

A REVIEW ON OPERATING SYSTEM CLASSIFICATION WITH MACHINE
LEARNING USING TCP/IP AND TLS INFORMATION

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

KRISTJAN PASHOLLARI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY, 2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Kristjan Pashollari

Signature:

ABSTRACT

A REVIEW ON OPERATING SYSTEM FINGERPRINTING WITH MACHINE LEARNING USING TCP/IP AND TLS INFORMATION

Kristjan Pashollari

M.Sc., Department of Computer Engineering

Supervisor: Dr. Ali Osman Topal

It is an issue of both security and management for all network administrators to determine the Operating Systems (OS) that are using their network. Identification of Operating Systems in any kind of network has been a real challenge due to the rapid changes of the encryption protocols and the quick enlargement of the data.

In order to solve this problem, there are plenty active and passive fingerprinting methods than can lead to finding the real OS behind the traffic, but on top of these outdated methods, the one that has a great interest from all researchers is undoubtedly using Machine Learning (ML). The difficulties in this field starts from building the dataset, to choosing the best algorithm to find the OS from some simple features of TCP/IP packets or from TLS handshake information.

In this thesis we will show how can OS fingerprinting can be achieved with machine learning and what are the tools that one may need to do this task. We will state also different methods of OS fingerprinting using network traffic.

Keywords: OS fingerprinting, Operation System detection, Machine Learning, Classification, Encrypted Network Traffic

ABSTRAKT

RISHIKIM MBI GJURMIMIN E SISTEMEVE TW OPERIMIT ME “MACHINE LEARNING” DUKE PERDORUR PAKETAT TCP/IP DHE NDERLIDHJEN TLS

Kristjan Pashollari

Master Shkencor, Departamenti i Inxhinierisë Kompjuterike

Udhëheqësi: Dr. Ali Osman Topal

Është një çështje sa menaxhimi po aq edhe sigurie për të gjithë administratorët e rrjeteve për sa I përket gjetjes së sistemit të operimit të pajisjeve që ndodhen në atë rrjet. Identifikimi I Sistemit të Operimit në cfarëdolloj rrjeti është bërë një sfidë e vërtetë me zhvillimet jashtëzakonisht të shpeshta të protokolleve të enkriptimit dhe të zgjerimit të tejskajshëm të të dhënave.

Për të zgjidhur këtë problem aktualisht kemi shumë metoda qoftë ato active apo pasive të cilat mund të na cojnë deri në gjetjen e sistemit të operimit të pajisjes që përdor rrjetin, por pavarësisht këtyre metodave të prapambetura padyshim që zgjidhjet me anë të “Machine Learning” janë më të kërkuarat dhe më të preferuarat nga të gjithë kërkuesit shkencorë. Vështirësitë në këtë fushë fillojnë që nga gjetja e datasetit deri në zgjedhjen e algoritms që ka rezultatet më të mira për gjetjen e sistemit të operimit në bazë të paketave TCP/IP ose të ndërveprimit fillestar TLS.

Në këtë punim diplome ne do të shfaqim sesi mund të bëhet gjetja e sistemit të operimit me anë të teknikave “Machine Learning” dhe se cilat janë mjetet e nevojshme për këtë problem. Gjithashtu ne do të paraqesim edhe disa metoda të tjera duke analizuar në brendësi ato.

Fjalët kyçe: Gjurmimi I Sistemit të Operimit, Gjetja e Sistemit të Operimit, Machine Learning, Klasifikim, Trafik I enkriptuar në rrjet

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Dr. Ali Osman Topal for his encouragement and motivation throughout my studies and especially during my thesis. He, untiredly helped me to surpass all obstacles and to achieve the most out of myself.

I am also thankful to all professors of Epoka University. Each of them in their own way helped me to walk further.

I am deeply thankful to my family. They made me who I am and continuously motivated to see further and further. Thankful for having you!

TABLE OF CONTENTS

| | |
|--|----|
| ABSTRACT..... | iv |
| ABSTRAKT | v |
| ACKNOWLEDGEMENTS | vi |
| LIST OF FIGURES | ix |
| LIST OF ABBREVIATIONS..... | xi |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1. Operation System Detection in closed network | 1 |
| 1.2. Structure of the Thesis..... | 3 |
| CHAPTER 2 | 4 |
| LITERATURE REVIEW | 4 |
| 2.1. Fingerprinting/Detection methods..... | 4 |
| 2.1.1. Active Fingerprinting..... | 4 |
| 2.1.2. Passive Fingerprinting | 5 |
| 2.1.3. Network Layers..... | 6 |
| 2.2. Classification using TCP/IP packet header information. | 11 |
| 2.3. Classification using TLS handshake information. | 13 |
| 2.4. Evaluation of Features and Methods..... | 15 |
| CHAPTER 3 | 16 |
| METHODOLOGY | 16 |
| 3.1. Methodology | 16 |
| 3.2. Datasets used and Challenges. | 17 |
| 3.2.1. Dataset-1 (Virtual Environment) | 17 |
| 3.2.2. Dataset-2 (Real Environment – University Campus) | 21 |
| 3.3. Dataset Comparison | 23 |

| | |
|---|----|
| 3.3.1. Libraries and Tools | 24 |
| 3.4. Types of Learning | 25 |
| 3.5. Multi-Layer Perceptron (MLP) of Artificial Neural Network (ANN)..... | 27 |
| 3.5.1. Multi-Layer Perceptron Architecture..... | 28 |
| CHAPTER 4 | 30 |
| IMPLEMENTATION AND RESULTS..... | 30 |
| 4.1. Experiments..... | 30 |
| Dataset – 1..... | 30 |
| Dataset – 2..... | 37 |
| CHAPTER 5 | 44 |
| CONCLUSION AND FUTURE WORK | 44 |
| Conclusions..... | 44 |
| Future Work..... | 44 |
| References..... | 45 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1. Active OS fingerprinting illustration | 2 |
| Figure 2. Passive OS fingerprinting illustration | 3 |
| Figure 3. OS fingerprinting components | 5 |
| Figure 4. Network Layers in OSI model | 7 |
| Figure 5. HTTP request/response Headers | 8 |
| Figure 6. User-Agent example | 8 |
| Figure 7. TLS handshake - negotiation of encrypted session | 9 |
| Figure 8. TCP packet header structure | 10 |
| Figure 9. Simple representation of Client-Server Synchronization using TCP protocol. | 11 |
| Figure 10. Unauthorized OS detection in enterprise network | 12 |
| Figure 11. Independent micro averages for each method experimented at [5] | 15 |
| Figure 12. VMware Software | 18 |
| Figure 13. Wireshark software interface | 19 |
| Figure 14. TLS versions share in dataset-2 | 23 |
| Figure 15. OS share in dataset-1 | 23 |
| Figure 16. OS share in dataset-2 | 23 |
| Figure 17. Three types of Learning | 26 |
| Figure 18. Model of a Simple Neuron | 28 |
| Figure 19. MLP first Model for dataset-1 | 30 |
| Figure 20. Precision, Recall, F1score (batch=8) | 31 |
| Figure 21. Precision, Recall, F1score (batch=64) | 31 |
| Figure 22. Precision, Recall, F1score (batch=128) | 31 |
| Figure 23. Accuracy and Loss (batch=8) | 32 |
| Figure 24 Accuracy Train & Test (batch=8) | 32 |
| Figure 25 Loss Train & Test (batch=8) | 32 |
| Figure 26 Accuracy and Loss (batch=64) | 32 |
| Figure 27 Accuracy Train & Test (batch=64) | 32 |

| | |
|---|----|
| Figure 28 Loss Train & Test (batch=64) | 32 |
| Figure 29 Accuracy and Loss (batch=128) | 32 |
| Figure 30 Accuracy Train & Test (batch=128) | 32 |
| Figure 31 Loss Train & Test (batch=128) | 32 |
| Figure 32 ROC & AUC (batch=8) | 32 |
| Figure 33 Confusion Matrix (batch=8) | 33 |
| Figure 34 ROC & AUC (batch=64) | 33 |
| Figure 35 Confusion Matrix (batch=64) | 34 |
| Figure 36 ROC & AUC (batch=128) | 34 |
| Figure 37 Confusion Matrix (batch=128) | 35 |
| Figure 38. New model for dataset-1 with Dropout layers | 35 |
| Figure 39. Loss Curve (train & test) with dropout layers | 36 |
| Figure 40. Accuracy Curve (train & test) with dropout layers | 36 |
| Figure 41. Model for the second dataset | 37 |
| Figure 42. Precision, Recall, F1score (batch=8) | 1 |
| Figure 43. Precision, Recall, F1score (batch=64) | 1 |
| Figure 44. Precision, Recall, F1score (batch=128) | 1 |
| Figure 45. Accuracy and Loss Curves (train & test) with batch=8 | 39 |
| Figure 46. Accuracy and Loss Curves (train & test) with batch=64 | 39 |
| Figure 47. Accuracy and Loss Curves (train & test) with batch=128 | 40 |
| Figure 32 ROC & AUC (batch=8) | 41 |
| Figure 33 Confusion Matrix (batch=8) | 41 |
| Figure 34 ROC & AUC (batch=64) | 42 |
| Figure 35 Confusion Matrix (batch=64) | 42 |
| Figure 36 ROC & AUC (batch=128) | 43 |
| Figure 37 Confusion Matrix (batch=128) | 43 |

LIST OF ABBREVIATIONS

| | |
|------|------------------------------------|
| TLS | Transport Layer Security |
| TCP | Transfer Control Protocol |
| IP | Internet Protocol |
| HTTP | HyperText Transfer Protocol |
| OSI | Open System Interconnection |
| SSL | Secure Socket Layer |
| RFC | Request For Comment |
| TTL | Time To Live |
| WS | Window Size |
| DF | Don't Fragment bit |
| SNI | Server Name Indication |
| CSV | Comma Seperated Values |
| IDE | Integrated Development Environment |
| ROC | Receiver Operating Characteristics |
| AUC | Area Under Curve |
| IOT | Internet of Things |
| OS | Operating System |
| ML | Machine Learning |
| ANN | Artificial Neural Network |
| MLP | Multi-Layer Perceptron |

CHAPTER 1

INTRODUCTION

Internet of Things (IOT) is now part of everyone's life and inevitably many kinds of devices use the same network. This emerging use of internet in the other hand must rely on a dynamic network administration. Aksoy and M. Gunes say that Network managers adopt multiple security mechanisms to protect the network from malicious activities. An important step in securing a network is to be aware of the devices that are attached to the network. [1]

Starting from older research [2] [3] we can obviously see the attraction to the Machine Learning (ML) approaches and its benefits. Lately, some researchers from Ariel University [4] have shown that by using the HTTP headers in an encrypted traffic the users OS, Browser and Application can be retrieved. M. Lastovicka and S. Spacek at [5] have proven that by taking Transport Layer Security (TLS) handshake information between a client and a server OS detection can be achieved using a trained model.

1.1. Operation System Detection in closed network

OS detection in closed network is very useful for network administrators to easily manage and secure their network from intruders or from devices with low level of security, which can be a bridge on failing a complete network. In the last decade, due to the emerging size of devices, OS detection methods have been in radar for researchers and we see that many kinds of methods are available. There exist two main

methodologies explained at [2] or approaches to OS detection, active and passive fingerprinting.

By “fingerprinting” we mean finding some useful traces that could lead back to the real OS and detect it. Both mentioned methods have their own pros and cons which will be discussed in this work in the upcoming chapters. Briefly we will explain here these two approaches.

Active OS fingerprinting is basically achieved by sending a stimulus to the flagged OS. This stimulus usually is a kind of packet that after being sent, the network administrator waits for responses. The response of the OS determines it in a very approximate way. There are some mature tools that have proven to be solid and accurate in active fingerprinting such as Nmap [6] and Xprobe [7].

