EXAMENSARBETE INOM ELEKTRONIK OCH DATORTEKNIK, GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2019*

Machine Learning for a Network-based Intrusion Detection System

Maskininlärning för ett Nätverksbaserat Intrångsdetekteringssystem

VILHELM GUSTAVSSON

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Machine Learning for a Network-based Intrusion Detection System

An application using Zeek and the CCIDS2017 dataset

Swedish title:
Maskininlärning för ett Nätverksbaserat Intrångsdetekteringssystem

Thesis project for the degree:
Bachelor of Science in Computer Engineering

Vilhelm Gustavsson
May 2019

Royal Institute of Technology, KTH
School of Electrical Engineering and Computer Science
Stockholm, Sweden

TRITA-CBH-GRU-2019:033

Abstract

Cyber security is an emerging field in the IT-sector. As more devices are connected to the internet, the attack surface for hackers is steadily increasing. Network-based Intrusion Detection Systems (NIDS) can be used to detect malicious traffic in networks and Machine Learning is an up and coming approach for improving the detection rate. In this thesis the NIDS Zeek is used to extract features based on time and data size from network traffic. The features are then analyzed with Machine Learning in Scikit-learn in order to detect malicious traffic. A 98.58% Bayesian detection rate was achieved for the CICIDS2017 which is about the same level as the results from previous works on CICIDS2017 (without Zeek). The best performing algorithms were K-Nearest Neighbors, Random Forest and Decision Tree.

Sammanfattning (Swedish abstract)

IT-säkerhet är ett växande fält inom IT-sektorn. I takt med att allt fler saker ansluts till internet, ökar även angreppsytan och risken för IT-attacker. Ett Nätverksbaserat Intrångsdetekteringssystem (NIDS) kan användas för att upptäcka skadlig trafik i nätverk och maskininlärning har blivit ett allt vanligare sätt att förbättra denna förmåga. I det här examensarbetet används ett NIDS som heter Zeek för att extrahera parametrar baserade på tid och datastorlek från nätverkstrafik. Dessa parametrar analyseras sedan med maskininlärning i Scikit-Learn för att upptäcka skadlig trafik. För datasetet CICIDS2017 uppnåddes en Bayesian detection rate på 98.58% vilket är på ungefär samma nivå som resultat från tidigare arbeten med CICIDS2017 (utan Zeek). Algoritmerna som gav bäst resultat var K-Nearest Neighbors, Random Forest och Decision Tree.

Keywords

- Machine Learning
- Cyber security
- Intrusion Detection System (IDS)
- Zeek
- Bro
- Flow-based Traffic Characterization
- Scikit-Learn

Contents

1 Foreword **4**

2 Introduction **5**

2.1 Problem description 5

2.2 Goal 6

3 Theory and background **7**

3.1 Network traffic 7

3.2 Malicious network traffic 8

3.3 Flow-based traffic characterization 9

3.3.1 Flow features 9

3.3.2 Feature extraction from flows 10

3.3.3 Flow-based characterization vs. Packet Inspection 10

3.4 Intrusion Detection Systems 10

3.4.1 Detection Rates 12

3.4.2 The Base-Rate Fallacy 13

3.4.3 Commercial IDS:s 13

3.4.4 Open Source IDS:s 14

3.5 Security Information and Event Management 14

3.6 Datasets for Intrusion Detection Evaluation 15
3.7 Machine Learning 16
3.7.1 Machine Learning Algorithms 17
3.7.2 Machine Learning Libraries 18
3.7.3 Machine Learning for Intrusion Detection Systems 19

4 Methodology **20**
4.1 Literature study 21
4.2 Testing existing scripts 22
4.3 Extracting flow features 22
4.4 Streaming log data to Python 23
4.5 Training and testing ML-models 23

5 Result **24**
5.1 Testing existing detection scripts for Zeek 24
5.2 Testing speed with custom scripts 25
5.3 Feature importance 26
5.4 Evaluating the ML-models 27
5.5 Python script for attack detection 29

6 Analyzis and discussion **30**
6.1 Zeek performance 30
6.2 Model evaluation 31
6.3 Applications and Sustainable Development 31

7 Conclusion **32**

8 Appendix A - Abbrevations **33**

Foreword

I want to thank the teachers at KTH Flemingsberg for a well-structured and relevant education program and my fellow coursemates for help and company along the way. Thanks to my family, my friends and other comrades for keeping me motivated. Also thanks to Robert Lagerström for being the thesis supervisor.

Introduction

Cyber attacks on digital infrastructure can result in damage on both humans, their property and the environment. Attacks on industrial process control systems could lead to life-threatening malfunctions or emissions of dangerous chemicals into the environment. A recent example is the "Triton" attack which targeted the process control systems of petrochemical plants [1]. To detect cyber security threats, Intrusion Detection Systems (IDS) can be used. An IDS monitors networks or computers in order to detect malicious activity. This thesis explores the use of Machine Learning (ML) algorithms to improve the detection rate of a Network-based IDS (NIDS) named *Zeek* [2].

2.1 Problem description

Zeek is a highly customizable platform for Network analyzis and Intrusion Detection. Configuring Zeek is done by adding scripts. There are a number of scripts for Zeek that can detect different types of malicious traffic, however, as new attacks are invented at a high rate, writing scripts for all of them becomes a time-consuming task. A more general solution is needed to complement Zeek:s detection capability. This is where Machine Learning (ML) becomes useful. An ML-model can be trained from a dataset with malicious and benign traffic. This model can then be used to detect malicious traffic flows in Zeek:s log files. Previous works have done similar experiments but with other IDS:s [3], different datasets [3] [4] [5], without a real NIDS integration [6] or without real-time ML-classification [4]. The novelty of this thesis project is the specific combination of Zeek, the selected dataset and real-time ML-detection.

2.2 Goal

The goal of this thesis project is to:

- Perform machine learning classification on Zeek logs to improve Zeeks detection of malicous traffic.
- Research which traffic features and machine learning algorithms that are suitable for detecting different kinds of malicious traffic.
- Write Zeek scripts to extract those features.
- Train machine learning models from a labeled dataset with benign and malicious traffic.
- Evaluate the performance of the different models and scripts.

Delimitation

The set of attacks tested in this project is limited to the attacks in the CIC-CIDS2017 dataset. The focus in this thesis is on *detecting* malicious traffic, not *preventing* it as Zeek lacks such capability. The machine learning is limited to statistical learning methods, no neural networks or deep learning is used. The alarms produced by the machine learning classification are just output to a text file or terminal, no user interface is designed.

Theory and background

This chapter describes the technologies used in the project and how they have been combined in previous works. The reader should here be provided with sufficient knowledge to understand the methodology, results and analyzis.

3.1 Network traffic

Network traffic consists of packets sent from a source to a destination. Network architecture is separated into layers with different *protocols* operating on each layer to perform specific tasks. The OSI (Open Systems Interconnection)-model describes seven layers from the physical layer which is the physical cable or wireless medium up to the application layer with the user interface. Most relevant for this project is layer 4, the transport layer. The most used protocols on layer 4 are TCP (Transport Control Protocol) and UDP (User Datagram Protocol) [7]. Most packets sent through the transport layer up to the application layer are encapsulated in either a TCP segment or UDP datagram. This means that TCP and UDP are intermediaries for most protocols in the above layers. Because of this, when trying to characterize traffic, looking only at TCP and UDP protocols can be sufficient [8]. The datasets used in this project contain malicious traffic that consists of TCP, UDP and also some ICMP packets. ICMP (Internet Control Message Protocol) operates on layer 2, the Network/Internet protocol layer and could also be taken into account for the characterization process.

