applied sciences

MDPI

Article

Sisyfos: A Modular and Extendable Open Malware Analysis Platform

Dimitrios Serpanos [1,2,*], Panagiotis Michalopoulos [2], Georgios Xenos [1,2] and Vasilios Ieronymakis [2]

[1] ATHENA Research Center, Industrial Systems Institute, GR-26504 Patras, Greece; gxenos@upnet.gr
[2] Department of Electrical and Computer Engineering, University of Patras, GR-26504 Patras, Greece; ece8125@upnet.gr (P.M.); bma2476@upnet.gr (V.I.)
* Correspondence: serpanos@ece.upatras.gr

Abstract: Sisyfos is a modular and extensible platform for malware analysis; it addresses multiple operating systems, including critical infrastructure ones. Its purpose is to enable the development and evaluation of new tools as well as the evaluation of malware classifiers. Sisyfos has been developed based on open software for feature extraction and is available as a stand-alone tool with a web interface but can be integrated into an operational environment with a continuous sample feed. We present the structure and implementation of Sisyfos, which accommodates analysis for Windows, Linux and Android malware.

Keywords: malware analysis; static malware analysis; dynamic malware analysis; malware classification; machine learning; random forest; support vector machines

Citation: Serpanos, D.; Michalopoulos, P.; Xenos, G.; Ieronymakis, V. Sisyfos: A Modular and Extendable Open Malware Analysis Platform. *Appl. Sci.* **2021**, *11*, 2980. https://doi.org/10.3390/app11072980

Academic Editor: Leandros Maglaras

Received: 22 February 2021
Accepted: 23 March 2021
Published: 26 March 2021

1. Introduction

Malicious software undoubtedly poses a serious threat to the security of computer systems. In the last decade, malware has been widely used by threatening actors to target private corporations, public organizations and individuals. Ransomware, for example, is increasingly used to attack major companies, organizations and persons; recent cases include Advantech, Canon and Cognizant, whose computer systems were encrypted by the attackers [1]. Recently, attackers also used malware to target government organizations and critical infrastructure with novel attacks, e.g., SolarWinds attacks [2]. Such attacks not only lead to serious data leakage but also have significant financial impact due to damages. It is estimated that the yearly cost of malware will surpass $ 6 trillion by 2021 [3]; the AV-TEST institute detects over 350,000 novel malware samples and potentially unwanted applications every day [4].

To defend against such attacks, a great deal of effort has been made to design effective malware detection and analysis systems. Researchers often use features derived from static analysis, in which information is extracted from a binary file without executing it [5–7]. As malware is usually heavily obfuscated and static analysis has limits [8], another approach is the use of dynamic analysis, where a suspicious program gets executed in a virtual environment and certain measurements are made [9–14]. Then, to differentiate between malware and benign files, classification techniques are employed, including machine learning algorithms. Common machine learning approaches include algorithms like random forests or support vector machines [15–18], while the use of deep learning neural networks is becoming increasingly prevalent [19–22].

Several efforts have been made to provide complete end-to-end solutions for malware analysis and detection; these include static analysis, dynamic analysis and learning models, which prove to be quite efficient at detecting specific pieces of malware on specific systems. Existing tools and platforms of wide use include VirusTotal [23], Joe Sandbox [24], Hybrid Analysis [25] and ANY.RUN [26], where a user uploads a suspicious sample file and receives a complete report that contains information gathered from both static and dynamic

analysis. Alternatives include systems like the VMRay Analyzer and Detector [27], where a malware analysis platform is either installed on-premises or is deployed in the Cloud. While these services are effective at analyzing and detecting malware, they typically include proprietary tool chains and require paid subscriptions.

Importantly, since malware classifiers adopt learning systems, they require significant amounts of reliable data for effective and efficient training. Up to date, no public data are available for malware classifier training and, actually, there is a significant lack of common data for training and comparison of alternative analyzers and classifiers. The inherent security and privacy concerns make it very difficult for researchers and private companies alike to publicize and share their data, thwarting collaboration among researchers.

To address these issues, we have developed and employed Sisyfos, a novel extensible and openly available malware analysis platform. Sisyfos has been developed based on a set of open software tools for feature extraction (from static and dynamic analysis) and integrates our own classifiers; the tools have been integrated through an orchestrator that has been designed to provide a robust and modular environment for malware analysis and classification. Although Sisyfos is a platform suitable for educational and research environments, for training and experimentation with new tools and classifiers, promoting collaboration within the research community, it also integrates into operational environments effectively and efficiently, providing invaluable feedback for the performance and requirements of operational environments.