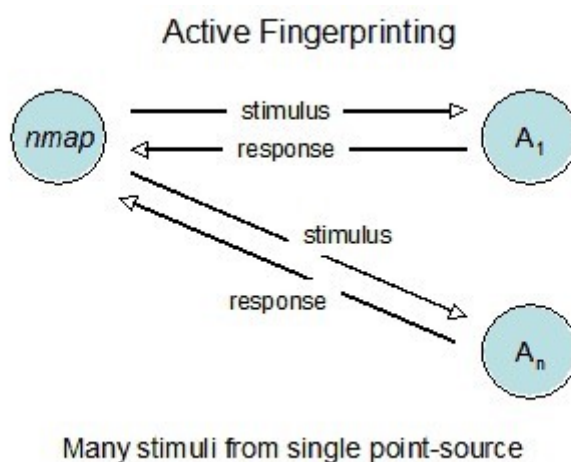


Figure 1. Active OS fingerprinting illustration

Passive OS fingerprinting is simply monitoring the traffic in a network with different tools and gather data that are available to the network administrator. By using this method, users (clients) do not have a clue on what is happening in the background. Although accuracy of this method is not as higher as active methods, still is more preferred. For passive fingerprinting we can mention a couple of tools that are available for use such as p0f [6] and SinFP [8].

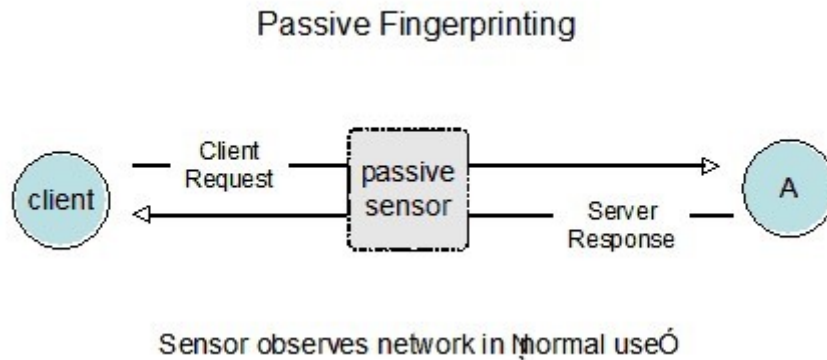


Figure 2. Passive OS fingerprinting illustration

1.2. Structure of the Thesis

The structure of this thesis will be composed of 5 chapters. In the first chapter we will show the main problems regarding OS detection in a closed network. The importance of researching in this field will be also briefly explained along with a short introduction to the main approaches to this problem.

In Chapter 2 we give an overview of what is the current status of researches in the field of OS detection in closed networks, mainly using the passive fingerprinting approach. Our focus will be on explaining the current researches that include ML techniques. By the end of this chapter, we will also explain the datasets that have been used for this research and their respective description.

In Chapter 3 we explain our methodology and the approach we have regarding this problem. As this thesis will be somewhat review oriented, we will explain the reasons of the used methodology. In this chapter the datasets will be explained in detail from the process of data acquisition to the final state.

Chapter 4 will be a showcase of our experiments and their results. Here we will show statistically the pros and cons of the actual solutions.

Our last Chapter is Chapter five where we conclude this our work and give some future recommendations prior to the results of the experiments.

CHAPTER 2

LITERATURE REVIEW

OS detection during the years is a matter of discussion on using active or passive fingerprinting, each of which possesses some nice benefits. In this chapter we will explain which is the right choice according to the literature and then will dive into passive fingerprinting in details. In this section we will show the datasets and features that have been used from the previous researchers, and at the end will conclude on evaluating the approaches.

2.1. Fingerprinting/Detection methods

2.1.1. Active Fingerprinting

As it was explained earlier in this research, there exist two main approaches to OS detection (fingerprinting), Active and Passive fingerprinting. We explained briefly in the first chapter that active fingerprinting happens by stimulating the OS and analyzing its response. This method is usually applied manually by the network administrator towards suspicious users/clients.

A thorough research [9] conducted on active fingerprinting methods and tools gives a better view on how this method deals with OS detection problem. Fingerprinting is generally composed of 4 steps or components:

1. Data acquisition in a network.
2. Finding fingerprints from the acquired data.
3. Fingerprints database where each of them is labelled.

4. Getting results by matching fingerprints with the database using an algorithm.

In the Figure 3 the above process is shown graphically.

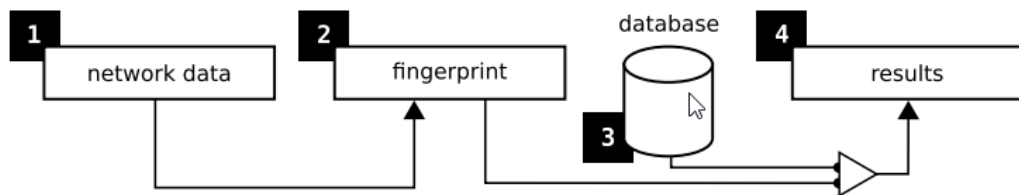


Figure 3. OS fingerprinting components

Data acquisition in a network varies from the method that a network administrator or an intruder follows to fingerprint a device. Most commonly we see the combination of Transmission Control Protocol (TCP) and Internet Protocol (IP) information. Another type of fingerprinting is by using Hypertext Transfer Protocol / Secure (HTTP/S) header information that clients send or receive to each other.

The person behind the device that probes actively the user's device must be in the same network. At this point we should mention that by probing an OS we can be detected or caught by that OS Firewall, and by doing so we may fail on detecting the OS. The accuracy of this method is very high and if we can successfully probe and get an observable response from the OS, then with an accuracy of more than 99% OS will be detected. Due to its drawback, we will not consider it for further studies and research in our work. It is worth mentioning here that this method is quite mature and can be used for some other use cases.

2.1.2. Passive Fingerprinting

While active methods and tools fail on being hidden and consistent, passive methods give more reliability when it comes to gathering data. In other words, passive fingerprinting happens by sniffing network packets from other devices in the same network. It is quite common that sniffing is not as accurate as actively acquiring something. In this case, we do not have all information that we need to precisely detect

an OS. Though, we still use and prefer passive methods for being more stable and sustainable.

Most commonly, in Passive fingerprinting we use the information from these protocols:

- TCP/IP [3] [1] [10]
- HTTP options [1]
- TLS handshake [5] [4]

2.1.3. Network Layers

In order to have a better view on how we can fingerprint an OS, firstly we must know how our data travels in the network. Generally, the data transportation from a device to another in a network is divided into 7 main layers where each of which encrypts or decrypts the data and sends to the layer after it. [11] This is the representation of Open Systems Interconnection Model (OSI), a conceptual framework that organizes a network system.

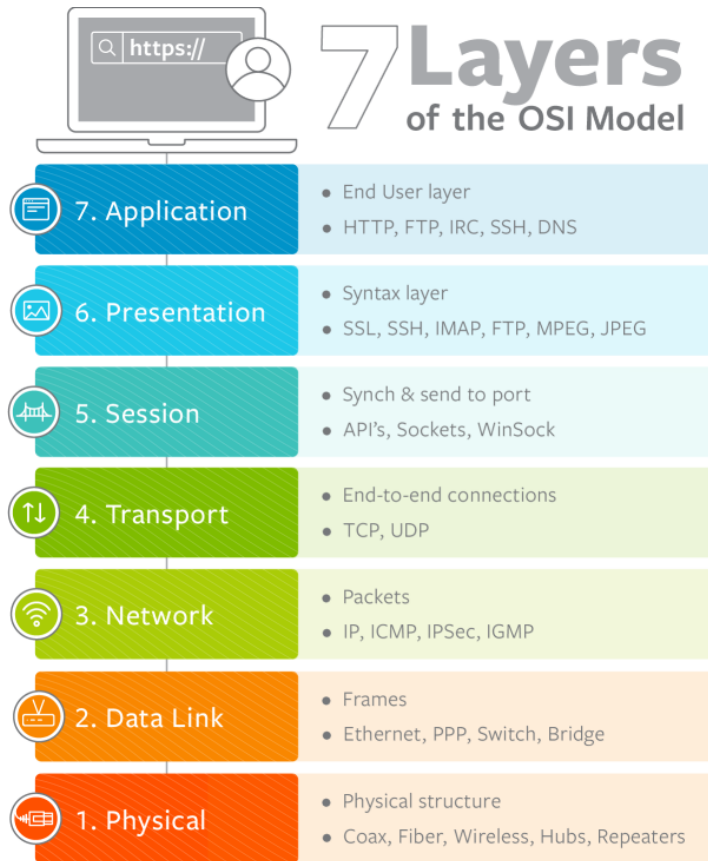


Figure 4. Network Layers in OSI model

In Figure 4 we see that HTTP, TCP, IP and TLS or Secure Socket Layer (SSL) are placed in the 7th, 6th, 5th, 4th and 3rd layers. The top 3 of them can be grouped into Application Layer for ease of use as their functionality is almost same. The question at this point would be, how these layers communicate with each other and what is the information that we need to get from this communication in order to find some OS fingerprints there?

HTTP information

Well, the first protocol that we see from top to bottom of Figure 4, is the HTTP protocol. All HTTP requests or responses have a header that stores information for that request. HTTP requests in fact do not hold too much information regarding the fact that is a top layer protocol. User-agent is a field that is stored in the headers of HTTP request and that is the most relevant feature to detecting OS. That feature stores the information

where the request comes from and what are the details of the browser. It is important to note here that not all requests must have a user agent. This method gives a very accurate result, but it is not consistent. [5]

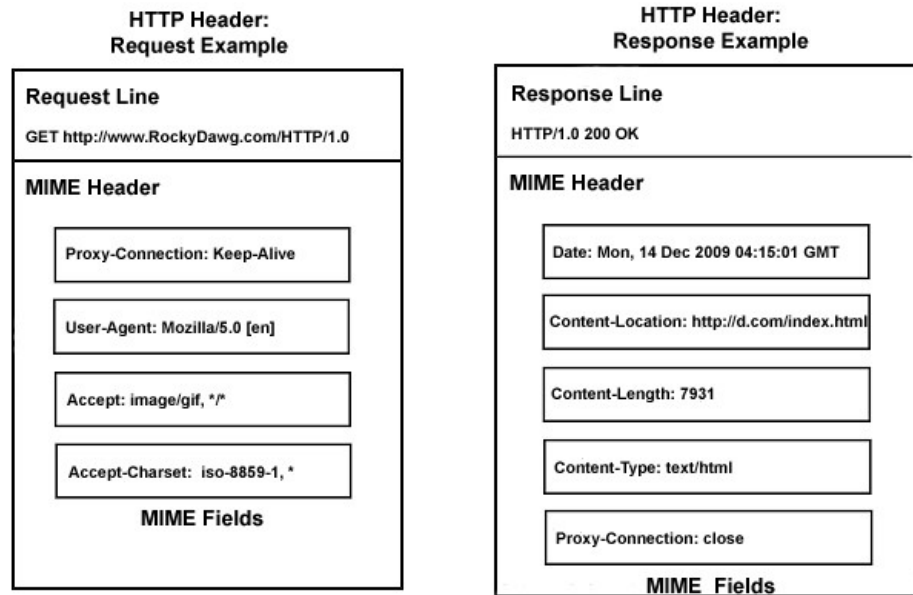


Figure 5. HTTP request/response Headers

| URL: http://intranet/Pages/default.aspx | |
|---|--|
| Request headers | |
| Key | Value |
| Request | GET /Pages/default.aspx HTTP/1.1 |
| Accept | text/html, application/xhtml+xml, */* |
| Accept-Language | en-ZA |
| User-Agent | Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0) |
| Accept-Encoding | gzip, deflate |
| Host | intranet |
| If-Modified-Since | Fri, 17 Aug 2012 14:52:07 GMT |
| Connection | Keep-Alive |
| Cookie | MSOWebPartPage_AnonymousAccessCookie=80; WSS_KeepSessionAuthenticated=80 |

Figure 6. User-Agent example

TLS/SSL

Transport Layer Security (TLS) is the updated and the last version of Secure Socket Layer (SSL) certificates. Usually, we hear the term SSL much more than TLS,

due to its being more popular, but as mentioned above TLS is the newest version of SSL certificates.

TLS or SSL protocols are placed in the Presentation Layer of OSI stack. This protocol encrypts the data according to an algorithm so that only sender and receiver must know how to decrypt and retrieve all information. The communication between two end-users or client and server, if it is done in a secure way then TLS has been used. As shown in the figure 7, Client makes a request to server and after receiving acknowledgment from the server that everything is settled, the encryption communication proceeds. [5] Dotted arrows represent encrypted communication while simple arrows represent plain text communication.