3.2 Malicious network traffic

Malicious network traffic is here defined as any traffic produced from an attack with the intent of doing harm or intrusion to a computer system or network. The datasets selected for this project.[1] contain the following classes of malicious traffic:

- **Bruteforce** attacks tries to guess passwords or encryption keys by trying a high number of possible combinations, often using some pattern or word list to reduce the number of guesses [9, p. 98].
- **Denial of Service (DoS)** attacks prevents the normal use of a network or computer system. It could be overloading a network with packets to degrade its performance or targeting a specific host computer to prevent users from accessing it [9, p. 36].
- **Botnet** attacks make use of a network of Bots (Zombies) which are computers infected with malware that can be activated to perform attacks, such as spam e-mail or DoS, on other machines [9, p. 229].
- **Port scanning** is when an attacker sends probe packets to a network or system and gather intelligence from the responses received. The attacker can detect which ports are open and if there are any vulnerable services running on the ports [9, p. 294].
- **SQL-injections** are malicious database queries, often designed to extract bulk data from a database. They can be injected through badly protected forms on web pages for example [9, p. 177].
- **Cross-site-scripting (XSS)** attacks are performed by putting malicious scripts in the HTML content of a web page. It can be used to steal cookies from a user in order to impersonate them [9, p. 391].
- **Heartbleed** is a bug in previous Open-SSL implementations of the Heartbeat Protocol , discovered in 2014. The bug leaves the memory on a host accessible to anyone who can exploit it over a network [9, p. 696].

[1] See section 3.6

3.3 Flow-based traffic characterization

A *flow* is here defined as a stream of packets that has a flow ID which consists of the features Source IP, Destination IP, Source Port, Destination Port and Protocol. The flows are bi-directional which means that the first packet of a flow determines which is the forward direction (source to destination). The response packets are sent in the backward direction (destination to source). The flows correspond to TCP connections and UDP streams on the OSI Transport layer and have a maximum duration (timeout). The timeout means that after a certain threshold, the current state of the flow is logged and analyzed. For long connections or streams this means they are split into several smaller flows.

In [8] it is proven that flow/time features can be used to detect certain types of network traffic and applications running over the network. The report states that a flow timeout of 15 seconds gave the best classification results. According to [6] it is also possible to detect certain types of malicious traffic such as DoS and Botnets using flow-based characterization. The flows that were analyzed had a timeout of 120 seconds for both UDP and TCP.

3.3.1 Flow features

The following list contains the main flow features, as suggested by [8] and [6]. For each feature in the list, the mean, standard deviation, minimum and maximum are also calculated.

- Forward Inter Arrival Time: Time between two packets in forward direction.
- Backward Inter Arrival Time: Time between two packets in backward direction.
- Flow Inter Arrival Time: Time between two packets in either direction.
- Active: Duration of sending packets for before going idle.
- Idle: Duration of idleness before starting to send packets again.
- Flow Bytes per second: Bytes sent per second in either direction.
- Flow packets per second: Packets sent per second in either direction.
- Duration: Time between the first and the last packet of the flow.

3.3.2 Feature extraction from flows

In this project, Zeek is used to extract the features. Zeek does not natively extract all the features[2], some scripting is needed to achive this. Flow features can also be extracted from a traffic dump(PCAP) file using CICFlowmeter [8]. Another possibility is the Cisco NetFlow Analyzer which also can extract some of these features from live traffic flows [10].

3.3.3 Flow-based characterization vs. Packet Inspection

Packet inspection is to inspect the payload data of each network packet in order characterize the traffic. This is a very resource consuming process and not always applicable as the payload can be encrypted. Flow-based traffic characterization is much more efficient, but on the downside less accurate than Packet Inspection [4].

3.4 Intrusion Detection Systems

An Intrusion Detection System(IDS) does not necessarily only detect intrusions but can also detect other types of malicious traffic such as DoS attacks or port scans. The architecture of IDS:s differ but generally they consist of three logical components:

- **Sensors** collect and decode packets and forward them to the analyzer.
- **Analyzers** are responsible for detecting suspicious traffic and taking action, like producing an alert.
- The **User interface** can display the output from the analyzer and can be used to configure the IDS.

Furthermore, IDS:es can be separated into two different classes, *Host-based* IDS (HIDS) and *Network-based* IDS (NIDS). A HIDS typically monitors activity on a host computer, keeping track of system calls, open processes, etc. A NIDS monitors the traffic of a network, looking for suspicious activity. An IDS works based on the assumption that malicious activity differs from normal activity and can be characterized by analyzing the traffic. There is not always an exact distinction between malicious and normal behaviour, rather some overlap between the two is to be expected [9, p. 278-280].

[2]See section 4.3

A NIDS sensor can either be installed as inline or passive. An inline sensor is often installed into another network device, such as a firewall or a router. This means the traffic has to pass through the NIDS, possibly affecting network performance. More common are passive NIDS which are installed as wire taps on the network, monitoring a copy of the traffic. A passive sensor has two network interfaces (NIC), one for the network tap and one to communicate with the central analyzer or a SIEM.[3] The main motivation for installing a NIDS sensor inline is to also use it as an *Intrusion Prevention System* (IPS), to block malicious traffic.

Where to place the NIDS sensor in the network topology is also an important consideration. Consider Figure 3.1 as a common company network setup. A sensor in position 2 has the advantages that it can see successful attacks from the outside world. A sensor in position 1 can also see attempted attacks stopped by the firewall. The IDS:es in the other locations can detect infiltration attacks (like botnets[4]). A sensor in position 1 has a high processing burden because it monitors all unfiltered traffic. Distributing the sensors and analyzers to positions like 3 and 4 can reduce the workload on single devices [9, p. 291-293].

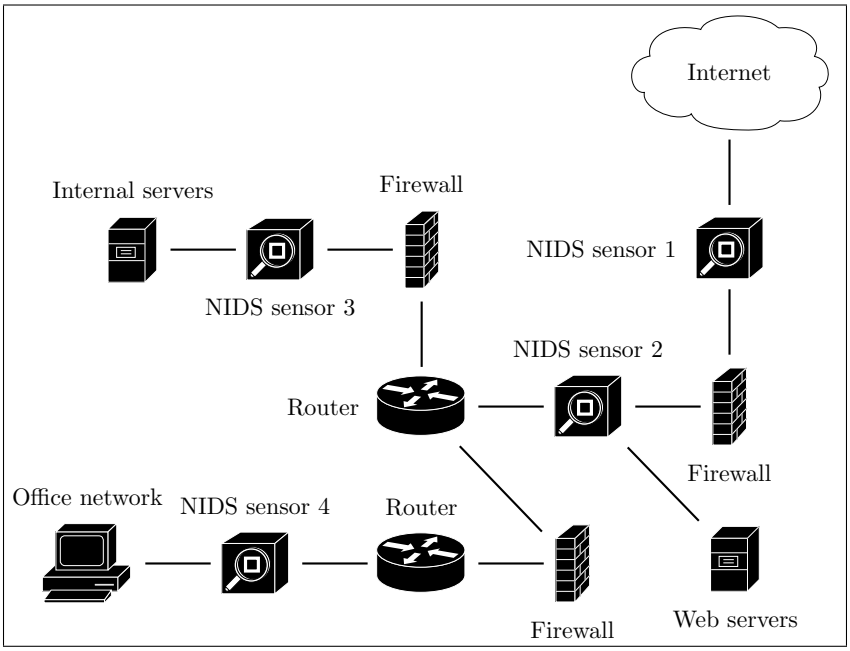


Figure 3.1: Example of NIDS sensor deployment. Based on the figure in [9, p. 292].