The paper is organized as follows: Section 2 briefly introduces the general architecture of Sisyfos and provides a description of its functionality; Section 3 describes the implementation of Sisyfos; and Section 4 presents our classifiers, which employ appropriate machine learning models that are effective in malware detection.

2. The Architecture of Sisyfos

Sisyfos is a general-purpose platform for malware analysis and is organized in three stages: static analysis, dynamic analysis and classification. Sisyfos accepts as input a software sample and extracts its classification category, currently as malware or benign software. Figure 1 depicts the structure of Sisyfos, showing a pipeline of analysis and classification steps.

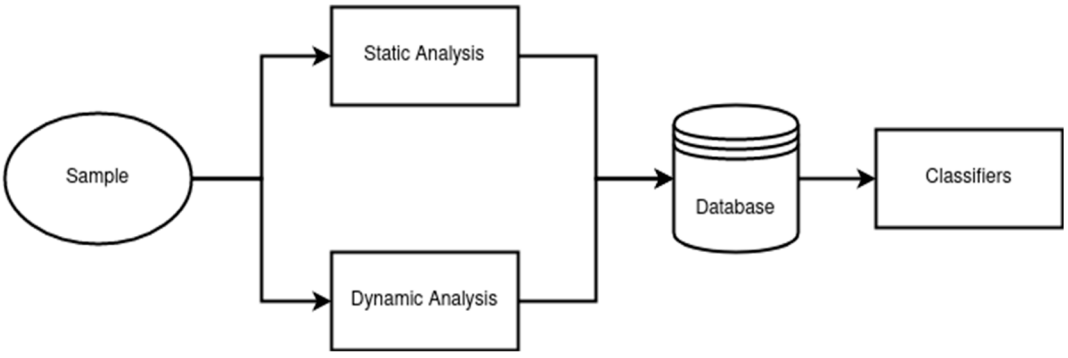

Figure 1. Sisyfos platform architecture.

Static analysis constitutes the first stage of analysis of a processed sample. Static analysis extracts features and measures parameters of the software sample through tools that analyze the sample code without executing it. This kind of analysis allows for quick and computationally inexpensive feature extraction; such typical features include information about the different sections of a file and the resources the executable is using as well as the calls to external libraries or to the underlying operating system. Static analysis includes scanning the binary sample to detect possible YARA rule matches. Sisyfos also submits the sample's hash to VirusTotal in order to obtain the existing analysis reports in the case that the sample has been analyzed in the past by VirusTotal; VirusTotal is a publicly available service integrating a multitude of well-known antivirus systems [23].

After the features are extracted and measured, analysis continues with the second stage, dynamic analysis. Dynamic analysis is the process by which the sample is executed inside a safe and isolated environment (a sandbox) and various behavioral characteristics related to its execution are collected. These features can be broadly categorized in the following general categories: file system, registry, network, and dynamic signatures. The first category contains features that describe the interaction of the sample with the file system, such as the number of files it created or opened. The second relates to Windows PE executables and contains information related to the Windows Registry, such as the keys accessed by the sample. The third category summarizes the network activity of the sample. It contains the number of TCP and UDP connections that were established, the number of unique IPs the sample connected to, the number of DNS queries performed, etc. Finally, the last category contains the names of the dynamic signatures created by the sandbox, which can be used to summarize the overall execution of the sample. These signatures are created by parsing the memory dump and the execution trace of the sample and searching for suspicious patterns, such as the allocation of read-write-execute memory areas (a possible indication of payload unpacking) or the injection of code into child processes. The raw results of the dynamic analysis, along with the extracted features, are stored in a central database, from which they can be accessed at later stages of analysis.

Finally, the features and measurements that result from the static and dynamic analysis are fed to the classification stage, which classifies the sample according to a category of the used classification scheme.

Sisyfos is built mainly with open software tools that are combined, which process and extract features of the analyzed software sample (the suspected malware), in order to provide accurate metrics and enable effective detection and classification of the sample. The platform has been designed to be highly modular and easily expandable. Its modularity and scalability are key to its success, since new tools can be easily and quickly integrated; Sisyfos' easy setup and highly scalable design enable implementation with various sizes of organizations, from small private enterprises to large public organizations.

3. The Implementation of Sisyfos

Sisyfos is implemented by exploiting open software. Various open software tools for static and dynamic analysis have been employed with the platform, whose implementation is shown in Figure 2. Dynamic analysis is performed using sandboxes that are implemented with VirtualBox through the Cuckoo Sandbox [28] for Windows and Linux; currently, work is under way to integrate the Mobile Software Framework (MobSF) for Android [29] to Sisyfos.

