International Journal of Information Security (2023) 22:541–561
https://doi.org/10.1007/s10207-022-00602-w

SPECIAL ISSUE PAPER

Intelligent and behavioral-based detection of malware in IoT spectrum sensors

Alberto Huertas Celdrán[1] · Pedro Miguel Sánchez Sánchez[2] · Miguel Azorín Castillo[2] · Gérôme Bovet[3] · Gregorio Martínez Pérez[2] · Burkhard Stiller[1]

Published online: 29 July 2022

Abstract

The number of Cyber-Physical Systems (CPS) available in industrial environments is growing mainly due to the evolution of the Internet-of-Things (IoT) paradigm. In such a context, radio frequency spectrum sensing in industrial scenarios is one of the most interesting applications of CPS due to the scarcity of the spectrum. Despite the benefits of operational platforms, IoT spectrum sensors are vulnerable to heterogeneous malware. The usage of behavioral fingerprinting and machine learning has shown merit in detecting cyberattacks. Still, there exist challenges in terms of (i) designing, deploying, and evaluating ML-based fingerprinting solutions able to detect malware attacks affecting real IoT spectrum sensors, (ii) analyzing the suitability of kernel events to create stable and precise fingerprints of spectrum sensors, and (iii) detecting recent malware samples affecting real IoT spectrum sensors of crowdsensing platforms. Thus, this work presents a detection framework that applies device behavioral fingerprinting and machine learning to detect anomalies and classify different botnets, rootkits, backdoors, ransomware and cryptojackers affecting real IoT spectrum sensors. Kernel events from CPU, memory, network, file system, scheduler, drivers, and random number generation have been analyzed, selected, and monitored to create device behavioral fingerprints. During testing, an IoT spectrum sensor of the ElectroSense platform has been infected with ten recent malware samples (two botnets, three rootkits, three backdoors, one ransomware, and one cryptojacker) to measure the detection performance of the framework in two different network configurations. Both supervised and semi-supervised approaches provided promising results when detecting and classifying malicious behaviors from the eight previous malware and seven normal behaviors. In particular, the framework obtained 0.88–0.90 true positive rate when detecting the previous malicious behaviors as unseen or zero-day attacks and 0.94–0.96 $F1$-score when classifying them.

Keywords IoT · Device behavior fingerprinting · Malware · Spectrum sensor · Machine learning

1 Introduction

The Internet-of-Things (IoT) growth reports an increment of Cyber-Physical Systems (CPS) connected to the Internet and available in the Industry 4.0 [1]. Some of the reasons influencing this trend are recent advances in technology, reduced cost of resource-constrained devices, and scenarios that the IoT brings to reality. In terms of scenarios, crowdsourcing is one of the most interesting ones since it combines a multitude of resource-constrained and physical devices sensing and sharing data with third parties. Different use cases within

✉ Alberto Huertas Celdrán
huertas@ifi.uzh.ch

Pedro Miguel Sánchez Sánchez
pedromiguel.sanchez@um.es

Miguel Azorín Castillo
miguel.azorinc@um.es

Gérôme Bovet
gerome.bovet@armasuisse.ch

Gregorio Martínez Pérez
gregorio@um.es

Burkhard Stiller
stiller@ifi.uzh.ch

[1] Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, 8050 Zurich, Switzerland

[2] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain

[3] Cyber-Defence Campus, Armasuisse Science and Technology, 3602 Thun, Switzerland

the Industry 4.0 umbrella are enabled by crowdsourcing platforms, being radio frequency spectrum sensing one of the most promising ones due to the scarcity of the radio frequency spectrum [2] and the increment of cyberthreats jamming or interfering critical radio transmissions [3]. In such a context, ElectroSense [4] is an IoT crowd-sensing platform that senses radio frequency spectrum in populated regions of the world and makes data available in real-time. Thus, countless resource-constrained sensors, implemented in Raspberry Pis or single-board devices, are deployed worldwide to collect spectrum data and send them to a backend platform. After analyzing and processing these data in the backend, different services, such as spectrum monitor or spectrum decoder, are provided through the ElectroSense Web site [5].

Despite the advantages of platforms using resource-constrained devices to sense the spectrum, they are vulnerable to cyberattacks. Vulnerabilities are influenced by the poor security mechanisms implemented in sensors due to a lack of resources or short development time. In this sense, cyberattacks affecting spectrum sensors in general, and ElectroSense sensors in particular, can be launched by different types of malware. Still, botnets [6], rootkits [7], backdoors [8], ransomware [9], and cryptojackers [10] have been highlighted as some of the most harmful families. Botnets are particularly interesting for crowdsourcing scenarios due to the number of devices and the possibility of recruiting them as zombies to launch Distributed Denial-of-Service (DDoS) attacks. Rootkits are also very powerful, because of their capabilities to hide unauthorized or illegal activities and to pivot to other devices in a subsequent phase. Backdoors allow unauthorized users to access devices and systems to control the device functionality and steal sensitive data or cryptographical material managed by spectrum sensors. Ransomware encrypts critical files asking for money to recover them. Finally, cryptojackers exploit the device processing resources to generate cryptocurrency and extract benefits from the infected devices. In summary, the previous five families of malware are some of the most dangerous affecting IoT Spectrum sensors due to their impact not only on the sensors availability and integrity, but also on the privacy, confidentiality, and integrity of sensitive data [11].

The literature has proposed different mechanisms and approaches to detect botnets, rootkits, backdoors, ransomware, cryptojackers, and other malware affecting different devices. On the one hand, classic static analysis approaches focus on detecting well-known malware signatures. However, signature-based approaches are useless against novel malware samples or small variations of the existing ones. To improve these drawbacks, dynamic analysis methods, such as behavioral fingerprinting [12], have gained popularity to detect unseen or zero-day attacks. This technique consists of monitoring the device behavior with the hypothesis that the data collected should allow differentiating between the “normal" behavior of the device and the behavior, when it is infected by malware. In this sense, Machine and Deep Learning (ML/DL) techniques play a key role in achieving this goal and obtaining a promising performance while reducing the false positive rate. In particular, the lifecycle of ML/DL-based device behavioral fingerprinting starts with the data collection process, when the device is working normally (assuring that it is not attacked). This data is used to train ML/DL-based models. If the detection mechanism pretends to detect anomalies produced by unseen attacks (zero-day attacks), only the normal behavior is needed to train the models. However, if the detection approach follows a classification approach, the behavior under attack must also be modeled and labeled properly during the training phase. Finally, the device is infected with malware during a testing phase, and the behavior under attack is evaluated against previous models.

Behavioral fingerprinting benefits from the simplicity, repetitive behavior, and reduced functionality of devices. However, in the field of IoT Spectrum sensing, the following open challenges need more investigation effort.

- There is a lack of related work analyzing the suitability and performance of device fingerprinting techniques to detect attacks affecting IoT Spectrum sensors. Existing literature focuses on network traffic analysis to detect IoT malware. However, it is not always possible in crowdsensing platforms where sensors are connected to private networks. Furthermore, some malware behaviors are not detectable through the network.
- There is a lack of solution considering and evaluating the suitability of the usage of kernel events as data source to create stable and precise fingerprints for IoT devices in general and spectrum sensors in particular.
- There is a lack of fingerprinting solutions designed and evaluated in real crowdsensing platforms that use resource-constrained and single-board sensors affected by different families of recent malware.
- Most of the fingerprinting solutions are focused on generic devices without limitations in terms of data sources and resources. These approaches have not been evaluated (and might not be deployed) on IoT spectrum sensors because selected data sources and events are not available in such devices.
- There is a need for fingerprinting solutions for IoT evaluating their detection performance with recent and IoT-oriented malware families, such as botnets, rootkits, backdoors, ransomware, and cryptojackers.
- There is a necessity of comparing the detection performance of fingerprinting solutions detecting anomalies and classifying recent malware affecting IoT spectrum sensors under different network conditions.

Therefore, the key contribution of this work is the design and implementation of an intelligent and modular detection framework that uses device behavioral fingerprinting and ML to detect anomalies and classify different botnets, rootkits, backdoors, ransomware, and cryptojackers affecting IoT spectrum sensors. The framework has been implemented as a Proof-of-Concept (PoC) composed of two main modules. The first module is deployed on the resource-constrained spectrum sensor that will be secured. It monitors kernel events, from seven different data sources (CPU, memory, network interface, file system, scheduler, random number generation, and device drivers). The second module is allocated on an external server and is in charge of (i) pre-processing behavioral data, (ii) training ML-based models for anomaly detection and classification tasks, and (iii) evaluating them to detect cyberattacks. ElectroSense has been selected as a crowdsensing platform to validate the usefulness of the proposed framework. In particular, a Raspberry Pi 4 acting as an ElectroSense sensor has been infected with two botnets (Bashlite [13] and Mirai [14]), three rootkits (Diamorphine [15], Beurk [16], and Bdvl [17]), three backdoors (HttpBackdoor [18], Backdoor [19], and TheTick [20]), one Ransomware (Ransomware_PoC [21]) and one Cryptojacker (Linux.MulDrop.14 [22]). Finally, two experiments were performed to evaluate the detection performance of the framework from anomaly detection and classification prisms and with different network configurations (with more and less network packet overhead and stability). Promising results across these experiments prove that the main contribution of this work, the framework, is valid and applicable in spectrum sensors.

The remainder of this paper is organized as follows. While Sect. 2 reviews solutions using device behavior fingerprinting to detect cyberattacks, Sect. 3 provides key design details of the ML-oriented framework detecting malware affecting spectrum sensors. Based on Sect. 4, depicting the scenario and conditions where the proposed solution will be deployed and tested, Sect. 5 outlines the deployment of the proposed framework and its performance experiments. Then, Sect. 6 summarizes and discusses the advantages and limitations of the proposed framework. Finally, Sect. 7 draws conclusions and outlines next steps.

2 Related work

Device fingerprinting and its applicability to malware detection was proposed for a few years in the scientific community. Table 1) summarizes the main aspects of the reviewed solutions, which details are provided below.

One research area of device fingerprinting focuses on the usage of Hardware Performance Counters (HPCs) to detect cyberattacks affecting devices. In this context, the authors of [24] focused on modern microprocessors and demonstrated the importance of ML Classifiers and the trade-off between the type and number of HPCs, and the malware detection performance. The authors proposed an ensemble learning classifiers to boost the performance of general ML classifiers. Results when testing against 100 malware from VirusTotal showed that *boosting* achieved a performance improvement of up to 17%. [25] trained Hidden Markov Models and Long Short Term Memory neural networks (LSTM) with HPCs of embedded devices for classification, offline anomaly detection, and online anomaly detection tasks. The authors achieved promising detection results with both algorithms and detection scenarios. Cronin and Yang [26] proposed a solution detecting malware in mobile devices. In such a context, HPCs modeling the entire device behavior were modeled using Multi-layer Neural Networks (MLP). In particular, 25 malware (trojans, adware, and kalfere) from VirusShare were used to test the performance with known and zero-day attacks. Pudukotai Dinakarrao et al. [27] combined the ML-based malware identification with a confinement policy, where devices suspected of being infected are separated from the rest of the network. The authors reported 87% accuracy. Only few approaches in the literature discuss APIs for accessing HPCs, e.g., [29]. In particular, the authors criticized other approaches, exposing and explaining selected flaws. They concluded with experiments to prove inter-process noise impact on performance and non-determinism in HPC measurements, which are the two main problems highlighted. Kadiyala et al. [30] address the previously mentioned non-determinism by applying ANOVA (Analysis of Variance) on available hardware and hardware cache events per system call. Later, this solution applies ML-based classification algorithms to identify 332 ARM-based malware from VirusShare and OpenMalware. From another perspective, [28] combined the use of HPCs with other sources of data such as memory dumps as greyscale images and API call sequences to classify malware.

