Exposing the Cracks: A Case Study on the Quality of Public Linux Malware Data Sets

Alessandro Sanna[1,*], Daniele Canavese[2], Leonardo Regano[1], Davide Maiorca[1] and Giorgio Giacinto[1,3]

[1]Department of Electrical and Electronic Engineering (DIEE), Università degli Studi di Cagliari
[2]Institut de Recherche en Informatique de Toulouse (IRIT)
[3]Consorzio Interuniversitario Nazionale per l'Informatica (CINI)

Abstract

Machine learning is extensively used for malware detection due to its accuracy, scalability, and adaptability. However, the effectiveness of ML models heavily depends on the quality of the datasets used for training and testing. This study evaluates popular public datasets for malware ARM ELF binaries from MalwareBazaar and VirusShare, complemented with benign binaries from Debian repositories. Using mnemonic frequency analysis, we found that these datasets lack the diversity found in Android or Windows. Using only the frequency of a single assembly mnemonic, we can distinguish the malware from the goodware with a balanced accuracy of 78%, and using three mnemonics, we achieved a balanced accuracy of 99%. We finally derive conclusions on the current state of Linux *publicly available* malware.

Keywords

Malware Analysis, Data Set Quality, Binary Analysis, Linux Malware, Machine Learning, Assembly Mnemonics

1. Introduction

Machine Learning (ML) is frequently used for malware detection [1, 2] because it can achieve high accuracy, scale to handle large volumes of data, continuously improve its precision, and potentially adapt to new threats. Many flavours of ML models and techniques have been applied, from simple logistic regression to advanced neural networks.

Independent of the ML model and methodology chosen to identify the malware (and the goodware), however, the performances of a machine learning system are strongly correlated with the quality of the dataset used in the training phase. Without a good data set for training and testing an ML model, the obtained results will be biased. Data set quality refers to various properties, such as being error-free and having enough observations with enough diversity to represent the real world.

We decided to analyze some popular public data sets for malware ARM ELF (Linux) binaries offered by MalwareBazaar and VirusShare. These websites offer freely downloadable labelled binaries that can be further analysed and provide an open way to train a batch of ML malware classifiers without a costly subscription. The main purpose of these data sets is to publish malicious binaries, so we complemented them with some realistic benign binaries from the Debian repositories.

A common approach to analyzing these malicious binaries is disassembling them and performing some assembly analysis. In our study, we decided to opt for one of the simplest approaches: counting the number of instructions, split by their mnemonic (e.g., `add`, `bx`). Similar approaches have already been proposed in the literature, such as analyzing mnemonics and opcodes with neural networks, support vector machines, decision trees, and a variety of other machine learning models [3, 4].

To determine the diversity of these data sets, we analyzed the collected mnemonic frequencies and tested our findings by training a batch of classifiers. Although our results are described in detail in

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03–08, 2025, Bologna, IT
alessandro.sanna96@unica.it (A. Sanna); daniele.canavese@irit.fr (D. Canavese); leonardo.regano@unica.it (L. Regano); davide.maiorca@unica.it (D. Maiorca); giorgio.giacinto@unica.it (G. Giacinto)
0000-0002-0610-7736 (A. Sanna); 0000-0002-4265-7743 (D. Canavese); 0000-0002-9259-5157 (L. Regano); 0000-0003-2640-4663 (D. Maiorca); 0000-0002-5759-3017 (G. Giacinto)

Section 5, we anticipate that we found a lack of diversity in these public data sets.

This may be because of two reasons:

1. The datasets do not represent the real-world scenario and overrepresent a specific kind of malware.
2. Currently, the state of Linux malware is such that the lack of diversity is rooted in the real world (i.e., malicious actors focus on certain malware strands).

To increase the reproducibility of our experiments and the transparency of our approach, we have published all the source code and the data sets online[1].

This paper is structured as follows. Section 2 contains some related works. Section 3 introduces our approach and describes what binaries we picked and how we analyzed them. Sections 4 and 5 contain our findings, respectively, on the data set and its features; in the last section, we also show the classification metrics of some very simple yet highly accurate machine-learning models trained on this data. Finally, Section 6 closes this paper by presenting our conclusions and plans for the future.

