Computers & Security 135 (2023) 103510

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Behavioral fingerprinting to detect ransomware in resource-constrained devices

Alberto Huertas Celdrán [a,*], Pedro Miguel Sánchez Sánchez [b], Jan von der Assen [a], Dennis Shushack [a], Ángel Luis Perales Gómez [c], Gérôme Bovet [d], Gregorio Martínez Pérez [b], Burkhard Stiller [a]

[a] Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, CH—8050 Zürich, Switzerland
[b] Department of Information and Communications Engineering, University of Murcia, Murcia 30100, Spain
[c] Departamento de Ingeniería y Tecnología de Computadores, University of Murcia, Murcia 30100, Spain
[d] Cyber-Defence Campus within armasuisse Science & Technology, CH—3602 Thun, Switzerland

ARTICLE INFO

Keywords:
Cybersecurity
Ransomware
Fingerprinting
Machine learning

ABSTRACT

The Internet of Things (IoT), a network of interconnected devices, has grown and gained traction over the last few years. This paradigm can impact our lives while also providing significant economic benefits. However, although resource-constrained IoT devices offer numerous advantages, they are also vulnerable to cyberattacks. As a result, ransomware severely threatens IoT devices managing sensitive and relevant information. Solutions based on Machine and Deep Learning (ML/DL) that consider behavioral data have been identified as promising. However, most detection solutions have been developed for Windows-based systems, which generally have more resources than IoT devices. As a result, these solutions are not suitable for resource-constrained components. In addition, no solution compares the pros and cons of different behavioral dimensions of resource-constrained devices. Thus, this work presents a framework that combines three different behavioral sources with supervised and unsupervised ML/DL algorithms to detect and classify heterogeneous ransomware impacting resource-constrained spectrum sensors. A pool of experiments has demonstrated the suitability of the proposed solution and compared its performance with a rule-based system. In conclusion, the usage of resources combined with local outlier factor and decision tree are the most promising combinations to detect anomalies and classify ransomware while consuming CPU, RAM, and time of devices in a reduced manner.

1. Introduction

The Internet of Things (IoT) paradigm has received great attention in recent years since resource-constrained devices significantly affect our daily lives and positively impact the economy and society. Nowadays, over 18 billion IoT devices are in use, bringing benefits to transportation, manufacturing, health care, or retail services, among other industries (Thierer and Castillo, 2015). However, using resource-constrained devices is not exempt from concerns and deficiencies. Poor security, lax device management procedures, software vulnerabilities, and low maintenance are common and can be exploited by cybercriminals.

After botnets, ransomware poses one of the most severe threats to the IoT ecosystem (Gazet, 2010). Ransomware is a type of malware that encrypts files to extort money from victims. Hive (2023a) and LockBit (2023b) are two emerging ransomware examples that target different systems, including Linux-based devices, and leverage a ransomware-as-a-service model. They have caused havoc in various industries, including healthcare, technology, and education. An example of such an attack occurred on August 15, 2021, when the Hive ransomware disrupted the daily operations of three hospitals by encrypting their IT infrastructure, forcing them to cancel essential surgical procedures and examinations.

Different options are available today in the literature to detect ransomware infections. Some are focused on static approaches looking at signatures, but they can be easily overpassed using obfuscation mechanisms. Then, dynamic strategies using behavioral fingerprinting were proposed to solve some limitations of static ones (Sánchez Sánchez et al., 2021). Within this family, manually defined policies of-

* Corresponding author.
E-mail address: huertas@ifi.uzh.ch (A. Huertas Celdrán).

https://doi.org/10.1016/j.cose.2023.103510
Received 29 March 2023; Received in revised form 20 July 2023; Accepted 24 September 2023
Available online 28 September 2023

fer lightweight solutions suitable for resource-constrained devices, but creating policies requires time, and specialized expertise (Huertas Celdrán et al., 2022). Most recently, with the increasing usage of Machine Learning (ML) and Deep Learning (DL) in cybersecurity, new intelligent and promising solutions have been proposed to detect ransomware.

Despite the benefits of previous solutions, the following challenges are still open and need more effort. Only a limited amount of research focuses on detecting and classifying ransomware on resource-constrained devices running Linux. Solutions focused on dynamic and intelligent detection (the most promising approach, as previously mentioned) do not study and compare the detection performance of heterogeneous behavioral data sources. Furthermore, most works only examine detection performance and ignore their resource consumption. Finally, there is insufficient research comparing the detection performance and consumption of resources of policy-based methods versus ML anomaly-based solutions.

To address the previously mentioned challenges, the work at hand has the following contributions:

- The design and implementation of an intelligent and lightweight framework able to detect anomalies produced by zero-day ransomware samples and classify well-known families of them. The proposed framework considers behavioral fingerprinting to monitor the usage of resources, kernel events, and system calls produced by resource-constrained devices. After that, unsupervised and supervised ML/DL models are trained to detect malicious behaviors of ransomware samples. The code of the framework is available at Shushack (2023b,a).
- The deployment of the framework on a Raspberry Pi 3 Model B acting as an IoT spectrum sensor of a publicly available and well-known crowdsensing platform called ElectroSense (2023). Then, the Raspberry Pi was infected with three heterogeneous and recent ransomware samples: DarkRadiation, RAASNet, and Ransomware-PoC.
- The execution of a pool of experiments to measure the detection capabilities and consumption of CPU, RAM, and time of the proposed framework while detecting the previous three ransomware samples in the Raspberry Pi. This work provides a detailed analysis of the detection performance and consumption of resources produced by the different configurations of the framework in terms of data sources and ML/DL models. Furthermore, the framework performance was compared to the one obtained by a rule-based system.

The remainder of this work is structured as follows. Section 2 reviews related work dealing with dynamic ransomware detection systems. While Section 3 presents the design of the proposed framework, Section 4 shows its implementation details. Then, Section 5 introduces the device and ransomware samples used to validate the framework. Section 6 evaluates its detection performance and consumption of resources in the previous scenario. Finally, Section 8 draws some conclusions and future steps.

2. Related work

This section reviews how the literature has tackled the ransomware detection field, paying special attention to the solutions dealing with resource-constrained devices.

Ahmed et al. (2022) proposed an intelligent and industrial IoT framework for detecting ransomware in the early stages of infection. System calls were collected by running benign and ransomware samples in a virtual sandbox environment. Six different ML classifiers were employed, achieving a maximum accuracy of 98.64% with a low false positive rate of 1.7%. Another ML-based ransomware detection framework was proposed by Almousa et al. (2021). In this work, 249 different ransomware-specific system API calls and 229 benign API calls were extracted by running 58 ransomware and 66 benign samples in a sandbox environment for 10 minutes each. As a result, the highest ransomware detection accuracy of 99.18% was achieved using the k-nearest neighbors (kNN) algorithm.

By extracting Windows invocation sequences, the authors of Bae et al. (2020) created a framework that can distinguish between ransomware, malware, and benign files. This feat was achieved using six different ML algorithms for multi-class classification and yielded an accuracy of 98.65%. Poudyal and Dasgupta (2021) used a hybrid reverse engineering technique to extract ransomware features from three different levels. These features include a list of the Dynamic Link Libraries (DLLs) called by the ransomware code, function calls, and the assembly instructions utilized by the malicious software. The ransomware detector combines a data mining approach with Natural Language Processing (NLP) to create a feature database and uses ML for classification. The highest ransomware detection accuracy of 99.72% with a 0.3% false positive rate was achieved with supervised learning using Support Vector Machine (SVM).

Almomani et al. (2021) extracted API and permission features from decompiled Android package (APK) files containing ransomware. For classification, the SVM algorithm was deployed alongside an oversampling technique. As a result, an accuracy of 97.5% was reported. Imtiaz et al. (2021) compared the effectiveness of detecting malware, including ransomware affecting smartphones, using DL and ML malware detection approaches. The model data set included dynamic features such as API calls and power usage and static features such as intents and permissions. The results showed that a DNN outperformed other ML techniques in detecting and identifying malware on a static and dynamic layer. Another technique to detect ransomware by monitoring the energy consumption of smartphones was proposed by Azmoodeh et al. (2018). By subsampling the collected power consumption data and applying the kNN algorithm with dynamic time warping (DTW), a detection rate of 94.27% was achieved.

Rhode et al. (2018) tried to identify 3000 ransomware samples by employing a behavior-based model analyzing a snapshot of performance counter benchmarks such as memory and CPU usage in the early stages of infection. Based on recurrent neural networks, this approach correctly identified ransomware with a 94% detection rate in the first 10 seconds of file execution. A dynamic analysis detection approach was proposed by Ramesh and Menen (2020) monitoring the changes regarding system resources, the retention state of applications, and file operations/movement. A Finite State Machine (FSM) was utilized for detecting ransomware. A state change is triggered whenever an anomaly in the monitored features is identified. Reaching one of three final states indicates a ransomware infection on the system. The paper achieved a breathtaking 99.5% accuracy and 0% false positive rate.