System calls are another type of data source considered in behavior fingerprinting to detect cyberattacks affecting heterogeneous devices. De Lorenzo et al. [32] presented VizMal, a solution using system call invocations and images with boxes to detect malware on Android. Each box represents one execution interval, with its color indicating the maliciousness of that interval, and the size is the activity level in terms of system calls. After training a Long Short-Term Memory (LSTM) Neural Network with samples labeled as malware or non-malware, the results showed a promising direction toward malware detection. VMGuard [33] is another intelligent security architecture that used system calls and ML to detect malware in a cloud scenario with Virtual Machines (VM). VMGuard monitors the processes and system calls of VMs to create a 'Bag of n-grams (BonG)' integrated with the Term Frequency-Inverse Document Frequency (TF-IDF)

Table 1 Comparison of work (sorted by feature type and publication date) using behavioral fingerprinting to detect malware affecting heterogeneous types of devices

Work	Scenario	Feature type	Modeled entity	Post-processing	Approach	Algorithm	Malware families	Analysis	Performance
[23] (2018)	Embedded device	HPC	Program	ML classifier	Sup.	SVM, MLP, Bayes Net, J48	Rootkit, Backdoor, Trojan	Local	Acc.: 0.61–0.93
[24] (2018)	Computer/Server	HPC	Program	Ensemble Learning	Sup.	Boosting, Bagging	100 Unspecified	Local	AUC: 0.94
[25] (2018)	Embedded device	HPC	Program	ML & Statistical	Unsup.	LSTM & HMMs	Run-time modifications	Local	Acc.: 0.98
[26] (2018)	Mobile device	HPC	Device	ML classifier	Sup.	Multi-layer Neural Net.	Trojan, Adware, Kalfere	Remote	Acc.: 0.87
[27] (2019)	IoT	HPC	Device	ML classifiers	Sup.	MLP, OneR, Logistic, Jrip	Backdoor, Rootkit, Virus, Trojan	Local	Acc.: 0.92
[28] (2019)	Computer/Server	HPC, Mem. dump & API calls	Program	ML classifiers	Sup.	Multi-layer BiGRU, RF, MLP, VGG19	9 families	Local	Acc.: 0.98
[29] (2019)	Computer/Server	HPC	Program	ML classifier	Sup.	J48, IBk, SMO	137 families	Local	F1: 0.89
[30] (2020)	Embedded device	HPC	System call	ML classifier	Sup.	DT, RF, NN, KNN	Backdoor, Exploit, Flooder, Trojan, Virus	Remote	F1: 0.99
[31] (2017)	Computer/Server	Syscalls	System call routines	Statistical	Unsup.	MISR, Bloom filter	Rootkits	Local	Acc.: 1.00
[32] (2020)	Mobile Device	Syscalls	Device	ML Classifier	Sup.	SVM, MIL, LSTM	Android Malware	n/a	FPR: 0.09, FNR: 0.55
[33] (2020)	Computer/Server	Syscalls	System Call	ML Classifier	Sup.	RF, NB, SVM	Intrusive System Call Traces	Local	Acc: 0.94–1.00
[34] (2018)	Computer/Server	Resource Usage	Device	ML Anomaly Detection	Unsup.	AR	DDoS	Local	MSE: 4.95×10^{-4}
[35] (2018)	Computer/Server	Resource Usage	Device	ML Anomaly Detection	Unsup.	K-Means, SVM, SOM	DDoS & Cryptominer	Local	Acc: 0.90–0.95
[36] (2018)	IoT	CPS Sensor Data	Device	Rule-based	Sup.	State machine	Physical misbehavior	Local	TPR: 1.00, FPR: 0.07

Table 1 continued

Work	Scenario	Feature type	Modeled entity	Post-processing	Approach	Algorithm	Malware families	Analysis	Performance
[37] (2019)	IoT	CPS Sensor Data	Device	Fuzzy logic	Sup.	HCAPN	Vulnerability Exploitation (DoS, Spoofing, etc.)	Local	TPR: 0.98, FPR: 0.40, FNR: 0.22
[38] (2020)	IoT	Kernel Events	Device	ML Classifier	Sup.	CNN	IoT Malware	Remote	Acc. 0.98
[39] (2021)	Computer/Server	Kernel Events	Device	ML Classifier	Sup.	SVM	43 Ransomware Families	Local	F1: 0.99
[40] (2018)	Computer/Server	Kernel events	Task & subtasks	ML classifiers	Sup.	KNN, SVM, ANN, KDE	Rootkit	Local	Acc.: 0.99
[41] (2019)	Computer/Server	Kernel Events	Device	ML Classifier	Sup.	RF & SVM	Common Attack Malware Patterns	Local	Acc: 0.99
[42] (2022)	IoT Spectrum Sensors	Kernel Events	Device	ML Anomaly Detection	Unsup.	Autoencoder	Spectrum Sensing Attacks	Local	TPR:0.6–1 TNR:0.9–0.98
[43] (2022)	IoT Spectrum Sensors	Kernel Events	Device	ML Classifier and Anomaly Detection	Unsup./Sup.	Autoencoder, MLP	Spectrum Sensing Attacks	Local	TPR:0.6–1 TNR:0.9–0.98
This Work	IoT (IoT Spectrum Sensors)	Kernel Events	Device	ML models	Sup./Unsup.	RF, SVM, KNN, GNB, XGB, DT/LOF, IF, OC-SVM	Botnets, Rootkits, Backdoors, Ransomware, Cryptojackers	Remote	F1 (Sup.): 0.94–0.96, TNR: (Unsup.): 0.94–0.96, TPR: (Unsup.): 0.82–0.90

method. The results obtained by a Random Forest demonstrate the applicability of VMGuard to classify intrusions. Zhou and Makris [31] proposed a system to detect intrusions in microprocessors. The validity of system call routines executed in response to a system call was evaluated using a Bloom filter. Perfect detection performance was reached with five different rootkits. Finally, [40] presented a hardware solution to achieve better security and performance in general computers. They considered hardware events and used different ML classifiers to obtain 99% detection accuracy.

The usage of internal resources is another data source used to create fingerprints. In this sense, [34] presented an anomaly detection mechanism that detects Denial-of-Service (DoS) attacks affecting cloud scenarios. This work used CPU resource statistics of micro-services running on cloud architectures and auto-regressive statistical models to detect DoS attacks. The authors concluded that the models accurately detect anomalous behaviors for applications with cyclical trends. RADS (Real-time Anomaly Detection System) [35] is another work that used resource usage fingerprinting to detect DDoS attacks affecting cloud data centers. This solution considered one class classification algorithm and a window-based time series analysis to detect VM-level anomalies due to DDoS and crypto-mining attacks. Evaluation results demonstrated that RADS achieves 90-95% accuracy with a low false-positive rate of 0-3%.

Kernel events are another data source of behavioral fingerprinting used to detect attacks. The authors of [39] proposed Peeler, a novel ransomware detection system relying on behavioral characteristics such as stealth operations performed before the attack, file I/O request patterns, process spawning, and correlations among kernel-level events. The experimental results of Peleer achieved more than 99% detection rate with 0.58% false-positive rate against 43 distinct ransomware families. AIDIS [41] is an Advanced Intrusion Detection and Interpretation System capable of explaining anomalous behavior within a network-enabled user session by considering kernel event anomalies identified through their deviation from a set of baseline process graphs. The authors implemented anomaly classification through a set of competency questions applied to graph template deviations and evaluated the approach using both Random Forest and linear kernel support vector machines. Finally, DAIMD [38] is a solution for IoT that monitors memory, network, virtual file system, process, and system calls to detect IoT Malware. Using this data, DAIMD creates images used to train a convolution neural network (CNN) model to classify images into benign and malicious. Although not focused on malware, [42] and [43] worked on the detection of spectrum sensing attacks in IoT spectrum sensors using kernel events. In [42], the authors employed an autoencoder, achieving 0.6–1 TPR and 0.9–0.98 TNR. In contrast, [43] applied a robust Federated Learning perspective both for classification and anomaly detection.

Regarding rule-based intrusion detection in IoT, [36] proposed the generation of a state-machine during the testing and debugging phase of Medical IoT devices. In other work [37], the same authors improved their previous proposal by adding a 2-layer Fuzzy-based hierarchical context-aware aspect-oriented model for misbehavior detection. However, these solutions are focused on the sensor/actuator parts of a CPS and not on the running software processes and operating system.

As one of the main conclusions from this review, there is no fingerprinting solution detecting cybersecurity issues in IoT Spectrum sensors, which is the main contribution of this work. Furthermore, there is no solution analyzing the suitability of ML techniques to detect recent malware samples affecting IoT spectrum sensors. Also, it can be appreciated that there is a preference for solutions applicable to computers, servers, and devices without computational limitations. However, the suitability of these solutions for resource-constrained IoT devices is not analyzed. Besides, there is a need for solutions calculating the performance of fingerprinting when detecting malware in real IoT platforms, a key factor that motivates the present work.

3 Framework design

The main objective of this work is to create a framework that applies device fingerprinting and ML to detect heterogeneous malware affecting IoT spectrum sensors. In such a context, the design details of the proposed framework address single-board spectrum sensors, but they are generic enough to cover other devices with similar capabilities. A general diagram of the proposed solution is depicted in Fig. 1, which covers the behavioral data acquisition (implemented in the spectrum sensor) to the analysis and detection tasks (implemented in a server). At this point, it is important to mention that the reaction module is out of the scope of this work.

The main functionality of these framework modules can be summarized as follows:

- *Data gathering* It periodically monitors different behavioral-based kernel events of the CPU, virtual memory, file system, network interface, scheduler, device drivers, and random number generation families.
- *Data sending & data receiving* These two modules are deployed on the sensor and server, respectively, allowing the sensor to send behavioral data to the server and the server to receive those.
- *Data pre-processing* It removes constant and noisy behavioral data that does not provide relevant informa-

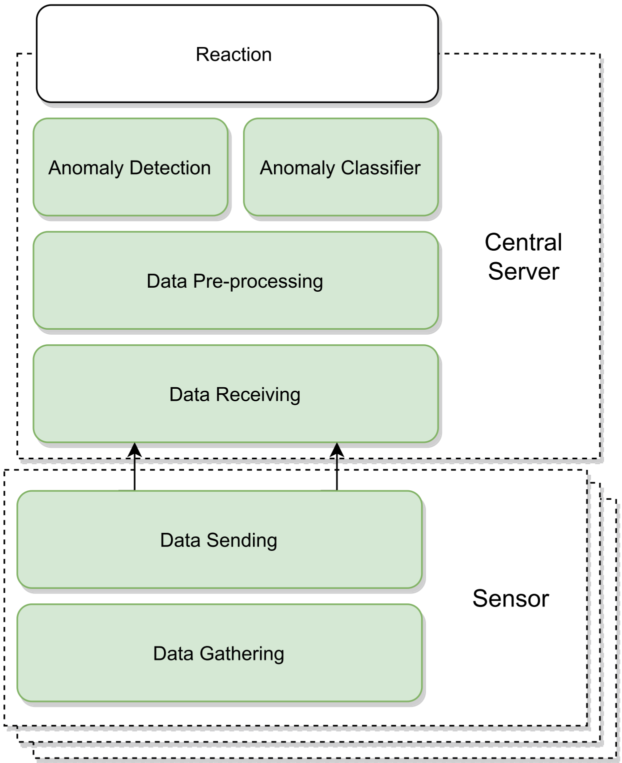


Fig. 1 Framework design

tion. It also normalizes data and extracts any additional features.

