Remote Host Operation System Type Detection Based on Machine Learning Approach

Leonid Kupershtein, Tatiana Martyniuk, Olesia Voitovych and Artur Borusevych

Vinnytsia National Technical University, Khmelnytske shoes str., 95, Vinnytsia, 21021, Ukraine

Abstract

There are the research results of using machine learning to solve the problem of the remote host operating system detection in the article. The analysis of existing methods and means of detection of the remote host operating system are carried out, the main advantages and disadvantages of their using are defined. Modeling of machine learning methods is carried out. The software architecture is designed and experimental application is developed. It uses a trained machine learning model that allows detecting the type and version of operating system with high accuracy.

Keywords [1]

Operating system detection, machine learning, computer networks, network protocol, scanning.

1. Introduction

Today, the number of devices in computer networks is growing every day. The list of devices includes routers, printers, IP phones, smart things, personal computers, laptops, smartphones, etc.

However, not all the network devices have the latest version of the operating system (OS) and updates related to its security. The reasons for this may be: lack of necessary funding or lack of required hardware for the new version of the operating system to work properly; unwillingness of device users to master the new interface or capabilities; lack of support for the software used in the new version of the operating system. The need for constant OS updating is an ever-increasing number of identified vulnerabilities [1, 2]. Exploiting these vulnerabilities could lead to breaches of the confidentiality, integrity, and availability of data and other software, such as web services [3, 4]. OS vulnerabilities cause the possibility of unauthorized access to database-oriented applications, which in turn requires additional protection [5]. Network administrators must be ready for possible attacks. It requires constant network monitoring to detect unauthorized devices or devices, which are running an old and/or vulnerable version of the operating system.

Penetration testing specialists and ethical hackers need to gather as much information about the object as possible to conduct authorized attacks in the initial stages. It is necessary to form the most effective vector of attack to identify potential vulnerabilities in the protection system of the researched infrastructure [6]. Knowledge about the family, type, and version of the operating system installed on network hosts can help them, because after all, each OS is associated with certain vulnerabilities in its software [1].

Currently, there are a significant number of software tools for the operating system detection, which allow some probability to determine its family, not to mention the ability to determine the type and version of the OS.

Therefore, it is very important to research and develop methods and tools that will determine detailed information about the remote host operating system with high reliability, which will increase the efficiency of identifying vulnerabilities and, consequently, increase the level of cyber security in general.

II International Scientific Symposium «Intelligent Solutions» IntSol-2021, September 28–30, 2021, Kyiv-Uzhhorod, Ukraine
EMAIL: kupershtein.lm@gmail.com (L.Kupershtein); martyniuk.t.b@gmail.com (T.Martyniuk); voytovych.op@gmail.com (O.Voitovych) ; borusevych.av@gmail.com (A.Borysevich) ORCID: 0000-0003-4712-3916 (L.Kupershtein); 0000-0003-3811-6183 (T.Martyniuk); 0000-0001-8964-7000 (O.Voitovych)

2. Methods and tools for the operating system detection

There are two main methods for detecting a remote host operating system: active and passive.

The active methods are based on sending a specially built service packets to the target machine [7]. Then after receiving answer analysis, a conclusion about the target node operating system is formed.

The advantages of this method are:

- speed – since the packets are sent to the target node, you can get a response faster, without having to wait for the necessary packets in the network;
- simplicity – usually you only need to compare the received answers with the database of signatures, without analyzing the parameters or their combination;
- flexibility – due to the packages are formed manually, it is possible to adjust the packages contents, adding new ones as needed.

However, there are also disadvantages:

- visibility – using this method, packets are sent over the network, so one can detect them and apply appropriate actions;
- signature database usage – record absence in the database, causes wrong detection or no answer at all;
- necessity of the node response receiving – if the response from the target node is not received, it is impossible to detect the operating system.

The passive methods of the operating system detection are based on network traffic listening, transmitted packets collecting, and then, their contents analyzing to form a conclusion about the remote host OS [8].

The advantages of this method are:

- invisibility – because of continuous listening, there is no activity on the listening network device, or this activity is so low that it can be perceived as normal traffic during the business day;
- no need to receive the target node response – traffic analysis from the target node to other devices is allowed.

The disadvantages of this method are:

- speed – it is necessary to wait for the appearance of certain packets in order to form a conclusion, that takes a long time in the case of the network activity absence;
- implementing complexity – since it is not possible to send self-generated packets, you need to use the information from intercepted packets.

In conclusion, the appropriate approach should be chosen depending on whether you want to perform the scan imperceptibly or you want to get the result quickly.

Existing tools of the remote node operating system detection also is considered. Currently, a small number of software products perform the task of scanning the remote node operating system. Often this task is not their main function, but only one of the menu options.

The most common tools that use the active detection method are Nmap, NetScanTools Pro and Xprobe.

Nmap is a free, open source software designed for network scanning and security auditing. The program also allows you to detect the available nodes in the network, active network services, types of firewalls, etc. [9].

Nmap uses an active method to detect the operating system. To do this, the tool creates a "fingerprint", sending TCP, UDP and ICMP packets to known potentially open and closed ports. Nmap analyzes the responses to these packets. As a result, a conclusion is formed, which indicates the type of node operating system and the reliability of this conclusion. If there are no complete matches with the signature, the score is performed (each parameter has a corresponding weight in points). The tool does not have the ability to detect the operating system by passive method, so the tool is designed specifically for active analysis [7, 10].

NetScanTools Pro uses responses to ICMP packets to detect the operating system. This is the main disadvantage of the tool – using only ICMP packets do not allow obtaining a reliable answer [11].

Xprobe is a software tool that relies on fuzzy signature matching, probabilistic assumptions, analysis of multiple matches simultaneously in the signature database [12].

The most common tools that use the passive detection method are p0f, Satori, NetworkMiner, PRADS and Ettercap. They all have a similar principle of operation, namely the analysis of incoming traffic based on the signature database [13-17].

It is worth noting the p0f tool, which uses many complex, purely passive mechanisms to detect the node operating system by any random connections. The tool can detect node operating systems in those networks, where Nmap packets provoke the security system operation.

Ettercap tool is also quite interesting. It is designed to implement man-in-the-middle attacks, while having the function of detecting the operating system by passive method.

The comparative description of the software for the host operating system detecting is presented in Table 1.

Table 1

OS detection software characteristics

Software	Method	Protocols	Last update	OS family
Nmap	Active	TCP, UDP, ICMP	23-04-2021	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows
p0f	Passive	TCP, HTTP	18-04-2016	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows
NetScanTools Pro	Active	ICMP	02-09-2020	-
Xprobe	Active	ICMP, TCP	27-07-2005	FreeBSD, Mac OSX, Linux, Windows
Satori	Passive	DHCP, TCP, HTTP, SMB	04-05-2021	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows
NetworkMiner	Passive	TCP, HTTP	23-09-2020	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows
PRADS	Passive	TCP, UDP, DHCP, ICMP	19-09-2020 (app), 16-02-2010 (DB signature)	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows
Ettercap	Passive	TCP	06-01-2021	FreeBSD, iOS, Mac OSX, OpenSolaris, Linux, Windows

You can also use Wireshark for passive detection [18]. In such case, you need to analyze certain fields by yourself: TTL, User-Agent, etc. For example, if the TTL value is 128 and the User-Agent parameter contains the value "Windows NT 10.0", you can conclude that the device has the Windows 10 operating system installed [9, 19]. However, in this case it is necessary to have database, where types of operating systems are in accordance with the packets content values [20].

There are also tools that can work using both active and passive operating system detection methods, for example are SinFP [21] and queso [22]. However, support for these tools is currently discontinued and download pages are unavailable.

3. Related works

In research [23] only mobile operating systems were analyzed. Operating systems, that were analyzed in that work, are: Android v2.3, Android v4.4, iOS 5, iOS 8, Symbian 3 and Win Phone 7.5. Dataset size is 489 GB of data, that was gathered during several months. Traffic, that was used to identify OS, was captured while watching videos on YouTube, downloading files, making video calls on Skype and combined traffic. Combined traffic included traffic, while all actions described were

made on OS's, that support multitasking. Research results show that detection accuracy is around 70% when analyzing traffic for 30 seconds, around 90% accuracy while analyzing traffic for 5 minutes, and 100% accuracy when using combined traffic for 30 seconds period.

