Highlights

Characterizing Linux-based Malware: Findings and Recent Trends

J. Carrillo-Mondéjar,J. L. Martínez,G. Suarez-Tangil

- We show that a data-driven approach is key for modern malware forensics. We also show that understanding malware is an important step in tackling the challenges posed to digital forensics by new computing platforms, such as those in the Internet of Things (IoT).
- We build an automated system to study the problem of malware in IoT systems. Our system uses both static and dynamic analysis, together with a similarity function designed to relate unknown samples to known threats.
- We use our system to assist a malware expert in the process of vetting recent unknown IoT malware samples and characterizing their behavior.
- We dissociate the problem of understanding Linux-based IoT malware from the IoT architecture used for malware in the wild.

Characterizing Linux-based Malware: Findings and Recent Trends

J. Carrillo-Mondéjar[a,*], J. L. Martínez[a] and G. Suarez-Tangil[b]

[a]Albacete Institute of Informatics (i3a). University of Castilla-La Mancha, Albacete, Spain
[b]Department of Informatics at King's College London (KCL), London, UK.

ARTICLE INFO

Keywords:
Malware Forensics
IoT
Embedded Systems
Data Analytics
Machine Learning
Expert Systems

Abstract

Malware targeting interconnected infrastructures has surged in recent years. A major factor driving this phenomenon is the proliferation of large networks of poorly secured IoT devices. This is exacerbated by the commoditization of the malware development industry, in which tools can be readily obtained in specialized hacking forums or underground markets. However, despite the great interest in targeting this infrastructure, there is little understanding of what the main features of this type of malware are, or the motives of the criminals behind it, apart from the classic denial of service attacks. This is vital to modern malware forensics, where analyses are required to measure the trustworthiness of files collected at large during an investigation, but also to confront challenges posed by tech-savvy criminals (e.g., Trojan Horse Defense).

In this paper, we present a comprehensive characterization of Linux-based malware. Our study is tailored to IoT malware and it leverages automated techniques using both static and dynamic analysis to classify malware into related threats. By looking at the most representative dataset of Linux-based malware collected by the community to date, we are able to show that our system can accurately characterize known threats. As a key novelty, we use our system to investigate a number of threats unknown to the community. We do this in two steps. First, we identify known patterns within an unlabeled dataset using a classifier trained with the labeled dataset. Second, we combine our features with a custom distance function to discover new threats by clustering together similar samples. We further study each of the unknown clusters by using state-of-the-art reverse engineering and forensic techniques and our expertise as malware analysts. We provide an in-depth analysis of what the most recent unknown trends are through a number of case studies. Among other findings, we observe that: i) crypto-mining malware is permeating the IoT infrastructure, ii) the level of sophistication is increasing, and iii) there is a rapid proliferation of new variants with minimal investment in infrastructure.

1. Introduction

The security community has been fighting against malicious software (malware) for over three decades. Prior to the proliferation of smartphones, malware developers mainly targeted Windows due to its market share. More recently, the focus of malware authors has slowly but steadily shifted towards Linux-based operating systems, the main factor being its growing popularity and the huge number of computing devices that are part of the Internet of Things (IoT).

The IoT is globally expanding (it is expected there will be 50 billion devices by 2020 [21]), providing diverse benefits in nearly every aspect of our lives, such as industry, transportation, smart-cities, health-care, entertainment and our daily life. One of the main characteristics of an IoT infrastructure is the ability to capture data, and this collected information can be the objective of attackers. Also, the huge number of devices with very limited characteristics in terms of available memory, bandwidth, power consumption, vulnerable software, etc. make it possible for a simple attack to achieve many objectives, for example to create a botnet network.

Embedded devices rely on a variety of different architectures. While PCs run predominantly on x86-flavored architectures, the IoT framework normally runs on open hardware architectures such as Advanced RISC Machine (ARM), PowerPC (PPC), and Microprocessor without Interlocked Pipeline Stages (MIPS), among others. From an attacker's point of view, IoT devices are quite attractive compared with PCs or laptops as they are online at all times and have no AntiVirus (AV) or Intrusion Detection Systems (IDS). These devices are created with innovative features to attract users, but often lack security and privacy measures. This makes many IoT devices vulnerable to critical security issues, ranging from the use of default passwords or insecure default settings to outdated software with known security issues [14]. All these weaknesses give attackers powerful attack vectors.

The trend for Linux-based malware that is especially designed for the IoT is relatively new compared with classical malware, and therefore the number of known malware families is still small. The first malware family especially designed for IoT devices is the Mirai botnet. This malware family was mostly aimed at performing Distributed Denial of Service (DDoS) attacks. The source-code was released in 2016 but this malware was first seen at the end of 2014. Since the release of Mirai's source-code, new variants of malware have proliferated. While some of these variants exhibit similar behaviors to that of Mirai[1], others present novel features and display completely different behaviors. Despite

*Corresponding author
javier.carrillo@uclm.es (J. Carrillo-Mondéjar);
joseluis.martinez@uclm.es (J.L. Martínez);
guillermo.suarez-tangil@kcl.ac.uk (G. Suarez-Tangil)

[1]In fact, these variants are adaptations of Mirai source code.

recent commercial reports having shown that IoT malware has been constantly growing [38], there is very little understanding of what the main characteristic features of recent malware samples are.

Due to the coexistence of multiple architectures in the IoT, malware analysis is an important part of the forensic process[2], allowing the forensic analyst to know how a sample behaves and what parts of the system it modifies. Therefore, machine learning and pattern recognition can be seen as a new paradigm for digital forensics [37, 44]. This is particularly relevant in malware forensics, where human analysts are confronted with a challenging adversarial task [27]. Advances in digital forensics require automated processes to aid malware analysts in the process of understanding: i) whether a binary seen in an investigation is malware or not (malware detection), and ii) which type of malware it is and what its expected behavior (malware characterization) might be. Furthermore, machine learning can be used to measure the trustworthiness of files collected during an investigation, but also to tackle challenges posed by tech-savvy criminals, such as the Trojan Horse Defense [8].

Contributions. This paper presents a comprehensive study of malware that is especially designed for Linux-based systems and tailored to IoT embedded devices. Our main contributions are as follows. (i) We build an automated system to study the problem of malware in IoT systems. Our system uses both static and dynamic analysis, together with a similarity function designed to relate unknown samples to known threats. (ii) We use our system to assist a malware expert (one of the co-authors of this paper) in the process of vetting recent unknown IoT malware samples and characterizing their behavior. We then present an in-depth analysis of the most recent unknown trends through a number of case studies. (iii) We dissociate the problem of understanding Linux-based IoT malware from the IoT architecture used for malware in the wild. We show the influence that the architecture has on the spread of malware. Finally, we have released the characterization produced for each of the clusters[3].

Findings. By systematizing the analysis of the most representative dataset of Linux-based malware collected by the community to date [14] and dissecting samples in the most relevant clusters, we have made the following findings:

1. We show that malware designed for architectures such as x86_64 is quite different to that designed to attack dedicated IoT architectures. Moreover, we also show that applying Machine Learning (ML) to understand large datasets of samples is challenging, especially when there is "noise" in the dataset. Due to the fact that malware especially designed for Linux-based systems is still simpler, the techniques based on static analysis work well, mainly because the disassembly code of malware is not obfuscated. However, dynamic analysis generally fails for two reasons: i) many samples cannot be executed on custom emulated IoT devices, as they do not provide the correct environment for which the malware has been designed; and ii) the Command and Control (C&C) server of the sample changes quickly over time, rendering the sample inoperative.

2. The level of sophistication of Linux-based malware varies significantly. It ranges from malware which uses classical but effective techniques such as brute force attacks, to malware which exploits some vulnerability to spread to other victims. Here, we show that crypto-mining malware is currently targeting IoT platforms. Finally, malware designed for the creation of bots to carry out denial of service attacks continues to appear.

3. Linux-based malware is publicly available on the Internet, causing the rapid proliferation of new variants which base their main techniques on artifacts that are publicly available on the Internet (e.g., on GitHub).

The rest of the paper is organized as follows. Section 2 introduces the threat overview related to this work. In Section 3.1 the methodology for understanding IoT malware is addressed. Section 4 presents the results of the classifiers when applied to labeled samples and the visualization of data. The results of the classifier tests with unlabeled samples are shown in Section 5. Section 6 shows the analysis of different groups of unlabeled samples related to each other. Section 7 contains a discussion about IoT malware, taking as its starting point the analysis presented in this paper. The state of the art in malware especially designed for IoT platforms is described in Section 8. Lastly, our conclusions are presented in Section 9.

2. Threat Overview

The IoT is a quickly developing entity made up of embedded and multi-platform devices. The majority of IoT malware focuses on devices with default login credentials, and these devices are being targeted by newly discovered Linux malware. In the last few years, malware in the IoT has gained more attention due to the damage caused by large-scale attacks such as the one carried out by the Mirai botnet. The increase in IoT devices together with the lack of updates in the face of the emergence of new vulnerabilities has led to the proliferation of malware targeting these platforms. In addition, the publication or leaking of the source code of some families such as Mirai or Bashlite causes new variants to appear or the creation of new families using part of their functionality or method to spread and infect new devices.

Unlike conventional malware that is aimed at a platform and/or specific architecture, IoT malware is available for different architectures such as MIPS, ARM or PowerPC, among others. Due to the wide range of Linux-based IoT devices,

[2] Locate, identify, collect and acquire data which may be relevant to an investigation.

[3] Characterization of the clusters can be downloaded from here: https://bitbucket.org/Dankitan/characterizing-linux-based-malware-findings-and-recent-trends/src/master/

the creation of malware capable of attacking and infecting many of them has become one of the priorities of cybercriminals.

Currently, the methods used to spread through the network are not very sophisticated. Primarily, they rely on the use of brute force attacks using default credentials or known vulnerabilities whose exploits are in most cases published on the Internet. In this way, they take advantage of the lack of updates and the use of default device settings by users.

Initially, IoT malware focused on the creation of bots to be marketed for the carrying out of denial of service attacks. Nowadays, the trend is changing and new samples are appearing for other purposes such as mining, which tries to take advantage of existing resources in the form of infected devices to mine cryptocurrencies.

Due to the time required to perform a manual malware analysis and the large number of attacks that appear every day, the security community is focusing its efforts on fighting a threat that is constantly growing. Therefore, it is necessary to build automated systems that, in addition to detecting new samples, allow the extraction of knowledge about the malware as well as its classification.

