IoT Malware and Rootkit Detections Using Electromagnetic Insights: Unveiling the Unseen

Duy-Phuc Pham[1] and Damien Marion[2] and Annelie Heuser[2]

[1] Trellix, [2] Univ Rennes, CNRS, Inria, IRISA Rennes, France

This paper was presented at Botconf 2024, Nice, 23-26 April 2023, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

The Internet of Things (IoT) is a collection of interconnected devices, becoming increasingly complicated and suffering from inadequate security measures. They frequently employ outdated hardware and software without taking security risks into account, which makes them a target for cybercriminals, particularly those specializing in malware and rootkits. In this paper, we will present two strategies for exploiting electromagnetic side channels and address two challenges: malware classification in the presence of obfuscations and rootkit detection. Our approach focuses on IoT devices, specifically targeting ARM and MIPS architectures in Raspberry Pi and Creator CI20 devices. The framework employs advanced data preprocessing methods, allowing analysts to select a variety of machine learning and deep learning models based on their specific requirements.

Our findings were published separately at (including data and codes):

- **ACSAC-2021:** "Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification" [1] (with an extended version presented at hardwear.io'22 USA),
- **RAID-2022:** "ULTRA: Ultimate Rootkit Detection over the Air" [2].

Keywords: Malware classification, obfuscation, side-channel analysis, rootkit detection, software-defined radio (SDR), machine learning, deep learning, Electromagnetic, IoT devices

1 Introduction

By 2025, a projection suggests an excess of 64 billion IoT devices [3], with continued production growth expected as beyond 5G technologies mature. Simultaneously, alongside the progression in IoT and embedded devices, cyberattacks' frequency and diversity have surged in recent years, rendering current security approaches outdated within a short timeframe [4, 5]. Many IoT manufacturers employ Linux-based operating systems, simplifying the migration of rootkits to target embedded devices. Traditional malware detection solutions rely on static or dynamic analysis, each presenting various limitations. Particularly challenging are issues associated with the diverse architectures of IoTs, obfuscation techniques [1], and the constrained resources or accessibility of most IoT devices.

In this paper, we introduce two frameworks designed to address the challenges outlined in the preceding paragraph. The first framework, named AHMA [1], and the second, ULTRA [2], are both built upon the foundation depicted in Fig. 1. The paper follows a similar structure: after a short state-of-the-art, we outline the acquisition process, followed by preprocessing, then the classification/detection results, and ultimately, we draw our conclusions. Throughout the paper, we will emphasize the similarities and differ-

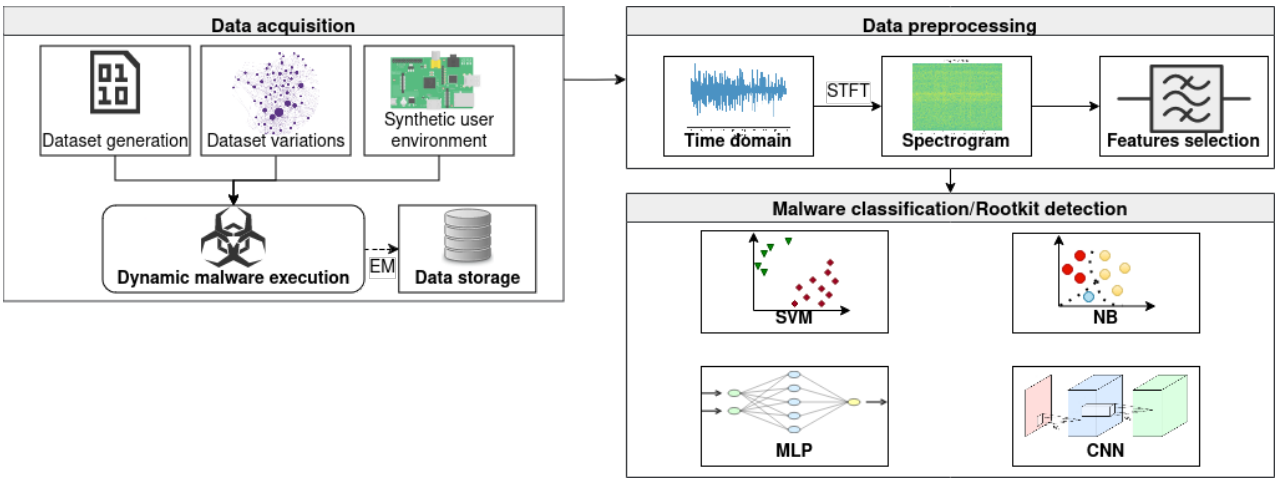

Figure 1: Workflow of our solutions

ences between our frameworks.

