Proceeding Paper

Applying Artificial Intelligence for Operating System Fingerprinting †

Rubén Pérez-Jove [1,2,*], Cristian R. Munteanu [1,2,3], Alejandro Pazos Sierra [1,2,3] and José M. Vázquez-Naya [1,2]

1 Grupo RNASA-IMEDIR, Departamento de Ciencias de la Computación y Tecnologías de la Información, Facultade de Informática, Universidade da Coruña, Elviña, 15071 A Coruña, Spain; c.munteanu@udc.es (C.R.M.); alejandro.pazos@udc.es (A.P.S.); jose@udc.es (J.M.V.-N.)

2 Centro de Investigación CITIC, Universidade da Coruña, Elviña, 15071 A Coruña, Spain

3 IKERDATA S.L., ZITEK, University of Basque Country UPVEHU, Rectorate Building, 48940 Leioa, Spain

* Correspondence: ruben.perez.jove@udc.es

† Presented at the 4th XoveTIC Conference, A Coruña, Spain, 7–8 October 2021.

Abstract: In the field of computer security, the possibility of knowing which specific version of an operating system is running behind a machine can be useful, to assist in a penetration test or monitor the devices connected to a specific network. One of the most widespread tools that better provides this functionality is Nmap, which follows a rule-based approach for this process. In this context, applying machine learning techniques seems to be a good option for addressing this task. The present work explores the strengths of different machine learning algorithms to perform operating system fingerprinting, using for that, the Nmap reference database. Moreover, some optimizations were applied to the method which brought the best results, random forest, obtaining an accuracy higher than 96%.

Keywords: operating systems; fingerprinting; Nmap; machine learning

check for updates

Citation: Pérez-Jove, R.; Munteanu, C.R.; Pazos Sierra, A.; Vázquez-Naya, J.M. Applying Artificial Intelligence for Operating System Fingerprinting. *Eng. Proc.* **2021**, *7*, 51. https://doi.org/10.3390/engproc2021007051

Academic Editors: Joaquim de Moura, Marco A. González, Javier Pereira and Manuel G. Penedo

Published: 26 October 2021

1. Introduction

The aim of operating system (OS) fingerprinting is to identify which family and version of OS is running behind a device analyzing the network traffic it generates. Knowing the specific details of the system controlling a machine is interesting in the way that it can support the detection of unauthorized devices in a network or assist in determining the vulnerability of a target host. One of the most known and spread tools to perform this task is Nmap [1], which can carry out active OS detection, as well as XProbe2 [2]. On the other hand, passive OS detection can be performed with the tool P0f [3].

Traditionally, the way of performing this task is a rule-based approach, followed by Nmap in its IPv4 analysis [4]. The process can be summed up in three steps: sending specific probes to the target, recollecting and parsing the responses, executing a set of tests onto these responses in order to generate a characteristic signature, and finally comparing that stamp with every single entry of its database of preprocessed signatures in turn.

The main aim of the present project is to develop a PoC of an operating system fingerprinting model based on the latest Nmap database, analyzing which classical machine learning algorithms provide better results. In fact, this approach, based on the logistic regression technique, is already followed by the tool in its IPv6 scan.

2. Materials and Methods

The process followed in this work can be split into two well distinguished phases, starting with the extraction and preparation of the data in a suitable format for the second stage, where the training and testing tasks were performed.

2.1. Dataset Preparation

The OS database of the latest version of Nmap (7.9) [5] was downloaded as the base point to construct our dataset. This file contained all of the fingerprints gathered by the community and grouped by Nmap, since 1998, in text format. A specific fingerprint consists of the collection of the results of the predefined tests that Nmap executes against the responses received of a particular known OS.

In order to simplify our approach, we filtered those fingerprints by only selecting those from a group of OS families representing the most widespread systems in use nowadays: Android, BSD, Linux, Solaris, Windows, iOS, and macOS. All these features represent different types of information, such as numerical or boolean values, as well as specific categories. For all these types, a codification keeping the idea of transforming the data to numerical values was chosen. In this way, the null or absent values were codified as -1.

Besides, a condition between more than one value could be specified, expressed as a combination of any of these operators: boolean OR, range of values, "more than" or "less than". We represented the OR operator as a lineal combination between the possible values of all the features of a single fingerprint, creating a new row for every alternative that would have produced less than 100,000 new combinations. On the other hand, the range operator was represented as a random value generated between both the limits of the range for every single row of the same fingerprint. The "more than" and "less than" operators were ignored because of the almost absence of them in the database. In order to obtain better results in the next stage, some preprocessing of this dataset, such as removing duplicated values and near zero variance features, was done. This process finished with a dataset of 264,852 cases and 233 features.

2.2. Machine Learning Modeling

Once we had a suitable dataset that represented, in a simplified manner, the knowledge of the Nmap database, the modeling process was performed. The first step consisted of splitting randomly the dataset in a training and test set, following a proportion of 90/10, respectively, leaving the same number of cases for each class in both groups. The dataset was imbalanced, as there were not the same number of examples for every OS family. To fix this, a vector with the class weights was calculated and passed as a parameter to the training algorithms.