In research [24] only TCP SYN packets were captured for analysis. There is no description on how packets were gathered. Developed system searches signature in signature database. Experiments had next results: an accuracy of 86.3% when finding exact match and additionally 9.2% were detected correctly, when using minimal distance match. Type I error is 4.5%. Developed system allows to classify OS as one of three classes: Windows 7 or 8, Windows 7 or Vista, Linux. It does not allow to detect exact version of OS, and classes are very general, because they unite 2 or more OS in 1 class.

SVM was used as a machine learning method in [25] to detect OS. Nmap signature database was used as a dataset for machine learning. Training set consisted of 1503 samples, testing set consisted of 1023 samples. Most of samples were classified as: Other system, Windows, Linux. Developed system allowed to classify detected OS as one from the list: Windows, Linux, FreeBSD, OpenBSD, MAC OS, Sun Solaris, Cisco, Other system. On average accuracy is 86.63%. Amount of errors depends on OS that was detected: for Windows amount of errors is 3,91%, for Linux – 5,19%; FreeBSD – 17,71%; OpenBSD – 15.85%; Mac – 25.8%; Solaris – 4.53%; Cisco – 24.22%; Other system – 9.74%.

In research [26] only headers of ICMP packets were analyzed. No system was developed, the OS detection was conducted manually by analyzing TTL value and identification field increment. Only 5 OS's were used during experiments: Windows 7, Windows 8.1, Windows 10, Linux 18.x, and Debian 7.x. In [27] 2 similar to our research algorithms were used: Decision Table and J48. Decision Table used data from ICMP protocol (checksum, checksum_status, ext.checksum, ident, length), J48 used data from IP, UDP, DNS protocols (IP: checksum_status, dsfield, dsfield.dscp, dsfield.ecn, flags, flags.df, flags.mf, flags.rb, frag_offset, hdr_len, len, proto, ttl, version). For Decision Table and J48 dataset size is around 79,000 packets. Operating systems, that were classified are: - Linux (Raspberry, Xubuntu), Mac OS (10.7, 10.11), Windows (7, 8, 10). Decision Tree has an accuracy of 0.994 and J48 has an accuracy of 0.94.

In [28] Decision Tree/C4.5 algorithm was used. Algorithm used data from TCP protocol (window size, ttl, don't fragment bit, packet size, options order, window size of fin packet, ttl of fin packet, don't fragment of fin packet, packet size of fin packet). Dataset size is around 30000 packets. Oss that were classified are: Windows Vista SP0-2, Windows 7 SP1, Windows 2000 SP2,4, Windows XP SP1+, Linux. Algorithm has an accuracy of 0.9086.

4. Problem statement

Methods and instruments analysis of the operating system type detection show that all of approaches are based on the signatures database. However, this imposes limitations if the signature of an OS is missing from the database, which can lead to low credibility. Herewith it is possible to form a huge base for all possible types of OS but it can lead to considerable time expenses on finding the corresponding signature. An alternative solution of the remote host OS type detection is use of the machine learning methods [23-29]. Many researchers present some results of these methods usage. They are also based on the signature database usage but with a purpose of model learning. Consequently, the signatures obtaining method, their preliminary analysis and processing may significantly affect the credibility of the OS type detection.

If focusing on the real software development the signature database should be formed manually. This is necessary for understanding the signature forming principles so in the future it would be possible to use the tool for actual tasks rather than leaving everything on the stage of developed model. In addition, this approach will allow system scaling, namely to gradually increase the detected OS number. For making the OS type detection software more convenient, both active and passive mode should be implemented.

In addition, it is important not only detect the type of OS, but also its version, what can have significant influence on the user's decision-making. Since the task of OS type detection is a classification task, the criteria of qualitative model obtaining is maximizing its accuracy and precision. In addition, an important metric of the tool grade evaluation is the response time. Most of the time is spent on the trial packet sending and receiving responses so it is a network delay.

Herewith, the attempts amount can also have great influence on the response receiving time. The complexity of the machine learning model can also significantly affect the system efficiency so upon reaching sufficient accuracy it should be as simple as possible. The volume of the studied test packages parameters and their pre-processing affect the time delays as well. Therefore, it will be advisable to select relevant protocol headers.

Thereby, it is possible to determine the main goals and criteria to develop the OS detection software based on the machine learning methods:

- precision maximization;
- response time minimization;
- self-dependent signature forming;
- scalability;
- cross-platforming;
- universality of use.

5. Data gathering and preprocessing

Solving the object classification problem, namely the problem of detection the OS type, involves the usage of algorithms for learning with the teacher (supervised learning) [30]. Although it is possible to set the problem as clustering of objects using unsupervised learning. In any case, a dataset is needed to build a machine learning model. In addition, when using supervised learning, this data set must be labeled, i.e., each class of the operating system is associated with a specific label.

A ready-made data set can be used to solve the set tasks [31], but it contains only families of operating systems. However, the user may be interested in the type and version of the operating system. Therefore, authors decide to form a data set independently.

The main idea of detecting the family / type of remote node OS is based on the analysis of network protocol headers of the OSI model application, transport and network layers. Despite the standards of network protocols (RFC, IEEE) in different operating systems of even one family, some header fields can differ significantly, such as TTL, DF, ToS IP [32].

The essence of the experiment of a training data set formation is to generate certain traffic, capture and analyze it. The analysis consists in parsing packets, redacting to one data format and relevant parameters (features) selecting. The creation of a dataset was performed under laboratory conditions. The following OS versions were studied: Linux (version 5.4.0), Mac OS (version 10.12.4 and 11.4), Windows 10 (Corporate 20h2, Home 20h2), Windows 7 (Professional), Windows XP (Professional SP3). At the same time, Windows XP and Linux 5.4 were studied using virtualization technology using VirtualBox software [33], and the others were installed on real PCs.

The experiment steps of dataset creation are shown in Figure 1.

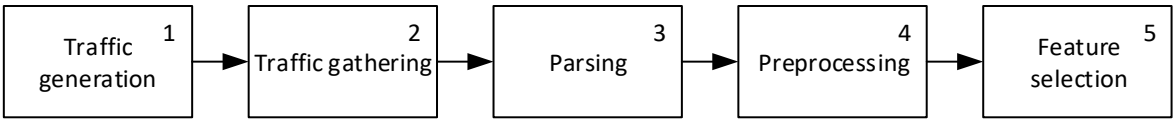

Figure 1: Experiment flow

Step 1. Traffic generation. Different types of traffic were used to form the dataset. To obtain it, the following steps were performed for each operating system:

- Sending 20 ICMP packets with the Type 8 Echo Request value from the studied OS;
- Viewing video content on the YouTube web resource for at least 30 seconds from the studied OS;
- Viewing different web pages from the studied OS;
- Downloading images from the web to the studied OS.

Step 2. Traffic capture. The most popular Wireshark traffic analyzer was used to capture packets from the network, which allows recording an interception session to a file for further processing without having to be connected to the network [19]. This software is distributed under the GNU GPL license. There are versions of Wireshark for various operating systems: Linux, Windows, MacOS, FreeBSD, Solaris. It is also possible to use no less popular console utilities "tshark and tcpdump [34]. Traffic files are saved in pcap format. As a result, so many packages were collected: MacOS 11.4 - 67949, Windows XP Professional SP3 – 21294, Windows 10 Home 20h2 – 19291, Windows 7

Professional – 17741, Mac OS X 10.12.4 - 16204, Linux 5.4.0 – 14307, Windows 10 Corporate 20h2 – 14072.

