Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification

Duy-Phuc Pham, Damien Marion, Mathieu Mastio, Annelie Heuser

Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification

Duy-Phuc Pham
duy-phuc.pham@irisa.fr
Univ Rennes, CNRS, Inria, IRISA
Rennes, France

Damien Marion
damien.marion@irisa.fr
Univ Rennes, CNRS, Inria, IRISA
Rennes, France

Mathieu Mastio
matthieu.mastio@irisa.fr
Univ Rennes, CNRS, Inria, IRISA
Rennes, France

Annelie Heuser
annelie.heuser@irisa.fr
Univ Rennes, CNRS, Inria, IRISA
Rennes, France

ABSTRACT

The Internet of Things (IoT) is constituted of devices that are exponentially growing in number and in complexity. They use numerous customized firmware and hardware, without taking into consideration security issues, which make them a target for cybercriminals, especially malware authors.

We will present a novel approach of using side channel information to identify the kinds of threats that are targeting the device. Using our approach, a malware analyst is able to obtain precise knowledge about malware type and identity, even in the presence of obfuscation techniques which may prevent static or symbolic binary analysis. We recorded 100,000 measurement traces from an IoT device infected by various in-the-wild malware samples and realistic benign activity. Our method does not require any modification on the target device. Thus, it can be deployed independently from the resources available without any overhead. Moreover, our approach has the advantage that it can hardly be detected and evaded by the malware authors. In our experiments, we were able to predict three generic malware types (and one benign class) with an accuracy of 99.82%. Even more, our results show that we are able to classify altered malware samples with unseen obfuscation techniques during the training phase, and to determine what kind of obfuscations were applied to the binary, which makes our approach particularly useful for malware analysts.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

malware classification, obfuscation, side-channel information

ACSAC '21, December 6–10, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8579-4/21/12…$15.00
https://doi.org/10.1145/3485832.3485894

ACM Reference Format:
Duy-Phuc Pham, Damien Marion, Mathieu Mastio, and Annelie Heuser. 2021. Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification. In *Annual Computer Security Applications Conference (ACSAC '21), December 6–10, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3485832.3485894

1 INTRODUCTION

IoT world is growing at a breathtaking pace, from 2 billion objects in 2006 to a projected 200 billion by the end of 2020[1], which is approximately 26 smart objects for every human being on Earth. Naturally, they are increasingly targeted by cyber criminals due to their occurrences, availability, and the ability to use infected devices for further attacks on their architecture.

IoT devices are given higher processing power, and some of them are running fully functional operating systems (OS) with multi-core processors. This increases the attack surface by making them vulnerable to similar threats as general purpose computers, in particular, malware exploitation.

Analysis systems relying on static and dynamic features still have various drawbacks for malware analysts. In particular, static features can be easily manipulated by packing or obfuscating techniques [26], whereas dynamic software-based monitoring may be detectable (*e.g.* by sandbox fingerprinting [37]) to terminate the malware execution, and thus hinder the possibility of behavioral analysis [1]. Moreover, unlike computer systems and servers, embedded cyber physical system may not have enough resources or accessibility to allocate to malware analysis solutions. All these factors make it difficult for malware analysts to automatically gain proper information about collected malware samples (*i.e.* nature, evolution, etc.) to be able to mitigate the security risks.

In this paper, we concentrate on the ElectroMagnetic (EM) field of an embedded device as a source for malware analysis, which offers several advantages. In fact, EM emanation that is measured from the device is practically undetectable by the malware. Therefore, malware evasion techniques cannot be straightforwardly applied unlike for dynamic software monitoring. Also, since a malware does not have control on outside hardware-level events (*e.g.* on EM

[1] https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html

emanation, heat dissipation), a protection system relying on hardware features cannot be taken down, even if the malware owns the maximum privilege on the machine. Therefore, with EM emanation it becomes possible to detect stealthy malware (*e.g.* kernel-level rootkits), which are able to prevent software-based analysis methods. Another advantage is that monitoring EM emanation does not require the alternation of the device in order to analyze it. In other words, it does not rely on device architecture and OS or without any computational overhead.

Previous works using EM emanation and power consumption investigated the detection of malware [2, 20, 31], abnormal behaviour [36], or distinct control flow tracking [38]). These works are in very constrained and static systems (in the case of anomaly detection) and mostly analyzed only laboratory-grown malware samples without any variations. This naturally raises the question of realistic evaluation:

What happens if the executable from embedded devices is performing malicious behaviors under obfuscation?

While malware detection concerns with the process of detecting the presence of malware whether a specific program is malicious or benign [18], malware classification refers to the process of distinguishing the unique types of malware from each other based on the identified malicious patterns. In this work, we derive a framework that is capable to *classify real-world* malware samples including protection mechanisms against static and dynamic malware analysis using only EM emanation from the device. Furthermore, we aim at classifying into malware types, family, possible protection mechanism, previously unseen variants, or even distinct executable classification, which makes our framework particularly suited for malware analysts.

Our analysis consists of three main phases:

(1) *Malware execution and measuring EM emanation*: Within our secured and randomized setup, we execute the malware sample while measuring EM emanation from the outside of the device without manipulating any internal behavior.
(2) *Data analysis and preprocessing*: The raw captured measurement traces include a large amount of noise. Thus, we analyse our data by the time-frequency domain, and select most suitable frequency bands.
(3) *Malware classification*: Given the 2D data, we derive deep neural models and compare them to more simplistic machine learning algorithms.

Our malware selection encompasses three types, which are accurately representing malware targeted on IoT devices in the wild: DDoS, ransomware, and kernel rootkits. To be compliant with real-world scenarios and to investigate the robustness of our approach, we extend our dataset by applying various software analysis protection mechanisms to the malware binaries. Including obfuscation techniques gives us new outcomes that have never been studied in the state-of-the-art. First, we determine if code obfuscation techniques (*e.g.* code rewriting, camouflage instructions, ...) can actually hinder our approach. Second, we derive the robustness of our approach against unseen malware samples, by creating a scenario where our system tries to predict samples with unknown obfuscation. This evaluation is of great importance due to the rapid evolution of malware variations and obfuscation created by attackers. Finally, we investigate if we are able to predict if an obfuscation has been applied, and to which technique it belongs.

In summary, this paper makes the primary contributions:

(1) **Obfuscated ARM malware.** We put in place a representative set of malicious ARM binaries, on which we applied various obfuscation techniques. By integrating obfuscation techniques against software-based malware analysis systems, we are able to determine if these techniques also hinder analysis based on EM emanation, and if we can distinguish the applied obfuscation procedures independent on the executed binary. To the best of our knowledge this has never been studied before, provides the largest distinct malware sample dataset, and is crucial for practical malware analysis.
(2) **Generic side-channel analysis environment.** Our approach does not make any alteration to the target device. In particular, we do not require software monitoring, precise triggering, or any additional overhead on the device. In our experiments, we use a multiprocessor hardware environment running a fully-functioning Linux OS to be applicable to realistic IoT systems in the wild, use a random initialized analysis environment, and "complex" benign activities.
(3) **Robust and resistant analysis techniques.** We derived a methodology on how to effectively extract suitable information about the binary, taking as input the raw EM traces. Our approach consists of preprocessing by selecting the most relevant frequency bands over time and then classifying in various scenarios with neural network models and simplistic machine learning models. Results show that our methodology is resistant against virtualization, packing, and static code rewriting.
(4) **Experimental scenarios compliant to malware analysts.** We compile various scenarios, each of them representing a real world malware analysis use case: type and family malware classification, exact malware executable profiling, virtualization and packer identification, obfuscation classification, and the classification of *unseen* obfuscated variants. These scenarios go way beyond detection scenarios considered in the state-of-the-art. Also, using our analysis on obfuscation we are the first to discuss the difficulties of malware evasion against our methodology.
(5) **Open-source.** The resources related to this work are publicly available at https://github.com/ahma-hub/analysis/wiki, where we provide our source code, datasets, malware classification models and raw results of our experiments.