RAPPER is a two-step ransomware detection approach introduced by Alam et al. (2020). First, a Deep Neural Network (DNN) was trained to detect anomalies with time-series data of monitored HPC events representing normal device behavior. Fast Fourier Transformation (FFT) was applied to transform the time domain values into frequency domain values. This was done to help identify repetitive patterns triggered by a ransomware infection, such as opening, encrypting, and closing a file. In the last step, the transformed data was fed into a second DNN. The proposed framework was able to detect the WannaCry ransomware in 5.313 seconds.

Berrueta et al. (2022) compared the ransomware detection effectiveness of three ML algorithms by searching for specific read and write patterns in the network communication (encrypted and clear-text file sharing) between clients and file servers. The data collected to train the models include both traffic generated by benign programs and ransomware while accessing and operating on shared network file servers. An astonishing detection accuracy of 99.8% and a low false-positive rate of 0.004% was reported using neural networks. Another network-based approach was proposed by Almashhadani et al. (2019), where packet and flow-based network traffic was monitored to detect ransomware.

Table 1
Overview of Ransomware Detection Techniques.

Work (year)	OS	Analysis	Approach	Algorithm	Domain	Accuracy
Ahmed et al. (2022)	Windows	Dynamic	ML (Classification)	LR, DT, kNN, SVM, AD, RF	Software & Processes	98.64%
Almousa et al. (2021)	Windows	Dynamic	ML (Classification)	RF, SVM, kNN	Software & Processes	99.18%
Bae et al. (2020)	Windows	Dynamic	ML (Classification)	RF, LR, NB, SGD, KNN, SVM	Software & Processes	98.65%
Poudyal and Dasgupta (2021)	Windows	Hybrid	ML, DL (Classification)	LR, AD, SVM, NN, RF, NLP	Other	99.72%
Almomani et al. (2021)	Android	Static	ML (Classification)	SVM	Other	97.50%
Imtiaz et al. (2021)	Android	Hybrid	ML/DL (Classification)	DNN	Resource Usage, Other	93.40%
Azmoodeh et al. (2018)	Android	Dynamic	ML, DL (Classification)	kNN, SVM, NN, RF	Resource Usage	94.27%
Rhode et al. (2018)	Windows	Dynamic	ML/DL (Classification)	RF, KNN, SVM, MLP, DT, RNN	Resource Usage	94.00%
Ramesh and Menen (2020)	Windows	Dynamic	Knowledge (AD)	Finite-State Machine	Resource Usage	99.50%
Alam et al. (2020)	Linux	Dynamic	DL (AD)	DNN (LTSM)	HPC	-
Berrueta et al. (2022)	Windows	Dynamic	ML, DL (Classification)	DT, TEs, NN	Network	99.80%
Almashhadani et al. (2019)	Windows	Dynamic	ML (Classification)	RF, RT, BN	Network	97.92 - 97.08%
Faghihi and Zulkernine (2021)	Android	Hybrid	Statistics (Classification)	-	Software & Processes	99.24%
Sharma et al. (2021)	Android	Static	ML (Classification)	GMM	Other	98.08%
Huertas Celdrán et al. (2022)	Linux	Dynamic	Rule (Classification, AD)	-	HPC, Resource Usage	89.80%
Huertas Celdrán et al. (2023a)	Linux	Dynamic	ML (Classification, AD)	RF, SVM, XGB, OC-SVM, IF, LOF	Resource Usage	100%
Sánchez Sánchez et al. (2023)	Linux	Dynamic	ML (AD)	OC-SVM, IF, LOF	Resource Usage	100%

For this, a multi-classifier detection framework based on ML was created, consisting of two parallel operating classifiers. With an accuracy of 97.92% percent, the Random Tree algorithm was found to be the most accurate in detecting packet-based ransomware network activity. At the same time, the Bayes Network model provided the highest scores in flow-based ransomware network detection with 97.08%.

Faghihi and Zulkernine (2021) proposed RansomCare, a hybrid analysis system detecting novel crypto-ransomware on smartphones and recovering lost data from an attack. The proposed framework identifies known ransomware types in a signature-based static analysis component by their hashes (SHA256). At the same time, zero-day crypto-ransomware strains were detected by looking for anomalies in file modification and deletion I/O events. This was accomplished by calculating the entropy of files that were modified or deleted and comparing it to a predefined threshold. As encrypted files are usually unstructured and contain a large amount of random data, they will also have a high entropy score, indicating a ransomware infection. An accuracy of 99.24% was observed with a false positive rate of 0.49%.

Sharma et al. (2021) extracted different ransomware features. These included permissions, images, intents, etc., from a total of 2076 ransomware and 2000 benign APKs. Using Gaussian Mixture Model (GMM), an unsupervised clustering-based ML technique as a classifier, Ransomdroid achieved a detection accuracy of 98.08%. Huertas Celdrán et al. (2022) analyzed ransomware affecting resource-constrained devices used as spectrum sensors in the crowdsensing platform Electrosense. Ransomware was identified by finding anomalies in Memory usage, CPU usage, I/O activities, HPC, and kernel events of the device. The administrator's policies would then classify the device behavior as malicious or normal. Anomalies generated by two different ransomware samples were correctly identified with an 89.80% true positive rate. The same authors also explored ransomware detection, together with other malware types, in spectrum sensors using kernel events and ML/DL classification and anomaly detection (Huertas Celdrán et al., 2023a; Sánchez Sánchez et al., 2023). They were able to perfectly identify the presence of a simple ransomware sample, namely Ransomware-Poc, employed usually for research purposes.

As can be seen in Table 1, despite the advances in ransomware detection made by the previous solutions, the following challenges are still open. First, most of the research dealing with ransomware detection has been conducted on platforms running Windows. There is little research into detecting ransomware affecting resource-constrained and Linux-based devices. Besides, dynamic and hybrid approaches combined with supervised ML/DL models are the most popular solutions in the reviewed literature, while anomaly detection is poorly studied. In other words, novel and sophisticated ransomware samples launching zero-day attacks, such as those provided as a customized service in the dark web, would not be detected. Additionally, different behavioral data sources have been used for detecting ransomware (resource usage and processes are the most popular). Still, there is no work comparing their efficiency regarding the detection and consumption of CPU, RAM, and time in resource-constrained devices. Finally, none of the papers compare policy-based approaches with ML/DL ones from detection and consumption points of view.

3. Framework design

This section describes the design details of the intelligent and behavioral-based framework used to detect and classify ransomware affecting resource-constrained devices. It includes an overview of the elements that make up the framework and its key design decisions.

The design of the framework follows a distributed architecture that allows its deployment on heterogeneous scenarios and devices with diverse hardware and software configurations. This design decision ensures that the workload on resource-constrained sensors remains as low as possible, providing the option of moving the ML/DL part to an external server. However, if the device is powerful enough, the whole framework can be deployed on the same device providing additional data privacy (behavioral data does not leave the device). Although the framework follows a distributed architectural design, it should be noted that it may not be suitable for IoT devices with extremely low computational power, as these devices may lack the ability to effectively monitor their behavior. Fig. 1 presents an overview of the framework architectural design with its two main layers, modules, and components. More in detail, the following two layers make up the framework:

1. *Fingerprinting Layer*, in charge of monitoring the internal behavior of resource-constrained devices. The correct selection of behavioral dimensions and events is key to achieve accurate detections of heterogeneous ransomware samples.
2. *Data Analysis Layer*, focused on preprocessing behavioral data as well as detecting and classifying ransomware families in almost real time. A proper data preprocessing selecting representative features is key to achieve accurate detection performance.

3.1. Fingerprinting layer

This layer hosts three modules that periodically monitor heterogeneous behavioral data from resource-constrained devices. Since the encryption phase of ransomware samples might consume a relevant amount of computational resources and manage (open, read, close) a vast number of files, the three modules focus on: Resource Consumption (RES), Kernel (KERN), and System Calls (SYS). The text below provides details of each module.