3. Methodology

In this section we present our methodology. We first provide an overview of the system and then describe each step in our pipeline.

3.1. Overview

Figure 1 shows an overview of the main building blocks of our pipeline, which consists of three phases. First, we have the feature extraction and the Modeling phase. The idea of this phase is to perform an exploratory analysis to guide the feature engineering process. Once the most relevant features in the IoT realm are selected, we build a supervised ML classifier capable of characterizing different known threats. Second, there is the Discovery phase. In this phase, we use the classifier trained in the previous phase and a metric based on sequences of opcodes from the disassembling of the malware to relate unknown samples to samples of known threats. Finally, we have the Vetting phase, in which we unveil novel threats that are unknown to the community. Specifically, we combine the features engineered in the first phase and the metric computed in the second phase with a custom distance function. This distance function is used to perform an unsupervised clustering task on unknown threats.

3.2. Modeling

In this section, we present the details of the Modeling phase. This phase is responsible for the collection and cleaning of the data as well as the selection of the characteristics that will be used for the learning of ML algorithms. The results of this phase will be presented in Section 4.

Data acquisition and cleaning

This research is based on a sizable dataset of Linux-based malware samples. Linux is the main operating system for the IoT infrastructure. For each malware sample, the dataset contains: a binary, features extracted from an automatic analysis and, in the majority of cases, a label characterizing the type of malware using AVClass [49] for readily available threat intelligence. Thus, the dataset consists of a set of different malware samples distributed among different architectures with Linux as the target operating system. The dataset used in this paper has recently been collected by the authors in [14] and the threat intelligence has been obtained from VirusTotal [55]. As mentioned above, AVClass has been used for labeling the samples. It is a tool which unifies sparse labels given by different AVs to each malware sample by assigning them to a malware family. Typically, AVClass relies on VirusTotal reports to label each sample on the basis of the labels provided by the AV. Basically, AVClass receives the labels from the AV engines and, once the duplicated labels have been removed, a tokenization and filtering process is applied. Finally, it applies an alias detection process and a ranking, using the most frequent labels as the family name for the family of a sample. One of the main limitations of AVClass is that it cannot label samples when there is no consensus among a minimum number of AV engines. Therefore, the final decision of labeling a malware sample depends on the quality of the AV labeling process. This process is an error-prone task, as pointed out previously [28] and as discussed below in Sections 4 and 6.3.

After obtaining the data and the labels, we clean the dataset by removing those samples that cannot be labeled by AVClass, either because they are not detected by more than two AV engines or because they are detected by the AV heuristics and, therefore, their labels are generic. Table 1 shows the number of samples distributed among the different architectures and how many of them are labeled.

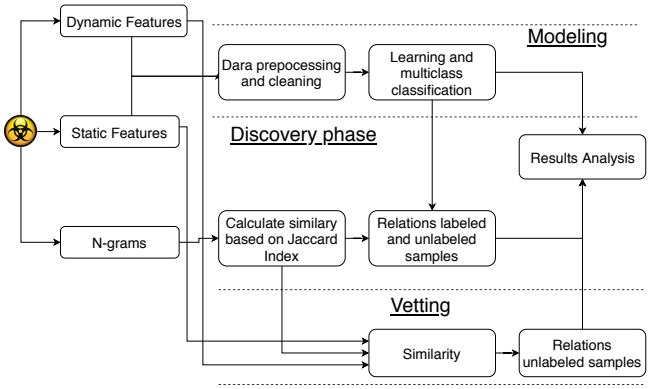

Figure 1: Overview of our system.

Feature analysis

The first approach to dealing with the problem of understanding malware behaviour, or even family malware classification, in the IoT is to apply Machine Learning techniques to as much information extracted from the dataset as possible. Although the use of ML to classify malware has been widely studied for other operating systems such as Windows or Android, there are not so many studies focused on dif-

Architecture	Labeled	Not Labeled
ARM 32-bit	493	62
ARM 64-bit	31	16
AMD x86-64	1765	1253
Intel 80386	607	113
MIPS I	1960	160
Motorola 68000	1169	47
SPARC	1144	26
SPARC Version 9	0	1
PowerPC	1509	60
PowerPC 64-bit	1	0
Hitachi SH	127	3
EM_ARC_A5	1	0
	8807	1741

Table 1
Samples distributed by architectures.

ferent operating systems for the IoT such as Linux-based ones. One of the problems that we face when building a generalized model is the diversity of existing architectures, as Linux-based operating systems and compilers make the same sample look different even when it belongs to the same malware family. At this point, the first step was to try to determine the most significant features from among all the information available in the dataset. For this, we use the following feature extraction techniques.

- *Static features*: analysis focusing on intrinsic characteristics in the executable itself without actually executing it. Within this group are characteristics of the binary file such as headers, strings that appear in the executable, machine code, imports of the library functions, entropy, etc.
- *Dynamic features*: analysis focusing on features obtained through monitoring the execution of an executable. For example, system calls made to interact with the system, information about opened files, memory, etc, that is, information about the behavior of the executable at run time.
- *Hybrid features*: the combination of the two previous methods, so that the characteristics used are extracted in a static and dynamic way.

To design our model, we decided to follow the three approaches. Although using only static features can cause a bad generalization of the model, since some of these characteristics could change between different architectures (especially in the IoT scenario), we believe that there are some static characteristics that should remain unchanged between different platforms, as for example strings. In terms of dynamic features, the behavior of the same sample must be similar regardless of the architecture on which it is executed. For this reason, we started with a single static model, then we included a dynamic model in the analysis and, finally, a model that combines both characteristics has been developed.

Static features selected

Machine learning models require a diverse set of features to learn. One of the disadvantages of static features is that they are susceptible to obfuscation, although in the IoT scenario it is not very common, with around only 3% of the samples inside the dataset being packed, most of them using Ultimate Packer for Executable (UPX) or some variant of it. This is mainly because, nowadays, malware for Linux-based systems does not implement techniques to avoid AV or IDS; with resource-constrainted devices, such as those used in the IoT infrastructure, there is not enough memory or battery power to deploy them. Thus, the proposed algorithm firstly looks for static features that do not change too much between different platforms. We do not consider features such as assembly code to be a good option since the same sample can be compiled for different architectures, and the instructions set for each one is different. This is an important issue because, as was shown in Table 1, the number of different architectures for Linux-based operating systems is large. Although there are several static features that could work well, such as text strings, we have focused on data based on Cyclomatic Complexity. Cyclomatic complexity [57] is a metric that is used in software engineering to calculate, in a quantitative way, the complexity at the logical level of a program or function. In our case, we have used the Cyclomatic Complexity of all program functions, as well as the maximum and average complexity. For example, say a malware sample has five functions with Cyclomatic Complexities 5, 7, 5, 3 and 5 respectively. We then account for 3 functions with complexity 5, 1 with 7 and 1 with 3. The maximum would be 7 and the average 5. We also use the number of basic blocks and its average. A basic block consists of a set of instructions that begins with a single instruction and ends with an instruction that redirects the execution flow to another basic block or ends the program. These characteristics allow us to know whether two programs are structurally similar at the machine code level. Since there are very few packed samples and we wish to check whether the use of static features works well for IoT malware, we have discarded packed samples from the dataset.

Dynamic features selected

Regarding characteristics based on behavior which are collected at run time, we have chosen the following features:

- Unique syscalls.
- Ioctl.
- Rename process.
- Number of processes created.
- Check user and group identifier.

These features have mainly been encoded in a binary way (one or zero), indicating whether they are used (one) or not (zero). For example, if a process uses the ioctl SIOCGIFHWADDR command to obtain the MAC address of an interface, we use the number one to indicate that that sample uses

Architecture	Labeled	Not Labeled
ARM 32-bit	445	47
MIPS I	1705	63
AMD x86-64	1600	787
Intel 80386	523	71
PowerPC	1492	49
TOTAL	5765	1017

Table 2
Number of final samples.

that feature. Otherwise, it will contain a zero. The number of processes created is encoded as a numerical feature.

While dynamic features are extremely useful to characterize the behavior of a program, we note that it is not always possible to obtain features for all samples. This is because performing dynamic analysis for certain IoT architectures is a challenging task [14]. This is due to the problem of managing different virtual machines to emulate a wide range of architectures. In addition, analyzing malware is a time sensitive task as, at the time of execution, C&C infrastructures or other external resources needed for the execution of the malware might be unavailable.

Moreover, families that do not contain at least ten samples have been removed from the dataset because we consider that less than ten samples is not enough for the model to generalize correctly. Table 2 shows the final sample set once the families are extracted. Likewise, packed samples and samples that do not contain dynamic feature are also filtered out from our analysis.

3.3. Discovery phase

This phase is aimed at characterizing the dataset of samples that are not labeled. For this, we will use the classifier built in the previous phase. Note that it is not possible to assess quantitatively whether the predictions made by the classifier are correct or not (there are no labels). Thus, in this phase we measure the similarity between predicted samples and labeled samples as a method to evaluate our accuracy (Section 5). Intuitively, samples that are associated with a family might be structurally similar to the samples of that family. This holds true for samples that are not obfuscated and thus we see this evaluation criteria as an underapproximation. It is also important to note that the performance of the classifier is formally evaluated in the previous phase using k-fold cross-validation, as shown in Section 4.1. However, this phase aims at discovering unknown threats and the similarity metric should be seen as a means of providing confidence in the decisions of the classifier.

The similarity metric is calculated as follows. First, we disassemble each executable. Then, we compute the sequences of operation codes (opcodes) of size n, namely n-grams. Listing 1 shows a small example with five assembly code instructions. For a sequence of $n = 3$, the resulting set of three-grams is: (push, mov, push), (mov, push, push) and (push, push, sub).

```
push rbp
mov rbp, rsp
push r12
push rbx
sub rsp, 10h
```

Listing 1: Extracting n-grams

Therefore, given two sets of n-grams, the similarity between them can be calculated using the Jaccard index [35] as:

$$jacc(s_1, s_2) = \frac{|s_1| \cap |s_2|}{|s_1| \cup |s_2|}, \tag{1}$$

where the numerator indicates the number of subsets (denoted by $|s_i|$ for $i \in \{1, 2\}$) that are found in both samples, s_1 and s_2, and the denominator indicates the total number of unique subsets between s_1 and s_2. The result is a value between 0 and 1 that indicates the degree of similarity between two sets. As mentioned above, the similarity based on n-grams works well as long as the samples are not obfuscated or packed. The results for the number of unlabeled samples that match at the n-gram level with labeled samples, and the number of hits of the classifier, are presented in Section 5.2.