2. Related works

Opcodes and Mnemonics are explored approaches in the State of the Art. Although the Linux landscape suffers from a bit of underdevelopment concerning its mobile and desktop equivalent leading solution, i.e., Android and Windows, respectively, quite a bit of work can be found in the State of the Art regarding malware targeting this solution. Specifically, Ngo et al. [5] compiled a survey of other works tackling the malware analysis problem on Linux. We take this as an initial step to compare our approach with opcode-based approaches in SotA. Specifically, to make a fair comparison, we compare only with the Opcode-based approaches and forgo more complex takes like Graph-based features. This is because we focus not on feature engineering but on data set quality. That is, if data set impropriety can be shown with an encoding as linear as opcode count, we claim that this property will transfer no matter the encoding, as it is rooted in data. We report the dimensions of data sets employed in previous work in Table 1.

Cozzi et al. [6] made a seminal work about Linux malware itself, conducting a large-scale empirical study to assess the complexity of existing Linux malware, how state-of-the-art malware analysis techniques and tools adapt to the characteristics of Linux malware, and providing an overview on ad-hoc evasion techniques employed by such malware. They studied a dataset comprising 10548 malicious samples targeting various architectures, including X86-64, MIPS, and PowerPC. They concluded that, at the time of writing (2018), Linux malware complexity was still not on par with the Windows counterparts. Still, malware authors were starting to evolve simple malware by adapting evasion techniques employed by Windows malware.

More recently, Carrillo-Mondéjar et al. [7] performed a study to characterize Linux malware specifically tailored to target IoT devices. Their approach combines static metrics (Cyclomatic Complexity, number of basic blocks), dynamic features (including unique syscalls and number of created processes), and an n-grams representation of mnemonics. They employ such features to train various classifiers (Random Forest, K-nearest, Decision Tree, Support Vector Machines, and Linear kernels) on a dataset of more than 10000 malicious samples, able to identify the family to which the analyzed malware belongs. Notably, unlike our study, they do not classify malware against benign software.

Phu et al. [8] propose CFDVex, a feature selection method for cross-architecture malware detection. Utilizing the Vex intermediate representation and the CFD method developed in their previous work [9], they extract opcode-based features from a dataset of Intel 80836 IoT malware. Notably, they found that their method could precisely detect MIPS malware (95.72% accuracy; 2.81% FPR), although the system was never exposed to it. Unlike our work, however, they consider a benign population mainly focused on router devices, diminishing the variety of behaviors considered.

Azmoodeh et al. [10] consider an opcode-based representation that relies on a compressed form of a Control Flow Graph in matrix form, from which they extract the eigenvalues and eigenvectors to select

[1] https://github.com/alessandro-sanna/Linux_Mnemonic_Analysis

features for a classifier. They find that this approach obtains good accuracy, although the performance degrades with a growing percentage of inserted junk code. Notably, they consider a limited dataset composed of less than 1500 samples, only 128 of which compose the malware class.

Furthermore, Dovom et al. [11] expand on the latter work considering the potential of Fuzzy Pattern Trees, defined as "tree-like structure in which the inner nodes are fuzzy logic arithmetic operators and the leaf nodes are associated with fuzzy predicates on input attribute," for malware detection. They extract the opcodes from their sample set and select the most beneficial features using a Class-wise Information Gain criterion [12]. Finally, they adopt the Eigenspace approach seen in the latter work by Azmoodeh et al. and consider the eigenvectors of the filtered Control Flow Graph. This work considers the most extended dataset in the explored State of the Art. However, they rely mainly on Vx-Heaven, an unmaintained source no longer available. Furthermore, their sources primarily comprise Windows malware, which behaves differently from Linux malware. For example, the behaviour variability in the Windows landscape is far greater, whereas most Linux malware is composed of botnet strains.