2 STATE-OF-THE-ART

One of the first works on malware detection [7], even though limited due to its constrained scenario, showed that the collection of power consumption on medical embedded devices is suitable to detect malware. [20] presents a malware detection solution by exploiting EM side-channel signals from embedded devices through Multi-Layer Perceptron (MLP) to detect handcrafted implementations mimicking malware. A common idea to take advantage of side channel information to detect anomalies is to observe how the system behaves in its normal state, and to raise an alert when a new behavior

Article	SCM detection	Anomaly detection	SCM classification	Real-world SCM	Real-world analysis environment	Samples size	Variations	Benign dataset	Window size	Open source	Device under test
WattsUpDoc [7]	✓	-	-	✓	-	15	-	-	5s	-	Windows XP Embedded 664 MHz
IDEA [19]	-	✓	-	-	-	3	-	-	<40μs	-	AT328p 16MHz, Cortex A8
REMOTE [33]	-	✓	-	✓	-	3	-	-	<10ms	-	Single-core ARM 1Ghz
Wang *et al.* [36]	-	✓	-	-	-	1	-	-	10s	-	Raspberry Pi, Arduino, Siemens PLC
Khan *et al.* [20]	✓	-	-	-	-	3	-	-	<150μs	-	Cyclone II FPGA & NIOS II soft-processor
DeepPower [12]	✓	-	✓	✓	-	5	-	-	1s	-	MIPS/ARM OpenWRT
Chawla *et al.* [6]	✓	-	✓	✓	-	137	-	✓	10s	-	Android Intrinsyc Open-Q 820
Our paper	*(✓)**	-	✓	✓	✓	35	✓	✓	2.5s	✓	Multi-core, 900 Mhz ARM

Table 1: Comparison with related works on side-channel malware (SCM) analysis using EM or power consumption. *(*)*: Our paper aims at SCM classification, however we also achieve good results in SCM detection scenario (Section 7.2).

is recorded. In [19, 33], the authors propose to detect malware by observing EM signals. During the monitoring, if the observed EM emanations deviate from the previously observed patterns, this is reported as an anomalous or malicious activity. [24] uses Short Time Fourier Transform (STFT) and Kolmogorov–Smirnov test to detect anomalies inside and between the loops through peaks in the EM spectrum. In [31], the authors put a wide-bandwidth radio frequency probe over the processor of the device and used a support vector machine to infer on the values of the registers. They monitor if the hamming distance of the registers deviates from the known signature, and use this information to detect cyberattacks. In [36], they use Autoencoders, Long Short-Term Memory (LSTM) units, and MLP on power consumption data to detect anomalies in the system. [12] shows an approach to detect malicious activities on IoT devices via analyzing power side-channel signals using Convolution Neural Networks (CNN). The use of physical hardware information, and particularly side-channel information, represents a great advance for malware detection. A comparison of the works using EM or power consumption to analyse malicious activities is provided in Table 1.

While some of the above-mentioned related works are successfully detecting malicious activity, there is a lack of research in the field on in-the-wild malware detection instead of proof-of-concept samples that may reflect only particular parts of realistic malware samples. Even more, none of the related works investigated the scenario of benign datasets and variants such as packed or obfuscated malware to test the robustness of their system. Except [6] considered CPU benchmark applications for Android benign dataset, however, it is very specific stress processes that are easy to detect and classify rather than a wide-range dataset of cleanware, long-running programs and device background services. Moreover, most of these works are using anomaly detection with low sample size, which has the advantage of detecting unknown threats, but is generally prone to raise a large number of false positives. Indeed, anytime a new feature is introduced to the system, it is detected as malicious. They only exploit an isolated malware execution environment (*e.g.* disabled outside connections), or an undefined malware execution environment, thus prone to evasion techniques and unclear if the malware actually executes malicious behaviors. Finally, none of them, to the best of our knowledge, are able to perform wide-ranging classification models in real-world malware analysis, i.e., determining precisely the type, obfuscation or variant of the malware infecting the system, due to their restricted malware dataset or analysis methods.

3 BACKGROUND ON DATA ANALYSIS

3.1 Feature extraction and transformation

Extracting features within large measurement traces can be a challenging step. In the field of physical side-channel analysis of cryptographic algorithms, several methods have been published relying on statistical measures such as mean and variance, for example, NICV [5], SOST/SOSD [14], the Pearson correlation coefficient [15, 22], TVLA [32]. In our methodology we will rely on NICV as it is straightforward to implement, time efficient, and not model-agnostic (contrary to the TVLA). NICV is defined as $\mathrm{NICV}(X, Y) = \frac{\mathrm{Var}[\mathbb{E}[X|Y]]}{\mathrm{Var}[X]}$ with X being the recorded data, Y being the labels and Var (resp. $\mathbb{E}$) the variance (resp. the expectation).

A popular supervised feature transformation algorithm is the linear discriminant algorithm (LDA) that finds a linear combination of features separating two or more classes [16]. LDA explicitly tries to model the difference between the classes of data which makes it a suitable preprocessing algorithm in case of large data. Note that the features are transformed into another feature space such that original dependencies (shapes, patterns) between feature may be lost.

3.2 Machine and deep learning models

The Naive Bayes (NB) classifier is based on applying Bayes' theorem with a strong (naive) independence assumption between the features. It is further based on a Gaussian distribution assumption, which is most often not given in practise, but has still shown comparable performances in the physical side-channel domain when revealing secret keys [28]. The strong benefit of NB is its low resource requirement, fast computation power, and no requirement of tunable parameters.

Another popular machine learning classification algorithm is Support Vector Machine (SVM) [16]. The principle behind the SVM

is finding hyperplanes that maximize the features' separation in classes. Using a kernel trick and transforming features into a higher-dimensional feature space (*e.g.* by using the Gaussian radial basis function), SVM is able to perform nonlinear classification. SVM is slightly more resource demanding than NB which comes typically with an increase of classification accuracy [28].

A generic neural network architecture widely used for a variety of purposes is a MLP. It tends to learn very specific features of the training data and does not always generalize well on new data. On the other hand, CNN has been developed specifically for image processing. They learn high level features of the images instead of focusing on low level data like MLPs, which makes them more flexible and less prone to overfit to the training dataset. We will concentrate in this work on these two architectures.

4 (OBFUSCATED) MALWARE & BENIGN

One of the most important blocks in building malware analysis systems, is the construction of datasets. In this paper, we aim at being realistic and to include common obfuscation techniques that are used by today's malware designers to avoid detection by hiding known signature and behavior or by making it more difficult to reverse engineer. While general purpose computers usually run on common architectures such as x86, embedded IoT devices are developed for a broad range of architectures, such as ARM, MIPS, PowerPC, etc. Major problems related to the diversity of the possible target IoT environments are described in [11]. This paper, at the time of writing, currently supports ELF ARM 32-bit architecture which can be executed properly on Raspberry Pi (see Section 6.1). In the following, we discuss the creation of our malicious and benign dataset.

4.1 Malware dataset

To understand the scope of ARM malware on IoT devices, we conduct a study on 4,790 32-bit ELF ARM malware samples collected from Virusign. Thereafter, we extract AV labels for each sample from VirusTotal reports to obtain malware variant name. To get normalized labels that can be used for classification, we use AVClass. It selects the top ranked corresponding family name through plurality vote. On collected dataset, AVClass was able to associate the collected malicious ARM samples to 19 different families. *Mirai* (43.5%) and *Bashlite* (35.8%) dominate the dataset. This result is consistent with previous epidemiologic studies of IoT malware [11, 23]. To construct a representative malware dataset, we use 3 different well-known malware variants: DDoS (*mirai*, *bashlite*), Ransomware (*gonnacry*), and kernel rootkits (*keysniffer*, *maK_It*). In our study, we reviewed their codebase to understand their malicious behaviors described as follows:

Bashlite creates a TCP communication to C&C server, then exchanges IRC commands and messages. Control commands and common behaviours of *bashlite* consist of scanner, password brute-force, TCP and UDP Flooding.

Mirai adopted concepts from previously discussed *bashlite*, with improved features such as anti-debugging, self-hidden, data obfuscation and botkiller which terminates bots from other families.

Gonnacry is an active ransomware variant that is open sourced in Python and C for research purpose. It finds all files in user's home directory, then encrypt those with matching extensions. The malware starts its encryption routine and creates a desktop file that will be useful for the decryptor to access the path, key and IV. We generate multiple malware variants from original *gonnacry* by extending with other crypto schemes such as AES and DES, in addition to the original Blowfish encryption algorithm.

Keysniffer is a Linux kernel module which has functionalities to hook and record keys pressed in the keyboard events to debugfs.

MaK_It shares the same rootkit ability to *keysniffer*, with addition of kernel module self-hidden, packet sniffer and reverse-shell backdoor.

4.2 Obfuscation

Malicious codes commonly use packers, obfuscators, and polymorphism to hinder static-analysis and evade detection by making analyses difficult to reverse-engineer. Collberg *et.al* [10] defines obfuscating transformation as follows: Let $P \xrightarrow{\mathrm{T}} P'$ be a transformation of a source program P into a target program P'. $P \xrightarrow{\mathrm{T}} P'$ is an *obfuscating transformation* if

(1) P and P' have the same observable behavior,
(2) P' is harder to analyze than P, and
(3) P' is no more than polynomially slower than P.

More precisely, in order for $P \xrightarrow{\mathrm{T}} P'$ to be a valid obfuscating transformation, the following conditions must hold: if P fails to terminate or terminates with an error condition, then P' may or may not terminate. Otherwise, P' must terminate and produce the same output as P.