3.4. Vetting

While the previous phase aims at associating unlabeled samples with known threats, this phase aims at characterizing novel threats from our dataset of non-labeled malware. For this, we cluster samples in the non-labeled set using unsupervised learning. This clustering process is driven by a custom distance function that relies on both the features extracted in phase one (Section 3.2) and the similarity index from phase two (Section 3.3). This function is formalized as follows:

$$\texttt{distance}(s_1, s_2) = \frac{\sum_{i=0}^{|F|} \frac{min(f_i^{s_1}, f_i^{s_2})}{max(f_i^{s_1}, f_i^{s_2})} \times \frac{1}{|F|} + jacc(s_1, s_2)}{2} \tag{2}$$

where $f_i^{s_1}$ is the feature i of sample s_1 (likewise for s_2) and $|F|$ is the total number of features. In other words, we normalize the feature vectors with respect to all other samples in our dataset samples. Given two vectors, the similarity index is calculated by means of the average distance of each pair of features. For example, given a sample with 100 functions and the use of the read, write and execve syscalls, and another sample with 80 functions and the use of the read, write and open syscalls, we obtain the following two vectors: $(100, 1, 1, 0, 1)$ and $(80, 1, 1, 1, 0)$. The resulting similarity index is 0.56, , which is computed as follows: $((0.8/1 + 1 + 1 + 0 + 0)/5)$. This means that the similarity between the two sets of features is 56%. Then, we compute the average between the index of similarity obtained at the n-gram level (calculated in the previous section) and at the level of features (Section 3.2).

Feature type	Algorithm	Features	Precision	Recall	FScore
Static	K Neighbors	470	80,99	78,22	77,27
		220	80,67	78,42	77,38
	SVM kernel='rbf'	470	73,46	80,91	72,53
		220	74,23	82,29	73,29
	SVM kernel='Linear'	470	62,10	56,33	57,09
		220	58,10	54,64	53,71
	Decision tree	470	82,18	83,44	79,61
		220	80,58	83,18	78,85
	Random forest	470	**83,01**	**83,67**	**81,02**
		220	**84,23**	**84,41**	**81,70**
Dynamic	K Neighbors	211	**86.17**	82.08	82.46
		134.67	**84.79**	81.42	81.99
	SVM kernel='rbf'	211	83.29	85.58	82.38
		134.67	82.28	86.50	82.13
	SVM kernel='Linear'	211	84.49	82.38	81.70
		134.67	82.09	81.15	80.32
	Decision tree	211	80.06	86.78	81.26
		134.67	80.32	88.71	81.93
	Random forest	211	83.28	**87.60**	**84.08**
		134.67	82.72	**88.96**	**83.91**
Hybrid	K Neighbors	681	86.60	83.40	83.31
		269.5	86.31	84.43	84.04
	SVM kernel='rbf'	681	**88.80**	84.17	84.47
		269.5	**88.26**	86.07	85.35
	SVM kernel='Linear'	681	87.92	85.39	85.37
		269.5	83.14	81.47	81.02
	Decision tree	681	81.36	84.80	80.30
		269.5	82.31	83.53	80.57
	Random forest	681	88.08	**87.17**	**86.55**
		269.5	87.70	**86.61**	**86.07**

Table 3
Results for the different types of algorithms and characteristics used in our model.

To cluster samples together, we use the similarity function in Equation 2. We consider that two samples are in the same cluster if they have an index higher than 0.8, that is, if they have a ratio greater than 80%. We chose this empirically after testing different thresholds. A higher threshold does not produce a notable increase in performance. However, a lower threshold produces false positives. We also use this distance function to represent the relationships between the malware samples in the form of a graph. For the purpose of our study, we consider that clusters of unlabeled samples alone constitute a group of samples that belong to a threat or family that it is currently unknown. Where applicable, we also connect clusters of unlabeled samples to clusters of known threats using the threshold described above. This enables us to vet the discovery phase discussed above (see Section 3.3). The most interesting clusters are further discussed in Section 6.

4. Modeling

This section evaluates the effectiveness of the modeling phase presented in Section 3.2.

4.1. Machine Learning

First, static features have been used to train the model. Then, dynamic characteristics have also been used and, finally, all the features have been included. Basically, we have carried out these experiments with different types of characteristics to understand how much each characteristic contributes to machine learning algorithms. This is because the automatic extraction of static features is far more efficient than automatic dynamic analysis, which in the best case can take several minutes of execution in a sandbox environment, causing the extraction of features to consume a lot of time.

Then, these same tests have been performed by applying feature selection to eliminate those features that contribute least to machine learning algorithms. Different machine learning algorithms have been used, such as Random Forest [9, 58], K-nearest [34, 58], Decision Tree [45, 58] and Support Vector Machines (SVM) [23, 58] using the Radial Basis Function (RBF) and Linear kernels [58]. The implementation of these algorithms was provided by the sklearn library [43], which is written in python.

As a dataset, 5765 samples containing more than 20 different families have been used. K-fold [50] has been performed as a cross-validation strategy with the dataset being divided into 6 stratified folds, so that there is the same per-

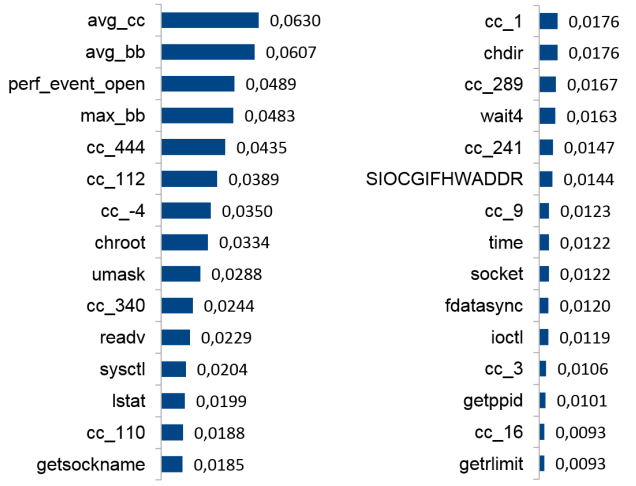

Figure 2: Importance of the 30 features that contribute most value to the Random Forest algorithm.

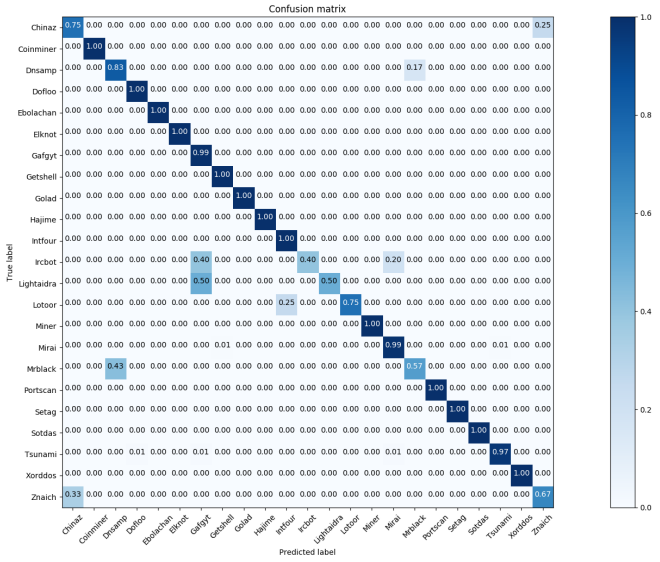

Figure 3: Confusion Matrix of the Random Forest Algorithm using the hybrid approach.

centage of samples from each family in each split. Feature selection has been performed for each training split in the cross validation loop, since the test is considered unseen.

As evaluation metrics, *precision*, *recall* and *Fscore* [12] have been used. These metrics have been applied to each class independently, with the final result being the average of all the classes, that is, each class has the same weight in the results regardless of the number of samples. Table 3 shows the results obtained with each of the classifiers. The best results obtained by the ML algorithms are in bold. It can be observed that in no case is there much difference in the use of all the features from the use of a subset of them, and in some cases it even improves the results when applying feature selection. In general terms, Random Forest offers the most balanced results in terms of *precision*, *recall* and *FScore*. In addition, it can be observed that the use of only static features does not work very well with SVM.

As mentioned above, the selection of variables produces similar results in most cases to those when working with the complete set of characteristics. Figure 2 shows the 30 most representative variables of the random forest (dynamic + static) algorithm, that is, those whose influence on the model is greatest. We can observe that the two characteristics that most influence this algorithm are the average cyclomatic complexity and the number of basic blocks. Among the other static characteristics that appear, it is important to highlight certain cyclomatic complexities of functions that have a very high value, such as cc_444, cc_340 and cc_289. Among the features based on its behavior are those related to syscall sockets, ioctl, perf_event_open, chroot, etc.

Table 3 shows the results obtained for the different algorithms and features used. We can observe that most of the models yield a performance of around 85% on average. As we discussed above, the metrics used have been applied to each class independently. The overall results are summarized using the average, which does not take into account the size of the class. Here, we observe misclassifications of families with a small number of samples (e.g., 10 samples), which brings the average down despite having large classes (e.g., 2K samples) correctly classified. The confusion matrix in Figure 3 details the percentage of correct and incorrect labels predicted in each class and provides a more accurate picture of the classification performance. In particular, as a representative example, Figure 3 shows the confusion matrix generated for the Random Forest hybrid model. If we look this matrix, we can see that 25% of the time the chinaz family is treated as znaich, and 33% of znaich is treated as chinaz. Actually, this makes sense if we look at it: the name of the family is very similar, chi-na-z and z-na-ich, and therefore it is very likely that it is the same family which is labeled differently or that both families have been created by the same authors. The same case occurs with Mrblack and Dnsamp: several anti-viruses label these samples as Mrblack and others as Dnsamp, so both are the same family. In the case of Ircbot and Lightaidra, some samples could have been incorrectly labeled. After reviewing the VirusTotal reports of some samples labeled as Ircbot and Lightaidra, we observed that some anti-viruses mark some of them as Gafgyt or Mirai. This makes us think that the prediction could be correct and some of these samples are these kinds of malware, or that as many of the malware samples in the IoT have had their source code leaked, there are similar code pieces in samples from different families, causing the antivirus to label them incorrectly due to the fact that they can match with static signatures or on the basis of their heuristics. We should bear in mind that AVClass labels are determined by the consensus of the different labels that the AV gives to a sample.Therefore, the label selected as AVClass may be incorrect.