Darabian et al. [4] trained various machine learning models (e.g., k nearest neighbours, support vector machines, neural networks). Still, instead of using the single mnemonic frequency as inputs (as in our work), they computed the frequency of the most common sequences of opcodes. They computed these sequences using MG-FSM (Mind the Gap: Frequent Sequence Mining), an efficient algorithm to compute the most common sub-sequences in a data set. They obtained high accuracy for all their models, ranging from 97% to 99%. Although tested on a different data set, it is interesting that our approach can obtain similar results but with much simpler models (see Section 5).

Haddadpajouh et al. [3] tested the performance of recurrent neural networks on a relatively small data set of ARM IoT malware binaries consisting of 281 malicious applications and 270 benign ones. They disassembled the ARM binaries, extracted the list of mnemonics, and removed all the consecutive occurrences of the same mnemonic to reduce the length of these sequences. They then performed a word embedding to transform each mnemonic into a vector of numbers and feed a list of embeddings (e.g., an encoded list of mnemonics) to several recurrent neural networks. The authors tested unidirectional and bidirectional LSTM (Long-Short Term Memory) neural networks. They experimentally found that the optimal results were with a two-layer LSTM with only 192 neurons, achieving an accuracy of about 98%.

Table 1
Comparison between the dataset used in the State of the Art.

PAPER	# OF MALWARE	# OF GOODWARE
[7][2]	10548	-
[8]	19710	4107
[10]	128	1078
[11][3]	22000+	22000+
[4]	247	269
[3]	280	271
our work	30304	65112

3. Our Approach

Our approach considers an intuitive metric forcibly present in all valid ELF samples: Assembly Mnemonics. These are defined as strings that "tell the assembler which machine instructions to assemble."[4] We

[2]This work is focused on family classification, so the benign class is absent.
[3]The distribution of benign and malicious samples is unclear, as the authors do not clarify this distinction and their main source, Vx-Heaven, is not available anymore
[4]https://onlinedocs.microchip.com/oxy/GUID-91118B1F-52AE-4822-A75E-B65FA4C48896-en-US-1/GUID-6E3B15DB-5560-44A3-93F1-91CA7F254F0A.html

can consider them a string equivalent of the byte representation of "opcodes." We avail of a Python 3 environment to interact with Ghidra 11[5]. We then execute a custom Ghidra script on each sample in order to extract the disassembled code from each function. From this, we extract the first word of each line of code: this is the Mnemonic Equivalent of the assembly operation pertaining to that Assembly line.

```
mov fp, 0
mov lr, 0
pop {r1}
mov r2, sp
str r2, [sp, -4]!
str r0, [sp, -4]!
ldr ip, sym._fini
str ip, [sp, -4]!
ldr r0, main
ldr r3, sym._init
b sym.__uClibc_main
```

mov	3
pop	1
str	3
ldr	3
b	1

Figure 1: Conversion from ARM assembly code to feature vector.

Having extracted this accumulation count for each sample, we then construct a single sparse matrix containing a column for each instruction present in each of the considered samples, be them malicious or benign. Then, each row will represent the count for the n-th sample of each found instruction. A visualisation of this matrix, constituting our final feature set, can be seen in Table 2.

Table 2
Section of the final feature matrix.

MD5 HASH[6]	[...]	RSBNE	ANDEQ	ORREQ	LDRH	BGT	LDRD	[...]
79701d6879119a7f5343b20e2f987ffb		1	1	1	0	0	0	
aff684c099331c2e1f1182a789fdae3a		0	1	0	4	1	1	
1d08419b46abd2a206086a735a00fcb1		0	0	0	3	13	16	
5f1a355f78a50798d762767db3253565	[...]	1	14	2	0	1	0	[...]
04618c7a170179adcc09287fe402943d		7	9	8	0	56	0	
ed79c233137e3a87b3eb6c0e9945555c		7	9	7	0	55	0	
fdd7b7ccb367c660fb8766a4b0cb18ba		13	3	10	63	35	0	
640af23dbc274ab0d62add53ca1871bd		4	1	8	55	102	0	

4. Data Set Analysis

4.1. Malware Section

Our dataset of choice has been obtained by integrating ELF files available in both MalwareBazaar[7] and VirusShare[8].

All of our samples were detected from at least one of the 75 antivirus solutions of VirusTotal [9].