Previous research classifies code obfuscation schemes into three main categories: data obfuscation, static code rewriting, and dynamic code rewriting. We use combinations of obfuscation transformations to enrich our datasets with static code rewriting that consists of *Opaque predicates*, *Bogus control flow*, *Instructions substitution* and *Control-flow flattening*, and dynamic code rewriting such as *Packer* and code *Virtualization*. To evaluate the robustness of our methodology and to explore possible protection techniques against side-channel monitoring, we apply state-of-the-art packers and obfuscators like UPX [25], Tigress [9], and Obfuscator-LLVM [17].

4.3 Benign dataset

The selection of a benign dataset is important to not only increase the difficulty of detection but also ensure the quality of classification. The benign samples must generate random activities such as computations, background processes with malware-free, or usual activities on embedded IoT devices. We generate benign datasets by collecting ARM executables from a fresh installation of Linux system. This similar approach of constructing benign dataset has been conducted from other generic malware studies [3, 21] outside from EM analysis. Furthermore, we complement benign executables under a layer of UPX packer to blend benign samples with packer. Additionally, the usual benign activities for an embedded IoT device such as Linux utilities, device sleep, photo capture, network connections, as well as long duration of executable runtime such as media player, camera capture, video encoder, data backup, data (de)compression (Table 2). This collection varies from short

Activities	Executables			
Linux Utilities	mknod	vdir	more	find
	zgrep	ls	cat	findmnt
	zmore	as	ed	rm
	touch	dmesg	sleep	cd
	less	grep	objdump	
Network	wget	hostname	ss	ip
Compression	gunzip	bunzip2	bzip2	tar
	uncompress			
Data backup	dd			
Scripting	python			
Photo & Video	raspistill	raspivid		
Video Encoding	MP4Box			
Audio player	mpg321			

Table 2: Linux binaries and activities used in benign dataset

to long duration of executable runtime, and from low to high CPU consuming processes.

Notably in previous studies using EM emanation, the construction of benign dataset is not considered, or benign activity is only associated with either free-malware activities or benchmark software [6, 19, 24, 33, 35]. It simplifies detection drastically and is not realistic where malware, update services as well as IoT activities may share the same behaviors by calling executables from system and third parties.

5 REAL-WORLD MALWARE ANALYSIS FRAMEWORK

We propose a malware classification framework that takes as an input an executable and outputs its predicted label by solely relying on EM side-channel information. Figure 1 illustrates our workflow, which will be detailed within the next subsections. First, we define our threat model and we collect EM emanation while the malware is executed on the target device. We setup an infrastructure to be able to execute malware with a realistic user environment while preventing any infection of our host controller system. Thereafter, as the collected data is very noisy, a preprocessing step is required to isolate relevant informative signals. Finally, using this output, we train neural network models and machine learning algorithms in order to classify malware types, binaries, obfuscation methods, and detect if an executable is packed or not.

5.1 Threat model

In general, malware analysts gather sets of malware from online feeds of intrusion detection systems and community database. In this threat model, malware analysts possessed real-world malware sets and physical target devices. Real-world malware feeds presumably contain unknown variants which exploit evasion techniques and attack a wide range of Linux device (*e.g. Mirai* variants actively infect Linux IoT devices and obfuscate its encoded strings). By leveraging the combination of bare-metal analysis and EM (Fig. 1), it avoids the necessity of software analysis tools update such as sandbox, hooking and anti evasion techniques. Moreover, malware analysts are fully able to control and customize their analysis environment in the most advantageous way by simulating network

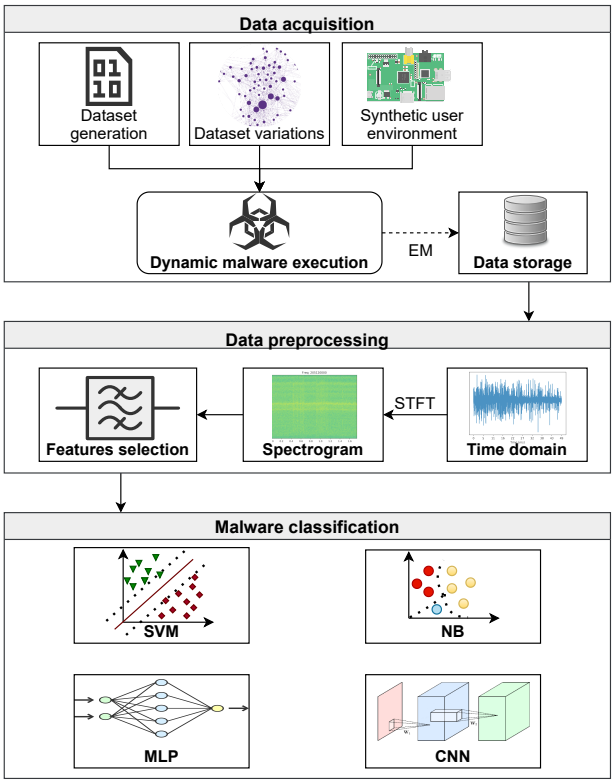


Figure 1: Illustration of our complete malware classification framework. *(i)* Data acquisition: from malicious binary execution to (noisy) EM measurements; *(ii)* Data preprocessing: from (noisy) EM measurements to exploitable data; *(iii)* Malware classification: from exploitable data to malware labels.

and setting up user synthetic environment. Therefore simulating side-channel noise, distant EM monitoring, random user activities, multiple malware attack the same device simultaneously are not their consideration. The proposed analysis framework supports a generic OS, so that it is applicable to any kind of malware with UNIX, from simple BusyBox utilities and Bash commands to ELF malware and high-level scripting (*e.g.* Python). Unlike FPGA-based systems malware detection approach that executes the samples on restricted bare-metal environment.

5.2 Data acquisition by dynamic malware execution

The first part of our framework relates to the data acquisition that can be divided into *dynamic malware execution* and *electromagnetic monitoring*.

5.2.1 Realistic malware execution environment. Traditional dynamic malware analysis solutions were built upon virtualization machines or emulation, which leave a large number of system artifacts for evasive malware to exploit [30]. In particular, sophisticated malware authors exploit fingerprints inside analysis system (*e.g.* number of cores, network MAC address, etc.) to avoid malware analysis or detection. Besides, in-guest monitoring components to observe

malware behaviors (*e.g.* syscall/API hooking, registry monitor, etc.) also leave artifacts for malware evasion. One way to prevent these artifacts is to patch the exploitable components of the virtual system such that these are indistinguishable from a real machine. However, this approach only guarantees to known evasion techniques. Another way is to implement a transparent analysis system that performs hardware virtualization extensions, to keep the CPU execution semantics of the host. It is fundamentally infeasible to make it perfectly transparent, since this system can be detected by timing attacks and CPU identification.

To overcome these difficulties, we propose an infrastructure leveraging side-channel information from a bare-metal sandbox rather than emulators or virtual machines. Additionally, an unrealistic configuration will not be able to trigger malware activities, so that we propose spoofed C&C servers which receive network connections and randomly returns control commands as well as a synthetic user environment dedicated for embedded malware which will be detailed in Section 6.1.2. We have confidence that bare-metal malware analysis does not expose any instrumentation indicators, and side-channel information will give us a snapshot of behavioral analysis. To prevent analysis information leakage or infection to the analyst's host machine, a local switch router and firewall under a controlled network are needed.

5.2.2 Electromagnetic monitoring. In the data acquisition procedure, the controller machine sends binary samples to the target device. The controller server is responsible for distributing samples to one or more embedded devices as well as collect recorded EM traces. We use a *malware initiator* to send trigger signals from the target device to oscilloscope through GPIO pin-outs and let the oscilloscope start its recording session. Note that the intended users are malware analysts who are unrestricted to set-up malware initiators or choosing the appropriate device. The *malware initiator* is only executed at the beginning of the analysis to trigger the oscilloscope once while malware/benign samples as well as the OS are kept untouched. This monitor scheme also means that our approach does not require data synchronization which is common in side channel analysis.

5.3 Data analysis and preprocessing

The second step in our complete framework preprocesses the collected (noisy) EM measurements (see Figure 1). This step is mandatory as the CPU of an electronic device executes programmed instructions every clock cycle, that will provoke variations in its internal circuitry. Moreover, modern target devices have multi-core architectures, thus the recorded EM activity is a mixture of various processes and it is impractical to identify correctly the process responsible for each observed variations from the electromagnetic trace itself. The strong signals existing in the system, like processor or memory clock, will act as a carrier, that will be amplitude or frequency modulated by the executed instructions [29]. This modulation will cause EM emanation, that leaks from any elements of the device.

It has been shown [34] that it is possible to monitor the EM spectrum to profile a program execution on the system. Each executed program has a specific loop pattern, that is revealed by peaks in frequency. This is why we preprocess the raw EM data to represent the fluctuations of the frequency content during the measurement time of the traces. For this purpose, we computed the spectrogram of the signal by taking the magnitude squared of the STFT: $spectro\{x(n)\}(m,\omega) = |\sum_{n=0}^{N} x(n)w(n-m)e^{-j\omega n}|^2$. A STFT breaks the signal into small segments of equal length, and performs a Fourier transform on each of the segments. Here, $x(n)$ represents the input signal at time n, ω the frequency, m the segment index, N the number of recording points, and w the window function. In our case, the window function splits the signal in chunks of length M, with an overlap O.