[3]See section 3.5
[4]See section 3.2

3.4.1 Detection Rates

There are some important parameters to take into account when measuring the accuracy of an IDS. They can be described using conditional probability (Equation 3.1) and Bayes theorem (Equation 3.2).

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{3.1}$$

Equation 3.1: Conditional probability.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{\sum_{i=1}^{n} P(A_i) \cdot P(B \mid A_i)} \tag{3.2}$$

Equation 3.2: Bayes theorem.

Where $P(A \mid B)$ is the conditional probability of A given that B has occurred.

The parameters used to describe detection rates are:

- **True Positive rate** or detection rate is the probability $P(A \mid I)$ i.e the probability that an alarm A will be raised given that there has been an intrusion, I.
- **False Positive rate** or false alarm rate is the probability $P(A \mid \neg I)$ i.e the probability that an alarm A will be raised even if there hasn't been an intrusion, $\neg I$.

The theoretical ideal IDS has the largest possible value for

- $P(I \mid A)$ the probability that an alarm indicates an intrusion called the **Bayesian detection rate**.
- $P(\neg I \mid \neg A)$ the probability that the absence of an alarm indicates that there has been no intrusions.

Bayes Theorem can be applied to calculate $P(I \mid A)$:

$$P(I \mid A) = \frac{P(I) \cdot P(A \mid I)}{P(I) \cdot P(A \mid I) + P(\neg I) \cdot P(A \mid \neg I)} \tag{3.3}$$

Equation 3.3: Bayesian Detection Rate.

Equation 3.3 shows that if the probability for an intrusion to occur $P(I)$ is low, then the false positive rate $P(A \mid \neg I)$ in the denominator will dominate the true positive rate $P(A \mid I)$ unless the FPR is very low, giving a small Bayesian Detection rate. This is is known as the Base-Rate Fallacy [11].

3.4.2 The Base-Rate Fallacy

A good IDS has a high detection rate and low false alarm rate. If the system has a low detection rate, it gives a false sense of security, and if it has a high false alarm rate, the security managers time is wasted analyzing false alarms. The Base-Rate fallacy states that if the amount of malicious traffic is small compared to the amount of benign traffic, then the false alarm rate will be high unless the IDS has a very high detection rate. This is important to take into account when choosing data for evaluation of an IDS [9, p. 281].

For example imagine a log with 100,000 records out of which 10 represent intrusions. This can be described with probabilities:

$$P(I) = \frac{1}{\frac{100000}{10}} = 0.0001\%$$

$$P(\neg I) = 1 - P(I) = 0.9999\%$$

Inserting these values into the equation for Bayesian Detection rate gives:

$$P(I \mid A) = \frac{0.0001 \cdot P(A \mid I)}{0.0001 \cdot P(A \mid I) + 0.9999 \cdot P(A \mid \neg I)}$$

Imagine the ideal TPR $P(A \mid I) = 1$. If a 90% Bayesian detection rate is desired, the false positive rate would have to be in the order of 1e-5, which is unrealistic because in real applications there is usually some overlap between intrusive and benign behaviour. This means that the FPR is what limits the Bayesian detection rate [11].

3.4.3 Commercial IDS:s

Juniper, **Palo Alto** and **Check Point** are the market-leading commercial IDS/IPS:s. In an evaluation of their NIDS capability with three datasets they all detected 50% of the attacks [12].

3.4.4 Open Source IDS:s

Snort works as both a NIDS and HIDS as well as IPS. Snorts NIDS analyzer tries to match each packet it receives with a set of rules defined by the Snort configuration. Rules can be specified to detect certain content in packet payload or other features found in the packet headers. When a rule is matched against a packet, Snort can take actions such as Alert, Log packet, Ignore packet or Drop packet [9, p. 302-306].

Suricata works much the same as Snort but is more scalable as its analyzer supports multi-threading. It's also faster than Snort but consumes more resources [3]. There is a Machine Learning integration for Suricata called OPNids MLE.

Zeek, formerly known as **Bro**, is a network security monitor that offers IDS capabilities. Zeek works different from Snort and Suricata as Zeeks analyzer consists of an event engine that is triggered upon different events in the traffic. The events are passed to Zeek:s *script land* where traffic features are further analyzed using Zeeks own scripting language which produces log files and notifications about suspicious or interesting activity. The scripting language is Turing-complete and makes Zeek a highly customizable platorm for traffic analyzis [2]. Figure 3.2 shows Zeek:s internal architecture.

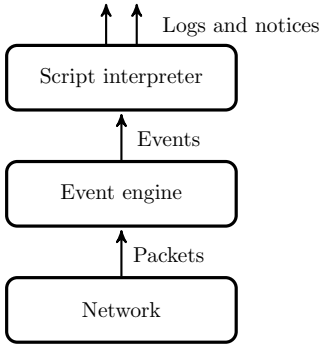


Figure 3.2: Zeek:s internal architecture.

3.5 Security Information and Event Management

A security information and event management (SIEM) system provides a user interface to gather data from several different IDS:s and similar applications. By aggregating and correlating the data, the SIEM can detect information or events that require action from the user [9, p. 463]. Apache Metron is a SIEM system made for distributed systems that offers integration with Zeek [13].

3.6 Datasets for Intrusion Detection Evaluation

The Canadian institute for Cyber Security (CIC) provides a comprehensive set of network traffic datasets used by security researchers world-wide [14]. The datasets that were encountered in the research and considered for this project are:

- The **CICIDS2017** dataset consists of *benign* traffic generated by users mixed with malicious traffic from the most common(according to McAfee 2016 [15, p. 45]) cyber attacks. The traffic was recorded in a controlled network environment over 5 days and include attacks such as Bruteforcing, DoS and Botnets. The dataset is delivered as a set of packet capture (PCAP) files, which can be replayed on a computers network interface. The dataset also comes with a time schedule for the attacks and a spreadsheet file which contains more than 80 extracted features for each traffic flow, labeled as either benign or <name-of-attack>. The total size is around 50 GB [6].
- The **CSE-CIC-IDS2018** dataset is similar to CICIDS2017 but includes more attacks and traffic from a larger network. It is organized in different PCAP files for each host on the network and also comes with spreadsheets for ML-analyzis. The total size is over 400 GB [16].
- The **NSL-KDD** dataset is an improvement of it's predecessor KDD-CUP99. It is distributed as a set of Attribute-Relation files (ARFF) and has 42 features per record [17].
- **ISCX-SlowDoS-2016** is a DoS dataset based on benign traffic from the ISCX-IDS dataset. It comes as a PCAP file with 24 hours of benign traffic mixed with different application layer DoS attacks [18].