2 State-of-the-art

WattsUpDoc [6] was one of the earliest efforts in malware detection through hardware side-channels, demonstrating the measurement of power usage on medical embedded devices. Recently, in [7], the authors proposed detecting and classifying malware by observing EM signals. This approach only detects active malware during its behavioral activity but fails to detect stealthy rootkits after their installation. [8] described a network time analysis approach for monitoring performance changes caused by hardware virtualization, aiming to detect hardware virtualization rootkits. [9, 10] identified rootkits using power-based malware detection on general-purpose computers, and [9, 11, 12] used machine learning (ML) and deep learning (DL) for behavioral detection based on CPU power consumption. Gibraltar [13] and Copilot [14] leveraged direct memory access (DMA) via physical PCI to detect rootkits in kernel memory from another machine. However, challenges such as system overhead, asynchronous kernel read/write, race conditions, and timing attacks remain significant obstacles for this solution.

3 Data acquisition

3.1 Malware and rootkit samples

Our first framework, AHMA aims to classify a wide variety of malwares on which obfuscation techniques have been applied. A comprehensive list of the malware utilized is accessible in Tab. 3 detailing the family, type, and all the obfuscation techniques applied, benign samples are also listed. Our second framework, ULTRA, enables the detection of rootkits, which is why the list of samples differs from that of AHMA. You can find them in Tab. 4. To effectively detect a rootkit, we require a covert method to prompt its presence. To achieve this, we define what we refer to as a "bait" (see Definition 1). Table 4 associates the baits and rootkits. The execution of baits can trigger specified benign activities (e.g., system calls, keystroke input, ...) without needing to know the exact rootkit family involved. Consequently, any deviations observed between bait executions in a clean state and an infected state will indicate the presence of a rootkit on the target device.

Definition 1. *A* bait*, which is a software or hardware stimulus on a device, has the following requirements:*
(i) The bait can trigger partial or full behavior of rootkits without knowing modus operandi of the rootkit in advance;
(ii) It has a variable duration time of execution activities that can be remotely controlled;
(iii) It cannot be distinguished from common benign behavior (e.g., it relies on unprivileged execution).

For example, *diamorphine* rootkit intercepts the *kill()* syscall to redirect it to *hacked_kill()*, which acts as a switch for three specific signal inputs (as shown in Fig. 2). If the signal matches, the call will result in either process hiding, module hiding, or root escalation.

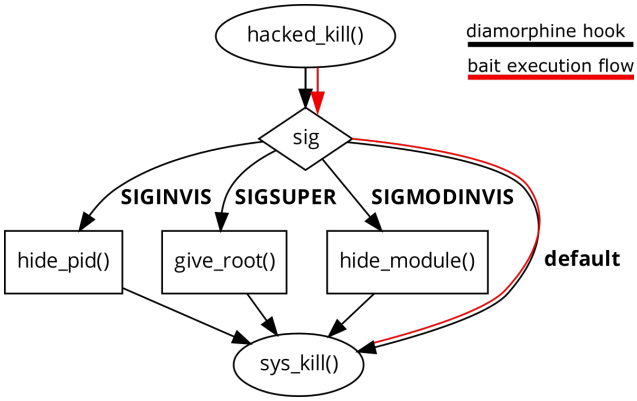

Figure 2: Schema of the execution flow for the kill bait operating under the influence of the diamorphine rootkit, which has infected the Linux kernel by hooking the *kill()* system call

3.2 Target devices

This work focuses on rootkit detection on IoT devices with restricted resources. We conducted experiments on two widely used embedded architectures (Table 1) known for their versatility, size, power consumption, and cost-effectiveness. Previous studies have shown that cryptographic and anomaly activities can be distinguished using EM signals from the Raspberry Pi. However, no prior research has investigated side-channel leakage on the MIPS Creator CI20, ensuring an unbiased experiment design. Our approach addresses rootkit detection across various hardware and software combinations rather than a specific device or architecture. We utilize fully-functioning Linux on MIPS and ARM, allowing for comprehensive IoT applications with typical internal noise such as background processes, services, and interrupts. Two different versions of Raspberry Pi have been used for AHMA and ULTRA.