Step 3. Traffic parsing. Packets were selected from all captured traffic, in which the IP address of the source corresponds to the IP address of the PC with the studied OS. After that, it is necessary to disassemble structure of a packet on headers fields of the main protocols: IP, ICMP, TCP, DNS and HTTP. As a result of parsing, the following field's values were obtained:

1. IP – version, hdr_len, dsfield, dsfield_dscp, dsfield_ecn, len, id, flags, flags_rb, flags_df, flags_mf, frag_offset, ttl, proto, checksum, checksum_status.
2. ICMP – type, code, checksum, checksum_status, ident, seq, seq_le, data, data_data, data_len.
3. TCP – hdr_len, flags, flags_res, flags_ns, flags_cwr, flags_ecn, flags_urg, flags_ack, flags_push, flags_reset, flags_syn, flags_fin, flags_str, window_size_value, window_size, window_size_scalefactor, checksum, checksum_status.
4. DNS – id, flags, flags_response, flags_opcode, flags_truncated, flags_recdesired, flags_z, flags_checkdisable, count_queries, count_answers, count_auth_rr, count_add_rr, qry_name_len, count_labels, qry_type, qry_class.
5. HTTP – user_agent.

Parsing is performed using the Python 3.8 and the PyShark library, which allows working effectively with pcap-files. The parsing results of each packet are saved in a csv-file for further analysis. In addition to the field values, each entry in the csv file also has an OS version. As a result, a dataset containing 42,318 records was formed. It can be used as a database of signatures, which can give a certain result the search for.

Step 4. Preprocessing. Almost all classifiers do not work directly with text data. They received features as well as numbers or booleans (which translate into numbers 0/1) that some feature is there or not. Therefore, it is necessary to convert all categorical (text) features (IP: dsfield, flags; TCP: flags, flags_str; DNS: flags, qry_class) into numbers. The following methods were used for this purpose [35]:

- label encoding – to encode the values of the operating system classes by assigning a certain number to each OS value;
- one hot encoding – to encode attribute values (protocol parameters) by creating columns where each column is responsible for single attribute value and the attribute value is set as 1 in the corresponding column, and 0 in the other columns of the attribute.

The Processing module from the Scikit-learn (Python) library was used as a tool. An example of the converted data is shown in Table 2, 3.

Table 2
Results of label encoding

Type/version OS	Label	Type/version OS	Label
Linux 5.9.0	0	Windows 10 Home 20h2	4
Mac OS 10.12.4	1	Windows 7 Professional	5
MacOS 11.4	2	Windows XP Professional SP3	6
Windows 10 Corporate 20h2	3		

Table 3
Example of one hot encoding

	Before	After	
Feature name	ip_flags	ip_flags_0x00000000	ip_flags_0x00004000
Values	0x00000000	1	0
	0x00004000	0	1

The id, checksum, and data fields of the TCP / IP protocols were not used during the transformation and were removed from the feature list due to lack of clear differences between OS families. Also for the study of some classifiers data normalization was performed, i.e. reduction of all values of each parameter with a mean of 0 to the standard deviation of 1 by expression [36]:

$$z_{ji} = \frac{(x_{ji} - \mu_j)}{\sigma_j}, \quad (1)$$

where z_{ij} – i-th normalize value of j-th feature, x_{ij} – i-th original value of j-th feature, μ_j – mean of j-th feature, σ_j – standard deviation of j-th feature. For this transformation, the "StandartScaler" method from Scikit-learn (Python) is used.

Step 5. Feature selection/importance. Since the size of the dataset has slightly increased from 54 to 156 after the some features transformation, it will be advisable to reduce the dimension and select the most relevant features. This is necessary to increase the learning speed of the model, computation time when using the trained model, avoid overfitting and increase the generalizing ability of the model. Among the significant number of methods for selecting features by importance, the Recursive Feature Elimination method was used. The essence of this method is to build a model (which includes all factors), which excludes the least significant factor (feature) from the point of view of the model. After that, a new model is built, which contains all the factors except those excluded in the previous stage, and so on [37]. The RFE module from the Scikit-learn library was used to solve the problem. The decision tree model with default hyperparameters was used as an estimator. As a result of selection, 14 features were identified, the parameters and the importance degree of which are shown in Table 4. The importance degree is obtained from the "feature_importance" attribute of the trained model. Moreover, although some features have little impact, in our opinion they are important and give some expressiveness to different operating systems.

Table 4
Characteristics of dataset features

Protocol	Parameter name	Description	Value example	Importance (%)
IP	hdr_len	the length of the IP header	20	0.0001
IP	flags	list of set flags	0x00004000	0.0001
IP	flags_df	the value of the flag Don't Fragment	1	0.0001
IP	ttl	packet lifetime	64	24.8843
IP	proto	the protocol used below	17 (UDP)	0.0001
ICMP	ident	identification of parts of the protocol packet	1	8.71
ICMP	seq	packet number in the sequence	558	0.0001
ICMP	seq_le	the current length of the sequence	142848	0.0088
ICMP	data_data	the contents of the message in the packet	61:62:…:76:77:61:62:63:64:65:66:67:68:69	0.0001
ICMP	data_len	the length of the message in the packet	32	36.87
TCP	hdr_len	the length of the protocol header	32	0.0001
TCP	window_size_value	window size value	513	0.008
TCP	window_size	the calculated value of the window	131328	0.0001
TCP	window_size_scale factor	the calculated window size modifier (2^n)	256	29.5181

6. Model selection and training

When solving the problem of classification by machine learning methods, the question of choosing a classifier model arises. At present, a significant number of classifiers are known and implemented,

which differ in approaches to the construction of the decision rule, as well as hyperparameters. Correct adjustment of hyperparameters is one of the key points; it allows receiving desirable results. In some models of machine learning, the number of hyperparameters can reach more than 10, and each hyperparameter can take different values. Finding the optimal combination is not an easy task. One of the options for solving this problem is to build a model for each possible combination for all given domains of hyperparameters.

As a result of preliminary data processing, a dataset was obtained, the structure of which is shown in Table 5.

Table 5
Dataset structure

Full dataset			Small dataset		
OS	Samples	Percent, %	OS	Samples	percent, %
MacOS 11.4	16949	40.05	MacOS 11.4	999	14.67
Windows XP Pro SP3	5281	12.48	Windows XP Pro SP3	999	14.67
Windows 10 Home	4717	11.14	Windows 10 Home	999	14.67
Windows 7 Pro	4385	10.36	Windows 7 Pro	999	14.67
Mac OS X 10.12.4	4024	9.51	Mac OS X 10.12.4	999	14.67
Linux 5.4.0	3549	8.39	Linux 5.4.0	816	11.98
Windows 10 Corp 20h2	3413	8.07	Windows 10 Corp	999	14.67
Total:	42318	100	Total:	6810	100

For each model, the desired metric is calculated, the best result of which determines the best model. This approach is implemented in the GridSearch package of the Scikit-learn library [38]. Although this approach is quite costly in terms of machine time, it gives the best result. In addition, although the Scikit-learn library does not support GPU computing, one can use a set of cuML libraries from the RAPIDS project to parallelize calculations based on CUDA technology [39, 40].

Using GridSearch technology, the best parameters for the most used classifier models were automatically selected:

1. Decision Tree (DT) (criterion='entropy', max_depth=6);
2. Multilayer Perceptron (MLP) (alpha=0.05, hidden_layer_sizes=(50, 50, 50));
3. Gaussian Naive Bayes (GNB) (var_smoothing=1e-9);
4. K-Nearest Neighbors (KNN) (leaf_size=5, n_neighbors=1);
5. Support Vector Machine (SVM) (C=1000, gamma=0.0001);
6. Logistic Regression (LR) (C=5, max_iter=500, solver='newton-cg');
7. Random Forest (RF) (criterion='entropy', max_features='sqrt', min_samples_leaf=4, n_estimators=1900).

Since the use of GridSearch is quite time consuming even with a large number of models, it was decided to use a small dataset (table 5). It is representative enough to obtain adequate results.

The following metrics were used to assess the quality of classifiers [3, 35]:

- Accuracy – share of correctly defined classes;
- F1 – weighted average estimate of type I and II errors;
- Precision – the ratio of the correctly defined classes number to the sum of the correctly defined classes and type I error (false positive) numbers;
- Recall – the ratio of the correctly defined classes number to the sum of the correctly defined classes and type II error (false negative) numbers;
- Confusion Matrix – a matrix showing the number of correct definitions and the number of erroneous definitions;
- FP – type I error (False Positive);
- FN – type II error (False Negative).

