Lightweight Behavior-Based Malware Detection

Marco Anisetti$^{1[0000-0002-5438-9467]}$, Claudio A. Ardagna$^{1[0000-0001-7426-4795]}$, Nicola Bena$^{1[0000-0003-4909-9892]}$, Vincenzo Giandomenico$^{1[0009-0001-7729-0574]}$, and Gabriele Gianini$^{1[0000-0001-5186-0199]}$

Department of Computer Science, Universita degli Studi di Milano, Milan, Italy
`{marco.anisetti,claudio.ardagna,nicola.bena,gabriele.gianini}@unimi.it`
`vincenzo.giandomenico@studenti.unimi.it`

Abstract. Modern malware detection tools rely on special permissions to collect data able to reveal the presence of suspicious software within a machine. Typical data they collect for this task are the set of system calls, the content of network traffic, file system changes, and API calls. Giving access to these data to an externally created program, however, means granting the company that created that software complete control over the host machine. This is undesirable for many reasons. In this work, we propose an alternative approach for this task, which relies on easily accessible data – information about system performances (CPU, RAM, disk and network usage) – and does not need high-level permissions to be collected. To investigate the effectiveness of this approach, we collected these data in the form of a multi-valued time series and ran a number of malware programs in a suitably devised sandbox. Then – to address the fact that deep learning models need large training sets – we augmented the dataset using a deep learning generative model (a Generative Adversarial Network). Finally, we trained an LSTM (Long Short Term Memory) network to capture the malware behavioral patterns. Our investigation found that this approach, based on easy-to-collect information, is very effective (we achieved 0.99 accuracy), despite the fact that the data used for training the detector are substantially different from the ones specifically targeted to this purpose. The real and synthetic datasets as well as corresponding source code are publicly available.

Keywords: Malware detection · behaviour analysis · LSTM · GAN

1 Introduction

Malware, that is, malicious software, is nowadays one of the most common vectors of cyberattacks. It was traditionally considered a problem only for companies, however in 2017 the situation changed when the ransomware *Wanna-Cry* diffused and infected even hospitals and simple users. Nowadays, malware still represents the prime threat [9], with ransoms skyrocketing up to 50M$ and single malware infections costing up to 1M$ per incident [20]. Consequently, the fight between researchers – who try to implement new approaches for detecting malware in the fastest way possible – and malware developers – who create increasingly complex malware using evasive strategies to avoid detection – is going on a day-to-day basis and with alternating fates.

The approaches for malware detection typically fall into one of two typologies: static analysis and dynamic analysis. The former is focused on features that can be extracted from the malware code itself, without executing it. Usually, these features are the hash signature computed on the compiled file, the strings that can be found in such file, and the Assembly operations that can be extracted from the compiled file using a disassembler (e.g., [27,13,6]). Unfortunately, static analysis is gradually becoming less effective, because it can be easily sidestepped using techniques like encryption, encoding, and polymorphism. Dynamic analysis relies instead on the fact that the malware behavior cannot be changed easily. To perform dynamic analysis, the malware is usually executed in a safe environment – a sandbox, to prevent self-infection – while its behavior is observed and analyzed. The features that can be extracted are many, however the most common are the API calls (i.e., system calls) that the malware executes to interact and eventually control/damage the machine which is installed on (e.g., [31,17]). Nowadays, advanced anti-malware tools combine static and dynamic analysis with machine learning (ML) to improve detection, giving better results than a simple antivirus limited to signature analysis [7,14,23,21].

Malware behavior can typically be spotted by how it interacts with the environment. For instance, suspicious behavior includes attempts to modify system files or connections, calls to known malware functions or functions that are typically not used by legitimate software, and system information requests. The crucial drawback is that anti-malware tools need to obtain high-level permissions on the machine which they are installed on to detect such a behavior. Granting anti-malware tools permission to access this kind of information, however, is equivalent to providing complete control of the machine to the company which produces the anti-malware tool. Users might be reluctant to grant such a high level of privileges to third parties, even just for compliance with the company's internal policies. There were even examples of software with the ability to inspect a machine for good purposes that has been used as vector for malware.[1]

The goal of the present work is to address this issue, by developing an alternative approach for malware detection that relies on easily accessible behavioral data – so as not to require high-level permissions – fed to a deep learning model that learns to detect malware behavior. We considered that the information related to the system performances (CPU, RAM, disk and network usage) does not require high-level permissions to be collected: we set out to find whether this information would be sufficient to train a behavioral model able to distinguish between malware and legit software.

