Operating System Fingerprinting using Machine Learning

Achintya Kumar, Ishan Soni, and Anand Kumar M[1]

Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
{achintya.191it203, ishu.191it121, m_anandkumar}@nitk.edu.in

Abstract. Operating system fingerprinting is used to determine the operating system running on the target system. Generally, while identifying an operating system, rule-based methods are used to identify the operating system. However, problems are seen when the operating system is novel and not widely used and sometimes when the packet information is not enough. Here, we have compared the operating system fingerprinting potential of different machine learning techniques, particularly Artificial Neural Networks, K-nearest neighbour, Decision Trees, Random Forest, and Naïve Bayes. The results show that better operating system identification can be attained through machine learning techniques using encrypted traffic of systems connected to the network to a commercial non-machine learning based method.

Keywords: Operating Systems · Machine Learning · Fingerprinting · Operating System Identification · ANN

1 Introduction

Since everyone is linked to the internet these days, being safe from breaches and incursions is crucial. For businesses, this external danger causes them to investigate various security alternatives, such as firewalls and intrusion detection mechanisms, to protect themselves against hackers. Operating system fingerprinting is a much-needed approach for spotting and identifying a target machine's identity by looking at the TCP/IP packets it generates consistently. The most generally used technique in the market is to employ rule-based matching methods to identify the OS. Unlike machine learning, this approach does not require a significant quantity of data and the speed for identification to take place is also very quick. In cases of insufficient information from the packets received for identification due to network settings, newer versions, or other factors, the method will not recognize the operating system, and the resulting accuracy will below.

Operating System fingerprinting techniques are categorized into two categories, active and passive. In Active fingerprinting, packets are sent to a target and received packets are analyzed. Nmap is a vital tool in this regard and is generally

used by network admins for security and testing purposes. Using Nmap, one can ensure that all of the firewalls in their network are appropriately configured, and the TCP/IP stacks are not malfunctioning. Passive fingerprinting works by sniffing the TCP/IP ports rather than using extra bandwidth for requests. Passive OS detection has gained recent interest in identifying the host with no trace left behind. For this detection technique, the primary focus is upon the different parameters of packet headers, some of which are window size, do not fragment bit, time to live(lifetime) and TCP flags.

In this paper, we use the Passive Fingerprinting method by analyzing TCP network packet header info as well as info from HTTP header using several machine learning methods, such as K nearest neighbours (KNN), Artificial Neural Network, Decision Trees, Naive Bayes and Random Forest.

The motivation for this study was to decide the most suitable OS fingerprinting approaches as the present tools in use for OS fingerprinting were not accurate enough and were unable to detect dissimilarity in many cases. Moreover, many modern operating systems have default policies and firewalls that messes with the network services which in many cases might result in lack of data for proper identification.

The rest of the paper is broken into six sections. Section 2 deals with the literature survey, section 3 explains the framework proposed,section 4 discusses the dataset used for the study, section 5 describes the methodology used. In section 6, the outcomes of the experimental work are analyzed. At last, the conclusion is discussed in section 7.

2 Literature Survey

There has been some research for active and passive fingerprinting techniques in the last 12-15 years. Lanze Spitzner[6] was the first to identify what passive OS fingerprinting was, how it worked and the uses cases. They also extensively compared both fingerprinting techniques using a wide array of tools. Al-Shehari et al. [1] had proposed machine learning techniques combined with traditional tools to build a system that can set up TCP/IP communication between different machines and then capture and inspect the TCP/IP packets for significantly better OS detection. Similarly, Takashi Matsunaka et al.[18] used DNS Traffic Analysis by analyzing the data sent by each OS and extract the characteristics for OS fingerprinting such as interval time pattern of DNS queries,OS specific query etc. Then, examine the estimation method by using DNS traffic in their own intra-network.
Lippmann et al. [5] had an interesting approach for near-match fingerprints, where they used machine learning classifiers to determine the most detectable OS categories that used fingerprinting. Tyagi et al. [4] also had the similar ap-

