Desktop and mobile operating system fingerprinting based on IPv6 protocol using machine learning algorithms

Saeed Salah*, Mohammad Abu Alhawa and Raid Zaghal

Department of Computer Science,
Al-Quds University,
Jerusalem P.O. Box 20002, Palestine
Email: sasalah@staff.alquds.edu
Email: moalhawa@qou.edu
Email: zaghal@staff.alquds.edu
*Corresponding author

Abstract: Operating system (OS) fingerprinting tools are essential to network security because of their relationship to vulnerability scanning and penetrating testing. Although OS identification is traditionally performed by passive or active tools, more contributions have focused on IPv4 than IPv6. This paper proposes a new methodology based on machine learning algorithms to build classification models to identify IPv6 OS fingerprinting using a newly created dataset. Unlike other proposals that mainly depend on TCP and IP generic features; this work adds other features to improve the detection accuracy. It also considers OSes installed in mobiles (Android and iOS). The experimental results have shown that the algorithms achieved high and acceptable results in classifying OSes. KNN and DT achieved high accuracy of up to 99%. SVM and GNB achieved 81% and 75%, respectively. Moreover, KNN, RF and DT achieved the best recall, precision, and f-score with almost the same as the achieved accuracy.

Keywords: operating system; fingerprinting; IPv6; network security; machine learning; mobile operating system; performance measures.

Reference to this paper should be made as follows: Salah, S., Abu Alhawa, M. and Zaghal, R. (xxxx) 'Desktop and mobile operating system fingerprinting based on IPv6 protocol using machine learning algorithms', *Int. J. Security and Networks*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Saeed Salah is an Assistant Professor and researcher at the Department of Computer Science at Al-Quds University in Jerusalem. He received his Master's in Computer Science from Al-Quds University in 2009 and PhD in Information and Communication Technologies from the Department of Signal Theory, Telematics and Communications of the University of Granada in 2015. His research interests are focused on network management, machine learning, data mining, information and network security, MANETs, routing protocols and blockchain. He published many peer-reviewed research papers in recognised international journals and conferences. Moreover, he acts as a reviewer for a number of journals in his field.

Mohammed Abu Alhawa is a Master student at the Department of Computer Science at Al-Quds University in Jerusalem. He received his BSc in Information Technology from Al-Quds Open University in 2010. Currently, he is working as a Lecturer, Network Administrator and Professional Diploma Coordinator at Al-Quds Open University. His research interests include computer networks, network security, IPv6 security and machine learning.

Raid Zaghal received his PhD in Computer Science from Kent State University in Ohio, USA in 2005 and Master's in Computer Science from the US University in Washington DC in 1996. Currently, he is an Assistant Professor, Lecturer and researcher at the Computer Science Dept at Al-Quds University (since 1996). His research interests are in network theories and protocols' design, routing protocols, MANETs, cloud computing and mobile applications. In the past 13 years, he has advised more than 20 Master students on a number of research projects in these fields and published many articles in international conferences and journals.

1 Introduction

Operating system (OS) fingerprinting method is the process of identifying OS which is installed in computer machines based on signatures or behaviours that are exclusively generated by it. This method is relevant to network security because of its relationship to penetration testing, tailoring of exploits, vulnerability scanning and network inventory. Thus, determining the OS is useful for different purposes:

1 Identifying OS helps to recognise potential vulnerabilities that might make the OS exposed to different attacks.

2 It might help in discovering, managing, controlling and securing network resources to identify systems that might be vulnerable to be secured from any possible attacks (Schwartzenberg, 2010).

3 Identifying OS inside a network provides the system administrator with the necessary information about any unpatched or unauthorised machine that might be connected to the network (Elejla et al., 2017).

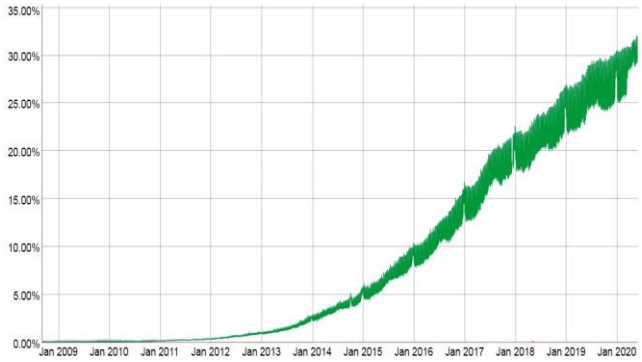
4 OS fingerprinting improves the administration process by giving the administrators a better overview of the network OS.

5 OS fingerprinting classifications allow the administrator to monitor and apply policies on OS types, versions and patch of the machines inside the network (Matoušek et al., 2014).

Internet protocol (IPv6) has been designed to eventually replace IPv4 due to the IPv4 problem of addresses exhaustion. A significant number of IPv6 users increases daily. Figure 1 shows the percentage of users that access Google services over IPv6 (Google IPv6 Statistics, 2020). It is shown that more than 30% of the total number of users that reached Google servers use IPv6 protocol. IPv4 has been sufficiently studied in terms of security and network management by many researchers. OS fingerprinting proposals coming from the IPv4 traffic are unable to classify IPv6 due to the structural difference between both protocols. Differences in the technical implementation of well-known internet protocols (IP and TCP) make it possible to identify the OS of a remote host by the generic characteristics of its TCP and IP protocols' headers, even in the absence or lack of application-layer information. Therefore, various techniques of OS fingerprinting over IPv4 traffic have been in use for over a decade; however, IPv6 OS fingerprinting has had comparatively scant attention in both research community and private sector. In the authors' best knowledge, very little work was proposed to provide the service of IPv6 OS fingerprinting classification. Moreover, these proposals suffer from low classification accuracy due to the used non-qualified features (fingerprints) or being exposed to be blocked by the network security systems. Some existing techniques have drawbacks related to negatively affecting the network performance due to their used probe (induction) packets (Elejla et al., 2017).

Machine learning is a promising topic that has been applied in several proposals for the purpose of OS fingerprinting area such as those cited in Fifield et al. (2015), Ordorica (2017) and Schwartzenberg (2010) and it is proven to be an effective method for classifications. Machine learning provides computers the ability to learn the behaviours without being explicitly programmed (configured). Moreover, the human role-based system cannot accurately cover (learn) all the scenarios and behaviours that might exist. However, machine learning can learn from the input data to build a classification model that accurately classifies any future data (Al-Shehari and Shahzad, 2014). Also, it can be retrained when fingerprints are updated or changed. Besides, despite the fact that the number of mobile devices is increased daily and reached 7.3 billion devices worldwide (https://www.statista.com/), a few of the exiting tools consider OS fingerprinting for mobile OSes such as Android and *iOS*. The generic methodology mainly depends on a set of features that will be extracted from the IPv6 packets to be fed to the machine learning algorithm. The chosen features can be used as a predefined and reference set of features for classifying OSes using different methods.

