Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
2024 | LIU-IDA/LITH-EX-A--2024/001--SE

Malware Fingerprinting on encrypted network traffic using Machine Learning

Fingeravtrycksdetektion av skadlig programvara på krypterad nätverkstrafik via maskininlärning

Markus Loborg

Supervisor : David Hasselquist
Examiner : Niklas Carlsson

External supervisor : János Dani

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

In the world of AIs good datasets are important to make the predictions viable. This means that there is a need to either find or create good datasets. That is why this project attempted to create a dataset generator for AIs that need labeled encrypted network traffic. The generator sets up a network and sends traffic over it which is then analyzed. The project tested the generated datasets on 3 different types of AI-models and concluded that the generated datasets were too simplistic. The results were that all models were 100% accurate, had 100% recall and 100% precision. While it sounds really good, in reality it is too good. The generated datasets had characteristics that uniquely identified whether or not the datapoint was malicious or not. While the generator likely could create more complex networks, that is something that future work would need to verify.

Acknowledgments

I would like to thank my supervisors and my examiner for guiding me on how to do this thesis. I would also like to thank my parents for helping me by proofreading my thesis. Finally I would like to thank Sectra for the opportunity to do this thesis.

Contents

Abstract iii

Acknowledgments iv

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Aim . . . 2
- 1.3 Research questions . . . 2
- 1.4 Approach . . . 3
- 1.5 Contributions . . . 3
- 1.6 Delimitations . . . 3

2 Background 4
- 2.1 Intrusion Detection System(IDS) . . . 4
- 2.2 Atomic Red Team (ART) . . . 4
- 2.3 Invoke-AtomicRedTeam (IART) . . . 4
- 2.4 Linux Network Namespace (LNN) . . . 4
- 2.5 Data point . . . 5
- 2.6 Neural Networks (NN) . . . 5
- 2.7 Support Vector Machines (SVM) . . . 5
- 2.8 K-Nearest Neighbour (KNN) . . . 5

3 Related Work 6
- 3.1 Data generator . . . 6
- 3.2 AI Models . . . 6

4 Method 8
- 4.1 Literature Study . . . 8
- 4.2 Code Implementation . . . 8
- 4.3 Evaluation . . . 18

5 Result 19
- 5.1 Data Generator . . . 19
- 5.2 AI-Models . . . 19

6 Discussion 21
- 6.1 Results . . . 21
- 6.2 Method . . . 22

6.3 Sources 22
6.4 The work in a wider context 22

7 Conclusion **23**

A XML Schema **24**
A.1 Data types 24
A.2 schema 26

B Computer specifications **28**

Bibliography **29**

List of Figures

4.1 Process for data generator 9
4.2 General Configuration 11
4.3 Implemented network 12
4.4 Trafic Configuration 13
4.5 Attack Configuration 15
4.6 Model training 17

List of Tables

5.1 The results for all models 19
5.2 Metadata about the datasets 20
5.3 Padded packets for attack and normal flows 20

B.1 Specifications for the computers 28

1 Introduction

Computers have become a thing that is used daily by most people. They have gone from being something used by researchers and universities to something the common man can use. While access to computers have increased the knowledge to protect the device from malicious people has not spread as fast. With the advancement of the internet our devices get more accessible to malicious people and many users do not know how to protect the device from these attacks. The attacks have also grown in complexity. They started as simple cryptic messages that would pop up after entering the infected disc into your computer[25]. Now with the internet malware comes from many sources such as malicious emails, ads, malicious websites disguising as legitimate ones and hackers attempting to get access by directly communicating with the device. So to protect against the increasing threats the companies that sell that protection have had to evolve that protection to combat the new attacks. One of the ways to protect the integrity of the data is to encrypt it. Encrypting the data makes it much harder for anyone to know what the data means even if they see it, so no matter if the attackers are just listening in on some of the network or stealing the data away from the network they will not be able to understand what the data means.

While encryption seems to be a beneficial protection mechanism it will also make it harder to identify any malware before that malware reaches the target device. It would be harder because when the antivirus programs attempt to check if the data is malware or not it would not be able to determine that since the data is encrypted. While it might be decrypted and therefore detectable when it reaches the device, it might be too late if it was a malware since the malware is then already on the computer. So with the advancement in malwares along with most of the network traffic using encryption, advanced methods are needed to detect the malwares.

A common detection method that has been used since early on is fingerprinting. In the same way that human fingerprints are unique the same can be said for a malware. While it started with static analysis of features or signatures e.g. matching a specific string of data, that is no longer enough. The fingerprinting methods had to evolve since the attackers improved the malwares to avoid these detection mechanisms. One way to improve the malwares is by using polymorphism which means the code will modify itself to look different but function the same way as it did before[2].

Another way the malware got harder to fingerprint is through encryption. Without encryption specific patterns that existed in certain malwares could be searched for to determine

if the data is a malware or not. With encryption that pattern becomes harder to detect. Some encryption methods would just convert that pattern into a new pattern which is easy enough to detect by just searching for the new pattern but other encryption methods would convert that pattern into a different pattern every single time depending on a number of factors. These factors include surrounding text, which devices the data is transmitted in between and when the transfer happens. This would then make it impossible to search for that pattern since you do not know what to search for. This has led to more advanced methods of fingerprinting such as finding patterns in the behavior of the network traffic. Behavior in this context being how the traffic flows to and from a device.[29]

1.1 Motivation

Malware fingerprinting looks in the network data as well as the files on a device for patterns that are commonly used by malwares. However with encryption that network data is obscured which makes it harder to find the patterns. While you can decrypt the data to run these detection programs, you might not have the necessary permissions to do so. One such situation is when cloud services are used, where the client is storing their data on someone else's server. To protect themselves the client application encrypts their data in an attempt to ensure that only they can access the data. Then the client pays someone to monitor their networks for malware. To do that the monitoring software needs to be able to identify whether or not the data is a malware. This is done in many different ways. One way is that the detection program is allowed to decrypt the data to check if it is safe and then encrypt it again before it sends it onwards. This approach however creates vulnerabilities since attacks on the detection program could result in the data being leaked. It also makes it cost more as well as slows down the monitoring process since it takes more time and resources to decrypt and re-encrypt the data.

Another approach is to attempt to analyze the encrypted data and attempt to detect malware without ever decrypting the data. While better in some ways since there is no worries about decrypted leaks or cost of decrypting and re-encrypting there are other problems. One of the biggest issues is that encryption is meant to obfuscate the data that is being transmitted to the point where an observer should not be able to draw any conclusions about the data. An observer drawing conclusions is exactly what the monitoring program would be, which means that encryption would prevent the monitoring program from doing its job.

To avoid that problem the monitoring program analyzes the metadata about the encrypted data instead of the actual data. The metadata includes things like where is the data going, where did it come from, when was it sent, how much data is it, how does the data flow etc. All these things are used to identify good and bad network data, almost like a fingerprint is used to identify a human, hence the term fingerprinting. Training an AI model on this kind of data allows it to see patterns that can be used to indicate if the data is good or bad.