- *Anomaly detection & anomaly classifier* These two modules are focused on detecting anomalies produced by zero-day attacks, and classifying well-known malware samples. Both modules implement the training and evaluation functions of a typical ML-based detection system. Their main differences rely on the data used for training, and the ML algorithms. In terms of data, the anomaly detection approach (semisupervised) is trained only with normal data to enable the detection of any malware (zero-day attack) affecting the device behavior. Regarding the classification approach (supervised), both normal and under-attack data are used during training.

3.1 Data gathering

The selection and acquisition of events belonging to different data sources are essential aspects of solutions based on behavioral fingerprinting, as they heavily influence the system detection performance.

In this work, the main criteria to select the final list of events was to cover every device area while keeping the events as generic as possible (e.g., a network usage event is more general than a TCP usage one). Covering every device area such as network, memory, or CPU helps generate a fingerprint more complete and adaptable to zero-day attacks. In this sense, a solution detecting rootkits due to CPU usage but missing network data might be useless against botnets even if the network events are highly affected. The reason behind the preference for general events is the reduction of specific or low-level events tracked while maintaining as much behavioral information as possible. It also improves compatibility across heterogeneous devices. Finally, it was considered as not a good practice to detect zero-day attacks the fact of building the list of monitored events according to the behavioral analysis of a reduced number of attacks affecting the sensor (as done in many works of the literature), since the behavior of other families is forgotten. Considering all the previous aspects, the proposed framework built a list of kernel events belonging to the following seven families of data sources: CPU, virtual memory, network, file system, scheduler, device drivers, and random number generation.

3.2 Data sending and receiving

On the one hand, the Data Sensing module is deployed on the spectrum sensor, and it is in charge of periodically sending the behavioral data acquired by the previous module. On the other hand, the Data Receiving module is allocated in a server (or servers) and receives the previous data for subsequent data pre-processing and analysis. The data may be sent to a single centralized server that concatenates all available data. Still, it is also possible to implement the platform in a distributed way by deploying several servers on the Edge and interconnecting them to share data and the generated models. In this way, processing time can be reduced, bringing the processing closer to the sensors and reducing network latency.

3.3 Data pre-processing

The Data Pre-processing module filters erroneous data, aggregates, and normalizes the data sent from the spectrum sensors. After performing these tasks, the datasets with sensor behavioral data are ready for the model training. In addition, this module is in charge of performing possible operations to extract additional features to those sent by the sensors, such as applying statistical or time series techniques. It is worthy to note that this module works both for the generation of the behavioral dataset and for evaluating the real-time behavioral data that need to be detected as normal or not.

3.4 Anomaly detection and classification

The proposed framework considers both anomaly detection and classification algorithms but for different purposes. Once the Anomaly Detection module detects an anomaly, the data is sent to the Anomaly Classifier, which infers the type of attack, if it has been previously seen. It could be done as precisely as to recognize the specific malware or just its family or any other information that might be helpful. No specific algorithms for any of the two approaches are chosen from the design point of view because the best-performing ones might vary from a scenario to another. As has been extensively demonstrated in the literature, in general, the performance of anomaly detectors based on ML is worse than well-trained classifiers. For this reason, choosing the sooner for the most critical decisions (deciding if the device is under an anomaly is considered more critical than inferring the type of anomaly) might seem counter-intuitive. Nonetheless, anomaly detectors are more realistic as the first detection mechanism in real environments with zero-day attacks.

3.5 Reaction

The Reaction module is in charge of triggering the necessary actions to mitigate the attacks occurring in the sensors. Despite the functionality and implementation of this module is out of the scope of this work, it would be able to take diverse actions ranging from the removal of suspicious software running on the sensor to the isolation or shutdown of sensors.

4 Scenario: ElectroSense sensors

This section introduces ElectroSense, the real crowd-sensing platform used to evaluate the detection capabilities of the proposed framework. After that, it provides the details of the scenario setup as well as the malware families and samples that have been used to infect a real ElectroSense sensor and test framework suitability.

4.1 ElectroSense description and scenario setup

ElectroSense is a crowd-sourcing initiative for collecting radio frequency spectrum data worldwide and making it accessible for any interested user. To achieve it, a multitude of resource-constrained spectrum sensors are deployed across the world. The sensors implementation relies on Raspberry Pis running open-source code and connected to Software Defined Radio (SDR) kits with an antenna to measure the spectrum. Sensors periodically send spectrum data to a back-end platform in charge of controlling some functionalities of sensors are well as collecting spectrum data. Furthermore, the back-end platform provides a website where end-users can monitor the spectrum occupancy and consume services decoding the following seven transmissions: FM and AM Radio, Automatic Dependent Surveillance-Broadcast (ADS-B), Automatic Identification System (AIS), Aircraft Communication Addressing and Reporting System (ACARS), and Long-Term Evolution (LTE).

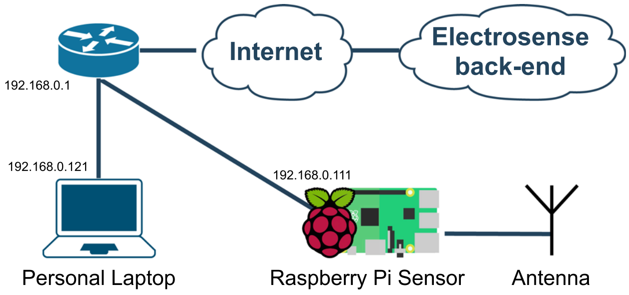


Fig. 2 Scenario used for experiments

In such a platform, this work has deployed a fully operative ElectroSense sensor with the following main hardware and software specification:

- Raspberry Pi 4 Model B.
  - CPU: Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
  - RAM: 2GB LPDDR4-3200 SDRAM.
  - Gigabit Ethernet.
  - Raspbian with spectrum sensing functionality, available in [5].
- RTL-SDR Blog V3 R820T2 RTL2832U.

The sensor has been connected to the internet through a local area network (LAN) which topology can be seen in Fig. 2. In particular, the LAN is composed of a router (whose IP address is 192.168.0.1), the ElectroSense sensor (Raspberry Pi with IP address: 192.168.0.111) connected the SDR kit and antenna, and a laptop (IP address: 192.168.0.121) that acts as command and control (C&C), or any other entity needed to run botnets ,rootkits, and backdoors, e.g., Mirai needs an extra *Report Server* and *Loader*.

4.2 Malware affecting ElectroSense sensors

As already stated, IoT-based scenarios such as ElectroSense are very interesting for malicious parties due to the lack of security implementation, the number of devices interconnected, the broad scope they constitute, and their usefulness for pivoting to others. Although every malware type is relevant when protecting the ElectroSense platform, rootkits, botnets, backdoors, ransomwares, and cryptojakers are the most representative and dangerous for ElectroSense. Rootkits are perfect for hiding viruses and malicious behaviors. Botnets take advantage of the number of ElectroSense sensors to launch distributed denial of service (DDoS) attacks. Backdoors are suitable for controlling sensors functionality and stealing sensitive spectrum data. Ransomware is perfect for getting money from ransoms required to decrypt data. Finally, cryptojakers have been proved dangerous for Raspberry Pis because they can use the CPU to mine cryptocurrencies such as Monero.

4.2.1 Rootkits

Rootkits are a type of malware that allows the control of a system by a malicious party while remaining hidden. Their main features are related to obfuscation and stealth. Two different types of rootkits are available for Linux-based systems: LD_PRELOAD and Loadable Kernel Module (LKM) based. To better understand the differences between the two types, a brief explanation is given below:

- LD_PRELOAD is an optional environmental variable containing one or more paths to shared libraries or objects that the loader will load before any other shared library. It fits perfectly into the functionality of a rootkit since custom shared object libraries can easily be used to hijack library calls in the specific binary running. These rootkits, also referred to as *preload* rootkits, operate in the user-space. Instead of using the environmental variables, they take advantage of a global file to make the loader preload the shared object library into every process it starts, including services and daemons that run as root. Their most significant limitation is that it only works for dynamically loaded ELF binaries.
- LKM rootkits are loaded as any other module into the kernel, which grants them the same execution privileges as any other part of the code integrated into it. It means they operate in kernel-space. While the objective of user-space rootkits is to intercept calls from binaries to libraries, a kernel-space rootkit tries to intercept system calls. This can be done by syscall hooking or by interrupt hooking. Those rootkits have two main drawbacks. First, high privileges are needed to add a kernel module, and second, they heavily depend on the kernel version and the underlying architecture of the device.

Among the publicly available rootkits affecting ARM-based architectures, Beurk, Bdvl and Diamorphine have been selected to infect the ElectroSense sensor of the scenario presented in Fig. 2. More in detail, the first two belong to the LD_PRELOAD class, and the third one to the LKM one. A brief summary of them is given below:

- Beurk [16]. Preload rootkit heavily focused on anti-debugging and anti-detection. Its features range from hiding pseudo-terminal backdoor clients, files, directories, and real-time log cleanup (on utmp/wtmp) to concealing processes, logins, and bypassing *unhide*, *lsof*, *ps*, *ldd* and *netstat* analysis.
- Bdvl [17]. Preload rootkit based on Vlany [44]. Some of its main objectives are to provide a more clean, robust, and manageable rootkit. Its functionality is immense, and it ranges from hidden backdoors that allow multiple connection methods to keylogging and stealing passwords and files.
- Diamorphine [15]. LKM rootkit for ARM 64b processors and relatively modern Linux-based operating system versions. Since Raspbian is 32b, this malware had to be modified to make it compatible. The module starts invisible when loaded, and its functionalities are (i) sending any process the signal 31 to hide/unhide it, (ii) the signal 63 to make the module invisible, and (iii) the signal 64 to gain execution root privileges as well as hiding files or directories that contain a specific prefix.

4.2.2 Botnets

Botnets are another type of malware that allows the control of the device by an attacker. There are many differences between a botnet and a rootkit. One of them is that the botnet only infects devices connected to a network, generally with lower capabilities and looser security implementations. Another difference is that botnets try to infect as many devices as possible, and their running privileges do not need to be exceptionally high since they are usually utilized to perform Distributed Denials of service (DDoS).

The most basic architecture of a botnet comprehends the devices infected (bots) and the entity controlling and sending malicious commands across all infected devices (C&C). Below, the details of the two botnet samples used in this work to infect the ElectroSense sensor are provided.

- Bashlite [13]. Its original version dates from 2014, and in the next 2 years, it gained popularity, infecting over one million devices by 2016 [45]. The botnet is composed of two files: “client.c” and “server.c”, the first is the payload needed to infect the device, and the second is the C&C to control every bot. Regarding the functionality, bots can spread the payload to other devices, brute-forcing weak telnet credentials, which helps grow the botnet to the point that DDoS is possible.
- Mirai [14], which is a much more sophisticated botnet that was supposedly first found in 2016 and is well-known for being used for one of the largest and most disruptive DDoS attacks [46]. Along with the expected botnet behavior, it also has a “worm” or “spreading” module, depicted in steps 1–4 of Fig. 3. In those steps, the bot tries to spread to other devices by brute-forcing weak telnet credentials and, if successful, it reports back to the *Report Server*, who commands the loader to infect that new device.