Figure 1 The percentage of internet users that access Google services over IPv6 (see online version for colours)



This paper proposes a novel approach to identify IPv6 OS fingerprinting based on IPv6 traffic that is generated from the end-user devices. The main contribution of this work is three-fold:

1 Unlike other OS fingerprint proposals that mainly depend on TCP and IP generic features for detecting the OS, this work adds other features to improve the detection accuracy.

2 The scope of this work is extended beyond the classification of OSes that are installed in desktop and laptop (such as Windows and Linux), it also considers OSes that are installed in mobiles and tablets (such as Android and *iOS*).

3 Since IPv6 suffers from lack of such datasets that can be used for OS fingerprinting purposes, due to privacy issue of the IPv6 information (such as IPv6 address and prefix), in this work a newly-created dataset was used and will be available for other researchers who conduct relevant works.

Besides the introduction section, this paper contains the following sections. A review of the related work is presented in Section 2. Section 3 details the proposed methodology. The dataset and the preprocessing steps are summarised in Section 4. The evaluation measures along with the experimental results are presented in Section 5. Finally, in Section 6, we conclude this research work and shed light to some future works.

2 Related work

OS fingerprinting techniques are divided into two categories: passive and active. The passive techniques are silent techniques that depend on capturing and analysing the normally generated traffic from the hosts. This traffic can be any request or response packets such as the SYN or SYN/ACK packets that use the TCP handshake mechanism. These techniques do not probe the target host to generate the needed traffic. Therefore, they might not find all the necessary packets that are needed for identifying the OS under consideration. Besides, they do not add any extra overload to the network and they perform the detection faster than the active mechanisms because they do not wait for the hosts' responses. On the other hand, the active OS fingerprinting techniques use probes packets that are designed to induce the machines to reply with specific responses. They have the disadvantage that this process triggers the OS to generate the needed traffic which might be noisy to the network. Moreover, this traffic is exposed to be blocked from the network firewall or intrusion detection system (IDS) which might negatively affect their accuracy (Elejla et al., 2017; Aksoy et al., 2017).

Both active and passive techniques use values from the packets to be compared against a set of rules or models to determine the generating OS. These values are called discriminating features that are extracted from the traffic layers headers (Nerakis, 2006). These features are considered the key factor that is used to build the classification model. OS fingerprinting classification depends on the assumption that each OS has different values for some of these features. For example, Windows 7 assigns a value of 128 to time to live (TTL), unlike Linux Red Hat 9 chooses the value of 64 (Siby, 2014).

IPv4 fingerprinting tools work well with IPv4 networks as they are proposed for such tasks. However, they should adapt their used mechanisms to support IPv6 traffic. Some IPv4 fingerprint tools that do not depend on the network layers (other layers fields) might support IPv6 traffic with slight modifications. Therefore, IPv4 OS fingerprinting tool cannot be directly applied to IPv6 due to the changes between the two protocols. To our best exploration, few of the exiting tools have supported OS fingerprinting using IPv6 protocol. Next, we summarise the most relevant works.

Some research efforts have paid attention to OS fingerprinting problem in IPv6 protocol. These efforts are either upgraded versions from IPv4 tools to support IPv6 protocol by considering the new IPv6 features, or exclusively proposed tools for IPv6 network. These proposals are organised into active and passive techniques based on the classification mechanisms they adopted. Elejla et al. (2017), Nerakis (2006), Schwartzenberg (2010) and Stopforth (2007) have provided a good summary of IPv6 OS fingerprinting tools and explained their main advantages and disadvantages.

NMAP (Lyon, 2009) is an active OS fingerprinting tool that uses 18 probe packets (TCP, UDP, ICMP) that support IPv6 user traffic and the hosts' responses are compared to the NMAP database of OSes signatures and the closest match is chosen. SinFP (Auffret, 2010) is a mix of active and passive tools that send three TCP probe packets with sharable signatures database and compare the results. In addition to the drawbacks of NAMP, this tool has high inaccuracy of OS detection and it does not support mobile OS fingerprint classification. These tools might be misclassified as attack activities due to their probe packets; they negatively affect the network availability, they depend on a small database of IPv6 traffic, they do not support auto-detection of IPv6 mobile OS fingerprint classification and they have low accuracy (Elejla et al., 2017). And finally, they cannot determine the OS unless there is at least one open port in the host machine (Matoušek et al., 2014).

Beck et al. (2007) proposed a tool called osfinger6; it has built-in OS fingerprint active mechanisms for IPv6 neighbour discovery protocol (NDP) that use 156 probe packets (forged NS). Based on the observations of the OSes responses, a decision tree of the available OSes is built. This tool does not support the recent changes of IPv6 extension headers, it does not support mobile OS fingerprint classification, it depends only on the response of the ICMPv6 NS packets and it was mainly designed for small testbed used for validating the tool itself.

P0f (Zalewski, 2012) is a passive tool that analyses nine TCP features. It extracts header information from TCP packets to compare them with a database of signatures for OS classification. Despite that it can detect traffic behind firewalls and network address translation (NAT) systems, this tool does not support auto-detection of IPv6 and mobile OS fingerprint classification and most of the newer OSes cannot be classified at the version level.

Fifield et al. (2015) proposed an active tool that uses 154 crafted probe packets for large datasets. When the intended hosts respond, those response packets will be analysed for features extraction. Ordorica (2017) is a passive tool that uses six features. These features are fed into the neural network and random forest classifiers. Since, the number of used features is very small, this tool does not support mobile OS fingerprint classification and does not use non-qualified features such as transport layer. The passive technique has not been sufficiently studied in IPv6 as only p0f (Zalewski, 2012) and Ordorica that have used this technique up to this time.

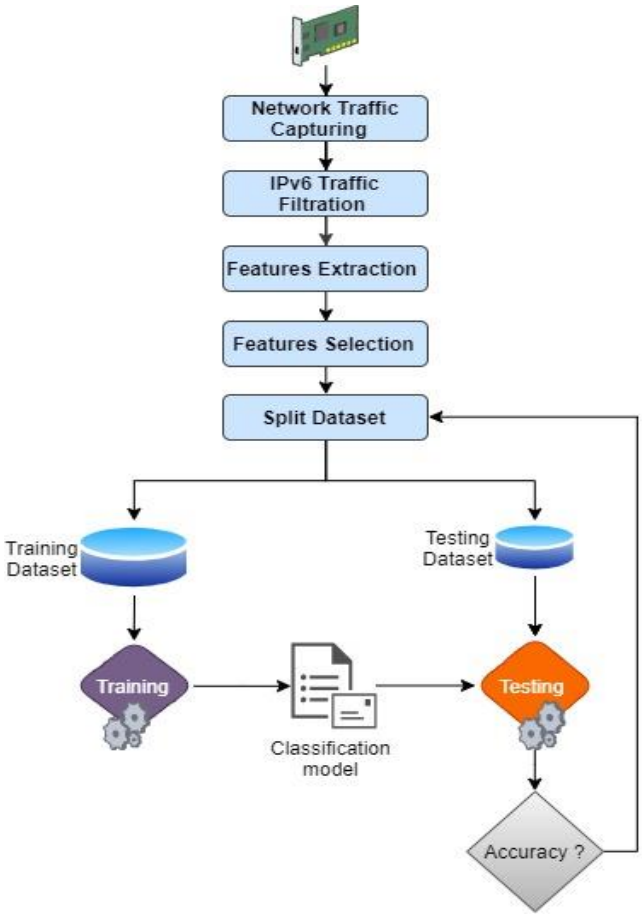
In summary, most of the existing tools suffer from several technical issues such as they might be misclassified as attack activities due to their probe packets, some of them negatively affect the network availability, others depend on a small database of IPv6 traffic and some others do not

support auto-detection of IPv6 OSes mobile OS fingerprint classification. Furthermore, some of them have low detection accuracy and cannot determine the OS unless some TCP port configurations are preconfigured on the host machine.