Table 2
Events Gathered by the RES Monitor.

time	seconds	net:netif_rx	task:task_newtask
ioread	block:block_bio_frontmerge	timer:tick_stop	sched:sched_stat_runtime
iowrite	block:block_dirty_buffer	sched:sched_process_exec	timer:timer_cancel
ioreadbytes	block:block_split	sched:sched_waking	timer:timer_init
iowritebytes	block:block_touch_buffer	gpio:gpio_value	timer:timer_start
ioreadtime	ext4:ext4_es_lookup_extent_enter	irq:softirq_exit	workqueue:workqueue_execute_start
iowritetime	ext4:ext4_ext_load_extent	pagemap:mm_lru_activate	branch-instructions
iobusytime	ext4:ext4_writepages_result	rpm:rpm_return_int	branch-misses
read_merge	ext4:ext4_journal_start	fib:fib_table_lookup	bus-cycles
write_merge	filemap:mm_filemap_add_to_page_cache	raw_syscalls:sys_enter	cache-misses
memory	filemap:mm_filemap_delete_from_page_cache	random:credit_entropy_bits	cache-references
net_in	jbd2:jbd2_handle_stats	kmem:kfree	cpu-cycles
net_out	ext4:ext4_da_update_reserve_space	kmem:kmem_cache_alloc	instructions
pkt_in	ext4:ext4_sync_file_enter	kmem:mm_page_alloc_zone_locked	context-switches
pkt_out	jbd2:jbd2_checkpoint_stats	kmem:mm_page_free	armv7_cortex_a7/l1d_cache_wb/
err_in	ext4:ext4_free_inode	mmc:mmc_request_done	armv7_cortex_a7/l1d_tlb_refill/
err_out	ext4:ext4_evict_inode	writeback:global_dirty_state	armv7_cortex_a7/l1i_cache/
drop_in	ext4:ext4_releasepage	writeback:sb_clear_inode_writeback	armv7_cortex_a7/l1i_cache_refill/
drop_out	ext4:ext4_unlink_enter	writeback:wait_on_page_writeback	armv7_cortex_a7/l1i_tlb_refill/
cpu	block:block_bio_remap	napi:napi_poll	armv7_cortex_a7/l2d_cache/
cpu-migrations	LLC-load-misses	armv7_cortex_a7/br_mis_pred/	armv7_cortex_a7/l2d_cache_wb/
minor-faults	LLC-loads	armv7_cortex_a7/br_pred/	armv7_cortex_a7/ld_retired/
page-faults	LLC-store-misses	armv7_cortex_a7/bus_cycles/	armv7_cortex_a7/mem_access/
L1-dcache-load-misses	LLC-stores	armv7_cortex_a7/cpu_cycles/	armv7_cortex_a7/pc_write_retired/
L1-dcache-loads	branch-load-misses	armv7_cortex_a7/exc_return/	armv7_cortex_a7/st_retired/
L1-dcache-store-misses	branch-loads	armv7_cortex_a7/exc_taken/	armv7_cortex_a7/unaligned_ldst_retired/
L1-dcache-stores	dTLB-load-misses	armv7_cortex_a7/inst_retired/	armv7_cortex_a7/cid_write_retired
L1-icache-load-misses	dTLB-store-misses	armv7_cortex_a7/l1d_cache/	tcp:tcp_probearmv7_cortex_a7/br_immed_retired/
L1-icache-loads	iTLB-load-misses	armv7_cortex_a7/l1d_cache_refill/	

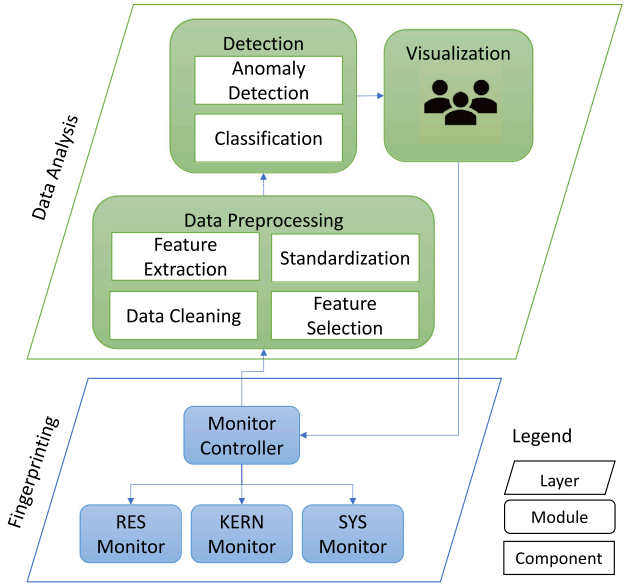


Fig. 1. Architectural Design of the Framework.

- The *RES Monitor* gathers specific behavioral events linked to the device hardware. These events include information on CPU and memory usage, disk utilization, kernel tracepoint events, or hardware performance counters, among others. Table 2 shows the list of events considered by the RES Monitor.
- The *KERN Monitor* focuses on specific aspects of the device and tracks events related to disk I/O, CPU, kernel memory, and system call statistics. Events gathered by the KERN Monitor are shown in Table 3.
- The *SYS Monitor* collects the system calls for the entire device to properly monitor the requests made from the user to the kernel mode of the operating system. This monitor intends to detect advanced ransomware families encrypting small portions of files.

Apart from the previous three modules, the Fingerprinting layer hosts the *Monitor Controller* module to provide the following functionality:

- Configure Monitors. Final users can initiate, stop, and control the monitoring components using the Monitor Controller. Furthermore, they customize different aspects of the monitors, such as the monitoring time, which monitor to run in parallel, or the location of the data analysis layer (same device or external server).
- Establish Secure connections. The Monitor Controller enables secure data exchange between two machines if the framework is deployed in a decentralized fashion by using asymmetric encryption.
- Send Data. It sends behavioral data collected by the RES, KERN, and SYS monitors to the Data Analysis Layer for further processing. In addition, it sends metadata about the gathered data to the Data Analysis layer to initiate training and testing processes. This information includes the locations where the data is stored or the monitors used for the monitoring session, among others.
- Live Commands. It allows final users to start online monitoring sessions on the device. For that, it collects evaluation data for a defined time frame and forwards it to the Data Analysis layer for evaluation.

3.2. Data analysis layer

This layer is in charge of preprocessing, in an automatic fashion, the collected behavioral data, detecting and classifying ransomware families, and providing a graphical interface for final users. The following three modules provide that functionality: *Data Preprocessing*, *Detection*, and *Visualization*

The *Data Preprocessing* module receives the raw behavioral data provided by the Monitor Controller and prepares it for the subsequent ML/DL pipeline. More in detail, behavioral data gathered by the RES and KERN Monitors contains only numerical values of the events indicated in Table 2 and Table 3. However, the system calls gathered by the SYS Monitor contain both strings and numbers. Therefore, data pre-

Table 3
Events Gathered by the KERN Monitor.

time	gpio:gpio_value	random:get_random_bytes	
timestamp	ipi:ipi_raise	random:mix_pool_bytes_nolock	
seconds	irq:irq_handler_entry	random:urandom_read	
connectivity	irq:softirq_entry	raw_syscalls:sys_enter	timer:hrtimer_start
alarmtimer:alarmtimer_fired	jbd2:jbd2_handle_start	raw_syscalls:sys_exit	timer:timer_start
alarmtimer:alarmtimer_start	jbd2:jbd2_start_commit	rpm:rpm_resume	udp:udp_fail_queue_rcv_skb
block:block_bio_backmerge	kmem:kfree	rpm:rpm_suspend	workqueue:workqueue_activate_work
block:block_bio_remap	kmem:kmalloc	sched:sched_process_exec	writeback:global_dirty_state
block:block_dirty_buffer	kmem:kmem_cache_alloc	sched:sched_process_free	writeback:sb_clear_inode_writeback
block:block_getrq	kmem:kmem_cache_free	sched:sched_process_wait	writeback:wbc_writepage
block:block_touch_buffer	kmem:mm_page_alloc	sched:sched_switch	writeback:writeback_dirty_inode
block:block_unplug	kmem:mm_page_alloc_zone_locked	sched:sched_wakeup	writeback:writeback_dirty_inode_enqueue
cachefiles:cachefiles_create	kmem:mm_page_free	signal:signal_deliver	writeback:writeback_dirty_page
cachefiles:cachefiles_lookup	kmem:mm_page_pcpu_drain	signal:signal_generate	writeback:writeback_mark_inode_dirty
cachefiles:cachefiles_mark_active	mmc:mmc_request_start	skb:consume_skb	writeback:writeback_pages_written
clk:clk_set_rate	net:net_dev_queue	skb:kfree_skb	writeback:writeback_single_inode
cpu-migrations	net:net_dev_xmit	skb:skb_copy_datagram_iovec	writeback:writeback_write_inode
cs	net:netif_rx	sock:inet_sock_set_state	writeback:writeback_written
dma_fence:dma_fence_init	page-faults	task:task_newtask	
fib:fib_table_lookup	pagemap:mm_lru_insertion	tcp:tcp_destroy_sock	
filemap:mm_filemap_add_to_page_cache	preemptirq:irq_enable	tcp:tcp_probe	
	qdisc:qdisc_dequeue		

processing is necessary, as ML and DL algorithms rely on numerically expressed data.