Even though the spectrogram is improving our data representation in terms of noise reduction, it also highly increases the amount of data. Using the full spectrogram will drastically increase the amount of time and space resources needed for classification, if even possible. We therefore apply feature extraction on the spectrogram using NICV (see Section 3.1). In particular, we apply the NICV on the spectrograms in order to identify the frequency bandwidths that may real behavioral information about the binary.

Let us denote X as a spectrogram of dimension $D \times F$ with D being the number of time features and F being the number of frequency bandwidths. Let Y be the label, *e.g.* the type of the malware. The computation of $\mathrm{NICV}(X, Y)$ results in a matrix of dimension $D \times F$ (see Eq. (3.1)). Next, from the NICV matrix, we select the ϵ frequency bands corresponding to the highest mean over D:

$$F_{\text{extract}} = \{\text{argmax}_{\epsilon}(\frac{1}{D}\sum_{d=0}^{D-1}[(\mathrm{NICV}(X,Y))_d^F]\} \quad (1)$$

with argmax_{ϵ} being a function that returns the ϵ indexes with the highest values and $(\cdot)_d^F$ represents the dth column of the matrix over all frequencies. Accordingly, F_{extract} contains the list of the ϵ indexes of the conserved frequencies. Note that, we extract the complete frequency band of the spectrogram instead of multiple chunks, which is mainly motivated by possible time delays or desynchronizations of unseen data in feature extraction process due to the absence of an exact triggering process.

5.4 Malware classification

Given the most informative spectrogram bands, our main objectives are to analyse to which extend a malware analyst is able to: (*i*) retrieve the type or family of the malicious binary, (*ii*) identify precisely which binary was being executed, (*iii*) classify the obfuscation technique, and (*iv*) classify the malware family even with an previously unknown obfuscation technique. Based on that, we assume that the analyst has a dataset of labeled malware binaries on which he can build supervised classification models.

Neural networks are particularly effective for computer vision and pattern recognition, and that is the reason to investigate on their efficiency to classify the spectrograms of monitored device's EM activity. We defined two distinct neural networks architectures (see Table 4,5 in appendix), and compared their efficiency on our classification tasks. The first architecture is a simple MLP, which takes as input flattened spectrogram bandwidths. Our CNN architecture is a bit more complex, but still rather simple compared with the state-of-the-art networks used for image recognition. It is constituted of a stack of three atomic blocks where each block is made with one or two convolution layer(s), followed by a Max Pooling layer.

We furthermore study the effectiveness of more simplistic and less resource demanding machine learning tools like Naive Bayes (NB) and Support Vector Machines (SVM). As NB and SVM (or in general most machine learning algorithms) are prone to the curse of dimensionality [4], meaning that they do not scale well with the number of input features, we do not take as input the selected bandwidths as for the neural network models, but perform feature transformation on the spectrogram using LDA (see 3.1). LDA is a commonly used tool together with machine learning algorithms that allows to (drastically) reduce data dimension. Note that, on the other hand in most cases, LDA (or any feature transformation) is not suitable for neural network models, as features are transformed into different feature spaces and dependencies between features (in particular shapes and patterns) may be disconnected and harder to learn.

6 EXPERIMENTS

6.1 Data aquisition components

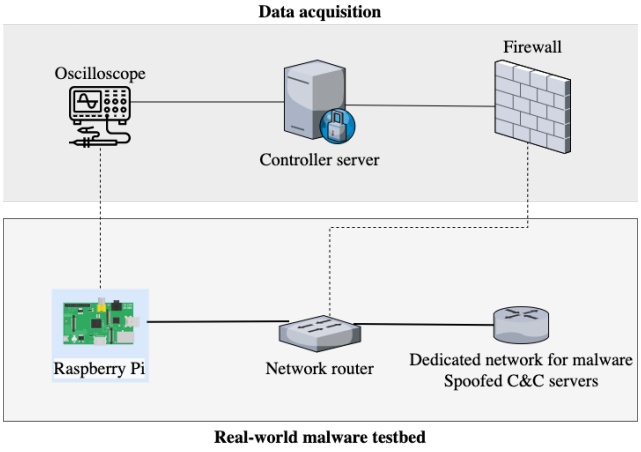


Figure 2: Overview of the proposed infrastructure: Experimental setup of malware testbed and data acquisition.

6.1.1 Target device. Evaluation of target device is critical for EM side-channel analysis. We determine three main requirements: *(i)* It must be a multi-purpose embedded device to support as many collected malware as possible rather than a specific set of malware or device; *(ii)* Its CPU must be a prominent architecture to avoid the lack of support of emerging IoT malware; *(iii)* It must be vulnerable to EM side-channel attacks.

We select Raspberry Pi 2B as a target device with 900 MHz quad-core ARM Cortex A7, 1 GB memory. Since its ARM architecture, size, power consumption and cost effective, make it a good candidate for any kinds of embedded and IoT scenarios including prototypes and developments. Our research focuses on a very general malware classification challenge rather than a narrowed solution to any specific device, in particular, as related works did not show diversity in results or analysis techniques across multiple devices (*e.g.* [33]).

It has been shown in previous works [13, 36] that cryptographic and anomaly activities are successfully distinguished by leveraging EM signals from the Raspberry Pi 2B. By not limiting the capabilities of the infrastructure with restricted bare-metal firmware, the Raspberry Pi is deployed with a fully-functioning Raspbian Buster OS of Linux 4.19.57-v7+ armv7l. A device under test with a fully functioning OS and multiple cores is to answer if it is possible to handle malware in more complex scenarios where malware is mixed with background processes, services and interrupts thus noisy EM traces.

To prevent the detection of any artificial artifacts by evasion techniques and keep a realistic environment, all background services are kept to their default with more than 100 running processes and services. Additionally, no adjustments, overclocking, or tuning of the processor clock rate are applied to the processor.

6.1.2 Malware testbed environment. To generate a practical victim environment that can trigger real malware, we applied different tools and techniques. We created honeypot directories under root folder, home folder, etc. Each malware execution will have a random initialized environment consisting of different valid files and extensions to assure that ransomware will be properly executed while not biased towards the recorded traces.

Besides, most IoT botnets architectures consist of one command and control server (C&C) which is continuously connected (except peer-to-peer botnet). To support our malware dataset, consist of *Mirai* and *Bashlite*, we implement a synthetic environment of central spoofed C&C server model. C&C servers are adopted to randomly deliver different commands to the botnet client in multiple attack scenarios (Fig. 2). To trigger a broad range of malicious activities, in each experiment the following commands are delivered: network scanner, flood targeted victim network in TCP/UDP, hibernation, or self-terminate etc. Furthermore, we installed multiple virtual machines in the same local network for absorbing network attacks coming from malware. The nature of malware samples, the execution coverage on software level in the device under testing are not altered, so that we presume no anti-analysis evasion techniques can survive during the bare-metal malware analysis.

6.1.3 Electromagnetic signal acquisition. We monitor the Raspberry Pi under the execution of benign and malicious dataset using a low to mid-range measurement setup. It consists of an oscilloscope with 1GHz bandwidth (Picoscope 6407) connected to a H-Field Probe (Langer RF-R 0.3-3), where the EM signal is amplified using a Langer PA-303 +30dB (Figure 3). To capture long-time execution of malware in the wild, the signals were sampled at 2MHz sampling rate.

The activity of the Raspberry Pi, when executing malware or generating benign activity, was recorded with a sample rate of 2MHz during 2.5 seconds. It has been chosen empirically based on (but not limited to) the constraints of the data acquisition components: imprecise trigger, and malware characteristics (e.g. sleep time with no activity of Mirai). The duration of 2.5 seconds is enough to obtain exploitable features for classification.

We collected 3 000 traces each for 30 malware binaries and 10 000 traces for benign activity. Thus, in total 100 000 traces were recorded, then we computed their short term Fourier transformation, as described in part 5.3.

The feature selection process with NICV on the spectrogram is illustrated in Figure 4. The left side shows the NICV where the

Figure 3: Probe setup consists of a H-Field probe placed 45 degree above the system processor.