To carry on this investigation, we executed a plethora of well-known malware, and a number of commonly used legit software within a suitably designed sandbox and collected the performance data at system level (i.e., related to the

[1] https://www.ccleaner.com/knowledge/security-notification-ccleaner-v5336162-ccleaner-cloud-v1073191

overall system rather than to individual processes) under the form of a multi-valued time series. Then, to address the fact that deep learning models need large training sets, we augmented the dataset using a Generative Adversarial Network (GAN); finally, we trained an LSTM network to capture the malware patterns. Our investigation found that this approach based on easy-to-collect information is very effective (0.99% accuracy), despite the fact that the data used for training are very different from those typically used for this purpose. Furthermore, the real and synthetic datasets as well as corresponding code are publicly available at hhttps://doi.org/10.13130/RD_UNIMI/LJ6Z8V.

The remainder of the paper is structured as follows. Section 2 points to the state of the art; Section 3 outlines the methodology; Section 4 discusses the results, and Section 5 draws the conclusions.

2 Related Works

Machine learning and deep learning have gained ground in many disparate domains [1,29,4,2,3,24], including the area of malware detection.

Static analysis is based on data that can be extracted from malware/legit code such as Windows API calls (e.g., [12]) and Assembly instructions (e.g., [13]). These approaches display an excellent classification performance: using a variety of classifier algorithms (e.g., decision tree, random forest, AdaBoost, Gradient Boosting, SVM, kNN) they almost always achieve accuracy, precision, recall well above 0.9. However, as we observed, they are heavily invasive. The analysis of Windows PE (Portable Executable) constitutes a large part of research. For instance, Patri et al. [27] modeled PE files in terms of their entropy to be then classified using ML; Naz et al. [26] considered the headers of such files only for feature extraction. Ling et al. [16] focused on the robustness against adversarial attacks of ML-based PE malware detectors; Demetrio et al. [7] conducted a similar evaluation.

Dynamic analysis improves over the inability of static analysis of dealing with encryption, obfuscation, and polymorphism. Hybrid analysis further improves it by combining static and behavioral information. For instance, Miller et al. [22] combined static and dynamic features retrieved from the *VirusTotal* dataset. Dai et al. [6] combined two types of static features: API calls and low-level information retrieved from the hardware of the device, such as performance counters. Their detector consists of an ensemble of ML models.

Android malware received significant attention. For example, Li et al. [15] focused on static analysis, by extracting features from apps' files containing information such as the permissions required and the API calls. Feng et al. [10] proposed a similar approach while taking into account the limited resources of a typical Android device. Ma et al. [19] extracted three types of data related to API calls from the control flow of the app, and trained an ensemble of ML models thereon. Sihang at el. [31] executed the malware inside an Android emulator; application logs are stored, transformed into a features vector, and fed to a deep learning model. Hybrid approaches have also been proposed. For instance, Lu et al. [17] considered static information by de-compiling the apps' APK file and dynamic information by executing the apps in a safe environment.

Lightweight malware detection is based on dynamic analysis performed on *simple* features that are overlooked in traditional detectors, and is in its infancy. For instance, Milosevic et al. [23] collected system- and device-level information on Android devices mostly related to the memory. A logistic regression model trained on a reduced set of these features achieved ≈0.84 accuracy and recall. McDole et al. [21] considered two-dimensional samples: the first one represents individual processes while the second process-level features such as CPU usage. Multiple deep learning models achieved ≈0.93 accuracy and ≈0.9 recall in the best cases. Our approach considers system-level information instead.

Data scarcity is an important issue in the development of malware detectors. Companies that develop malware detectors do not publish their datasets as competitors could steal information, while attackers could study the dataset to make new malware that are not detected. This problem can be addressed using synthetic data. Among the ways to create synthetic data, we considered a GAN (Generative Adversarial Network) [11]. A GAN is a ML model composed of two networks that are trained against each other. One network (Generator) generates new data preserving the same distribution of real data and the other network (Discriminator) evaluates the synthetic data by computing the probability that the evaluated data is real or synthetic. Augmentation has already been evaluated in the context of malware detection. For example, Lu et al. [18] used a Deep Convolutional GAN to augment the *Malimg* dataset representing malware as images [25]. Malimg has been created by converting the malware executable code into 8-bit vectors then transformed into a gray-scale image. Interestingly, malware of the same family have a similar image representation. Wang et al. [33] proposed a similar approach to represent malware as black-white images. Another model that can be used to generate synthetic data is Variational Autoencoders. Burks et al. [5] used this approach to augment Malimg showing that the GAN-based approach of Lu et al. [18] yields better results.