On the one hand, for behavioral data gathered by the RES and KERN monitors, a general preprocessing procedure cleans the data, removes samples with constant values, removes highly correlated features, and normalizes their values. On the other hand, the Bag-of-Words approach is applied to the system calls gathered by the SYS monitor. More in detail, system calls parameters, timestamps, and other metadata (apart from the system call name) are deleted. Then, a bag (document or dictionary) with the syscalls (terms or grams) monitored in all time windows is built. Then, it creates a vector with the syscalls executed per time window. Each bag entry can contain one or more syscalls depending on the selected *n-Gram*, and each vector position represents n syscalls. Finally, there are two ways to fill vector positions (extract features). The first is called *Frequency* and counts the number of times that each *n-Gram* occurs in the time window. The second is *Term Frequency-Inverse Document Frequency (TF-IDF)* and reflects the importance of the *n-Gram* in the vector by considering not only the repetitions but also the number of different syscalls per time window. Equation (1) presents the calculation of the TF-IDF value for a term t in a document d, which belongs to the vector D, where n stands for the number of terms in d, $f_{i,d}$ stands for the frequency of term i in d, N is the number of documents in the vector, and $|\{d \in D : t \in d\}|$ is the number of documents containing the term t.

$$tf - idf(t, d, D) = \frac{f_{t,d}}{\sum_{i=1}^{n} f_{i,d}} * \log \frac{N}{|\{d \in D : t \in d\}|} \quad (1)$$

The hashing approach is another feature extraction method similar to bag-of-words. Rather than just storing a dictionary of tokens (vocabulary), this method creates a sparse matrix with the token occurrence counts, utilizing hashing. This hashing trick can be memory efficient when dealing with large data sets, as this extraction method does not require holding a large vocabulary in memory (HashingVectorizer, 2023).

Once data is ready for the ML/DL algorithms, the *Detection* component is in charge of training and evaluating ML and DL models. This component is split into two main components. The first one focuses on anomaly detection, while the second deals with classification. On the one hand, the anomaly detection module can detect zero-day ransomware attacks. For this purpose, the framework employs both ML and DL algorithms, and the training data represents the device normal behavior (without ransomware infection). Then, testing data can contain both normal and under-attack behaviors. On the other hand, the classification component identifies the ransomware family and its behavior. For that, this method requires training data from each ransomware family as well as normal device behavior. Furthermore, this data must be labeled and balanced. Finally, it is important to remark that the anomaly detection and classification algorithms are evaluated in parallel during live monitoring to determine the device status (infected versus not infected) and the ransomware family (in case of being infected).

The last module of the Data Analysis layer is the *Visualization*, which provides final users with a graphical interface for presenting live behavioral data of devices and the outputs of the Anomaly Detection and Classification components. Using this graphical interface, final users can choose the algorithm and monitoring data (RES, KERN, SYS, or combinations of them) they want to use to detect anomalies and classify behaviors. In addition, different plots help display live evaluation data.

4. Framework implementation

The proposed intelligent detection framework has been implemented as a proof-of-concept for Linux-based and single-board devices. As mentioned, the framework is generic enough and suitable for heterogeneous devices with different hardware and software configurations since it follows a distributed architecture comprising two main layers. The only requirement to deploy the framework is that devices must be able to monitor the selected data sources. The implementation details of the main components belonging to each layer are provided below.

4.1. Fingerprinting layer

To capture behavioral data on Linux-based and single-board devices, three different monitoring scripts are implemented using the Linux *Perf* performance analysis tool.

The RES monitoring script gathers the events shown in Table 7. More in detail, this module utilizes the library *Psutil* and the Linux utility Perf. The main code logic is split into classes, each tracking specific device metrics. There are two customization options for data output: CSV file creation or asynchronous messaging. The script gathers each event every five seconds. Therefore, every data sample represents five seconds of monitoring data. The time-window decision was made according to different experiments and solutions existing in the literature (Huertas Celdrán et al., 2023b; Sánchez Sánchez et al., 2022).

The KERN Monitor script monitors the events shown in Table 8, which are related to disk I/O, CPU, kernel memory, and system calls and is written in bash. This module is executed every five seconds (as in RES monitor, this decision is made according to preliminary work (Huertas Celdrán et al., 2023b; Sánchez Sánchez et al., 2022)). A DNS server

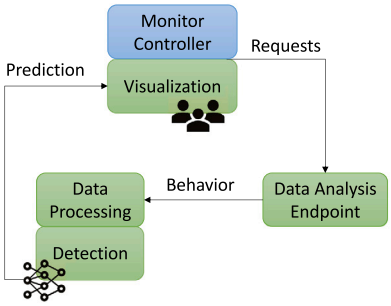


Fig. 2. Overview of modules and interactions performed in the Data Analysis Layer.

is pinged as a first step to confirm that the device is connected to the internet (needed by some ransomware families). Then, Perf monitors the events defined by the user (see Section 3) for the specified time window. Finally, the SYS Monitor bash script records system call information repeatedly over a ten-second interval. The Perf utility uses the *trace* command to capture all system calls executed on the sensor and saves these in a log file.

The Monitor Controller manages the RES, KERN, and SYS monitoring scripts, which are implemented as *systemd* services. The following aspects identify the key benefits of deploying the scripts as a service.

- Provides a simple way to start, stop and control the monitoring scripts.
- Provides the ability to easily manage the monitoring scripts as services with *systemctl*.
- Allows for defining the behavior after a program failure or reboot of the device.
- Streamlines the integration of the monitoring scripts into the Monitor Controller.

The Monitor Controller provides multiple customization possibilities while controlling the entire monitoring process. It also manages and records the status and elapsed time of current and past monitoring sessions. This command-line interface tool is written in Python 3.5 and uses an *SQLite* database for data persistence. In addition, the *click* Python module is utilized for argument parsing and allows for simple user commands to be supplied, such as show, collect and send. Finally, the *requests* Python package facilitates communication with the server via REST API calls. A simple installer script was also developed to download and install the required dependencies automatically.

4.2. Data analysis layer

The Data Analysis layer has been developed using the *Flask* open-source web micro-framework for Python. Flask is a well-documented and user-friendly framework that integrates well with ML/DL-specific Python packages. Fig. 2 provides an overview of the design architecture, and a more detailed description of each component is shown below.

The Data Analysis Endpoint module exposes the REST endpoints through which the Monitor Controller module and final users can communicate with the Data Analysis Layer to detect and classify ransomware families. Its primary function is to manage the request flow by acting as an intermediary between the Visualization and the ML/DL models trained in the Detection module. As can be seen in Fig. 2, the Data Analysis Endpoint accepts requests from the Monitor Controller and the Visualization modules. The received data is then validated and transformed into a valid format. Finally, it is forwarded to the Data Processing and Detection modules for further processing. Devices and final users also have their own endpoints. Device-specific endpoints forward incoming request data to the Detection module. A response is then directly sent back to the device Monitor Controller. User endpoints handle users' inputs from the Flask graphical interface (Visualization Layer). Behavioral data is exchanged between the Data Analysis Endpoint, Detection, and Visualization to detect ransomware attacks and show the models predictions. Finally, the Detection module contains the core layer logic, including interactions with the dataset, data preprocessing, and ML training/evaluation. The detection module uses *Pyod* and *Tensorflow* as Python packages for ML/DL. After storing and preprocessing the incoming behavioral data, it is split into training (90%) and test (10%), and both sets are standardized. Then, the ML/DL algorithms are trained.

Table 4
Complexity of ML/DL Algorithms (n = number of points in the Training set, d = dimensionality of the data, s = number of Support Vectors, i, j, k, l = number of nodes of the Autoencoder layers, t = training epochs).

Algorithm	Complexity Order
Isolation Forest	$O(n * log_2(n))$
One Class-Support Vector Machine	$O(n^2)$
Local Outlier Factor	$O(k * n * d)$
Autoencoder	$O(nt * (ij + jk + kl))$
Logistic Regression	$O(nd)$
Decision Tree	$O(n * log(n) * d)$
Support Vector Machine	$O(n^2)$
Random Forest	$O(n * log(n) * d * s)$

In the case of ML-based anomaly detection, the framework has considered One-Class Support Vector Machine (OC-SVM), Local Outlier Factor (LOF), and Isolation Forest (IF). For DL-based anomaly detection, Autoencoders are utilized. Table 4 shows the computational complexity of each algorithm during training. Once the algorithms are selected, the outliers present in the training data set, also known as the contamination factor, are set to 5%. Next, using the training data set, the anomaly detector ML/DL models are trained. Then, the test data is evaluated with the trained models to predict its anomaly scores. The training process of DL anomaly detection is performed similarly. As the number of features can vary depending on the training data set, the dimensions of the Autoencoder layers are calculated before instantiation. In addition, the TensorFlow Early Stopping Callback prevents overfitting the model by stopping the training procedure when the validation error (mean absolute error) no longer decreases. After training the Autoencoder, the validation data set is processed, and its reconstruction errors are calculated. The Autoencoder uses two different thresholds for detecting anomalies. The first threshold is specified by the interquartile range rule, while the second is defined as three standard deviations away from the mean of the reconstruction errors (as suggested in Patro and Sahu, 2015). The anomaly detection thresholds and reconstruction errors are then used to assign the anomaly scores to the validation data (0 = normal, 1 = abnormal).