3.7 Machine Learning

Machine Learning (ML) is a sub-category of Artificial Intelligence (AI) and is based on statistical learning methods used to understand and make predictions from data. ML can be separated into two categories, supervised learning and unsupervised learning.

In supervised learning, input data and labeled output data is used to *train* a statistical model in order to *predict* the output of new input data. The simplest mathematical example is perhaps the least-square method, which produces a straight line (the model) from a set of x and y coordinates. Then for any new x-value (input) the line will predict a y-value (output).

Unsupervised learning can be used when there is no labeled output data at hand. Unsupervised learning is used to analyze the relationships between the input data points. One such approach is clustering, which groups together the input data points based on the similarity of their features [19].

As this thesis project makes use of *labeled* datasets and the goal is to make predictions about malicious traffic, the ML performed here falls into the category of supervised learning. Depending on the datasets, the different ML algorithms have different accuracy and performance. The difficulty in using ML for a project like this lies in finding the most relevant features and finding the optimal algorithm and parameters.

Building the ML-model from a dataset is called *training*. The training is performed on a subset of the dataset called training data. The rest of the dataset is used for validating or *testing* the model. The split is usually done with a majority of the data used for training For example 80/20 (%) for training/testing.

When training a model, an important goal is to find the optimal bias-variance trade-off. Variance is a measure of how much the model is changed when using new training data, i.e. the flexibility or complexity of the model. And bias is the error introduced by applying a simple model to a real-life problem. The optimal bias-variance trade-off has the lowest possible bias while maintaining the lowest possible variance. The two quantities are correlated because a simple model with high bias(like least-squares method) has a low variance and a flexible model with with high variance has low bias [19, p. 33-36].

High variance can lead to an important problem called *overfitting*. This means the model follows errors in the training data too closely, finding patterns that shouldn't exist, yielding an overly complex model and a bad accuracy [19, p. 28].

The labels in a dataset can be either quantitative or qualitative. Qualitative labels are discrete values belonging to a class, in this project the labels are qualitative since the labels of the malicious traffic are the names of the at-

tack. Quantitative labels on the other hand are continuous values. Data with qualitative labels require *classification* algorithms while data with quantitative labels require *regression* algorithms. The features of the data can also be both qualitative or quantitative but this is less important when choosing the right algorithm [19, p. 28-29].

The features in a dataset are not always correlated with the output in the label. Such features should be excluded because they can result in an unnecessarily complex model. *Feature selection* can be performed to remove features with a low correlation to the output [19, p. 78].

3.7.1 Machine Learning Algorithms

Machine learning algorithms mentioned or used in this thesis are listed here with a short conceptual description.

- **Support Vector Machines (SVM)** uses hyperplanes in high dimensions to separate data into different classes [19, p. 350].
- **K-Nearest-Neighbors (KNN)** classifies new observations by using the majority class of the K nearest observations in the training dataset [19, p. 39].
- **Naive Bayes (NB)** uses conditional probability , i.e. the *Bayes Theorem*[5] to assign the most likely class to an observation, given the values of its features [19, p. 38].
- **Quadratic Discriminant Analyzis (QDA)** classifies new observations by using Bayes theorem and covariance matrices for each class [19, p. 149].
- **Artificial Neural Networks (ANN)**, also called *Deep Learning*, are inspired by the behaviour of neurons in brains. Logically they consist of interconnected nodes in layers that performs calculations to make classifications. The layers can be many, making the network *deep* [20]. Deep learning is not used in this project, but it is worth mentioning for future work.
- **Decision Tree (DT)** classifies data by traversing through a tree structure, asking relevant questions about the features of the data, when the traversing reaches a leaf node, the data point is classified according to the class in the leaf node [19, p. 303]. ID3 is a popular version of DT. See Figure 3.3.

[5] See Equation 3.2.

- **Random Forest (RF)** aggregates and produces a mean of the result of several Decision Trees trained from different random subsets of the training data [19, p. 319].

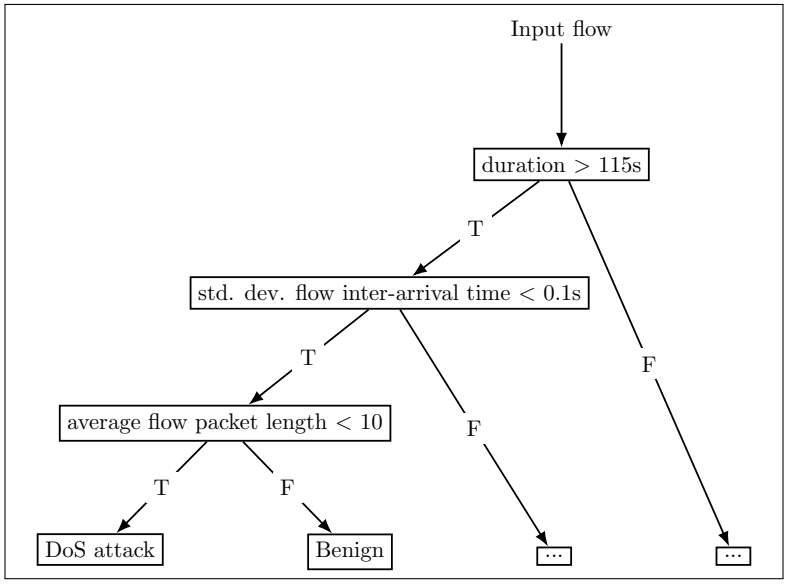


Figure 3.3: Part of a decision tree model for flow-based traffic characterization (this is just an example with mock values).

3.7.2 Machine Learning Libraries

The ML-libraries mentioned in this project are:

- **Scikit-Learn** is built on the other Python libraries NumPy (for mathematical functions) and Scipy (for scientific tasks). It offers a wide range of implemented algorithms for classification, regression, clustering, dimensionality reduction and preprocessing of data for use with the algorithms [21].
- **MLLib** is a part of Apache Spark which is an analytics engine for large-scale data processing. [22] MLlib offers much of the same functionality as scikit-learn but is more scalable for large distributed applications. It works with the programming languages Java, Scala, Python and R [23].
- **Weka** is written in Java and offers the same range of functionality as Scikit-Learn and MLLib. Weka also supports Deep Learning [24].

3.7.3 Machine Learning for Intrusion Detection Systems

An article from 2016 compared the performance on different machine learning algorithms on a dataset called NSL-KDD with 42 features per record. The best performing algorithm was RF with 99% accuracy. Suggested for future work using the classifiers for multi-class classification (classifying multiple attacks) and with a different set of features [5].