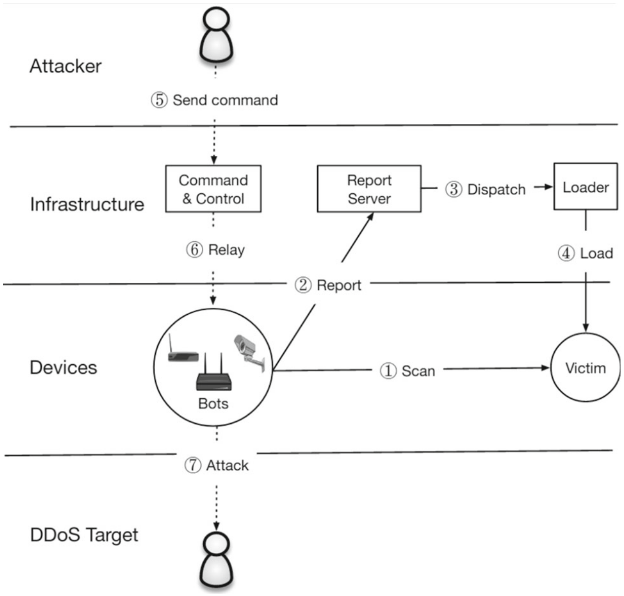


Fig. 3 Mirai behavior

4.2.3 Backdoors

Backdoors create access points in systems for unauthorized third parties. Backdoors are composed of two elements, a client and a server. Furthermore, they do not attack the device or system directly but act like a gateway for other malicious activity. In this sense, malware implementations exhibit and combine attributes from many different malware types. Backdoors also present different functionalities according to their complexity and type. In this work, we highlight the following main behaviors implemented by backdoors.

- *Execution* This behavior consists of running basic commands of the sensor.
- *Download* This behavior downloads new files and malware onto the sensor.
- *Removal* It deletes sensitive data and files on the infected sensor.
- *Data leak* This behavior leaks confidential information from the sensor.
- *DNS* resolves domain names at the sensor side.
- *Privilege* This behavior changes the access flags of sensor files.

In order to execute the previous behaviors, this work have selected the following three backdoor samples to infect the ElectroSense sensor are provided.

- HttpBackdoor [18], is the simplest and purest form of backdoor of the selected ones. It creates a Web server on the IoT spectrum sensor (Raspberry Pi) to which the attacker (client) sends HTTP requests. This backdoor implements the three first behaviors (execution, download, and removal) of the previous list and shows two basic functionalities. Firstly, the attacker can extract basic system information such as OS version and name or saved SSH keys by sending a GET request. Secondly, by sending a POST request, the attacker can execute command line commands on the Raspberry Pi.
- Backdoor [19] consists of two parts, the client and the server-side. The server-side sends commands to a specified IP address and port on which the client side (deployed on the Raspberry Pi) listens. Backdoor implements execution, download, removal, and data leak functionality. Furthermore, the most notable feature of this backdoor is the possibility of opening a shell on the IoT spectrum sensor. Additionally, the backdoor offers the functionality to pull the contents of a file on the Raspberry Pi onto the attackers' machine.
- TheTick [20] is the most complete of the selected backdoors because it implements the six behaviors indicated in the previous list. Similar to the previous backdoor, it relies on a Command & Control (C&C) server structure, with the difference that TheTick can have multiple clients connected at the same time and switch between them seamlessly. In other words, several IoT spectrum sensors could potentially be infected and act as simultaneous clients. Finally, the attacker can send commands to each client individually.

4.2.4 Ransomware

Ransomware is a malware type in charge of restricting or blocking access to a device or its data, asking for a ransom to remove such restriction. It uses cryptographic algorithms to encrypt sensitive and critical files of the device and demands a ransom to re-gain access to the encrypted files. Usually, ransomware is composed of four stages [47]. The first phase is when the infection occurs. Then, during the second phase, the ransomware generates the encryption keys or retrieves them from a central server. In the third phase, the encryption phase starts. As the name suggests, it encrypts the device files using known symmetric encryption algorithms such as AES with a randomly generated key. Once all files are encrypted, the ransomware sends a ransom note with the attacker's contact information. It usually contains the coordinates for a payment, typically a cryptocurrency wallet. After the ransom is paid, the fourth stage occurs, where the attacker sends to the victim the key needed for the data decryption.

This work has considered Ransomware_PoC to infect the ElectroSense sensor due to its functionality.

- Ransomware_PoC [48]. This ransomware has the encryption functionality of a typical one, except that it is not controlled by a C&C server. Therefore, encryption keys are embedded into the source code. Ransomware_PoC uses an AES 256-key to encrypt the content of all files contained in the Raspberry Pi. The AES 256-key is subsequently encrypted via an RSA public key, and the encryption algorithm traverses directories on the system and encrypts the content of every file with a valid extension.

4.2.5 Cryptojacker

Cryptojackers are malware allowing attackers to illicitly mine cryptocurrencies in third parties. This malware type can target different types of devices, from servers to IoT devices. GPUs are naturally more efficient when performing cryptomining tasks. However, there are still cryptocurrencies that rely on CPU mining such as Monero[49]. It makes mining accessible even to resource-constrained devices such as Raspberry Pis. Monero (also called XMR) was introduced in 2014 and is a good example of cryptocurrency mined on ARM CPUs. Although cryptojackers can execute different behaviors (some of them already considered by the previous ransomware), this work focuses on the mining task.

In particular, Linux.MulDrop.14 has been selected to infect the ElectroSense sensor since it is a real and recent cryptojacker affecting ARM architectures.

- Linux.MulDrop.14 [22]. This cryptojacker is also known as UNIX_PIMINE and appeared in 2017 to transform Raspberry Pis into mining devices. Its functionality consists of scanning the local network to establish connections via an SSH server. Once the device is infected, it writes a copy of itself in a random directory, kills competing malware, downloads the cryptominer with its dependencies, installs them, and initiates the mining process.

5 Framework implementation and experiments

This section provides the details of the framework PoC implementation in the previously defined scenario, together with a set of experiments conducted to measure its performance when detecting the previous samples of botnets, rootkits, and backdoors affecting an ElectroSense sensor.

5.1 Framework implementation

The proposed framework relies on the hypothesis that the behavioral data collected should allow ML algorithms to infer that a device has been infected by different malware. In this sense, this section pretends to demonstrate if that hypothesis is correct through implementing a PoC. Below, the implementation details of the sensor and server sides of the framework are discussed.

5.1.1 Sensor

Regarding the Data Gathering module, it was developed an script to monitor the selected kernel events. In this sense, the script uses *perf* [50], which is also one of the most used along the works reviewed. Apart from perf, some of the other options considered were *LIKWID* [51] and *PAPI* [52]. On the one hand, *LIKWID* was discarded due to the lack of official testing for ARMv7 processors. On the other hand, *PAPI* did not meet the necessities for this work since it only measures events that occur in the user-space and only tracks hardware events. In addition, *perf* also has several advantages. The first one is that it is included in the Linux kernel, and it is available in the package *linux-tools-common* for Debian-based systems such as the Raspberry Pi used by ElectroSense sensors, making it easily accessible. Another advantage is that it uses multiple sources for the measurements, combining hardware, hardware cache, Kernel PMU, software, and tracepoint events, making it perfect for this work since the initial idea was to use all of them. The command *perf* allowed to track 1149 different events in the Raspberry Pi 4 used for testing. These kernel events are divided into 80 families, except for 63 events that do not belong to any specific family. After many iterations to refine the list, it was reduced to 75 events. The reasons for the selection of events have already been explained in Sect. 3.1. Table 2 shows the families of the selected events and the monitored resources.

Regarding the use of HPCs and despite their importance in this field, they are not monitored because they did not give enough information (results showed almost no feature relevance in them). While this justification might seem in contrast to the design statement of not discarding events solely based on correlation or results, it was also taken into consideration that no device area would remain unmodelled after their removal. Once selected the list of events, the monitoring routine was implemented as a Bash script for commodity, but it is expected to be moved to C for any implementation outside of a PoC to achieve even lower overhead. As shown in Fig. 4, the monitoring window used in the PoC is of 5 s, after which *perf* needs around 6.8 s to pre-process and calculate the values of the selected 75 events. In this extra time, as well as in the connectivity test (which can take up to 1.5 s if there is no connectivity, but usually takes much shorter), there is no monitoring of the behavior of the device. The connectivity test was implemented to achieve more reliability in networks with unstable internet. Nevertheless, to the best of our knowledge, the extra time needed by *perf* to calculate the events values can only be reduced by reducing the number of events to be monitored, which impacts the solution performance.

Table 2 Monitored resources and event families

Monitored resource	Perf event families (kernel)
Network	tcp:, upd:, net:, qdisc:, skb:, sock:, fib:
Virtual Memory	writeback:, kmem:, page-faults, pagemap:
File System	jbd2:, block:, cachefiles:, filemap:
Scheduler	sched:, signal:, task:, alarmtimer:, cpu-migrations, cs
CPU	clk:, rpm:, ipi:
Device Drivers	irq:, mmc:, preemptirq:, gpio:, dma_fence:
Random Number Generation	random:

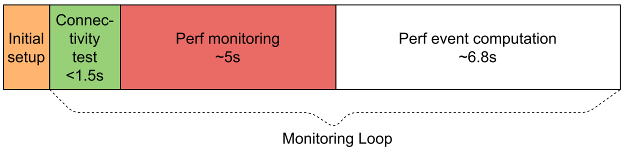


Fig. 4 Temporal view of monitoring script implemented

Parallelism was discarded to avoid noise in the monitoring window.

5.1.2 Server

From the Server perspective, once received the behavioral data from the Sensor, the Data Pre-processing module was implemented using Python. Apart from basic pre-processing (data cleaning and normalization), the datasets were balanced, allowing for a maximum of a 10% difference between them. The samples removed were selected randomly (with a constant seed to achieve repeatable results). For feature normalization, both the *StandardScaler* and *MinMaxScaler* were tested, but the latter was discarded due to worse results in every test.

In terms of ML algorithms used by the Anomaly Detection and Classifier modules, *sklearn* [53] was the library used to train and evaluate multiple algorithms. For classification, experiments were performed using Random Forest (RF), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), XGBoost (XGB) (from *xgboost* library), and Decision Tree (DT). For anomaly detection, the algorithms used were One-Class SVM, Isolation Forest (IF), and Local Outlier Factor (LOF). Table 3 shows the hyperparameters tested for each algorithm. Although the selected anomaly detection algorithms are unsupervised, since hyperparameter tuning is performed with attack data, it is considered "semi-supervised". More in detail, despite not using the abnormal data to train the algorithms, the results of the evaluation of that data are also used to decide the best configuration. More details regarding the methodology followed to train and evaluate the models are provided in the next section.

5.2 Experiments

The framework detection performance (from anomaly detection and classification perspectives) is measured through two experiments using different network configurations to obtain robust results. The specifics of each experiment are explained in the correspondent subsection, and common aspects are discussed below to avoid redundancy.