We collect informative JSONs for each sample using the VirusTotal APIs. For the scope of this study, we use Linux's `readelf` to obtain the architecture information and further restrict the dataset to all the

[5] https://ghidra-sre.org
[6] We report the MD5 hash instead of the SHA-256 one because of visualisation purposes.
[7] https://bazaar.abuse.ch
[8] https://virusshare.com
[9] https://www.virustotal.com

`32-bit ARM` samples. This leads us to a total set composed of 30304 malicious samples. The dataset composition is shown in Table 3.

Table 3
Distribution of the malware set over the two sources considered.

ARCHIVE NAME	ARM SAMPLES
`MalwareBazaar`	13723
`VirusShare`	16581
total	30304

4.1.1. Temporal Distribution

In order to account for temporal shift in our dataset, we collect the year of submission of each sample. The set contains samples in a 12 year span, from 2010 to 2022. This wide range allows us to better evaluate malicious code's evolution over time, and it is needed to conduct a realistic analysis of the system's performance [13].

Figure 2 illustrates the temporal distribution of the malicious set on a yearly basis.

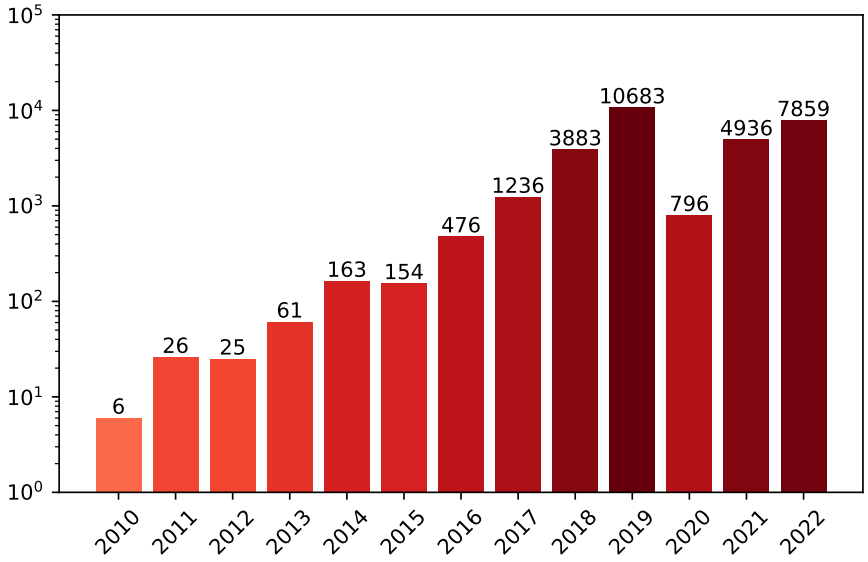


Figure 2: Temporal distribution of the malicious set.

4.1.2. Family Distribution

We use AVClass2 [14] to compute the malware family of each sample and provide more detail on the malware we consider. Table 4 shows the top 10 families in our set. We do not report information about all the 91 in our set for brevity. Briefly, the greatest portion of our set is made of botnets. The most notable, *Mirai* [15] and *Gafgyt* [16], are two related botnets used in Distributed Denial of Service attacks around the world. Predictably, the overwhelming majority is related to these families. This reflects recent trends [17, 18] that highlight how Mirai is one of the most widespread malware in the evergrowing IoT landscape.

Table 4
Family distribution of the malicious data.

FAMILY	INFO	THREAT	SAMPLES
`mirai`	[15]	Botnet	13901
`gafgyt`	[16]	Botnet	4475
`tsunami`	[19]	Botnet	172
`dofloo`	[20]	Botnet	72
`ngioweb`	[21]	Botnet	35
`pnscan`	[22]	Trojan	35
`kaiji`	[23]	Trojan	23
`mobidash`	[24]	Adware	20
`hajime`	[25]	Botnet	20
`ddostf`	[26]	Trojan	20
singleton	-	-	11261

4.2. Goodware Section

We generate our goodware dataset as follows: we select all available indexed packages from Debian 12 (Bookworm) [10], from both the `armhf` and the `armel` distributions. Then, we decompress the obtained DEB packages via `dpkg`. Finally, we check the magic number of the obtained files and restrict to only the ELF files (magic number: `7f 45 4c 46`).