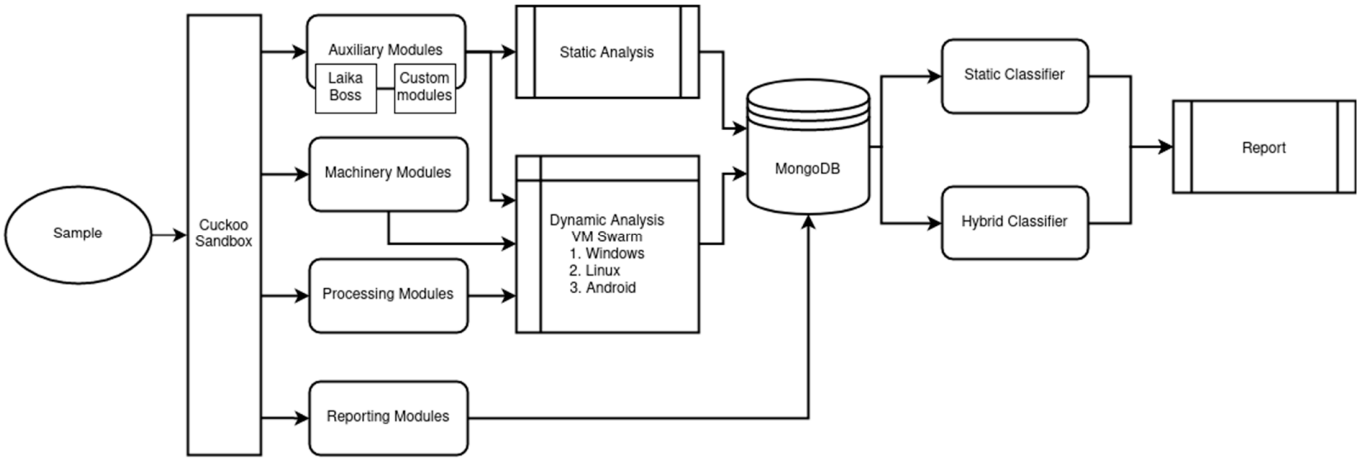

Figure 2. Sisyfos implementation diagram.

The tools have been integrated using a non-trivial orchestrator which achieves two main goals, resilience and modularity. For the first goal, the orchestrator spawns a new process for every submitted sample, which manages the analysis (e.g., interacts with Cuckoo, stores the results in the database and passes the sample to the classifier). This strengthens the resilience of Sisyfos, because if an analysis fails, the rest of the platform will continue normal operation; at the same time, Sisyfos' throughput capabilities increase. For the second goal, the orchestrator is constructed in a tool-agnostic way, enabling the easy replacement of one tool (e.g., Cuckoo) with another, increasing the modularity of Sisyfos. The orchestrator has been built in Python.

Importantly, Sisyfos is built so that it can be used either as a stand-alone Web service, where a user uploads a software sample for analysis, or as a service that can be integrated and automated for a continuous feed of samples in an operational setting.

In the following, we present the main implementation characteristics of Sisyfos for: (i) the analyses stages (static and dynamic) and (ii) the classification stage. The presentation focuses on Windows and Linux malware analysis.

Static analysis is implemented using two different open software tools, Cuckoo [28] and LaikaBOSS [30]. First, the executable sample is sent to Cuckoo, which processes it with its own static analysis tools. Cuckoo collects the hash of the file along with the names, addresses and sizes of its different sections. The names, offsets and sizes of the resources that are used by the binary sample are also extracted. Furthermore, Cuckoo extracts the printable strings contained in the binary; Cuckoo specifies these printable strings as UTF-8/16 strings with lengths between 6 and 1024 characters and limits the number of extracted strings to 2048. Finally, it searches for whether the sample matches any sample on VirusTotal [23] and fetches the associated signatures extracted by the detecting antivirus systems.

Cuckoo also allows for auxiliary modules to be loaded. One such module measures the entropy of the file and its different sections. Using entropy measurements, the system can detect if the file is using obfuscation techniques, e.g., packers, to avoid detection.

In parallel with Cuckoo static analysis, the sample is sent to LaikaBOSS, which is integrated as an auxiliary module in Sisyfos. LaikaBOSS, an open software tool, is an object scanner and intrusion detection system that recursively extracts child objects from a file. Such objects can be archives, wrappers or obfuscators. Finally, Laika scans the binary against a repository of YARA rules and creates a detection flag for each rule that gets matched.

Dynamic analysis is implemented using Cuckoo, taking special steps to conceal the virtual analysis environment and make it appear as a normal workstation. Three elements contribute to this. First, the Disguise module of Cuckoo and the VMCloak tool [31] change specific environmental values that could otherwise reveal the virtualization environment. Second, we install commonly found software such as office suites and web browsers and ensure the creation of dummy files in the user's personal folders, along with creating browser history. Finally, we allocate typical resources to the virtual machines (hard disk larger than 60 GB, RAM > 1 GB, and CPU cores more than 2). We have evaluated these modifications with the use of the Paranoid Fish tool [32] which tries to detect if it is running inside a virtualized environment using a wide array of techniques commonly used by malware.