After obtaining the optimal parameters of the classifiers based on the maximizing "accuracy" criterion, they were trained on a full dataset. Naive Bayes trained the fastest in time, and Logistic Regression the longest: DT - 0.194 s, GNB - 0.0409 s, KNN - 0.742, LR - 482.51 s, RF - 88.369 s, SVM - 5.08 s, MLP - 39.43. Training of models was performed on a PC with AMD A8-4500M APU, 1.9 GHz (4 cores, 4 threads) and 8 Gb RAM. Dataset is divided into training and test in the amount of 70% and 30%, respectively. The test results of the classifiers are presented in Table 6.

Table 6
Classifiers metrics

Classifier	Accuracy	Precision	Recall	F1	FP, %	FN, %	Average cross-validation score
DT	1.0	1.0	1.0	1.0	0	0	0.9999
MLP	0.99964	0.99952	0.9995	0.9995	0.002	0,004	0.9998
GNB	0.90548	0.91690	0.8736	0.8474	1.7	0,189	0.9031
KNN	0.99763	0.99682	0.9960	0.9964	0.016	0,031	0.9949
SVM	0.99952	0.99926	0.9994	0.9993	0.009	0	0.9987
LR	0.97861	0.97287	0.9880	0.9792	0.378	0,059	0.9784
RF	1.0	1.0	1.0	1.0	0	0	0.9999
Frequency analysis [23]	0.7-1	-	-	-	-	-	-
Euclidean distance [24]	0.955	-	-	-	4.5	-	-
SVM [25]	0.8663	-	-	-	-	13.36	-
Decision Table [26]	0.994	-	-	-	-	-	-
DT/J48 [27]	0.94	-	-	-	-	-	-
DT/C4.5 [28]	0.9086	-	-	-	-	-	-

As can be seen from Table 6, the Decision Tree models and its ensemble modification Random Forest have best results without any errors. The worst classifier is Gaussian Naive Bayes, which showed the highest number of erroneous predictions, but in general, all classifiers have high metrics. These results are confirmed by 5-Fold cross validation. Compared with the known research results, our metrics are better. However, for a more adequate comparison it is necessary to carry it out at least for identical families / types / versions of OS.

According to the simulation results, confusion matrixes is obtained, which are shown in Appendix A. They contain the results of model prediction, namely the following indicators: True Positive, True Negative and FP, FN.

Analyzing the matrix of classifier errors (Figs.A.1 – A.6), we can note that most classifiers have errors within the OS family:

- Multilayer Perceptron: incorrect definition of Windows 10 Home (defined as Windows 10 Corporate) and Windows 7 Professional (defined as Windows XP Professional);
- Gaussian Naive Bayes: incorrect definition of Linux (defined as Windows 10 Home and Corporate), MacOS (defined as Linux), Windows 10 Corporate (defined as Linux), Windows 10 Home (defined as Windows 10 Corporate and Linux);
- K-Nearest Neighbors: incorrect definition of Windows 10 Corporate (defined as MacOS 11.4 and Windows 10 Home), Windows 10 Home (defined as Windows 10 Corporate);
- Support Vector Machine: incorrect definition of Windows 10 Home (defined as Windows 10 Corporate) and Windows XP (defined as Windows 10 Home);
- Logistic Regression: incorrect definition of MacOS 11.4 (defined as MacOS 10.12.4) and incorrect definition of Windows 10 Corporate (defined as Windows 10 Home and Linux).

Because the Decision Tree model is architecturally simpler than Random Forest, it is therefore more appropriate to use it in software. The tree consists of five levels and has 13 leaves. Due to the bulkiness of the tree, its image is not given in this article.

7. Application architecture and experiments

A software is developed for application use of trained classifier model. The software architecture is shown in Figure 2.

It consists of four modules: visualization module, scanning module, preprocessing module and intelligent analysis module. System has its own database also.

Visualization module is the first one to start. It initializes graphical components of program: icons, buttons, windows etc. It is also responsible for providing dialogue windows for user interaction, providing detection results for user.

Scanning module has two purposes: sniffing network traffic to capture packets from the host, which OS needs to be detected; sending probes (specific packets) to the host, which OS needs to be detected, in order to receive responses and conduct OS detection.

Preprocessing module is responsible for gathering fields from captured packets that are needed in order to create OS signature and creating OS signature. Signature is a string, where all fields needed from received packets are written, each field is separated by comma.

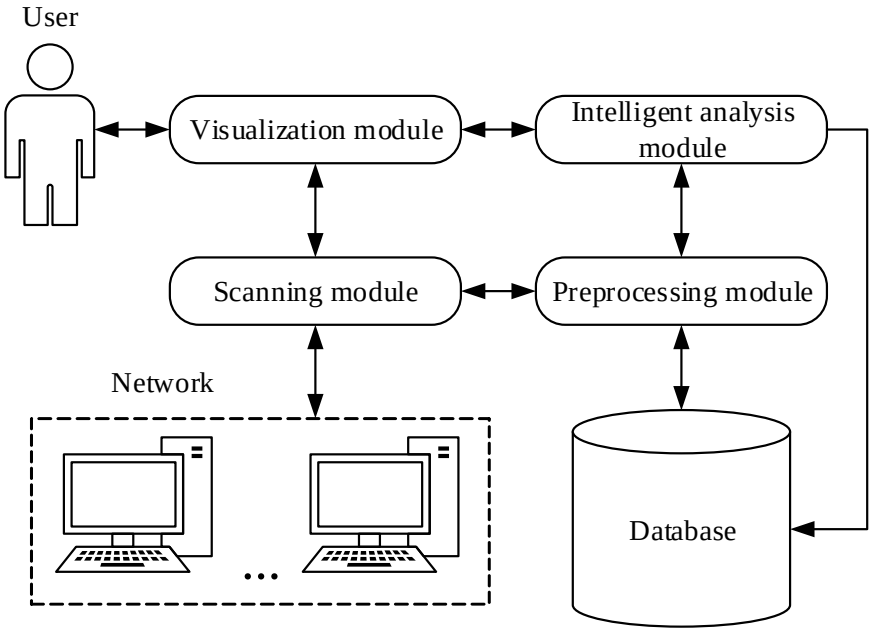

Figure 2: Application architecture

Intelligent analysis module uses trained machine learning classification algorithm to conduct OS detection on provided signature. Classification results are sent to visualization module in order to show them to user. Application database is used for storing previous OS detection results for using later. It can be used for signature storing and pcap-files also.

Based on proposed architecture a software is developed using Python 3.8 programming language with PyQt5 library. SQLite is used as a database. Software has two modes: online and offline.

In offline mode, user has an opportunity to upload pcap-file from external or internal resources. It can be useful when you already have captured traffic and need to detect type/version of OS after some time. Pcap-file should contain information about one specific host. Otherwise it can be filtered by ip-address and then resaved. Online mode is available for both active and passive scanning. Passive scanning takes more time, since, as stated before, all needed packets are required for analysis. Software is in waiting mode until all needed packets are not captured. However, it can be sped up by, for example, visiting it if detected host is a webserver and is available through http or ftp. In addition, software provides a way to detect OS by analyzing User Agent from HTTP packets.

Active mode is more appropriate since it does not require waiting, because software send probes: four ICMP requests along with up to 10 TCP SYN requests. The procedure will be repeated up to 5 times if no response is received. After preprocessing module receives packets it sends formed signature to intelligent analysis module. Classifier predicts OS family/type/version of the target host and prediction probability. Next, an experiment is conducted by checking how software works and compare results with Nmap. First a scan for Windows 10 Corporate was conducted, it's IP address in local network is 192.168.1.170. Results of the active scan mode are shown in Appendix B.

Results are stating that Nmap couldn't detect exactly installed operating system. Also in results we can see a wide specter of possible operating systems (Windows 10, 7, Windows Phone, Windows Server, FreeBSD), and probability is equal to 0.92 compared to 0.955 of developed software (Fig. B.1, 2). By executing similar actions for host, that has IP address 192.168.43.60 developed software correctly detected Windows 10 Home 20h2 with probability of 0.937 (Fig. B.3). Nmap do not provide a result and states that host has too many signature matches (Fig. B.4). Also rented host with white IP address was used in experiments. It has OS Linux 5.4 installed beforehand. Developed software detected it with probability of 0.987 (Fig. B.5). Nmap gave a result of it having Windows 7 installed (Fig. B.6).