proach of using TCP/IP communication for identifying prohibited operating systems on private internal networks. For optimization and quick results, Yufei Gu et al.[6], focused on using only memory for fingerprinting and caching the codehash of the kernel from guest Operating System for faster results.
Jinho Song et al. [2] analyzed the identification capabilities of several ML methods with each having unique approach for classifying. The Models were based upon Decision Trees, K nearest neighbours and Artificial Neural Networks and showed 94% probability of getting the prediction correct. They found ANN to perform best of three when the dataset was large and adequately trained. KNN was second in line with no bounds to data size and performed consistently. In the meanwhile, Robert Beverly[12] also used Naive Bayes Classifier for the same approach.
This paper draws inspiration from such authors' Machine Learning approach to OS fingerprinting and has employed many such methods from different author's research.

Table 1. Summary of Literature Survey

Authors	Methodology	Merits
Taher Al-Shehari et.al [1]	TCP/IP header packet info for ML and new extended tool	Simple algorithm and use in real life
Martin et.al [11]	Used TLS Fingerprints for OS Identification	High accuracy as TLS , TCP/IP and HTTP headers are used
Jinho Song et.al [2]	Analysis of OS identification using ML techniques	Employed many ML models for higher accuracy
Gu Yufei et.al [7]	Memory-Only Operating System Fingerprinting in the Cloud	Very quick results and wide range
Tyagi et.al [4]	TCP SYN packets for OS fingerprinting	Easy to compute and gather data
Takashi et al. [18]	Analysis of DNS traffic	Novel approach, works better in some cases

3 The Proposed Method

This paper deals with the following problem statement: To Determine the best ML classifier and the most influencing parameter.

The problem statement can be broken further into two parts:

– How to analyze all the different classifiers to find the most suitable classifier taking size, time taken and other costs into consideration.

– How to determine which parameter(s) play a major role in identification of operating systems and are necessary for fingerprinting.

Many authors and researchers use TCP/IP and HTTP features for passive OS fingerprinting and TLS features for fingerprinting specific browsers. We propose a system that combines both of these features to identify all kinds of desktop and mobile(handheld) operating systems.

We have used conventional machine learning algorithms such as Decision Trees, K-nearest neighbours and Artificial Neural Networks from [2] and tried some of the newer algorithms such as Random Forest and Bayes algorithm too in the proposed implementation. We have also analyzed the probability of correct identification across different operating systems and the role of various parameters involved in the process.

Fig.1 shows the architecture of OS fingerprinting by combining Machine Learning Classifier and Conventional rule-based matching methods. Considering the cost of computation and time taken, the proposed model first uses cheap and quick method of identification using Rule based matching method and incase of inconclusive or partial match, the data from TCP/IP packet headers and HTTP headers go through ML classifier and then the results of both methods are compared. In case of no discrepancy, the output is given as a result.

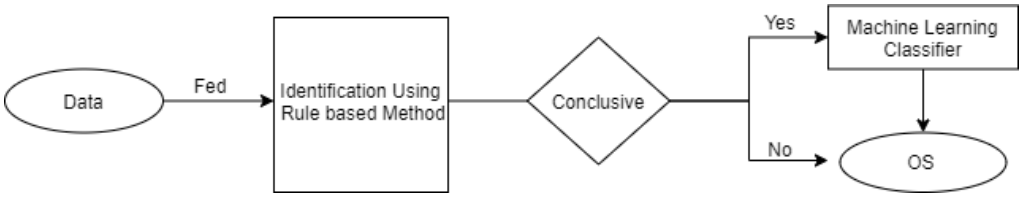

Fig. 1. Proposed Model

4 Dataset and model setup

The dataset was acquired from [4] paper where authors had posted the dataset on Zenodo.org. It contains data from TCP/IP network, HTTP connection and other metadata from the connection. Following are the different fields available in the dataset.