Time series has already been used in some of the aforementioned works (e.g., [27,19,17]). A general discussion about time series classification can be found in [8].

LSTM is the model of choice in many time-series analyses. For instance, Čeponis et al. [32] compared two deep learning approaches for time-series malware classification, showing that the simpler approach gives equal or better results than the other one. Sayadi et al. [30] focused on malware hidden inside legit software, whose dataset is a set of time-series representing branch instructions gathered at run time.

In summary, the research community is putting a lot of attention on ML-based malware detection and lightweight detection promises the low overhead of static analysis with the quality of static analysis. Our approach puts forward this idea, considering a reduced set of 6 system level-features whose collection does not require any high-level permissions.

3 Methodology

We present on overview of our approach for lightweight behavior-based malware detection (Section 3.1) and detail the data collection process (Section 3.2), the creation of the dataset (Section 3.3), and the classification method (Section 3.4).

3.1 Our Approach at a Glance

Figure 1 shows an overview of our approach. Our sandbox implementation considers a Linux machine requesting the execution of malware/legit software in a Windows virtual machine, to collect the initial dataset in a safe environment. Such dataset was then fed to a GAN network, responsible for learning its peculiarities and generating similar synthetic data. Generated data were visually inspected together with real data in lower-dimensional spaces, through the use of *Principal Component Analysis* (PCA) and *t-Distributed Stochastic Neighbor Embedding* (t-SNE). Real and synthetic datasets were merged into the final dataset, split into training, validation, and test sets, to train and evaluate an LSTM model. Section 4 describes the results of this procedure. The real and synthetic datasets, the corresponding code, and a detailed description of the complete process are publicly available at https://doi.org/10.13130/RD_UNIMI/LJ6Z8V.

3.2 Sandbox Implementation

Running malware to analyze its behavior introduces is the risk of self-infection. The use of a sandbox can mitigate or remove this risk. A sandbox is an isolated environment where the malware can be safely executed. This approach is not always feasible, because some malware are capable of understanding whether they are running inside a sandbox. When this happens, some malware may change their behavior or interrupt their execution; some advanced malware are even capable of escaping the sandbox causing the infection of the system where the sandbox is installed. For this reason, we used a combination of Linux and Windows machines. Figure 2 shows their interaction. Specifically, we tested Windows malware and legit software on a Windows 7 virtual machine (VM) hosted by a Linux machine. The Windows VM is isolated from the Internet by a *host-only connection*. This way, the VM does not have access to the physical network card of the host machine, preventing any malware connections to the Internet.

Executing malware on a machine that cannot communicate on the Internet, however, has some limitations: some malware needs to connect to remote hosts to carry out their activities (e.g., Wanna-Cry). To allow the malware to still create connections without going to the Internet, we set up a second Linux VM. This VM runs the software *iNetSim*[2] to simulate Internet connections. With this

[2] https://www.inetsim.org/

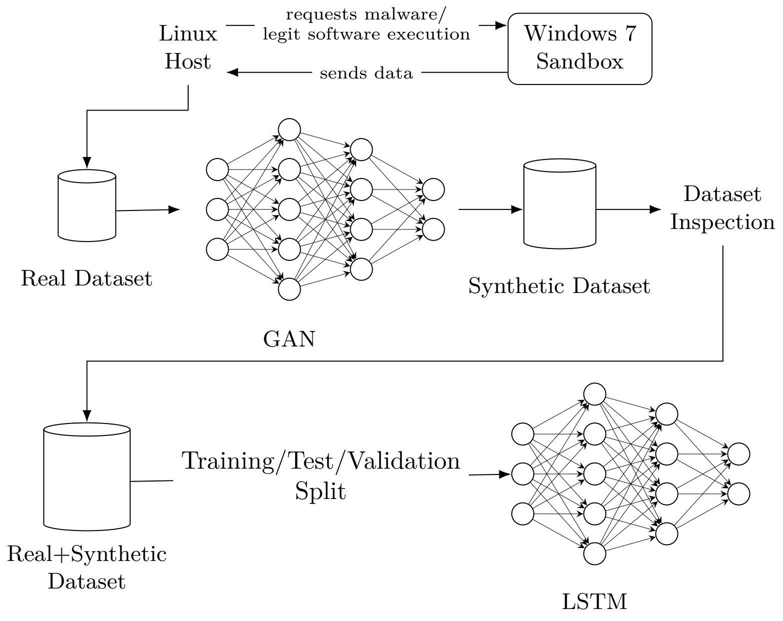

Fig. 1: Overview of our approach

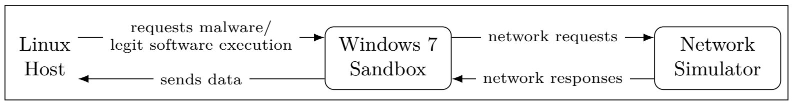

Fig. 2: The sandbox

configuration, the malware executed on the Windows VM can still make requests and obtain responses without resorting to outside connections.