In the case of classification, the primary function is to classify a data sample appropriately. Compared to anomaly detection, this method requires training data from each class. For multi-class classification, the supervised ML algorithms Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), and Random Forest (RF) are utilized. Table 4 shows the computational complexity of each algorithm during training. Unlike anomaly detection, classification algorithms require labeled data and recommend a balanced number of samples for all classes. The procedure is similar to that used in training ML anomaly detection algorithms. Each pre-processed incoming data set is split (following the same 90/10% rate as in the past) and standardized. After this, the classifiers are trained (using the training data with its respective class labels). Finally, a classification report is generated by comparing the actual class labels with the predicted labels of the validation data.

5. Validation scenario: crowdsensing platform affected by ransomware

This section presents a real scenario where the proposed framework has been deployed and evaluated. In particular, crowdsensing is one of the most promising applications of the IoT paradigm, where large groups of individuals collaborate in a crowdsourcing fashion, typically leveraging resource-constrained devices. Spectrum sensing, consisting of monitoring the electromagnetic environment with sensors, has grown in prominence due to its broad applicability in consumer, regulatory, and military applications. In this context, the ElectroSense initiative marks an ideal solution due to its network of affordable radio sensors monitoring spectrum data for further analysis. Becoming part of this network requires a single-board computer such as a Raspberry Pi with a stable internet connection linked via a dongle (Radio Frontend) to an antenna. The data collected by the different sensors is sent to a central backend server and displayed to users on the ElectroSense website for further analysis (Rajendran et al., 2018).

Since Raspberry Pis are versatile single-board computers that in the past already were victims of ransomware attacks, this work has considered ElectroSense as a validation scenario. More in detail, infected Raspberry Pis of ElectroSense could have their data encrypted and, therefore, could not work correctly, significantly impacting the platform overall operation and accessibility to spectrum data. Therefore, this work has considered a Raspberry Pi 3 Model B with 1 GB of RAM and a Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU. The ElectroSense Software, which is based on Raspbian, is utilized as an operating system. The Raspberry Pi is connected via a Radio Frontend (RTL-SDR Silver v3) to an antenna, creating a suitable sensor for the ElectroSense platform. To guarantee a continuous internet connection, the device is connected via Ethernet cable to a Broadband internet provider.

As a proof-of-concept, the RansomwarePoC, DarkRadiation, and RAASNet ransomware families have been used to encrypt the Raspberry Pi and evaluate the detection and classification performance of the proposed framework. At this point, it is essential to mention that these three ransomware samples present differences in library installation, encryption algorithms, encryption rate, encrypted files, and communications with external servers. These aspects cover the main differences between ransomware samples in the wild.

- *Ransomware-PoC* is a proof of concept open-source Python ransomware payload that can be downloaded from the following GitHub repository (Ransomware-PoC, 2023). This software allows malicious users to encrypt or decrypt files by providing a starting directory. The ransomware scans a selected directory and its sub-directories for files with the proper extensions (list of file extensions in the source code). Then, it encrypts an AES (256-bit) key with an RSA public key, which is used for file encryption. As this ransomware allows the user to decrypt their files after encryption, the private RSA server key is also hardcoded into the payload. In this work, the Ransomware-PoC source code was modified slightly to display the starting and ending timestamps of the encryption phase.
- *DarkRadiation* is a ransomware that targets Linux-based systems. This sophisticated ransomware is implemented entirely in bash, using the messaging application called Telegram, as its command-and-control server. DarkRadiation employs an SSH worm that downloads the ransomware payload after connecting with the victim's device. The ransomware communicates with the attackers using the Telegram API and encrypts files using the OpenSSL AES algorithm (256-bit key length). As a first step, the ransomware checks to determine if it has root privileges and then downloads all the needed dependencies using curl and OpenSSL. Next, changes in user activity (new logins, logouts) are transmitted to the command and control server (Lakshmanan, 2021).
- *RAASNet*. As ransomware attacks have become more lucrative and prevalent in recent years, a new type of service, Ransomware-as-a-Service (RaaS), has appeared on the darknet. This service is offered as a franchise model and is marketed toward attackers with little or no previous programming experience. This new way of selling a tool to commit a crime enables ordinary (non-technical) individuals to participate in the ransomware economy (Meland et al., 2020). In this context, an open-source Ransomware-as-a-Service written in Python called RAASNet shows how simple it is to create and deploy ransomware and can be downloaded from GitHub (RAASNet, 2021). This cross-platform tool offers an intuitive graphical user interface, allowing users to develop customized ransomware. In addition, it includes a built-in command and control server for receiving private encryption keys.

6. Experiments

This section presents a pool of experiments that evaluates the performance of the proposed framework while detecting and classifying the ransomware families introduced in Section 5. It is important to mention that during the experiments, diverse malicious behaviors considered by each ransomware sample (such as the installation of different dependencies, the encryption algorithm and rate, and the communications with external servers) have been considered.

6.1. Evaluation metrics

Specific metrics help evaluate the ML and DL algorithms. Different performance indicators can be employed depending on the training category (supervised, unsupervised). This work uses confusion matrices as a basis for the evaluation. The true positives (TP) refer to the number of ransomware infections correctly identified by the ML/DL algorithms, while the true negatives (TN) reflect correctly identified instances of normal behavior. False positive (FP) and false negative (FN) values indicate the number of incorrect predictions.

Regarding anomaly detection, the following two performance metrics have been considered:

1. **True Positive Rate (TPR)**. It indicates the proportion of correctly predicted ransomware infections from all actual ransomware infections:

$$TPR = \frac{TP}{TP + FN}$$

2. **True Negative Rate (TNR)**. It indicates the proportion of correctly predicted instances of normal behavior from all actual instances of normal behavior:

$$TNR = \frac{TN}{TN + FP}$$

Similarly, a confusion matrix can be created for multi-class classification. Matrix labels would indicate different classes instead of the state of infection. In classification problems, the **TPR**, also known as **recall**, indicates the number of correct class predictions divided by the total number of class instances. The following performance metrics have been used to evaluate classification performance:

1. **Accuracy** compares the accurate predictions according to all predictions:

$$Accuracy = \frac{TP + TN}{TN + TP + FP + FN}$$

2. **Precision** measures the overall precision of correctly predicting a class out of all predictions of this class:

$$Precision = \frac{TP}{TP + FP}$$

Table 5
Anomaly Detection Performance.

Monitor	Features	Algorithm (ML/DL)	Training Time (s)	Testing time per sample (s)	TNR Val (%)	TPR Dark (%)	TPR PoC (%)	TPR RAAS (%)
RES Monitor	114	IF	0.79	0.030	98.10	95.88	96.94	75.78
	114	OC-SVM	2.05	0.001	96.19	100.00	100.00	94.53
	114	LOF	0.19	0.240	93.33	100.00	100.00	100.00
	114	Autoencoder STD	16.49	0.160	97.14	100.00	100.00	38.28
	114	Autoencoder IQR	16.49	0.160	96.19	100.00	100.00	80.47
KERN Monitor	77	IF	0.62	0.033	98.21	94.74	93.52	55.30
	77	OC-SVM	2.20	0.001	94.64	100.00	100.00	90.90
	77	LOF	0.16	0.009	90.17	100.00	100.00	100.00
	77	Autoencoder STD	17.49	0.083	98.21	100.00	100.00	30.30
	77	Autoencoder IQR	17.49	0.083	98.21	100.00	100.00	56.06
Hashing 1-gram SYS	1024	IF	7.81	0.032	96.29	91.89	67.57	52.69
	1024	OC-SVM	8.72	0.002	0.00	100.00	100.00	17.20
	1024	LOF	0.51	0.016	0.00	100.00	100.00	19.35
	1024	Autoencoder STD	7.04	0.102	0.00	100.00	100.00	12.90
	1024	Autoencoder IQR	7.04	0.102	0.00	100.00	100.00	27.96
Frequency 1-gram SYS	2443	IF	9.36	0.038	96.29	86.49	56.76	41.94
	2443	OC-SVM	24.03	0.004	100.00	100.00	2.70	15.05
	2443	LOF	0.93	0.029	100.00	62.16	0.00	6.45
	2443	Autoencoder STD	36.82	0.098	100.00	100.00	37.84	30.11
	2443	Autoencoder IQR	36.82	0.098	97.53	100.00	59.46	64.51
TF-IDF 1-gram SYS	2443	IF	9.68	0.051	96.29	89.18	67.57	51.61
	2443	OC-SVM	24.49	0.005	100.00	100.00	8.11	17.20
	2443	LOF	1.00	0.038	100.00	56.76	1.35	9.68
	2443	Autoencoder STD	36.27	0.092	100.00	100.00	51.35	26.88
	2443	Autoencoder IQR	36.27	0.092	96.29	100.00	71.62	79.57

3. **F1-Score** combines the recall and precision metrics:

$$\text{F1-Score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

6.2. Anomaly detection experiment

The anomaly detection experiment aims to evaluate and compare the performance and effectiveness of ML/DL models detecting anomalies produced by ransomware. Furthermore, this experiment attempts to identify the best ML/DL model and behavioral dimension for detecting ransomware.