During dynamic analysis, Cuckoo identifies the requirements to execute each sample and invokes the appropriate analysis module. The sandbox includes special software that may be required by the sample, e.g., PDF reader. Then, the sample is uploaded to a virtual machine and through a monitoring process, Cuckoo intercepts the system calls that are made and the network traffic that is initiated by the sample. Cuckoo is highly modular and configurable and is capable of outputting a large volume of behavioral information and execution statistics for each sample it analyses. Extracted measurements and statistics by the dynamic analysis stage include:

- File System: number of files opened, read, deleted, created, recreated, written and checked for existence;
- Registry: number of records opened and read;
- Network: number of TCP sessions, UDP sessions, unique IPs, DNS A/CNAME/PTR/MX queries, HTTP GET/HEAD/POST requests and HTTP 200/300/400/500 response status codes.

Importantly, Sisyfos implements a sophisticated interface, enabling users to upload samples to Sisyfos in two ways. The first one is through a web UI, shown in Figure 3, which has been set up as an easy-to-use web application; anyone can upload a sample that will be delivered to our platform. As Figure 3 shows, the submission page of the web interface consists of a modified version of the corresponding Cuckoo page. The backend interface of the platform, which provides real-time information for each submitted sample, is shown in Figure 4. The second way to transfer a sample to Sisyfos is through a transfer.sh service. Using this transfer service, one can send a sample to Sisyfos, at any time, through a simple curl command over the terminal; this enables connectivity of the platform to multiple network computers or many sample sources.

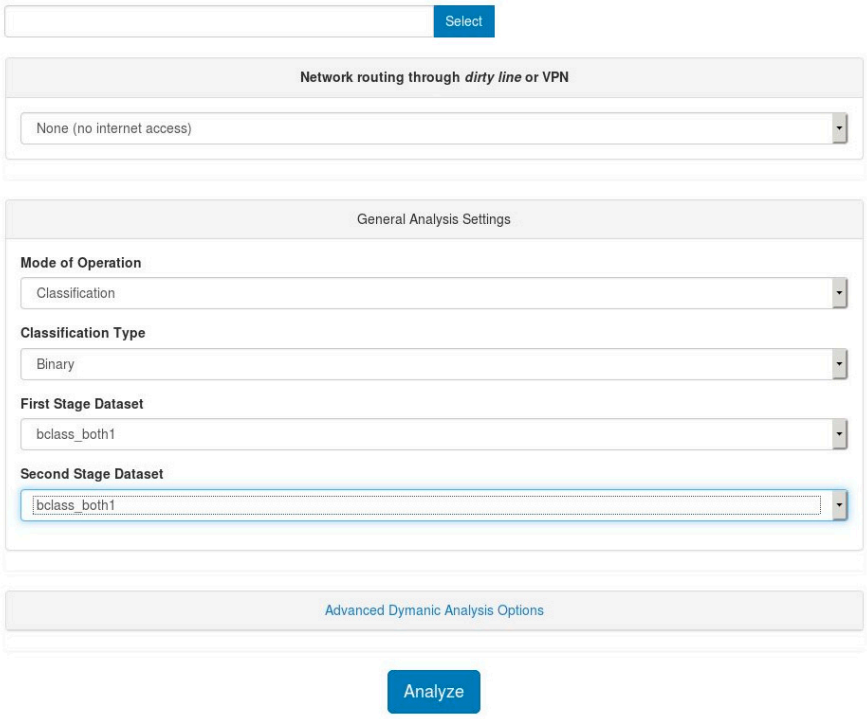

Figure 3. Sisyfos web interface.

```
4|start    | Processing file:2bff031a774971dd32470d120a73551c
4|start    | Starting static analysis for 2bff031a774971dd32470d120a73551c
4|start    | Waiting for submissions...
4|start    |  Static analysis for 2bff031a774971dd32470d120a73551c completed succesfully
4|start    | Submitting 2bff031a774971dd32470d120a73551c for cuckoo scan...
4|start    | Submission was successful with task id 2330. Now we wait for the results...
Waiting for submissions...
Waiting for submissions...
Waiting for submissions...
4|start    |  Dynamic analysis for 2bff031a774971dd32470d120a73551c terminated.
4|start    | Saving 2bff031a774971dd32470d120a73551c in the database as malwr.
Waiting for submissions...
Waiting for submissions...
```

Figure 4. Real time information we collect from each sample.