Device	Arch.	CPU	RAM	Linux	Fram.
Rasp. B+	ARM32	700	512MB	4.1.7	ULTRA
Rasp. 2B	ARM32	900	1GB	4.19.57	AHMA
Creator CI20	MIPS32	1200	1GB	3.18.3	both

Table 1: Target devices specification, architectures (Arch.), their CPU frequency in MHz, RAM, Linux version and the framework (Fram.) tested on.

3.3 Electromagnetic leakage acquisition

In AHMA, we monitor targets under the execution of benign and malicious datasets using a low to mid-range measurement setup, Fig. 3. This setup includes a 1GHz bandwidth oscilloscope (Picoscope 6407) connected to an H-Field Probe (Langer RF-R 0.3-3). The EM signal is amplified with a Langer PA-303 +30dB amplifier. To capture long-term malware execution in the wild, signals are sampled at a 2MHz sampling rate.

Figure 3: AHMA framework acquisition involves recording for 2.5 seconds

In ULTRA, target devices are monitored with a slightly different setup aimed at a lower-cost solution. The oscilloscope is replaced by a HackRF SDR device with a frequency range of 1MHz-6GHz, Fig. 4.

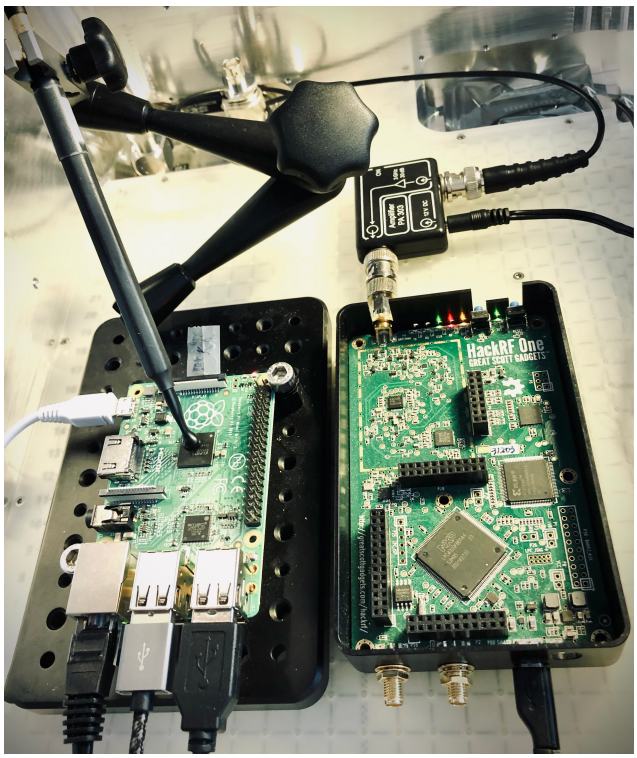

Figure 4: ULTRA framework acquisition involves recording for 0.5 seconds

4 Analysis and results

4.1 Preprocessing

Before exploiting the recorded data, we need to apply some preprocessing. First, we convert the time representation to the time-frequency domain using the Short-Time Fourier Transform (STFT) with a window size of 8192 and an overlap window size of 4096. Then, we apply a well-known feature selection algorithm called NICV [15] to extract the most relevant bandwidth.

4.2 Classifiers

To classify malware with AHMA, we selected two machine learning algorithms: Naive Bayes (NB) and Support Vector Machine (SVM), and crafted two small neural networks: one Multi-Layer Perceptron (MLP) and one Convolutional Neural Network (CNN). With the ULTRA detection framework, only NB, SVM, and MLP are used. NB and SVM have been integrated with dimension reduction algorithms: Linear Discriminant Analysis (LDA) in AHMA and Kernel Principal Component Analysis (KPCA) in ULTRA.

Our dataset is divided into three segments: a test dataset, which remains untouched during the model training phase, a validation dataset used to evaluate the model's performance on unseen data, and a training dataset. By default, we allocate 20% of the dataset for testing purposes and utilize the remaining 80% for training and validation.