The first step was to gather the behavioral data modeling the normal behavior of the Raspberry Pi, which is required for training ML/DL models. More in detail, the Monitor Controller and the RES, KERN, and SYS monitoring scripts were deployed and executed on the Raspberry Pi 3, acting as an ElectroSense sensor. The monitoring session was performed for 15 hours to ensure an adequate training data sample size.

Then, each ransomware sample was executed, and the three monitoring scripts ran for 15 minutes in parallel. More in detail, RAASNet and Ransomware-PoC begin the file encryption process shortly after execution starts (after approximately two seconds). In the case of DarkRadiation, it first installs several dependencies before starting the file encryption process. To compare the three ransomware strains, the monitoring procedure of DarkRadiation was initiated after receiving the *encrypt home files began* message from the C&C. Ransomware-PoC was utilized in a second attack scenario to test the capability of encrypting the entire sensor. It took only two minutes and 27 seconds for the ransomware to crash the sensor due to the encryption of the system and library files.

The first step in the evaluation process was to verify the ML/DL algorithms to identify normal behavior on the sensor correctly. As the collected data should reflect the normal state of the sensor, a TNR close to 100% was expected. The results received from the 10% validation data sample are shown in Table 5. It can be seen how the TNR was superior to 95% in most cases. The best-performing algorithm for KERN and RES was LOF, having 93% TNR, the fastest training time, and the best ransomware detection accuracy (with 100% overall ransomware samples detection). However, when syscall hashing was performed as preprocessing, the TNR of SVM, LOF, and Autoencoder was 0%. In this case, the model always outputs *anomaly*, being unable to evaluate the provided data samples correctly. From the attack detection perspective, ML and DL models trained with system call monitoring data (Monitor SYS) detected the ransomware DarkRadiation without issue. However, most of these ML algorithms failed to identify a Ransomware-PoC or RAASNet infection. DL seems to be more promising. The DL Autoencoder with an interquartile-based threshold (IQR) using the TF-IDF system call features recognized all ransomware samples (TPR $>50\%$). Similarities emerge when comparing the detection performance of algorithms trained with the RES and KERN monitoring data. Both DarkRadiation and Ransomware-PoC were detected accurately. However, a RAASNet ransomware infection appeared more challenging for the algorithms to identify. This is most likely due to its slow encryption process (the slowest out of the ransomware samples).

As expected, the training time depends on the ML/DL model and the number of features used. DL Autoencoders showed the most extended training times, followed by One-Class SVM (OC-SVM). In terms of testing time, it is quite acceptable, being a few milliseconds for almost all models. As more features were included in the extracted system call data, the training and testing times increased.

6.3. Classification experiment

Similar to the previous experiment, the goal of this one is to evaluate and compare the performance and effectiveness of ML/DL models and behavioral data sources while classifying different ransomware families and normal behavior.

For this experiment, the following four behavioral classes were defined: normal behavior, DarkRadiation, Ransomware PoC, and RAASNet. The first step was to collect samples for each class. Therefore, each of these data collection sessions ran for approximately four hours. Note that the standard storage capacity of the sensor was insufficient to provide the required number of target files for encrypting for four hours.

Table 6
Classification Performance.

Monitor	Features	Algorithm	Training Time (s)	Testing time per sample (s)	Macro avg F1-score	Weighted avg F1-score
RES Monitor	114	Logistic Regression	50.00	0.0001	0.99	0.99
	114	Decision Tree	0.27	0.0003	0.99	0.99
	114	Support Vector Machine	0.22	0.0006	0.99	0.99
	114	Random Forest	1.50	0.0130	0.99	0.99
KERN Monitor	77	Logistic Regression	34.94	0.0002	0.99	0.99
	77	Decision Tree	0.15	0.0002	0.99	0.99
	77	Support Vector Machine	0.26	0.0004	0.98	0.98
	77	Random Forest	1.15	0.0120	0.98	0.98
Hashing 1-gram SYS	1024	Logistic Regression	124.93	0.0001	1.00	1.00
	1024	Decision Tree	0.18	0.0001	1.00	1.00
	1024	Support Vector Machine	3.03	0.0013	0.93	0.93
	1024	Random Forest	0.69	0.0159	1.00	1.00
Freq. 1-gram SYS	2084	Logistic Regression	349.24	0.0003	1.00	1.00
	2084	Decision Tree	0.21	0.0002	1.00	1.00
	2084	Support Vector Machine	7.99	0.0082	0.99	0.99
	2084	Random Forest	0.65	0.0130	1.00	1.00
TF-IDF 1-gram SYS	2084	Logistic Regression	326.19	0.0004	1.00	1.00
	2084	Decision Tree	0.25	0.0003	1.00	1.00
	2084	Support Vector Machine	7.88	0.0029	0.99	0.99
	2084	Random Forest	0.81	0.0120	1.00	1.00

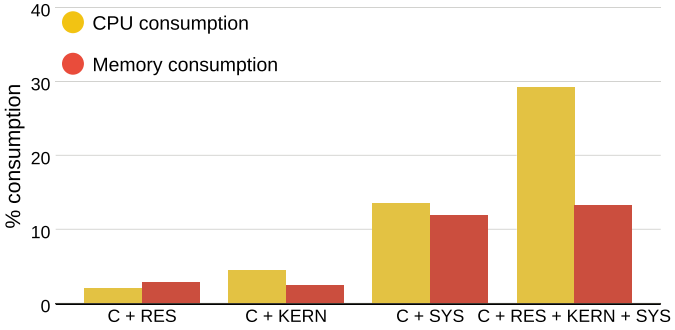


Fig. 3. Monitor Controller Resource Usage (C: Controller).

Therefore, an external solid-state drive (SSD) containing 100 gigabytes of sample *dummy* files was used and the ransomware was configured to encrypt those files and achieve a four-hour monitoring session. These include files of varying sizes and file types. Then, the data was prepared for training by randomly shuffling, splitting between training and testing, and normalizing.

Table 6 presents the macro and weighted average F1-scores calculated for the different classification algorithms. As can be seen, exceptional classification results were observed for all monitoring scripts (SYS, RES, and KERN) with an overall accuracy of 90-100%. In terms of time, DT and SVM combined with RES and KERN events are the models needing less time to be trained (less than 1 second).

6.4. Resource consumption experiment

This experiment aims to measure the CPU and RAM consumption of the framework when deployed on the Raspberry Pi acting as an ElectroSense sensor. For that, the resource consumption data was measured while running the Monitor Controller and the Data Analysis layer.

First, the resource consumption of the Monitor Controller was assessed for the four distinct monitoring configurations. Fig. 3 shows the results obtained by performing ten-minute monitoring sessions. As can be seen, the RES monitor consumes less CPU and RAM than the SYS monitor. This is due to the complexity of gathering events by each monitor and the amount of data acquired by each monitor.

Table 7
Consumption of Training & Evaluation with RES.

Category	Task	CPU (%)	RAM (%)
Processing	Preprocessing RES	17.74	50.90
Anomaly detection	Train IF	31.19	52.40
	Evaluate IF	27.22	52.40
	Train OC-SVM	26.48	52.45
	Test OC-SVM	39.74	52.50
	Train LOF	31.18	52.69
	Test LOF	56.05	53.18
	Train Autoencoder	38.41	54.59
	Test Autoencoder	30.40	55.37
Classification	Train Logistic Regression	29.38	55.38
	Test Logistic Regression	47.37	55.40
	Train Decision Tree	29.00	55.40
	Test Decision Tree	nan	nan
	Train SVM	30.84	56.99
	Test SVM	30.60	55.60
	Train Random Forest	29.35	56.05
	Test Random Forest	11.26	57.60

Secondly, the resource consumption of the Data Analysis layer running on the Raspberry Pi was evaluated by executing classification and anomaly detection training and testing procedures for each monitor (KERN, RES, and SYS). Timestamps were employed to track specific tasks, such as preprocessing, training, and evaluation. For anomaly detection, 2133 data samples were used, and 4266 for classification. 10% of the data was utilized for evaluation.

Table 7, Table 8, and Table 9 show the Raspberry Pi CPU, and RAM consumed by the framework. In terms of preprocessing, the RES monitor is the one consuming less CPU (~17%) compared to KERN (~25%) and SYS (~50%). The RAM is similar for the three monitors (~50%). Dealing with the CPU consumption during training for Anomaly Detection and Classification, no major differences exist between the algorithms used with the data gathered by RES and KERN monitor (between 26% and 38%). However, the models trained with the SYS monitor consumed more CPU (~50%). Finally, regarding RAM, there are no significant differences between the three monitors, having a consumption close to 50%.

Table 8
Consumption of Training & Evaluation with KERN.