Considering privacy issues in Sisyfos, in addition to the typical disclaimer that raises the privacy risks and ascertains the corresponding liabilities to uploaders of samples in the main page, Sisyfos offers a mechanism to users to erase the uploaded samples and the analysis features from the systems, if they wish. Furthermore, Sisyfos does not have an agent that automatically collects samples from users and, thus, it does not create any privacy risks as do systems that collect client data automatically through agents [33].

4. Classification

The final step of Sisyfos operation is to classify the software sample, based on the classification scheme that is employed by Sisyfos. The platform is independent of the classification scheme and can accommodate any scheme that is associated with the features and measurements of the Sisyfos tools. Currently, we use binary classification for all samples, i.e., we classify them either as malicious or benign; this decision is made based on the information extracted during both static and dynamic analysis phases.

4.1. Static and Dynamic Classifiers

Sisyfos currently includes classifiers that exploit machine learning techniques. We have developed two different classifiers, one based on static features and one based on dynamic ones. We evaluate the effectiveness of these classifiers through a small dataset that contains 6000 Windows Portable Executable samples which include both malware and benign samples. The samples originate from VirusShare, a popular web repository of malware files. The VirusShare malware samples are augmented with benign installation files of well-known benign programs, in order to construct a balanced dataset of samples. We verified the labels of our dataset using the VirusTotal API [23]. The samples are then processed by the platform to produce feature matrices. We divide our dataset into two subsets, one containing 5000 samples used for training and one containing 1000 samples used for testing. Figure 5 shows the distribution of our dataset. Our current classification method uses approximately 500 features derived from static analysis and approximately 150 derived from dynamic analysis. Examples of such features are data extracted from the different sections of the executable, different matching YARA rules, DNS and HTTP requests, files dropped, downloaded files or detected PowerShell commands that are suspicious.

Figure 5. Sample distribution in our training and test subsets.

In order to identify the most efficient machine learning algorithm for our dataset, we use TPOT [34], an open source tool, which executes different algorithms and tries different values for the hyper-parameters and calculates the optimal combination. For our specific dataset, the optimal algorithm calculated by TPOT is the gradient boosting classifier. Table 1 presents the optimal hyper-parameters for our case and Table 2 presents the results of our static and dynamic model.

Table 1. Optimal hyper-parameters of the gradient boosting classifier.

Hyper-Parameter	Value
Number of trees	5000
Minimum samples in leaf	1
Number of features per tree	Sqrt (m) [1]
Learning rate	0.1
Max depth	9

[1] Where m is the number of all features.

Table 2. Accuracy, precision and recall of the static and dynamic models.

Model	Accuracy	Precision	Recall
Static Model	99.21%	98.78%	99.86%
Dynamic Model	96.53%	96.99%	96.99%

4.2. A Random Forest Static Classifier

Sisyfos enables sample classification exploiting static and dynamic features extracted through static and dynamic analysis as demonstrated in Section 4.1. We exploit Sisyfos to evaluate classifiers using various data sets of software samples, focusing on developing a reliable static classifier with machine learning techniques. However, the unavailability of standard datasets raises issues for the objective comparison of research efforts by different research groups. There is significant and well-known lack of standardized sample data sets in the research community; such datasets should include both malware and benign samples. Reproducible and comparable experiments require common samples and features. Although many online malware repositories exist today, such as VirusTotal [23], one would need specific parameters (such as the MD5 hash of each file) in order to extract specific sample subsets. Most of the researchers though, do not publish the specific parameters for selecting samples from such online repositories. To address these issues in our static classifier, we use one of the very few publicly available datasets, the EMBER dataset [35], which contains both malware and benign files. EMBER unfortunately does not distribute the binary files, but instead provides a feature set for each sample. Due to Sisyfos' modularity, we can train the model on those features and then we can easily integrate it into Sisyfos, as we describe below.

4.2.1. Model and Training

Our classifier adopts a model based on random forests, an algorithm that uses bagging, where many smaller noisy decision trees are averaged to produce the final model. This algorithm is selected because it can capture complex interactions between the features, and due to its inherent parallelism, can construct multiple decision trees at the same time.

The EMBER dataset [35], which was used to train the model, contains 2351 features extracted with static analysis from 1.1 million malicious and benign portable executable samples. EMBER comes with a baseline pre-trained model along with a distinct test dataset, allowing us to directly compare and evaluate the efficiency of our algorithm.

During the training phase we used features from 600,000 samples labeled as either malware (300,000 samples) or benign (300,000 samples), as shown in Figure 6.

Figure 6. EMBER dataset distribution of samples in the training set (labeled data only).