3 Methodology

The methodology of the proposed mechanism is designed to combine two different steps to achieve the research goals. In the first step, an IPv6 user traffic is captured, collected, processed and stored in a database. The second step is responsible for training and validating the machine learning algorithm to build a classification model. The classification accuracy, precision, recall and F-score will be used as the metrices to evaluate the goodness of the built model. In Figure 2 shows the details of the architecture of the proposed methodology.

Figure 2 The proposed methodology which was followed to achieve the main objective of this work (see online version for colours)



As shown in Figure 2, the methodology starts by capturing the traffic, then it keeps only traffic belonging to IPv6. Next, the traffic is prepared by extracting and selecting the best set of features then splitting the dataset into training and testing parts. The training part will be fed to the machine learning algorithm to be trained on them and build a classification model. The model is evaluated on the testing part to calculate its accuracy and be retrained until a satisfying accuracy is achieved. The proposed methodology steps are explained in detail in the following subsections.

3.1 Network traffic capturing

The role of the passive OS classification is to monitor the whole traffic of the network without affecting any of the network nodes. The proposed methodology aims to build a learning-based model to classify OSes in the network based on their generated traffic. Therefore, the methodology starts by capturing the network traffic needed for the classification. This stage is designed to capture the network traffic and prepare it to be in a suitable format as well as free of noisy traffic. The output of the stage is a PCAP (packet capture) file of packets that are sent or received by any of the network nodes.

3.2 IPv6 traffic filtration

Filtering traffic is an important step due to the processing complexity that can be added by the unneeded packets. Moreover, including such packets might confuse the applied machine learning algorithm and leads a decrease in the classification ability. Since this research interest is focusing on IPv6 network, IPv4 packets are filtered out from the network traffic. The output of this stage is a PCAP file of IPv6 packets that will be fed as inputs to the next stage.

3.3 Features extraction

The function of this stage is to extract the needed discriminating features from the received PCAP file. These features are chosen as carrying differentiative information based on the author's domain knowledge of the field, studying relevant works and conducting empirical experiments. The chosen features are selected based on justifications of their selection and are claimed to contribute to accurately classifying the OSes. The output of this stage is a new set of features that have been extracted and derived from the traffic. Each of them has different values in each OS. To this research, we assume some header fields to be potential features for IPv6 OSes fingerprinting classification. These features will be inspired by different sources such as related works, dataset investigation, the author's domain knowledge and empirical experiments. These potential features are assumed to carry unique values for each generation of the OS. Table 2 shows the set of the discriminative features that are assumed to accurately differentiate the behaviour of each OS.

3.4 Features selection

This stage is the post-stage of building the classification model using the machine learning algorithm. Although each feature has been chosen based on a reasonable justification and assumption, some of them might be redundant or not strongly related to the classification process. Therefore, the

features need to be evaluated and only the most contributed features to the classification will be selected. This stage aims to unselect any non-contributed features to avoid extra processing overhead or any packet misclassification. Moreover, the feature selection helps to reduce the training times that are needed by the machine learning algorithm to build the model. To choose the best features that can distinguish OSes, a feature ranking algorithm was applied to the dataset. Features that get the highest-ranking score will be selected and extracted from the traffic for further processing. The traffic will be prepared and represented only with these features and fed as inputs to the next stage.

3.5 Split the dataset

This stage is needed to prepare the training and testing datasets for the algorithm. The classification process requires two separate datasets: one for training and one for testing. The splitting mechanism splits the dataset into two parts: one part of the data to train the model (training dataset) then uses a different part (testing datasets) to evaluate the accuracy of the trained model. Given the large size dataset, it is randomly split into the two parts: 80% of the packets were used in training and 20% were used for testing, while maintaining the original class distribution in both the parts.

3.6 Applying machine learning algorithms

In this stage, a machine learning algorithm is chosen to be applied to the dataset to be trained on the available OSes data. The cross-validation testing mechanism is used by training the algorithm on 80% of the dataset. Next, a testing phase was done for their trained models on the rest of the dataset (20%). The process chose different 20% from the dataset for testing and the rest (80%) was chosen for training. This stage aims at proving that the chosen features can differentiate between the different OSes. The training and testing processes will continue until a trustworthy classification model is built. Also, parameter tuning was applied to get the best possible model in terms of classification accuracy and the other evaluation metrics. This stage output is a trustworthy trained model that can operate online to classify the OSes in the network.

4 Dataset: data collection and processing

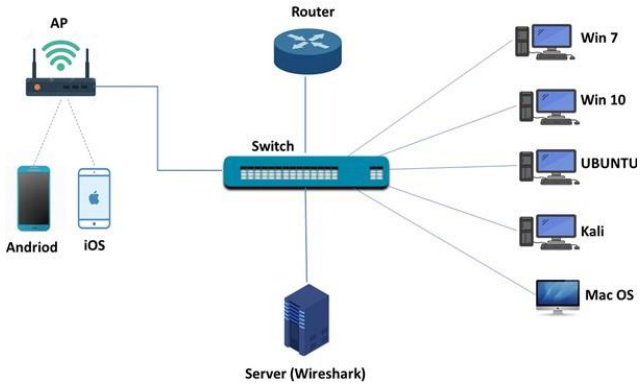
As previously mentioned, a dataset is a core need to develop passive OS fingerprinting techniques. OS fingerprinting techniques use the datasets for different goals like:

1 investigating the traffic to choose the features (fingerprints) that differentiate an OS from others

2 building, evaluating and conducting a comparison amongst the different models.

However, IPv6 suffers from a lack of such datasets that can be used for OS fingerprinting purposes. The reason behind this lack is linked with a privacy issue of the IPv6 information (such as IPv6 address and prefix). This information might be used by the attacker and expose the used network to outside attacks. Some researchers (Richardson et al., 2010) suggested the use of encryption or mapping techniques to solve the security breaches of using IPV6 dataset, but these techniques are very limited in scope and scalability.