1.2 Aim

This thesis aims to develop a dataset generator that can be used to create datasets for the training of AI-models to detect malware fingerprints in encrypted network traffic.

1.3 Research questions

During this thesis the following research questions will be answered:

1. Can the dataset generator be relied upon to create datasets that represent real-world situations well enough so that a model trained on said datasets is viable as a detection tool?

2. Which of the tested types of ai-models performs best in regards to accuracy, false negatives and false positives on the generated datasets?

1.4 Approach

The project is structured as follows: It starts with a literature study of past projects and studies on the subject matter. Then it proceeds by creating the dataset generator. Once the generator is created, it is used to generate datasets that are then used to train the AI-models. These models are Neural networks,Support Vector Machines(SVMs) and Nearest Neighbour algorithms. Then the models are compared to find the answer to research question two. The models will be compared on accuracy, recall, precision, and F1 score. The answer to question one will be found by comparing the results of the models with other literature.

1.5 Contributions

- Dataset generator producing datasets of normal and malicious network traffic.

1.6 Delimitations

This thesis is limited in the datasets and what they include. The datasets will only have https,ssh and http traffic. Another limitation is in the traffic simulation. The traffic scripts will use frequency and duration to decide when to send traffic. This does limit how close the network can represent real-world situations but more complex modeling would take too much time to implement.

2 Background

2.1 Intrusion Detection System(IDS)

An Intrusion Detection System is a system that exists on most networks. It attempts to detect intrusions in the network. How it does this varies a bit from system to system but commonly it looks at logs and network traffic and uses certain algorithms to determine if attacks are occuring. It might just log the attack in its own log file. This is common in default installations of personal firewalls. It could also create an alert that is sent to a security analyst for further examination, which is common in company systems. [19]

2.2 Atomic Red Team (ART)

Atomic Red Team is a python library of attacks used to test the security of a device.[3] The attacks are based on the Mitre Att&ck framework[20]. This is normally used to test personal devices, networks and intrusion detection systems. In our case we use the attacks to generate the attack traffic for the data generator. The library contains scripts that cover many types of attacks such as escaping out of containers, exfiltrating data to other devices, escalating user privilege levels etc. For each of these tests there exists code to execute it on multiple different types of systems and sometimes there exists multiple versions of the same attack for some of the systems.

2.3 Invoke-AtomicRedTeam (IART)

Invoke-AtomicRedTeam is a python library that is used to wrap the functionality that ART provides into easy-to-use methods.[14] For remote attacks (as in attacks over a network) this library only allows for ssh connections which limits its usage. This library was used to make the usage of ART easier as well as allowing one single script to call any and all ATR scripts.

2.4 Linux Network Namespace (LNN)

Linux Network Namespaces are virtual network stacks that exist on a single device. They work by separating and isolating certain system resources which are used when commu-

nicating over a network. This allows each namespace to behave like its own independent device on a virtual network. Thus allowing a user to create multiple "devices"(namespaces) that can interact with each other on a virtual network even though they all exist on a singular host. This is used because of how easy it is to programatically set up and control via scripts as well as the fact that it comes with default installation of linux.[17]

2.5 Data point

A datapoint is a n-dimensional array of numbers. Each number represents a particular feature that the datapoint has, for example which hour a message was sent can be a feature. Another feature might be the length of the message. All features are converted into a discrete set of numbers. This means some features like hour-of-day cover all possible options, while others like length of message will be discretized into groups like short, middle and long. The length of messages could also be divided into 0-1000 bytes, 1000-5000, 5000-10000, 10000-50000 and 50000+. All of these choices are made by the creator of the model and will change from model to model depending on what is needed and important.

2.6 Neural Networks (NN)

Neural Networks are AI-models whose functionality is inspired by how biological neurons work in the brain. In the same way that synapses get stronger between neurons that have correlated output, the nodes (which is the NNs representation of a neuron) have connections in between each other that are strengthened or weakened during training depending on how well they combine to get a desired output.

The connections between the nodes get strengthened or weakened based on two values that exist in each node.These values are a weight and a threshold. The threshold is used to decide if the node should send data to the next layer or not. The weight is a multiplier that either amplifies or weakens the signal that is sent from that node.

These values are calculated when training the model by testing different values on the nodes and seeing if the outcome becomes what was desired or not. This is the same way that the human brain learns how to do things like standing,walking etc[7].

The NNs have several layers of nodes. These layers are divided as follows: an input layer, one or more hidden layers and an output layer. In the brain that division would be sensory input, the brain deciding what to do and finally the resulting action.

2.7 Support Vector Machines (SVM)

Support Vector Machines are a type of model that attempts to divide the data into two parts. It does this by first creating a n-dimensional space where n is dependent on the amount of features the data points has. Then it maps all data points into that space. Then using algebraic equations it divides the two parts of the data by calculating a hyperplane that is as far away from any data point as possible while still separating the classes on either side of the plane as best it can.[4]

2.8 K-Nearest Neighbour (KNN)

K-Nearest Neighbour is a type of classifier that uses proximity to separate data into classes. It assumes that data points in the same class would be close to each other. To check the class of a new datapoint it calculates the distance between that point and the existing points. Then it looks at the k closest points and whichever class occurs most in those points will be the class that the new datapoint get.[27]

3 Related Work

3.1 Data generator

New datasets will always be needed since the attacks and the normal traffic changes constantly. This means that datasets that are a few years old can be hard to use because they will lack data about new protocols and attacks. It can also make the models trained on the old datasets unable to detect the new types of attacks. They can also have other deficiencies that have been discovered since they were created[21]. This means that methods to generate datasets are needed, but there are only a few studies about such generators. Instead most studies revolve around creating a model for a specific problem and a singular dataset that can be used to test that model. Some of them do note that the dataset could be used as a benchmark test for other models but only a few talk about methods to actually generate datasets[8][11].

One of those studies is Greff et.al.[13] who created a dataset generation method for images. While that study was about images and cannot be directly applied to this situation we can still draw some information about the methodology used to create the dataset. Greff used a pool of workers that each generated a single input at a time. That means that the work is divided into many small tasks that can be done independently of each other. Each worker takes the next task whenever they finish their current work. This is similar to how we generate the network traffic. The generation works by using a pool of threads that run scripts where each script is responsible for creating one or more data flows in between two devices at a time.

3.2 AI Models

When it comes to AI models the amount of studies are vast ranging from general studies on techniques to specific studies on which model is best for any given purpose. The purpose ranges from identifying patterns in medical data[1], guessing what word comes next(Chat GPT)[6] to creating the logic that rules how game characters will move and attack.

One study about models that can analyze network traffic and detect malwares is "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things"[18]. This study uses a Neural network as its base and then looks at how applying different techniques to that base model changes how well it performs its detection.

Some other studies take a biological approach such as the model in "Malware Analysis and Attribution using Genetic Information" (MAAGI).[23] MAAGI uses a slew of different AI techniques to give a comprehensive analysis of malwares. It operates on the assumption that malwares behave similar to biological processes and therefore one could use biologically inspired methods to detect and analyze malwares.