To allow an effective execution of the malware inside the Windows VM, all protection controls like firewalls, Windows update, and Windows defender have been disabled and certain group policies have been changed to give the malware the capability to act as an administrator. The need to modify these policies motivates the use of an old Windows version, namely Windows 7.

3.3 Dataset

Malware and legit software. We retrieved real-world malware from *VirusShare*.[3] The website hosts nearly 55 million malware specimens, among which we considered ≈5,000 PE Windows files. For what concerns legit software, we installed commonly-used software on the Windows VM, to make the environment as re-

[3] https://virusshare.com

alistic as possible. Legit software includes Internet Explorer, Firefox, Mozilla Thunderbird, Spotify, WinRaR.

Dataset creation. We executed malware and legit software for a fixed amount of time while collecting performance metrics. The choice of this time span was critical. On the one hand, a short time span allows to immediately detect malware and hence preventing system infection. On the other hand, if the time span is too short, the amount of data will not be enough for the detection. To identify a suitable time span, we first generated several datasets, comparing the span and the accuracy of the subsequent classification phase. An acceptable trade-off was given by a span of 60 seconds.

We performed 10,000 executions varying between malware and legit software. At each execution, the Windows VM was restored from a clean snapshot (following the state of the art [23]) and the chosen software run for the given time span. During each execution, we collected the multi-valued time series consisting of 6 features: *i)* CPU usage percentage, *ii)* RAM usage percentage, *iii)* bytes written out and *iv)* bytes read from the disk, *v)* bytes received and *vi)* sent to the network. Collected data are sent back to the Linux host where they are saved.

The usage of the LSTM model requires all time-series to have the same length. For this purpose, we preprocessed collected data normalizing the time series to a fixed length by padding the shorter time series and pruning the longer ones. Each resulting time-series contains 10 items each associated with the 6 aforementioned features. Being the time span of 60 seconds, the sampling time was of 6 seconds. We note this time is slightly lower than similar approaches [21].

Dataset augmentation. Deep learning models requires a high number of training samples. Our dataset of ≈10,000 samples is not large enough, but real data collection is very expensive. Consequently, we opted for the generation of synthetic data, endowed with the same statistical properties as real-world data. To this purpose, we used one of the most effective methods currently available: GAN. More specifically, *TimeGAN* [34].[4] The code to instantiate the TimeGAN is as follows.

```
arg = ModelParameters(batch_size=128, lr=5e-4, noise_dim=32,
    layers_dim=128)
gan = TimeGAN(model_parameters=arg, hidden_dim=10, seq_len=10, n_seq=6,
    gamma=1)
```

We fed our normalized, real dataset to the GAN so that the model could learn its statistical characteristics and replicate them into the synthetic data. We first separated the real dataset into malware and legit software. We then fed each individual dataset to a separate instance of TimeGAN generating a synthetic dataset of 50,000 samples. We merged the two synthetic datasets and

[4] https://pypi.org/project/ydata-synthetic/

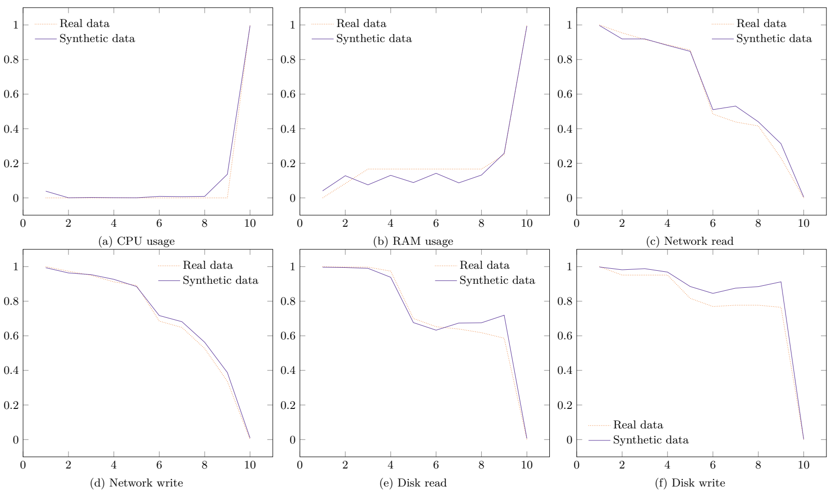

Fig. 3: Comparison of features value of real and synthetic malware samples.

obtained 100,000 samples in total (10-fold increase). The training of each GAN took approximately 5 hours. We then validate the quality of synthetic dataset according to several comparisons as follows.