4.2. Data visualization

The objective of this section is to visualize the set of samples using a three-dimensional plane so that we can observe in a more visual way whether the chosen character-

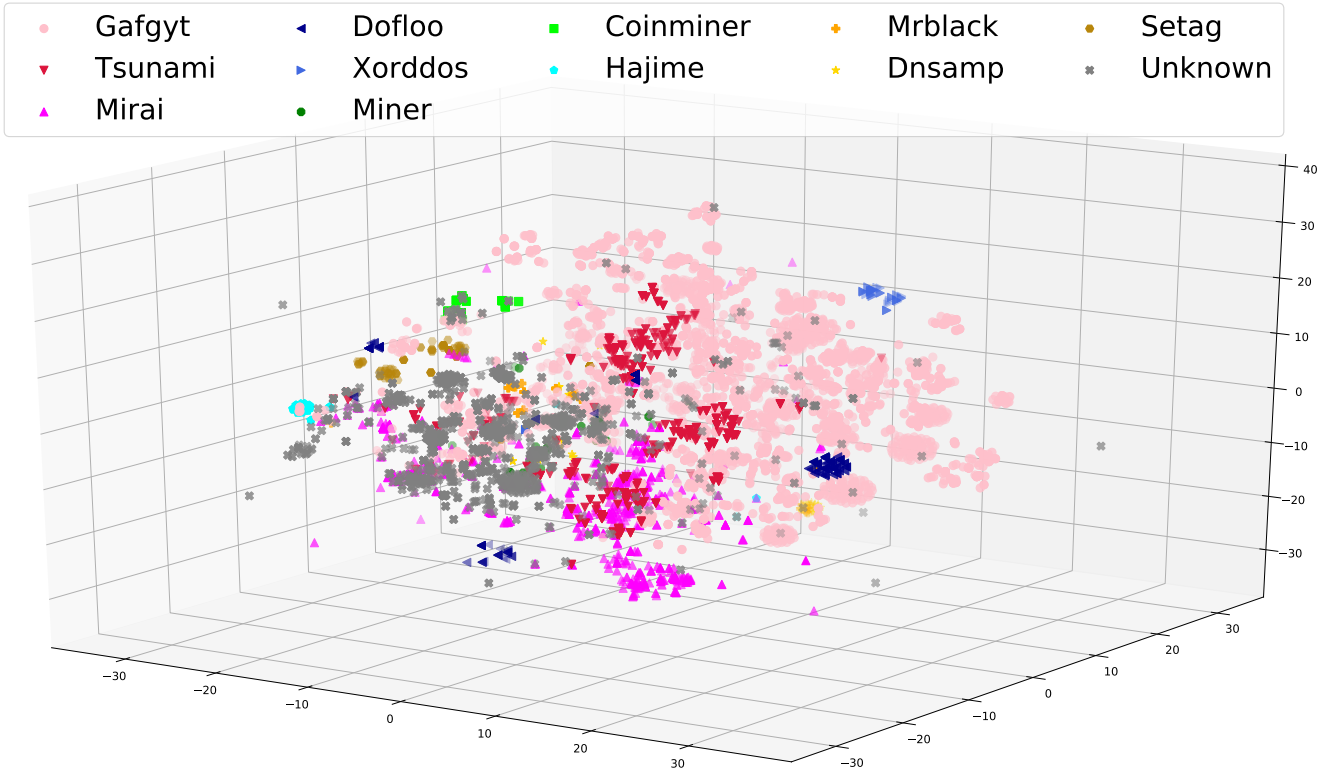

Figure 4: Visualization of the labeled malware samples with the greatest presence in the dataset and unlabeled samples.

istics allow the generalization of the families that are part of the dataset. Since the number of features is much higher, we have used T-Distributed Stochastic Neighbor Embedding (T-SNE) [36] to reduce the data dimensionality and visualize them correctly. In order to improve the visualization of the samples and make this clearer, we will only show the families with the largest number of samples on the chart.

Figure 4 shows the most populated families of the dataset. We plot the families and their labels together with the unlabelled samples. Note that it includes samples compiled for different architectures. We see that our set of features are representative regardless of the architecture. In general, samples of a family are close in the plane to other samples of the same family. For example, the families with the largest number of samples. We observe that samples from Gafyg, Mirai and Tsunami are grouped together. We observe also that there are sub-clusters which are attributed to either set of samples from the same architecture or to variants within the family. Intuitively, when we look at families that are less populated such as Hajime or Dofloo, we observe fewer subclusters. On the other hand, when we look at the representation of the unknown families, we see that they are very close to each other and distributed over several clusters. This tells us that unlabeled samples can be attributed to a handful of new, but prolific, families.

5. Discovery phase

In this section, we test the classifier with the unknown set of samples.

5.1. Description

As we mentioned in Section 3.3, this set of samples is not labeled so we do not have a quantitative way to know whether the prediction is correct or not. Therefore, we calculate the level of similarity between unlabeled and labeled samples and we check how many unlabeled samples are similar at the n-gram level with labeled samples and whether the label predicted by the classifier is correct in those samples that display similarity.

The steps followed to calculate the similarity are the following:

1. Extract the sequences of opcodes from each of the malware samples and generate the n-grams.
2. Compare each sample of unlabeled malware with the other samples of that same architecture since each architecture has a different set of instructions.
3. Check how many unlabeled samples are related to labeled samples.

The n-gram tests have been carried out evaluating different values of n (6, 8, 10, 12 and 14). In the case of a small size of n-gram, a high similarity level might be obtained due to the fact of having sets with small opcode sequences in different parts of the program functions, and the fact that the Jaccard Index only takes into account unique sets of n-grams. Increasing the n-gram size allows us to match similar samples in a more reliable way because it is more complicated to match sequences of, for example, 14 opcodes without there

being any relation. The threshold used to consider that they are the same sample is 0.8, that is, that the degree of similarity between two samples is greater than 80%. As with the size of the n-gram, the higher the threshold, the greater the probability that two samples that exceed that threshold will be from the same family, that is, the probability of false positives will be lower. Based on the similarity at the n-gram level, there are between 126 and 134 (around 12%-13%) unlabeled samples that are related to samples whose label is available.

5.2. Evaluation

Once the similarities between unlabeled samples and others have been obtained, we can test the classifier with those samples whose label we do not know and verify, in a quantitative way, how many labels it correctly predicted. The procedure that has been carried out is the following:

1. We pass the new samples through the different classifiers using both static and dynamic features and feature selection, obtaining the predicted label for each of the samples according to the classifiers.

2. For each predicted sample we search with the samples with similarity higher than 80% calculated previously. We check those samples that have a label if it matches the label that the classifier predicts. A threshold of 80% has been chosen so that the samples that exceed it have a high probability of being from the same family.

Table 4 shows the results obtained for each algorithm and size of n-gram used. The results are classified by architecture, indicating how many samples have been predicted from the totals for that architecture. Also, the hits column is shown as the total number of hits with respect to the total number of unlabeled samples that have a relationship with samples with labels at that n-gram size. Finally, the last column shows the calculated percentage of success.

If we look at the table in more detail, we can see that with the ARM, MIPS and PowerPC architectures, almost all the algorithms come close to correctly predicting almost all the samples. However, for the AMD x86-64 architecture, in most cases it correctly predicts nearly 66%. This makes sense, since it is the CPU architecture that is most widely used on desktop computers and in many cases the malware's source code has been leaked on the Internet, so it may have been modified, mixed with different malware functionalities, etc. This produces new variants of the malware which are created to improve the sample or to try to avoid detection by antivirus systems. In addition, as it is the architecture that is most widely used on personal computers, even a simple port scanner can be classified as malware and fall within the dataset.

6. Case study

After grouping similar samples together, we next characterize their functionality and study the most relevant threats as discussed in Section 3.4. For the purpose of this case study, we use a similarity of 80% or higher when building the clusters. We chose this threshold empirically after manually evaluating the resulting clusters. We refer the reader to Section 7 for a discussion on the implications behind our choice.

In this section, we report results that are unknown to the community (i.e., unlabeled samples). Thus, clusters of samples that are not associated with an unlabeled sample are not reported. We then describe a number of Linux-based malware threats targeting the following architectures: AMD, MIPS, ARM, PowerPC and Intel. Table 5 summarizes the findings of the unknown samples for each platform.

6.1. AMD x86-64

AMD x86-64 is the architecture with the largest number of unknown threats as well as the most targeted platform, as shown in Table 2. Figure 5 shows a number of unlabeled clusters of AMD x86-64 Linux-based malware samples (depicted in gray) and their relationship with known threats (a different color per threat). Samples are represented as nodes of the graph and the edges represent that two samples are similar. The samples that have labels are represented by different colors and have some relation with some unlabeled samples. We observe three distinct patterns: i) clusters of unlabeled samples alone (clusters #2-#16); ii) clusters with unknown samples associated with known threats (unnumbered clusters); and iii) unknown samples that are not similar to any other sample (i.e., singletons). We next present a number of case studies characterizing primarily the most relevant clusters of i), i.e., unknown clusters alone. However, we also describe cases where a large number of unlabeled samples are clustered together with a small number of labeled samples. While the former are relevant to understanding novel threats, the latter can shed some light on why signatures for "semi-known" are not successfully detecting all other related threats.

6.1.1. Dropper (clusters #9 and #12)

Our vetting process shows that samples in clusters #9 and #12 in Fig. 5 have a similar behavior, namely they are both droppers. In both clusters, all samples contain a payload that is encrypted and stored in the data section of the executable. We also observe that the decryption routine is similar. In particular, they seem to be using the RC4 algorithm or a variation of it. While these two clusters share a similar backbone, they have been grouped in different clusters mainly because the dynamic features of samples in cluster #9 differ from those in #12. This indicates that samples in these two clusters might belong to the same actor (family and/or botnet), although they participate in different campaigns.