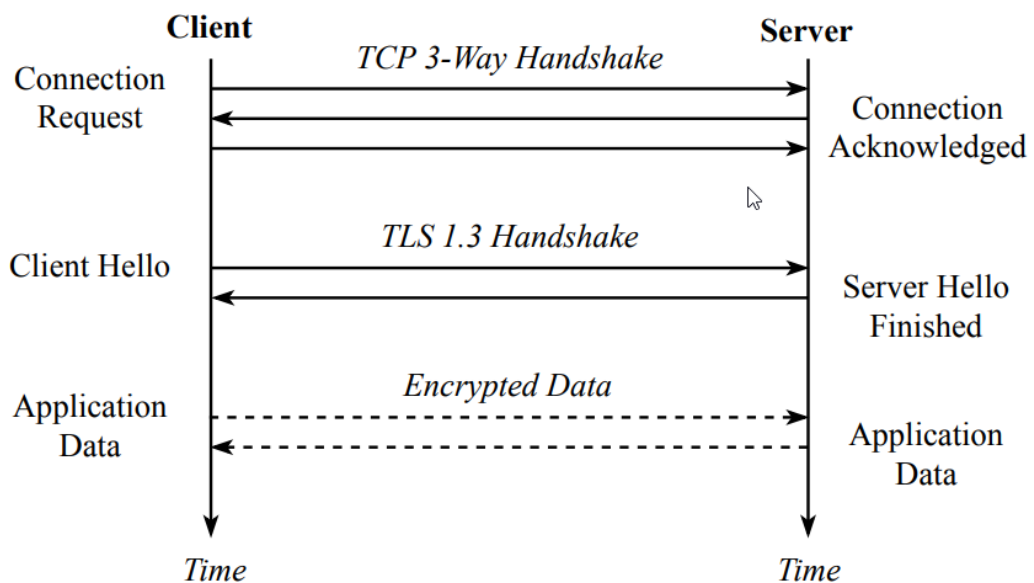


Figure 7. TLS handshake - negotiation of encrypted session

TCP/IP

Computers have similar rules they must follow when communicating over a network, the most common of which is the TCP/IP suite of protocols. IP is a method of assigning and managing logical addresses for each host on the network, while TCP ensures that all packets are delivered correctly. These protocols must be implemented in any operating system that wants to talk on the Internet. Both protocols are described

in their respective Request For Comment's (RFC)s, [RFC-791 [12] and RFC-793 [13] for IP and TCP respectively]

Transmission Control Protocol is the protocol in Transport Layer of OSI stack and its main functionality is to take care of all data shared from client to server and vice-versa. The sender and receiver communicate by sending packets and waiting for acknowledgements, responses, for each sent packet. This protocol is wisely used from applications to ensure data sustainability.

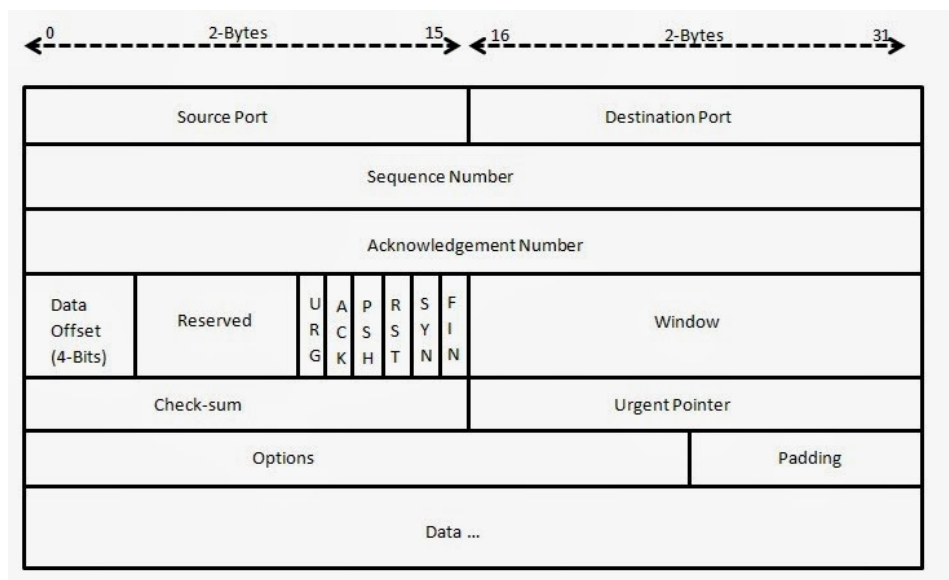


Figure 8. TCP packet header structure

From Figure 8 we can have a look on what are the most important parameters and how a TCP packet is structured. Table 1 shows in detail on what TCP Flags are and what is their functionality.

Table 1. TCP Header Flag fields [14]

| TCP Flags | Full Names | Descriptions |
|-----------|-----------------|---|
| URG | Urgency pointer | Indicates the TCP priority of the packets. |
| ACK | Acknowledgment | Designates this packet as an acknowledgment of receipt. |
| PSH | Push | Flushes queued data from buffers. |

| | | |
|-----|-----------------|---|
| RST | Reset | Resets a TCP connection on completion or being aborted. |
| SYN | Synchronization | Synchronizes a connection. |
| FIN | Finished | Finishes a transmission. |

In the literature we see that the parameters of TCP are taken and analyzed along with those of IP protocol. [1] [10] [4] It seems like TCP and IP together have a match on the OS to be detected. Below (Figure 9) shows how TCP protocol synchronizes client and server for the packet transmission.

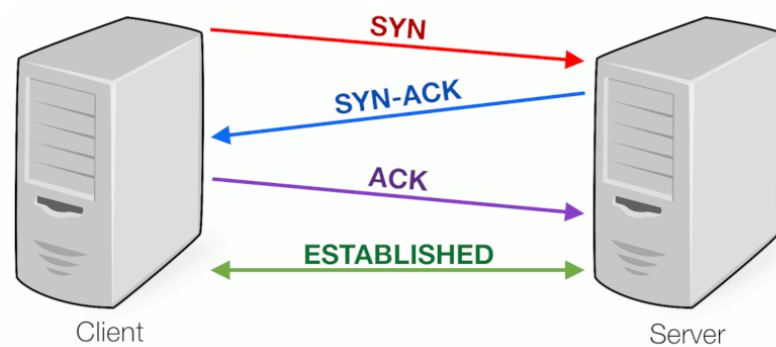


Figure 9. Simple representation of Client-Server Synchronization using TCP protocol.

Each OS structures differently the TCP and IP parameters on the packets that are sent. Features such as Time To Live (TTL), Window Size (WS), Don't Fragment bit (DF), SYN flag, FIN flag, and some others give a clue on what the OS could be. We should note here that TTL may be same when we access something on a Windows or Linux, but in many cases the combination of all or many TCP/IP features results in a very accurate detection, which is 99.1% reached by Aksoy and Gunes at [1]

2.2. Classification using TCP/IP packet header information.

Aksoy and Gunes at their work [1] used Genetic Algorithm to populate a set of rules from which will be decided which features will be selected to perform the experiment. They applied many Machine Learning algorithms into their dataset to see which of them fits well into this problem. Experiments conducted in full set of features

are no good than those conducted in a particular set of features which was selected from the Genetic Algorithm.

In another work one year later, Aksoy and Gunes again represented the same approach and experiments with a higher number of devices and Operating Systems. Their second work is almost the same with the first one and the experiments were conducted in a virtual environment.

The accuracy at this scale is quite an achievement and proves that OS can be detected using TCP/IP parameters, but this study is tested in a closed network of virtual machines which leaves the option of Wireless connection untested. Even though TCP connection does not differ whether the machine is real or virtual and the network is wired or wireless, it is a future work testing this approach in real life.

In another work by R. Tyagi and T. Paul at [15] we see a similar approach by analyzing TCP/IP header flags and options. Their intention is to deny access to the unauthorized OS'es in the enterprise network. In figure 10 the proposed system is shown.

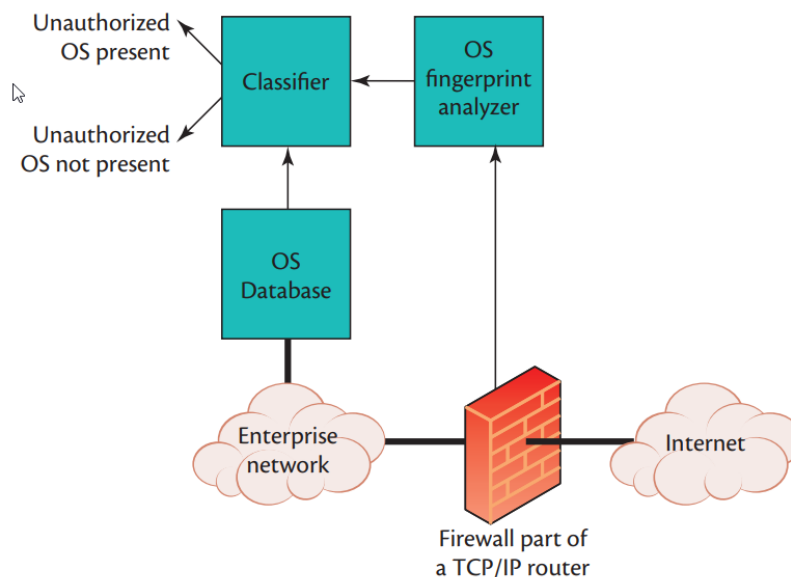


Figure 10. Unauthorized OS detection in enterprise network

In their work they have also shown the TCP options and IP header options for popular Operating Systems until 2015. The table below (Table 2) shows precisely these data and noticeably you can see the difference between OS's.

Table 2. Popular OS's IP and TCP Header options [15]

| OS | TTL | Packet size (bytes) | NOP (No Operation) | Selective acknowledgment (SACK OK) | Don't Fragment (DF) | Time stamp |
|--------------|-----|---------------------|--------------------|------------------------------------|---------------------|------------|
| Linux | 64 | 60 | 1 | 1 | 1 | 1 |
| OpenBSD | 64 | 64 | 1 | 0 | 1 | 1 |
| AIX 4.3 | 64 | 44 | 0 | 0 | 1 | 0 |
| Windows 2000 | 128 | 48 | 1 | 1 | 1 | 0 |
| Windows 7 | 128 | 52 | 1 | 1 | 1 | 0 |
| Windows 8 | 128 | 52 | 1 | 1 | 1 | 0 |
| Cisco IOS | 255 | 44 | 0 | 1 | 0 | 0 |
| Solaris 7 | 255 | 44 | 0 | 1 | 1 | 0 |
| MAC | 64 | 60 | 1 | 0 | 1 | 1 |

2.3. Classification using TLS handshake information.

In a work presented by M. Husak and M. Cermak at [16] HTTPS traffic has been analyzed in order to detect the Operating System. TLS handshake has been monitored and the unencrypted data has been taken. As explained earlier in this chapter, when client and server want to communicate through a secured connection, firstly they negotiate the parameters of the encrypted connection. These plain text parameters have been processed and results are not so satisfactory. Authors in this work tried to explain these three questions:

1. Which parameters of a SSL/TLS handshake can be used for client identification?
2. Can we pair selected SSL/TLS handshake parameters and HTTP header fields?
3. Can we utilize the SSL/TLS fingerprinting in network security monitoring and intrusion detection?

Experiments based on only TLS/SSL handshake options and messages tells us that those are not enough to correctly identify the client and that we do need some extra information for this task.

Another work [5] have used TLS and SSL parameters to fingerprint OS but in their case additional options have been processed from TCP and IP headers. Their results are quite impressive, and this suggests us that a combination of protocols leads into an accurate OS detection. In the table below (Table 3) are listed all features processed in this work

Table 3. Features extracted from network data in [5]

| <i>Protocol</i> | <i>Feature Name</i> |
|-----------------|-------------------------------|
| TCP | SYN packet size |
| TCP | TTL of TCP SYN packet |
| IP | Windows Size (WS) |
| HTTP | User Agent |
| HTTP | Hostname |
| TLS | Server Name Indication (SNI) |
| TLS | Client version |
| TLS | Cipher suites |
| TLS | Extension types |
| TLS | Extension length |
| TLS | Supported groups |
| TLS | Elliptic curves point formats |

In their work they have found that TLS is responsible for more than 97% of network traffic and this indicates that TLS features must be considered for analyzation. Independently they analyzed TCP/IP, TLS handshake, Specific Domain and User-Agent methods and results for each of them tell us that only a combination of all of them leads into perfect results. Below, in figure 11 you can find the results.