Figure 3 The network topology used for network traffic collection (see online version for colours)



There is no existing benchmark datasets or well-known features that can be considered as a reference to all IPv6 OS fingerprint techniques. Nevertheless, we created a dataset to achieve the purpose of this work and to select potential features that might be used as fingerprints. To create the dataset to work with, we implemented a network topology at the department computer labs. The topology is shown in Figure 3. During the data collection process, we adhered to the five specifications that have been suggested by Sperotto et al. (2010) to have a reliable and good dataset. We considered the following specifications when collecting the network traffic:

1 the dataset should be real and reflects the actual circumstance of the OSes

2 the dataset traffic should be diverse and includes all the possible scenarios of the traffic

3 the dataset records should be fully and correctly labelled

4 the dataset should be balanced between the various OSes under consideration

5 the features should be relevant and represent the differences between the OSes.

To fulfil these requirements, the network traffic has been collected using *Wireshark* sniffing tool (Nath, 2015) for a period of three months (10/1/2020 to 9/4/2020). This traffic was captured from a real network to satisfy the first requirement to have a reliable dataset. Also, the capturing

period has been chosen long enough to ensure that the different network traffic scenarios have been included. Moreover, to ensure the diversity of the scenarios, the collection was done at different times with different users and applications. The network has been created and configured with 62 devices with different OSes that are connected using a switch. The switch is connected to the rest of the university campus intranet with internet access. Each of the devices has preconfigured with a known IPv6 address to be used later for the data labelling process. The diversity of the used OSes supports and emphasises the second requirement of a good dataset. Table 1 shows a description of the collected traffic in terms of the number of devices and the number of captured packets.

Table 1 The software specifications of the end-user devices that relate to the network topology

Type of OS	*OS version*	*# of devices*	*# of packets*
Windows	10	12	108,396
Windows	7	9	108,562
Kali	GNOME 3.30.2	5	36,003
Mac	OS X El Capitan 10.11	5	27,031
Ubuntu	18.4	5	36,305
iOS	Various versions	15	43,596
Android	Various versions	11	40,107

One of the devices has been set to promiscuous mode to be used as a master device to collect other devices' traffic. The switch port of this device is configured with a mirror port mode. This mode forces the switch to collect the device's traffic and sends a copy of any packet that has been sent or received by other devices to the promiscuous mode device.

The *Wireshark* has been configured to filter out IPv4 traffic and keep only IPv6 traffic which is the scope of this research. The collected traffic is exported as a *CSV* file for further processing.

The *CSV* file is exported with the specified potential features which are expected to contribute to the OS classification process. The potential features are assumed to carry a unique value for each generating OS. The chosen features are selected based on justifications of their selection and are claimed to contribute to the process of classifying the OSes accurately. Table 2 shows the discriminative features (fingerprints) that are assumed to accurately differentiate the behaviour of each OS.

It is worth noting that, each of the potential features has been used in IPv4 OS fingerprinting using its similar field values. In addition, each of them has been suggested or used by research proposals for IPv6 OS fingerprinting purposes. For example, ICMPv4 has been used for OS fingerprinting in many IPv4 OS fingerprinting research such as Arkin et al. (2003), Taleck (2004), Jiang et al. (2003) and Arkin (2000). Moreover, ICMPv6 has been already used as probe packets in active IPv6 OS fingerprinting techniques (Fifield et al., 2015; Beck et al., 2007; Eckstein and Atlasis, 2011). Therefore, ICMPv6 was used to extract some potential features from it. Moreover, The IPv6 source address is used as the criteria to label the traffic. Knowing the IPv6 source address determines the OS that generates the packets. A new column called OS is added to the *CSV* file to include the OS of each packet. The process of labelling the records satisfies the third condition of reliable dataset requirements.

5 Experimental results and discussion

5.1 Evaluation measures

Several generic evaluation measures are used to evaluate the machine learning algorithms. In this work, we focus on the most common ones, specifically we consider accuracy, precision, recall, F-score and training time. The first three measures can be computed using the confusion matrix as shown in Figure 4. Referring to this figure:

- *True positive (TP)* is defined as the number of correct predictions, i.e., positive class correctly identified as positive.
- *False negative (FN)* is the number of incorrect predictions, i.e., positive class incorrectly identified as negative.
- *False positive (FP)* is the number of incorrect predictions, i.e., negative class incorrectly identified as positive.
- *True negative (TN)* is the number of correct predictions, i.e., negative class correctly identified as negative.

The training time is the number of seconds needed by the algorithm for training on the training dataset to build the classification model and the F-score is the weighted harmonic mean of precision and recall. It can be represented mathematically as:

$$F\text{-}score = 2 \times (recall \times precision)/(recall + precision)$$

Kaggle Data Science Platform (https://www.kaggle.com) was used to further preprocess the data. A new Kaggle kernel has been created to write and apply python scripts to the data. The *CSV* file is uploaded and stored in the kernel. By plotting the histogram of the available OSes records, the number of OS is not balanced as shown in Figure 5. To balance the dataset, resampling techniques *up_sample* is used to have a balanced number of packets for each OS. The resampling techniques randomly duplicate packets belonging to OSes with smaller values to have the same number of packets to reach the max number of packets belonging to an OS with highest value. By investigating the OSes, Windows 10 has the highest number of packets. Therefore, other OSes packets are resampled to have the same number of packets to meet this requirement.

Table 2 The potential set of discriminative features (fingerprints)