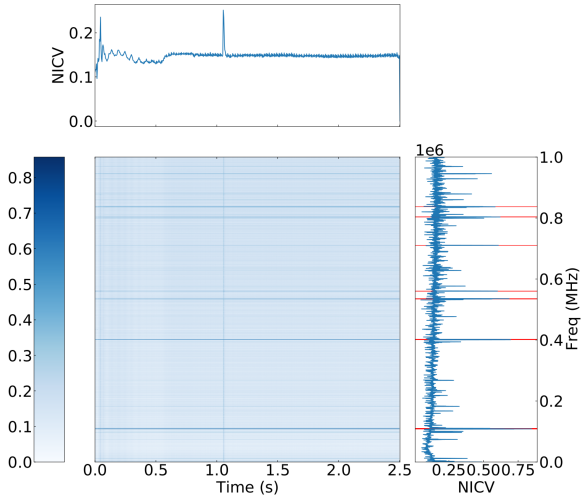


Figure 4: NICV (center) and in red the 20 selected frequencies ($\epsilon = 20$) on the mean over the time (right) and the mean over the frequencies (top)

right side highlights the selected frequencies that correspond to the twenty frequencies with the highest NICV mean over time (D).

6.2 Classification framework

6.2.1 Model input. In Section 5.3, we described how we generate the spectrograms from the EM activity recorded by the oscilloscope. We measure 500 000 points to get a 2.5s recording with a sampling rate of 2MHz, which is about 8MB per trace. As explained above, it was necessary to generate tens of thousands of spectrograms to train our neural network models. We choose $(M, O) = (8192, 4096)$ as parameters for the STFT. To reduce the dimension of the analyzed data and reduce the noise, we apply the feature extraction process described in Subsection 3.1 to conserve only ϵ different bandwidths. We tested $\epsilon \in \{4 \times i\}_{0<i<8}$ to determine for each experiment the number of conserved bandwidth ϵ_{opt} that reach the best accuracy.

Our dataset is split into three parts: a test dataset which will never be used during the training phase of the model, a validation dataset to assess the efficiency of the model on unseen data, and a training dataset. We kept by default 20% of the dataset for testing and used the 80% remaining for training and validation.

6.2.2 Training procedure. The neural network models have been trained over 50 epochs, where we stored the model according to the highest validation accuracy. In our setup using one RTX 2080 Ti GPU, CNN took around 50s per epoch, which gives $50 \times 50s = 2500s = 41$ min of training time, MLP performed 1 epoch within 9s, that gives $50 \times 9s = 2500s = 7$min. Testing one sample takes roughly 0.75s for MLP and 0.27s for CNN. NB and SVM are much less resource demanding than neural network models, and can be computed using standard CPU computation systems. On our system with an Intel(R) Xeon(R) Silver 4214 @2.20GHz with 24 cores and 128GB RAM, NB took 0.14s to train, and SVM 18.90s. The testing of one sample took for NB around 1μs, and for SVM between 1ms and 6ms depending on the number of features considered.

Results were obtained using the Keras backend of tensorflow running on one RTX 2080 Ti GPU for MLP/CNN and scikit-learn [27] library (version 0.23.2) for NB/SVM/LDA.

7 RESULTS AND DISCUSSION

7.1 Experimental results

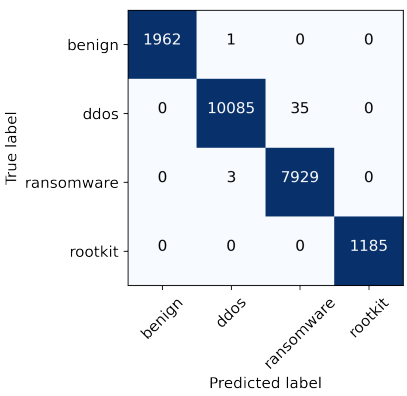
A synthesis of the results we obtained can be found in Table 3. The first column indicates the name of the scenario. In the second column we state the number of outputs (*i.e.* classes) of the network. Finally, the other columns show the accuracy with the optimal number of bandwidths as well as precision and recall of the two neural network models, and the two machine learning algorithms on the test dataset. Details about the construction of the scenarios can be found in Table 6 in the Appendix.

Type classification. We used traces measured during the activity of 30 malware samples, plus traces of benign activities (random, video, music, picture, camera activities), both in a random user environment to avoid biases. As explained in Section 4, the malware binaries are variations of five families: *gonnacry, keysniffer, maK_It, mirai* and *bashlite*, including seven different obfuscation techniques. In this scenario, we aim to retrieve the type of malware (if any) infecting the device at the time of the recording. This gives us a 4-class classification problem: ransomware, rootkit, DDoS, and benign. All models are very efficient for this problem (> 98% accuracy), and clearly obfuscation does not hinder type classification. We can observe that CNN (99.82%) is slightly more accurate than MLP, NB, and SVM. The confusion matrix is illustrated in Figure 5a, which illustrates the predicted classes (predicted label) from the network per executed binary (true label). The darker the color, the higher the proportion of correctly predicted labels. We can observe no confusion for the benign and rootkit class to any other class, and a minor confusion between DDos and ransomware in both directions.

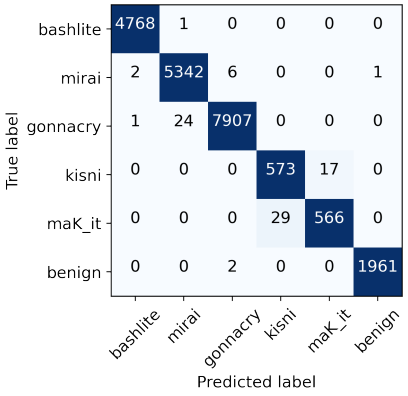
Family classification. In this scenario, we classify into the malware family plus benign class, which gives six classes: *bashlite, mirai, gonnacry, keysniffer, maK_it,* and benign. CNN gives the

		MLP			CNN			LDA + NB			LDA + SVM		
Scenarios	**#**	**AC [ϵ_{opt}]**	**RC**	**PR**	**AC [ϵ_{opt}]**	**RC**	**PR**	**AC [ϵ_{opt}]**	**RC**	**PR**	**AC [ϵ_{opt}]**	**RC**	**PR**
Type	4	99.75 [28]	99.83	99.85	**99.82** [28]	99.89	99.88	98.01 [24]	98.84	98.35	98.08 [24]	98.71	98.76
Family	6	99.57 [28]	93.13	93.11	**99.61** [28]	98.61	98.60	97.19 [28]	90.78	90.99	97.27 [28]	91.12	91.14
Virtualization	2	95.60 [20]	95.76	94.99	**95.83** [24]	96.19	95.14	91.29 [6]	91.07	90.49	91.25 [6]	90.69	90.62
Packer	2	93.39 [28]	93.36	93.06	**94.96** [20]	94.94	94.70	83.62 [16]	83.13	83.08	83.58 [16]	83.08	83.04
Obfuscation	7	73.79 [28]	72.77	72.79	**82.70** [24]	82.08	82.08	64.29 [10]	63.08	63.01	64.47 [10]	63.22	63.00
Executable	35	73.56 [24]	74.66	76.75	**82.28** [24]	83.08	83.11	70.92 [28]	72.29	71.94	71.84 [20]	72.47	72.32
Novelty (family)	5	88.41 [16]	92.35	91.01	**98.85** [24]	98.59	98.59	98.25 [28]	98.69	98.53	98.61 [28]	98.90	98.82

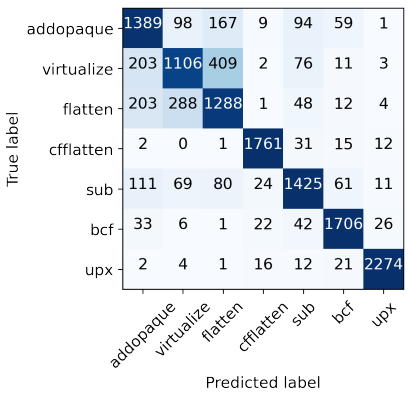
Table 3: Accuracy (AC), recall (RC) and the precision (PR) obtained with MLP, CNN, LDA + NB and LDA + SVM applied to several scenarios. ϵ_{opt} gives the value ϵ from Eq. (1) (the number of extracted bandwidth) obtaining the highest accuracy. Bold numbers indicate the highest accuracy on the testing set per scenario. The column "#" gives the number of classes per scenario.



(a) CNN type classification (acc 99.82%). ddos: mirai, obfuscated mirai, bashlite, obfuscated bashlite. Ransomware: gonnacry using blowfish, gonnacry using aes, gonnacry using des. Packed and without packing. rootkit: maK_it and kisni. benign: random, video, music, picture, camera activities using random user environments.



(b) CNN family classification (acc 99.61%). bashlite: original & obfuscated. mirai: original & obfuscated. Gonnacry: gonnacry using blowfish, gonnacry using aes, gonnacry using des. Packed & without packing. maK_it: original rootkit. kisni: original keysniffer rootkit. benign: random, video, music, picture, camera activities using random user env.



(c) CNN obfuscation classification (acc 82.70%). *addopaque*: opaque predicates, *virtualize*: virtualization, *flatten*: control flow flattening using Tigress, *cfflatten*: control flow flattening using O-LLVM, *sub*: instruction substitution, *bcf*: bogus control flow, *upx*: UPX packing.