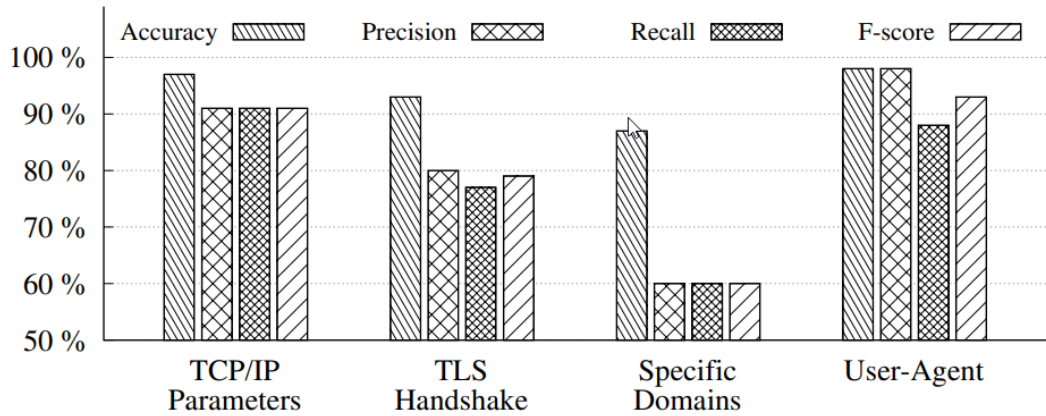


Figure 11. Independent micro averages for each method experimented at [5]

2.4. Evaluation of Features and Methods

The results and literature conducted until now shows us that protocols independently do not hold precise information in order to make a perfect OS detection and fingerprinting. The only safe method that we should use is that of combining different options and parameters from different protocols such as TCP/IP, TLS and HTTP. In the next chapters we will conduct two experiments for 2 different datasets by combining as much features as possible to get acceptable results.

CHAPTER 3

METHODOLOGY

3.1. Methodology

In this thesis we use both methodologies the Qualitative research and the Quantitative research. The thesis aim is to go deeper and dive in the field of OS fingerprinting and to have a better understanding on what needs to be improved. In the other hand we will tend to prove or suggest methods and approaches on solving this problem. For this reason, here you will find implemented both types of research.

Qualitative Research

Qualitative Research is considered to be particularly suitable for exploratory research (e.g., during the pilot stage of a research project, for example). It is primarily used to discover and gain an in-depth understanding of individual experiences, thoughts, opinions, and trends, and to dig deeper into the problem at hand. [17]

For this reason, we indeed need to include Qualitative Research design in this work to have a better representation of the facts and methods used in this field.

Quantitative Research

Quantitative research is all about numbers and figures. It is used to quantify opinions, attitudes, behaviors, and other defined variables with the goal to support or refute hypotheses about a specific phenomenon, and potentially contextualize the results from the study sample in a wider population (or specific groups). As quantitative

research explicitly specifies what is measured and how it is measured in order to uncover patterns. [17]

3.2. Datasets used and Challenges.

In this work we will compare two types of datasets each of which acquired and built in a different way with different features. Devices are connected in a network in too many ways where each of them uses different application and protocols. To make a simple comparison we created a dataset in a virtual environment where each machine was connected virtually via ethernet to the network and the second dataset was built by some research in a university campus. Both datasets will be explained more in the following subsection.

3.2.1. Dataset-1 (Virtual Environment)

This dataset contains packets from 4 different Operating Systems running in a virtual machine. We created these virtual machines by using VMware software [18] which serves us to create Virtual Machines which run the same as in real life. Operating Systems used are: Windows 10, Windows 7, Xubuntu, RaspBerry Pi.

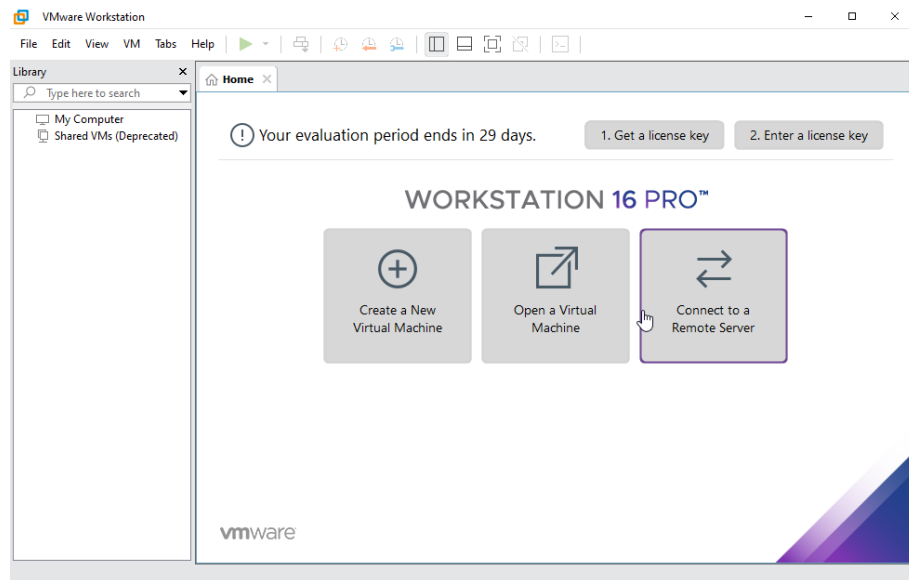


Figure 12. VMware Software

After successfully installing and setting up these Virtual Machines, in each of them we installed Wireshark Software. Wireshark is the world's foremost and widely used network protocol analyzer. It lets you see what is happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. [19]

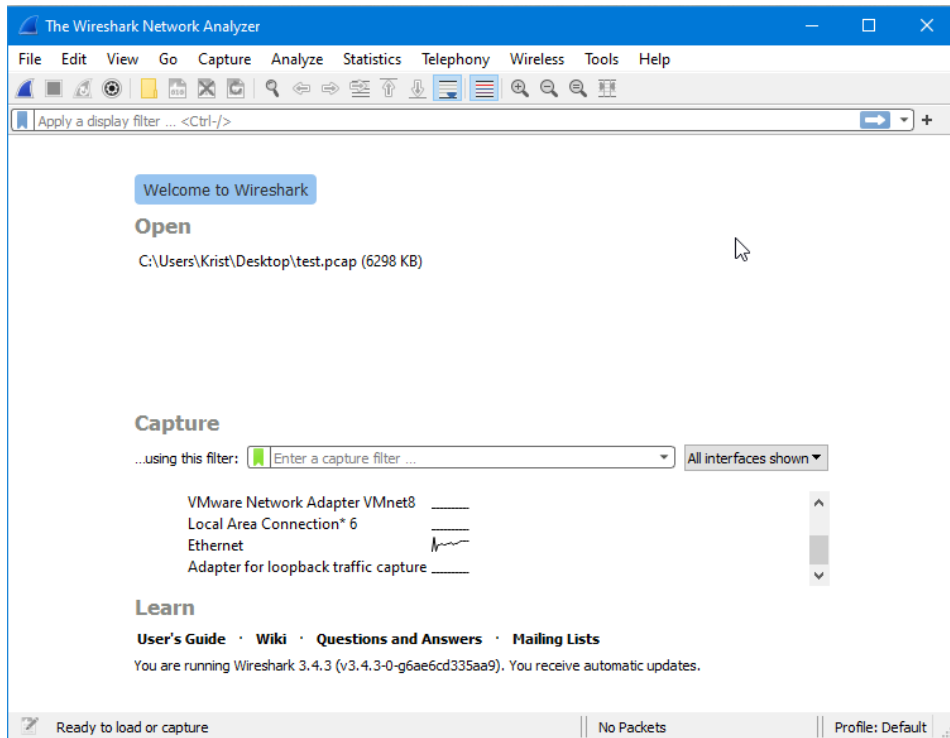


Figure 13. Wireshark software interface

In each of devices we started Wireshark packet tracing options to gather all incoming or outgoing packets from the virtual network adapter. In each of devices we ran some simple tasks such as accessing some websites, opening some minutes of YouTube video, sharing files etc., in order to test as much as possible, the network in many different protocols. At the end of this process, Wireshark itself has the feature to export all gathered packets in a single file named with the extension “.pcap”.

The pcap files do not have a meaning in their own without reading them with the proper software. We indeed do need something to convert them into a meaningful and applicable set of data, which for us would be Comma Separated Values (CSV) file type. After some research on how to extract features from .pcap file types we found a community solution written in python that extracts the following features: [20]

Table 4. PCAP features extracted using python script. [20]

| Feature | Description |
|---------|-------------|
|---------|-------------|

| | |
|------------------------------|--|
| Avg_syn_flag | |
| Avg_urg_flag | |
| Avg_fin_flag | |
| Avg_ack_flag | |
| Avg_psh_flag | |
| Avg_rst_flag | |
| Avg_DNS_pkt | The average pf DNS packets in a window of packets |
| Avg_TCP_pkt | |
| Avg_UDP_pkt | |
| Avg_ICMP_pkt | |
| Duration_window_flow | The time from the first packet to last packet in a window of packets. |
| Avg_delta_time | The average of delta times in a window of packets. Delta time is the time from a packet to the next packet |
| Min_delta_time | The minimum delta time in a window of packets |
| Max_delta_time | The maximum delta time in a window of packets |
| StDev_delta_time | The Standard Deviation of delta time in a window of packets |
| Avg_pkts_lenght | The average of packet leghts in a window of packet |
| Min_pkts_lenght | |
| Max_pkts_lenght | |
| StDev_pkts_lenght | |
| Avg_small_payload_pkt | |
| Avg_payload | |
| Min_payload | |
| Max_payload | |
| StDev_payload | |
| Avg_DNS_over_TCP | The average of ration DNS/TCP in a window of packets |
| Label | 0 1 respectively if pcap is legitimate or malware |

At the end of this process, we had 4 CSV files each of them for its respective Operating System. We labelled each of them from 0 to 3 for the respective OS show below in Table 5.

Table 5. Dataset-1, Labelling Operating Systems

| <i>Label</i> | <i>Operating System</i> |
|--------------|-------------------------|
| 0 | Windows 10 |
| 1 | Raspberry Pi |
| 2 | Windows 7 |
| 3 | Xubuntu |

3.2.2. Dataset-2 (Real Environment – University Campus)

In a late research from M. Lastovicka [5] the authors have created a huge dataset of packets shared from a network in their University campus. This dataset is interesting due to its being only on a Wireless network and all packets generated in the network are sent using mobiles, tablets or computers by using wireless connection of the campus for approximately 2 days.

Features of this dataset are shown below in table 6 but not all of them have descriptive data for the Operating System behind the packet. Some of them, such as time initiated, or time finished will not be considered at all.

Table 6. Features in 2nd Dataset

| | |
|---|---------------|
| 1 | Date flow end |
| 2 | Src IPv4 |
| 3 | sPort |
| 4 | Dst IPv4 |
| 5 | dPort |
| 6 | SYN size |
| 7 | TCP win |

| | |
|----|------------------------|
| 8 | TCP SYN TTL |
| 9 | TLS SNI |
| 10 | TLS SNI length |
| 11 | TLS Client Version |
| 12 | Client Cipher Suites |
| 13 | TLS Extension Types |
| 14 | TLS Extension Lengths |
| 15 | TLS Elliptic Curves |
| 16 | TLS EC Point Formats |
| 17 | HTTP Host |
| 18 | HTTP UA OS |
| 19 | HTTP UA OS MAJ |
| 20 | HTTP UA OS MIN |
| 21 | HTTP UA OS BLD |
| 22 | SSH Client Version |
| 23 | SSH Client Application |
| 24 | SSH Client Encryption |
| 25 | SSH Client MAC |
| 26 | SSH Client Compression |
| 27 | Session ID |
| 28 | Ground Truth OS |

A short analysis in this dataset would be to see the TLS or SSL versions used by devices. We thought that this may reveal some information regarding the OS too but that was not the case. TLS version 1.2 is used the most and for more than 95% of the devices are making the secure connection via this version. SSL as can be seen from figure 14 below, is insignificant and thus that is something irrelevant to this problem.