In a previous work from 2017, the authors developed an ML-plugin for Snort using the Weka library. Because Snorts analyzer only detects known malicious traffic, the plugin was run in parallel to detect unknown or altered attacks, it also effectively reduced the False-alarm rate of Snorts analyzer. The ML-models were trained on three different datasets and the tests performed with new generated traffic. The algorithms tested were SVM, Decision trees, Fuzzy Logic, BayesNet and NaiveBayes and hybrid versions of these. The best results were achieved using the hybrid versions of the SVM algorithm. In their conclusion, the authors suggest their work is extended to cover other intrusion detection systems, other hybrid ML-algorithms and more fine-tuning of parameters [3].

The creators of the CICIDS2017 dataset analyzed their data using ML-algorithms from Scikit-Learn. First 80 different features from each TCP/UDP flow were extracted using the CICFlowmeter as feature extractor. Using Random Forests they calculated the most relevant features for detecting each attack and were able to correctly classify the traffic with up to 98% accuracy. For example, the best features for detecting DoS traffic were Average packet size, Flow duration, Standard deviation of flow inter-arrival time and Standard deviation of packet lengths in backward direction. The best performing algorithms and precisions were RF (98%), ID3 (98%) (a type of Decision Tree), KNN (96%) and QDA (97%) [6].

In an article from 2017 the authors ran Zeek (named Bro at that time) against labeled datasets and used the standard set of features extracted from Zeek logs. They performed ML-analyzis with Weka to decide the most relevant features for detecting three different botnet attacks. The features were then used to write Zeek scripts which gave a high TPR but also a high FPR. Because of the high FPR (20% for some attacks), they suggest to use payload inspection as a complement to flow-based detection [4].

In a masters thesis from 2018 the students used Deep Learning with the Deep Q Network on the CICIDS2017 dataset and got an accuracy of 92%. They also state that their model can detect unknown attacks [25].

Methodology

A literature study was conducted to gather important knowledge and hints from previous works in the field of Intrusion Detection and Machine Learning. Then Zeek:s architecture and scripting language were studied to produce the scripts for feature extraction. The scripts were then run on a dataset and the logs analyzed with Machine learning techniques from Scikit-Learn. Figure 4.1 shows an overview of the method.

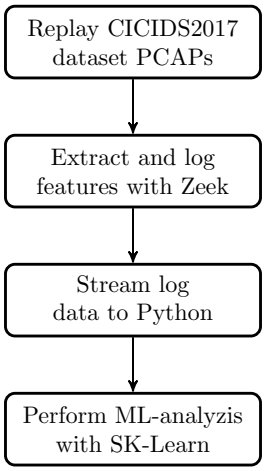


Figure 4.1: Overview of the methodology.

4.1 Literature study

Descriptions of the works studied in the literature study can be found in section 3.7.3. This project is partly inspired from an article on a performance comparison between the Snort and Suricata IDS:s [3]. The article proposes a Machine Learning(ML) plugin for Snort and also suggests to do similar experiments with other IDS:s. The main motivations to use Zeek IDS for this project were:

- Like Snort and Suricata, Zeek is free and Open-source and also matches some of their functionality.
- Zeek is highly configurable through its own scripting language.
- Zeek have had tools developed for doing log analyzis with Python [26]. The Python integration is important because the ML-library Scikit-Learn supports Python.
- The other popular open-source NIDS, Suricata, already has an ML-integration called OPNids MLE [27].

The Snort/Suricata article has a reference to The Canadian Institute for Cyber Security which has a comprehensive set of datasets for IDS evaluation and ML-analysis.[1] CICIDS2017 was chosen over the other datasets because it contains a reasonable amount of traffic, is up-to-date and organized in a good way with PCAP files that can be parsed with Zeek.

The article on CICIDS2017 [6] uses the approach of flow-based traffic characterization. The CICIDS2017 article also refers to [8] which further explains the concept of flow-based traffic characterization.

Previous works used a number of different ML-algorithms for classification. Those with the best performance were chosen for evaluation in this project.[2]

[1] See section 3.6.
[2] See section 3.7.

4.2 Testing existing scripts

Existing detection scripts were searched for to see which attacks in the dataset that Zeek can detect without using ML-algorithms. Scripts for FTP bruteforce, Heartbleed, SQLi and Port Scanning were found in the Zeek distribution and a script for DoS Slowloris was found from an external source [28].

4.3 Extracting flow features

A new script was written to extract the features described in section 3.3.1. For every new packet, the script stores Inter-arrival times, packet lengths, etc. in a record. The record is passed on to each subsequent packet in the same flow. When the flow is finished either by a connection teardown or timeout, the flow features are calculated from the values in the record and output to a log file.

For each new connection, the script also initiates a polling process that uses a callback function every 120 seconds to calculate and log the features of the connection. This is done to split up long connections into smaller flows so that possible malicious flows can be detected before the connection is over. This is needed because connections can potentially have hour-long durations.

The script uses a low-level function which is called for every packet processed by Zeek. According to the Zeek documentation, the function is "a very low-level and expensive event that should be avoided when at all possible" [29]. Therefore, another simpler script which doesn't use the function is also written. It extracts significantly less features and miss those features ruled most important by [6]. However, according to [4] these features might still give decent classifications. The simple script also splits up long connections into smaller flows.

4.4 Streaming log data to Python

Using a Python script called "log to dataframe" from Bro Analyzis Tools (BAT) [26] with slight modifications, the log data is streamed into Python and converted to a Pandas dataframe which is a format compatible with the ML-library Scikit-Learn. This works both for streaming an active log and for reading a static log.

Another way to stream the log data is to install a Zeek plugin which uses the Apache Kafka message queue to stream log data into Pyspark. Pyspark has its own MLLib which can be used for the ML-classification.[3] The Kafka plugin is a part of the Apache Metron project and is suitable for distributed systems with many Zeek instances running in parallel, like in Figure 3.1. As the application in this project runs only one instance of Zeek and the ML-classification on the same host, BAT:s log_to_dataframe script and Scikit-Learn is sufficient.

After streaming the logs into Python, the flows are labeled using the attack schedule from the CICIDS2017 dataset web page [14].

4.5 Training and testing ML-models

Scikit-Learn has the functions necessary to perform data preprocessing such as splitting the training and testing data, performing the actual ML with different algorithms such as RF, SVM or KNN and for producing metrics like feature importance, accuracy scores and confusion matrices as seen in Result

The ML-models are persisted to a file using SK-learns persistence module Joblib. The file can be loaded by a Python script that monitors the Zeek logs and performs real-time classifications as Zeek logs the live traffic on a network interface.

[3] See section 3.7.2.

Result

The following tests and comparisons were conducted to decide the best way to extract the features and train a model. All tests were run on a virtualized Ubuntu machine with 4×2.7GHz processor cores and 16GB memory.

5.1 Testing existing detection scripts for Zeek

To see if Zeek can detect any of the attacks without any ML-algorithms, existing detection scripts were tested on the dataset. Table 5.1 shows the results. All the scripts except the SQLi script produced notices which correctly identified the attacking machines IP address.

Table 5.1: Success of existing detection scripts tested.