In order to boost the accuracy and reduce the training time of our model, we performed a light hyper-parameter tuning, searching only in a small subspace of commonly used values [36]. To achieve this, we used an open software tool called TPOT [34] that allowed us to choose a specific range of possible values for each hyper-parameter that it then used to find the optimal combination of values. To expedite the hyper-parameter finding process, we used only a subset of the whole dataset, specifically features from 100,000 samples: 50,000 malware and 50,000 benign, maintaining the original balance of the dataset. In order to validate the results on a separate dataset, we did an additional 80/20 training/validation split of our data; specifically, we shuffled the dataset and picked at random 20% of the samples, while maintaining again the balance of the dataset as shown in Figure 7.

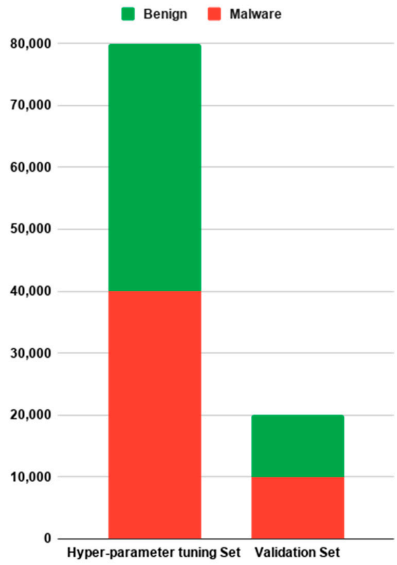

Figure 7. Sample distribution we used during the hyper-parameter tuning process.

Table 3 presents the optimal major hyper-parameters of the final model.

4.2.2. Model Integration into Sisyfos

The modular architecture of the platform then allows us to easily integrate our model into the system. The feature extraction logic that is not already included in our tools can be described in a single module that gets executed whenever a user submits a file. Additionally, the platform supports quick swapping between different machine learning

models, enabling us to easily load and use the trained model. This, combined with the fact that the platform saves every sample in a database, allows us to update the model often with new data, in order to have an up-to-date prediction system at all times.

Table 3. Optimal hyper-parameters of the model.

Hyper-Parameter	Value
Number of trees	1000
Minimum samples in leaf	1
Number of features per tree	Sqrt (m) [1]

[1] Where m is the number of all features.

4.2.3. Results

We measured the performance of the model through its accuracy at different false positive rates (FPR) and we compared it with the baseline model that came with EMBER.

The model was trained with 600,000 samples using all the 2351 features at about 36 h, on a single 8-core processor, and achieved 99% accuracy at 1% FPR and 98.8% accuracy at 0.1% FPR. Table 4 provides a comparison of our results with the EMBER baseline model, demonstrating that the model is capable of very accurately classifying both malware and benign executables at very low FPRs, performing better than the baseline model.

Table 4. Accuracy, precision and recall of the model for different false positive rates (FPRs) in comparison to the baseline model.

Recall		Precision		Accuracy		
Random Forest	**Ember**	**Random Forest**	**Ember**	**Random Forest**	**Ember**	**False Positive Rate**
99%	98%	99%	98.98%	99%	98.50%	1%
97.7%	92.9%	99.89%	99.89%	98.80%	96.40%	0.10%

Another important metric we used to evaluate the performance of our model was the receiver operating characteristic (ROC) curve. The ROC curve is a plot that illustrates the detection ability of a classifier at various false positive (FP) rates. To plot it, we measured the false positives while varying the discrimination threshold of the algorithm. Figure 8 plots the ROC curves of our model and the EMBER baseline model, demonstrating that our model clearly outperforms the EMBER baseline model in the FPR range of $[10^{-5}, 10^{-2}]$ with improvements reaching 3% in true positive rate.

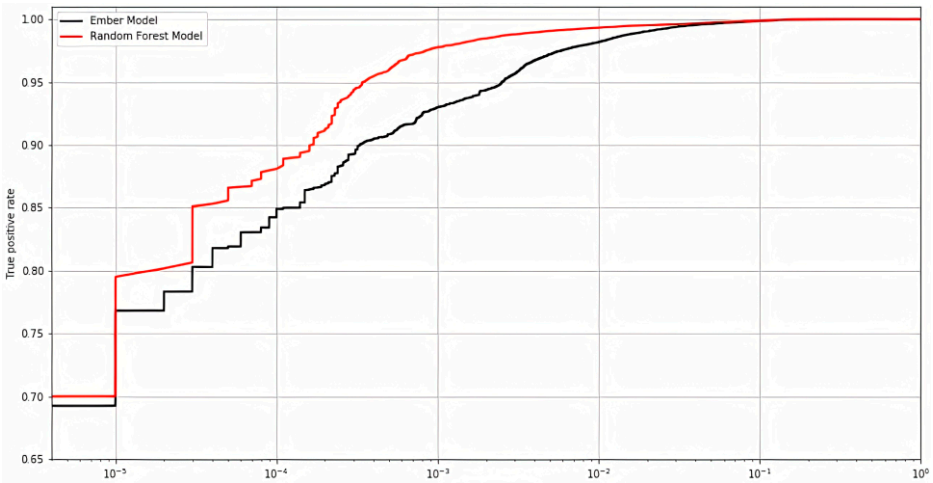

Figure 8. Receiver operating characteristic (ROC) curve of our model, in comparison with the EMBER baseline.