4.2.1. Temporal Distribution

Our set contains samples ranging from 2010 to 2023. In order to make a fair comparison with the malicious set, we also restrict this set up until 2022 goodware, included. This results in a set of 65112 benign samples. Figure 3 illustrates the temporal distribution of the benign set on a yearly basis.

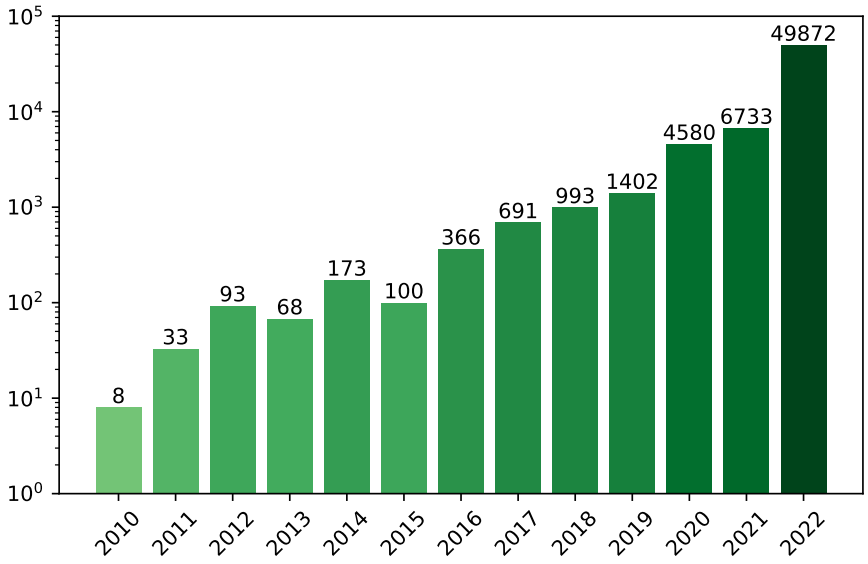


Figure 3: Temporal distribution of the benign set.

[10] https://packages.debian.org/bookworm/allpackages

5. Experimental Results

In this section, we report our findings about the data set we built and some metrics on various machine-learning models we trained.

5.1. Mnemonic analysis

The data set presented in this paper (see Section 4) contains various benign and malicious samples from widely used public repositories. When we started to analyse it, we noticed that the malware samples presented substantial differences when compared with the benign ones. Table 5 shows some statistics relative to the top 10 features (e.g., assembly mnemonics) with the highest separability index and 10 with the lowest.

In this context, we define as the *separability index* of a feature f a value stating how easy it is to split the observations into benign and malicious samples by performing a univariate analysis of f. In our case, the separability index is a percentage, and the higher the value, the higher the ability of a certain mnemonic to identify the malware and the goodware. To compute this value, we used k-means (with $k = 2$) to split the samples into two clusters using a single feature, and then we computed the accuracy of such unsupervised classification with the ground truth.

In addition to the separability indexes, Table 5 also shows the average value of the various mnemonic counts and their standard deviations.

From Table 5a we can see a peculiar trend. All these instructions tend to be used only in the malicious binaries (i.e., the means are not zero), while the goodware eschews them (i.e., the means are close to zero). By only looking at the presence of these instructions, we can immediately have a good indicator of a suspicious binary.

On the other hand, Table 5b shows that the mnemonics with the lowest separability index have similar means (close to zero), making them poorly suited as malware indicators. These behaviors are consistent with our definition of the separability index.

Table 5
Statistics of the mnemonic features.

CODE	MALICIOUS		BENIGN		SEP.
	MEAN	STD	MEAN	STD	
`mlaeq`	1.06	0.99	0.06	1.42	84.34
`cmpcc`	3.44	3.44	0.09	2.35	83.90
`swi`	34.54	38.40	0.04	2.68	82.70
`movnes`	1.12	0.96	0.13	0.55	82.25
`ldmdbge`	1.56	1.50	0.00	0.03	82.13
`stmiage`	2.38	2.30	0.00	0.16	82.12
`stmdbge`	1.56	1.50	0.00	0.09	82.12
`ldrbge`	2.09	2.00	0.03	1.94	82.05
`strbgt`	2.72	2.79	0.07	0.95	82.03
`strbge`	2.10	2.03	0.04	0.92	81.97

(a) The most separable mnemonic codes.