– Metadata about network

  - Beginning of connection
  - End of connection
  - Port used
  - Src IPv4(address)
  - Dst IPv4(address)
  - receiver port

- Features of HTTP
  - HTTP UA OS,MAJ,MIN,BLD (information about major/minor versions of OS)
  - HTTP Hostname

- TCP/IP features
  - SYN size(packetsize)
  - TCP SYN TTL(Time to live)
  - TCP win(size of window)

After careful consideration and testing, some of the parameters were used for training and testing the model. Unique identifying datafields were removed and rest were considered for the study.

- SYN size
- TCP win
- TCP SYN TTL
- HTTP UA OS
- HTTP UA OS MAJ
- HTTP UA OS MIN
- HTTP UA OS BLD
- Ground Truth OS

4.1 Pre-Processing

As shown in the Dataset Description, the data consisted of HTTP Features, network packet information and other metadata. Each attribute had some blanks for specific instances, so the dataset was trimmed and prepared for the experiment. We took 232391 instances of data where approximately 80% of the data is for training the model, and the rest 20% is used to test it.

4.2 Model Setup

In the mentioned study, the model incorporates one layer each for input, hidden and output. Seven attributes values were used for the input layer: SYN size, TCP win, TCP SYN, TTL, HTTP UA OS, HTTP UA OS MAJ, HTTP UA

OS MIN, HTTP UA OS BLD. The output layer was configured to concieve four outputs which covered basic and widely used Operating systems namely Windows, Linux, Android and MAC.

The calculated loss rate was close to 0.01% with near perfect accuracy using testdata. The Models were tested repeatedly with testdata to record fluctuations in output and corrected accordingly till the changes were insignificant

5 Methodology

This study compared the widely used Machine Learning algorithms suitable for our usecase, They consist of Decision Trees, KNN, Random forest, Bayes and Artificial Neural Network algorithms. Each of the algorithm is explained below, along with their advantages and disadvantages and the approach taken in their implementation.

5.1 Decision trees

Decision trees initially learn, then forms decisions for splitting, and finally outputs in a tree-like structure. It has the advantages of not converting data into the decimal, less data cleaning required, and it works fast when the tree's depth/height is specified. However, since depth has a significant role, results change sometimes. For our implementation, Depth=(5, 15) was used.

5.2 Artificial Neural Networks

ANNs are machine learning algorithms that are meant to learn from data patterns. It is separated into three layers, as indicated in Figure 3, input, hidden and output layers.The input layer takes the data from the user/source There is not any limit to the number of layers. It offers the advantages of being fault-tolerant, great accuracy when there is a massive volume of data contrary to other machine learning algorithms. Overfitting is a problem when the dataset is not too large.

5.3 K Nearest Neighbours

KNN is a machine learning method that uses the closest neighbours' info by measuring separation to previous data when novel data is entered. The distance is calculated using the Euclidean calculation method. The KNN algorithm shines when it requires fast processing speed, and comparison data is not significant because, unlike others, learning is not necessary. Still, when the dataset is on the smaller side, performance takes a hit. For the KNN model, three separate learning models, with K = 5, 40, 100 , were implemented.

5.4 Random Forest

Random Forest is based upon ensemble learning, which combines multiple classifiers to solve a complex problem and slightly improves performance. More specifically, Random Forest is a classifier that employs many decision trees and takes its average to improve the accuracy of the dataset. Although it combines the Decision trees, it takes a considerably big time and is not particularly good for OS that is rare to be seen. For Random Forest model, three separate learning models, with number of trees = 10, 50, 120 variations were implemented.

5.5 Naive Bayes

Naive Bayes algorithm is loosely based on the famous Bayes Theorem of probability and statistics. It is simple and yet one of the powerful ML algorithms today and is categorized as a probabilistic classifier. One of the reasons for this is that it assumes one feature in a class does not affect the other. The drawback of Bayes is that it does not relate all the parameter together and treats everything independent, which can prove results to be unpredictable at times. Multiple runs were done using this model and the average was taken.

In table 2, A summary comprising of the algorithms used and the different models implemented per algorithm is shown. Furthermore, a comprehensive list of limitations and overall accuracy of all the algorithms tested is discussed in table 3.