Another neural network study is the one by Prasse et.al. [24] where they used a type of NN to detect malware in encrypted traffic. Their data however was taken from a general network and was labeled based on what the antivirus programs detected after the fact. This kind of approach to generating a dataset has its own pros and cons. A benefit of that way is that you do not have to initiate every attack but let the malwares act naturally which gives realistic data. A con however is that you are totally reliant on the fact that the antivirus will detect the malwares to label them. If some malware bypasses that detection then the training data will be skewed and incorrect which in turn might affect the AI to predict incorrectly.

4 Method

The thesis is split into 3 parts: a literature study, code implementation and evaluation. The pre-study was done to find relevant AI techniques that could be used to create models that can test the data generator. The implementation consisted of two parts, creation of the data generator and implementing the AI-models. The results of the AI-models were then compared and evaluated against each other as well as the scores that the models achieved in other literature.

4.1 Literature Study

The literature study was done to find out about relevant studies as well as to focus the project by answering questions relating to what method should be used. One question was if all datasets should be self created or if there existed public datasets that could be used for the thesis's purposes. Another was regarding which AI-models to use. To answer these questions google scholar[12] and kaggle[15] was used. Google scholar was used to find other studies by searching for the terms "network traffic AI","network traffic malware" and "AI for malware detection in network traffic". Kaggle was used to look for existing datasets. Search terms there were "network traffic" and "malware in network traffic". From these scholar searches any result that came from reputable sources was viewed and read to see if they could be of use either as inspiration or related to this study in any way. The kaggle searches sadly did not come up with anything that could be used since the datasets were either unclassified or contained the wrong type of traffic.

4.2 Code Implementation

The generator and models were implemented in python. First the data generator was coded. Then once the generator started producing data the models were implemented. After that the models were trained with the generated data to produce the results that can be seen in chapter 5.

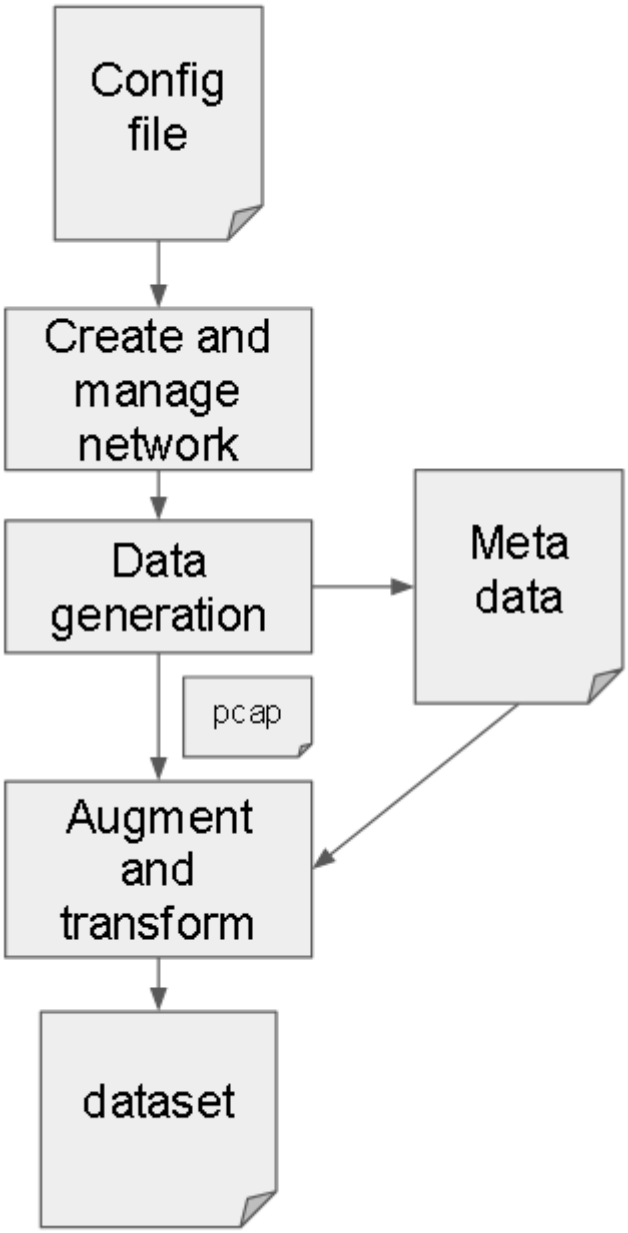


Figure 4.1: Process for data generator

The steps to generate a dataset where actual files are shown with a folded corner.

4.2.1 Data Generation

The generator was divided into several parts. These parts are: reading configuration files, creation and management of the simulated network, generation of data and converting data into dataset files.

The generator works in the following way. Upon starting the program input parameters are selected. These parameters tell the program what system it is running on, which con-

figuration file should be used, what type of format the output data should have and specify several other options. These other (optional) options tell the program to use an existing model to predict the outcome of a dataset among other things. All parameters and their purpose can be found at the github[9].

With these options the program will then validate and read the configuration file. Using the configuration the management part will take over and create the network and start generating the traffic. These are the "create and manage" as well as "data generation" steps in figure 4.1. The traffic is generated by a set of scripts that run on the devices in the network and all traffic is recorded in pcap files. Once the generation is over the program will start the conversion part. This conversion will take the pcap files and convert them into a single output file of the correct format. This is the augment and training data steps in figure 4.1

Configuration file

The configuration file format was incrementally developed, to support the type of networks we were interested in. A XML schema was used to ensure that the configuration files will be in the proper format. The file has to contain a set of general settings for the run. It also supports assigning individual settings for devices. The file also contains information about what traffic scripts and attack scripts are to be used and by which device. One of the config files used during the data generation is shown in the image below. The schema and configuration files used in this work can be found in the code repository[10].

```
<root>
    <general>
        <max_groups>3</max_groups>
        <max_depth>3</max_depth>
        <max_duration>480</max_duration>
        <duration_type>MIN</duration_type>
        <name>AUTO</name>
        <amount>3</amount>
    </general>
    <individual>
        <vm>
            <name>server</name>
            <group>/test1</group>
        </vm>
        <vm>
        <name>saya</name>
            <group>/test1</group>
        </vm>
        <vm>
        <name>kalle</name>
            <group>/test1</group>
        </vm>
    </individual>
    <traffic...>
    <attacks...>
</root>
```

Figure 4.2: General Configuration

One of the configurations used with focus on the network setup. We create 3 devices and then name them and allocate them to a subnet.