8. Conclusion and future work

The analysis of existing methods and means of detection of the remote node operating system shows the main advantages and disadvantages of their using. Two main approaches, active and passive, of the remote node OS detection are considered. The combination of these approaches can make the process detection more flexible and accurate. The existing method of remote host OS detection is mostly based on signature model of decision making, but in such way they have a lot of errors because of undecidability in case of properly signature absence. That is why the machine learning methods are carried out. To form dataset for model learning the five-stage process is realized: traffic generating and gathering, parsing, preprocessing and feature selection.

As a result of the research, a classifier based on the Decision Tree model was trained. At the same time, other classifiers also showed high metrics with low rate errors within the OS family. The high accuracy of detection the type of OS indicates well-chosen features, as well as a sufficient size of input data set. This helped to avoid the effect of overfitting.

The trained model can detect seven OS versions within several families with absolute accuracy. The system architecture is proposed to remote OS detection. It realized in crosspaltform application based on trained model. This software tool allows scanning hosts in both offline and online modes. At the same time, both active and passive scanning are implemented online. This adds a means of versatility compared to analogues. The developed tool can be used by an ethical hacker for intrusion testing, network administrator for auditing, checking the network for new unknown devices. You can also use the tool to test the effectiveness of server protection tools against detecting their OS.

As a result of the experiment, the developed software was more effective compared to Nmap. However, Nmap allows to define many more types of OS. Therefore, further research will be associated with scaling the model, as well as expanding the functionality of application. One such future function will be to implement the ability to provide a list of ranked vulnerabilities inherent in a particular OS. It provides faster vector attack creation or decision making for network protection.

9. References

[1] CVE details: The ultimate security vulnerability datasource – Operation systems. URL: https://www.cvedetails.com/product-list/product_type-o/vendor_id-0/firstchar-W/Operating-Systems.html.

[2] Vulnerability and threat trends 2020: Reseatch report. URL: https://lp.skyboxsecurity.com/rs/440-MPQ-510/images/Skybox_Report_2020-VT_Trends.pdf

[3] Leonid Kupershtein, Tatiana Martyniuk, Olesia Voitovych, Bohdan Kulchytskyi, Andrii Kozhemiako et al. "DDoS-attack detection using artificial neural networks in Matlab," Proc. SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019. doi: 10.1117/12.2536478

[4] Voitovych, O.P., Yuvkovetskyi, O.S., Kupershtein, L.M. "SQL injection prevention system", 2016 IEEE International Scientific Conference "Radio Electronics and Info Communications", UkrMiCo 2016 - Conference Proceedings. doi: 10.1109/UkrMiCo.2016.7739642

[5] Voitovych, O., Kupershtein, L., Lukichov, V., Mikityuk, I. "Multilayer Access for Database Protection", 2018 International Scientific-Practical Conference on Problems of Infocommunications Science and Technology, PICS&T'2018 - Proceedings, pp. 474-478. doi: 10.1109/INFOCOMMST.2018.8632152

[6] Gurpreet K. Juneja1 “Ethical hacking: a technique to enhance information security”, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 2, Issue 12, pp. 7575-7580.

[7] A. Orebaugh, B. Pinkard, Nmap in the Enterprise: your guide to network scanning, Burlington, MA : Syngress Pub., 2008.

[8] G. Fyodor Lyon, Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning, Nmap Project, 2009, 468 p.

[9] Nmap: the Network Mapper - Free Security Scanner. URL: https://nmap.org.

[10] Fingerprinting Methods Avoided by Nmap. URL: https://nmap.org/book/osdetect-other-methods.html#osdetect-passive.

[11] NetScanTools Pro OS Fingerprinting Tool Description. URL: https://www.netscantools.com/nstpro_os_fingerprinting.html.

[12] xprobe2(1) - Linux man page. URL: https://linux.die.net/man/1.
[13] p0f v3 (version 3.09b). URL: https://lcamtuf.coredump.cx/p0f3.
[14] GitHub - xnih/satori: Python rewrite of passive OS fingerprinting tool. URL: https://github.com/xnih/satori.
[15] NetworkMiner. URL: https://www.netresec.com/?page=NetworkMiner.
[16] GitHub - gamelinux/prads: Passive Real-time Asset Detection System. URL: https://github.com/gamelinux/prads.
[17] Ettercap Home Page. URL: https://www.ettercap-project.org.
[18] OS Fingerprinting using Wireshark. URL: https://andytanoko.wordpress.com/2020/07/19/os-fingerprinting-using-wireshark.
[19] Wireshark Tutorial: IdentifyingHosts and Users. URL: https://unit42.paloaltonetworks.com/using-wireshark-identifying-hosts-and-users.
[20] OS Detection Techniques. URL: https://jonathansblog.co.uk/os detection-techniques.
[21] P. Auffret, SinFP, unification of active and passive operating system fingerprinting. Journal in Computer Virology, 2008, 6(3), pp. 197–205. doi:10.1007/s11416-008-0107-z
[22] C. Trowbridge, An Overview of Remote Operating System Fingerprinting, White paper, 2003. URL: https://sansorg.egnyte.com/dl/dp8wFpM37k/?
[23] Gurary, Jonathan & Zhu, Ye & Bettati, Riccardo & Guan, Yong. (2016). Operating System Fingerprinting. doi: 10.1007/978-1-4939-6601-1_7.
[24] Tyagi, Rohit & Paul, Tuhin & Bs, Manoj & B., Thanudas. (2015). Packet Inspection for Unauthorized OS Detection in Enterprises. IEEE Security & Privacy. 13. pp. 60-65. doi: 10.1109/MSP.2015.86.
[25] B. Zhang, T. Zou, Y. Wang and B. Zhang, "Remote Operation System Detection Base on Machine Learning," 2009 Fourth International Conference on Frontier of Computer Science and Technology, 2009, pp. 539-542. doi: 10.1109/FCST.2009.21.
[26] Song, Jinho & Kim, Yonggun & Won, Yoojae, Operating System Fingerprint Recognition Using ICMP, 2020. doi: 10.1007/978-981-13-9341-9_49.
[27] Aksoy, Ahmet & Louis, Sushil & Gunes, Mehmet, Operation system fingerprinting via automated network traffic analysis, 2017, pp. 2502-2509. doi: 10.1109/CEC.2017.7969609.
[28] Al-Shehari, Taher & Shahzad, Farrukh. Improving Operating System Fingerprinting using Machine Learning Techniques. International Journal of Computer Theory and Engineering, 2014.
[29] A. Aksoy, S. Louis and M. H. Gunes, "Operating system fingerprinting via automated network traffic analysis," 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 2017, pp. 2502-2509. doi: 10.1109/CEC.2017.7969609.
[30] Martyniuk, T.B., Kozhemiako, A.V., Kupershtein, L.M. "Formalization of the Object Classification Algorithm", Cybernetics and Systems Analysis, 2015, 51 (5), pp. 751-756. doi: 10.1007/s10559-015-9767-0.
[31] Dataset Using TLS Fingerprints for OS Identification in Encrypted Traffic. URL: https://zenodo.org/record/3461771.
[32] De Montigny-Lebouf A. A Multi-Packet Signature Approach to Passive Operating System Detection, Communications Research Centre, Canada, 2005. URL: https://apps.dtic.mil/sti/pdfs/ADA436420.pdf.
[33] Virtualization: IBM Cloud Education. URL: https://www.ibm.com/cloud/learn/virtualization-a-complete-guide.
[34] Tracing network traffic using tcpdump and tshark. URL: https://techzone.ergon.ch/tcpdump
[35] Albon C. Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning, O'Reilly Media, Inc., 2018, 336 p.
[36] D. Singh, B. Singh, Investigating the impact of data normalization on classification performance, Applied Soft Computing, Vol. 97, Part B, 2020. doi: 10.1016/j.asoc.2019.105524.
[37] Kuhn M., Kjell J. Feature Engineering and Selection: A Practical Approach for Predictive Models: Chapman and Hall/CRC, 2019. 310 p.
[38] Tuning the hyper-parameters of an estimator. URL: https://scikit-learn.org/stable/modules/grid_search.html.
[39] Scikit-learn Tutorial – Beginner's Guide to GPU Accelerating ML Pipeline. URL: https://developer.nvidia.com/blog/scikit-learn-tutorial-beginners-guide-to-gpu-accelerating-ml-pipeline.
[40] Mahler P. RAPIDS Release 21.06. URL: https://medium.com/rapids-ai/rapids-release-21-06-f9bd2e5a9aa4.