Table 2. Accuracy for different parameter settings

Algorithm	Parameters	Accuracy
Decision tree	Depth=5	93.96%
	Depth=10	95.02%
	Depth=15	95.22%
KNN	K=5	91.26%
	K=40	96.17%
	K=100	96.25%
Random Forest	Trees=10	92.46%
	Trees=50	94.18%
	Trees=120	95.88%

6 Experimentation and Results

This section explains the various different experiments performed, metrics used and the results obtained through each experiments by breaking them into subsections.

6.1 Comparing ML algorithms

Here, In fig 3, We can see different models compared using metrics such as precision, accuracy etc. Here, Y-axis denotes the percentage and X-axis the algorithm used. F1-score is scaled to 100 for percentage depiction purposes. KNN appeared as the best ML model for OS identification. Random Forest Classifier also performed well and overtook KNN when the dataset was huge.

Table 3. Comparison of Algorithms

Model	Limitations	Accuracy
Decision trees	Unstable and High training time	95.62%
K-NN	In the case of large data, the speed suffers.	96.22%
ANN(Artificial Neural Network)	For smaller quantity of data, accuracy rate declines and possible Overfitting occurs.	75.22 %
Naive Bayes	All features are assumed to be independent,the relationship between features is not considered.	79.88 %
Random Forest	It is not suitable for rare outcomes and overfitting problems possible	95.88 %

6.2 Comparison between parameters

A study was also done on the parameters involved to recognize the parameter that had the most effect on identification. For this we took an approach of removing parameters one at a time and retraining the proposed model without the removed part.The parameter whose absence showed the greatest significant decline was termed the most influencing parameter. As shown in table 4, it is evident that TCP SYN TTL [shows a decline of 23.65%] is the most influencing parameter of the bunch.

6.3 Comparing ease of prediction across OS

We also took time to compare the accuracy of prediction across different operating systems. We can see the chart comparing the accuracy of predicting different

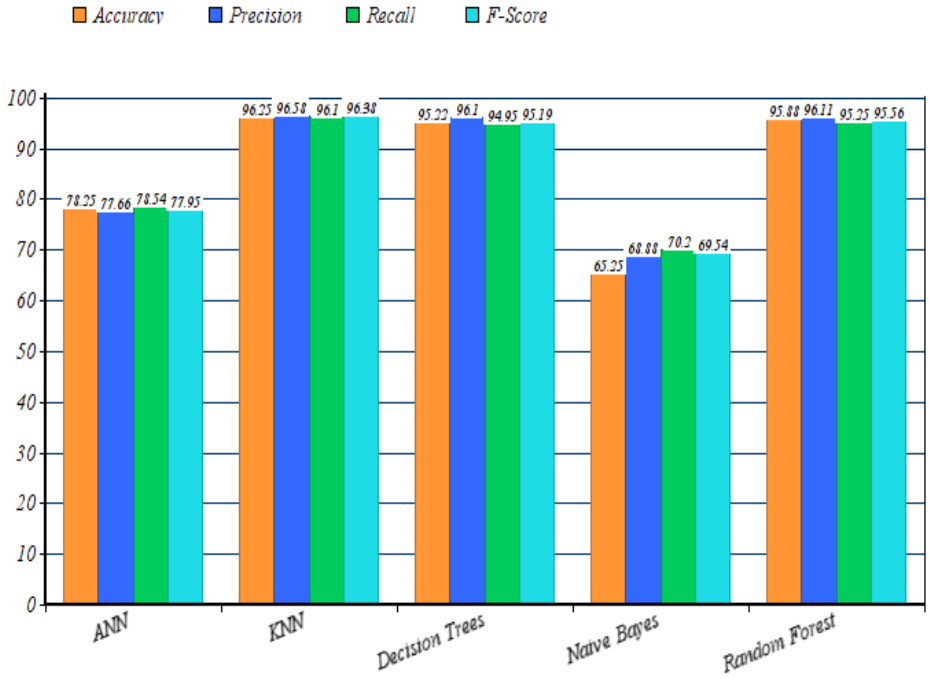

Fig. 2. Comparison between different Models

operating systems in fig 3 with Operating System category on y-axis and percentage accuracy obtained for each operating system in x-axis.
Comparing the Results with OS identification solutions that use traditional Rule-based matching methods, an overall significant raise of 20% accuracy was observed over recognized operating systems and the 5% improvement for unknown data samples.