The groups refer to different sub networks in the created network. The max_depth is referring to how many layers of groups there can be. If the individual tag does not specify a group the device ends up under the root group. The duration is the maximum duration the network will be up and running generating data. The amount is the amount of devices and the name refers to if each device has to specify names or if they can be generated. The actual network that was used for all runs is shown in figure 4.3

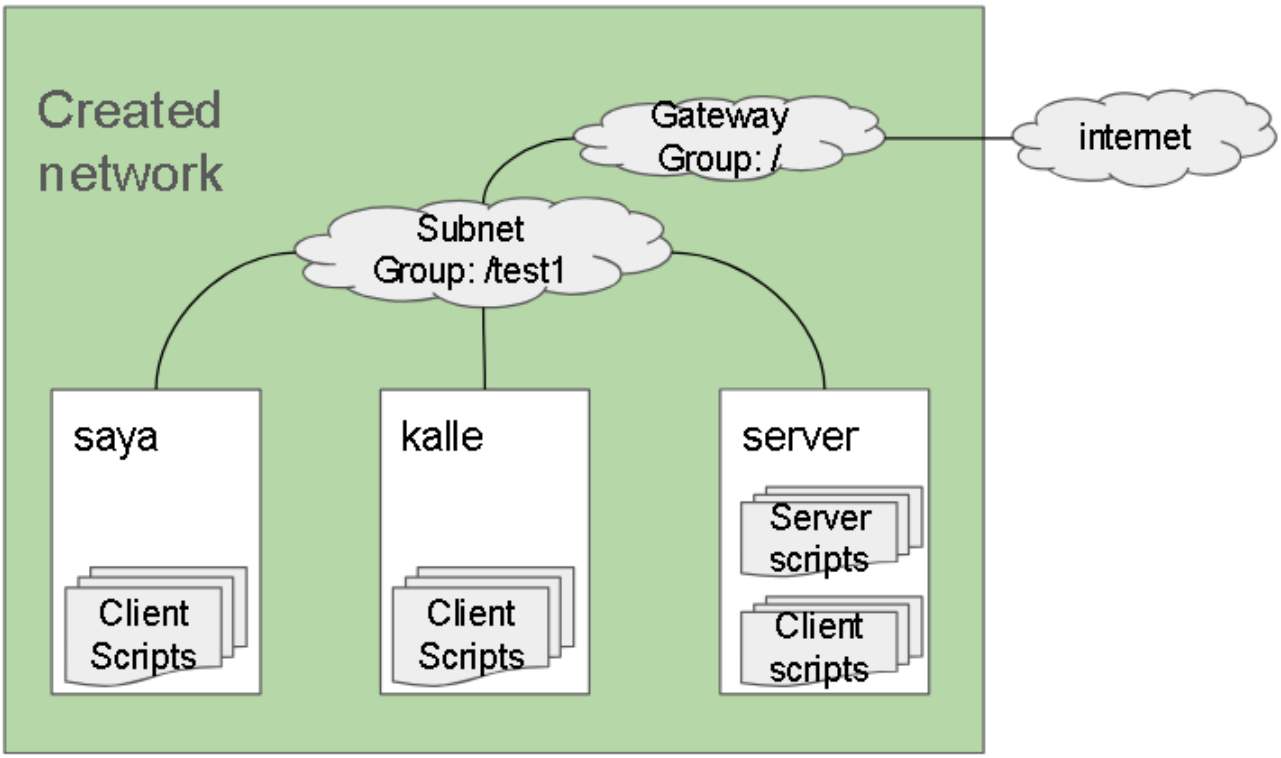


Figure 4.3: Implemented network

The network that was created. The scripts scheduled for execution on each device are shown as three stacked pages. The devices are the white boxes and clouds represent a network address space. The subnet and gateway are virtualized network bridges but they take the place of their respective name. The green box is everything that was created and therefore can be controlled or changed. It was created on a single physical machine.

4.2.2 Managers

The managers are a set of python classes with a common manager interface that are responsible for everything that is platform related. Each manager class is responsible for one particular platform e.g. one manager is responsible for windows, one for virtualbox windows, one for virtualbox linux and a fourth for normal linux etc. This is due to the fact that each system will have different ways to create and interact with devices. So each manager has methods to create devices and start scripts on them as well as any help functions needed to do that on that particular system.

4.2.3 Traffic Generation

The traffic generation are done via a script that sends normal traffic over the network. These scripts are divided into two main categories with several subcategories. The main categories are external and internal. External is traffic to sites outside the network such as google, stackoverflow etc while internal is traffic to internal servers or other devices on the network. The subcategories are the different types of communication such as ssh and https. Figure 4.3 depicts the network that was created and used for all datasets. Three devices acted as clients,

one of which doubled as a server. All of them sent traffic to each other and generated traffic by fetching websites from the internet.

In the configuration file there is information about each traffic script that is to be used. This information includes which of the created devices the script should be run on. That device will be the origin of the traffic and is aptly named origin in the configuration. If the device is not specified the script runs on all devices. It also contains information about how long the script should run for, how often it should run and how many times it should run. It might also include a target depending on the type of script. The target will be one of the other devices in the network so it is only used in internal scripts. If no target is given, either its an external script or the script has a built in target.

The information will also include any potential extra arguments that the script needs such as a specific external site, a port number etc. The configuration for one of the scripts that were used can be seen in figure 4.4.

```
<traffic>
    <traffic_instance>
        <script>https/external/random_traffic.py</script>
        <model>
        <frequency>15</frequency>
        <frequency_type>hour</frequency_type>
        <duration>1</duration>
        <duration_type>min</duration_type>
        </model>
        <params>
            <name>site</name>
            <value>https://stackoverflow.com</value>
        </params>
        <params>
            <name>depth</name>
            <value>3</value>
        </params>
        <params>
            <name>source</name>
            <value>ORIGIN</value>
        </params>
    </traffic_instance>
    <traffic_instance...>
    <traffic_instance...>
    <traffic_instance...>
    <traffic_instance...>
</traffic>
```

Figure 4.4: Trafic Configuration

One configuration of a script that runs 15 times an hour for 1 minute each time. It has three parameters that can be modified and the full format for parameters can be found in appendix A. In this example it will fetch stackoverflow.com and then follow links from that page to other pages and repeat the process a total of 3 times.

The generated traffic includes https traffic to a few semi-randomized external websites. The script either randomly chooses a few sites from a list or are given the starting sites as a parameter. It will then follow links from these sites to get to new sites. This process is

repeated n times depending on the parameter depth. That parameter was selected in the configuration file and was chosen to be similar to how a worker might scroll around on a break. The traffic also included fetching json data from a webserver about some charts. The charts were randomized in type and content using its own script.

4.2.4 Attack Generation

The attack scripts are the scripts that are used to send malicious commands, which is the malicious traffic. They have the same format as the traffic scripts in the configuration file with the exception that target and origin are obligatory. They are not divided into external and internal scripts, instead they are usually divided based on type of attack such as ddos, xss etc. The reason for this is that no external attacks were used. All attacks came from one of the devices we created and controlled. In some cases the division into type of attack does not work, for example certain libraries of attacks such as ART can not be divided in the code since IART assumes all ART scripts are located in the same folder and will only look in that folder to execute ART scripts. One configuration of an attack script that was used can be seen in figure 4.5.