– *visual feature comparison:* we randomly drawn samples from real and synthetic datasets. For each feature and extracted sample, we plotted their value to visually compare the differences between the real and the synthetic samples. Figures 3 and 4 shows the similarity of two random samples of malware and legit software, respectively.
– *comparison with reduced dimensionality (PCA):* we performed PCA reduction to a 2-dimensional space on real and synthetic datasets (limited to 500 samples), and plotted the results for visual comparisons. Figures 5(a)–(b) show that the synthetic data match real data.
– *comparison with reduced dimensionality (t-SNE):* we performed t-SNE reduction to a 2-dimensional space on real and synthetic datasets (limited to 500 samples). Compared to PCA, t-SNE performs a non-linear transformation. We plotted and visually compared the results. Figures 6(a)–(b) show that the synthetic data match real data.

We finally created the overall dataset by merging the real and the synthetic datasets.

3.4 LSTM Model

Table 1(a) describes the structure of the LSTM model we trained, composed of 4 layers (3 LSTM layers and 1 dense layer) interleaved with 3 batch normalization layers. Table 1(b) describes the parameters of the training process. We used

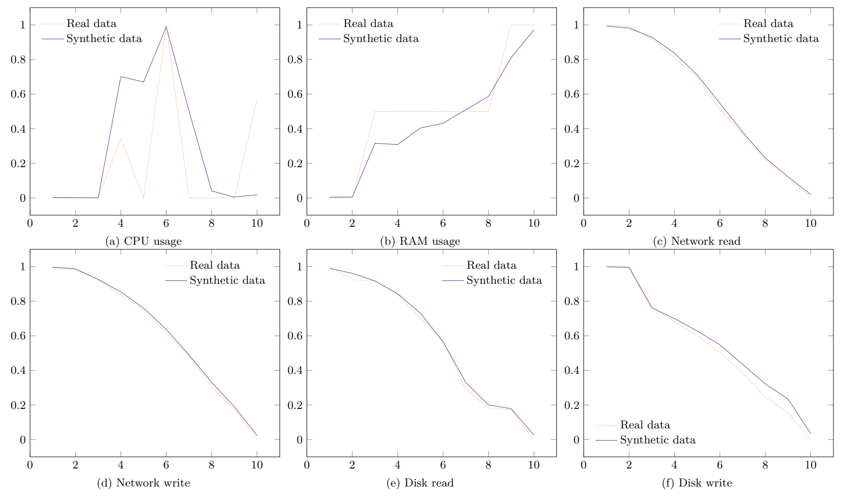

Fig. 4: Comparison of features value of real and synthetic legit software samples.

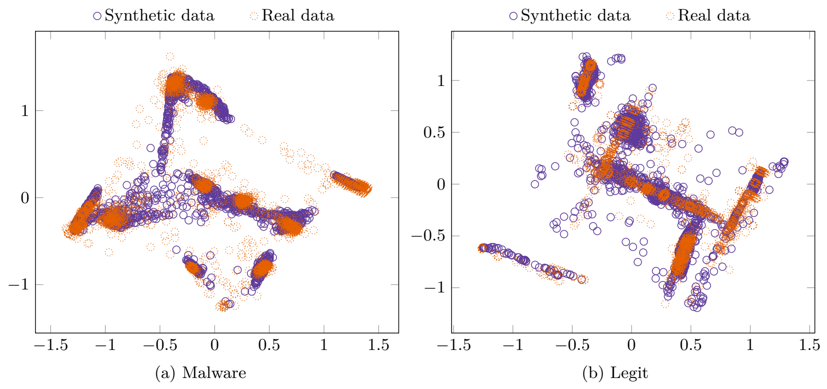

Fig. 5: Scatter plots of malware (a) and legit (b) real and synthetic data in a two-dimensional space according to PCA.

64,871 samples for the training set and 21,624 samples for the validation and test sets along 200 epochs with optimizer *Adam*, loss function *binary cross-entropy*, and initial learning rate of 0.05. In addition, training is based on early stopping (stop if loss function value retrieved from the validation set does not improve in 30 epochs), and on dynamic reduction of the initial learning rate (of a factor of 0.5 if loss function value retrieved from the validation set does not improve in one epoch). Further details can be found in our public code at https://doi.org/10.13130/RD_UNIMI/LJ6Z8V.