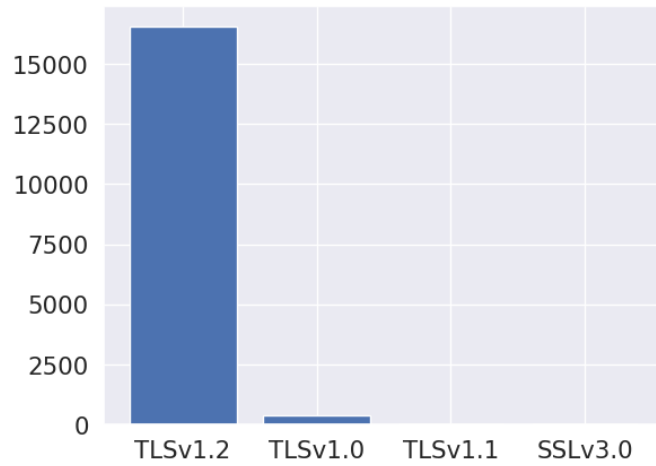


Figure 14. TLS versions share in dataset-2

3.3. Dataset Comparison

In both datasets we have 4 labels, ground truths. The OS share in each dataset is as shown below on figures 15 and 16. In

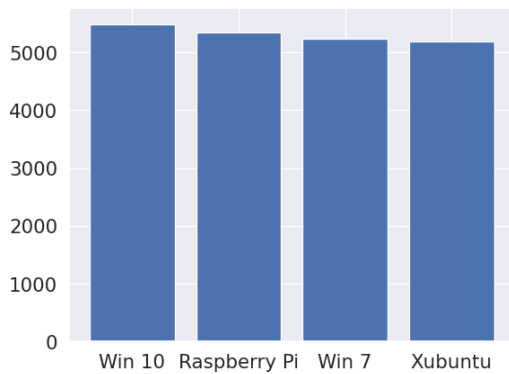


Figure 15. OS share in dataset-1

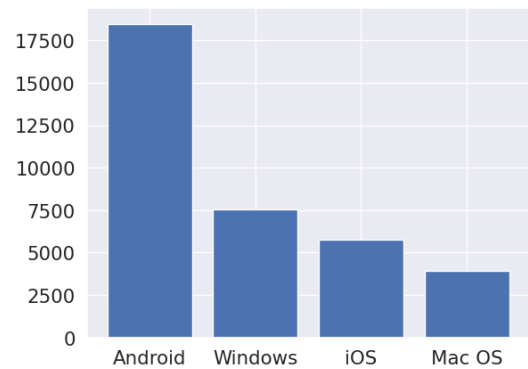


Figure 16. OS share in dataset-2

Table 7. Dataset detail comparison

| | Dataset 1 | Dataset 2 |
|---------------------------|------------------|------------------|
| Nr. of rows | 26587 | 35654 |
| Train set | 21269 | 70% |
| Test set | 5318 | 30% |
| Nr of Features | 24 | 28 |
| Processed Features | 24 | 13 |

3.3.1. Libraries and Tools

The experiments in this thesis are done using Python Scripting Language because of its benefits regarding AI. All the experiments can be done in pure Python Language but since It provides many packages libraries which make the code short, understandable and organized, we have chosen the shortest path.

Below are listed the main packages used and a short explanation for the main packages used in this project:

Numpy - When we work with Images, we always convert them into multidimensional arrays, in 1 or 2 dimensional arrays. In order to do this conversion and to work with arrays we use the Numpy package.

Pandas - We use this package for data manipulation and analysis. It is very helpful when our data is in .csv format.

Matplotlib - This library helps us with image visualizations. All the graphics and visualizations in this thesis are created using Matplotlib.

Seaborn – It is almost same as MatplotLib but kind of more advanced.

Sklearn – Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Tensorflow/Keras - Keras is a Neural Network library which is built on top of Tensorflow and is easier to use when we compare it with TF. All the layers of our Network are composed by Keras API.

Since we have done many experiments and we have tried to choose the best system and working environment for this project we used two Integrated Development Environment (IDE):

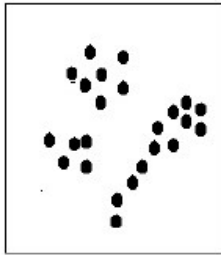
PyCharm is a very good choice when the code is organized in different files/modules and directories. It is a professional IDE and requires firstly to setup python and all libraries that we will use.

Google Colaboratory is very useful, powerful and handy IDE which can be accessed through Google Apps. It is the perfect choice for no-setup use and that's why we chose to use that.

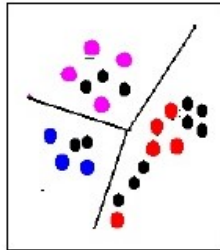
3.4. Types of Learning

In Deep Learning there are three types of learning: Unsupervised Learning, Supervised learning and Semi-Supervised Learning.

Unsupervised Learning



Supervised Learning



Semi-Supervised Learning

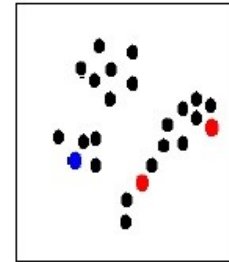


Figure 17. Three types of Learning

Unsupervised learning is used mainly to discover patterns and detect outliers in data. Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised Learning has two main categories of algorithms: Clustering and Association.

Semi-supervised learning learning is a combination of supervised and unsupervised machine learning methods. In this type of learning, the algorithm is trained upon a combination of labeled and unlabeled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabeled data. The basic procedure involved is that first, the programmer will cluster similar data using an unsupervised learning algorithm and then use the existing labeled data to label the rest of the unlabeled data.

Supervised learning consists of target or outcome variable (dependent variable). That target variable is to be predicted from given independent variables (Predictors). With the help of these variables, we generate a function that maps to desired outputs. We need to train this process until we get the desired level of accuracy on training data. The algorithm modifies according to the pattern it perceives in the input and output received. Supervised Learning has two categories of algorithms: Classification and Regression.

Supervised learning always has a clear objective and can be easily measured for accuracy. The training of the machine is also tightly controlled, which leads to very specific behavioral outcomes.

On the downside, it is often very labor-intensive, as all data needs to be labeled before the model is trained, which can take hundreds of hours of specialized human effort. The costs can become astronomical. This creates an overall slower training process and may also limit the data that it can work with.

3.5. Multi-Layer Perceptron (MLP) of Artificial Neural Network (ANN)

Multi-layers perceptrons are the most important and used type of neural networks, that is a single neuron model, precursor to larger neural networks.

Neural networks learn the representation in the training data and how to best relate it to the output variable that is to be predicted. This means that neural networks learn a mapping. Neural networks have a hierarchical or multi-layered structure, that makes them very good predictors. They learn to represent features at different scales or resolutions and combine them into higher-order features.

The artificial neurons are the computational units with weighted input signals which serve as building block for neural network. They produce an output signal using a function, which is called activation function.

The figure below shows a model of a simple neuron.

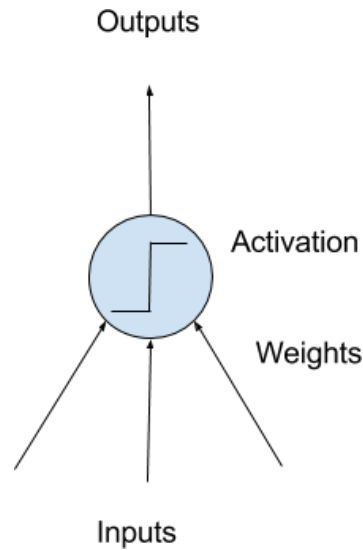


Figure 18. Model of a Simple Neuron

The activation function does a mapping of the sum of the weighted input to the neuron's output. It controls the threshold where the neuron is activated and the strength of the output.

Non-linear activation functions make possible that the network combines in complex ways the inputs and then provide a stronger capability in the functions that they model. They are also called the sigmoid function. Their output is a value between 0 and 1 with an s-shaped distribution, and the hyperbolic tangent function, called tanh outputs the same distribution over the range -1 to +1.

3.5.1. Multi-Layer Perceptron Architecture

Input or Visible Layers

The visible layer is the bottom layer that takes input from your dataset, being the exposed part of the network. Mostly, a neural network is drawn with a visible layer with one neuron per input value or column in the dataset. These pass the input value through the next layer, so they are not neurons.

Hidden Layers

Hidden layers are after the input layer and are not directly exposed to the input. The simplest network structure has a single neuron in the hidden layer which outputs the value directly. Very deep neural networks can be constructed with increases in

computing power and efficient libraries. Having many hidden layers can be described as Deep Learning. They take seconds or minutes to train using modern techniques and hardware.

Output Layer

The output layer is the final hidden layer, responsible for outputting a value or vector of values that correspond to the format required for the problem. Choosing the activation function in the output layer is constrained by the type of problem. Below, we list some points to take into consideration:

- Having a regression problem is related to a single output neuron and the neuron may have no activation function.
- Having a binary classification problem is related to a single output neuron and it can use a sigmoid activation function, that outputs a value between 0 and 1, which represents the probability of predicting a value for the class 1.
- Having a multi-class classification problem is related to multiple neurons in the output layer, one for each class. An activation function, called softmax may be used to output a probability of the network predicting each of the class values.

CHAPTER 4

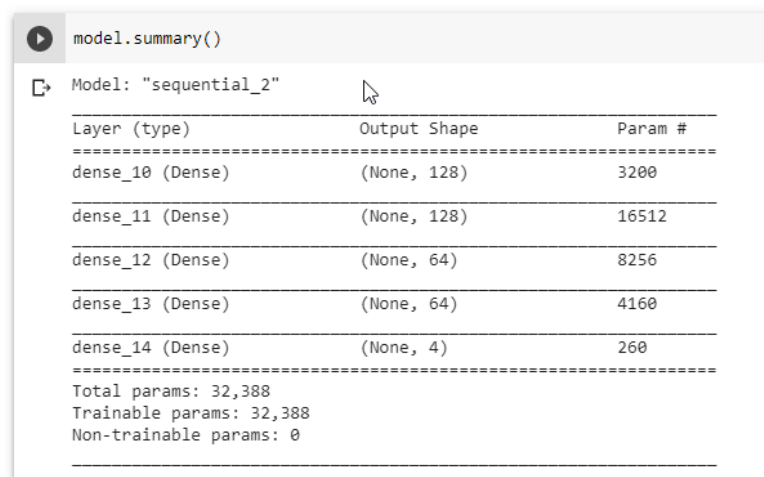
IMPLEMENTATION AND RESULTS

4.1. Experiments

In this section we will explain whats and hows of our experiments and show the main problems and solutions in this area. We firstly started by experimenting the first dataset, the one that we built ourselves using Virtual Machines and Wireshark to trace packets. This section will be divided into 2 where for each sub-section we will show the steps for each experiment.

Dataset – 1

Firstly, we built a Multi-Layer Perceptron model with the following layers and activation functions shown in figure 19. In each trial we change the epochs and batch size to see if there is a difference in the model or not.



```
model.summary()
```

| Layer (type) | Output Shape | Param # |
|------------------|--------------|---------|
| dense_10 (Dense) | (None, 128) | 3200 |
| dense_11 (Dense) | (None, 128) | 16512 |
| dense_12 (Dense) | (None, 64) | 8256 |
| dense_13 (Dense) | (None, 64) | 4160 |
| dense_14 (Dense) | (None, 4) | 260 |

Total params: 32,388
Trainable params: 32,388
Non-trainable params: 0

Figure 19. MLP first Model for dataset-1

With the model shown previously we tested the dataset 3 times with the respective batch size: 128, 64, 8. The accuracies that we received from these tests are

respectively: 81.3%, 82.4%, 82.3%. It is quite common that lowering the batch size for small datasets produces a better outcome. In this case we did not see a great optimization but still it is acceptable.