CODE	MALICIOUS		BENIGN		SEP.
	MEAN	STD	MEAN	STD	
`cmp.le.w`	0.00	0.11	0.02	0.25	62.30
`pop.ge`	0.00	0.04	0.02	0.15	61.37
`lsrs.eq.w`	0.00	0.07	0.02	0.15	61.37
`rrxs`	0.00	0.03	0.02	0.15	61.37
`orr.le.w`	0.00	0.04	0.03	0.22	61.31
`cmp.cs.w`	0.00	0.02	0.03	0.20	61.28
`bicnes`	0.01	0.16	0.08	0.36	61.19
`tsteq`	0.03	0.52	0.16	0.73	61.19
`rsbcss`	0.02	0.14	0.05	0.22	60.53
`eorgt`	0.03	0.35	0.14	0.63	60.29

(b) The least separable mnemonic codes.

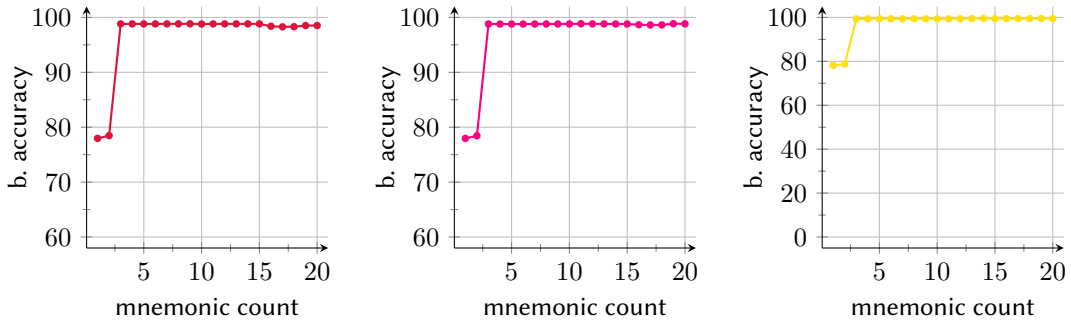
5.2. Classification results

We aggregate the 65112 and the 30304 feature vectors obtained via the featurization process. Therefore, we get a feature matrix of 95416 lines. We then pair each sample with its timing information with day-level granularity and sort the matrix rows by date. Finally, we split it using the older 75% of the data set to train various machine learning models and the remaining more recent 25% for testing purposes. We purposefully avoid random-split cross-validation to avoid introducing temporal biases in our evaluation [27].

Figures 4 and 5 show the balanced accuracy we obtained by testing various machine learning models. In particular, we decided to train three categories of models: logistic regression models, Support Vector Machines (SVMs) with a linear kernel, and some decision trees. We deliberately picked these three model categories since they are among the simplest ones in order to highlight the clear separation between the goodware and malware classes. We then implemented our experiments using the popular `scikit-learn` Python package. We used the default hyperparameters for every model, thus avoiding any hyperparameter optimization. The full source code can be found in our repository.