4.3 Results

With AHMA, which aims to classify malware, we selected different scenarios listed in the header of Table 3. For clarity, we provide only the results obtained with the neural networks for the AHMA framework in Table 2. We observe extremely high accuracies for *type* and *family*. For *novelty (family)*, we achieve 99.92% and 96.65% accuracy for the CI20 and Raspberry Pi, respectively, indicating that we can detect the family of unseen (during the training phase) samples with a high probability through electromagnetic emanations. The packing techniques can also be classified, with a slight degradation of the score. Finally, *obfuscation* and *type* are more challenging to identify.

With ULTRA, which aims to detect rootkits, we illustrate our results in Fig. 5 using the bait *getdents*. Since the acquisition involves a simple call to *getdents*, we can repeat the experiments and average the recorded EM traces to decrease the noise effect. Regarding the efficiency of ULTRA, all rootkits can be detected with an accuracy of 100% by at least one classifier on each target (excluding *m0hamed* on Raspberry Pi, which reaches 99.7%). Even better, on CI20, the NB classifier (as well as the SVM one) trained on *vlany* can detect all other samples.

5 Conclusions and on-going works

ULTRA and AHMA, employing electromagnetic traces as a novel approach, significantly contribute to malware classification and rootkit detection. ULTRA, targeting rootkit detection, showcases encouraging results with nearly flawless accuracy in identifying rootkits across various platforms, marking a notable advancement in malware detection techniques. Meanwhile, AHMA specializes in malware classification, demonstrating remarkable accuracy in identifying malware types and families, despite facing challenges in detecting obfuscation. The innovative utilization of electromagnetic traces by both frameworks represents a promising step forward in malware and rootkit analysis through side-channels.

Currently, we are focusing on the reproducibility of our results. First, we are in contact with researchers who are building the same setup. Second, we have developed student projects to make the ULTRA framework more portable by using a Jetson Nano board that embeds a GPU. Finally, we are collaborating to improve the classification step.

Acknowledgment: 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 and Olivier Zendra who provided hardware and side-channel insights and greatly assisted this work.

Author details

Duy-Phuc Pham

Trellix
duyphuc.pham@trellix.com
https://phuc.fr/

Damien Marion

Univ Rennes, CNRS, Inria, IRISA Rennes, France
damien.marion@irisa.fr
https://damien-marion.github.io/

Annelie Heuser

Univ Rennes, CNRS, Inria, IRISA Rennes, France
annelie.heuser@irisa.fr
https://axnxlxe.github.io/

	#	MLP		CNN		MLP		CNN	
Scenarios		**Rasp.**				**ci20**			
Type	4	98.21	[8]	**99.03**	[7]	99.72	[7]	**99.95**	[3]
Family	6	98.00	[14]	**99.36**	[11]	96.96	[11]	**96.99**	[11]
Novelty (family)	5	**96.65**	[11]	89.04	[4]	**99.92**	[25]	99.91	[11]
Virtualization	2	94.20	[12]	**95.11**	[8]	**59.50**	[8]	51.00	[1]
Packer	2	92.02	[18]	**92.30**	[8]	**84.80**	[10]	49.10	[1]
Obfuscation	7	71.10	[11]	**78.90**	[15]	42.80	[5]	**44.70**	[5]
Executables	35	70.70	[5]	**77.10**	[10]	29.90	[9]	**34.30**	[6]

Table 2: Accuracy obtained with MLP and CNN applied to several scenarios for the Raspberry Pi and CI20. The first column indicates the scenario, the second column indicates the number of classes, followed by the accuracies. The numbers in parentheses refer to the number of bandwidths used.