Figure 5: Confusion matrices of several classification scenarii using the best performing algorithm.

highest accuracy with 99.61%, but also MLP and ML provide results > 97%. The confusion matrix is illustrated in Figure 5b, which shows that all classes are mostly correctly classified with a small confusion on both sides between *keysniffer* and *maK_it*. Again, we see that obfuscation does not hinder the classification.

Novelty classification. The previous experiments showed that it was possible to classify correctly known malware in its corresponding types and families. While this is certainly a first step, in real life malware analysis, it is very common to encounter unknown variations of a threat. To emulate this scenario, we split the dataset of malicious binaries according to the applied variations. Some of the variations of each of the five malware families were not used in the training dataset and reserved only for testing (detailed in Table 6 in Appendix). In addition, we did not include in the test dataset any of the variations we used for training. We reserved the processed spectrograms representing the activity of 18 binaries for training purposes, and the spectrograms representing the activity of the remaining binaries for testing purposes. Also, maK_It was only present in the training, and kisni only in the test dataset. As we can observe, even though the models are predicting unseen (obfuscated) variants, all model perform with an accuracy of > 92%, with CNN 99.38%. Accordingly, even *unseen* variations in the training phase do not hinder our methodology.

Virtualization and packer identification. In next two scenarios, we test if the binary is protected with virtualization or packing, which results to two (two-class) detection problems. For each of them, we used the traces of the original malware (*mirai*, *bashlite* and *gonnacry*) as well as the traces of the corresponding protected variation. We see that virtualization is slightly easier to detect than packing, with CNN performing with the highest accuracy of 95.83% and 94.96% respectively.

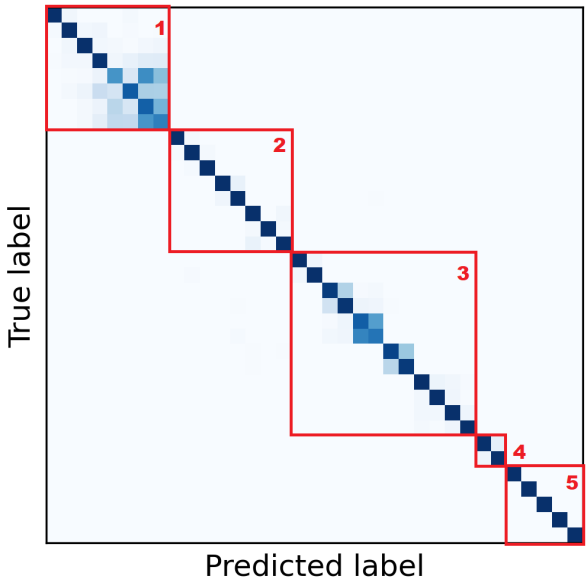


Figure 6: Confusion matrix of a CNN classification into 35 binaries from left to right (with and without obfuscation). (1) *bashlite_cfflatten, bashlite_upx, bashlite_bcf, bashlite, bashlite_addopaque, bashlite_sub, bashlite_flatten, bashlite_virtualize*; (2) *mirai_sub, mirai_bcf, mirai_cfflatten, mirai, mirai_upx, mirai_addopaque, mirai_flatten, mirai_virtualize*; (3) *gonnacry_des, gonnacry_des_upx, gonnacry, gonnacry_aes, gonnacry_aes_upx, gonnacry_upx, gonnacry_flatten, gonnacry_virtualize, gonnacry_addopaque, gonnacry_bcf, gonnacry_sub, gonnacry_cfflatten*; (4) *keysniffer, maK_It*; (5) *benign:* encode video, play audio, take picture, record camera, random.

Obfuscation classification. Here we are interested in classifying into the 7 obfuscation techniques: Opaque predicates, bogus control flow, control-flow flattening using O-LLVM or Tigress, instruction substitution, virtualization, or packing. Both of the network models were able to learn to differentiate obfuscation techniques independently of the five underlying malware families. CNN is more efficient achieving 82.70% (vs a random guess of 14.29%). Again, MLP was slightly worse and ML techniques show a gap of around 10%. The confusion matrix is illustrated in Figure 5c, which shows that for each obfuscation technique CNN predicted the correct label (the darkest color/highest number on the diagonal). Some confusion can be observed between addopaque, virtualize, and flatten, which are executed using Tigress, and indeed they share similar options[2].

This result shows that our methodology is not only able to distinguish between malicious activities, but even focus solely on behavioral features independent from the underlying binary execution.

Executable classification. This scenario is a straightforward executable identification, where the model is trying to profile exactly the binary that generated the spectrogram. This translates into a classification problem of 35 classes (including distinct benign activities), identifying the family and variant with possible obfuscation of the malware. For the number of classes and the closeness of the underlying executions, all models get very good results, where CNN is more effective with 82.28% vs a random guess of only 2.86%.

The confusion matrix of the CNN binary classification is given in Figure 6. As we can see, if confusions between classes happen, it happens between binaries that belong to the same family (squared in red in the figure). In addition, we observe that in most of the cases, the "darkest" color appears on the diagonal, which means that the highest score (output of the CNN) occurs for the true class label. So, in most cases, obfuscation does not hinder exact binary profiling.

However, we still observe groups of binaries which are harder to distinguish and one misclassification. More precisely, one can observe that *bashlite_cfflatten*, *bashlite_upx*, *bashlite_bcf*, and *bashlite* have (nearly) no confusion to other binaries, which means that the obfuscation does not mask the behavior of the binary and the obfuscation technique itself is visible and distinguishable; *bashlite_addopaque* is misclassified as *bashlite_flatten* which is inline with our previous explanation on similarities of the underlying techniques, and there is a confusion between *bashlite_flatten* and *bashlite_virtualize*.

For *mirai* and its variants we see a much smaller effect of the obfuscation techniques on the classification outcome as for *bashlite* and *gonnacry*. Meaning that the obfuscation technique is clearly identifiable and does not mask the behavior of mirai itself.

For gonnacry we have several distinct groups:

(1) *gonnacry-des-upx*, *gonnacry-des*: only a very minor confusion can be visible between the packed and unpacked version using des. Interestingly, there is no confusion using des with the version of aes and blowfish and their packed variants.
(2) *gonnacry*, *gonnacry-aes*: *gonnacry* and *gonnacry-aes* are slightly confused.
(3) *gonnacry-des* and *gonnacry-des-upx* are not confused with any other binary;
(4) *gonnacry* and *gonnacry-aes* have a slight confusion, which means that in some cases the encryption with blowfish and aes are not clearly distinguishable;
(5) this effect is even more present when the binaries are packed, i.e. for *gonnacry-upx* and *gonnacry-aes-upx*;
(6) again, like for bashlite, we see a slight confusion between *gonnacry_flatten* and *gonnacry_virtualize*.
(7) *gonnacry_addopaque*, *gonnacry_bcf*, *gonnacry_sub*, *gonnacry_cfflatten*: we observe only nearly no confusion between these four obfuscation techniques.

We see nearly no confusion when predicting maK_it and keysniffer. Finally, as before the benign binaries show no confusion with any other malicious binaries, and there is no confusion between each of the benign classes.

[2] http://tigress.cs.arizona.edu/transformPage/docs/flatten/index.html

7.2 Discussion

Novel malware. The results we obtained show that our approach is successful to classify various malware samples into their types, families, exact binaries, and identify/classify obfuscation. The closeness between accuracy, recall, and precision of each experiment indicates robustness and no overfitting to any specific class. We are able to classify malware variations unseen during the training phase which is particularly relevant in practical scenarios when considering the evolution of malware. Furthermore, we realized the measurements in a stable lab environment, still no exact triggering, nor any restriction on the (background) processes of the target device have been done, which corresponds to a setup a malware analyst could exploit.

Obfuscation resiliency. In this paper, we examined 7 obfuscation techniques including packers and virtualization, which have been used by real-world malware as a growing trend to exploit cryptors and packers to hide the true intent of malware samples. While previous solutions such as signature-based packer detection can be evaded, our results show that we can distinguish between obfuscation techniques solely based on EM traces, which gives the opportunity to analyze the evolution of IoT malware since new obfuscation techniques will be reformed to thwart detection.

ML algorithms. From our results, we can observe that the accuracy of SVM and NB are close for more straightforward classification problems like type and family, but the gap gets bigger when consider more complex scenarios (i.e. executable, obfuscation).

Note that, all our results have been obtained by considering that the malware analyst measures only one trace per binary to predict the correct class. However, it could be possible that he has the resources to measure multiple times the same binary execution and to reduce noise, computes the mean over each of these execution traces. Results using this approach are given in Figure 7 in the Appendix for SVM and NB, which shows a drastic improvement in many scenarios. For example, NB could reach $> 80\%$ for binary classification and 100% in type and novelty classification. Interestingly, we could not observe an improvement for MLP and CNN, which is discussed in more detail in the Appendix.