An important detail is that the five samples of the installed rootkits and botnets remain completely passive when collecting data. For rootkits, no connection or any other functionality other than the active by default is used. It is what is defined as passive behavior. In the botnet case, the sensor is infected and connected to the C&C, but no malicious command is sent or executed by the bot. It has two main advantages: (i) testing becomes much simpler and uniform, and results are easily reproducible and (ii) if the solution is able to detect passive malware behavior, it should also detect it when it is doing more work, so this would be considered testing it in a worst-case scenario. The three backdoors executed in the Raspberry Pi acting as an ElectroSense sensor are able to show the following behaviors. HttpBackdoor runs Execution, Download, and Removal behaviors; Backdoor focuses on the same three plus Data Leak, and TheTick executes all the six behaviors explained in Sect. 4.

During the data acquisition process, for each network configuration (explained below), the following types of behaviors were collected in the sensor:

- *Normal*: Sensor running a by-default spectrum monitoring campaign.
- *NormalMode1*: Sensor running the FM radio decoder service.
- *NormalMode2*: Sensor running the AM radio decoder service.
- *NormalMode3*: Sensor running the ADS-B service.
- *NormalMode4*: Sensor running the AIS service.
- *NormalMode5*: Sensor running the ACARS service.
- *NormalMode6*: Sensor running the LTE service.
- *Mirai*: Sensor running Mirai in a passive way.
- *Bashlite*: Sensor running Bashlite in a passive way.

Table 3 ML algorithms and hyperparameters tested

Algor.	Hyperparameters tested
GNB	No hyperparameter tunning required
KNN	$k \in [3, 20]$
SVM	$C \in [0.01, 100]$, $gamma \in [0.001, 10]$ $kernel \in \{'rbf','linear','sigmoid','poly'\}$
SGD	$loss \in [log, modified\_huber, squared\_hinge, perceptron]$, $penalty \in [l2, l1]$, $alpha \in [0.0001, 0.1]$, $learning\_rate = optimal$
XGB	$lr \in [0.01, 0.30]$, $max\_depth \in [3, 15]$ $colsample\_bytree \in [0.3, 0.7]$, $gamma \in [0, 0.5]$, $min\_child\_weight \in [1, 7]$
DT	$max\_depth \in [None, 5, 10, 15, 20]$, $min\_samples\_split \in [2, 3, 4, 5]$
RF	$number\_of\_trees \in [50, 1000]$ $max\_depth \in [None, 5, 10, 15, 20]$ $min\_samples\_split \in [2, 3, 4, 5]$
LOF	$n\_neighbors \in [3, 25]$
OCSVM	$gamma \in [0.001, 100]$, $kernel \in \{'rbf','linear','sigmoid','poly'\}$, $degree \in [2, 5] (only\ poly\ kernel)$
IF	$Number\_of\_trees \in [50, 1000]$

- *Beurk*: Sensor running Beurk in a passive way.
- *Bdvl*: Sensor running Bdvl in a passive way.
- *Diamorphine*: Sensor running Diamorphine in a passive way.
- *DiamorphineSSH5s*: Sensor running Diamorphine and receiving an ssh connections from the laptop every 5 s.
- *Execution*: Backdoor executing basic commands on the sensor.
- *Download*: Backdoor creating a new directory, and cloning a repository on the sensor.
- *Removal*: Backdoor deleting files of a selected directory of the sensor.
- *Data leak*: Backdoor pulling a file from the sensor to the server.
- *DNS*: Backdoor deploying a Domain Name Server (DNS) on the sensor.
- *Privilege*: Backdoor changing files privileges on the sensor.
- *Ransomware_PoC*: Sensor running Ransomware_PoC while encrypting files on the sensor.
- *Cryptojaker_Linux.MulDrop.14*: Sensor running the Linux.MulDrop.14 cryptojaker to mine Monero.

After monitoring each behavior type, a dataset was created and the sensor image was restored to a clean one to avoid permanent changes that would affect the sensor behavior. The total time monitored for each dataset had a minimum duration of 7 h. Datasets were normalized and balanced during pre-processing, allowing for a maximum of 10% extra length between the shortest and the longest ones. In each experiment, 75% of data is divided for training and cross-validation, and the remaining 25% for testing. The metric used to measure the detection performance is $F1$-score, which is based on Precision and Recall.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{1}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2}$$

$$F1\text{-}score = \frac{2 * precision * recall}{precision + recall} \tag{3}$$

5.2.1 Fiber to the home (FTTH) experiment

In this experiment, a monitoring campaign for all types of behaviors was performed. Its main particularity is that the network connection for both the sensor and the auxiliary laptop is based in Fiber to the Home (FTTH). It provides a robust bandwidth of up to 300 Mbps with a latency of around 30ms and no major instabilities. It would act as the ideal setup for the sensor. It is essential to mention that DiamorphineSSH5s was monitored in such a scenario because it is not detected when working passively, as shown in the subsequent results.

Table 4 Anomaly detection results for FTTH experiment across all algorithms tested

Class	Metric	OC-SVM	IF	LOF
Normal	TNR	0.96	0.28	0.89
Abnormal	TPR	0.84	0.70	0.84
Average		**0.90**	0.56	0.86

The bold text highlights the model providing the best results

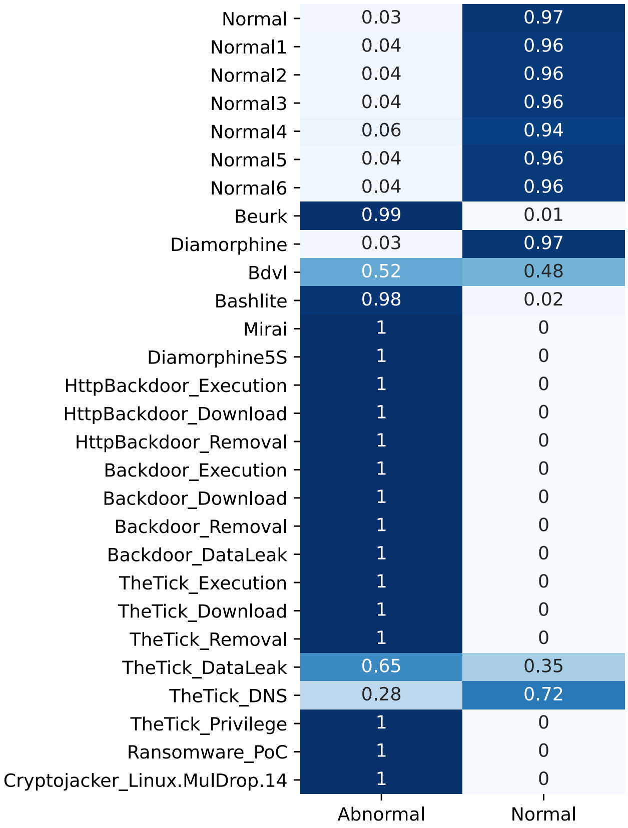


Fig. 5 OC-SVM confusion matrix for anomaly detection in FTTH experiment

This is due to the negligible activity of this rootkit, which is not harmful on its own; its passive behavior is reduced to getting loaded as a kernel module and hiding itself. If this rootkit were employed to control the sensor secretively, a simple approach would be to launch an SSH and hide the implied processes.

From the anomaly detection approach, and given its semi-supervised nature, the algorithms are only trained using 75% of the normal behavior datasets. The testing is performed against their remaining 25% and all the attack datasets. The results for all three algorithms tested can be seen in Table 4, with OC-SVM obtaining the best average TNR and TPR: 0.90.

Figure 5 shows the confusion matrix for the best performing algorithm (OC-SVM), which also had the highest true negative rate, with a noticeable difference. The hyperparameters explicitly set are *{'kernel': rbf, 'gamma': 0.02, 'nu': 0.01}*. The results show that more than 95% of samples belonging to the different normal behaviors are correctly detected as "normal." Looking at the rootkits, the passive and innocuous behavior of Diamorphine is not detected, but when it establishes an SSH connection every five seconds, it is identified as malicious. In the case of passive behavior of Bdvl, it is detected only half of the time. In terms of Backdoors, 34% of the samples belonging to Data Leak behavior executed by TheTick are not detected correctly. Furthermore, the DNS behavior only is detected 27% of the time. Regarding Ransomware and Cryptojacker, their vector samples are perfectly detected. At this point, it is important to mention that those two behaviors have a minimum impact on the sensor integrity and data confidentiality, as the leaked data is only a few Kb of sensitive data. To conclude, it is important to highlight that the rest of the malicious behaviors are detected in an almost perfect fashion.

From the malware classification perspective, after preprocessing the 25 datasets and using 75% for training and 25% for testing, the obtained results are shown in Table 5. The best performing algorithms are RF and XGB, obtaining an $F1$-score of 0.96. The resulting confusion matrix, focused on RF, can be seen in Fig. 6. It shows that the results are almost perfect except for (i) Diamorphine, which is sometimes confused as Normal, (ii) TheTick_DataLeak, where 25% of samples are confused with Normal behavior, and (iii) TheTick_DNS, with the 12% of samples wrongly classified as Normal. The hyperparameters used for RF are *{'bootstrap': False, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 10}*.

5.2.2 4G mobile network experiment

In this experiment, the same methodology as the previous one is followed. The unique difference is that a new set of 25 datasets was created (one per behavior) in a new monitoring campaign. This second campaign is performed in the same conditions as the first one except for the router, which provides the internet connection via 4G. The usual bandwidth of this connection was about 12–18 Mbps, and the latency would fluctuate between 70 and 130 ms, with peaks of more than 200 ms. This experiment is performed to test if the behavior of the device remains stable when the internet connection conditions are not so good.

The algorithms for anomaly detection are trained and evaluated as in the previous experiment (trained using 75% of the normal behavior datasets and tested against their remaining 25% plus all malware datasets). The results obtained by OC-SVM and LOF are very similar in terms of TNR and TPR, as shown in Table 6, with IF performing worse than the oth-

Table 5 Classification results ($F1$-score) for FTTH experiment across all classifiers tested

Class	RF	SVM	KNN	GNB	XGB	DT
Normal	0.93	0.88	0.87	0.02	0.93	0.89
Normal1	1.00	1.00	1.00	1.00	1.00	1.00
Normal2	1.00	1.00	1.00	1.00	1.00	1.00
Normal3	1.00	1.00	1.00	0.68	1.00	1.00
Normal4	1.00	0.98	0.97	0.95	1.00	0.98
Normal5	0.96	0.86	0.68	0.49	0.98	0.87
Normal6	0.98	0.91	0.78	0.01	0.99	0.91
Beurk	1.00	1.00	0.97	0.63	1.00	1.00
Diamorphine	0.68	0.00	0.34	0.40	0.74	0.60
Bdvl	1.00	1.00	1.00	0.87	1.00	1.00
Bashlite	1.00	1.00	0.96	0.75	0.99	0.99
Mirai	1.00	1.00	1.00	1.00	1.00	1.00
Diamorphine SSH5S	1.00	1.00	1.00	1.00	1.00	1.00
HttpBackdoor Execution	0.98	1.00	0.88	0.58	1.00	0.97
HttpBackdoor Download	1.00	0.98	0.89	0.61	1.00	1.00
HttpBackdoor Removal	1.00	1.00	0.90	0.70	1.00	0.98
Backdoor Execution	0.93	0.81	0.57	0.57	0.94	0.84
Backdoor Download	1.00	1.00	0.67	0.81	1.00	0.98
Backdoor Removal	1.00	1.00	0.94	0.46	1.00	1.00
Backdoor DataLeak	0.90	0.72	0.74	0.44	0.90	0.79
TheTick Execution	1.00	0.92	0.81	0.00	0.98	0.89
TheTick Download	1.00	0.99	0.88	0.99	1.00	0.99
TheTick Removal	1.00	1.00	0.94	1.00	1.00	1.00
TheTick DataLeak	0.68	0.38	0.37	0.28	0.70	0.66
TheTick DNS	0.87	0.00	0.47	0.13	0.92	0.72
TheTick Privilege	1.00	1.00	1.00	0.19	1.00	1.00
Ransomware PoC	1.00	1.00	1.00	1.00	1.00	1.00
Cryptojacker Linux.MulDrop.14	1.00	1.00	1.00	1.00	1.00	1.00
Average	**0.96**	0.89	0.88	0.57	**0.96**	0.93