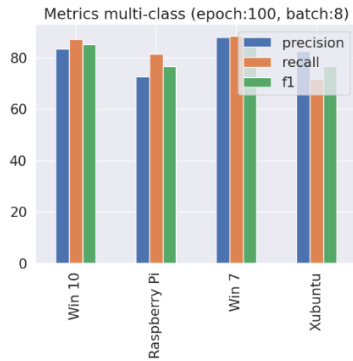


Figure 20. Precision, Recall, F1score (batch=8)

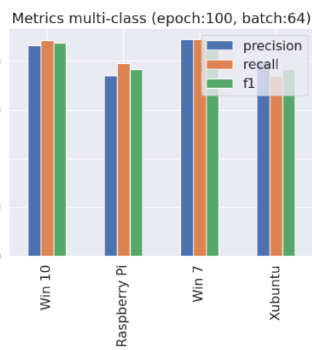


Figure 21. Precision, Recall, F1score (batch=64)

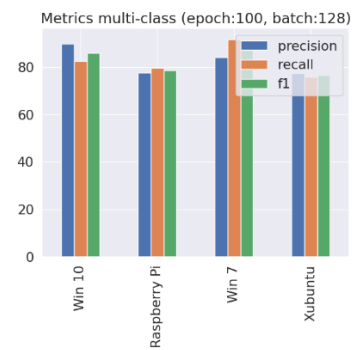


Figure 22. Precision, Recall, F1score (batch=128)

The results just for TCP/IP packet tracing are not so desirable. Along with this Accuracy and Loss curves tells us that the model does a good job on a training set but fails a bit in the test set. Which means that we have a small overfitting the data and this could be from many reasons, some of which:

- There are so many identical packets that model overfits.
- For such task 25000 rows may not be enough.
- Problems with model design.

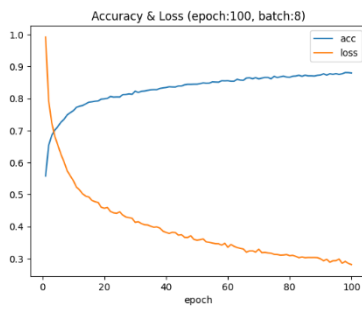


Figure 23. Accuracy and Loss (batch=8)

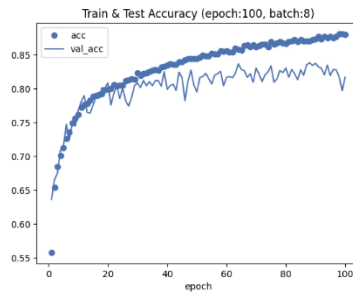


Figure 24 Accuracy Train & Test (batch=8)

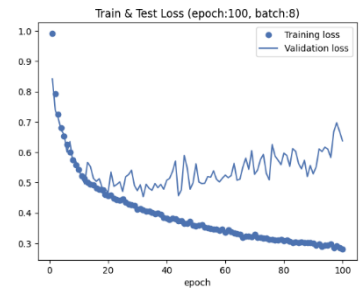


Figure 25 Loss Train & Test (batch=8)

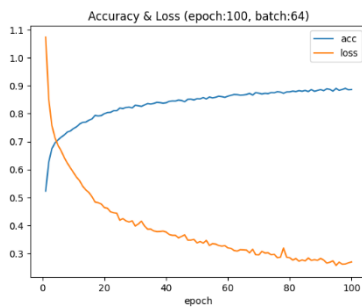


Figure 26 Accuracy and Loss (batch=64)

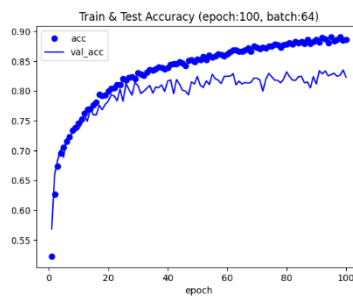


Figure 27 Accuracy Train & Test (batch=64)

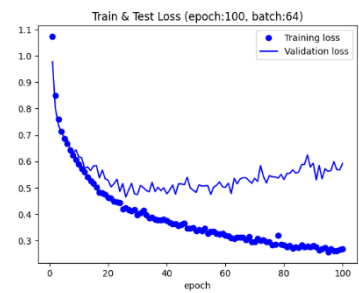


Figure 28 Loss Train & Test (batch=64)

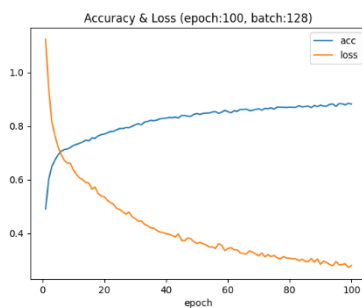


Figure 29 Accuracy and Loss (batch=128)

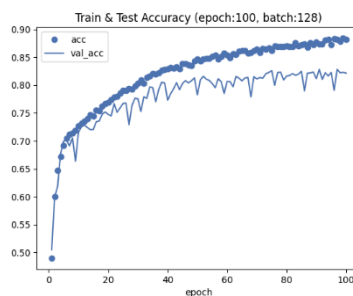


Figure 30 Accuracy Train & Test (batch=128)

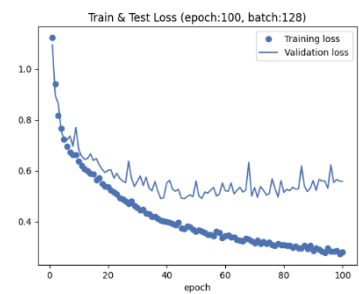


Figure 31 Loss Train & Test (batch=128)

As can be seen from the pictures above, we must improve those accuracy and loss curves between test and trains. The accuracy is not a problem because 83% for this task is quite enough due to TCP/IP features processed only. Below you can see the Receiver Operating Characteristics (ROC) for our multi-class classifier. The ROC seems to be very stable. In the figures are shown Area Under Curve (AUC) values too. For a matter of visualization, we have also shown the Confusion Matrix for each Operating System.

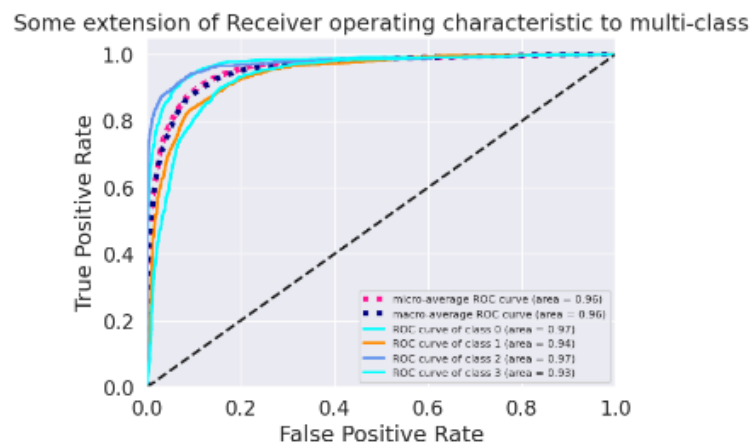


Figure 32 ROC & AUC (batch=8)

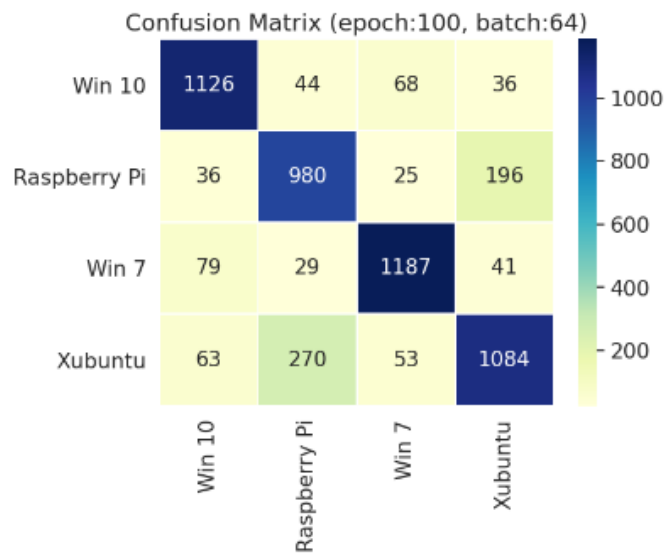


Figure 33 Confusion Matrix (batch=8)

Some extension of Receiver operating characteristic to multi-class

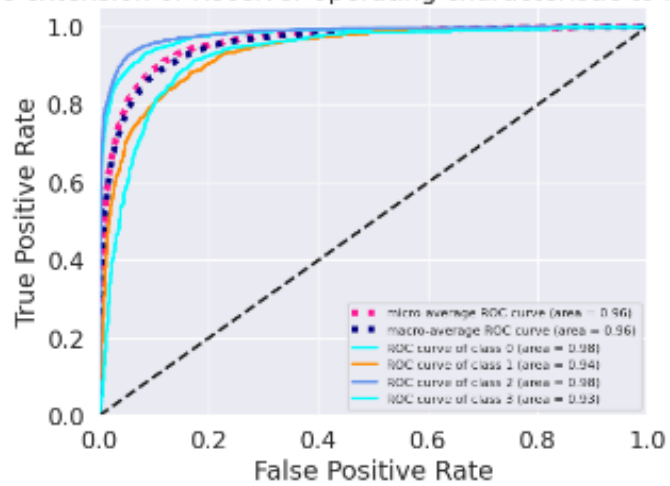


Figure 34 ROC & AUC (batch=64)

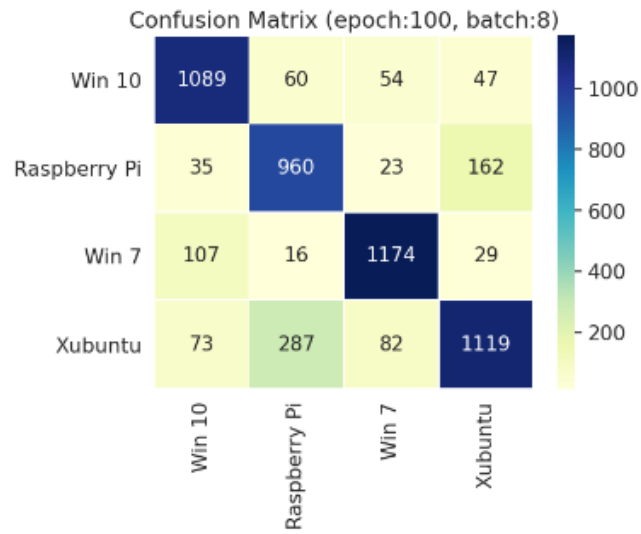


Figure 35 Confusion Matrix (batch=64)

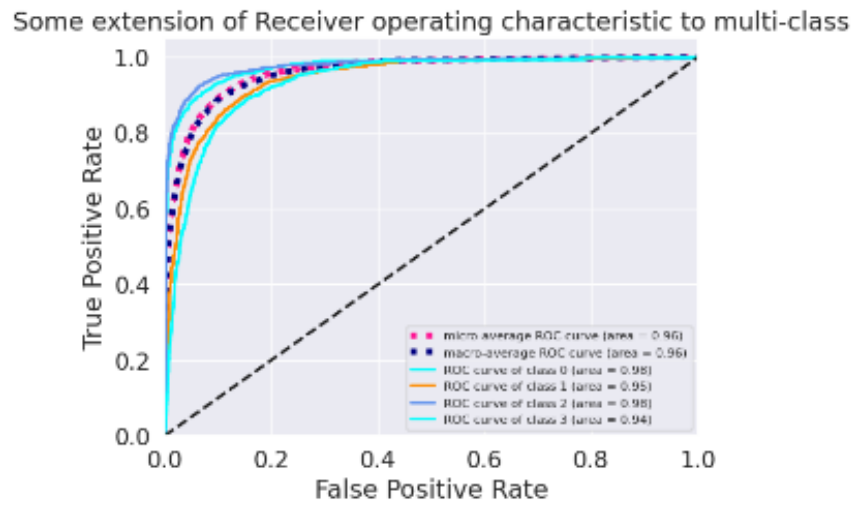


Figure 36 ROC & AUC (batch=128)

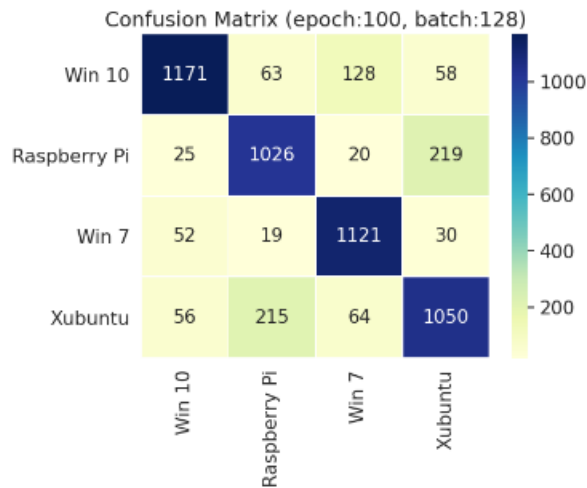


Figure 37 Confusion Matrix (batch=128)

To overcome the problem of that small overfitting of the model, we added two dropout layers. The results are quite impressive in terms of reducing over fitness but still the accuracy remains at the same levels, 83%. Figure 38 show the model with the dropout layers:

```

model.summary()
Model: "sequential_1"
-----
Layer (type)                Output Shape                Param #
-----
dense_5 (Dense)              (None, 128)                 3200
dense_6 (Dense)              (None, 128)                 16512
dropout_1 (Dropout)          (None, 128)                 0
dense_7 (Dense)              (None, 64)                  8256
dense_8 (Dense)              (None, 64)                  4160
dropout_2 (Dropout)          (None, 64)                 0
dense_9 (Dense)              (None, 4)                   260
-----
Total params: 32,388
Trainable params: 32,388
Non-trainable params: 0

```

Figure 38. New model for dataset-1 with Dropout layers

From the figures below you may see how overfitting is reduced and the train and test accuracies are almost same. This tells us that the learning curve is good and the model is stable, however is not perfect due to its accuracy.