Feature	*Justification*	*Suggested by*
Protocol	Helps to identify packet protocol types.	
Length	Similar to IPv4 total length IPv4 OS fingerprinting such as Arkin et al. (2003), Lippmann et al. (2003) and Schwartzenberg (2010)	Eckstein and Atlasis (2011), Fifield et al. (2015) and Nerakis (2006)
Hop Limit	Similar to IPv4 TTL IPv4 OS fingerprinting such as Arkin et al. (2003), Auffret (2010), Beck et al. (2007), Lippmann et al. (2003) and Schwartzenberg (2010)	Auffret (2010), Eckstein and Atlasis (2011), Fifield et al. (2015) and Nerakis (2006)
Traffic class	Similar to IPv4 Type of Service IPv4 OS fingerprinting such as Arkin (2000), Arkin et al. (2003), Auffret (2010), Lippmann et al. (2003) and Nerakis (2006)	Eckstein and Atlasis (2011), Fifield et al. (2015) and Nerakis (2006)
Payload length	Similar to IPv4 payload length IPv4 OS fingerprinting such as Arkin et al. (2003), Lippmann et al. (2003) and Schwartzenberg (2010)	Eckstein and Atlasis (2011), Fifield et al. (2015) and Nerakis (2006)
Flow label	Similar to IPv4 ident. field IPv4 OS fingerprinting such as Arkin (2000), Arkin et al. (2003), Auffret (2010) and Taleck (2004)	Eckstein and Atlasis (2011), Fifield et al. (2015), Nerakis (2006), Schwartzenberg (2010) and Stopforth (2007)
TCP Window size	Has a unique value for each OS including mobile OS (Nerakis, 2006; Chen et al., 2014). IPv4 OS fingerprinting such as Aksoy et al. (2017), Al-Shehari and Shahzad (2014), Auffret (2010) and Lippmann et al. (2003)	Eckstein and Atlasis (2011), Fifield et al. (2015), Lyon (2009) and Nerakis (2006)
TCP flags	IPv4 OS fingerprinting such as Aksoy and Gunes (2016), Auffret (2010), Chen et al. (2014) and Matoušek et al. (2014)	Fifield et al. (2015), Gagnon and Esfandiari (2011) and Lyon (2009)
TCP options	IPv4 OS fingerprinting such as Aksoy and Gunes (2016) Auffret (2010), Brinley et al. (1960) and Chen et al. (2014)	Fifield et al. (2015) and Lyon (2009)
TCP options length	IPv4 OS fingerprinting such as Aksoy and Gunes (2016), Auffret (2010), Brinley et al. (1960) and Chen et al. (2014)	Fifield et al. (2015) and Lyon (2009)
TCP initial sequence number	IPv4 OS fingerprinting such as Prigent et al. (2010) and Schwartzenberg (2010) and Taleck (2004)	Fifield et al. (2015) and Lyon, 2009)
TCP Window scale	IPv4 OS fingerprinting such as Aksoy and Gunes (2016), Auffret (2010), Brinley et al. (1960) and Chen et al. (2014)	Fifield et al. (2015) and Lyon (2009)
TCP max segment size	IPv4 OS fingerprinting such as Aksoy and Gunes (2016), Auffret (2010), Brinley et al. (1960) and Chen et al. (2014)	Fifield et al. (2015) and Lyon (2009)
HTTP user-agent	Browsers include some information inside the user-agent field that helps in OS classification (Anderson and McGrew, 2017)	
ICMPv6 type	Similar to ICMPv4 type IPv4 OS fingerprinting such as Arkin (2000), Arkin et al. (2003), Jiang et al. (2003) and Taleck (2004)	Eckstein and Atlasis (2011), Fifield et al. (2015), Beck et al. (2007) and Nerakis (2006)
ICMPv6 identifier	Similar to ICMPv4 identifier IPv4 OS fingerprinting such as Arkin (2000), Arkin et al. (2003), Jiang et al. (2003) and Taleck (2004)	Eckstein and Atlasis (2011) and Nerakis (2006)
ICMPv6 sequence	Similar to ICMPv4 sequence IPv4 OS fingerprinting such as Arkin (2000), Arkin et al. (2003), Jiang et al. (2003) and Taleck (2004)	Eckstein and Atlasis (2011) and Nerakis (2006)
Delta time	Used in IPv4 OS fingerprinting (Boukhtouta et al. (2013)	
DHCPv6 Lifetime	Similar to DHCPv4 Lifetime IPv4 OS fingerprinting (Kollmann, 2007; Zalewski, 2012)	Kollmann (2007) and Ordorica (2017)
DHCPv6 vendor class	Similar to DHCPv4 Vendor Class IPv4 OS fingerprinting (Kollmann, 2007; Zalewski, 2012)	Kollmann (2007) and Ordorica (2017)
DHCPv6 client identifier	Similar to the DHCPv4 Client Identifier IPv4 OS fingerprinting (Kollmann, 2007; Zalewski, 2012)	Kollmann (2007) and Ordorica (2017)
DHCPv6 options length	Similar to DHCPv4 options length IPv4 OS fingerprinting (Kollmann, 2007; Zalewski, 2012)	Kollmann (2007) and Ordorica (2017)
DHCPv6 options type	Similar to DHCPv4 options type IPv4 OS fingerprinting (Kollmann, 2007; Zalewski, 2012)	Kollmann (2007) and Ordorica (2017)

Figure 4 The confusion matrix and evaluation measures (see online version for colours)

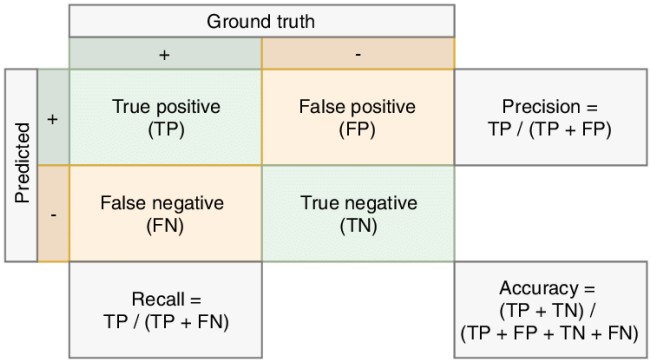
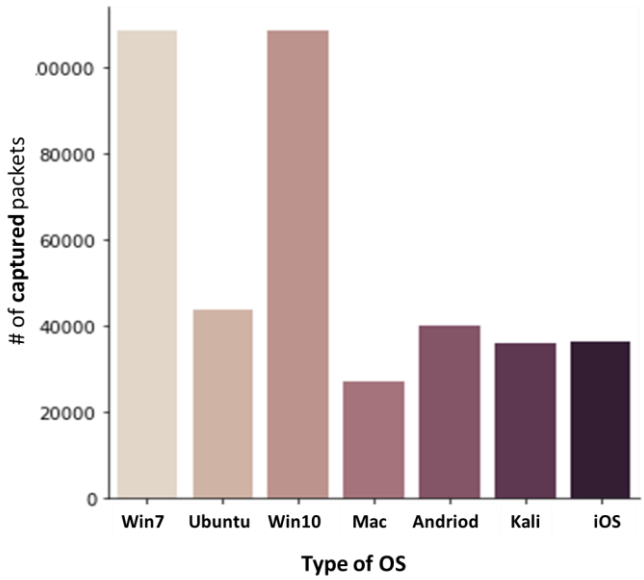


Figure 5 A histogram that shows the number of packets of each OS in original dataset (see online version for colours)

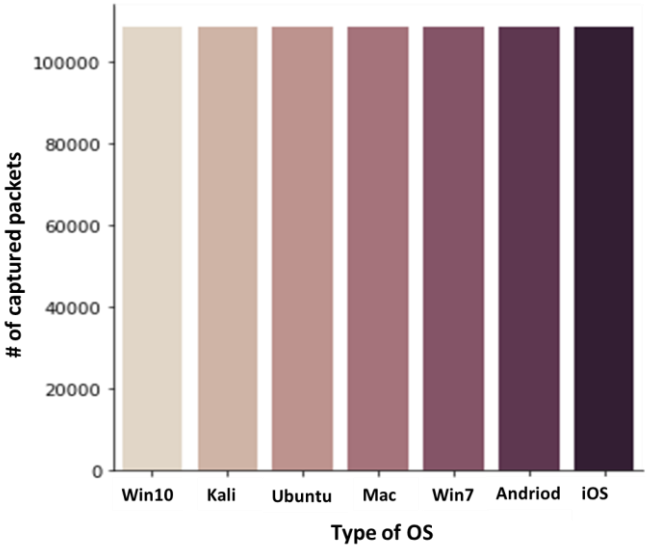


After the resampling process, the dataset has a balanced number of packets between all the OSes. Figure 6 shows the histogram plot of the balanced OSes. To this point, the dataset contains 758,772 packets from different OSes (108,396 for each OS) which is considered a sufficient number of packets. Balancing the dataset process satisfied the fourth condition of reliable dataset requirements.