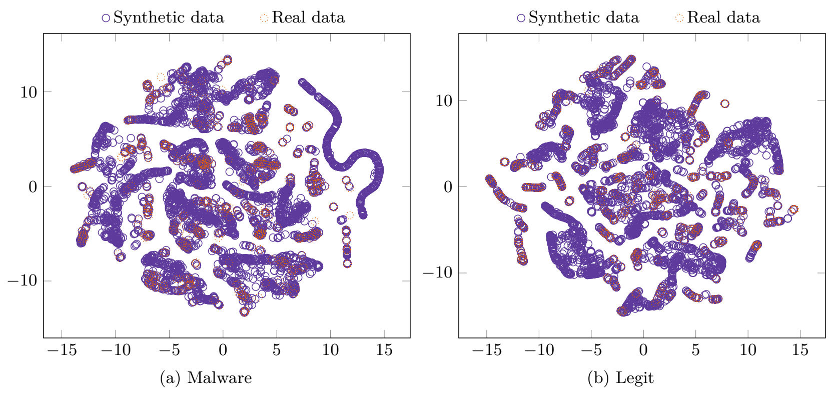

Fig. 6: Scatter plots of malware (a) and legit (b) real and synthetic data in a two-dimensional space according to t-SNE.

Layer Type	**Output Shape**	**# Params**
LSTM	(None, 10, 8)	480
Batch normalization	(None, 10, 8)	32
LSTM	(None, 10, 8)	544
Batch normalization	(None, 10, 8)	32
LSTM	(None, 8)	544
Batch normalization	(None, 8)	32
Dense	(None, 1)	9

(a) LSTM model structure

Parameter	**Value**
Epochs	200
Batch size	32
Optimizer	Adam
Learning rate	0.05, halved if loss does not improve in 1 epoch, down to $1{\cdot}10^{-8}$
Early stopping	Loss does not improve in 30 epochs
Loss function	Binary crossentropy

(b) Training parameters

Table 1: Details of LSTM training process.

4 Analysis Outcome

Starting from the collected real dataset, we executed our experiments on a VM equipped with 16 vCPU Intel Xeon CPUs E5-2620 v4 @ 2.10 GHz and 48 GBs of RAM. The VM features Ubuntu 22.02.4 x64, Python v3.10.6 and ML libraries *scikit-learn* v1.2.2 [28] and *Keras* v2.11.0.

4.1 Results

Training took ≈40 minutes, and completed in 40 epochs out of 200 due to early stopping (see Table 1(b)).

Tables 2(a)–(b) show the results retrieved from the test set. Table 2(a) shows our confusion matrix. Our approach achieves remarkable results, correctly identifying virtually all malware as well as legit samples. Table 2(b) shows other classification metrics. Our approach can distinguish an infected machine from an uninfected one with an accuracy of 0.99. The same value is achieved in any

	Predicted Positive	**Predicted Negative**
Actual Positive	TP = 10,869	FN = 29
Actual Negative	FP = 10	TN = 10,701

(a) Confusion matrix

Metric	**Definition**	**Value**
Precision	$\frac{TP}{TP+FP}$	0.9977
Recall	$\frac{TP}{TP+FN}$	0.9973
Specificity	$\frac{TN}{TN+FP}$	0.9976
F1-Score	$2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$	0.9975
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	0.8566
AUC	Area under ROC curve	0.9975

(b) Classification metrics

Table 2: Confusion matrix (a), where TP=True Positive count, FN=False Negative count, FP=False Positive count, TN=True Negative count; and classification metrics (b).

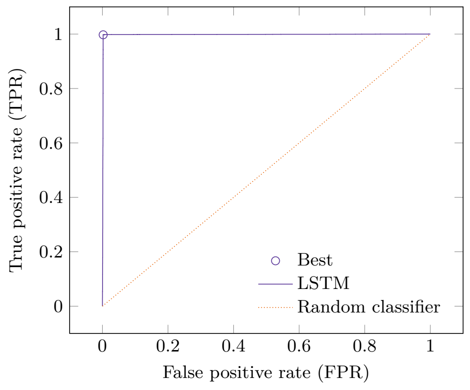

Fig. 7: ROC curve

other metrics, meaning that the detected malware is virtually always malware (precision) and the number of false negatives is negligible (recall). Figure 7 shows the ROC (Receiver Operating Characteristics) curve, which represents the true positive rate (TPR, retrieved as TPR=$\frac{\text{TP}}{\text{TP+FN}}$) vs. the false positive ate (FPR, retrieved as FPR=$\frac{\text{FN}}{\text{FP+TN}}$) varying the decision threshold of the LSTM classifier. The closeness to both axes once again proves the quality of our approach.