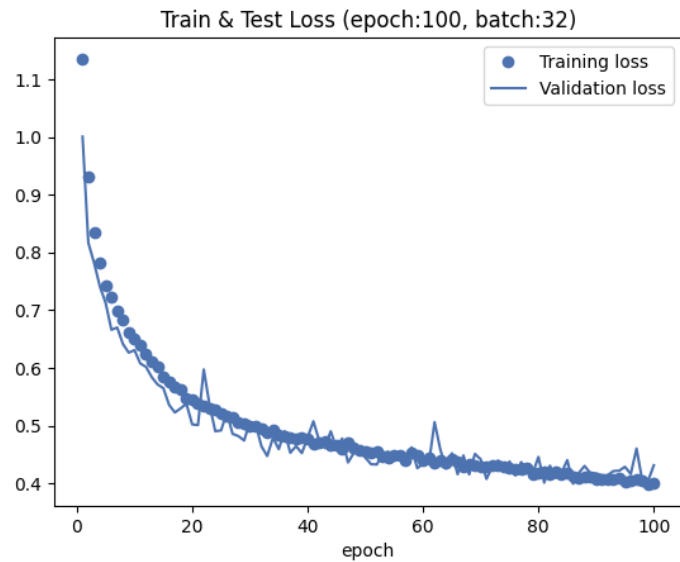


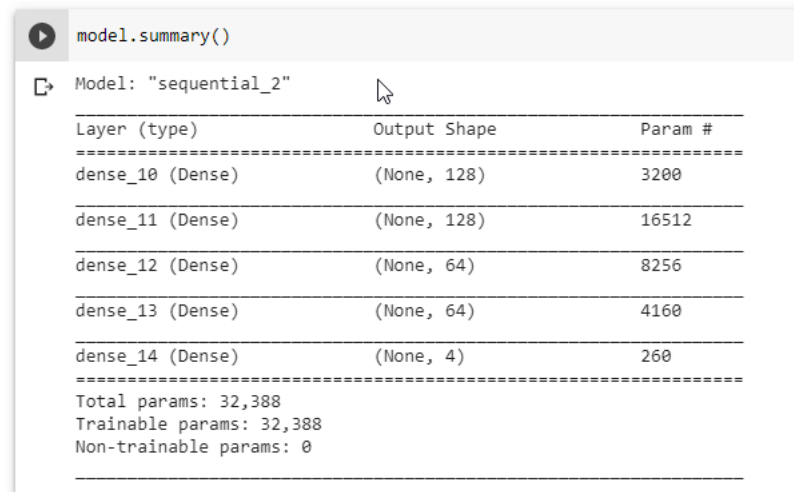
Figure 39. Loss Curve (train & test) with dropout layers



Figure 40. Accuracy Curve (train & test) with dropout layers

Dataset – 2

Same as we did in the first Dataset, we will proceed with this one too. The second dataset has features from TCP/IP, HTTP and TLS handshake. We do expect this to have a higher accuracy and to be more stable. The model is the same due to both datasets have the same number of labels (Ground truths).



```
model.summary()
Model: "sequential_2"
Layer (type)                Output Shape                Param #
-----
dense_10 (Dense)            (None, 128)                 3200
dense_11 (Dense)            (None, 128)                 16512
dense_12 (Dense)            (None, 64)                  8256
dense_13 (Dense)            (None, 64)                  4160
dense_14 (Dense)            (None, 4)                   260
-----
Total params: 32,388
Trainable params: 32,388
Non-trainable params: 0
```

Figure 41. Model for the second dataset

For this model in the second dataset, we received for each batch size 8, 64, 128 an accuracy of 93.4%, 92.6%, 92.5% respectively. Notably we can observe from these numbers that decreasing the batch size improves a bit on accuracy. Comparing these results to the first one we indeed have a better outcome. The main reason of this is the type of dataset, which contains more features than in the first one. Below in the Figures 42, 43 and 44 you can see how different scores apply for each label, one vs all scores.

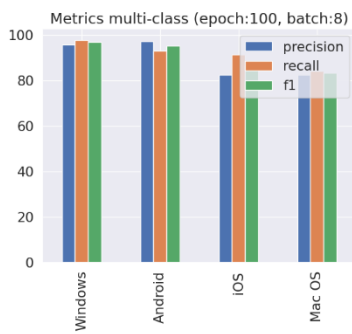


Figure 42. Precision, Recall, F1score (batch=8)

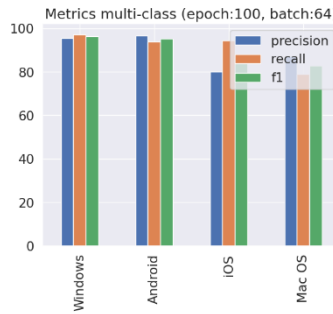


Figure 43. Precision, Recall, F1score (batch=64)

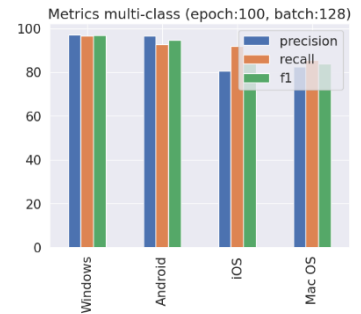


Figure 44. Precision, Recall, F1score (batch=128)

From the above figures we clearly see that Windows and Android have better scores while iOS and Mac OS are not as good as the first two. This may result from iOS and Mac OS being the Operating System of the same company and the parameters they generate for both TCP and TLS could be similar. This is a hypothesis that arouses from these results and would be nice to research on.

Different from the first dataset, accuracies and loss curves are almost same for train and test sets, which make this model perfect for the dataset. Thus, this model does not over fit in any case. Below you can see the figures that show accuracy and loss curves for 8, 64 and 128 batch size tests, respectively.

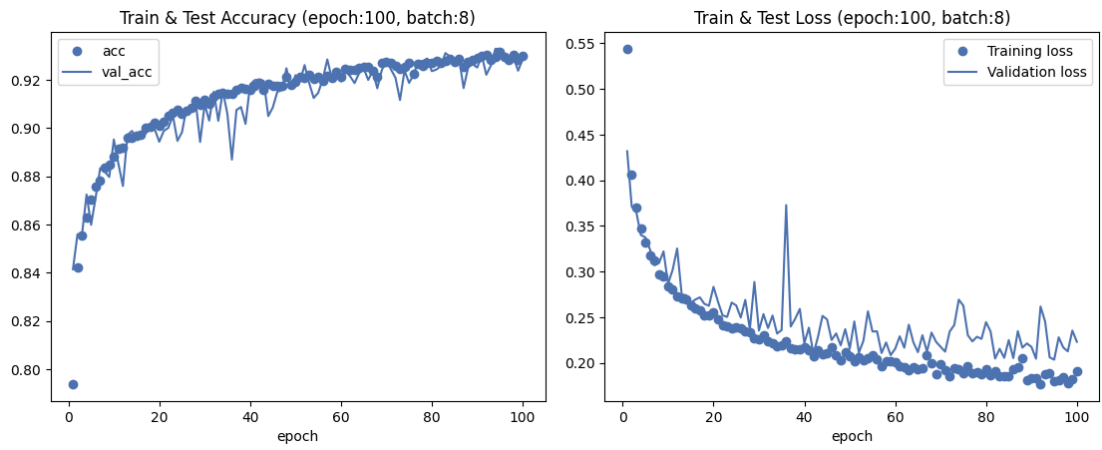


Figure 45. Accuracy and Loss Curves (train & test) with batch=8

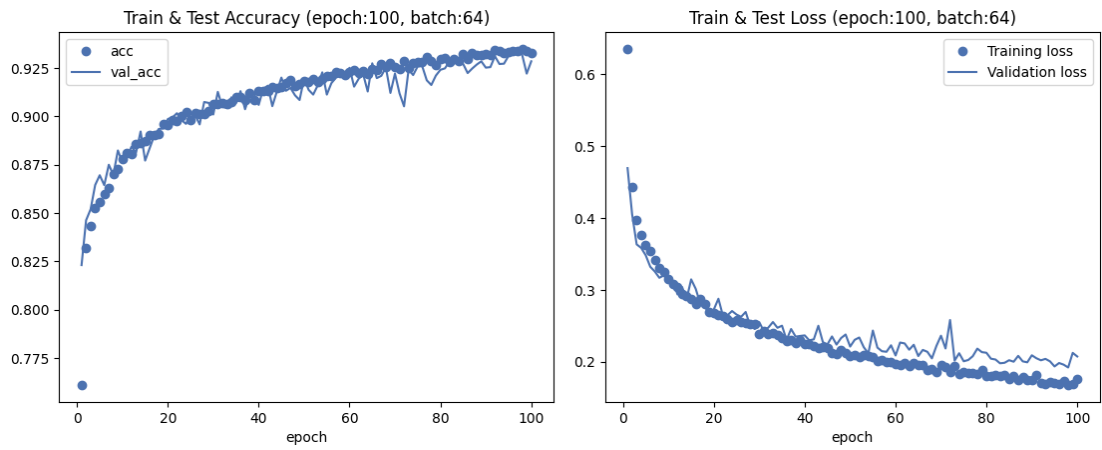


Figure 46. Accuracy and Loss Curves (train & test) with batch=64

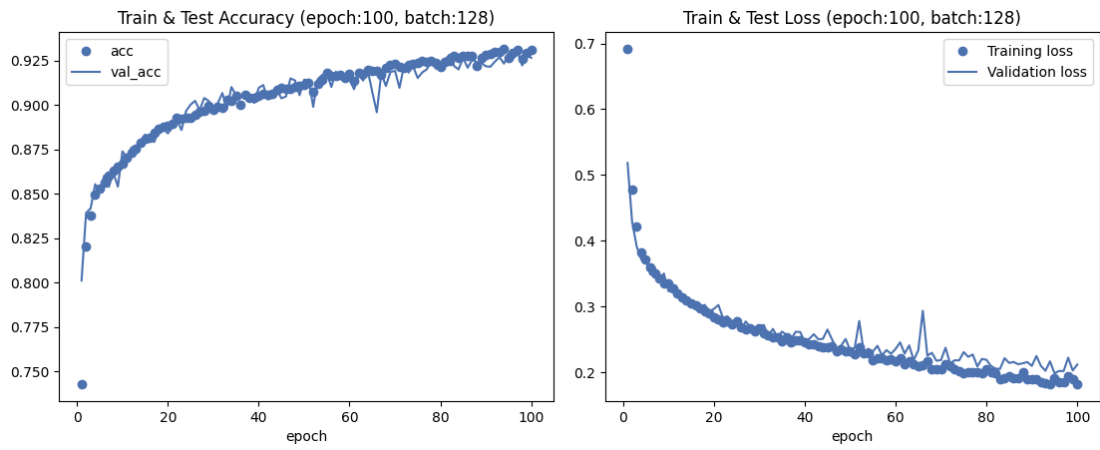


Figure 47. Accuracy and Loss Curves (train & test) with batch=128

As this model performs perfectly and we did not have any overfit, we did not make any further improvements with dropout layers.