The non-features columns such as the IPv6 source address is dropped from the dataset to avoid any irrelevant feature from affecting the process of building the OS classification, i.e., keeping such kind of features might give a biased result where the model will be highly dependent on them as they uniquely identify each packet. For example, the IPv6 source address identifies its packets. However, the IPv6 is not a feature where it can be changed in another scenario or network and, in this case, the model will not work probably (Elejla et al., 2018).

Some of the features are available for all IPv6 packets such as flow label and traffic class. Some others are only available in a special kind of IPv6 packets such as: TCP flags which only exist in TCP packets and ICMPv6 which only exists in ICMPv6 packets. These features' values will be empty for other packets types. For example, ICMPv6 packets will have empty values in TCP packets as well TCP Flags will have empty values in ICMPv6 packets.

Figure 6 A histogram that shows the number of packets of each OS in the balanced dataset (see online version for colours)



Having missing values mislead the process of building the model and add an extra overhead to classification process. Moreover, some classification and features ranking techniques do not accept missing values within the dataset during the process of building the model. In addition, these missing values negatively affect the process and produce a biased model (Ahmadi et al., 2019). To avoid replacing the missing value with real value (such as the most frequent values) or dropping the whole packet that might negatively affect the building process, the missing values were replaced by (–1) as an indicator that the values were missing (Fifield et al., 2015).

5.2 *Features ranking*

The features have been chosen based on assumptions about their contribution to the OS classification. To evaluate the feature's contribution and relation to the OS, features ranking technique is applied to the dataset to choose the most related features. These chosen features will be selected and others will be excluded from the dataset.

Feature ranking algorithms need the datasets to contain numeric values only. However, some features such as ICMPv6 type contain categorical values. These non-numeric features need to be converted and mapped to numeric values. To do that, *the label encoder class* from *sklearn library* is imported. The *fit_transform* function converts each unique value of the object features to a unique number. Also, a *random forest feature importance* algorithm is used to choose the best highly related features. Random forest feature importance algorithm is chosen due to its popularity, good accuracy, robustness and ease of use. Random forest feature importance gives each feature a ranking value that represents its importance in classifying the classes. Table 3 shows the given ranking value of each

feature as an output of the random forest feature importance.

Table 3 The ranking values of the selected features as generated by the random forest feature importance algorithm

#	*Feature*	*Rank value*
1	Flow label	*18.0733*
2	Length	*17.34066*
3	Hop limit	*12.90306*
4	Payload length	*12.45772*
5	ICMPv6 identifier	*11.84691*
6	Delta time	*10.25097*
7	ICMPv6 type	*4.513342*
8	ICMPv6 sequence	*4.085161*
9	Protocol	*3.780252*
10	DHCPv6 client identifier	*1.382102*
11	DHCPv6 lifetime	0.9290888
12	DHCPv6 options length	0.798176
13	DHCPv6 options type	0.5888674
14	DHCPv6 vendor class	0.4792899
15	TCP flags	0.1966203
16	TCP Window size	0.1492546
17	TCP initial sequence number	0.1461586
18	Traffic class	0.0629026
19	TCP options	0.0047998
20	TCP options length	0.0046869
21	TCP max segment size	0.0038159
22	TCP Window scale	0.0028445
23	HTTP user-agent	9.92E-06

Features that achieve a ranking value > 1 are selected to represent the dataset for the classification process. These features are only kept while the others are dropped from the dataset. The selected features are: *flow label*, *length*, *hop limit*, *payload length*, *ICMPv6 identifier*, *delta time*, *ICMPv6 type*, *ICMPv6 Sequence*, *Protocol* and *DHCPv6 client identifier*. By selecting these highly relevant features, the fifth requirement of a good dataset is satisfied.

5.3 Applying machine learning algorithms

The common machine learning algorithms were implemented with Python with the help of a library called *sklearn*. Machine learning algorithms in this library divide the dataset into four parts: the training dataset (the features and their labels) and the validation dataset (the features and their labels). These parts are named *x_train*, *x_valid*, *y_train*, *y_valid*. x_train and x_valid are the training and testing features without labeling, respectively. y_train and y_valid are the labels for training and testing features, respectively. The x and y are assigned to the *train_test_split* function to be split into the four datasets x_train, x_valid, y_train and y_valid. The chosen splitting factor is the most popularly used by the literates for large datasets which 80% training and 20% testing (Richardson et al., 2010). The dataset is split with assigning the shuffle option the Boolean value 'true' to avoid data overfitting problems.

y_train and y_valid contain categorical data which is the name of the OS. This data needs to be converted to numeric values where some of the machine learning algorithms do not accept the categorical label. To convert them to numeric, the fit_transform function is used again.

5.4 Evaluation measures results

Several machine learning algorithms have been applied to the dataset after preparing them. These algorithms are *decision tree (DT)*, *Gaussian Naïve Bayes (GNB)*, *support vector machine (SVM)*, *K nearest neighbours (KNN)* and *random forest (RF)*. They were chosen because they are considered as the most common algorithms. The classification accuracy, recall, precision and F-score are calculated using *accuracy_score*, *recall_score*, *precision score* and *F-score* functions, respectively, which are available in *sklearn.metrics library*. The training time is calculated by subtracting time after the training ends from the time before the training starts.

Each algorithm was applied to the full dataset with the ten selected features (see Table 3). Each algorithm has been applied with its default parameter without any parameter tuning. The evaluation metrics are calculated and their values are summarised in Table 4.

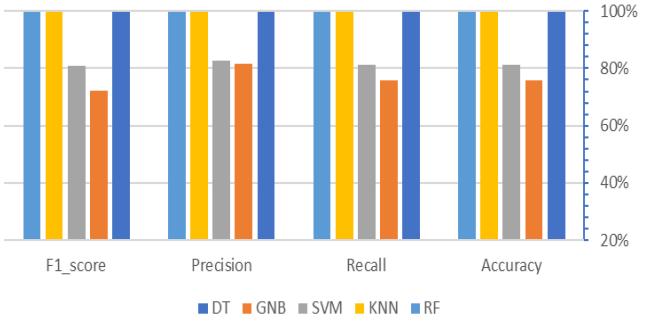
Table 4 The achieved evaluation metrics of the selected machine learning algorithms

Algorithm	*accuracy*	*recall*	*precision*	*F-score*	*Training time (sec)*
DT	0.99823	0.99823	0.99823	0.99823	1.966
GNB	0.759	0.7591	0.81617	0.7207	0.1988
SVM	0.81086	0.811	0.827	0.807	7764.32
KNN	0.99692	0.99692	0.99692	0.99692	7.8494
RF	0.99841	0.99841	0.99841	0.99841	70.94878

Figure 7 shows a graphical representation of these results. KNN, RF and DT achieve high accuracy of up to 99%. SVM achieves a quite high accuracy of up to 81%. GNB achieves the lowest accuracy up to 75%. These achieved accuracies prove the efficiency of the potential features that have been suggested for the purpose of this work. Moreover, KNN, RF and DT achieve the best Recall, Precision and F-score which are almost the same as the achieved accuracy. SVM provides almost the same good results in terms of accuracy, recall, precision and F-score which proved the robustness of the classification model. GNB achieves 0.75 recall larger than precision (0.81) due to the high number of FN records. Also, the F-score is lower than both the recall and precision which proved that FP and FN are comparatively high. These bad results achieved by GNB is due to its simplicity in classifying the records. GNB depends on the elementary Bayes' theorem. It greatly

simplifies learning by assuming that features are not dependent given the class variable.