10. Appendix

Appendix A. Confusion matrixes

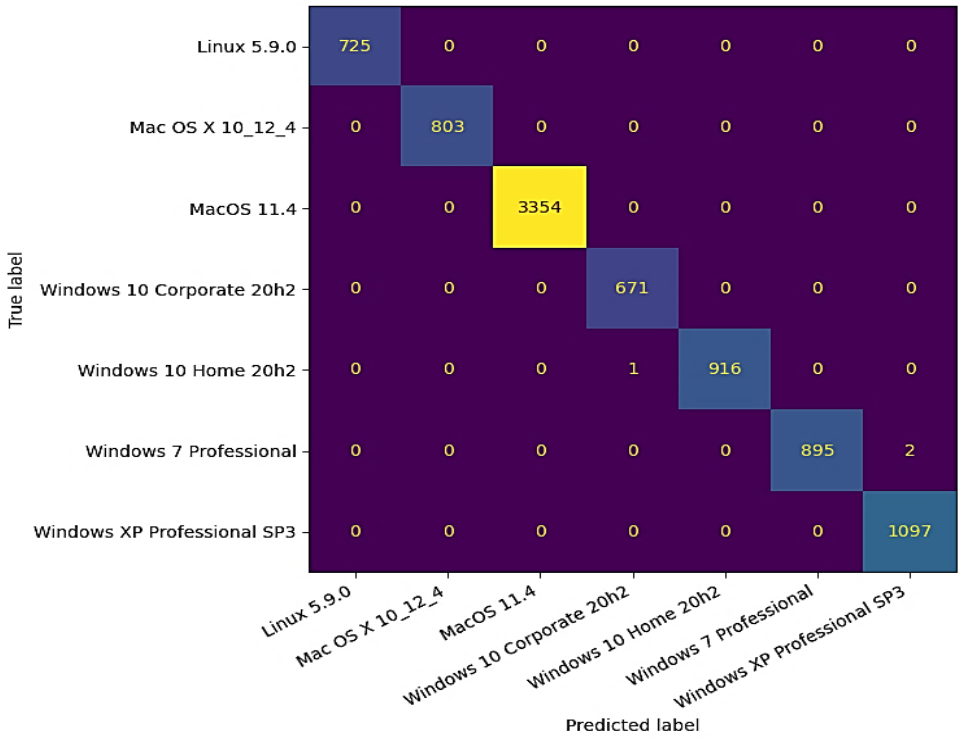

Figure A.1: Multilayer Perceptron

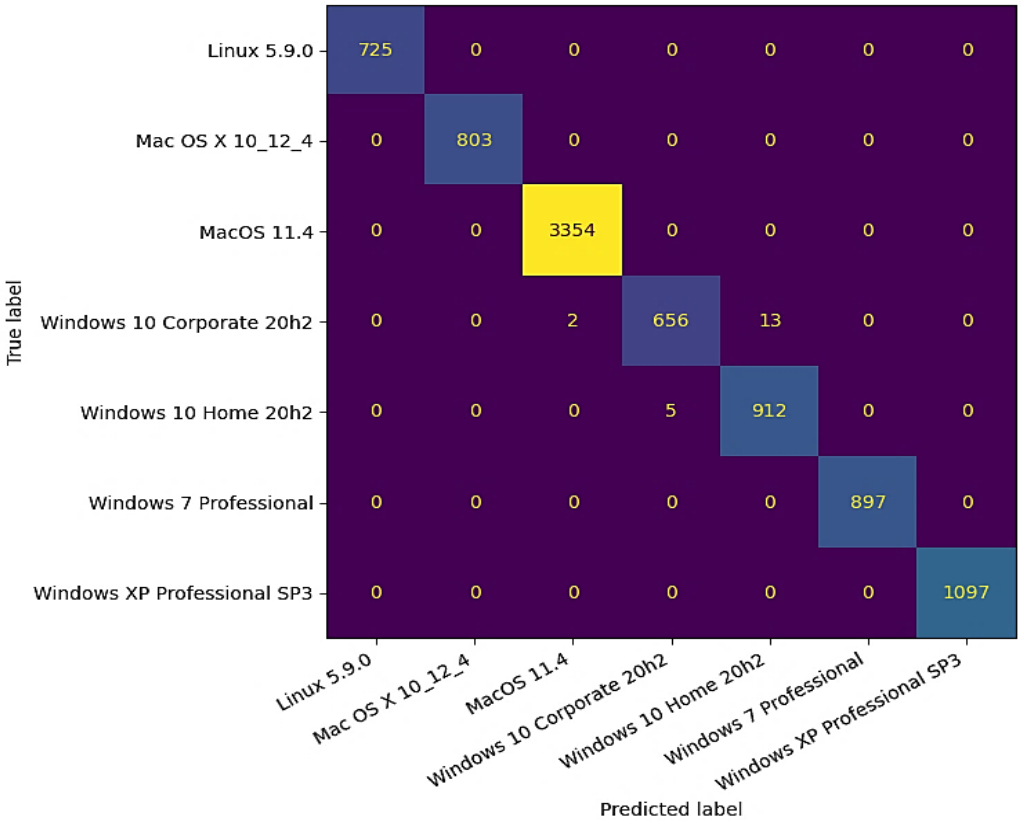

Figure A.2: K-Nearest Neighbors

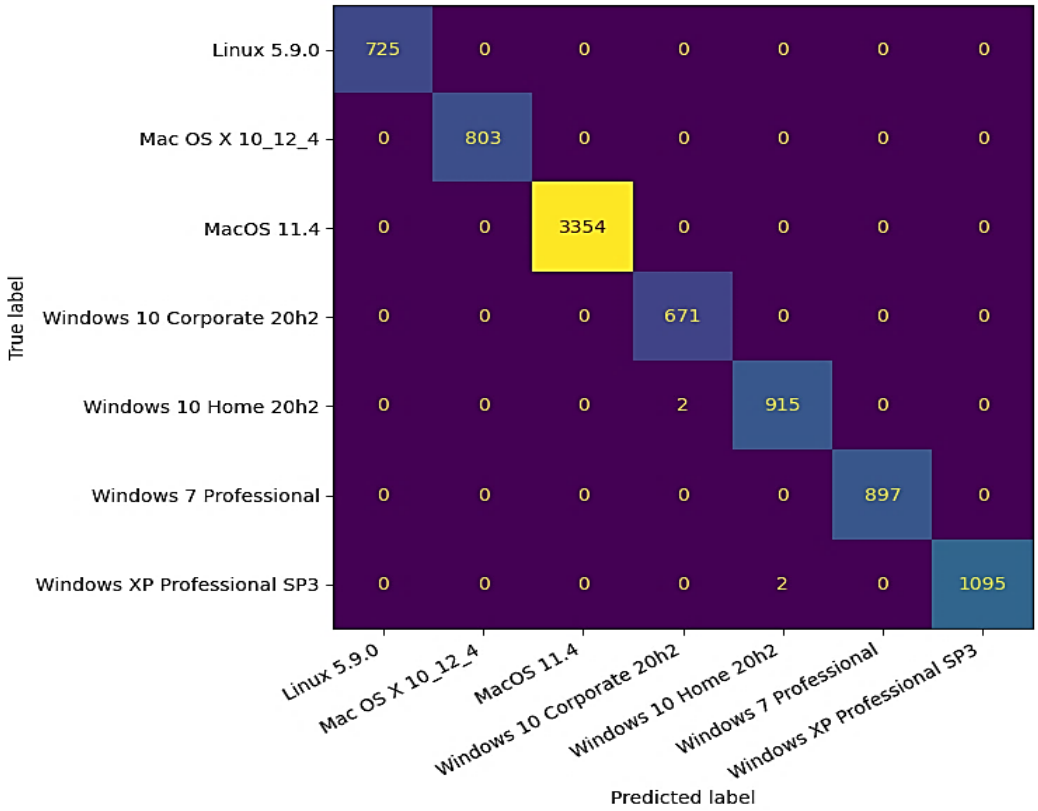

Figure A.3: Support Vector Machine

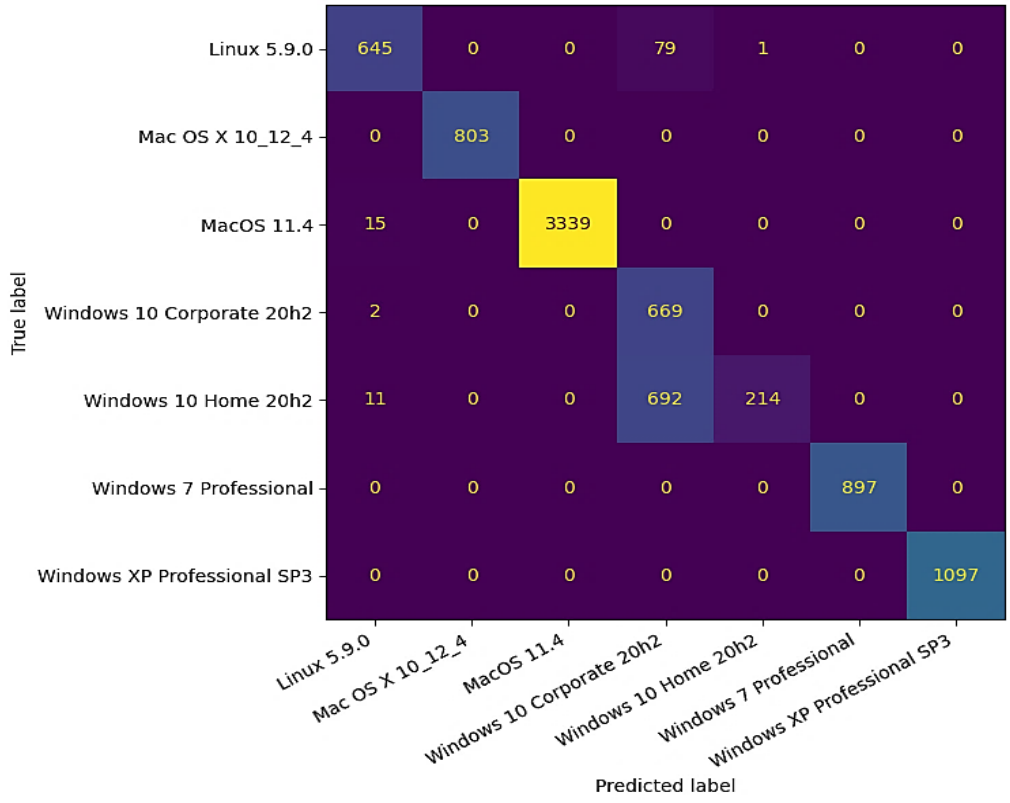

Figure A.4: Gaussian Naive Bayes

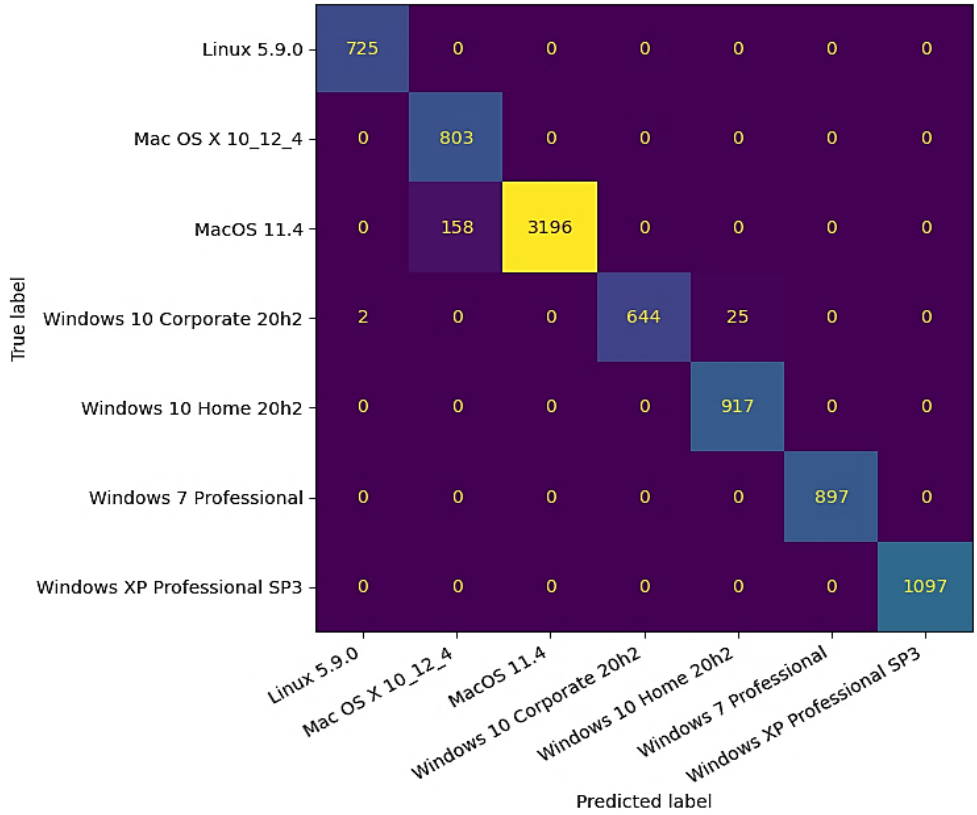

Figure A.5: Logistic Regression

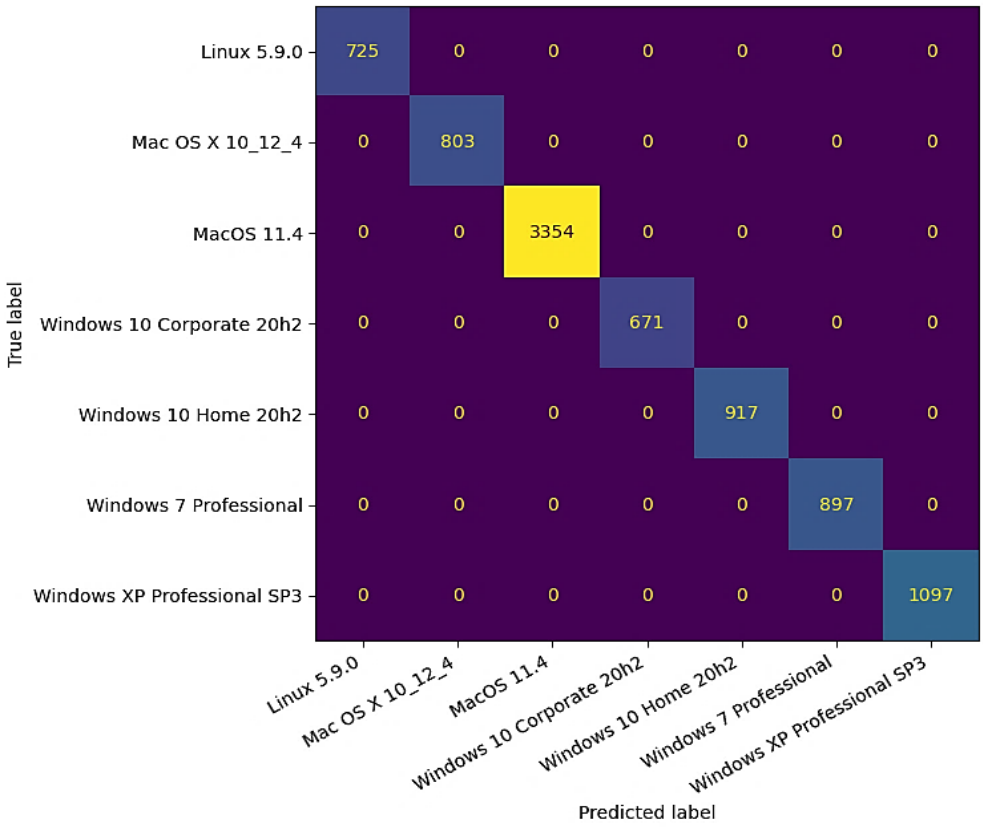

Figure A.6: Random Forest

Appendix B. Experiments results

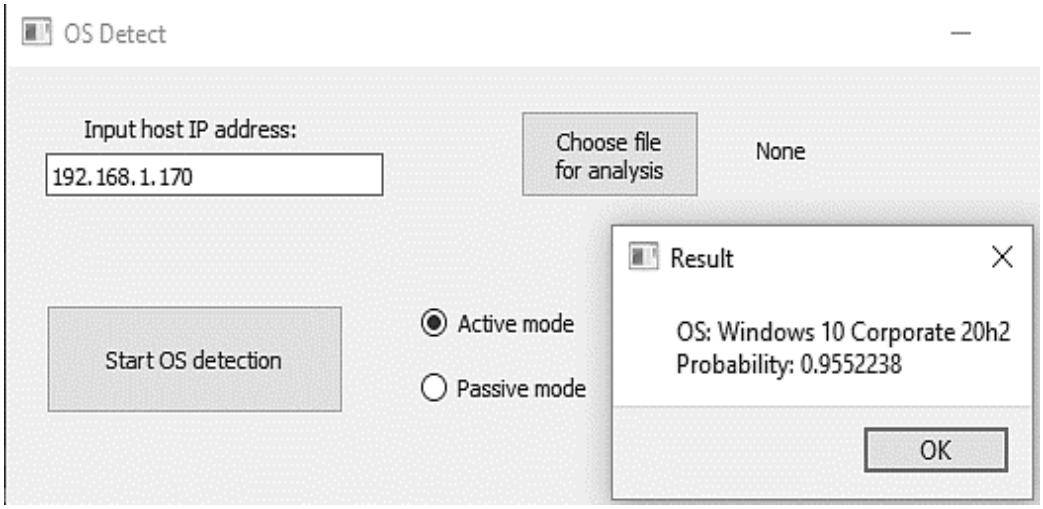

Figure B.1: Developed application

```
Nmap scan report for 192.168.1.170
Host is up (0.00020s latency).
Not shown: 96 filtered ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-wbt-server
MAC Address: A4:17:31:13:C5:27 (Hon Hai Precision Ind.)
Warning: OSScan results may be unreliable because we could not find at least 1 open