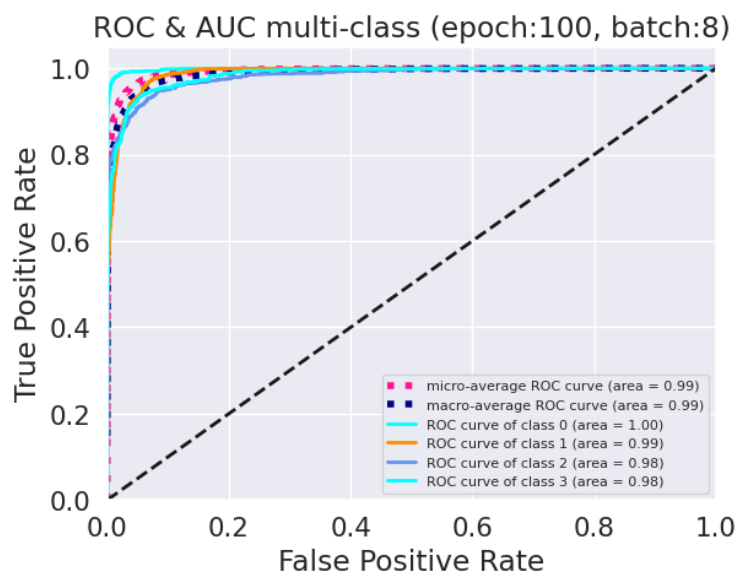


Figure 48 ROC & AUC (batch=8)

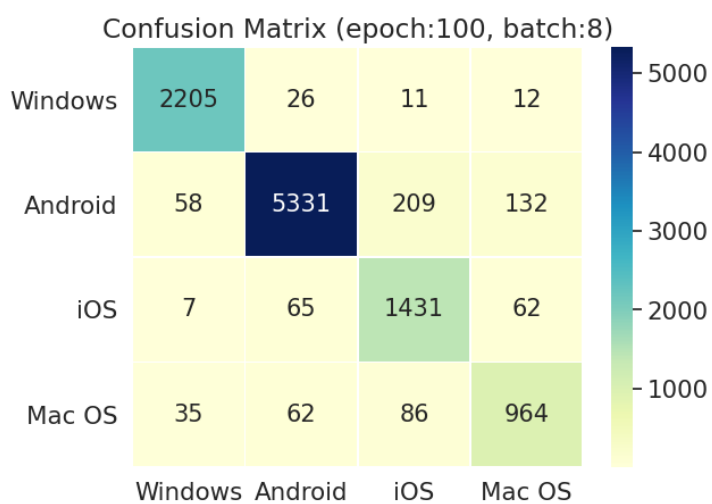


Figure 49 Confusion Matrix (batch=8)

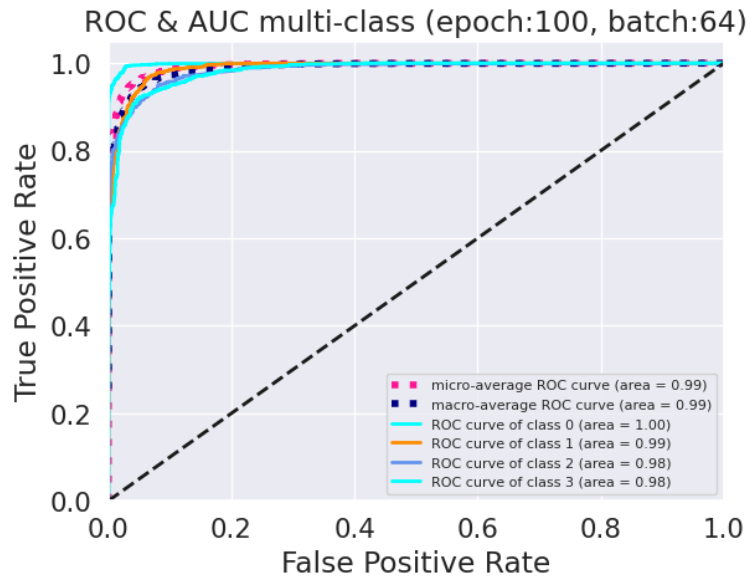


Figure 50 ROC & AUC (batch=64)

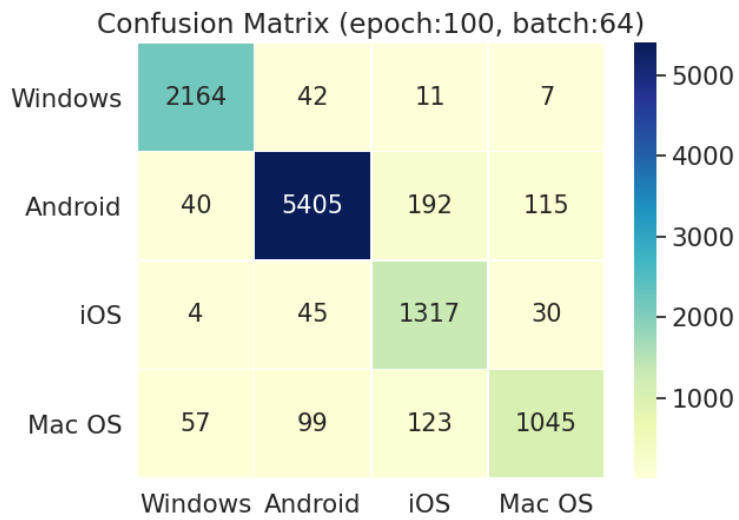


Figure 51 Confusion Matrix (batch=64)

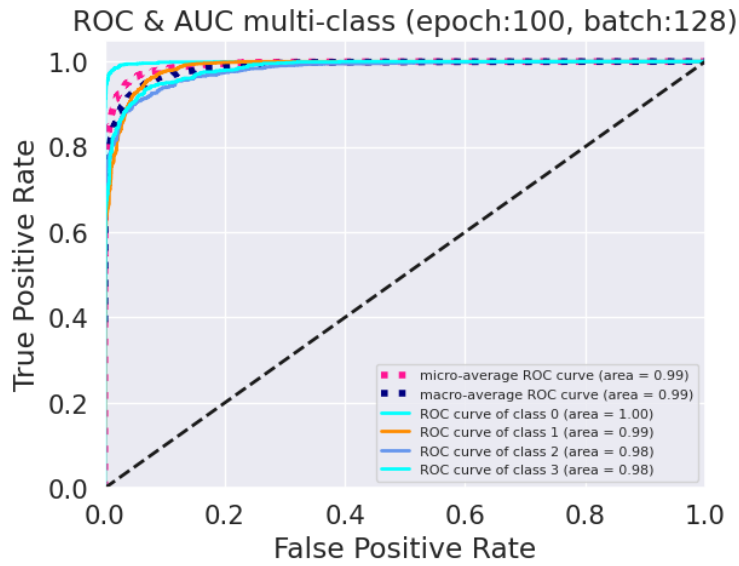


Figure 52 ROC & AUC (batch=128)

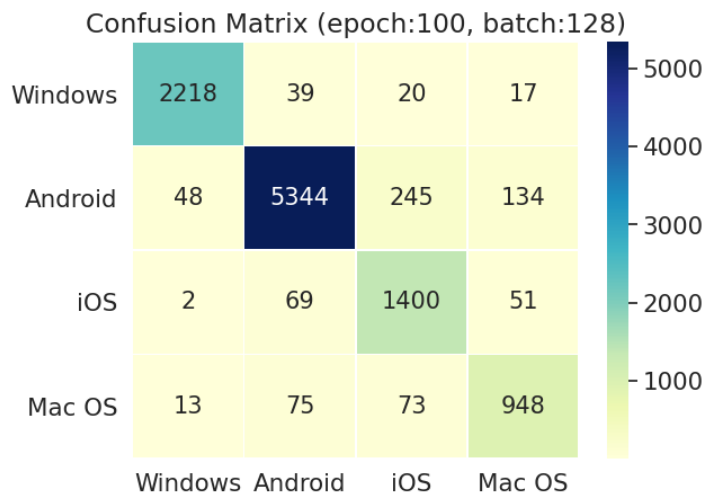


Figure 53 Confusion Matrix (batch=128)

From ROC Curves we observe that are quite near to the top left corner, which tells us that model perform very well. AUC values are 0.98 and above which is relatively a very high score.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Conclusions

In this thesis we aimed to compare the two most used types of datasets in the field OS detection using Machine Learning approach. We firstly described what's and how's of OS detection and fingerprinting and what are the basics of network communication.

In this thesis we applied Multi-Layer Perceptron, an Artificial Neural Network deep learning algorithm. Other ML algorithms may be used for this task, but as far as we have seen on testing, MLP outperforms them with at least 3-4%.

The field of OS fingerprinting is quite huge and emerging because the devices are now running different OS's and the network protocols change by time. Until now the most used parameters for OS Classification with ML are TCP/IP, HTTP and TLS parameters whereas we proved that TCP in its own cannot achieve more than 84%. At the other side, the combination of multiple parameters from multiple protocols can have a much higher accuracy, of at least 10% more the first one.

Future Work

Firstly, as a future work we suggest on researching further if there is a difference in a packet sent from wireless or wired connection. These two methods may enforce the way that OS prepares the packets and makes the connection in network.

Another task that needs a lot of effort, is building a Web Application Firewall (WAF) software that will use the model suggested in this thesis to automatically detect and not authorize some old and insecure Operating Systems.

References

- [1] A. Aksoy and M. H. Gunes, "Operating System Classification Performance of TCP/IP Protocol Headers," in *IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, Dubai, 2016.
- [2] F. Gagnon and B. Esfandiari, "A hybrid approach to operating system discovery based on diagnosis," *International Journal of Network Management*, March 2011.
- [3] Al-Shehari, Taher & Shahzad, Farrukh, "Improving Operating System Fingerprinting using Machine Learning Techniques.," in *International Journal of Computer Theory and Engineering*. 6., 2014.
- [4] Muehlstein, Jonathan & Zion, Yehonatan & Bahumi, Maor & Kirshenboim, Itay & Dubin, R. & Dvir, Amit & Pele, Ofir. , "Analyzing HTTPS Encrypted Traffic to Identify User Operating System, Browser and Application," in *IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2017.
- [5] M. Laštovička, S. Špaček, P. Velan and P. Čeleda, "Using TLS Fingerprints for OS Identification in Encrypted Traffic," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, Budapest, 2020.
- [6] G. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, 2009.
- [7] Arkin O, Yarochkin F., *Xprobe*, 2010.
- [8] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *Journal in Computer Virology*, 2010.
- [9] Joao Paulo S. Medeiros, Agostinho de Medeiros Brito Junior,, "A Qualitative Survey of Active TCP/IP," *Computational Intelligence in Security for Information Systems*, 2011.

- [10] A. Aksoy, S. Louis and M. H. Gunes, "Operating system fingerprinting via automated network traffic analysis," in *IEEE Congress on Evolutionary Computation (CEC)*, San Sebastian, 2017.
- [11] James Kurose, Keith W. Ross, *Computer Networking: A Top-Down Approach*, 6th Edition, Brooklyn: Pearson, 2013.
- [12] RFC 791, "Request For Comment," IETF, [Online]. Available: <https://tools.ietf.org/html/rfc791>. [Accessed 09 02 2021].
- [13] RFC 793, "Request For Comment," IETF, [Online]. Available: <https://tools.ietf.org/html/rfc793>. [Accessed 10 02 2021].
- [14] RFC website, "ietf.org," IETF, [Online]. Available: <https://tools.ietf.org/html/rfc793#page-15>. [Accessed 10 02 2021].
- [15] Tyagi, R., Paul, T., Manoj, B. S., & Thanudas, B., " Packet Inspection for Unauthorized OS," *IEEE Security & Privacy*, 2015.
- [16] Martin Husák, Milan Cermák, Tomáš Jirsík and Pavel Celeda, "HTTPS traffic analysis and client identification," *EURASIP Journal on Information Security*, 2016.
- [17] Bonnie Kaplan and Dennis Duchon, "Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study".
- [18] VMware, "VMware," VMware, [Online]. Available: <https://www.vmware.com/>. [Accessed 08 10 2020].
- [19] Wireshark, "Wireshark," [Online]. Available: <https://www.wireshark.org/>. [Accessed 08 09 2020].
- [20] L. D. Vita, "Github," [Online]. Available: https://github.com/lucadivit/Pcap_Features_Extraction. [Accessed 16 01 2021].