Category	Task	CPU (%)	RAM (%)
Processing	Preprocessing KERN	25.19	45.58
Anomaly detection	Train IF	37.20	46.56
	Evaluate IF	33.90	46.6
	Train OC-SVM	31.60	46.74
	Test OC-SVM	33.75	46.8
	Train LOF	35.29	47.09
	Test LOF	63.15	47.48
	Train Autoencoder	35.26	48.98
	Test Autoencoder	4.18	49.50
Classification	Train Logistic Regression	29.38	53.70
	Test Logistic Regression	8.13	53.73
	Train Decision Tree	27.67	53.7
	Test Decision Tree	nan	nan
	Train SVM	33.12	55.63
	Test SVM	30.17	53.80
	Train Random Forest	30.89	54.17
	Test Random Forest	28.42	56.00

Table 9
Consumption of Training & Evaluation with SYS.

Category	Task	CPU (%)	RAM (%)
Proc.	Cleaning log files	51.63	45.30
	Create Corpus	55.34	48.08
	Create Count Vectorizer Freq.	52.13	48.21
	Apply Count Vectorizer Freq.	51.41	48.46
	Create TF-IDF Vectorizer	51.29	48.11
	Apply TF-IDF Vectorizer	51.44	48.58
	Create Hashing Vectorizer	nan	nan
	Apply Hashing Vectorizer	52.7	50.37
AD	Train IF Freq.	53.84	47.76
	Evaluate IF Freq.	52.43	47.80
	Train IF TF-IDF	53.66	47.80
	Evaluate IF TF-IDF	51.65	47.80
	Train IF Hashing	53.28	48.08
	Evaluate IF Hashing	53.38	47.90
	Train OC-SVM Freq.	54.13	47.8
	Evaluate OC-SVM Freq.	nan	nan
	Train OC-SVM TF-IDF	47.03	47.80
	Evaluate OC-SVM TF-IDF	nan	nan
	Train OC-SVM Hashing	45.03	47.90
	Evaluate OC SVM Hashing	nan	nan
	Train LOF Freq.	48.6	47.8
	Evaluate LOF Freq.	53.68	47.8
	Train LOF TF-IDF	72.03	47.8
	Evaluate LOF TF-IDF	63.73	47.8
	Train LOF Hashing	55.43	47.93
	Evaluate LOF Hashing	54.96	47.7
Classif.	Train Logistic Regression Freq.	47.03	51.40
	Evaluate Logistic Regression Freq.	nan	nan
	Train Logistic Regression TF-IDF	24.53	51.40
	Evaluate Logistic Regression TF-IDF	nan	nan
	Train Logistic Regression Hashing	33.99	51.41
	Evaluate Logistic Regression Hashing	17.03	51.40
	Train Decision Tree Freq.	nan	nan
	Evaluate Decision Tree Freq.	nan	nan
	Train Decision Tree TF-IDF	63.73	51.40
	Evaluate Decision Tree TF-IDF	nan	nan
	Train Decision Tree Hashing	15.69	51.40
	Evaluate Decision Tree Hashing	nan	nan
	Train Support Vector Machine Freq.	nan	nan
	Evaluate Support Vector Machine Freq.	57.03	51.40
	Train Support Vector Machine TF-IDF	57.03	51.40
	Evaluate Support Vector Machine TF-IDF	nan	nan
	Train Support Vector Machine Hashing	25.03	51.40
	Evaluate Support Vector Machine Hashing	nan	nan
	Train Random Forest Freq.	26.13	51.40
	Evaluate Random Forest Freq.	28.88	51.40
	Train Random Forest TF-IDF	26.89	51.40
	Evaluate Random Forest TF-IDF	29.36	51.40
	Train Random Forest Hashing	27.21	51.40
	Evaluate Random Forest Hashing	27.53	51.40

6.4.1. Time assessment of ML/DL per sample

It illustrates the time-related performance metrics per sample for each monitoring script (RES, KERN, and SYS). It considers the time required by the framework to:

1. Preprocess a single data sample.
2. Evaluate the sample with the best anomaly detection algorithm.
3. Evaluate the sample with the best classification algorithm.

Table 10 summarizes the previous results. The preprocessing column refers to preprocessing the data for anomaly detection. The preprocessing for classification is omitted, as it is almost the same procedure and requires around the same time. Furthermore, the preprocessing for the SYS monitor is the time needed to clean the data, create the corpus, and apply the appropriate vectorizer.

In conclusion, the resource usage experiment demonstrated that training and evaluation are possible on the Raspberry Pi for monitoring data KERN and RES. However, training ML/DL algorithms with extracted system-call data (SYS Monitor) on the sensor is not recommended. As only storage capacity is limited and a large amount of memory is required, the training/evaluation should only occur on a high-powered machine. Furthermore, online evaluation is a viable method for live anomaly detection and classification because it does not require a significant amount of time to preprocess the data and evaluate the ML/DL algorithms.

6.5. Comparison with related work

This experiment compares the detection performance and resource utilization of the proposed framework with an existing rule-based solution that can be found in Huertas Celdrán et al. (2022). The objective of this comparison is to identify the strength and weaknesses of each solution.

At this point, it is essential to mention that Ransomware-PoC and DarkRadiation affecting a Raspberry Pi 3 acting as an ElectroSense sensor are considered in both works. Furthermore, since the RES monitor obtains the best detection performance in the paper at hand, and the rule-based approach previously mentioned and described in Section 2 uses the same monitor, the comparison is focused on that dimension. In this context, Huertas Celdrán et al. (2022) proposed the three rules detailed in Table 11 and described below. The rules were created by following the next steps. First, the maximum and minimum values of each metric were selected as thresholds. Then, the administrator decided what metrics and behavioral categories were incorporated into each rule. Then, the importance of each metric and category was defined, as well as the thresholds of each category and the rule. Finally, during evaluation time, if the thresholds of a given metric were fulfilled, the metric weight was added to the category weight. If the sum of all metrics weights of one category reached the category threshold, the category weight was added to the rule weight. Finally, if the rule weight reaches the rule threshold, the rule is executed.

- *Abnormal Rule*. It is created by only looking at normal behavior and its goal is to detect anomalies or zero-day attacks.
- *Ransomware-PoC Rule*: It is created looking at normal and Ransomware-PoC behaviors. Therefore, it classifies the previous two behaviors, and each behavioral category has a different weight.
- *DarkRadiation Rule*: It is created looking at normal and DarkRadiation behaviors. Different weights are selected for metrics within the CPU category.

Table 10
Evaluation Time for a Single Data Sample.

Monitor	Preprocessing (s)	Best Anomaly Algorithm	Evaluation Time (s)	Best Classification Algorithm	Evaluation Time (s)
RES	0.015	LOF	0.240	Decision Tree	0.0003
KERN	0.011	LOF	0.009	Decision Tree	0.0002
SYS Hashing	0.531	IF	0.032	Decision Tree	0.0001
SYS Frequency	0.524	Autoencoder with IQR	0.098	Decision Tree	0.0002
SYS Tf-idf	0.519	Autoencoder with IQR	0.092	Decision Tree	0.0003

Table 11
Rules Metrics and Weights.

Category	CPU	I/O	Memory	Netw.	Others
Abnormal Rule					
Category weight	500	500	500	500	500
Metrics weight	16.0	18.0	35.0	40.0	45.85
N of metrics	59	28	9	11	6
Ransomware-PoC Rule					
Category weight	250	250	100	0	100
Metrics weight	35	17.5	50	-	100
N of metrics	4	15	2	0	1
DarkRadiation Rule					
Category weight	250	250	100	0	100
Metrics weight	20-50	13.71	50	-	33.34
N of metrics	7	17	2	0	3

Table 12
Rule-base System Detection Performance.

Behaviors	Rules		
	Abnormal	Ransomware-PoC	DarkRadiation
Normal	100% TNR	100% TNR	100% TNR
Abnormal	100% TPR	100% TNR	96.49% TNR
Ransomware-PoC	94.91% TPR	93.22% TPR	98.30% TPR
DarkRadiation	89.80% TPR	100% TNR	55.10% TPR

Table 12 shows that the Abnormal rule almost perfectly identified the four different behaviors (normal, abnormal, Ransomware-PoC, and DarkRadiation). In addition, the rule focused on classifying Ransomware-PoC also provided excellent results. In the case of the DarkRadiation rule, normal, abnormal, and Ransomware-PoC were well detected, but the behavior of DarkRadiation was only detected with 55.10% TPR. In terms of resource consumption, the previous rules consumed about 10% of CPU and 2MB of RAM. Finally, the detection time was about 10 seconds.

Comparing the rule-based approach to the ML/DL-based proposed in this work, all ML/DL algorithms from the experiments outperformed the rule-based approach (see 5). Regarding classification, a similar observation can be made when looking at the F1-score values of the Decision Tree in Table 6. As noted above, the ML/DL results might have been impacted by the shorter duration of training data captured. Furthermore, creating the previous three policies requires a significant investment of time to identify the various levels between normal and ransomware infection behaviors. However, a key advantage of the rule-based approach is that it requires fewer CPU and RAM than ML/DL algorithms.