and 1 closed port
Device type: general purpose|phone
Running (JUST GUESSING): Microsoft Windows 10|2008|7|Phone (92%), FreeBSD 6.X (90%)
OS CPE: cpe:/o:microsoft:windows_10 cpe:/o:freebsd:freebsd:6.2 cpe:/
o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_7 cpe:/
o:microsoft:windows
Aggressive OS guesses: Microsoft Windows 10 (92%), FreeBSD 6.2-RELEASE (90%),
Microsoft Windows Server 2008 SP1 (87%), Microsoft Windows 7 (87%), Microsoft Windows
10 1703 (86%), Microsoft Windows 10 1511 - 1607 (86%), Microsoft Windows Phone 7.5 or
8.0 (85%)
No exact OS matches for host (test conditions non-ideal).
```

Figure B.2: Nmap

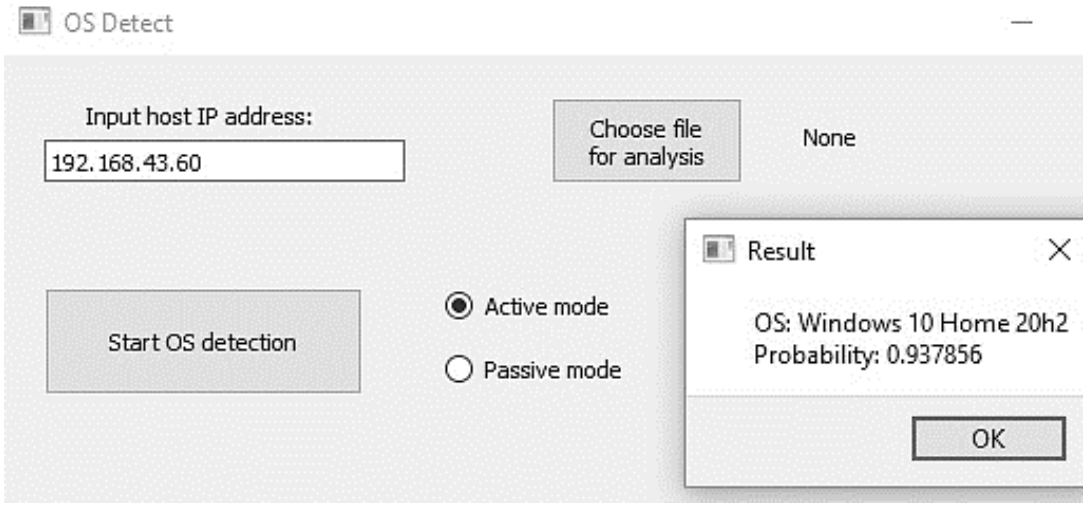

Figure B.3: Developed application

```
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-09 13:11 Oeieyiaey (eaoi)
Nmap scan report for DESKTOP-T2G8KQ0 (192.168.43.60)
Host is up (0.00s latency).