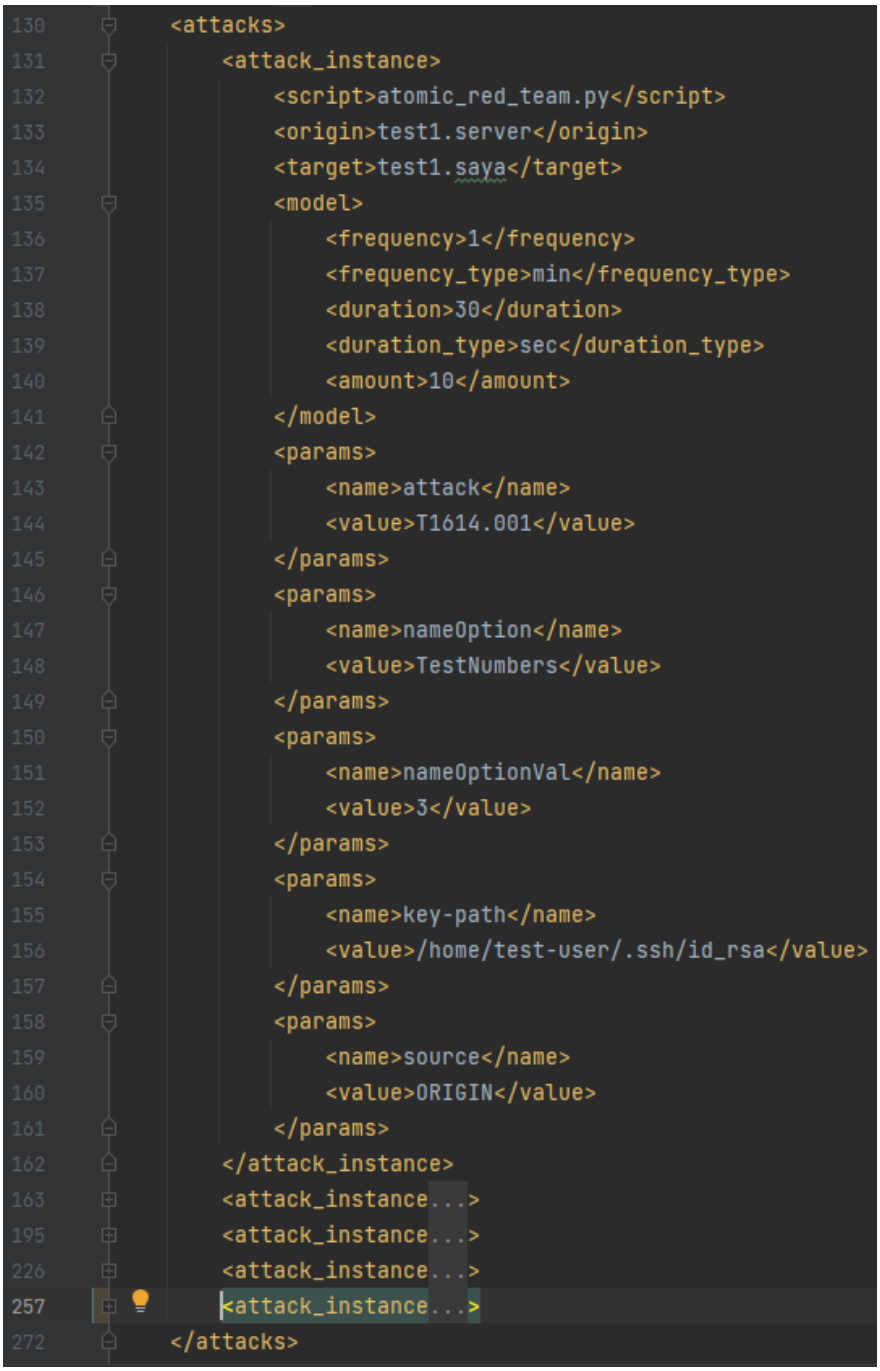


Figure 4.5: Attack Configuration

A configuration of an attack script that uses an ART attack with a predetermined ssh key. It runs once every minute for 30 seconds. It will run a total of ten times.

The scripts used were a few of the attacks that exist in the ART-library. The attacks were triggered over an ssh-connection to the targeted device. The ART scripts used were selected due to them working on the created network and that they extracted data from the device and sent it back over the network. Which ones exactly can be found in the config file which can be found on github[10]. In addition to ART some of the scripts also used basic DoS attacks. The

DoS attack used a simple script found on github [28] which was modified to suit the project's needs.

4.2.5 Data Transformers

The transformers are a set of classes that are responsible for taking multiple pcap and meta-data files and converting them into a single file that makes up a dataset. Each transformer transforms the files into one format, to match the software used in training the models. So one transformer will convert the data into a csv file while another might convert it into a yaml file. These classes read the metadata as well as the pcap files and then create flows from that data, one flow per script, while also removing duplicated data. The flows are then written into a new file in the wanted format. Currently only a single transformer exists and it transforms the generated data into a csv file.

The process works as follows:

- Each device generates a pcap file that contains all incoming and outgoing traffic for that device. The meta-data is created as the scripts run to keep track of all data. Each script that runs creates its own meta-data file. The file contains lines in the following format:

  ```
  S_IP:S_PORT, T_IP:T_PORT, START_TIMESTAMP, END_TIMESTAMP,
  IS_ATTACK
  ```

  S stands for source and T stands for target.

- The manager coalesces all the meta-data files into a single one removing duplicated data. The duplicates can occur due to some lines' start and end times being subsets of or overlapping other lines. This occurs due to the meta-data partly being collected by querying the device about processes that are running. Each script does this which means they can detect each other so a round of removing duplicate data is necessary.

- The transformer then takes the pcap files and the coalesced metadata file and goes through each pcap entry and storing it in the following format:

  ```
  S_IP, S_PORT, T_IP, T_PORT, TRAFFIC_TYPE, START_TIMESTAMP,
  END_TIMESTAMP, NR_OF_PACKETS, TOTAL_SIZE, IS_ATTACK, LIST_OF_PACKETS
  ```

The total size is in bytes and the list of packets contain 20 elements, each containing the timestamp,direction and the size of that packet. The 20 packets are the first 20 in that flow. If more packets exist they are truncated and if less exist it is padded with zero sized packets.

4.2.6 Model Training

The model training follow the general format of:

1. read dataset files
2. optimize hyper parameters on the first dataset
3. train on the second dataset
4. evaluate on the third dataset
5. save model into file for later usage.