6.4 Comparison with existing tools

We compared the KNN, which has the best accuracy, to some of the most popular alternatives present in the market. In our case, p0f was used for Operating System identification. WireShark is a network packet analyzer that was used to capture essential data from the data obtained in p0f for comparison. The data from 4 different types of Operating systems (Mac, Linux, Windows and Android) was available, disregarding the different versions and distributions available for the same OS.

A total of 1956 known and 490 unknown data samples from p0f were used for this study. NetworkMiner, another OS fingerprinting tool that uses only packet

Table 4. Influence of Parameters used

Parameter	Description	Decline
SYN Size	Request sent before a connection	16.44%
TCP SYN TTL	The lifecycle of a single TCP SYN packet.	23.65%
TCP win	The window size of TCP connection	17.63 %
HTTP UA OS	HTTP User-agent Operating System	19.28 %
HTTP UA OS MAJ	HTTP user-agent Major Version of the OS	5.88 %
HTTP UA OS MIN	HTTP user-agent Minor Version of the OS	4.16 %
HTTP UA OS BLD	HTTP user-agent Build Version of the OS	9.39 %

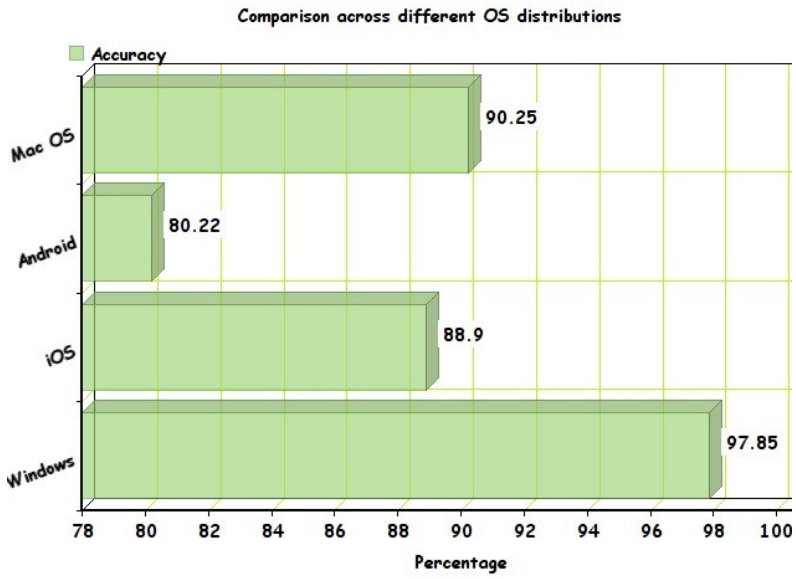

Fig. 3. Comparison between different OS accuracy

sniffing for identification, was used to obtain the anonymous data.
In our results, we found that our best performing model KNN gave an accuracy of 95% (fig 3) , with 72% probability for the unknown OS. Among 1956 datasets used, p0f had a precision of 52%, and the Artificial neural network model correctly identified the OS with a chance of 79%.

7 Conclusion and Future Scope

This proposed model using machine learning and Operating system attributes (SYN size, TCP win, TCP SYN TTL, HTTP UA OS, HTTP UA OS MAJ, HTTP UA OS MIN, HTTP UA OS BLD), achieved probability of accurate determination of OS was more than 96%, much higher than traditional methods. On the other hand, individual OS versions could not be precisely categorized due to them having similar attribute values and generally little implementation changes between them. Moreover, recently launched Operating Systems could not be identified in many cases in the rule-based strategy as the information about them is scarce.

