D. Smys, "DDOS ATTACK DETECTION IN TELECOMMUNICATION NETWORK USING MACHINE LEARNING Secured Self Organizing Network Architecture in Wireless Personal Networks View project DDOS ATTACK DETECTION IN TELECOMMUNICATION NETWORK USING MACHINE LEARNING," *J. Ubiquitous Comput. Commun. Technol.*, vol. 01, no. 01, pp. 33–44, 2019.

[28] ENISA, "ENISA Threat Landscape 2020 - Botnet — ENISA," Oct. 2020.

[29] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of Botnet DDoS attack detection using machine learning," *Evol. Intell.*, vol. 13, no. 2, pp. 283–294, Jun. 2020.

[30] M. Aamir and S. M. A. Zaidi, "Clustering based semi-supervised machine learning for DDoS

attack classification," *J. King Saud Univ. - Comput. Inf. Sci.*, Feb. 2019.

[31] M. Belshe, BitGo, R. Peon, I. Google, E. M. Thomson, and Mozilla, "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)," May-2015.

[32] I. Sreeram and V. P. K. Vuppala, "HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm," *Appl. Comput. Informatics*, vol. 15, no. 1, pp. 59–66, Jan. 2019.

[33] L. Masinter, T. Berners-Lee, and R. T. Fielding, "Uniform Resource Identifier (URI): Generic Syntax," *Netw. Work. Gr.* , 2005.

[34] Y. Yu, H. Yan, H. Guan, and H. Zhou, "DeepHTTP: Semantics-Structure Model with Attention for Anomalous HTTP Traffic Detection and Pattern Mining," Oct. 2018.

[35] K. Li, R. Chen, L. Gu, C. Liu, and J. Yin, "A method based on statistical characteristics for detection malware requests in network traffic," in *Proceedings - 2018 IEEE 3rd International Conference on Data Science in Cyberspace, DSC 2018*, 2018, pp. 527–532.

[36] P. Sornsuwit and S. Jaiyen, "A New Hybrid Machine Learning for Cybersecurity Threat Detection Based on Adaptive Boosting," *Appl. Artif. Intell.*, vol. 33, no. 5, pp. 462–482, Apr. 2019.

[37] Z. Zhang, G. Zhang, Y. Shen, and Y. Zhu, "Intrusion detection model based on GA dimension reduction and MEA-Elman neural network," in *Advances in Intelligent Systems and Computing*, 2019, vol. 773, pp. 354–365.

[38] A. Tekerek, "A novel architecture for web-based attack detection using convolutional neural network," *Comput. Secur.*, vol. 100, p. 102096, Jan. 2021.

[39] B. Yong, X. Liu, Q. Yu, L. Huang, and Q. Zhou, "Malicious Web traffic detection for Internet of Things environments," *Comput. Electr. Eng.*, vol. 77, pp. 260–272, Jul. 2019.

[40] J. M. G. Iv, D. Bhansali, M. Gratian, and M. Cukier, "A comprehensive evaluation of HTTP header features for detecting malicious websites," in *Proceedings - 2019 15th European Dependable Computing Conference, EDCC 2019*, 2019, pp. 75–82.

[41] S. Rovetta, G. Suchacka, and F. Masulli, "Bot recognition in a Web store: An approach based on unsupervised learning," *J. Netw. Comput. Appl.*, vol. 157, p. 102577, May 2020.

[42] "User Agents Analysis :: udger.com." [Online]. Available: https://udger.com/. [Accessed: 08-Nov-2020].

[43] A. Laughter, S. Omari, P. Szczurek, and J. Perry, "Detection of Malicious HTTP Requests Using Header and URL Features," Springer, Cham, 2021, pp. 449–468.

[44] O. Awotipe, "Log Analysis in Cyber Threat Detection," *Creat. Components*, Jan. 2020.

[45] "Suricata | Open Source IDS / IPS / NSM engine." [Online]. Available: https://suricata-ids.org/. [Accessed: 08-Nov-2020].

[46] Santiago Bassett and Mike Paquette, "Improve Security Analytics with the Elastic Stack, Wazuh, and IDS | Elastic Blog," 23-Oct-2018. [Online]. Available: https://www.elastic.co/blog/improve-security-analytics-with-the-elastic-stack-wazuh-and-ids. [Accessed: 08-Nov-2020].

[47] O. Negoita and M. Carabas, “Enhanced Security Using Elasticsearch and Machine Learning,” in *Advances in Intelligent Systems and Computing*, 2020, vol. 1230 AISC, pp. 244–254.

[48] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, “Newest collaborative and hybrid network intrusion detection framework based on suricata and isolation forest algorithm,” in *ACM International Conference Proceeding Series*, 2019, pp. 1–11.

[49] “pandas.DataFrame — pandas 1.2.3 documentation,” *Pandas*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. [Accessed: 02-Apr-2021].

[50] MDN contributors, “HTTP request methods - HTTP | MDN,” *Mozilla*, 04-Dec-2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. [Accessed: 02-Apr-2021].

[51] MDN contributors, “HTTP response status codes - HTTP | MDN,” *Mozilla*, 17-Mar-2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. [Accessed: 02-Apr-2021].

[52] C. Whitnall, E. Oswald, and L. Mather, “An exploration of the Kolmogorov-Smirnov test as a competitor to mutual information analysis,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 7079 LNCS, pp. 234–251.

[53] MDN contributors, “User-Agent - HTTP | MDN,” *Mozilla*, 19-Feb-2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent. [Accessed: 03-Apr-2021].

[54] E. R. Fielding and E. J. Reschke, “RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,” 2014.

[55] W. Zhang, T. Yoshida, and X. Tang, “A comparative study of TF*IDF, LSI and multi-words for text classification,” *Expert Syst. Appl.*, vol. 38, no. 3, pp. 2758–2765, Mar. 2011.

[56] F. Pedregosa FABIANPEDREGOSA *et al.*, “Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot,” 2011.

[57] A. Navada, A. N. Ansari, S. Patil, and B. A. Sonkamble, “Overview of use of decision tree algorithms in machine learning,” in *Proceedings - 2011 IEEE Control and System Graduate Research Colloquium, ICSGRC 2011*, 2011, pp. 37–42.

[58] Y. Wang, C. Song, and S.-T. Xia, “Unifying Decision Trees Split Criteria Using Tsallis Entropy,” Nov. 2015.