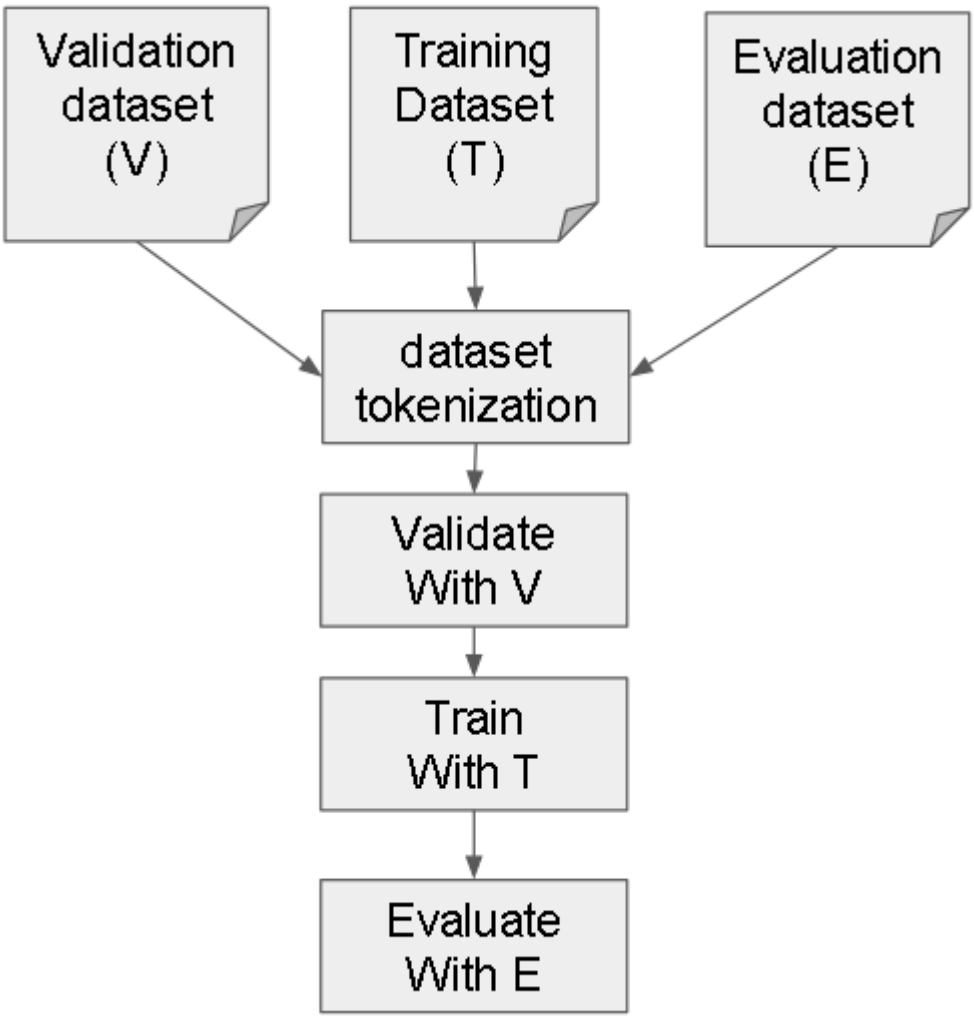


Figure 4.6: Model training

General flow of how the model training is done in discrete steps. Folded corners indicate actual files. Tokenization occurs for all datasets where the biggest gets tokenized first and then that tokenization is used on the other two.

The reading of files is dependent on the format of the file but generally it parses the file back into objects that can be used to train on. The validation step, also referred to as the optimisation step) is training the model multiple times on a small dataset where the hyper-parameters of the model are changed each run. Then the parameter values that gave the best result are chosen and used for the actual training of the model. The save function is also taken directly from the library. The training and evaluation steps are more specific to each type of model.

Neural Network

A specific neural network was used. This NN was taken from "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things"[18]. The NN was implemented with the scikit-learn library[26] and used 8 layers as described in the original article[18]. With that the training and optimizing was also predetermined since we had to

use the same optimization and training as the original article to compare the two results. The training was done with an ADAM optimizer with a 0.02 learning rate and 80 epochs. The optimization was a grid search. This model was chosen due to being made for identifying malware and because it had the best result in a study where many models were compared on their capability to identify malwares.

Support Vector Machines

The Support Vector Machine was a default version taken from the scikit-learn library[26]. The optimization was a gridsearch over the parameters *C*, *kernel* and *gamma* using scikits built in methods. For *C* values **0.1**, **1** and **10** were tested. For *Kernel* the values **linear** and **rbf** were tested. For *gamma* the values **scale** and **auto** were tested.

K-Nearest Neighbor

The Nearest Neighbor was a default version taken from the scikit-learn library. The optimization used was a cross validation of **1**, **3**, **5**, **7** and **9** neighbors. Once again scikits built-in methods for this type of optimization was used.

4.3 Evaluation

The evaluation of the models was mostly done via the classification_report function from the scikit-learn library. This is a function that takes the model along with a dataset of labeled data and outputs a table containing the accuracy, precision, recall and f1 score for the model. The only exception was the neural model. The reason for that is that it used the same evaluation as the paper it was from originally which, while calculated in a slightly different way, still outputs the same scores. This is why it can still be compared to the other models.

5 Result

5.1 Data Generator

The generator generated about 85000 flows in around 7 hours on a high end desktop computer while it took 48 hours to generate the same amount on a laptop with lower specifications. The exact specification can be found in Appendix B

5.2 AI-Models

The 3 AI-models all got the same result. That result is a 100 percent accuracy, f1, precision and recall which can be seen in table 5.1. The support column denotes the amount of flows that the model used to train on for one of the runs and that is the only value that changes between different runs of the models. In table 5.2 the average bytes/flow for attack and normal traffic are shown, as well as the biggest and smallest package in the different kinds of flows. In table 5.3 we show the amount of padded packages that exist in each flow for both attacks and normal traffic. The models training time varied slightly in between each model but on average it took around 3 hours.

Table 5.1: The results for all models

	precision	recall	f1-score	support
Traffic	1.00	1.00	1.00	80266
Attack	1.00	1.00	1.00	80432
accuracy			1.00	160698
macro avg	1.00	1.00	1.00	160698
weighted avg	1.00	1.00	1.00	160698

Table 5.2: Metadata about the datasets

	total bytes	flow amount	average bytes	min bytes	max bytes
attack	613693896	14601	43645.11030509921	0	334936
normal	6102145300	180985	33716.30411360058	77	16015370

Table 5.3: Padded packets for attack and normal flows

# of padded frames	attack	normal
0	2619	56643
1	0	1711
2	0	4015
3	0	3295
4	0	965
5	0	670
6	0	233
7	0	41
8	0	865
9	0	993
10	0	4691
11	3	0
12	0	5761
13	0	14
14	0	10671
15	0	20
16	0	42845
17	0	58
18	0	47444
19	0	38

6 Discussion

6.1 Results

As seen in the results our models seem perfect with 100 percent accuracy. However if you see table 5.3 you clearly see that the attacks had very distinctive patterns that were easily identified. That is not the case in actual traffic, which can be seen by the fact that the models were only around 96% accurate[18] in general literature as well as the literature from where the neural network model came from.

So the results are not a correct representation of the real situation. But that is not a fault of the model since it was taken from another study and that study used real network traffic and got a more reasonable result as mentioned above. Therefore the problem has to lie in the dataset. If the models were trained on a better dataset they would likely have given more accurate results. But this begs the question why the dataset was bad.

The reason the dataset was bad lies in the fact that the traffic scripts used could not accurately represent a real situation. The normal traffic scripts were fine since they were just accessing normal sites thereby producing realistic traffic.

The problem lies in the attack scripts. Most of them came from the ART library of attacks which works fine[22][5][16] but due to limitations in how the network was set up all ART attacks were done in an already opened ssh channel with administrator privilege. This limits the amount of attacks that can be done since many of them revolve around getting that access. Another set of attacks could not be used since they either exist for situations that the network did not support like escaping docker containers or were for another operating system then what was used on the devices in the network.