The samples in cluster #12 belong to the largest unlabeled cluster in our dataset. After deciphering the payload of the samples in this cluster, we observe an in-memory bash script that is executed during runtime via the *execvp* syscall. A dump of the script is presented in Listing 2. This routine simply checks whether the target machine has wget or curl to download a file and prepares for its execution. Also, it informs the server of the user name of the machine, the IP address and operating system. At the time of writing, some

Algorithm	ngram	ARM 32-bit	MIPS I	AMD x86-64	Intel 80386	PowerPC	Hits	Percent
K neighbors	6	35/38	18/19	38/60	3/4	11/13	105/134	78,36 %
	8	35/37	18/18	38/59	3/4	11/13	105/131	80,15 %
	10	35/37	18/18	36/57	3/3	11/13	103/128	80,47 %
	12	35/37	18/18	36/57	3/3	10/12	102/127	80,31 %
	14	35/37	18/18	36/56	3/3	10/12	102/126	80,95 %
SVM kernel=rbf	6	38/38	19/19	40/60	3/4	12/13	112/134	83,58 %
	8	37/37	18/18	40/59	3/4	12/13	110/131	83,97 %
	10	37/37	18/18	38/57	3/3	12/13	108/128	84,38 %
	12	37/37	18/18	38/57	3/3	11/12	107/127	84,25 %
	14	37/37	18/18	38/56	3/3	11/12	107/126	84,92 %
SVM kernel=linear	6	37/38	19/19	39/60	3/4	12/13	110/134	82,09 %
	8	37/37	18/18	38/59	3/4	12/13	108/131	82,44 %
	10	37/37	18/18	36/57	3/3	12/13	106/128	82,81 %
	12	37/37	18/18	36/57	3/3	11/12	105/127	82,68 %
	14	37/37	18/18	36/56	3/3	11/12	105/126	83,33 %
Decision Tree	6	37/38	19/19	37/60	3/4	11/13	107/134	79,85 %
	8	36/37	18/18	37/59	3/4	11/13	105/131	80,15 %
	10	36/37	18/18	37/57	3/3	11/13	105/128	82,03 %
	12	36/37	18/18	37/57	3/3	10/12	104/127	81,89 %
	14	36/37	18/18	36/56	3/3	10/12	103/126	81,75 %
Random Forest	6	38/38	19/19	41/60	4/4	13/13	115/134	85,82 %
	8	37/37	18/18	39/59	4/4	13/13	111/131	84,73 %
	10	37/37	18/18	38/57	3/3	13/13	109/128	85,16 %
	12	37/37	18/18	38/57	3/3	12/12	108/127	85,04 %
	14	37/37	18/18	38/56	3/3	12/12	108/126	85,71 %

Table 4
Predictions of the classifiers with the unlabeled samples that maintain similarities at the n-gram level with labeled samples.

Arch	Type	Comments
AMD x86-64	Dropper	Script Bash encrypted with RC4 and stored in the data section. Once decrypted, the script downloads an executable related to cryptomining campaigns.
	Flooder	Malware used to perform flooding and botnets with flooding capabilities or DDos.
	Shellcode and Exploits	Dirtycow and some type of privilege elevation exploits.
	Goscanssh	Malware written in Goland that spreads through SSH services vulnerable to brute force attacks.
	Python	Embedded Python interpreter. Some python scripts belong to a Drobur malware, others are benign.
ARM 32	Botnets	Known malware such as Mirai and Gagyt.
	Exploit	Privilege escalation exploits on Android devices.
MIPS I	Botnets	Known malware such as Mirai, Gafgyt, Remaiten, TheMoon, Dofloo and Dnsamp.
	Hacktool	Hacking tools such as aircrack or ones to clean the footprints of a system.
	Downloader	Mirai malware downloader.
PowerPC	Botnets	Known malware such as Mirai, Gafgyt, Tsunami and Remaiten. Mainly used to perform DDos attacks.
	Exploit/Shellcode	Generic shellcodes and exploits.
Intel 80386	Virus	ELF File infector known as Vit.
	Backdoor	Ganiw/Setag family, which executes the commands received by the cybercriminals.
	Rex	Malware family written in Goland with mining, flooder and brute force capabilities.

Table 5
Summary of the results found in the unknown samples for each of the platforms analyzed.

samples observed in this campaign are active and the infrastructure still operational. In particular, the executable dropped by the samples of this campaign fetch an AMD-compatible version of a crypto-mining tool. This tool uses CryptoNight Proof-of-Works (PoW) through Stratum, a Pool mining protocol [46]. Pool mining protocols are used to build a distributed partnership to verify the different PoWs in a block. In this way, the complexity required to check a block is distributed among the participants. Therefore, the set of samples in #12 are droppers that are responsible

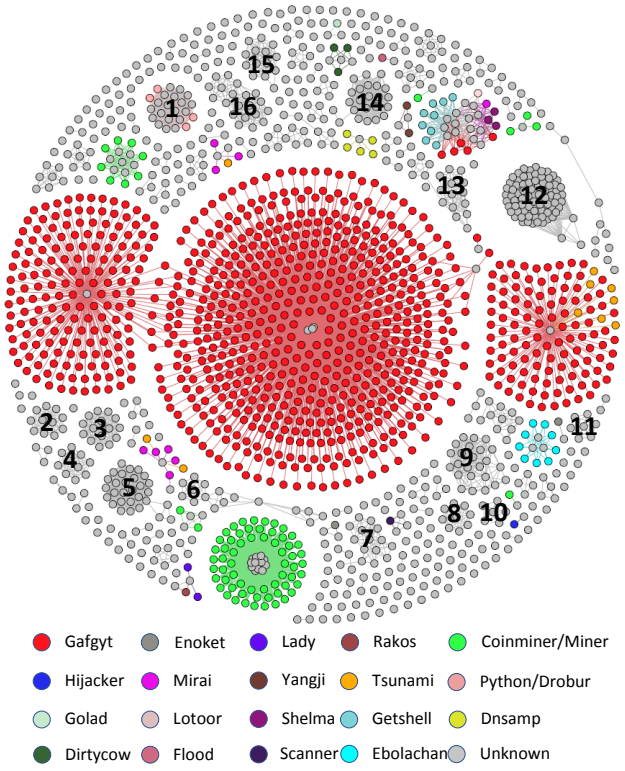

Figure 5: Unknown AMD x86-64 clusters and their relationship with known threats.

for downloading and executing custom crypto-mining malware. The use of CryptoNight PoWs indicates that droppers are probably part of a larger botnet mining ASIC-resistant cryptocurrencies [41]. ASIC-resistant mining prevents the use of dedicated mining hardware. Thus, using IoT malware to mine can be profitable, especially when mining through a Pool. Finally, we also observe a few unlabeled samples that are related to known mining campaigns (c.f., green cluster in Fig. 5 for a Coinminer campaign).

Listing 2: bash version

```
#!/bin/bash
if [ ! -f "/var/tmp/. "/x ] || [ ! -f "/var/tmp/. "/xh ] ; then
        #echo File not present. Installing.

    if [ -x /usr/bin/wget ]; then
       wget --dns-timeout 10 --user-agent="wget" --connect-timeout 20
            --read-timeout 30 -q -O $mydir/xx.tgz
            --header "Host:www.btcsavetheworld.org"
            "hxxp://104.24.123.53/xlatest.tgz" 2> /dev/null >> /dev/null
    else
       if [ -x /usr/bin/curl ]; then
           curl -f -A "curl" --connect-timeout 10 -qs --max-time 500
                --header 'Host:www.btcsavetheworld.org'
                "hxxp://104.24.123.53/xlatest.tgz"
                -o $mydir/xx.tgz >> /dev/null 2> /dev/null
       fi
    fi
    if [ -f $mydir/xx.tgz ]; then
        mypwd="$(pwd)" 2>/dev/null >> /dev/null
        cd $mydir 2>/dev/null >> /dev/null
        tar -zxvf xx.tgz 2>/dev/null >> /dev/null
        rm -rf xx.tgz 2>/dev/null >> /dev/null
        mkdir -p "/var/tmp/. " 2>/dev/null >> /dev/null
        mv -f init "$myfile" 2>/dev/null >> /dev/null
        mv -f xh "$myfile" 2>/dev/null >> /dev/null
        chmod +x "$myfile" "$myxh" 2>/dev/null >> /dev/null
        cd "$mypwd" 2>/dev/null >> /dev/null
        #echo File is installed
    fi
fi
```

When looking at samples in #9, we observe that the main differences lie in the dynamic characteristics, presenting different types of syscalls (e.g., ioctls). This relates to the differences in the behavior of the dropped file. Understanding in detail the characterizing features of the executable files dropped by this campaign is the aim of our future work.

6.1.2. *Flooder (#2, #5, #7, #10, #11, #14, and #15)*

The next most relevant threat is structured around clusters #2, #5, #7, #10, #11, #14, and #15 (see Fig. 5), and it relates to network flooders. We also found a few unlabeled samples assigned to clusters largely populated with samples of known flooders such as: *Gafgyt*[4], *Mirai*, *Tsunami* or *Flood*. These are tools that are typically used to perform Distributed Denial of Service (DDoS) attacks. We have vetted most of the samples in these seven unlabeled clusters. We have observed that all these flooders can be characterized into the following groups according to the protocol they rely on: UDP, SSDP, NTP and Netbios. Similarly to what we observe with the droppers, some of these variants share a common backbone. However, in this case the fact that some clusters share a common backbone does not mean that the clusters are part of different campaigns from the same actor. Instead, they have proliferated after the source code of some of these samples became available [47]. During our analysis, we have been able to track some characterizing features (e.g., strings that are referenced in the disassembly) of the samples in these clusters to IoT flooders with source code that it readily available online. This might explain why this type of threat has proliferated so quickly over the last few years [4, 31].

6.1.3. *Shellcode and Exploits (#4, #6, #8, #13, and #16)*

We have observed a wide range of clusters with malware that runs shellcode and executes exploits. The most notable cluster is #4, which uses an exploit that leverages the so-called *dirtycow* vulnerability [17]. This exploit takes advantage of a race condition in the kernel's memory management system to escalate privileges. We performed a reverse search between the exploit found in the samples of cluster #4 and exploit-db [20], a popular open-source repository for exploits. We found that the samples analyzed share code with one exploit-db entry.

Likewise, samples in clusters #6, #8 and #13 belong to some type of privilege elevation exploit. The flow of the logic and the disassembled samples are very similar, but we did not find any payload that would allow us to characterize these clusters. There are similar functions (such as "kernel_code" or "get_kernel_sym") in all the samples but they are not referenced in the programs that we have vetted. We have not been able to identify any corresponding CVE associated with the exploit either. These clusters probably correspond to more sophisticated malware samples than those observed before.