Malware evasion. From our results, one can observe that malware evasion (*i.e.* prevention from our methodology) is not straightforward. Particularly, we derived that our system is robust against various code transformation/obfuscation, including random junk insertion, packing, and virtualization, even when the transformation is previously not known to the system. Another approach of evasion, instead of obfuscating malicious behavior, could be to hide exploitable information by lowering the signal-to-noise ratio. This could, for example, be achieved by forcing highly parallel/multi-core executions. However, as our methodology relies on EM emanation, that can be observed on a local and global scale, and on frequency transformation with filtering, it is unclear if hiding exploitable information is (easily) achievable at all. Even more, unexpected highly parallel/multi-core activities can be more easily detected as abnormal behavior, which is not in the interest of malware designers. We therefore see the topic of malware evasion against physical side-channel information as a new open direction with nonstraightforward solutions.

8 CONCLUSIONS AND PERSPECTIVES

We have demonstrated in this paper that by using simple neural network models, it is possible to gain considerable information about the state of a monitored device, by observing solely its EM emanations. We were indeed able to not only detect, but also determine the type of real-world malware infecting a Raspberry Pi running a full Linux OS, with an accuracy of 99.89% on a test dataset including 20 000 traces from 30 different malware samples (and five different benign activities). We demonstrated that software obfuscation techniques do not hinder our classification approach, even if the obfuscation technique was not known to the analyst before. Even more, we showed that it is possible to detect a particular obfuscation and classify between them (or groups of obfuscation techniques), and classify the family with its exact variant/obfuscation labels.

Given our experimental results, malware analysts therefore profit from our robust methodology to gain a better understanding about the variant, type/family, forensic, and/or evolution of malware groups and campaigns, particularly in the context when software systems fail (due to malware evasion) or cannot be applied (due to restricted resources or update processes on the embedded device).

Another interesting direction could be the investigation of other architectures and devices, to assess in which measure the knowledge learned by a model on one device can be transferred to another one. Our work can be considered as a first step towards (detailed) behavioral analysis through electromagnetic emanation opening a new research direction for future work.

Acknowledgement. The work was supported by the French Agence Nationale de la Recherche (ANR) under reference ANR-18-CE39-0001 (AHMA). We thank our colleague Ronan Lashermes who provided hardware and side-channel insights that greatly assisted this work.

REFERENCES

[1] 10 evil user tricks for bypassing anti-virus 2013. https://blog.netspi.com/10-evil-user-tricks-for-bypassing-anti-virus/.

[2] Amin Azmoodeh, Ali Dehghantanha, Mauro Conti, and Kim-Kwang Raymond Choo. 2018. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *Journal of Ambient Intelligence and Humanized Computing* 9, 4 (Aug. 2018), 1141–1152. https://doi.org/10.1007/s12652-017-0558-5

[3] Donabelle Baysa, Richard M Low, and Mark Stamp. 2013. Structural entropy and metamorphic malware. *Journal of computer virology and hacking techniques* 9, 4 (2013), 179–192.

[4] Richard Bellman. 1957. *Dynamic Programming* (1 ed.). Princeton University Press, Princeton, NJ, USA. http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false

[5] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. 2014. NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. In *International Symposium on Electromagnetic Compatibility (EMC '14 / Tokyo)*. IEEE. eprint version: https://eprint.iacr.org/2013/717.pdf.

[6] Nikhil Chawla, Harshit Kumar, and Saibal Mukhopadhyay. 2021. Machine Learning in Wavelet Domain for Electromagnetic Emission Based Malware Analysis. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3426–3441. https://doi.org/10.1109/TIFS.2021.3080510

[7] Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, and Kevin Fu. 2013. WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices. In *2013 USENIX Workshop on Health Information Technologies (HealthTech 13)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/healthtech13/workshop-program/presentation/clark

[8] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M. Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, Daisuke Nakatsu, Kazuo Ohta, Kazuo Sakiyama, Laurent Sauvage, Werner Schindler, Marc Stöttinger, Nicolas Veyrat-Charvillon, Matthieu Walle, and Antoine Wurcker. 2014. Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest. *J. Cryptogr. Eng.* 4, 4 (2014), 259–274. https://doi.org/10.1007/s13389-014-0075-9

[9] Christian Collberg, Sam Martin, Jonathan Myers, Bill Zimmerman, Petr Krajca, Gabriel Kerneis, Saumya Debray, and Babak Yadegari. [n.d.]. The Tigress C Diversifier/Obfuscator. http://tigress.cs.arizona.edu/index.html

[10] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. A taxonomy of obfuscating transformations.

[11] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2018. Understanding Linux Malware. In *2018 IEEE Symposium on Security and Privacy (SP)*. 161–175. https://doi.org/10.1109/SP.2018.00054 ISSN: 2375-1207.

[12] Fei Ding, Hongda Li, Feng Luo, Hongxin Hu, Long Cheng, Hai Xiao, and Rong Ge. 2020. DeepPower: Non-intrusive and Deep Learning-based Detection of IoT Malware Using Power Side Channels. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 33–46.

[13] Ibraheem Frieslaar and Barry Irwin. 2017. Recovering AES-128 encryption keys from a Raspberry Pi. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. 228–235.

[14] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. 2006. Templates vs. Stochastic Methods. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings (Lecture Notes in Computer Science)*, Louis Goubin and Mitsuru Matsui (Eds.), Vol. 4249. Springer, 15–29. https://doi.org/10.1007/11894063_2

[15] Annelie Heuser and Michael Zohner. 2012. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings (Lecture Notes in Computer Science)*, Werner Schindler and Sorin A. Huss (Eds.), Vol. 7275. Springer, 249–264. https://doi.org/10.1007/978-3-642-29912-4_18

[16] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated.

[17] Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin. 2015. Obfuscator-LLVM – Software Protection for the Masses. In *Proceedings of the IEEE/ACM 1st International Workshop on Software Protection, SPRO'15, Firenze, Italy, May 19th, 2015*, Brecht Wyseur (Ed.). IEEE, 3–9. https://doi.org/10.1109/SPRO.2015.10

[18] Stefan Katzenbeisser, Johannes Kinder, and Helmut Veith. 2011. *Malware Detection.* Springer US, Boston, MA, 752–755. https://doi.org/10.1007/978-1-4419-5906-5_838

[19] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, R. L. Callan, A. Yeredor, M. Prvulovic, and A. Zajic. 2019. IDEA: Intrusion Detection through Electromagnetic-Signal Analysis for Critical Embedded and Cyber-Physical Systems. *IEEE Transactions on Dependable and Secure Computing* (2019), 1–1.

[20] Haider A. Khan, Nader Sehatbakhsh, Luong N. Nguyen, Milos Prvulovic, and Alenka G. Zajic. 2019. Malware Detection in Embedded Systems Using Neural Network Model for Electromagnetic Side-Channel Signals. *J. Hardware and Systems Security* 3, 4 (2019), 305–318. https://doi.org/10.1007/s41635-019-00074-w

[21] Jared Lee, Thomas H Austin, and Mark Stamp. 2015. Compression-based analysis of metamorphic malware. *International Journal of Security and Networks* 10, 2 (2015), 124–136.

[22] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power analysis attacks - revealing the secrets of smart cards.* Springer.

[23] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. 2018. N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Computing* 17, 3 (July 2018), 12–22. https://doi.org/10.1109/MPRV.2018.03367731 Conference Name: IEEE Pervasive Computing.

[24] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic. 2017. EDDIE: EM-based detection of deviations in program execution. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 333–346.

[25] Markus F.X.J. Oberhumer, László Molnár, and John F. Reiser. [n.d.]. UPX the Ultimate Packer for eXecutables. https://upx.github.io/

[26] P. O'Kane, S. Sezer, and K. McLaughlin. 2011. Obfuscation: The Hidden Malware. *IEEE Security & Privacy* 9, 5 (2011), 41–47. http://dblp.uni-trier.de/db/journals/ieeesp/ieeesp9.html#OKaneSM11

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[28] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. 2017. Template attack versus Bayes classifier. *J. Cryptogr. Eng.* 7, 4 (2017), 343–351. https://doi.org/10.1007/s13389-017-0172-7

[29] Milos Prvulovic, Alenka Zajić, Robert L Callan, and Christopher J Wang. 2016. A method for finding frequency-modulated and amplitude-modulated electromagnetic emanations in computer systems. *IEEE Transactions on Electromagnetic Compatibility* 59, 1 (2016), 34–42.

[30] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. 2007. Detecting System Emulators. In *Information Security*, Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[31] R. A. Riley, J. T. Graham, R. M. Fuller, R. O. Baldwin, and A. Fisher. 2019. A New Way to Detect Cyberattacks: Extracting Changes in Register Values From Radio-Frequency Side Channels. *IEEE Signal Processing Magazine* 36, 2 (March 2019), 49–58. https://doi.org/10.1109/MSP.2018.2888893

[32] Tobias Schneider and Amir Moradi. 2016. Leakage assessment methodology - Extended version. *J. Cryptogr. Eng.* 6, 2 (2016), 85–99. https://doi.org/10.1007/s13389-016-0120-y

[33] N. Sehatbakhsh, A. Nazari, M. Alam, F. Werner, Y. Zhu, A. Zajic, and M. Prvulovic. 2020. REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals. *IEEE Trans. Comput.* 69, 3 (2020), 312–326.