5. Conclusions and Future Work

Malware analysis platforms constitute fundamental infrastructure for the detection and mitigation of malware. Malware platforms not only classify and detect malware but enable the analysis of their features and enhance understanding of their structure, enabling efficient and effective detection. Machine learning techniques are quite promising in malware detection and classification but require significant sizes of reliable data for effective classifier training as well as for the selection of the appropriate machine learning techniques in variable operational environments.

Sisyfos constitutes a significant step in the development of open, modular and extensible malware platforms that support operational environments, including critical infrastructures, and enable the academic community to develop new tools and experiment with novel classifiers. However, significant effort needs to be spent in the specification, collection and public sharing of appropriate datasets, which will enable objective comparison of malware analysis methods and platforms and will lead to effective solutions.

In this direction, our future work includes exploring and improving classification methods by training and evaluating more machine learning algorithms with the adoption of larger datasets such as the newly released dataset, SoReL-20M [37]; this includes designing binary as well as multi-class classification methods in order to differentiate among different malware types and families. Furthermore, in addition to the next version of Sisyfos, which will include the ability to analyze Android files by integrating open-source tools such as MobSF [29], we will work toward improving Sisyfos' robustness and fault tolerance, especially in boundary cases where samples may lead to platform failures.

Author Contributions: All authors have contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: Part of the work that was performed at ISI/ATHENA was supported by the project "I3T—Innovative Application of Industrial Internet of Things (IIoT) in Smart Environments" (MIS 5002434) which is implemented under the "Action for the Strategic Development on the Research and Technological Sector", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund. Part of the work that was performed at ISI/ATHENA was supported by the European project "CONCORDIA".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Openly available dataset. Data was obtained from EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models and can be found here: https://github.com/elastic/ember (accessed on 20 February 2021).

Acknowledgments: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Choudhury, A. Top 8 Ransomware Attacks of 2020 that Shook the Internet. *Anal. India Mag.* Available online: https://analyticsindiamag.com/top-8-ransomware-attacks-of-2020-that-shook-the-internet/ (accessed on 20 February 2021).
2. Halpern, S. After the SolarWinds Hack, We Have No Idea What Cyber Dangers We Face. Available online: https://www.newyorker.com/news/daily-comment/after-the-solarwinds-hack-we-have-no-idea-what-cyber-dangers-we-face (accessed on 19 February 2021).
3. Morgan, S. Report: Cyberwarfare in the C-Suite. 2021. Available online: https://1c7fab3im83f5gqiow2qqs2k-wpengine.netdna-ssl.com/wp-content/uploads/2021/01/Cyberwarfare-2021-Report.pdf (accessed on 20 February 2021).
4. Malware Statistics & Trends Report | AV-TEST. Available online: /en/statistics/malware/ (accessed on 19 February 2021).
5. Shalaginov, A.; Banin, S.; Dehghantanha, A.; Franke, K. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. *arXiv* **2018**, arXiv:1808.01201.