Comparing the parameter's influence for identification of the Operating system, one can infer that the TTL(Time to Live) of a SYN Packet differs across different operating systems. Using TLS features with HTTP parameters enhanced the machine learning model's efficiency, and as a result, we found that, the proposed model, using the Machine Learning approach in tandem with the conventional rule-based matching method can yield better results than the tools we use now.

References

1. Taher Al-Shehari and Farrukh Shahzad : *Improving Operating System Fingerprinting using Machine Learning Techniques*: International Journal of Computer Theory and Engineering, Vol. 6, King Fahd university of petroleum and minerals, Saudi Arabia
2. Jinho Song, ChaeHo Cho, Yoojae Won: *Computers and Electrical Engineering 78 (2019)* : Chungnam National University,Korea
3. Blake Anderson and David McGrew: *OS Fingerprinting: New Techniques and a Study of Information Gain and Obfuscation*:2017 IEEE Conference on Communications and Network Security (CNS), Cisco Systems
4. Tyagi, Rohit Paul, Tuhin Bs, Manoj B., Thanudas. (2015): *Packet Inspection for Unauthorized OS Detection in Enterprises.* : IEEE Security Privacy. 13. 60-65.
5. R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein: *Passive Operating System Identification from TCP/IP Packet Headers* :IEEE Workshop on Data Mining for Computer Security (DMSEC), 2003, pp.40–49.
6. L. Spitzner:*Passive fingerprinting* : vol. 3, pp. 1–4, May 2003.
7. Yufei Gu,Yangchun Fu,Aravind Prakash ,Zhiqiang Lin,Heng Yin: *OS-SOMMELIER: Memory-Only Operating System Fingerprinting in the Cloud* : SOCC'12, October 14-17, 2012, San Jose, CA USA

8. Gregory Conti and Kulsoom Abdullah:*Passive Visual Fingerprinting of Network Attack Tools* :In ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)pages 45–54, 2004..
9. Tim Dierks and Eric Rescorla. :*The Transport Layer Security (TLS) Protocol* : Version 1.2. RFC 5246 (Proposed Standard), 2008.
10. Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson: *The security impact of https interception.* : In Network and Distributed Systems Symposium (NDSS17), 2017
11. Charles Elkan: The Foundations of Cost-Sensitive Learning:*In International Joint Conference on Artificial Intelligence.* ,(IJCAI): pages 973–978, 2001
12. Stephan Friedl, Andrei Popov, Adam Langley, and Emile Stephan: *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension* :RFC 7301 (Proposed Standard), 2014
13. Beverly, Robert:*A Robust Classifier for Passive TCP/IP Fingerprinting* : Passive and Active Network Measurement, Springer Berlin Heidelberg
14. Lastovicka, Martin and Spacek, Stanislav and Velan, Petr and Celeda, Pavel: *Using TLS Fingerprints for OS Identification in Encrypted Traffic* : NOMS 2020-2020 IEEE,pages 1-6 04/2020
15. Lloyd Greenwald and Tavaris Thomas T: *Toward Undetected Operating System Fingerprinting.* : In USENIX Workshop on Offensive Technologies (WOOT),pages 1–10, 2007
16. Martin Husák, Milan Čermák, Tomáš Jirsík and Pavel Čeleda: *HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting.* : EURASIP Journal on Information Security volume 2016, Article number: 6 (2016)
17. M Majkowski:*SSL fingerprinting for p0f* : https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/
18. T Matsunaka, A Yamada, A Kubota:*Passive OS fingerprinting using DNS traffic analysis.* : Advanced Information Networking and Applications, (2013)
19. Jon Mark Allen:*OS and Application Fingerprinting Techniques* : SANS.edu Graduate Student Research
20. Ofir Arkin: *Fun with IP Identification Field Values(Identifying Older MS Based OSs)* : RFC 0791
21. Zalewski Michal: *p0f v3 (version 3.09b)* : https://lcamtuf.coredump.cx/p0f3/README