[34] Nader Sehatbakhsh, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. 2016. Spectral profiling: Observer-effect-free profiling by monitoring EM emanations. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–11.

[35] Baljit Singh, Dmitry Evtyushkin, Jesse Elwell, Ryan Riley, and Iliano Cervesato. 2017. On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. ACM, New York, NY, USA, 483–493. https://doi.org/10.1145/3052973.3052999 event-place: Abu Dhabi, United Arab Emirates.

[36] Xiao Wang, Quan Zhou, Jacob Harer, Gavin Brown, Shangran Qiu, Zhi Dou, John Wang, Alan Hinton, Carlos Aguayo Gonzalez, and Peter Chin. 2018. Deep learning-based classification and anomaly detection of side-channel signals. In *Cyber Sensing 2018*, Vol. 10630. International Society for Optics and Photonics, 1063006.

[37] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. 2016. SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion. In *Research in Attacks, Intrusions, and Defenses*, Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 165–187.

[38] Alenka Zajic, Milos Prvulovic, Haider Adnan Khan, and Monjur Alam. 2018. Detailed tracking of program control flow using analog side-channel signals: a promise for IoT malware detection and a threat for many cryptographic implementations. In *Cyber Sensing 2018*, Peter Chin and Igor V. Ternovskiy (Eds.). SPIE, Orlando, United States, 5. https://doi.org/10.1117/12.2304382

9 APPENDIX

9.1 Neural network architectures

Layer	Size	Filter	Activation
Flatten	spectrogram_size	_	leaky relu
Dense	500	_	leaky relu
Dense	200	_	leaky relu
Dense	100	_	leaky relu
Dense	N	_	softmax

Table 4: MLP architecture

Layer	Size	Filter	Activation
Convolution	64	7×7	leaky relu
Max Pooling	64	2×2	_
Convolution	128	3×3	leaky relu
Convolution	128	3×3	leaky relu
Max Pooling	128	2×2	_
Convolution	256	3×3	leaky relu
Convolution	256	3×3	leaky relu
Max Pooling	256	2×2	_
Dense	128	_	leaky relu
Dense	64	_	leaky relu
Dense	N	_	softmax

Table 5: CNN architecture

9.2 Experimental results on the meaning of test traces

Figure 7 shows accuracy of NB and SVM when calculating the mean over t execution measurements from the same binary in the test dataset. We see an improvement in all scenarios. Therefore, when the number of executions/ measurements per unknown binary is not a restricting factor for the malware analyst, then computing the mean over t traces will result in a more accurate prediction. This meaning process is usually in the side-channel domain, as in [8]. Interestingly, we could not observe a straightforward improvement when applying this technique to MLP and CNN classifications. One reason could be that the random user environment changes for each execution, thus even though the binary stays unaltered, the measurement trace changes. Now when calculating the mean, the patterns of features that may help MLP and CNN to make correct predictions may be mixed or changed. Contrary. NB and even SVM may not be able to model these patterns due to their more simplistic nature and computing the mean can be seen as noise reduction instead.

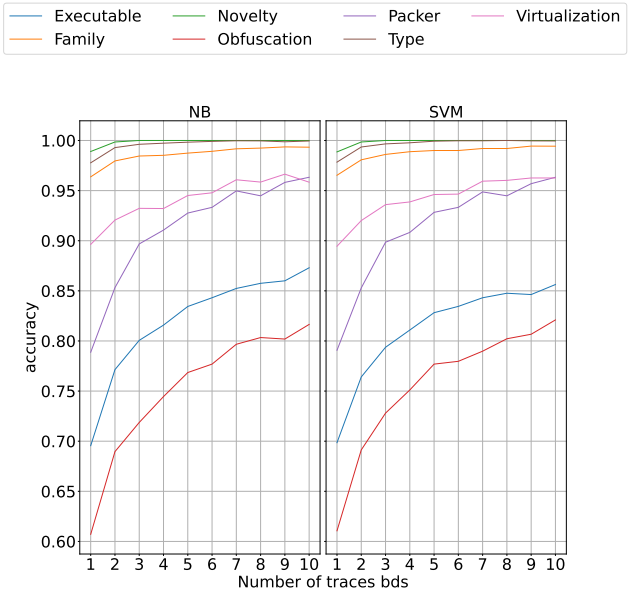


Figure 7: Accuracy when computing the mean over $t = [1, 2, 3, \ldots, 10]$ samples per binary in the test dataset. One can observe a drastic performance improvement for SVM and NB for the scenarios where the accuracy was lower to begin with. For type and novelty classification the accuracy reaches 100%.

Binaries names	#	Types tags	Family tags	Virtualization tags	Packer tags	Obfuscation tags	Executable tags	Novelty (family) tags
random34	3000	benign	benign				random34	benign
mirai.arm7	3000	ddos	mirai	original	not_packed		mirai	mirai [*]
mirai_addopaque	3000	ddos	mirai			addopaque	mirai_addopaque	mirai [*]
mirai_virtualize	3000	ddos	mirai	virtualized		virtualize	mirai_virtualize	mirai [+]
mirai_flatten	3000	ddos	mirai			flatten	mirai_flatten	mirai [+]
mirai-bcf	3000	ddos	mirai			bcf	mirai-bcf	mirai [*]
mirai-cfflatten	3000	ddos	mirai			cfflatten	mirai-cfflatten	mirai [+]
mirai-sub	3000	ddos	mirai			sub	mirai-sub	mirai [+]
upx-mirai	1000	ddos	mirai		packed	upx	mirai-upx	mirai [*]
gonnacry	6000	ransomware	gonnacry	original	not_packed		gonnacry	gonnacry [*]
upx-gonnacry	3000	ransomware	gonnacry		packed	upx	gonnacry-upx	gonnacry [*]
aes-upx-gonacry	3000	ransomware	gonnacry		packed	upx	gonnacry-aes-upx	gonnacry [+]
aes-gonacry	3000	ransomware	gonnacry		not_packed		gonnacry-aes	gonnacry [+]
des-gonnacry	3000	ransomware	gonnacry		not_packed		gonnacry-des	
des-upx-gonnacry	3000	ransomware	gonnacry		packed		gonnacry-des-upx	
gonnacry_Virtualize2	3000	ransomware	gonnacry	virtualized		virtualize	gonnacry_virtualize2	gonnacry [*]
gonnacry_flatten	3000	ransomware	gonnacry			flatten	gonnacry_flatten	gonnacry [*]
gonnacry_bcf	3000	ransomware	gonnacry			bcf	gonnacry_bcf	gonnacry [*]
gonnacry_sub	3000	ransomware	gonnacry			sub	gonnacry_sub	gonnacry [*]
gonnacry_cfflatten	3000	ransomware	gonnacry			cfflatten	gonnacry_cfflatten	gonnacry [+]
gonnacry_addopaque	3000	ransomware	gonnacry			addopaque	gonnacry_addopaque	gonnacry [*]
maK_it4.19.57-v7+.ko	6000	rootkit	maK_it				rootkit_maK_it	rootkit [*]
kisni-4.19.57-v7+.ko	3000	rootkit	kisni				rootkit_kisni	rootkit [+]
bashlite	3000	ddos	bashlite	original	not_packed		bashlite	bashlite [*]
bashlite_bcf	3000	ddos	bashlite			bcf	bashlite_bcf	bashlite [*]
bashlite_flatten	3000	ddos	bashlite			flatten	bashlite_flatten	bashlite [+]
bashlite_upx	3000	ddos	bashlite		packed	upx	bashlite_upx	bashlite [*]
bashlite_addopaque	3000	ddos	bashlite			addopaque	bashlite_addopaque	bashlite [*]
bashlite_cfflatten	3000	ddos	bashlite			cfflatten	bashlite_cfflatten	bashlite [*]
bashlite_sub	1000	ddos	bashlite			sub	bashlite_sub	bashlite [*]
bashlite_virtualize	6000	ddos	bashlite	virtualized		virtualize	bashlite_virtualize	bashlite [+]
playaudio	1000	benign	benign				playaudio	benign
recordcamera	3000	benign	benign				recordcamera	benign
takepicture	3000	benign	benign				takepicture	benign
encodevideo	1000	benign	benign				encodevideo	benign

Table 6: Malware tag map. The first column lists all malware and benign samples, followed by the number of recorded traces. Then each column refers to a scenario and gives for each sample the group it belongs to if it has been used. [*] (resp., [+]) means the sample has been used only during the training phase (resp. the testing phase), by default samples are used during both phases (80% for training, 20% for testing).