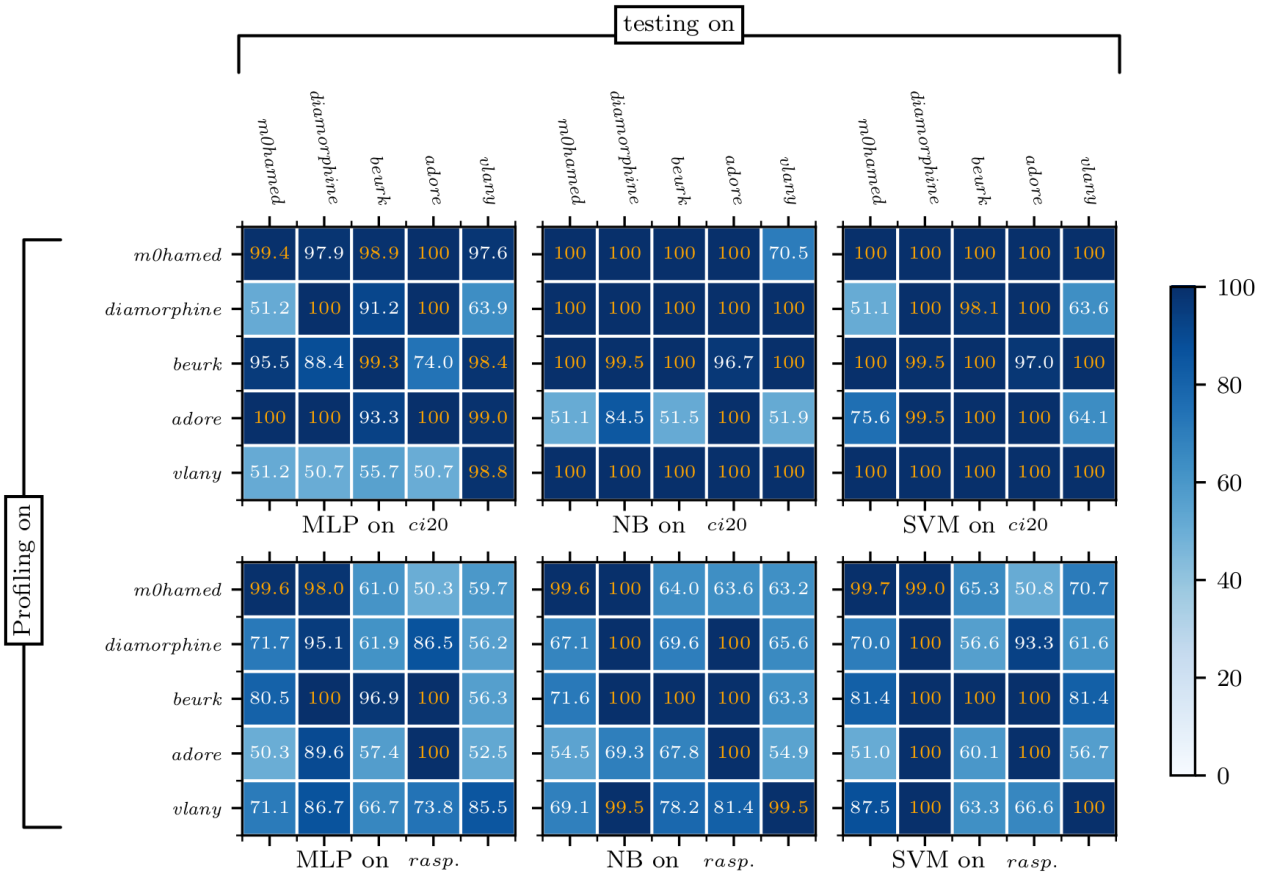

Figure 5: Novelty rootkit detection. Each cell refers to an experiment, the row (resp. the column) informs on the rootkit(s) seen during the offline learning phase (resp. the online testing phase). Except on the diagonal, learning and testing sets are exclusive. Numbers are balanced accuracies, the darker the blue color the higher the balanced accuracy. All experiments share the same bait *getdents*, and benign activities are drawn randomly. CI20 on top; Raspberry Pi on bottom

References

[1] D.-P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification," in *Annual Computer Security Applications Conference (ACSAC)*, 2021.

[2] D.-P. Pham, D. Marion, and A. Heuser, "ULTRA: Ultimate Rootkit Detection over the Air," in *25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022.

[3] K. Riad, T. Huang, and L. Ke, "A dynamic and hierarchical access control for IoT in multi-authority cloud storage," *Journal of Network and Computer Applications*, vol. 160, p. 102633, 2020.

[4] V. Adat and B. B. Gupta, "Security in Internet of Things: issues, challenges, taxonomy, and architecture," *Telecommunication Systems*, vol. 67, no. 3, pp. 423–441, 2018.