DoS Slowloris	FTP Bruteforce	Heartbleed	SQLi	Port scanning
✓	✓	✓	✗	✓

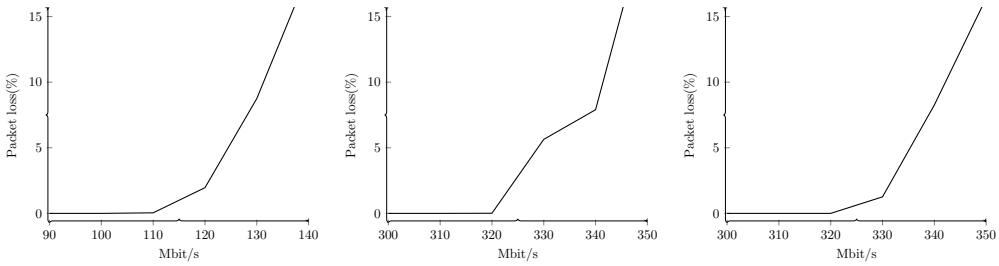
5.2 Testing speed with custom scripts

Two custom scripts were written for Zeek. One more complex script that extracts 50 flow features and one simple that extracts only 10. The reason for writing two scripts is that the complex script has to make use of a low-level function which drastically reduces Zeek:s performance.

Adding new scripts to Zeek increases the workload and might reduce the volume of traffic that Zeek can handle. A test was conducted to see how much Zeek is slowed down by the scripts.

A sample PCAP file was replayed on the network interface at different speeds using the Linux program Tcpreplay. When the workload gets too high for Zeek, it starts to drop packets. This is referred to as packet loss. The packet loss was irregular and differed for runs with the same speed settings, therefore the mean packet loss of five runs per speed setting was used for the plots in Figure 5.1.

(Note! The x scale on plot (a) differs from (b) and (c).)



(a) With complex script (b) With simple script (c) Without scripts

Figure 5.1: Packet loss for Zeek running with and without the custom scripts.

When running Zeek with the complex script, the packet loss starts to occur at around 110Mbit/s. Running the simple script or without any custom scripts at all works up until around 320Mbit/s. This indicates that the simple script only has a small impact on the performance, but with the complex script, Zeek can only handle about a *third* of the traffic volume.

5.3 Feature importance

SK-Learns function SelectKBest was used to see the importance scores for the features. The scores shows how important each feature is for making correct classifications. Figure 5.2 shows the importance scores for the simple script and Figure 5.3 shows the 10 best features (out of 50) for the complex script.

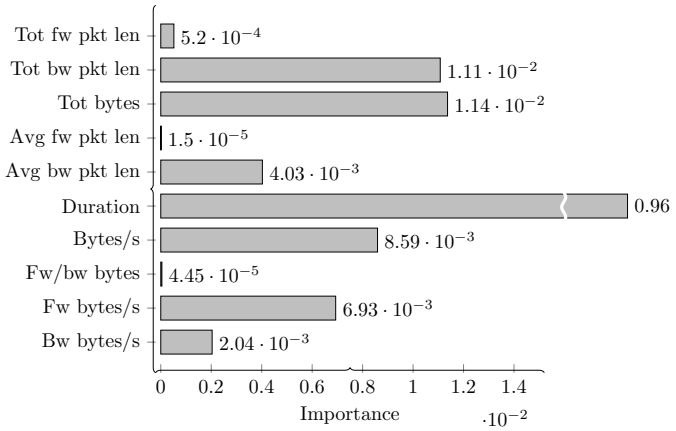


Figure 5.2: Normalized feature importance scores for the simple script.

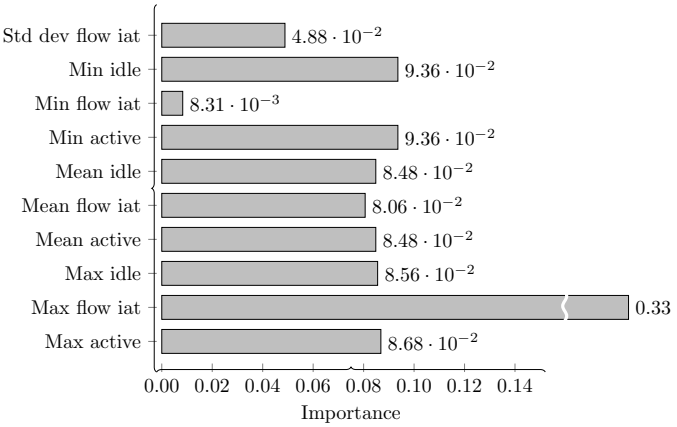


Figure 5.3: The 10 best normalized feature importance scores for the complex script.

The scores for duration in Figure 5.2 and Max flow iat in Figure 5.3 are clipped off because they are much larger than the others. Worth noting is also that Figure 5.2 has none of the 10 best features showed in Figure 5.3.

5.4 Evaluating the ML-models

Using SK-learns Random Forest Classifier, which was deemed the best algorithm by both [6] and [5], the accuracies and speeds for both the simple and complex scripts were measured. Figure 5.4 shows the classification accuracies for each attack.

The total amount of data in the dataset traffic is 48 GB sent over 41 hours. After parsing with Zeek, a total of 2,254,964 flows were generated. This gives an average of about 15 flows per second. 75% of the data was used for training and 25% was used for testing, which corresponds to 563741 test flows. Table 5.2 shows the test performance of SK-Learns Random Forest Classifier as the average number of flows classified per second.

Table 5.2: Total execution time and classifications per second.

	Simple script	Complex script
Total time	3.62	2.35
Flows / s	155,730	239,890

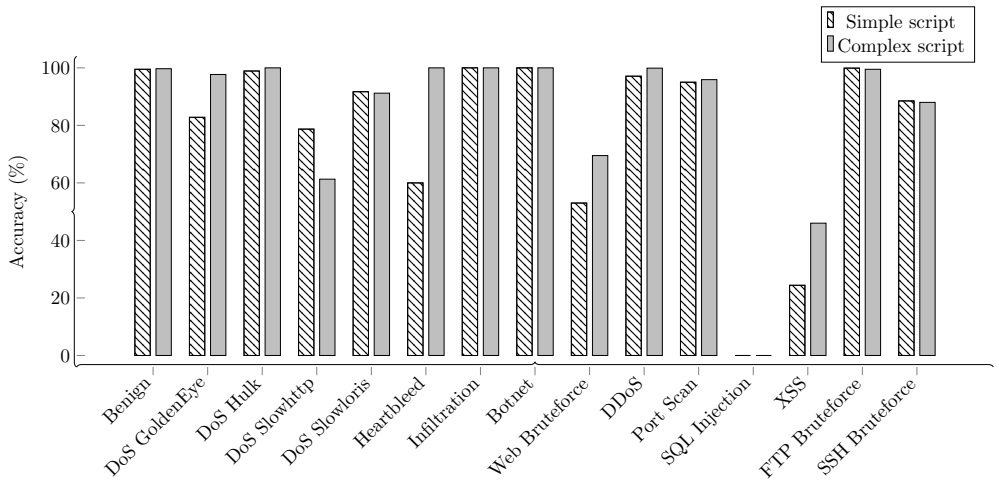


Figure 5.4: Classification accuracies for the attacks.

Since the complex script had such a huge drop in performance of Zeek, and the accuracy doesn't differ too much between the scripts different feature sets, the simple script was chosen for further experiments.

The different algorithms were tested and their speeds and accuracies are compared in Table 5.3. SK-learns default hyper parameter settings for the algorithms were used.