The bold text highlights the model providing the best results

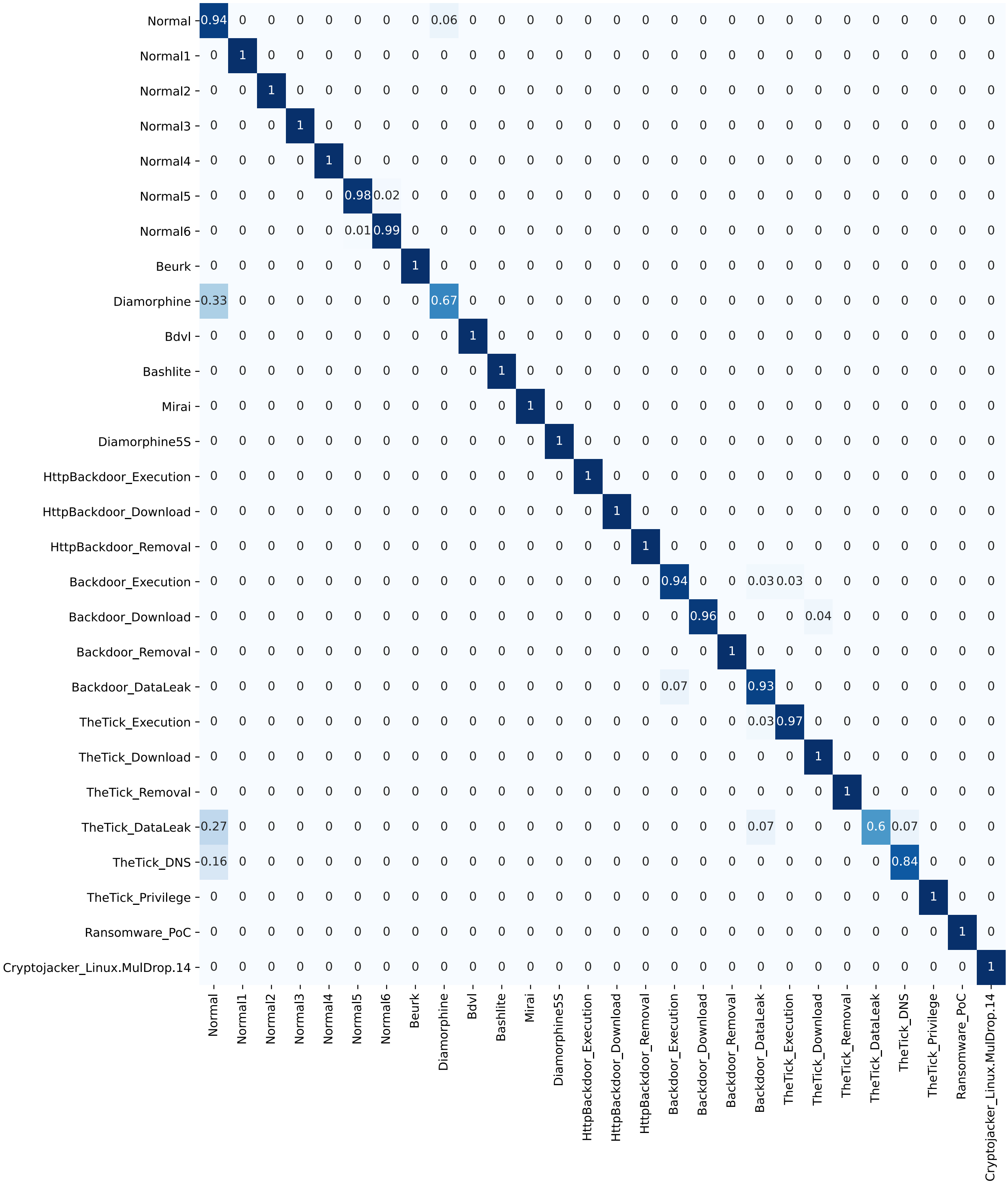
ers. Figure 7 shows the confusion matrix for LOF, which results are aligned with those obtained in the previous experiment with optical fiber. More in detail, all normal behaviors are still detected correctly, but with a lower TNR (2% less than with fiber). In terms of attacks, the opposite happens because the averaged TPR increases 2%. In particular, LOF detects Bdvl almost perfectly (in contrast to the 52% of the previous experiment). It can be due to the effect of the network in the normal behavior (more unstable), and how it affects the threshold of the Anomaly detector (the same % is increased and decreased for normal and under-attack behaviors). Furthermore, TheTick_DataLeak and TheTick_DNS are detected with a higher TPR. Finally, LOF provides similar results as OC-SVM in the previous experiment for the rest of the attacks. The hyperparameters selected to obtain the previous results are: *{'n_neighbors': 5, 'contamination': 0.002}*

The classifiers are also considered as in the previous experiment (trained with 75% of all datasets and tested with the remaining 25%). The obtained results are also very similar to the ones of the previous experiment, as shown in Table 7. The best performing algorithm is XGB, which performs well but presents some problems (TPR and TNR lower than 80%) to classify Normal1, Normal2, and TheTick_DataLeak. The details can be appreciated in Fig. 8, which shows the results for the best hyperparameters configuration found for XGB: *{'learning_rate': 0.01, 'gamma': 0.5, 'max_depth': 10, 'subsample': 0.9, 'colsample_bytree': 0.75}*.

6 Summary and discussion

This work proposes an ML-based and behavioral fingerprinting framework to detect anomalies and to classify different types of botnets, rootkits, backdoors, ransomware and cryptojackers affecting IoT spectrum sensors. The framework design is based on a distributed approach, where behavioral data is collected on the fingerprinted sensor and analyzed, using several supervised and semi-supervised ML-based algorithms, on a central server. Different kernel events belonging to the CPU, virtual memory, network interface, file system, scheduler, generation of random numbers, and device drivers data sources have been analyzed, selected, and monitored to create fingerprints. The usefulness of the framework has been validated as a PoC in a Raspberry Pi 4 (acting as a sensor) that was infected with two botnets (Mirai and Bashlite), three rootkits (Diamorphine, Beurk, and Bdvl), three backdoors (HttpBackdoor, Backdoor, TheTick), one Ransomware (Ransomware_PoC), and one Cryptojacker (Linux.MulDrop.14). A set of experiments using two different network configurations (optical fiber with low network traffic and 4G network with high congestion) and different supervised and semi-supervised ML algorithms have been performed.

Dealing with the advantages of the proposed solution, to the best of our knowledge, this is the first work combining intelligent behavioral fingerprinting into a real IoT spectrum sensor to detect and classify malicious behaviors of recent and real heterogeneous malware. The IoT Spectrum sensing scenario is one of the most important parts of this work. Therefore, a physical IoT spectrum sensor (Raspberry Pi equipped with an SDR kit) belonging to the real-world crowdsensing platform, called ElectroSense, has



	Normal	Normal1	Normal2	Normal3	Normal4	Normal5	Normal6	Beurk	Diamorphine	Bdvl	Bashlite	Mirai	Diamorphine5S	HttpBackdoor_Execution	HttpBackdoor_Download	HttpBackdoor_Removal	Backdoor_Execution	Backdoor_Download	Backdoor_Removal	Backdoor_DataLeak	TheTick_Execution	TheTick_Download	TheTick_Removal	TheTick_DataLeak	TheTick_DNS	TheTick_Privilege	Ransomware_PoC	Cryptojacker_Linux.MulDrop.14
Normal	0.94	0	0	0	0	0	0	0	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal5	0	0	0	0	0	0.98	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal6	0	0	0	0	0	0.01	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Beurk	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diamorphine	0.33	0	0	0	0	0	0	0	0.67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bdvl	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bashlite	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mirai	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diamorphine5S	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Execution	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Backdoor_Execution	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.94	0	0	0.03	0.03	0	0	0	0	0	0	0
Backdoor_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.96	0	0	0	0.04	0	0	0	0	0	0
Backdoor_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Backdoor_DataLeak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0	0	0.93	0	0	0	0	0	0	0	0
TheTick_Execution	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.97	0	0	0	0	0	0	0
TheTick_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
TheTick_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
TheTick_DataLeak	0.27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0	0	0	0.6	0.07	0	0	0
TheTick_DNS	0.16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.84	0	0	0
TheTick_Privilege	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Ransomware_PoC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Cryptojacker_Linux.MulDrop.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 6 RF confusion matrix for classification in FTTH experiment

Table 6 Anomaly detection results for 4G experiment across all algorithms tested

Class	Metric	OC-SVM	IF	LOF
Normal	TNR	0.94	0.14	0.91
Abnormal	TPR	0.82	0.38	0.86
Average		**0.88**	0.26	**0.88**