[5] V. Rey, P. M. Sánchez Sánchez, A. Huertas Celdrán, and G. Bovet, "Federated Learning for Malware Detection in IoT Devices," vol. 204, p. 108693, 2021. Accessed on 2022-01-14.

[6] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, and K. Fu, "WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices," in *2013 USENIX Workshop on Health Information Technologies (HealthTech 13)*, (Washington, D.C.), USENIX Association, Aug. 2013.

[7] N. Sehatbakhsh, A. Nazari, M. Alam, F. Werner, Y. Zhu, A. Zajic, and M. Prvulovic, "REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 312–326, 2020.

[8] I. Kyte, P. Zavarsky, D. Lindskog, and R. Ruhl, "Enhanced side-channel analysis method to detect hardware virtualization based rootkits," in *World Congress on Internet Security (WorldCIS-2012)*, pp. 192–201, 2012.

[9] P. Luckett, J. T. McDonald, W. B. Glisson, R. Benton, J. Dawson, and B. A. Doyle, "Identifying stealth malware using CPU power consumption and learning algorithms," *Journal of Computer Security*, vol. 26, no. 5, pp. 589–613, 2018.

[10] R. Bridges, J. H. Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell, "Towards malware detection via cpu power consumption: Data collection design and analytics," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 1680–1684, IEEE, 2018.

[11] F. Ding, H. Li, F. Luo, H. Hu, L. Cheng, H. Xiao, and R. Ge, "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*, pp. 33–46, 2020.

[12] X. Wang, Q. Zhou, J. Harer, G. Brown, S. Qiu, Z. Dou, J. Wang, A. Hinton, C. A. Gonzalez, and P. Chin, "Deep learning-based classification and anomaly detection of side-channel signals," in *Cyber Sensing 2018*, vol. 10630, p. 1063006, International Society for Optics and Photonics, 2018.

[13] A. Baliga, V. Ganapathy, and L. Iftode, "Detecting Kernel-Level Rootkits Using Data Structure Invariants," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 670–684, 2011.

[14] N. L. Petroni Jr, T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot-a Coprocessor-based Kernel Runtime Integrity Monitor.," in *USENIX security symposium*, pp. 179–194, San Diego, USA, 2004.

[15] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage," in *International Symposium on Electromagnetic Compatibility (EMC '14 / Tokyo)*, IEEE, May 12-16 2014. eprint version: `https://eprint.iacr.org/2013/717.pdf`.

Annexe

Binaries names	#	Types	Family	Virtualization	Packer	Obfuscation	Executable	Novelty (family)
random34	6000	benign	benign				random34	benign
mirai.arm7	6000	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	3000	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	3000	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	3000	ddos	bashlite			sub	bashlite_sub	bashlite [*]
bashlite_virtualize	3000	ddos	bashlite	virtualized		virtualize	bashlite_virtualize	bashlite [+]
playaudio	1000	benign	benign				playaudio	benign
recordcamera	1000	benign	benign				recordcamera	benign
takepicture	1000	benign	benign				takepicture	benign
encodevideo	1000	benign	benign				encodevideo	benign

Table 3: 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).

	RK	Baits									Activities				
		System calls							*Net.*	*Key.*					
		getdents	readir	open	kill	read	write	stat	renameat	tcp	emu	H	N	L	B	R
kernel	diamorphine	✓			✓							✓				✓
	diamorphine(*)	✓			✓							✓				✓
	m0ham3d	✓		✓		✓	✓					✓	✓			✓
	m0ham3d(*)	✓		✓		✓	✓					✓	✓			✓
	adore-ng	✓	✓	✓						✓		✓	✓		✓	✓
	spy										✓			✓		
	maK_it										✓			✓		
user	beurk	✓	✓	✓				✓		✓		✓	✓		✓	
	vlany	✓	✓	✓				✓	✓	✓		✓	✓			✓

Table 4: Input baits, that handled by system calls, network activities (*Net.*) and keyboard emulator (*Key.*), targeting rootkit (RK) variants including obfuscated variants(*). List of RK activities: (H) Hide file/module/process; (N) Hide network port/socket; (L) Keylogger; (B) Remote access trojan; (R) LPE.