Table 5.3: Speeds and accuracies of the different ML-algorithms.

	DT	RF	KNN	QDA	NB	SVM
Total time (s)	**0.36**	3.56	182.85	3.95	2.46	-
Accuracy (%)	98.87	98.95	**99.16**	19.18	10.33	-

The SVM algorithm seemed to get stuck. It did not finish in over 3 hours and was therefore ruled out. DT, RF and KNN were the best performing algorithms, and were therefore chosen for further comparison.

Table 5.4 is the confusion matrix for the RF-classifier. The columns shows the predicted attack class and the rows shows the actual attack class of the flows. The diagonal holds the True Positives, i.e. the correct classifications.

Table 5.4: Confusion matrix for the RF-classifier with the simple scripts feature set.

	Benign	DoS GoldenEye	DoS Hulk	DoS Slowhttp	DoS Slowloris	Heartbleed	Infiltration	Botnet	Web Bruteforce	DDoS	FTP Bruteforce	Port Scan	SQL Injection	SSH Bruteforce	XSS
Benign	$9 \cdot 10^5$	939	108	465	53	1	2	2	212	785	2	1,641	4	59	135
DoS GoldenEye	811	3,830	4	19	2	0	0	0	0	0	0	1	0	0	0
DoS Hulk	155	14	41,314	341	12	0	0	0	4	0	0	0	0	0	1
DoS Slowhttp	540	17	271	3,806	278	0	0	0	0	0	0	1	0	0	0
DoS Slowloris	41	2	15	278	3,793	0	0	0	0	0	0	20	0	0	0
Heartbleed	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Infiltration	3	0	0	0	0	0	13	0	0	0	0	0	0	0	0
Botnet	0	0	0	0	0	0	0	182	0	0	0	0	0	0	0
Web Bruteforce	168	0	2	0	0	0	0	0	324	0	0	9	0	0	123
DDoS	1,078	0	0	0	0	0	0	0	0	36,970	0	0	0	0	0
FTP Bruteforce	4	0	0	0	0	0	0	0	0	0	986	0	0	0	0
Port Scan	1,922	0	0	0	12	0	0	0	8	0	0	38,132	0	0	1
SQL Injection	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SSH Bruteforce	73	0	0	0	0	0	0	0	0	0	0	0	0	569	0
XSS	119	0	3	1	0	0	0	0	110	0	0	8	0	0	69

Actual attack class

Predicted attack class

The classification can be made into a binary classification problem if all the attacks are grouped together as just Malicious. From the binary classification confusion matrices, the TPR and FPR can be derived. To make sure the Base-Rate Fallacy[1] is avoided, the Bayesian Detection Rates (BDR) are also calculated. The values are shown in Table 5.5.

Table 5.5: TPR, FPR and BDR of the best performing ML-algorithms.

	DT	RF	KNN
TPR (%)	96.47	96.50	**97.76**
FPR (%)	0.57	0.48	**0.21**
BDR (%)	96.2	96.79	**98.58**

5.5 Python script for attack detection

The Decision Tree model was saved to a file using SK-Learns persistence module *joblib*. A Python script was produced that loads the model from the file, streams the Zeek logs and then classifies the flows as Zeek logs the live traffic. Replaying the CICIDS2017 dataset worked and the classifications were accurate as in the results above. As another simple test, the ISCX-SlowDoS-2016 dataset was also replayed on the network interface and the script was able to detect many of the DoS attacks but classified most of them as either FTP bruteforce or Port scans.

[1] See section 3.4.2.

Analyzis and discussion

The final result shows a good detection capability and a promising performance for real-world applications. The lack of IPS capabilities in Zeek is a problem however as attacks not only need to be detected but also prevented. This has to be complemented with other software. This application should be integrated with a larger platform that has a proper User Interface, such as Apache Metron, for a good overview of the intrusion alerts produced.

6.1 Zeek performance

The huge drop in performance when using the complex script makes it infeasible to use for real applications. However, the accuracy scores shows that the simple script perform nearly as good, altough not using the best features. It is worth noting that none of the ten most important features from the complex script are used with the simple script but still the accuracies are good. Zeek produces many other logs from which other data could possibly be aggregated and used to find more attacks and make more accurate classifications. This is left for future work.

The only attack that could not be detected by any of the scripts was SQL Injection, this could because SQLi flows and benign flows mostly differ by the content of their payloads, which is not inspected when performing only flow-based traffic characterization. Except for the SQLi script, the existing detection scripts worked for their intended attacks. There are numerous other Zeek scripts available for detecting other events. The approach of integrating Zeek with SK-learn should merely be seen as a complement to the Zeeks existing functionality.

6.2 Model evaluation

The three best performing algorithms are DT, RF, and KNN. They have a similar performance in accuracy but DT and RF clearly outperforms KNN in speed. This is in line with the results of previous works [6] [5]. The Bayesian Detection Rates of the three algorithms are close to their accuracy scores so the Base-Rate Fallacy[1] is avoided. This is because the dataset has a sufficient amount of malicious traffic compared to the benign traffic volume. Since the classifications for 5 days worth of traffic were made in a matter of seconds, the ML-algorithms should have no trouble keeping up with high network speeds.

Looking at the confusion matrix for the RF-classifier, the absolute majority of the misclassifications comes from Benign flows being classified as an attack or an attack being classified as Benign. This indicates that the set of attacks is diverse. The only exceptions are some notable confusion between the different DoS attacks and between the XSS and Web Bruteforce attacks. The overall misclassification is low, hence the good detection rates.

6.3 Applications and Sustainable Development

The classifiers performs well on a diverse set of attacks. This gives a hint that it can be used to detect many other kinds of attacks and traffic. To do this, the model has to be trained with more datasets that include traces of the kind of attacks which are to be detected. For example a penetration tester could record the traces when testing attacks and use them to train the model.

The security of IOT devices is often neglected [30]. An IDS could record normal IOT behaviour and build an ML-model from it which could detect when the devices start to behave in a way that differs from normal operation.

Without protection, critical computer systems are left open for attacks. In some cases this can have dangerous consequences for both humans and the environment. In 2010 a virus that sabotaged Irans nuclear program, Stuxnet, was discovered [31]. The attack on the Ukrainian power grid in 2015 left people without electrical power for several hours during the winter [32]. In 2017 the Triton malware was discovered which was an attempt to sabotage petrochemical plants in the Middle East [1]. An IDS could help in detecting such attacks. This project is intended to be a small piece in a larger puzzle which can help to prevent cyber attacks to cause damage on humans and the environment.

[1] See section 3.4.2.

Conclusion

Zeek can effectively be used with Python and Scikit-Learn to improve real-time detection of malicious traffic in networks. This work also shows that a smaller subset of the features suggested in [8] can still perform good in network traffic characterization. The amount of flow features that can be extracted using Zeek scripts is limited because a function needed in the script drastically reduces the speed. Using just 10 flow features, the best performing ML-algorithms are KNN, RF and DT with Bayesian Detection Rates around 97% for the CICIDS2017 dataset. DT and RF had superior classification speeds compared to KNN.