The set of classic machine learning algorithms tested was: Gaussian Naive Bayes (GNB), linear discriminant analysis (LDA), logistic regression (LR), multilayer perceptron classifier (MLPC), decision tree (DT), random forest (RF), and bagging classifier (Bag). For each model created with these methods, the accuracy, precision, recall, and f1-score metrics were calculated in order to compare them.

In a second improvement phase, a hyperparameter optimization was performed using the grid search cross validation method. The base algorithm used in this process was the one which raised the best results in the previous stage. We chose a list of values for a selection of the available parameters of this method and executed the grid search fitting three-fold for each candidate.

3. Results and Discussion

The metrics calculated on the models generated during this work (see Table 1) point to several potential options when it comes to performing fingerprinting of operating systems with classic machine learning methods. Specifically, the model that was developed more deeply was RF, since it yielded the best results in the first approach with an accuracy value of 0.96055. However, all of the tested models, with the exception of GNB and LR, offered prediction results higher than 90%. In fact, even choosing the RF algorithm in the second improvement stage because of its results, we could have chosen the LDA method, as, in terms of complexity, it is much simpler, meaning it could execute the category prediction faster.

Table 1. Machine Learning algorithm validation results.

Method	Accuracy	Precision	Recall	F1-Score
GaussianNB	0.10764	0.62907	0.10764	0.05814
LinearDiscriminantAnalysis	0.95734	0.96275	0.95734	0.95857
LogisticRegression	0.11013	0.78129	0.11013	0.07955
MLPClassifier	0.93147	0.94640	0.93147	0.91723
DecisionTreeClassifier	0.95216	0.95168	0.95216	0.95185
RandomForestClassifier	**0.96055**	**0.96360**	**0.96055**	**0.95844**
BaggingClassifier	0.95794	0.96102	0.95794	0.95782

After choosing RF as the best option for this problem, we attempted to improve its results carrying out some optimizations. The grid search method applied generated 648 different models, where the concrete parameters of the best generated model was: bootstrap = False; max_depth = 20; max_features = auto; min_samples_leaf = 1; min_samples_split = 5; n_estimators = 50. This progress allowed us to get an accuracy of 0.96096. Besides, some ideas can be concluded from the confusion matrix of this latest model. In general, the solution responds correctly to the classification of every category, but it has some problems in distinguishing between Android and Linux, as well as between iOS and macOS. Taking into account that Android is in its basis a Linux system, and that iOS and macOS are both the mobile and laptop operating systems of Apple, these results show that the model is capable of learning the generalizations and main differences between families of operating systems without being over-fitted to all the specific cases of each cases.

This work was an initial approach to the problem of fingerprinting operating systems using machine learning. With restrictions, in terms of execution time and the amount of computational resources, a simplified dataset was created, and some basic models were generated. In spite of being a prototype, the obtained results evidence that this kind of work can be successfully conducted with classical machine learning techniques, with an acceptable grade of complexity of the process. In this researching line, there are plenty of improvements and other different approaches that can be performed in order to get more effective models, such as balancing the dataset, improving the codification of the features, scaling its values, or attempting to apply autoML or deep learning to the problem. It is worth mentioning that the code developed during this work is publicly available in the following GitHub repository: https://github.com/rubenperezudc/osfingerprintingia (accessed on 25 October 2021).

Author Contributions: Conceptualization: R.P.-J., C.R.M., and J.M.V.-N.; methodology: R.P.-J., C.R.M., and J.M.V.-N.; software: R.P.-J. and C.R.M.: formal analysis: R.P.-J. and C.R.M.; investigation, R.P.-J. and C.R.M.; resources: R.P.-J., C.R.M., A.P.S. and J.M.V.-N.; data curation: R.P.-J. and C.R.M.; writing—original draft preparation: R.P.-J.; writing—review and editing: R.P.-J., C.R.M., A.P.S., and J.M.V.-N.; visualization, R.P.-J. and C.R.M.; supervision, C.R.M., A.P.S., and J.M.V.-N. All authors have read and agreed to the published version of the manuscript.

Funding: CITIC, as a research center accredited by the Galician University System, is funded by "Consellería de Cultura, Educación e Universidade from Xunta de Galicia", supported—80% through ERDF, ERDF Operational Programme Galicia 2014–2020, and the remaining 20% by "Secretaría Xeral de Universidades (Grant ED431G 2019/01). This project was also supported by the "Consellería de Cultura, Educación e Ordenación Universitaria" via the Consolidation and Structuring of Competitive Research Units–Competitive Reference Groups (ED431C 2018/49) and the COST Action 17124 DigForAsp, supported by COST (European Cooperation in Science and Technology, www.cost.eu, (accessed on 25 October 2021)).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nmap: The Network Mapper—Free Security Scanner. Available online: https://nmap.org/ (accessed on 31 July 2021).
2. XProbe2—Linux Man Page. Available online: https://linux.die.net/man/1/xprobe2 (accessed on 31 July 2021).
3. p0f. Available online: https://lcamtuf.coredump.cx/p0f3/ (accessed on 31 July 2021).
4. Chapter 8. Remote OS Detection Nmap. Available online: https://nmap.org/book/osdetect.html (accessed on 31 July 2021).
5. Operating Systems Fingerprint Database (version 7.9) Nmap. Available online: https://svn.nmap.org/nmap-releases/nmap-7.90/nmap-os-db (accessed on 31 July 2021).