The only attacks that did not come from the ART library was a simple DDoS attack. This resulted in attacks that did not mimic any normal traffic and therefore was extremely easy to detect. While that is true for any DDoS attack no matter what network it is run on it still shows the issue of too few attacks that were complex enough to be difficult to detect. While it is likely that a more advanced network with more options and more sophisticated attacks could have created a better dataset, it was not something that could be tested in this project due to time constraints along with bad planning.

6.2 Method

As mentioned there were issues with the generation of attack data. While that is the case the method in of itself does not have a problem since the study it was taken from produced reasonable results. If there is no issue in the theoretical method but the result was bad then it seems logical that the fault lied in the implementation.

Then why did it get implemented this way? It was implemented this way for several reasons. Among these reasons are the fact that the time limit only allowed for the use of existing libraries of attacks instead of implementing attacks ourselves. It was also assumed that it would save time that could be put to better use on other parts of the project.

Another reason was that several issues delayed the time it took to start analyzing the data. These issues included among others a rework of the generation process since the first version failed to work. There was also a problem with file sizes that grew too large which resulted in the program crashing and due to the implementation lost the data that had been generated. There was also a power outage during a weekend causing the loss of 4 days of generated traffic. All these issues led to the discovery of the fact that the dataset was too simple too late to change or add any attacks.

A third was a lack of technical know-how of how to quickly implement these types of attacks on a virtual Machine that did not have any obvious weaknesses. So to implement these attacks one would also have to implement the weakness in the VMs. This was something we did not know how to do or have the time to learn and implement. So the network that was created was too good for the attacks that were available.

In retrospect some critical thinking could have made this more obvious since the literature study did hint toward this or at the least serious problems with creating data generators since there existed few examples and sources of data generators of this type.

Now could this have been fixed, yes if it was found early enough so that there was enough time to implement attacks and weaknesses of different types to try and confuse the AI-models more. This does make this study very unreliable as a better implementation would probably give a different result.

6.3 Sources

The sources chosen are mostly from scientific articles. The few sources that are not scientific articles are either from the official site for the project or library that is being referenced, or from reputable businesses like IBM where if they spread misinformation it would get noticed and questioned and then taken down.

6.4 The work in a wider context

The study was supposed to be helpful to society in the fact that it would allow people to use the generator to generate datasets of their own that they could use in any way they wanted. This would also allow for ethical gathering of the data since one would not be gathering data from someone else's network activity nor would you use attack data that existed due to other people being attacked. Now it can only act as a ground to build better generators from.

6.4.1 Future work

To improve upon this generator several things could be done. There are capabilities in the configuration that are untested but could allow for better datasets. These could be tested. One could also add the creation of devices that are better suited for this type of generation, such as proper vms that can have accounts and weaknesses to allow a larger range of attacks to be carried out.

7 Conclusion

The major conclusion that can be drawn from this is that it requires a lot of resources to create a network that would be complex enough to generate data for the purpose of training Ai:s on malware detection. Existing configuration files lack complexity to create a dataset that could train a usable AI so it can not be confirmed that this method could be used to generate viable datasets.

For the models they all performed the same since they all got 100% accuracy and 0 false outcomes. Comparing that to the literature would not be viable though since it has been proven that the training data was insufficient to properly train the AI-models to represent a real-world scenario.

A XML Schema

A.1 Data types

A.1.1 time_type_opt

time_type_opt is a string that represents the unit of the frequency and duration values. It can take one of the following values:

- SEC
- MIN
- HOUR
- DAY
- WEEK
- MONTH
- YEAR

A.1.2 group_type

A group_type is a simple string with a pattern restriction on it. The string needs to follow this pattern: [A-Za-z0-9/]*

A.1.3 run_model

run_model describes how a script should run and includes frequency, duration if it should run a set amount and if it should always run. It contains the following fields in order:

- frequency (integer) - how often a script should run(in units of frequency_type)
- frequency_type (time_type_opt)

- duration (integer) - The max duration (in units of duration_type) a script runs from start to finish. If the script is shorter it will only run as long as it needs but if it is longer or could be infinite then it will only run this long.
- duration_type (a time_type_opt)
- amount (optional integer) - Specifies how many times a script should run. If not present the script will run multiple times (dependent on frequency) until the generation stops.
- perpetual (optional element) - If present duration is ignored and each instance of the script will run until generation stops.

A.1.4 vm_model

vm_model is a model over a singular vm that specifies its settings. It can have the fields in order:

- name (optional string) - The name of the device.
- ram (optional integer) - How much ram the device should have. Currently unused and exists as a relic from attempts at a VirtualBox Manager.
- cpus (optional integer) - The amount of cpus the device should have. Currently unused and exists as a relic from attempts at a VirtualBox Manager.
- os (optional string) - The operating system the device should run on. Currently unused and exists as a relic from attempts at a VirtualBox Manager.
- storage (optional integer) The harddrive capacity of the device. Currently unused and exists as a relic from attempts at a VirtualBox Manager.
- group (optional group_type) - Which subnet the device belongs to.

A.1.5 param_type

A param_type is a simple element that has two fields. A name and a value.

- name (string) - The name of the parameter.
- value (string) - The value of the parameter.

A.1.6 traffic_type

traffic_type is the data type that describes a traffic_instance. It contains information about how a traffic script should be run. It has the following fields in order:

- origin (optional string) - The device the script it executed from. (If omitted the script is executed on every device in the network.)
- target (optional string) - The device the script is targeting. (If omitted the script targets should be implicit in the script itself.)
- script (string) - The name of the script that should be run. (Has to be a relative path from the traffic_scripts folder.)
- model (run_model)
- params (optional param_type) (The field can have an infinite amount of copies after each other.)

A.1.7 attack_type

The attack_type is the data type that describes an attack_instance. It contains information about how an attack script should run. It is almost identical to the traffic_type with the exception that origin and target are obligatory.

A.2 schema

A.2.1 /root

/root is the root element in the schema and contains the following sequence of elements in the given order:

- defaults (optional element) - Default values for vms or physical machines. See A.2.2
- general (mandatory element) - See A.2.3.
- individual (optional element) - Sequence of vm_models that specify a device separately.
- traffic (optional element) - Sequence of elements called traffic_instance that are of the type traffic_type.
- attacks (optional element) - Sequence of elements called attack_instance that are of the type attack_type.

A.2.2 /root/defaults

This is an obsolete section. Exists due to an old attempt of a VirtualBox manager. The defaults element contains the following fields:

- audio-controller (string) - The type of audio-controller the device should have. (obsolete, exists due to old attempt of a VirtualBox manager)
- graphics-controller (string) - Type of graphics controller the device should have. (obsolete, exists due to old attempt of a VirtualBox manager)
- ram (optional integer) - How much ram the device should have. (obsolete, exists due to old attempt of a VirtualBox manager)
- cpus (optional integer) - The amount of cpus the device should have. (obsolete, exists due to old attempt of a VirtualBox manager)
- os (optional string) - The operating system the device should run. (obsolete, exists due to old attempt of a VirtualBox manager)
- storage (optional integer) - The harddrive capacity of the device. (obsolete, exists due to old attempt of a VirtualBox manager)

A.2.3 /root/general

The general element contains the following fields in given order:

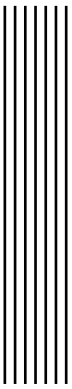
- max_groups (integer) - The max amount of groups that are allowed to exist under any other group.
- max_depth (integer) - The max amount of groups that can
- max_duration (integer) The maximum time spent on gathering data.
- duration_type (time_type_opt) - The unit of the max_duration value.