Future work

Future work could try extracting more features from Zeeks other logs and including them in the model. The approach suggested in this thesis could be modified and integrated into the Apache Metron project, using the Apache Kafka messaging queue for streaming the logs and Apache Sparks MLlib for machine learning. Other datasets and deep learning techniques should also be explored.

Appendix A - Abbrevations

- IDS - Intrusion Detection System
- HIDS - Host-based Intrusion Detection System
- NIDS - Network-based Intrusion Detection System
- IPS - Intrusion Prevention System
- SIEM - Security Information and Event Management
- OSI - Open Systems Interconnection
- IP - Internet Protocol
- UDP - User Datagram Protocol
- TCP - Transport Control Protocol
- ICMP - Internet Control Message Protocol
- FTP - File Transfer Protocol
- DNS - Domain Name System
- HTTP - HyperText Transfer Protocol
- HTML - HyperText Markup Language
- SSL - Secure Sockets Layer
- AI - Artificial Intelligence
- ML - Machine Learning
- SVM - Support Vector Machine
- KNN - K-Nearest-Neighbor

- RF - Random Forests
- DT - Decision Trees
- NB - Naive Bayes
- ID3 - Iterative Dichotomiser 3
- QDA - Quadratic Discriminant Analyzis
- TPR - True Positive Rate
- FPR - False Positive Rate
- BDR - Bayesian Detection Rate
- BAT - Bro Analysis Tools
- CIC - Canadian Institute for Cyber Security
- SQL - Structured Query Language
- SQLi - SQL injection
- DoS - Denial of Service
- DDoS - Distributed Denial of Service
- XSS - Cross-Site Scripting
- PCAP - Packet Capture
- FIN - Finish
- OPNids - Open Intrusion Detection System
- MLE - Machine Learning Engine
- GB - Giga Byte
- Mbit/s - Mega Bit Per Second
- GHz - Giga Hertz
- SSD - Solid State Drive
- IOT - Internet Of Things
- IT - Information Technology

List of Figures

3.1 Example of NIDS sensor deployment. Based on the figure in [9, p. 292]. 11

3.2 Zeek:s internal architecture. 14

3.3 Part of a decision tree model for flow-based traffic characterization (this is just an example with mock values). 18

4.1 Overview of the methodology. 20

5.1 Packet loss for Zeek running with and without the custom scripts. 25

5.2 Normalized feature importance scores for the simple script. . . . 26

5.3 The 10 best normalized feature importance scores for the complex script. 26

5.4 Classification accuracies for the attacks. 27

Bibliography

[1] S. Miller, N. Brubaker, D. Kapellmann Zafra, and D. Caban, "TRITON Actor TTP Profile, Custom Attack Tools, Detections, and ATT&CK Mapping," *Fireeye Threat Research Blog*, April 2019, accessed 19-may-2019. [Online]. Available: https://www.fireeye.com/blog/threat-research/2019/04/triton-actor-ttp-profile-custom-attack-tools-detections.html

[2] "Zeek documentation," accessed 6-April-2019. [Online]. Available: https://docs.zeek.org/en/stable/intro/

[3] S. Ali Raza and B. Issac, "Performance of Intrusion Detection Systems and Application of Machine Learning to Snort System," *Future Generation Computer Systems*, vol. 80, pp. 157–170, 2017.

[4] H. Alaidaros and M. Mahmuddin, "Flow-Based approach on Bro Intrusion Detection," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 2-2, pp. 139–145, 2017.

[5] M. Belavagi and B. Muniyal, "Performance Evaluation of Supervised Machine Learning Algorithms for Intrusion Detection," *Procedia Computer Science*, vol. 89, pp. 117–123, 12 2016.

[6] I. Sharafaldin, A. Lashkari Habibi, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," 2017, pp. 108–116.

[7] B. A. Forouzan, *Data Communications and Networking*, 5:th ed. McGraw-Hill, 2013, ISBN:978-0-07-337622-6.

[8] A. H. Lashkari, M. S. I. Mamun, G. Draper Gil, and A. A. Ghorbani, "Characterization of TorTraffic using Time based Features," 2017.

[9] W. Stallings and L. Brown, *Computer Security, Principles and Practice*, 4:th ed. Pearson, 2018, ISBN:1-292-22061-9.

[10] "Nfdump documentation," accessed 6-April-2019. [Online]. Available: https://github.com/phaag/nfdump

[11] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security(TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[12] D. Hedemalm, "An empirical comparison of the market-leading ids's," 2018.

[13] "Logging bro output to kafka," accessed 6-April-2019. [Online]. Available: https://github.com/apache/metron-bro-plugin-kafka

[14] "CIC Datasets," accessed 6-April-2019. [Online]. Available: https://www.unb.ca/cic/datasets/index.html

[15] McAfee Labs, "McAfee Labs Threats Report," March 2016, accessed 6-April-2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2016.pdf

[16] "A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)," 2018, accessed 6-April-2019. [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018/

[17] "NSL-KDD dataset," accessed 6-April-2019. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[18] H. Hadian Jazi, G. Hugo, N. Stakhanova, and G. A. A., "Detecting HTTP-based Application Layer DoS attacks on Web Servers in the presence of sampling," *Computer Networks*, vol. 121, no. C, July 2017.

[19] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 1:st ed. Springer, 2017, ISBN:978-1-4614-7137-0.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[21] "SciKit-learn," accessed 19-May-2019. [Online]. Available: https://scikit-learn.org/stable/index.html

[22] "Apache Spark," accessed 6-April-2019. [Online]. Available: https://spark.apache.org/

[23] "MLlib," accessed 6-April-2019. [Online]. Available: https://spark.apache.org/mllib/

[24] "Weka 3: Machine learning software in java," accessed 6-April-2019. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[25] S. Hossen and A. Janagam, "Analysis of Network Intrusion Detection System with Machine Learning Algorithms (Deep Reinforcement Learning Algorithm)," Master's thesis, Blekinge Institute of Technology, Faculty of Computing, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden, October 2018.

[26] B. Wylie, "Bro Analysis Tools (BAT)," 2017, accessed 6-April-2019. [Online]. Available: https://github.com/SuperCowPowers/bat

[27] S. M. Kerner, "OPNids Integrates Machine Learning Into Open-Source Suricata IDS," *eWeek*, 5 2018, accessed 6-April-2019. [Online]. Available: https://www.eweek.com/security/opnids-integrates-machine-learning-into-open-source-suricata-ids

[28] S. Hall, "HTTP Stalling Detector," 2018. [Online]. Available: https://github.com/corelight/http-stalling-detector

[29] "Zeek documentation, new_packet event," accessed 6-April-2019. [Online]. Available: https://docs.zeek.org/en/stable/scripts/base/bif/event.bif.bro.html#id-new_packet

[30] A. B. Team, "What risks do IoT security issues pose to businesses?" *Avast blog*, February 2019.

[31] McAfee, "What is Stuxnet?" 2019, accessed 22-May-2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/what-is-stuxnet.html

[32] Electricity Information Sharing and Analysis Center (E-ISAC), "Analysis of the Cyber Attack on the Ukrainian Power Grid," March 2016, accessed 22-May-2019. [Online]. Available: https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf

TRITA TRITA-CBH-GRU-2019:033