4.2 Discussion

The approach in this paper relies on information related to the system performances (i.e., CPU, RAM, disk and network usage) that does not require high-level permissions to be collected, and it represents an alternative to the heavily invasive approaches in the state of the art.

The achieved accuracy matches the accuracy retrieved in state-of-the-art hybrid approaches, clearly suggesting that, under the conditions of our settings, malware can be easily distinguished and hence, possibly, blocked in one minute at most considering system-level performances only. To the best of our knowledge, there exist few comparable works in literature (see Section 2). Milosevic et al. [23] considered a larger set of system-level features related to the global behavior of Android apps (e.g., total CPU usage) as individual samples rather than as time series. A logistic regression model achieves 0.86 accuracy at most. McDole et al. [21] considered virtually the same set of features of this work but at process-level rather than system-level again as individual samples rather than as time series. A convolutional neural network achieves ≈0.93 accuracy in the best case. Overall, our approach is far superior than its competitors, mainly due to the usage of a large synthetic dataset retrieved from a real dataset modeled using time series rather than individual, disconnected samples. The release of our complete dataset could pave the way for further tuning of ML models, on one side, and for retrieving additional insights from malware behavior characteristics.

5 Conclusions

Malware detection represents a urgent problem which is continuously being investigated by the research community. The approach in this paper sheds new light on the usage of data that can be collected with ease and can distinguish between legit and malware behavior. The paper leaves space for future work. First, we plan the extend the set of features to other system-level and easily-accessible information. Second, we plan to specifically focus on obfuscating malware. Third, we plan to strengthen the classifier from evasion attacks using dedicated techniques such as adversarial training.

Acknowledgments

The work was partially supported by the projects *i)* MUSA - Multilayered Urban Sustainability Action - project, funded by the European Union - NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5: Strengthening of research structures and creation of R&D “innovation ecosystems”, set up of “territorial leaders in R&D” (CUP G43C22001370007, Code ECS00000037); *ii)* SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU.

References

1. Alhashmi, N., Almoosa, N., Gianini, G.: Path Asymmetry Reconstruction via Deep Learning. In: Proc. of IEEE MELECON 2022. Palermo, Italy (June 2022)
2. Almazrouei, E., Gianini, G., Almoosa, N., Damiani, E.: What can Machine Learning do for Radio Spectrum Management? In: Proc. of ACM Q2SWinet 2020. Alicante Spain (November 2020)

3. Almazrouei, E., Gianini, G., Almoosa, N., Damiani, E.: Robust Computationally-Efficient Wireless Emitter Classification Using Autoencoders and Convolutional Neural Networks. Sensors **21**(7) (2021)
4. Almazrouei, E., Gianini, G., Mio, C., Almoosa, N., Damiani, E.: Using AutoEncoders for Radio Signal Denoising. In: Proc. of ACM Q2SWinet 2019. Miami Beach, FL, USA (November 2019)
5. Burks, R., Islam, K.A., Lu, Y., Li, J.: Data Augmentation with Generative Models for Improved Malware Detection: A Comparative Study. In: Proc. of IEEE UEMCON 2019. New York, NY, USA (October 2019)
6. Dai, Y., Li, H., Qian, Y., Yang, R., Zheng, M.: Smash: A malware detection method based on multi-feature ensemble learning. IEEE Access **7** (2019)
7. Demetrio, L., Coull, S.E., Biggio, B., Lagorio, G., Armando, A., Roli, F.: Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. ACM TPS **24**(4) (2021)
8. Dinger, T.R., chi Chang, Y., Pavuluri, R., Subramanian, S.: What is time series classification? https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/ (2022)
9. Eurpean Union Agency for Cybersecurity: ENISA Threat Landscape 2022. Tech. rep., Eurpean Union Agency for Cybersecurity (10 2022)
10. Feng, R., Chen, S., Xie, X., Ma, L., Meng, G., Liu, Y., Lin, S.W.: Mobidroid: A performance-sensitive malware detection system on mobile platform. In: Proc. of ICECCS 2019. Guangzhou, China (November 2019)
11. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks. Commun. ACM **63**(11) (2020)
12. Hardy, W., Chen, L., Hou, S., Ye, Y., Li, X.: DL4MD: A Deep Learning Framework for Intelligent Malware Detection. In: Proc. of DMIN 2016. Las Nevas, NV, USA (July 2016)
13. Kan, Z., Wang, H., Xu, G., Guo, Y., Chen, X.: Towards Light-Weight Deep Learning Based Malware Detection. In: Proc. of IEEE COMPSAC 2018. Tokyo, Japan (July 2018)
14. Li, D., Li, Q., Ye, Y.F., Xu, S.: Arms Race in Adversarial Malware Detection: A Survey. ACM CSUR **55**(1) (2021)
15. Li, D., Wang, Z., Xue, Y.: Fine-grained Android Malware Detection based on Deep Learning. In: Proc. of IEEE CNS 2018. Beijing, China (May–June 2018)
16. Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J., Wu, Y.: Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. Computers & Security **128**, 103134 (2023)
17. Lu, T., Du, Y., Ouyang, L., Chen, Q., Wang, X.: Android Malware Detection Based on a Hybrid Deep Learning Model. Security and Communication Networks **2020** (Aug 2020)
18. Lu, Y., Li, J.: Generative Adversarial Network for Improving Deep Learning Based Malware Classification. In: Proc. of WSC 2019. National Harbor, MD, USA (February 2019)
19. Ma, Z., Ge, H., Liu, Y., Zhao, M., Ma, J.: A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. IEEE Access **7** (2019)
20. Malwarebytes: 2023 state of malware. Tech. rep., Malwarebytes (2023)