Finally, cluster #16 is largely related to malware using shellcode. Contrary to what we observed in the previous

[4]For instance, Gafgyt (depicted in red in Fig. 5) is a known IoT botnet that has flooding capabilities.

clusters, we find that most of the samples in this cluster are simple programs. These samples are mainly used to test and debug the shellcode, where they print the size of the binary and make a call at the beginning of the code. Typically, this is used to verify that the code is executed correctly before including an exploit in it or distributing it in online forums [42] or underground markets [1, 54]. Within these samples, there are several that invoke system functions using a PowerShell command as an argument. PowerShell has been heavily abused by criminals to easily create cross-platform *fileless* infections — files that are fetched dynamically and executed directly in memory [24].

6.1.4. GoScanSSH (#3)

Another prominent cluster is #3, which is also shown in Fig. 5. Samples in this family are related to the GoScanSSH family [10]. This family spreads by searching for servers that are vulnerable to brute force attacks over SSH. The malware is written in the Go language and all samples are stripped binaries with layout obfuscation (i.e., no meaningful function names). However, Go maintains a section called ".gopclntab" with a manifest of the functions that might keep the function names if not properly obfuscated. Despite the binary being stripped, we were able to map this cluster to the GoScanSSH family by looking at the ".gopclntab" section. An analysis of this section of the binary reveals that the samples in cluster #3 attempt to perform ssh activity, that they also generate wordlists (typical of dictionary attacks), and that the malware maintains a blacklist of domains. Judging by the VirusTotal *first seen* attribute associated with the samples in this cluster and the first known report modeling this threat [10], we are able to tell that some of these samples have remained undetected for more than a year.

6.1.5. Python and Drobur (#1)

This refers to cluster #1 in Fig. 5. Although the unknown samples in this cluster are related to several labeled samples, there is no consensus among the labeled samples. In particular, some AV vendors label some of these samples as Python and some others as Drobur. When we go deeper into analyzing these samples, we verify that they all have an independent Python executable. Specifically, the executable has the embedded Python interpreter, as well as its dependencies, so that it can be executed without the need of having Python installed on this architecture. Many IoT platforms are hardened, and it is thus common to find malware samples that piggyback all the dependencies needed to set the scene.

Note that what characterizes the behavior of the samples is the Python script that is executed in each infection, rather than the piggybacked tools. This means that although the distance between these samples might be small, the common factor is generally the use of the same auxiliary tool. The fact that they all have the Python interpreter might mean that these samples belong to the same campaign (e.g., same dropper or the same Pay Per Install botnet). However, it could also be that a given sample is not related to the others at all.

We have randomly chosen several samples and further analyzed them manually. After extracting the Python bytecode and reconstructing the original script, we have verified that some of the samples do not seem to contain harmful code. Some of them contain the same code as the sample labeled as Drobur, which appears to be an IRC bot. However, two of the samples look benign: one is a dot file editor and the other is a *Taxii Service Connector*. This makes us think that some of the samples detected by some antivirus engines and, therefore, marked as malicious in VirusTotal, use some kind of signature that flags any executable with Python embedded.

6.2. MIPS, ARM and PowerPC

We now look at the MIPS, ARM and PowerPC architectures together. Figure 6 shows the clusters for the different malware samples on the MIPS I architecture. Due to space constraints, we do not visualize clusters for ARM or PowerPC. However, unlabeled clusters are structured in a similar way on ARM and PowerPC to that on MIPS. That is, there are almost no singletons and most of the unlabeled samples are connected to a known family.

For MIPS, as discussed in Section 4, we can observe that Mrblack and Dnsamp are the same sample that has been labeled differently by the antivirus engines. For ARM, we can observe that most of the unlabeled samples are related to Mirai, and on PowerPC with Gafgyt and to a lesser extent with Mirai.

Finally, we have vetted most of the singletons shown on these three architectures. In general, these samples seem to belong to known malware families, corresponding to generic malware samples. However, we have also seen some samples piggybacking exploits that we have not seen before, similarly to what we saw on AMD x86-64 (see Section 6.1.3). We also found some malware piggybacking known hacking tools such as Aircrack[5] or a tool to clean fingerprints in a system.[6]

6.3. Intel 80386

We next focus on the Intel 80386 architecture. Although this case study resembles the one in the previous section (Section 6.2), the main difference here lies in the existence of a relatively large group that contains only unlabeled samples. Figure 7 shows the relationship between the unlabeled samples and known threats. It can be seen that several unlabeled samples are related mainly to the Setag family and to the Ganiw family. After further analysis, we observe that the Ganiw family is also named after Setag by some AVs. These findings enable us to re-label these new samples and approach the supervised learning phase in Section 4 with a better ground-truth quality.

When looking at the unknown samples alone, we first focus on the larger clusters. The largest cluster belongs to a family called Vit.4096, which is a virus that has been around

[5] Suite of tools to assess WiFi network security widely used to crack WiFi passwords (c.f., https://www.aircrack-ng.org/.)

[6] UNYUNZ Log Remove & Rename Utility.

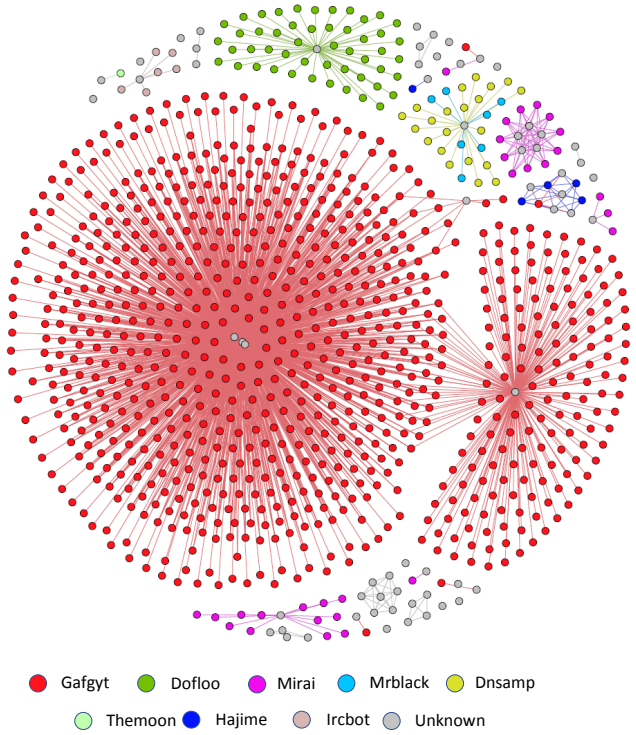

Figure 6: Clusters on the MIPS I architecture.

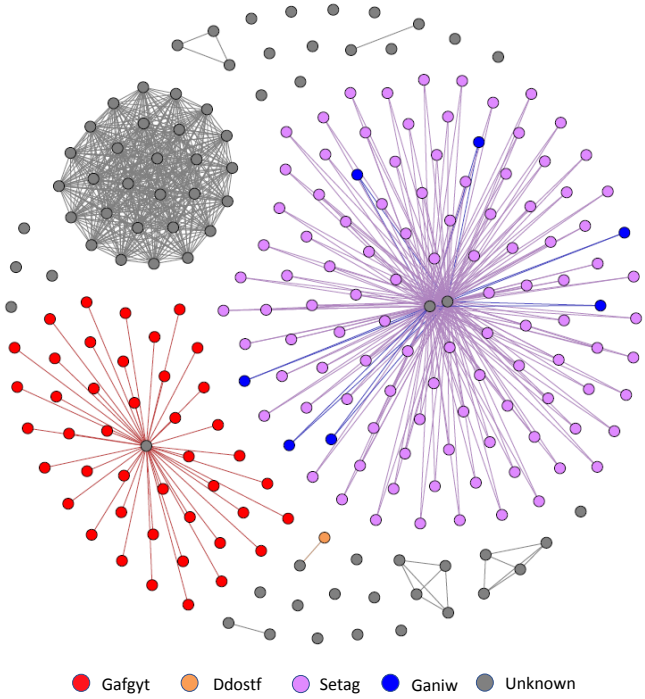

Figure 7: Clusters on the Intel 80386 architecture.

for over two decades. This virus replicates itself by adding 4096 bytes after the header of other ELFs in the system. The first reference to the virus dates back to 1996 [26]. However, signatures for this malware only appear in VirusTotal for the first time in 2017. In addition to this cluster, there are two other small clusters of 4 samples, each of which also belongs to this family. When reversing all these samples, we can confirm that they do indeed belong to the Vit family. We note that there is some small difference in the number of functions among the samples of different groups. The samples have a different cyclomatic complexity in some of their functions. Furthermore, there is a small difference in the overall number of functions among samples from different clusters. All the samples have a unique Indicator of Compromise (IoC) — the string "Vi324.tmp" appears across all the clusters and it is used as a file name for a temporary file during virus replication.

When looking at other clusters we find: i) a Bitcoin Wallet Bot; ii) the Rex family [3], which is malware with mining capabilities that propagates by exploiting wordpress, drupal, and magento vulnerabilities; iii) flooders; and v) more brute force attacks on SSH services as well as on telnet services.

Finally, when looking at the singletons we observe malware carrying several exploits and miscellaneous shellcode.

7. Discussion

Our work is based on the most representative dataset of Linux-based malware collected by the community to date [14]. In this section, we first discuss the main limitations of our work. Despite these limitations, our findings provide a fresh understanding of recent trends in Linux-based malware. In particular, we focus on characterizing a large number of samples that were unknown to the community. The set of non-characterized threats constitute 15% of the samples in [14]. The analysis performed in the discovery phase can help in the modeling of TTPs (Tactics, Techniques and Procedures), which is key in the attribution of malware. Matching different malware samples may identify different actors and their TTPs [15], which is very helpful in supporting incident response teams [53]. To the best of our knowledge, there is nothing published on malware attribution issues in IoT environments. Thus, we finally report key findings derived from vetting the most prevalent threats in the non-characterized set.

7.1. Threats to Validity

The proliferation of IoT malware is currently bound by the architecture of the different types of IoT devices. Based on this, we first describe how this affects the scope of our analysis and we then discuss the limitations of our system.