Figure 7 The values of the evaluation metrics (accuracy, recall, precision, F-score and training time) of the machine learning algorithms (see online version for colours)

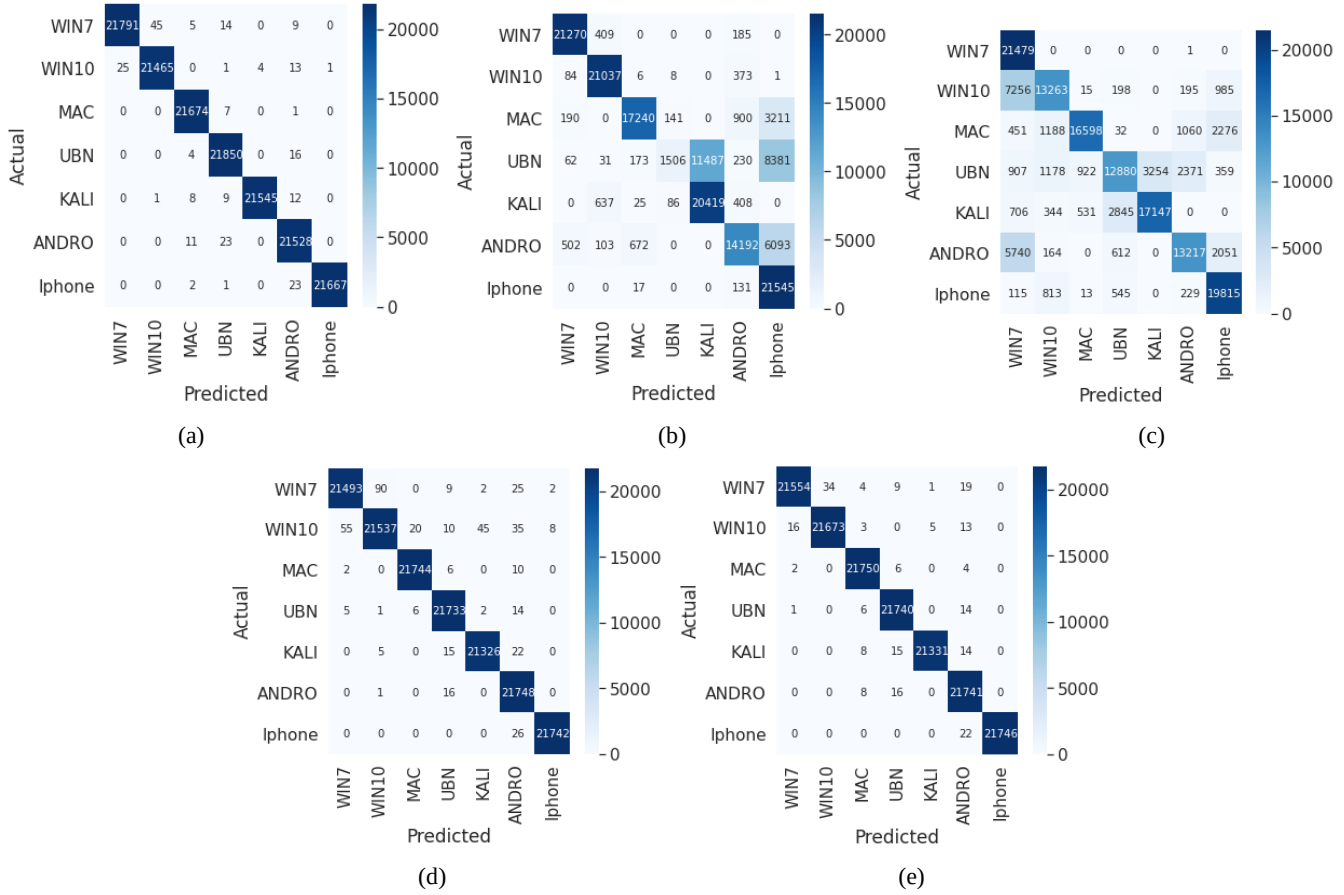


Although KNN, RF and DT provide the best results in terms of the evaluation metrics, they have varied results in terms of training time. DT needed less than a second to finish its training which is the smallest training time compared to KNN and RF. KNN took around eight seconds and RF took around 70 seconds.

The output of the confusion matrix of each algorithm is shown in Figure 8. It shows the number of correctly and incorrectly classified packets for every OS. All algorithms (except GNB and SVM) successfully classified most of the OSes packets. For example, DT correctly classified 21,791 out of 21,864, 21,465 out of 21,509, 21,674 out of 21,682, 21,850 out of 21,870, 21,545 out of 21,575, 21,528 out of 21,562 and 21,667 out of 21,693 of Win7, Win10, Mac OS, Ubuntu, Kali, Android and iOS, respectively. DT misclassified 73, 44, 8, 20, 30, 34 and 26 packets of the considered OSes, respectively.

However, GNB has the worst result among others in classifying the OSes. It successfully classified 21,270 out of 21,864, 21,037 out of 21,509, 17,240 out of 21,682, 1,506 out of 21,870, 20,419 out of 21,575, 14,192 out of 21,562 and 21,454 out of 21,693 of Win7, Win10, Mac OS, Ubuntu, Kali, Android and iOS, respectively. It failed to classify most of the packets belonging to Ubuntu correctly. This failure return to the simplicity in dealing with features during the classification.

Figure 8 The output of the confusion matrix of each of the considered machine learning algorithms (see online version for colours)



(a) (b) (c)

(d) (e)

Notes: (a) Decision tree, (b) Gaussian Naïve Bayes, (c) support vector machine, (d) K-nearest neighbour and (e) random forest.

6 Conclusions and future work

This paper proposed a new machine-learning based approach to identify IPv6 OS fingerprinting. This work considered the addition of fingerprint features to improve the detection accuracy and the scope of this work is extended beyond the classification of OSes which are installed in desktops and laptops, it also considered OSes that are installed in mobiles and tablets. In addition, a new dataset was created and will be available for the research community. The experimental results proved the efficiency of the proposed methodology to classify OSes based on IPv6 traffic. The selected set of features are discriminative enough to differentiate between the various OSes. The results obtained by using the Kaggle platform have shown that the highest accuracy, recall, precision and F-score are achieved by DT, followed by KNN and RF algorithms, respectively with more than 99%. Then comes the SVM with an average of 81%. And finally, the lowest performance refers to the GNB with an average of 75%.