21. McDole, A., Abdelsalam, M., Gupta, M., Mittal, S.: Analyzing CNN Based Behavioural Malware Detection Techniques on Cloud IaaS. In: Proc. of CLOUD 2020. Honolulu, HI, USA (September 2020)
22. Miller, B., Kantchelian, A., Tschantz, M.C., Afroz, S., Bachwani, R., Faizullabhoy, R., Huang, L., Shankar, V., Wu, T., Yiu, G., Joseph, A.D., Tygar, J.D.: Reviewer Integration and Performance Measurement for Malware Detection. In: Proc. of DIMVA 2016. San Sebastián, Spain (July 2016)
23. Milosevic, J., Malek, M., Ferrante, A., Malek, M.: A friend or a foe? detecting malware using memory and cpu features. In: Proc. of SECRYPT 2016. Lisbon, Portugal (July 2016)
24. Mio, C., Gianini, G.: Signal reconstruction by means of Embedding, Clustering and AutoEncoder Ensembles. In: Proc. of IEEE ISCC 2019. Barcelona, Spain (June–July 2019)
25. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware Images: Visualization and Automatic Classification. In: Proc. of VizSec 2011. Pittsburgh, PA, USA (2011)
26. Naz, S., Singh, D.K.: Review of Machine Learning Methods for Windows Malware Detection. In: Proc. of ICCCNT 2019. Kanpur, India (July 2019)
27. Patri, O., Wojnowicz, M., Wolff, M.: Discovering Malware with Time Series Shapelets. In: Proc. of HICSS 2017. Waikoloa, HI, USA (January 2017)
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12** (2011)
29. Ramos, I.F.F., Gianini, G., Damiani, E.: Neuro-Symbolic AI for Sensor-based Human Performance Prediction: System Archi-tectures and Applications. In: Proc. of ESREL 2022. Dublin, Ireland (August–September 2022)
30. Sayadi, H., Gao, Y., Mohammadi Makrani, H., Lin, J., Costa, P.C., Rafatirad, S., Homayoun, H.: Towards Accurate Run-Time Hardware-Assisted Stealthy Malware Detection: A Lightweight, yet Effective Time Series CNN-Based Approach. Cryptography **5**(4) (2021)
31. Sihag, V., Vardhan, M., Singh, P., Choudhary, G., Son, S.: De-LADY: Deep learning based Android malware detection using Dynamic features. JISIS **11** (May 2021)
32. Čeponis, D., Goranin, N.: Investigation of Dual-Flow Deep Learning Models LSTM-FCN and GRU-FCN Efficiency against Single-Flow CNN Models for the Host-Based Intrusion and Malware Detection Task on Univariate Times Series Data. Applied Sciences **10**(7) (2020)
33. Wang, F., Al Hamadi, H., Damiani, E.: A Visualized Malware Detection Framework with CNN and Conditional GAN. In: Proc. of IEEE Big Data 2022. Osaka, Japan (December 2022)
34. Yoon, J., Jarrett, D., van der Schaar, M.: Time-series Generative Adversarial Networks. In: Proc. of NeurIPS 2019. Vancouver, Canada (December 2019)