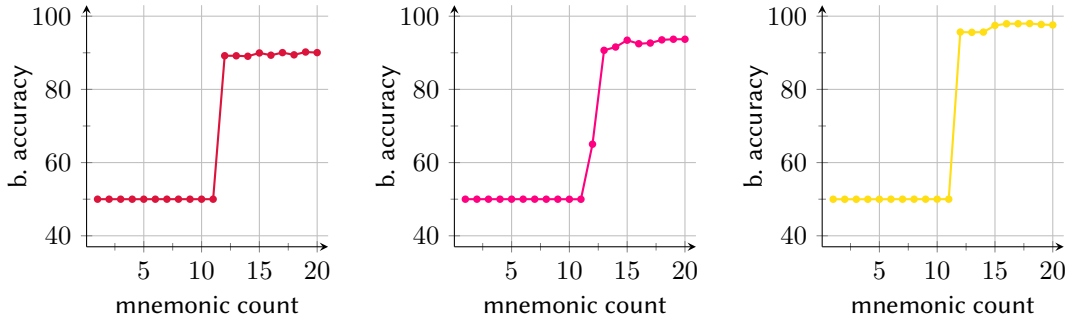
Figure 4 shows the results of these model categories when training them with the first n features (mnemonic counts) with the highest separability indexes. From the plots, we can see that by using only the `mlaeq` counts, all the models have a balanced accuracy of about 78%. After using only the first three of them (`mlaeq`, `cmpcc`, and `swi`), they reach a score of about 98%.

We then repeated a similar experiment by picking the features with the lowest separability indexes. Figure 5 shows our findings. The bottom 11 mnemonics (see also Table 5b) are inconclusive, and all the models show a balanced accuracy of 50%. This is the worst-case scenario since the classifiers are unsure if a binary is malware or goodware. However, if we add more features, the accuracy increases. By picking the bottom 20 features, all the models become very precise. In particular, they achieve a balanced accuracy of about 90%, 93%, and 97%, respectively, for the logistic regression, SVM, and decision tree.



(a) Logistic regression. (b) Linear SVM. (c) Decision tree.

Figure 4: Balanced accuracies from the most separable mnemonics.



(a) Logistic regression. (b) Linear SVM. (c) Decision tree.

Figure 5: Balanced accuracies from the least separable mnemonics.

6. Conclusions

In this study, we demonstrate the low variability of existing free datasets, which severely hinders research on detection methodologies for malware. Considering the increasing spread of malware

targeting Linux machines, coupled with the widespread adoption of IoT solutions based on this OS, we believe that the community should undertake a serious effort toward the creation of more extensive free datasets to bolster the research on malware analysis and detection methodologies in the Linux landscape.

Notably, one could consider that focusing on ARM architectures is an ill-posed effort because the process can be trivialised. However, we point out that ARM is the most used architecture in the IoT domain [28], and the market share of this domain is projected to grow exponentially in the future[11]. Given this, we argue that a proactive effort to strengthen the current classification paradigms is necessary.

Considering the results presented in this paper, we plan to extend our study to assess the quality of Linux malware datasets for other hardware platforms, e.g., Intel and MIPS. We also plan to increase the granularity of our research to precisely evaluate the effect of the variability of datasets when targeting the detection of specific malware families. We also plan to establish a precise methodology to assess the quality and variability of datasets to provide a standardized benchmarking procedure to test ML/AI-based malware detection approaches, reducing the bias introduced by the chosen datasets for training and testing.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) and by project SETA (PNRR M4.C2.1.1 PRIN 2022 PNRR, Cod. P202233M9Z, CUP F53D23009120001, Avviso D.D 1409 14.09.2022). Both these projects are under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This work was also partially supported by the European research projects H2020 LeADS (GA 956562), Horizon Europe DUCA (GA 101086308), ARN TrustInClouds, and CNRS IRN EU-CHECK.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] P. Maniriho, A. N. Mahmood, M. J. M. Chowdhury, A survey of recent advances in deep learning models for detecting malware in desktop and mobile platforms 56 (2024). URL: https://doi.org/10.1145/3638240. doi:10.1145/3638240.

[2] J. Singh, J. Singh, A survey on machine learning-based malware detection in executable files, Journal of Systems Architecture 112 (2021). doi:10.1016/j.sysarc.2020.101861.

[3] H. HaddadPajouh, A. Dehghantanha, R. Khayami, K.-K. R. Choo, A deep recurrent neural network based approach for internet of things malware threat hunting, Future Generation Computer Systems 85 (2018) 88–96. URL: https://www.sciencedirect.com/science/article/pii/S0167739X1732486X. doi:https://doi.org/10.1016/j.future.2018.03.007.

[4] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, K.-K. R. Choo, An opcode-based technique for polymorphic internet of things malware detection, Concurrency and Computation: Practice and Experience 32 (2020) e5173.