The bold text highlights the model providing the best results

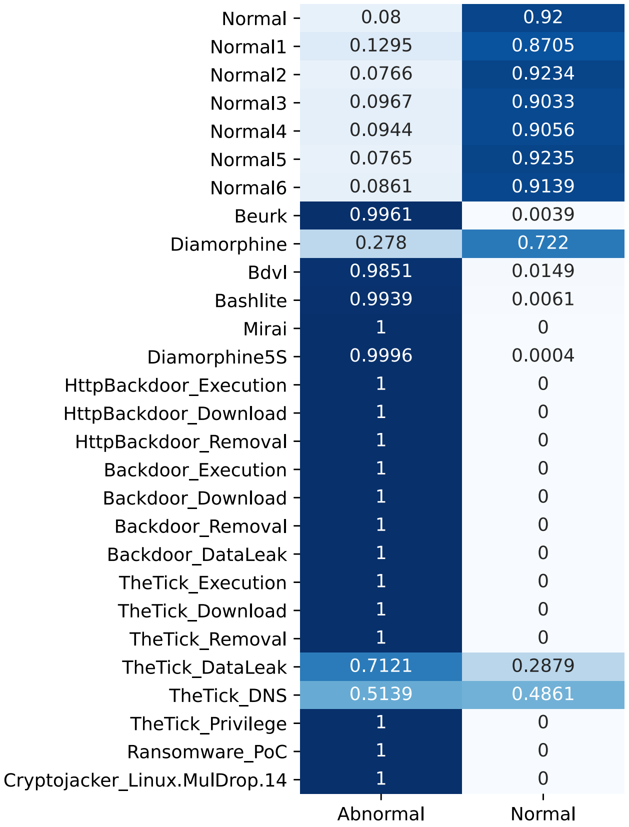


Fig. 7 LOF confusion matrix for anomaly detection in 4G experiment

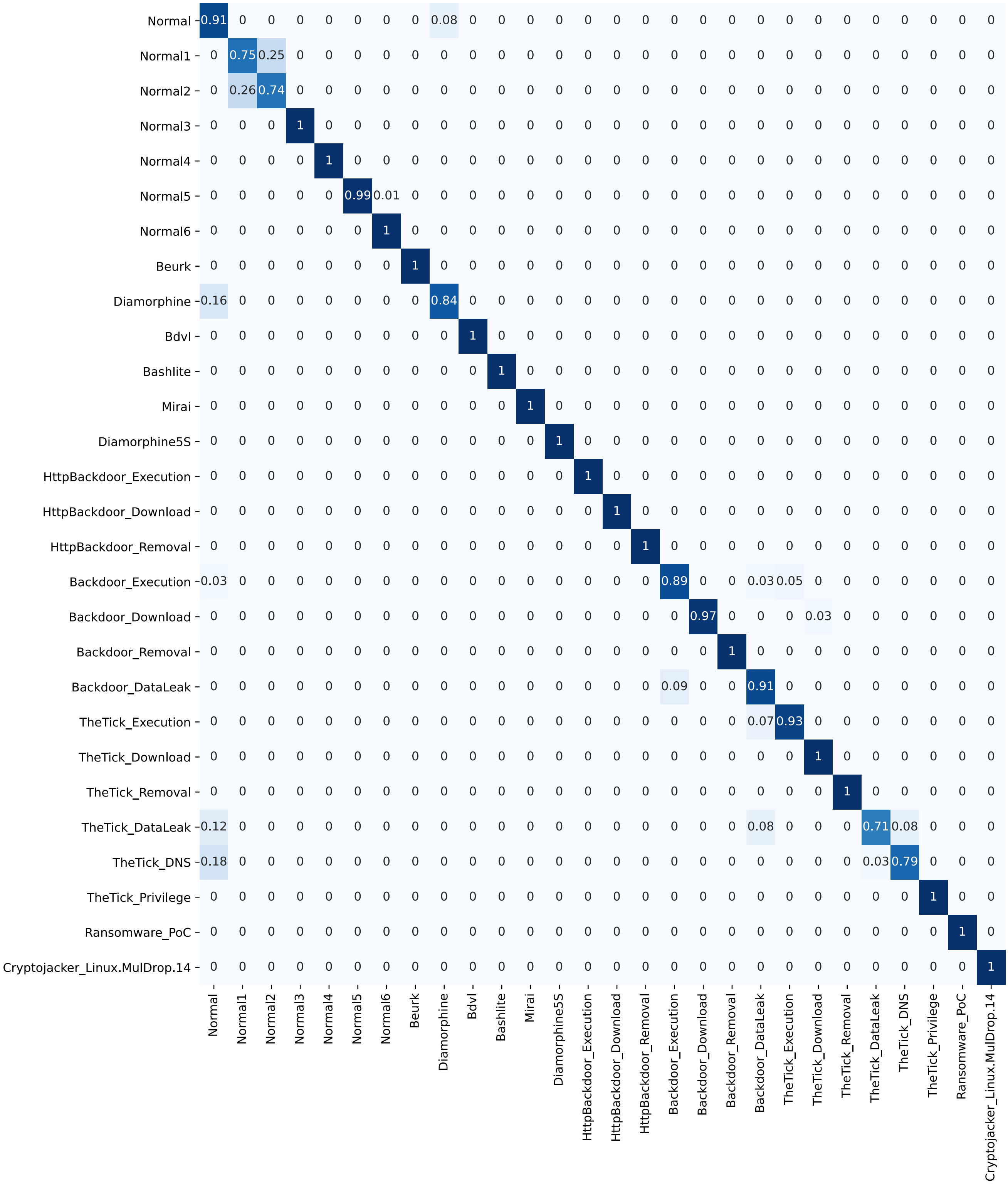
been selected. ElectroSense sensors can be deployed on different networks with heterogeneous bandwidth, congestion, and delay capabilities. These aspects could influence the normal behavior of the sensor, its stability across time, and the detection performance of the proposed system. Thus, this work has evaluated the framework detection performance in two completely different local area networks with different capabilities, one with optical fiber and another with 4G.

After a pool of experiments in both network configurations, it can be concluded that Mirai, Bashlite, Beurk, the three behaviors of HttpBackdoor, the four behaviors of Backdoor, four behaviors of TheTick, Bdvl, Ransomware_PoC, and Linux.MulDrop.14 are well detected from the anomaly detection (80–100% $F1$-score) and classification (almost 100% $F1$-score) perspectives in both network scenarios (stable optical fiber and unstable 4G network). Although Diamorphine is classified with acceptable performance (more than 70% $F1$-score) in both network scenarios by Random Forest, it remains mainly undetected (between 3% and 22% $F1$-score) across the anomaly detection experiments of both network scenarios. However, it is essential to mention that Diamorphine does not have a passive harmful behavior. Finally, Bdvl, TheTick_DataLeak, and TheTick_DNS are detected in a better way when using the 4G router. These differences could be influenced by the threshold selected by the algorithm (the average TNR rate for normal behavior decreases the same % as the TPR increases for malicious behavior), or due to unexpected incidents during the monitoring campaign, such as an update from or to the ElectroSense back-end.

When comparing the performance of the proposed solution to related work, the following aspects are critical to performing a fair comparison: (i) device hardware and software, (ii) malware family and sample, and (iii) detection technique.

Dealing with the first point, the hardware configuration and the functionality of the IoT spectrum sensor strongly affect the normal behavior fingerprints. Therefore, these aspects should be similar in different works to compare the performance of the proposed detection approach. As demonstrated in Sect. 2, there is no existing solution that detects malware attacks affecting IoT Spectrum sensors and, more in particular, using the ElectroSense platform. Indeed, this is one of the main novelties of this work. The malware type, sample, and functionality are also critical aspects for a fair comparison. In this context, as can be seen in Table 1, there are many works detecting different malware types and families, some of them with excellent detection performance. However, most of them are deployed and validated on general computers and servers. It means these malware samples differ in terms of implementations details and functionality from those used for Raspberry Pis. In particular, Raspberry Pis have ARM processor architectures, while computers and servers consider x86 architectures. Finally, the detection approach is also relevant when comparing the performance and suitability of different detection solutions. In this sense, Sect. 2 compares a relevant number of related work using ML-based approaches like the ones considered in this work.

Despite the contributions of this work, it is also important to mention its most relevant limitations. In this context, a greater variety of IoT spectrum sensors should be considered to generalize the obtained results to other crowdsensing platforms. In this sense, one of the main objectives of this work was to consider real-world IoT spectrum sensors and platforms, and ElectroSense is one of the most suitable alternatives due to its public availability, open-source, and crowd-sourcing nature. However, this platform currently only supports Raspberry Pis as spectrum sensors. Another limitation is the lack of networks with different conditions



	Normal	Normal1	Normal2	Normal3	Normal4	Normal5	Normal6	Beurk	Diamorphine	Bdvl	Bashlite	Mirai	Diamorphine5S	HttpBackdoor_Execution	HttpBackdoor_Download	HttpBackdoor_Removal	Backdoor_Execution	Backdoor_Download	Backdoor_Removal	Backdoor_DataLeak	TheTick_Execution	TheTick_Download	TheTick_Removal	TheTick_DataLeak	TheTick_DNS	TheTick_Privilege	Ransomware_PoC	Cryptojacker_Linux.MulDrop.14
Normal	0.91	0	0	0	0	0	0	0	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal1	0	0.75	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal2	0	0.26	0.74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal5	0	0	0	0	0	0.99	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Beurk	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diamorphine	0.16	0	0	0	0	0	0	0	0.84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bdvl	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bashlite	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mirai	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diamorphine5S	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Execution	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
HttpBackdoor_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Backdoor_Execution	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.89	0	0	0.03	0.05	0	0	0	0	0	0	0
Backdoor_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.97	0	0	0	0.03	0	0	0	0	0	0
Backdoor_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Backdoor_DataLeak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0	0.91	0	0	0	0	0	0	0	0
TheTick_Execution	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0.93	0	0	0	0	0	0	0
TheTick_Download	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
TheTick_Removal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
TheTick_DataLeak	0.12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08	0	0	0	0.71	0.08	0	0	0
TheTick_DNS	0.18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.79	0	0	0
TheTick_Privilege	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Ransomware_PoC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Cryptojacker_Linux.MulDrop.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 8 XGB confusion matrix for classification in the 4G experiment

Table 7 Classification results ($F1$-score) for the 4G experiment across all classifiers tested

Class	RF	SVM	KNN	GNB	XGB	DT
Normal	0.90	0.75	0.79	0.04	0.92	0.87
Normal1	0.73	0.69	0.61	0.67	0.76	0.62
Normal2	0.70	0.47	0.58	0.2	0.72	0.62
Normal3	1.00	1.00	0.99	0.99	1.00	1.00
Normal4	1.00	1.00	0.98	0.60	1.00	1.00
Normal5	0.99	0.99	0.89	0.55	1.00	0.99
Normal6	1.00	0.99	0.91	0.00	1.00	0.99
Beurk	1.00	1.00	0.96	0.0.11	1.00	1.00
Diamorphine	0.80	0.58	0.34	0.50	0.82	0.71
Bdvl	1.00	1.00	0.99	0.94	1.00	1.00
Bashlite	1.00	1.00	0.98	0.99	1.00	1.00
Mirai	1.00	1.00	1.00	1.00	1.00	1.00
Diamorphine SSH5S	1.00	1.00	1.00	1.00	1.00	1.00
HttpBackdoor Execution	1.00	1.00	0.85	0.54	1.00	0.98
HttpBackdoor Download	1.00	1.00	0.95	0.58	1.00	1.00
HttpBackdoor Removal	1.00	1.00	0.86	0.68	1.00	1.00
Backdoor Execution	0.91	0.82	0.56	0.65	0.90	0.87
Backdoor Download	1.00	1.00	0.81	0.81	1.00	1.00
Backdoor Removal	1.00	1.00	0.95	0.42	1.00	1.00
Backdoor DataLeak	0.84	0.71	0.73	0.41	0.85	0.80
TheTick Execution	0.98	0.88	0.81	0.06	0.95	0.90
TheTick Download	1.00	1.00	0.93	0.97	1.00	1.00
TheTick Removal	1.00	1.00	0.90	1.00	0.98	1.00
TheTick DataLeak	0.65	0.39	0.37	0.27	0.78	0.67
TheTick DNS	0.75	0.00	0.47	0.13	0.80	0.74
TheTick Privilege	1.00	1.00	0.98	0.21	1.00	1.00
Ransomware PoC	1.00	1.00	1.00	1.00	1.00	1.00
Cryptojacker Linux.MulDrop.14	1.00	1.00	1.00	1.00	1.00	1.00
Average	**0.94**	0.88	0.87	0.64	**0.95**	0.91

The bold text highlights the model providing the best results

to see how the number of devices connected to the same network, the management tasks of the network, and the bandwidth and latency, to mention a few aspects, affect the fingerprints of different sensors.