Scope. IoT malware targets a wide range of devices [2], each having a particular architecture. Well-known processor architectures are MIPS and ARM, but also AMD and Intel have recently been competing to provide energy-efficient chips [6, 39]. On top of the hardware, these devices are increasingly incorporating additional layers of software to deliver cross-platform controllers and/or operating systems. This is the case, for instance, of Arduino Yun[7], which runs Linux on top of an Arduino Leonardo. Another example is

[7] https://www.arduino.cc/en/Guide/ArduinoYun

Google Brillo[8], which has a scrubbed-down version of Android (also Linux-based). Currently, a wide range of the devices susceptible to being attacked by IoT malware run a flavour of Linux. Thus, architectures commonly used in desktop computers can also be targeted by the same malware and are part of this ecosystem. Likewise, malware developed for a desktop computer, could easily be re-compiled to run, for example, on top of a Linux-based ARM device. In our work, we approximate the analysis of the IoT malware landscape by focusing mainly on Linux-based malware. In particular, we look at this threat through the lens of a dataset collected by Cozzi et al. [14] in 2018. Although this dataset has been collected to evaluate a malware detection sandbox tailored to different IoT architectures, we note that it has been collected using protected malware Threat Intelligence. Thus, we acknowledge that there might be some multi-purpose malware samples that could also run on desktop computers. However, we also note that when the malware is collected through honeypots, this figure will be very small or even non-existent.

Limitations. IoT malware is an emerging threat and, therefore, does not yet have features as advanced as its predecessors on platforms such as Windows or Android. This leads to the use of both static characteristics and the use of disassembler n-gram sequences to perform correctly. In the near future, we expect to see an increase in the sophistication of IoT malware with the adoption of obfuscation or packing, thus hindering static analysis and limiting the visibility of our system. Another weakness derived from the use of static features is caused by binaries that are not necessarily malicious and that are embedded in the malware, such as those that are built with PyInstaller [52]. Tackling this challenge might require dedicated techniques similar to those proposed for dealing with repackaged malware in Android [51]. In our work, we partly overcome these limitations in two ways: i) we first use dynamic analysis to be able to characterize malware intended to hinder static analysis; and ii) we maintain a high similarity threshold to avoid clustering samples with different payloads but which piggyback the same library.

7.2. Key-findings

Our system builds a guided system to better characterize unknown threats in IoT malware. We next describe the main trends, focusing on those obtained from previously unknown threats.

Trends. We found that crypto-mining malware is targeting IoT platforms. This provides cybercriminals with an infrastructure connected 24 hours a day that allows them to mine cryptocurrencies in a distributed manner and obtain profits without investing in hardware or paying the electricity bill. Originally, the malware aimed at IoT platforms had the main purpose of creating a network of bots to perform denial of service attacks or to make money by selling access to a network of bots on the black market (Botnet-as-a-service) [48]. Nowadays, this is changing and cybercriminals are taking advantage of the resources of the infected devices. These stolen resources are currently used to mine cryptocurrencies and, therefore, obtain a greater economic benefit [42]. The study of the trends in malware targeting IoT devices could help develop forensic tools and methodologies tailored to such threats. This is because the characterization of IoT malware can lead to the discovery of evidence that can change the course of an investigation (i.e., increasing the percentage of CPU usage or battery consumption as evidence of infection by ransomware or crypto-mining malware [5]). Thus, the analysis performed can put forensic analysts in an advantaged position in the prosecution of criminals using the Trojan Horse Defense.

Sophistication. We have observed that the level of sophistication of IoT malware varies considerably. While there is certainly a large number of families with little sophistication, we are increasingly seeing more complex malware families (e.g., droppers and exploits, as reported in Sections 6.1.1 and 6.1.3, respectively). This is supported by recent commercial reports, such as the one in [33], which provide anecdotal evidence of this phenomenon.

Infrastructure. We also found that samples in a wide range of clusters rely on publicly available repositories such as exploit-db and GitHub (flooderscode) to manufacture new variants of malware. For example, the code of some of the most widespread botnets is available on the Internet [18], such as Mirai, Bashlite (Gafgyt), or pnscan. This encourages the development of novel samples that simply reuse part of the available code. This leads to the rapid proliferation of new variants with little investment and high economic returns.

8. Related work

Celeda et al. [11] present an analysis of the Chuck Norris malware, which they discovered due to the growth of port 23 scanners in their networks. The authors prepared a vulnerable device for the botnet and monitored all the network connections that were generated in that device for the purpose of an exhaustive analysis of the botnet.

Bohio [7] performs a technical analysis of the Dofloo malware, using static and dynamic code analysis. It details the commands supported by the malware as well as the communication mechanisms with the C&C. It also proposes a network traffic detection signature for Snort as well as commitment indicators to detect malware in a compromised system.

Wang et al. [56] distinguish two main infection methods for malware that is focused on IoT platforms. The authors use Mirai as an example of malware that uses brute force to infect devices, and Bashlite and Darlloz as examples to analyze malware that exploits vulnerabilities for its propagation.

In [30] the authors carry out a review of the state of the art of botnets, analyzing the techniques used by the Mirai and Hajime botnets. Finally, they propose countermeasures for the detection of these botnets.

In [4] a complete study on the Mirai botnet is presented in which the authors analyze its evolution, before and after the release of its source code. They describe the main devices

[8]Code-named Android Things https://developer.android.com/things

affected and the different types of attacks that the botnet carries out, as well as the targets which it is aimed at. Finally, as a case study, they analyze attacks on three Mirai victims.

Edwards et al. [19] perform a complete analysis of the Hajime botnet, detailing everything from the recognition and infection phase of new victims to the format of custom files used to store the configuration and payload. In addition, they describe the types of messages that Hajime supports. Finally, they provide countermeasures for detecting the botnet.

Herwig et al. [25] perform a study of the Hajime botnet from different points of view, such as the geographical locations of the infected devices and the types of devices. In their study, they also analyze the size of the botnet and the churn rates, as well as the TR-064 vulnerability that Hajime exploits by analyzing the queries collected from a DNS root server.

De Donno et al. [16] perform a study on the state of the art of malware in the IoT, presenting a classification of DDoS attacks according to different features such as the botnet architecture, protocol, scanning strategy, etc. They also present a description of the main families of botnets in the IoT and the relationships between them, showing the types of DDoS attacks that each one is capable of performing, the CPU architectures that support them and the DDoS architecture of the botnet. Finally, they present an analysis of the Mirai botnet.

In [14], the authors present the details of a platform they have developed that is focused on malware analysis based on Linux. Their platform supports the main target architectures of current IoT malware. In addition, they describe in detail different techniques used by the malware in Linux as well as the statistics of how many of the samples out of the total that comprised their study implement these techniques.

In [13], a study on malware in the IoT and its families is presented, summarizing the size of botnets as well as the estimated time they remain active. It also includes an analysis of the rules for IDS and the time window from when the malware appeared until the first rules appeared. In addition, they present a series of errors and inconsistencies found in their studies as well as the ambiguity of vulnerability references, analysis information, etc. Finally, they present a framework for analyzing malware dynamically that is based on the open source tool Cuckoo Box [22].

Cosa Nostra is a toolkit for clustering malware created by Joxean Koret [32]. The tool generates a call graph signature for each sample of malware analyzed using the complex cyclomatic of all the functions and assigning a prime number to that cyclomatic complexity. Finally, it generates a hash based on the multiplication of those prime numbers. When two hashes are equal it is considered that the samples are structurally equal, and if they are different, it breaks down the hash into its prime factors to determine how much they differ from each other. It allows the creation of phylogenetic trees of malware samples that are structurally similar. This can be visualized in a Web GUI.

Isawa et al. [29] propose the use of static features to compute the similarity between samples of IoT malware, since the extraction of static features consumes less time than obtaining characteristics based on their behavior. For their experiments, they build similarity matrices based on the use of n-grams for both disassembled code and system call traces. Finally, they visualize these matrices, verifying that the use of disassembled code works well for the classification of malware in the IoT.

Nguyen et al. [40] compare three approaches for malware detection using Convolutional Neural Networks (CNN). The first is based on fixed-size byte sequences, the second uses fixed-size color images, and the third assembler instruction sequences of variable size. Their tests are performed on 1000 samples of malware and 1000 goodware samples for the x86 architecture, showing by their experimental results that the approaches based on CNN work well in the detection of malware in the IoT.

9. Conclusions

In this paper we have presented a study of malware that targets IoT platforms. Through data analysis, we extracted static and dynamic features to systematically characterize malware into different threats. The proposed methodology allows the identification of new malware samples and the relationships they maintain with previous ones. Our evaluation over a dataset of labeled samples shows that our system can accurately perform this task.

Our methodology allows the extraction of knowledge about large groups of connected samples by analyzing some of their samples and extrapolating the results obtained. We have used this to investigate a number of unlabeled malware samples found in the wild. Where applicable, we have associated unknown clusters with known threats. This showed that the current detection mechanisms deployed by commercial AntiVirus systems are behind in the arms race. We have also gone one step further and studied each of the unknown clusters by using state-of-the-art reverse engineering techniques and our expertise as malware analysts. In this way, we were able to verify that the relationship formed by samples from the same group was correct, identifying the groups with the highest number of samples. We have also provided an in-depth analysis of what the most recent unknown trends are. We have shown, for instance, that crypto-mining malware is currently attacking the IoT infrastructure. Our hope is that this characterization will help the community to devise better detection strategies against these specific threats. Finally, in order to foster the development of these strategies, we have released the characterization we have produced for each of the clusters.

Acknowledgments

This work has been supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM under the project SB-PLY/17/-180501/000353 and by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105.

References

[1] Allodi, L., 2017. Economic factors of vulnerability trade and exploitation, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 1483–1499.

[2] Ammar, M., Russello, G., Crispo, B., 2018. Internet of things: A survey on the security of iot frameworks. Journal of Information Security and Applications 38, 8–27.

[3] Ancel, B., 2016. From website-locker to ddos: Rex! URL: https://thisissecurity.stormshield.com/2016/08/17/from-website-locker-to-ddos-rex/.

[4] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al., 2017. Understanding the mirai botnet, in: 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 1093–1110.

[5] Azmoodeh, A., Dehghantanha, A., Conti, M., Choo, K.K.R., 2018. Detecting crypto-ransomware in iot networks based on energy consumption footprint. Journal of Ambient Intelligence and Humanized Computing 9, 1141–1152. URL: https://doi.org/10.1007/s12652-017-0558-5, doi:10.1007/s12652-017-0558-5.

[6] Blem, E., Menon, J., Sankaralingam, K., 2013. Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures, in: 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), IEEE. pp. 1–12.

[7] Bohio, M., 2015. Analyzing a backdoor/bot for the mips platform. SANS Institute, Tech. Rep. .