[5] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, D.-H. Nguyen, A survey of iot malware and detection methods based on static features, ICT Express 6 (2020) 280–286. URL: https://www.sciencedirect.com/science/article/pii/S2405959520300503. doi:https://doi.org/10.1016/j.icte.2020.04.005.

[6] E. Cozzi, M. Graziano, Y. Fratantonio, D. Balzarotti, Understanding linux malware, in: 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 161–175. doi:10.1109/SP.2018.00054.

[11] https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307

[7] J. Carrillo-Mondéjar, J. Martínez, G. Suarez-Tangil, Characterizing linux-based malware: Findings and recent trends, Future Generation Computer Systems 110 (2020) 267–281. doi:https://doi.org/10.1016/j.future.2020.04.031.

[8] T. N. Phu, L. H. Hoang, N. N. Toan, N. D. Tho, N. N. Binh, Cfdvex: A novel feature extraction method for detecting cross-architecture iot malware, in: Proceedings of the 10th International Symposium on Information and Communication Technology, SoICT '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 248–254. URL: https://doi.org/10.1145/3368926.3369702. doi:10.1145/3368926.3369702.

[9] T. N. Phu, L. Hoang, N. N. Toan, N. Dai Tho, N. N. Binh, C500-cfg: A novel algorithm to extract control flow-based features for iot malware detection, in: 2019 19th International Symposium on Communications and Information Technologies (ISCIT), 2019, pp. 568–573. doi:10.1109/ISCIT.2019.8905120.

[10] A. Azmoodeh, A. Dehghantanha, K.-K. R. Choo, Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning, IEEE transactions on sustainable computing 4 (2018) 88–95.

[11] E. M. Dovom, A. Azmoodeh, A. Dehghantanha, D. E. Newton, R. M. Parizi, H. Karimipour, Fuzzy pattern tree for edge malware detection and categorization in iot, Journal of Systems Architecture 97 (2019) 1–7. URL: https://www.sciencedirect.com/science/article/pii/S1383762118305265. doi:https://doi.org/10.1016/j.sysarc.2019.01.017.

[12] Artificial Immune System, John Wiley & Sons, Ltd, 2016, pp. 1–25. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119076582.ch1. doi:https://doi.org/10.1002/9781119076582.ch1. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119076582.ch1.

[13] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, TESSERACT: Eliminating experimental bias in malware classification across space and time, in: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, 2019, pp. 729–746. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[14] S. Sebastián, J. Caballero, Avclass2: Massive malware tag extraction from av labels, in: Annual Computer Security Applications Conference, ACSAC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 42–53. doi:10.1145/3427228.3427261.

[15] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.mirai, 2023.

[16] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.bashlite, 2024.

[17] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. C. Chaves, I. Cunha, D. Guedes, W. Meira, The evolution of bashlite and mirai iot botnets, in: 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 00813–00818. doi:10.1109/ISCC.2018.8538636.

[18] CrowdStrike, 2022. URL: https://www.crowdstrike.com/blog/linux-targeted-malware-increased-by-35-percent-in-2021.

[19] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.tsunami, 2023.

[20] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.dofloo, 2024.

[21] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.ngioweb, 2020.

[22] FortiGuard Labs, https://www.fortiguard.com/encyclopedia/virus/10077710, 2022.

[23] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.kaiji, 2022.

[24] FortiGuard Labs, https://www.fortiguard.com/encyclopedia/virus/6570964, 2015.

[25] Fraunhofer FKIE, https://malpedia.caad.fkie.fraunhofer.de/details/elf.hajime, 2022.

[26] FortiGuard Labs, https://www.fortiguard.com/encyclopedia/virus/8065787, 2019.

[27] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Dos and don'ts of machine learning in computer security, in: Proc. of USENIX Security Symposium, 2022.

[28] M. Jiang, Q. Dai, W. Zhang, R. Chang, Y. Zhou, X. Luo, R. Wang, Y. Liu, K. Ren, A comprehensive study on arm disassembly tools, IEEE Transactions on Software Engineering 49 (2023) 1683–1703. doi:10.1109/TSE.2022.3187811.