- speed_multiplier (integer) - How much faster the network should produce data in comparison to the real world. (currently not functioning)
- name (enum) - Two options, AUTO or INDIVIDUAL, representing if the name of each device has to be manually entered in the configuration file or automatically created during runtime.

B Computer specifications

Table B.1: Specifications for the computers

	High-end desktop	standard office laptop
CPU	Intel(R) Xeon(R) W-2125 @ 4.0 Ghz	Intel(R) Core(TM) i7-1255u @1700 Mhz
RAM	32 GB	16 GB
GPU	Nvidia Quadro RTX 4000	Intel(R) Iris(R) Xe Graphics
Disk	240 GB	480 GB



Bibliography

[1] Dongmei Ai, Yuduo Wang, Xiaoxin Li, and Hongfei Pan. "Colorectal Cancer Prediction Based on Weighted Gene Co-Expression Network Analysis and Variational Auto-Encoder". In: *Biomolecules* 10.9 (2020). ISSN: 2218-273X. DOI: 10.3390/biom10091207. URL: https://www.mdpi.com/2218-273X/10/9/1207.

[2] Tibra Alsmadi and Nour Alqudah. "A Survey on malware detection techniques". In: *2021 International Conference on Information Technology (ICIT)*. 2021, pp. 371–376. DOI: 10.1109/ICIT52682.2021.9491765.

[3] *Atomic Red Team*. accessed 14/10-23 13:15. URL: https://atomicredteam.io/.

[4] Dustin Boswell. "Introduction to support vector machines". In: *Departement of Computer Science and Engineering University of California San Diego* 11 (2002).

[5] Valentina Casola, Alessandra De Benedictis, Carlo Mazzocca, and Vittorio Orbinato. "Secure software development and testing: A model-based methodology". In: *Computers & Security* 137 (2024), p. 103639. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103639. URL: https://www.sciencedirect.com/science/article/pii/S0167404823005497.

[6] *Chat-GPT*. Accessed 09/09-24 20:08. URL: https://openai.com/index/chatgpt/.

[7] Kalpathy-Cramer J Choi RY Coyner AS, Chiang MF, and Campbell JP. "Introduction to Machine Learning, Neural Networks, and Deep Learning". In: Transl VIs Schi Technol 9(2); (2020). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7347027/.

[8] Robertas Damasevicius, Algimantas Venckauskas, Sarunas Grigaliunas, Jevgenijus Toldinas, Nerijus Morkevicius, Tautvydas Aleliunas, and Paulius Smuikys. "LITNET-2020: An Annotated Real-World Network Flow Dataset for Network Intrusion Detection". In: *Electronics* 9.5 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9050800. URL: https://www.mdpi.com/2079-9292/9/5/800.

[9] *Dataset main python file*. URL: https://github.com/Ayustatus/thesis2023/blob/master/dataset/main.py.

[10] *Git repo*. URL: https://github.com/Ayustatus/thesis2023.

[11] Prasanta Gogoi, Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Packet and Flow Based Network Intrusion Dataset". In: *Contemporary Computing*. Ed. by Manish Parashar, Dinesh Kaushik, Omer F. Rana, Ravi Samtaney, Yuanyuan Yang, and Albert Zomaya. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 322–334. ISBN: 978-3-642-32129-0.

[12] *Google Scholar*. URL: https://scholar.google.com/.

[13] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J. Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. "Kubric: A Scalable Dataset Generator". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 3749–3761.

[14] *Invoke Atomic Red Team*. accessed 15/10-23 12:45. URL: https://github.com/redcanaryco/invoke-atomicredteam/wiki.

[15] *Kaggle*. URL: https://www.kaggle.com/.

[16] Max Landauer, Klaus Mayer, Florian Skopik, Markus Wurzenberger, and Manuel Kern. *Red Team Redemption: A Structured Comparison of Open-Source Tools for Adversary Emulation*. 2024. arXiv: 2408.15645 [cs.CR]. URL: https://arxiv.org/abs/2408.15645.

[17] *Linux Network Namspace Man page*. accessed 10/10-23 9:30. URL: https://man7.org/linux/man-pages/man7/network_namespaces.7.html.

[18] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things". In: *IEEE Access* 5 (2017), pp. 18042–18050. DOI: 10.1109/ACCESS.2017.2747560.

[19] Peter Mell. "Understanding Intrusion Detection Systems". In: *EDPACS* 29 (Nov. 2001), pp. 1–10. DOI: 10.1201/1079/43273.29.5.20011101/31414.1.

[20] *Mitre Attack Framework*. accessed 13/10-23 14:30. URL: https://attack.mitre.org/.

[21] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[22] Momoka Okuma, Koki Watarai, Satoshi Okada, and Takuho Mitsunaga. "Automated Mapping Method for Sysmon Logs to ATT&CK Techniques by Leveraging Atomic Red Team". In: *2023 6th International Conference on Signal Processing and Information Security (ICSPIS)*. 2023, pp. 104–109. DOI: 10.1109/ICSPIS60075.2023.10343783.

[23] Avi Pfeffer, Brian Ruttenberg, Lee Kellogg, Michael Howard, Catherine Call, Alison O'Connor, Glenn Takata, Scott Neal Reilly, Terry Patten, Jason Taylor, et al. "Artificial intelligence based malware analysis". In: *arXiv preprint arXiv:1704.08716* (2017).

[24] Paul Prasse, Lukáš Machlica, Tomáš Pevný, Jiří Havelka, and Tobias Scheffer. "Malware Detection by Analysing Network Traffic with Neural Networks". In: *2017 IEEE Security and Privacy Workshops (SPW)*. 2017, pp. 205–210. DOI: 10.1109/SPW.2017.8.

[25] Josh Schneider. *The history of malware: A Primer on the evolution of cyber threats*. created November 6 2023, accessed May 1 2024. 2023. URL: https://www.ibm.com/blog/malware-history/.

[26] *scikit-learn*. URL: https://scikit-learn.org/stable/.

[27] Jingwen Sun, Weixing Du, and Niancai Shi. "A Survey of kNN Algorithm". In: *Information Engineering and Applied Computing* 1 (May 2018). DOI: 10.18063/ieac.v1i1.770.

[28] T7hM1. *Origin of ddos script*. 'last accessed at 02/02-24 19:34'. URL: http://github.com/t7hm1/pyddos.

[29] Aaron Walker and Shamik Sengupta. "Malware Family Fingerprinting Through Behavioral Analysis". In: *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 2020, pp. 1–5. DOI: 10.1109/ISI49825.2020.9280529.