All 100 scanned ports on DESKTOP-T2G8KQ0 (192.168.43.60) are filtered
MAC Address: A4:17:31:13:C5:27 (Hon Hai Precision Ind.)
Too many fingerprints match this host to give specific OS details
```

Figure B.4: Nmap

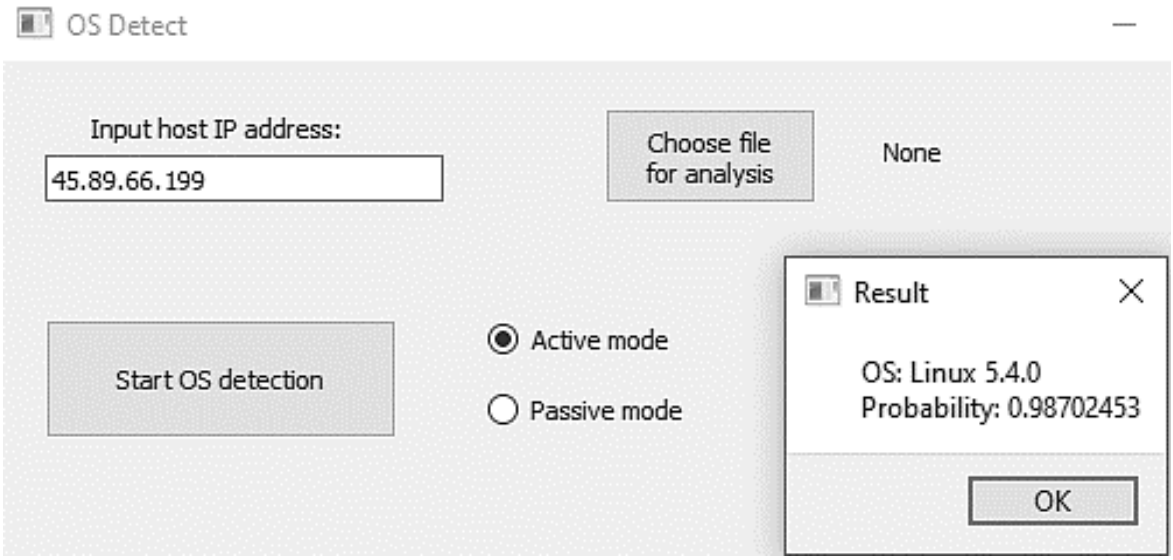

Figure B.5: Developed application

```
nmap -T4 -O -F 45.86.66.199

Starting Nmap 7.70 ( https://nmap.org ) at 2021-07-14 19:29 Oeieyiaey (eaoi)
Nmap scan report for 45.86.66.199
Host is up (0.21s latency).
Not shown: 98 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
49154/tcp open  unknown
Warning: OSScan results may be unreliable because we could not find at least
Device type: general purpose
Running: Microsoft Windows 7
OS CPE: cpe:/o:microsoft:windows_7
OS details: Microsoft Windows 7

OS detection performed. Please report any incorrect results at https://nmap.o
Nmap done: 1 IP address (1 host up) scanned in 8.70 seconds
```

Figure B.6: Nmap