[8] Bowles, S., Hernandez-Castro, J., 2015. The first 10 years of the trojan horse defence. Computer Fraud & Security 2015, 5–13.

[9] Breiman, L., 2001. Random forests. Machine learning 45, 5–32.

[10] Brumaghin, E., Williams, A., Zidouemba., A., 2018. Forgot about default accounts? no worries, goscanssh didnâĂŹt. URL: https://blog.talosintelligence.com/2018/03/goscanssh-analysis.html.

[11] Celeda, P., Krejci, R., Vykopal, J., Drasar, M., 2010. Embedded malware-an analysis of the chuck norris botnet, in: 2010 European Conference on Computer Network Defense, IEEE. pp. 3–10.

[12] Chen, Y.W., Lin, C.J., 2006. Combining SVMs with various feature selection strategies, in: Feature extraction. Springer, pp. 315–324.

[13] Costin, A., Zaddach, J., 2018. Iot malware: Comprehensive survey, analysis framework and case studies. BlackHat USA .

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

[15] Davis II, J.S., Boudreaux, B., Welburn, J.W., Aguirre, J., Ogletree, C., McGovern, G., Chase, M.S., 2017. Stateless Attribution. RAND Corporation.

[16] De Donno, M., Dragoni, N., Giaretta, A., Spognardi, A., 2018. Ddos-capable iot malwares: Comparative analysis and mirai investigation. Security and Communication Networks 2018.

[17] Details, C., . CVE-2016-5195. URL: https://www.cvedetails.com/cve/CVE-2016-5195/.

[18] Ding, F., . Iot-malware. URL: https://github.com/ifding/iot-malware.

[19] Edwards, S., ioannis Profetis, 2016. Hajime: Analysis of a decentralized internet worm fot IoT devices. Technical Report. URL: https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf.

[20] FireFart, 2016. Linux Kernel 2.6.22 < 3.9 - 'Dirty COW'. URL: https://www.exploit-db.com/exploits/40839.

[21] Gartner, 2015. Gartner says 6.4 billion connected., in: Retrieved February, 2019 from http://www.gartner.com/newsroom/id/3165317.

[22] Guarnieri, C., . Cuckoo Sandbox - Automated Malware Analysis. URL: https://cuckoosandbox.org/.

[23] Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J., Scholkopf, B., 1998. Support vector machines. IEEE Intelligent Systems and their Applications 13, 18–28. doi:10.1109/5254.708428.

[24] Hendler, D., Kels, S., Rubin, A., 2018. Detecting malicious powershell commands using deep neural networks, in: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ACM. pp. 187–197.

[25] Herwig, S., Harvey, K., Hughey, G., Roberts, R., Levin, D., 2019. Measurement and analysis of hajime, a peer-to-peer iot botnet. Network and Distributed System Security (NDSS) Symposium URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_02B-3_Herwig_paper.pdf, doi:10.14722/ndss.2019.23488.

[26] Hispasec, . Virus para linux. URL: https://unaaldia.hispasec.com/1999/02/virus-para-linux.html.

[27] Holzinger, A., Kieseberg, P., Weippl, E., Tjoa, A.M., 2018. Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI, in: Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E. (Eds.), Machine Learning and Knowledge Extraction, Springer International Publishing, Cham. pp. 1–8. doi:10.1007/978-3-319-99740-7_1.

[28] Hurier, M., Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer. pp. 142–162.

[29] Isawa, R., Ban, T., Tie, Y., Yoshioka, K., Inoue, D., 2018. Evaluating disassembly-code based similarity between iot malware samples, in: 2018 13th Asia Joint Conference on Information Security (AsiaJCIS), pp. 89–94. doi:10.1109/AsiaJCIS.2018.00023.

[30] Kambourakis, G., Kolias, C., Stavrou, A., 2017. The mirai botnet and the iot zombie armies, in: MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), pp. 267–272. doi:10.1109/MILCOM.2017.8170867.

[31] Kolias, C., Kambourakis, G., Stavrou, A., Voas, J., 2017. Ddos in the iot: Mirai and other botnets. Computer 50, 80–84.

[32] Koret, J., . Cosa nostra. URL: https://github.com/joxeankoret/cosa-nostra.

[33] Kroustek, J., Iliushin, V., Shirokova, A., Neduchal, J., Hron, M., . Torii botnet - not another mirai variant. URL: https://blog.avast.com/new-torii-botnet-threat-research.

[34] Laaksonen, J., Oja, E., 1996. Classification with learning k-nearest neighbors, in: Proceedings of International Conference on Neural Networks (ICNN'96), pp. 1480–1483 vol.3. doi:10.1109/ICNN.1996.549118.

[35] Leskovec, J., Rajaraman, A., Ullman, J.D., 2014. Mining of massive datasets. Cambridge university press.

[36] Maaten, L.v.d., Hinton, G., 2008. Visualizing Data using t-SNE. Journal of Machine Learning Research 9, 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[37] Marturana, F., Tacconi, S., 2013. A machine learning-based triage methodology for automated categorization of digital media. Digital Investigation 10, 193–204.

[38] Mikhail Kuzin, Yaroslav Shmelev, V.K., 2018. New trends in the world of iot threats. Available Online.

[39] Nakhkash, M.R., Gia, T.N., Azimi, I., Anzanpour, A., Rahmani, A.M., Liljeberg, P., 2019. Analysis of performance and energy consumption of wearable devices and mobile gateways in iot applications.

[40] Nguyen, K.D.T., Tuan, T.M., Le, S.H., Viet, A.P., Ogawa, M., Le Minh, N., 2018. Comparison of three deep learning-based approaches for iot malware detection, in: 2018 10th International Conference on Knowledge and Systems Engineering (KSE), IEEE. pp. 382–388.

[41] Pastrana, S., Suarez-Tangil, G., 2019. A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. arXiv preprint arXiv:1901.00846 .

[42] Pastrana, S., Thomas, D.R., Hutchings, A., Clayton, R., 2018. Crimebb: Enabling cybercrime research on underground forums at scale, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, International World Wide Web Conferences Steering Committee. pp. 1845–1854.

[43] 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., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

[44] Quick, D., Choo, K.K.R., 2014. Impacts of increasing volume of digital forensic data: A survey and future research challenges. Digital Investigation 11, 273–294.
[45] Quinlan, J.R., 2014. C4.5: Programs for Machine Learning. Elsevier. Google-Books-ID: b3ujBQAAQBAJ.
[46] Recabarren, R., Carbunar, B., 2017. Hardening stratum, the bitcoin pool mining protocol. Proceedings on Privacy Enhancing Technologies 2017, 57–74.
[47] ring04h, . ring04h/WebTerror. URL: https://github.com/ring04h/WebTerror.
[48] Seals, T., . Themoon rises again, with a botnet-as-a-service threat. URL: https://threatpost.com/themoon-botnet-as-a-service/141393/.
[49] Sebastián, M., Rivera, R., Kotzias, P., Caballero, J., 2016. Avclass: A tool for massive malware labeling, in: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (Eds.), Research in Attacks, Intrusions, and Defenses, Springer International Publishing, Cham. pp. 230–253.
[50] Stone, M., 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. Journal of the Royal Statistical Society. Series B (Methodological) 36, 111–147. URL: https://www.jstor.org/stable/2984809.
[51] Suarez-Tangil, G., Stringhini, G., 2018. Eight years of rider measurement in the android malware ecosystem: evolution and lessons learned. arXiv preprint arXiv:1801.08115 .
[52] Team, P.D., . Pyinstaller quickstart – pyinstaller bundles python applications. URL: https://www.pyinstaller.org/.
[53] Tounsi, W., Rais, H., 2018. A survey on technical threat intelligence in the age of sophisticated cyber attacks. Computers & security 72, 212–233.
[54] Van Wegberg, R., Tajalizadehkhoob, S., Soska, K., Akyazi, U., Ganan, C.H., Klievink, B., Christin, N., Van Eeten, M., 2018. Plug and prey? measuring the commoditization of cybercrime via online anonymous markets, in: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1009–1026.
[55] Virus Total, . https://www.virustotal.com. [Online; accessed February-2019].
[56] Wang, A., Liang, R., Liu, X., Zhang, Y., Chen, K., Li, J., 2017. An inside look at iot malware, in: International Conference on Industrial IoT Technologies and Applications, Springer. pp. 176–186.
[57] Watson, A.H., Wallace, D.R., McCabe, T.J., 1996. Structured testing: A testing methodology using the cyclomatic complexity metric. volume 500. US Department of Commerce, Technology Administration, National Institute of âĂę.
[58] Witten, I.H., Frank, E., Hall, M.A., 2011. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., Burlington, MA.

Javier Carrillo Mondéjar received a BSc degree in Computer Science and a Master's degree in Advanced Computer Science from the University of Castilla-La Mancha, Spain, in 2016 and 2017, respectively. Currently, he is enrolled full-time on the PhD Program in Advanced Information Technology at this university. In 2016, he joined the Computer Architecture and Technology Group of the Informatics Research Institute of Albacete (I3A) as a researcher. His research interests are related to malware detection and classification techniques, as well as the methods used in malware to spread and remain hidden in computer systems. He has also been a visiting researcher at King's College London.

José Luis Martinez received his M.Sc. and Ph.D. degrees in Computer Science and Engineering from the University of Castilla-La Mancha (Spain) in 2007 and 2009, respectively. In 2005, he joined the Department of Computer Engineering at the University of Castilla-La Mancha, where he was a researcher in the Computer Architecture and Technology group at the Albacete Research Institute of Informatics (I3A). In 2010, he joined the department of Computer Architecture at the Complutense University in Madrid, where he was an assistant lecturer. In 2011, he rejoined the Department of Computer Engineering of the University of Castilla-La Mancha, where he is currently a lecturer. His research interests include video coding and transcoding, and topics related to security. He has also been a visiting researcher at the Florida Atlantic University, Boca Raton (USA) and the Centre for Communication System Research (CCSR), at the University of Surrey, Guildford (UK). He has over 100 publications in these areas in international refereed journals and conference proceedings.

Guillermo Suarez-Tangil is a lecturer at King's College London (KCL). His research focuses on systems security and malware analysis and detection. In particular, his area of expertise lies in the study of smart malware, ranging from the detection of advanced obfuscated malware to automated analysis of targeted malware. Before joining KCL, he was senior research associate at University College London (UCL), where he was also actively involved in other research areas involved with detecting and preventing Mass-Marketing Fraud.