7 Conclusions and future work

The conclusions of this work can be synthesized in the following four statements: (i) the detection capabilities of classifiers (supervised approach) are almost perfect across all experiments (stable optical fiber and unstable 4G networks), showing an average $F1$-score of 0.96, where most missed predictions are between Diamorphine (acting passively and harmlessly), TheTick with two low impact attacks, and the device normal behavior; (ii) the anomaly detection approach shows very promising results, when detecting anomalies produced by most of the attacks Beurk, Bashlite, Mirai, HttpBackdoor, Backdoor, Ransomware_PoC, and Linux.MulDrop.14, and four of the six behaviors of TheTick), except for the passive behavior of Diamorphine and Bdvil (both of them harmless), TheTick_DataLeak and TheTick_DNS (with a low impact on the sensor integrity and data confidentiality; (iii) in general, the experiments show few rates of false positives (normal samples evaluated as attack), which is particularly important in this type of solution; and (iv) these results obtained are consistent, robust, and reliable, since they have been analyzed with data obtained under different network conditions.

Future work will analyze the performance of the framework upon monitoring other types of devices apart from Raspberry Pis is foreseen. Finally, further reviews of the monitored event list are expected, along with further experimentation with the monitoring window size.

Acknowledgements This work has been partially supported by (a) the Swiss Federal Office for Defense Procurement (armasuisse) with the RESERVE and CyberTracer (CYD-C-2020003 and R-3210/047-31) projects and (b) the University of Zürich UZH. Finally, we would like to thank Jan von der Assen, Fabio Sisi, Konstantin Moser, and Timucin Besken for their work on the malware samples.

Funding Open access funding provided by University of Zurich

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals.

References

1. Lueth, K.L.: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections/-surpassing-non-iot-for-the-first-time/ (2020). Accessed 29 Sept 2021

2. Boulogeorgos, A., Karagiannidis, G.: Low-cost cognitive radios against spectrum scarcity. IEEE Techn. Comm. Cognit. Netw. Newslett. **3**, 30–34 (2017)
3. Wang, X., Wang, J., Xu, Y., Chen, J., Jia, L., Liu, X., Yang, Y.: Dynamic spectrum anti-jamming communications: challenges and opportunities. IEEE Commun. Mag. **58**(2), 79–85 (2020)
4. Rajendran, S., Calvo-Palomino, R., Fuchs, M., Van den Bergh, B., Cordobés, H., Giustiniano, D., Pollin, S., Lenders, V.: Electrosense: open and big spectrum data. IEEE Commun. Mag. **56**(1), 210–217 (2018)
5. Electrosense. Collaborative spectrum monitoring. https://electrosense.org/
6. Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., HPC Chaves, M., Cunha, Í., Guedes, D., Meira W.: The evolution of bashlite and mirai iot botnets. In: 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00813–00818. IEEE (2018)
7. Tian, D., Ma, R., Jia, X., Hu, C.: A kernel rootkit detection approach based on virtualization and machine learning. IEEE Access **7**, 91657–91666 (2019)
8. Li, C., Chen, X., Wang, D., Wen, S., Ahmed, M.E., Camtepe, S., Xiang, Y.: Backdoor attack on machine learning based android malware detectors. IEEE Trans. Dependable Secure Comput. (2021). https://doi.org/10.1109/TDSC.2021.3094824
9. Adamov, A., Carlsson, A.: The state of ransomware. Trends and mitigation techniques. In: EWDTS, pp. 1–8 (2017)
10. Calvão, F.: Crypto-miners: digital labor and the power of blockchain technology. Econ. Anthropol. **6**(11), 123–134 (2019)
11. Rey, V., Pedro Miguel Sánchez, S., Alberto Huertas, C., Gérôme, B.: Federated learning for malware detection in IoT devices. Comput. Netw. **204**, 108693 (2022)
12. Sánchez, P.M.S., Valero, J.M.J., Celdrán, A.H., Bovet, G., Gil Pérez, M., Pérez, G.M.: A survey on device behavior fingerprinting: data sources, techniques, application scenarios, and datasets. IEEE Commun. Surv. Tutor. **23**(2), 1048–1077 (2021)
13. hammerzeit. An archive of bashlite source code. https://github.com/hammerzeit/BASHLITE (2016). Accessed 29 Sept 2021
14. jgamblin. Leaked mirai source code for research/ioc development purposes. https://github.com/jgamblin/Mirai-Source-Code (2016). Accessed 29 Sept 2021
15. m0nad. Lkm rootkit for linux kernels 2.6.x/3.x/4.x/5.x (x86/x86_64 and arm64). https://github.com/m0nad/Diamorphine (2013). Accessed 29 Sept 2021
16. unix thrust. Beurk experimental unix rootkit. https://github.com/unix-thrust/beurk (2015). Accessed 29 Sept 2021
17. Error996. bdvl. https://github.com/Error996/bdvl (2020). Accessed 29 Sept 2021
18. SkryptKiddie. httpbackdoor. https://github.com/SkryptKiddie/httpBackdoor (2020). Accessed 6 Aug 2021
19. Jo ao Koritar. backdoor. https://github.com/jakoritarleite/backdoor (2020). Accessed 6 Aug 2021
20. Nccgroup. The tick: a simple embedded Linux backdoor. https://github.com/nccgroup/thetick/ (2021). Accessed 6 Aug 2021
21. Jimmyly00. Ransomware PoC GitHub repository. https://github.com/jimmy-ly00/Ransomware-PoC (2020). Accessed 15 Apr 2022
22. Cimpanu, C.: Linux.multdrop.1y4. https://www.bleepingcomputer.com/news/security/linux-malware-mines-for-cryptocurrency-using-/raspberry-pi-devices/ (2017). Accessed 30 May 2022
23. Sayadi, H., Makrani, H.M., Randive, O., Sai Manoj, P.D., Rafatirad, S., Homayoun, H.: Customized machine learning-based hardware-assisted malware detection in embedded devices. In: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1685–1688. IEEE (2018)
24. Sayadi, H., Patel, N., Manoj, S., Sasan, A., Rafatirad, S., Homayoun, H.: Ensemble learning for effective run-time hardware-based malware detection: a comprehensive analysis and classification. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (2018)
25. Ott, K., Mahapatra R.: Hardware performance counters for embedded software anomaly detection. In: 2018 IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 528–535. IEEE (2018)
26. Cronin, P., Yang C.: Lowering the barrier to online malware detection through low frequency sampling of hpcs. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 177–180. IEEE (2018)
27. Pudukotai Dinakarrao, S. M., Sayadi, H., Makrani, H. M., Nowzari, C., Rafatirad, S., Homayoun, H.: Lightweight node-level malware detection and network-level malware confinement in IoT networks. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 776–781. IEEE (2019)
28. Dai, Y., Li, H., Qian, Y., Yang, R., Zheng, M.: Smash: a malware detection method based on multi-feature ensemble learning. IEEE Access **7**, 112588–112597 (2019)
29. Das, S., Werner, J., Antonakakis, M., Polychronakis, M., Monrose, F.: Sok: the challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 20–38. IEEE (2019)
30. Kadiyala, S.P., Jadhav, P., Lam, S., Srikanthan, T.: Hardware performance counter-based fine-grained malware detection. ACM Trans. Embed. Comput. Syst. **19**(5), 1–17 (2020)
31. Zhou, L., Makris Y.: Hardware-based on-line intrusion detection via system call routine fingerprinting. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 1546–1551. IEEE (2017)
32. De Lorenzo, A., Martinelli, F., Medvet, E., Mercaldo, F., Santone, A.: Visualizing the outcome of dynamic analysis of android malware with VizMal. J. Inf. Secur. Appl. **50**, 102423 (2020)
33. Mishra, P., Varadharajan, V., Pilli, E.S., Tupakula, U.: VMGuard: a VMI-based security architecture for intrusion detection in cloud environment. IEEE Trans. Cloud Comput. **8**(3), 957–971 (2020)
34. Ravichandiran, R., Bannazadeh, H., Leon-Garcia, A.: Anomaly detection using resource behaviour analysis for autoscaling systems. In: 2018 4th IEEE Conference on Network Softwarization, pp. 192–196 (2018)
35. Barbhuiya, S., Papazachos,, Z., Kilpatrick, P., Nikolopoulos D.S.: RADS: Real-Time Anomaly Detection System for Cloud Data Centres (2018)
36. You, I., Yim, K., Sharma, V., Choudhary, G., Chen, I.-R., Cho, J.-H.: Misbehavior detection of embedded IoT devices in medical cyber physical systems. In: 2018 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), pp. 88–93 (2018)
37. Sharma, V., You, I., Yim, K., Chen, R., Cho, J.-H.: Briot: behavior rule specification-based misbehavior detection for IoT-embedded cyber-physical systems. IEEE Access **7**, 118556–118580 (2019)
38. Jeon, J., Park, J.H., Jeong, Y.S.: Dynamic analysis for IoT malware detection with convolution neural network model. IEEE Access **8**, 96899–96911 (2020)
39. Ahmed, M.E., Kim, H., Camtepe, S., Nepal, S.: Peeler: profiling kernel-level events to detect ransomware. *CoRR*. arXiv:2101.12434 (2021)
40. Zhou, L., Makris Y.: Hardware-assisted rootkit detection via online statistical fingerprinting of process execution. In: 2018 Design,

Automation & Test in Europe Conference & Exhibition (DATE), pp. 1580–1585. IEEE (2018)

41. Luh, R., Janicke, H., Schrittwieser, S.: Aidis: detecting and classifying anomalous behavior in ubiquitous kernel processes. Comput. Secur. **84**, 120–147 (2019)
42. Celdrán, A.H., Sánchez, P.M.S., Bovet, G., Pérez, G.M., Stiller, B.: Cyberspec: intelligent behavioral fingerprinting to detect attacks on crowdsensing spectrum sensors. arXiv e-prints, pages arXiv-2201 (2022)
43. Sánchez, P.M.S., Celdrán, A.H., Schenk, T., Iten, A.L.B. , Bovet, G., Pérez, G.M., Stiller, B.: Studying the robustness of anti-adversarial federated learning models detecting cyberattacks in iot spectrum sensors. arXiv preprint arXiv:2202.00137 (2022)
44. mempodippy. Linux ld_preload rootkit (x86 and x86_64 architectures). https://github.com/mempodippy/vlany (2016). Accessed 29 Sept 2021
45. Kovacs, E.: https://www.securityweek.com/bashlite-botnets-ensnare-1-million-iot-devices (2016). Accessed 29 Sept 2021
46. Krebs, B.: Krebsonsecurity hit with record ddos. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/ (2016). Accessed 29 Sept 2021
47. Cisco ransomware defense validated design guide. https://www.eschoolnews.com/files/2017/02/494454_Ransomware-Defense-Validated-Design-Guide.pdf (2016). Accessed 30 May 2022
48. jimmy ly00. Ransomware poc github repository. https://github.com/jimmy-ly00/Ransomware-PoC (2020). Accessed 29 Sept 2021
49. Kons, A.: Monero mining: So klappt das xmr mining. https://de.beincrypto.com/lernen/monero-mining-so-klappt-das-xmr-mining/ (2021). Accessed 30 May 2022
50. Kernel Wiki. perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page. Accessed 26 Feb 2022
51. Treibig, J., Hager, G., Wellein, G.: Likwid: a lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th International Conference on Parallel Processing Workshops, pp. 207–216. IEEE (2010)
52. Dongarra, J., London, K., Moore, S., Mucci, P., Terpstra, D.: Using papi for hardware performance monitoring on Linux systems. In: Conference on Linux Clusters: The HPC Revolution, vol. 5. Citeseer (2001)
53. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)