6. Baldangombo, U.; Jambaljav, N.; Horng, S.-J. A Static Malware Detection System Using Data Mining Methods. *arXiv* **2013**, arXiv:1308.2831. [CrossRef]
7. Amin, M.; Tanveer, T.A.; Tehseen, M.; Khan, M.; Khan, F.A.; Anwar, S. Static Malware Detection and Attribution in Android Byte-Code through an End-to-End Deep System. *Future Gener. Comput. Syst.* **2020**, *102*, 112–126. [CrossRef]
8. Moser, A.; Kruegel, C.; Kirda, E. Limits of Static Analysis for Malware Detection. In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430.
9. Anderson, B.; Quist, D.; Neil, J.; Storlie, C.; Lane, T. Graph-Based Malware Detection Using Dynamic Analysis. *J. Comput. Virol.* **2011**, *7*, 247–258. [CrossRef]
10. Wong, M.Y.; Lie, D. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. In *Proceedings of the 2016 Network and Distributed System Security Symposium*; Internet Society: San Diego, CA, USA, 2016.
11. Sgandurra, D.; Muñoz-González, L.; Mohsen, R.; Lupu, E.C. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and Use for Detection. *arXiv* **2016**, arXiv:1609.03020.
12. Mohaisen, A.; Alrawi, O.; Mohaisen, K. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Comput. Secur.* **2015**, *52*. [CrossRef]
13. Park, Y.; Reeves, D.; Mulukutla, V.; Sundaravel, B. Fast Malware Classification by Automated Behavioral Graph Matching. In Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW '10), Oak Ridge, TN, USA, 21–23 April 2010; pp. 45:1–45:4. [CrossRef]
14. Shijo, P.V.; Salim, A. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]
15. Li, W.; Ge, J.; Dai, G. Detecting Malware for Android Platform: An SVM-Based Approach. In Proceedings of the 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 3–5 November 2015; pp. 464–469.
16. Zhao, M.; Ge, F.; Zhang, T.; Yuan, Z. AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android. In *Information Computing and Applications*; Liu, C., Chang, J., Yang, A., Eds.; Springer: Berlin, Germany, 2011; Volume 243, pp. 158–166. ISBN 9783642275029.
17. Zhu, H.-J.; .; Cheng, L. HEMD: A Highly Efficient Random Forest-Based Malware Detection Framework for Android. *Neural. Comput. Applic.* **2018**, *30*, 3353–3361. [CrossRef]
18. Garcia, F.C.C.; Muga, F.P., II. Random Forest for Malware Classification. *arXiv* **2016**, arXiv:1609.07770.
19. Yuan, Z.; Lu, Y.; Wang, Z.; Xue, Y. Droid-Sec: Deep Learning in Android Malware Detection. In Proceedings of the 2014 ACM conference on SIGCOMM, Chicago, IL, USA, 17 August 2014; pp. 371–372.
20. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Venkatraman, S. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access* **2019**, *7*, 46717–46738. [CrossRef]
21. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware Detection by Eating a Whole EXE. *arXiv* **2017**, arXiv:1710.09435.
22. Fleshman, W.; Raff, E.; Sylvester, J.; Forsyth, S.; McLean, M. Non-Negative Networks Against Adversarial Attacks. *arXiv* **2019**, arXiv:1806.06108.
23. VirusTotal. Available online: https://www.virustotal.com/gui/ (accessed on 22 February 2021).
24. Automated Malware Analysis—Joe Sandbox Cloud Basic. Available online: https://www.joesandbox.com/ (accessed on 22 February 2021).
25. Free Automated Malware Analysis Service—Powered by Falcon Sandbox. Available online: https://www.hybrid-analysis.com/ (accessed on 22 February 2021).
26. ANY.RUN—Interactive Online Malware Sandbox. Available online: https://any.run/ (accessed on 22 February 2021).
27. Malware Analysis Sandbox & Malware Detection Software. Available online: https://www.vmray.com/products/analyzer-malware-sandbox/ (accessed on 22 February 2021).
28. Cuckoo Sandbox—Automated Malware Analysis. Available online: https://cuckoosandbox.org/ (accessed on 20 February 2021).
29. OpenSecurity Mobile Security Framework (MobSF). Available online: https://github.com/MobSF/Mobile-Security-Framework-MobSF (accessed on 22 February 2021).
30. Lockheed Martin. Laika BOSS: Object Scanning System. Available online: https://github.com/lmco/laikaboss (accessed on 22 February 2021).
31. Hatching VMCloak. Available online: https://github.com/hatching/vmcloak (accessed on 22 February 2021).
32. Ortega, A. Paranoid Fish. Available online: https://github.com/a0rtega/pafish (accessed on 22 February 2021).
33. Goodin, D. Kaspersky: Yes, We Obtained NSA Secrets. No, We Didn't Help Steal Them. ArsTechnica, 16 November 2017. Available online: https://arstechnica.com/information-technology/2017/11/kaspersky-yes-we-obtained-nsa-secrets-no-we-didnt-help-steal-them/ (accessed on 15 March 2021).
34. Olson, R.S.; Bartley, N.; Urbanowicz, R.J.; Moore, J.H. Evaluation of a Tree-Based Pipeline Optimization Tool for Automating Data Science. In Proceedings of the Genetic and Evolutionary Computation Conference 2016, Denver, CO, USA, 20 July 2016; pp. 485–492.
35. Anderson, H.S.; Roth, P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv* **2018**, arXiv:1804.04637.

36. Hastie, T.; Tibshirani, R.; Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed.; Springer series in statistics; Springer: New York, NY, USA, 2009; ISBN 9780387848570.
37. Harang, R.; Rudd, E.M. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. *arXiv* **2020**, arXiv:2012.07634.