Regarding the training time, it is shown that the GNB has the smallest training time interval compared to the rest of the algorithms. Considering all these evaluation matrices, we conclude that DT is the best and fastest algorithm that can achieve the best reliable model.

Compared to other works, the achieved accuracy obtained by the DT, KNN and RF outperformed the same measures cited in Ordorica (2017) that has an accuracy of up 93.2%, which means that the overall enhancement is 6.0% increase in accuracy.

References

Ahmadi, S.S., Rashad, S. and Elgazzar, H. (2019) 'Efficient feature selection for intrusion detection systems', *IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, October, pp.1029–1034.

Aksoy, A. and Gunes, M.H. (2016) 'Operating system classification performance of TCP/IP protocol headers', *IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, November, pp.112–120.

Aksoy, A., Louis, S. and Gunes, M.H. (2017) 'Operating system fingerprinting via automated network traffic analysis', *IEEE Congress on Evolutionary Computation (CEC)*, June, pp.2502–2509.

Al-Shehari, T. and Shahzad, F. (2014) 'Improving operating system fingerprinting using machine learning techniques', *International Journal of Computer Theory and Engineering*, Vol. 6, No. 1, p.57.

Anderson, B. and McGrew, D. (2017) 'OS fingerprinting: new techniques and a study of information gain and obfuscation', in *IEEE Conference on Communications and Network Security (CNS)*, October, pp.1–9.

Arkin O. (2000) *ICMP Usage in Scanning* [online] http://ofirarkin.files.wordpress.com/2008/11/icmp_scanning_v30.pdf (accessed 18 April 2021).

Arkin, O., Yarochkin, F. and Kydyraliev, M. (2003) *The Present and Future of Xprobe2: The Next Generation of Active Operating System Fingerprinting*, Sys-Security Group. [online] https://ofirarkin.files.wordpress.com/2008/11/present_and_future_xprobe2-v10.pdf (accessed 18 April 2021).

Auffret, P. (2010) 'SinFP, unification of active and passive operating system fingerprinting. *Journal in Computer Virology*, Vol. 6, No. 3, pp.197–205.

Beck, F., Festor, O. and Chrisment, I. (2007) *IPv6 Neighbor Discovery Protocol Based OS Fingerprinting*, Technical Report, RT-0345, INRIA, pp.27 [online] https://hal.inria.fr/inria-00169990/en/ (accessed 18 April 2021).

Boukhtouta, A., Lakhdari, N.E., Mokhov, S.A. and Debbabi, M. (2013) 'Towards Fingerprinting Malicious Traffic', *In ANT/SEIT*, June, Vol. 19, pp.548–555.

Brinley Jr, F.J., Kandel, E.R. and Marshall, W.H. (1960) 'Potassium outflux from rabbit cortex during spreading depression', *Journal of Neurophysiology*, Vol. 23, No. 3, pp.246–256.

Chen, Y.C., Liao, Y., Baldi, M., Lee, S.J. and Qiu, L. (2014) 'OS fingerprinting and tethering detection in mobile networks', *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*, Association for Computing Machinery, New York, NY, USA, pp.173–180.

Eckstein, C. and Atlasis, A. (2011) *OS Fingerprinting with IPv6. Infosec Reading Room*, SANS Institute., Information Security Reading Room

Elejla, O.E., Anbar, M., Belaton, B. and Alijla, B.O. (2018) 'Flow-based IDS for ICMPv6-based DDoS attacks detection', *Arabian Journal for Science and Engineering*, Vol. 43, No. 12, pp.7757–7775.

Elejla, O.E., Belaton, B., Anbar, M. and Alijla, B. O. (2017) 'IPv6 OS fingerprinting methods', *In International Visual Informatics Conference*, Springer, Cham, November, pp.661–668.

Fifield, D., Geana, A., MartinGarcia, L., Morbitzer, M. and Tygar, J.D. (2015) 'Remote operating system classification over IPv6', in *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, October, pp.57–67).

Gagnon, F. and Esfandiari, B. (2011) 'A hybrid approach to operating system discovery based on diagnosis', *International Journal of Network Management*, Vol. 21, No. 2, pp.106–119.

Google IPv6 Statistics (2020) [online] https://www.google.com/intl/en/ipv6/statistics.html/ (accessed 8 October 2020).

Jiang, W-h., Li, W-h. and Du, J. (2003) 'The application of ICMP protocol in network scanning', in *Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies*, IEEE, August, pp.904–906.

Kollmann, E. (2007) *Chatter on the Wire: A Look at DHCP Traffic* [online] http://myweb.cableone.net/xnih/download/chatter-dhcp.pdf (accessed 8 October 2020).

Lippmann, R., Fried, D., Piwowarski, K. and Streilein, W. (2003) 'Passive operating system identification from TCP/IP packet headers', in *Workshop on Data Mining for Computer Security*, November, Vol. 40.

Lyon, G.F. (2009) *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure. [online] https://nmap.org/book/ (accessed 18 April 2021).

Matoušek, P., Ryšavý, O., Grégr, M. and Vymlátil, M. (2014) 'Towards identification of operating systems from the internet traffic: IPFIX monitoring with fingerprinting and clustering', in *5th International Conference on Data Communication Networking (DCNET)*, IEEE., August, pp.1–7.

Nath, A. (2015) *Packet Analysis with Wireshark*, Packt Publishing Ltd., Birmingham, UK.

Nerakis, E. (2006) *IPv6 Host Fingerprint*, Naval Postgraduate School Monterey, CA.

Ordorica, A. (2017) *Operating System Identification by IPv6 Communication Using Machine Learning Ensembles*, Doctoral dissertation, University of Arkansas).

Prigent, G., Vichot, F. and Harrouet, F. (2010) 'IpMorph: fingerprinting spoofing unification', *Journal in Computer Virology*, Vol. 6, No. 4, pp.329–342.

Richardson, D.W., Gribble, S.D. and Kohno, T. (2010) 'The limits of automatic OS fingerprint generation', in *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, October, pp.24–34.

Schwartzenberg, J. (2010) *Using Machine Learning Techniques for Advanced Passive Operating System Fingerprinting*, Master's thesis, University of Twente.

Siby, S. (2014) *Default TTL (Time To Live) Values of Different OS*. [online] https://subinsb.com/default-device-ttl-values/ (accessed 18 April 2021).

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A. and Stiller, B. (2010) 'An overview of IP flow-based intrusion detection', *IEEE Communications Surveys & Tutorials*, Vol. 12, No. 3, pp.343–356.

Stopforth, R. (2007) Techniques and Countermeasures of TCP/IP OS Fingerprinting on Linux Systems, Doctoral dissertation.

Taleck, G. (2004) '*Synscan: Towards Complete TCP/IP Fingerprinting*', CanSecWest, Vancouver BC, Canada, pp.1–12.

Zalewski, M. (2012) *P0f V3: Passive Fingerprinter* [online] https://lcamtuf.coredump.cx/p0f3/ (accessed 8 October 2020).