From the perspective of resource usage monitoring and ML/DL-based ransomware detection, previous works (Huertas Celdrán et al., 2023a; Sánchez Sánchez et al., 2023) have claimed perfect performance detecting Ransomware-PoC in Linux devices, as can be seen in the comparison of Table 13. However, as Table 5 shows, this ransomware is easy to detect compared to more sophisticated samples such as RAASNet, which has not been evaluated in previous works. Besides, these works only consider resource usage as the monitoring approach, while the paper at hand has explored and compared the usage of diverse resources (RES and KERN monitors) and the monitoring of the device system calls.

Table 13
Overview of Ransomware Detection Techniques.

Work (year)	Approach	ML/DL Algorithm	Ransomware Samples	Accuracy
Sánchez Sánchez et al. (2023)	Resources Usage	(ML/DL) AD	Ransomware-PoC	100%
Huertas Celdrán et al. (2023a)	Resources Usage	(ML/DL) Classification and AD	Ransomware-PoC	100%
This work (2023)	Resources Usage, Syscalls	(ML/DL) Classification and AD	Ransomware-PoC, DarkRadiation, RAASNet	90-100%

7. Discussion

This section discusses the main approach, contributions, and limitations of the proposed framework. As previously mentioned, the proposed framework combines three behavioral dimensions (usage of resources, system calls, and kernel events) with ML/DL models to detect zero-day ransomware samples and classify them. A set of experiments has compared the detection performance, time, and resource consumption with a well-known approach based on rules. At this point, it is important to mention that the framework has been designed to be deployed on different devices with heterogeneous hardware and software. However, the detection performance of the framework depends on several aspects. In this context, a crucial aspect to consider is the stability of the normal behaviors of the device. If a device exhibits a consistent set of normal behaviors, then the proposed framework is effective (as it has been demonstrated in the previous experiments with an ElectroSense sensor). However, as the level of behavioral freedom increases, the problem becomes much more complex due to the difficulty of distinguishing between normal and abnormal behaviors. In this sense, aspects such as network variations could affect the framework performance detection.

Another interesting discussion topic is the alternatives to ML/DL in order to detect attacks. In this context, the literature has identified rules, statistical, knowledge, and time series-based solutions (Sánchez Sánchez et al., 2021). Table 14 compares the advantages and limitations of each detection mechanism. More in detail, rule-based solutions are commonly used for ransomware classification and anomaly detection by creating behavioral profiles in a straightforward fashion. This approach is ideal for devices with well-defined behavior and a limited range of actions. It defines a set of rules that dictate the desired behavior of the device, essentially creating a behavioral fingerprint. These rules can be static (predetermined actions) or dynamic (based on the device historical actions). Any deviation from the rules can be considered an anomaly provoked by ransomware attacks. In addition, rules can also model different ransomware samples to classify them. The main advantages of this approach are its simplicity and speed, but it requires prior knowledge of the device behavior and is not suitable for complex or changing scenarios. Statistical-based approaches are another alternative and involve basic statistical data processing techniques to extract meaningful information from behavioral data samples. This approach is commonly employed in data preprocessing and ransomware detection. The key advantages of this approach are its simplicity and the fact that it does not necessitate large datasets. However, it may struggle with multi-

Table 14
Detection Approaches Comparison.

Approach	Simplicity	Expert knowledge	Low consumption	Large datasets	Large training time	Explainability
ML/DL-based	no	no	no	mainly DL	mainly DL	partial
Rule-based	yes	yes	yes	no	no	yes
Statistical	yes	yes	yes	no	no	no
Knowledge-based	partial	no	no	no	no	yes
Time series	no	no	no	yes	yes	no

dimensional data, and making consistent evaluation decisions requires domain expertise. Statistical functions such as average, standard deviation, quartiles, maximum, and minimum are often used to infer features during preprocessing. Knowledge-based solutions aim to extract knowledge from received data and build a reasoning system capable of inferring new knowledge. Typically, this knowledge is constructed using ontologies, and decision-making relies on if-then derivation rules. The main benefits of this approach are the explainability of the inferred solutions and the ability to solve problems with incomplete data. However, this approach is time-consuming and lacks scalability, as the system can become overly complex when large amounts of data are used. Knowledge-based approaches are primarily utilized for ransomware detection, with finite-state machines being the main techniques employed. Time series analysis uses sequential data measurements where each value is related to the previous and subsequent ones. This approach encompasses a wide range of algorithms and models, including those based on ML/DL or statistical methods. Time series analysis is employed for both anomaly detection and ransomware classification, either directly in model generation or as a step in data preprocessing. The main advantage of this approach is its superior performance compared to single-value processing methods. However, it requires a substantial amount of data to identify temporal patterns, and the processing time can be significant. Time series analysis methods can be classified into two types: frequency-based methods, which analyze data as a signal with a specific frequency, and time-based methods, which examine data evolution over time. Finally, the approach considered in this work focuses on ML/DL, which has gained significant traction in various research fields, driven by increased processing power and available data. ML/DL approaches offer several advantages, including the ability to detect complex data patterns, handle multi-dimensional and multi-variate data, and adapt to dynamic and heterogeneous scenarios using extensive data. However, a major drawback is the lack of explainability in model decisions due to the black-box nature of some of the models. Additionally, DL algorithms, in particular, require substantial amounts of training data and consume significant time and resources for training. Most algorithms also require parameter tuning, which involves repeating the training process multiple times.

Despite the excellent performance of ML/DL models considered by the proposed framework, ethical aspects are key due to the potential impacts and implications for society. As an example, the proposed ransomware detection framework involves analyzing sensitive data, such as the usage of resources, system calls, and kernel events. In this sense, ethical considerations should ensure that privacy rights are respected by minimizing the risk of unauthorized access or misuse. Trustworthy AI is another important aspect that has not been considered by the proposed solution (Huertas Celdran et al., 2023). As humans, we should be able to know if a model or prediction is trustworthy or not. In this context, trusted AI can greatly assist in the detection of ransomware in particular and malware in general by ensuring the reliability and integrity of the detection process. One of the primary challenges in malware detection is the ability to differentiate between genuine and malicious behaviors accurately. Trustworthy AI systems employ a range of techniques to enhance the accuracy and effectiveness of malware detection algorithms. These systems prioritize transparency, interpretability, robustness, and fairness, enabling security analysts to understand the decision-making process and identify potential biases or false positives. Additionally, trustworthy AI systems integrate robust security measures to protect against adversarial attacks, ensuring that the detection models remain resilient to malicious attempts to evade detection. By instilling trust in the detection process, trustworthy AI empowers security professionals with reliable tools to combat ever-evolving malware threats and safeguard critical systems and data.

Aligned with data privacy and its ethical implications, Federated Learning (FL) is another powerful paradigm (Rey et al., 2022), not considered by the proposed framework, that trains models collaboratively between different devices while protecting the privacy of data. In this context, FL can play a crucial role in the detection of ransomware by leveraging its decentralized and privacy-preserving approach. In traditional malware detection systems, sensitive data is often centralized, posing potential security and privacy risks. However, with FL, the model training process takes place directly on users' devices, allowing them to contribute their local data without exposing it to a central server. This distributed approach enables collective learning from a set of clients (Raspberry Pis in this work) while maintaining data privacy. By aggregating the knowledge and insights from diverse sources, FL can effectively detect ransomware patterns, identify new threats, and continuously update and improve the detection models. Furthermore, FL allows for real-time updates and adaptability, ensuring a more robust and proactive defense against evolving malware threats in a privacy-preserving manner. These aspects have not been explored in the document at hand and would be powerful extensions to improve detection capabilities. Finally, the evaluation and comparison of performance (in terms of detection rate, time, and resource consumption) with more devices and families of malware such as botnets, backdoors, cryptojackers, or rootkits is also something missing in this work and worthy of exploring in the future.

8. Conclusion

This work designed and developed a distributed anomaly detection and classification framework for resource-constrained devices. Different behavioral data sources focused on the usage of resources, kernel events, and system calls were considered to train ML/DL algorithms. The detection performance of ML/DL was analyzed and compared while detecting and classifying three families of ransomware (RansomwarePoC, DarkRadiation, and RAASNet) affecting Raspberry Pis acting as spectrum sensors. These results were then compared to an existing rule-based solution.

In conclusion, this work has demonstrated that ML and DL can assist in ransomware detection and classification while consuming resources in an acceptable manner. In this sense, when trained with RES and KERN monitoring data, anomaly detection algorithms yielded the best results (overall ransomware samples). The LOF algorithm was identified to be the most promising regarding anomaly detection when using RES and KERN monitoring data. However, compared to rule-based systems, it is also more CPU-demanding during training. Regarding classification, SYS, KERN, and RES monitoring data have been demonstrated as suitable for ransomware detection based on ML classification. The decision tree classifier yielded the best overall classification results (performance and training time) for RES, KERN, and SYS. The implemented ML/DL algorithms outperformed the rule-based approach regarding anomaly detection and classification. Regarding resource consumption, system call monitoring produces a large volume of data compared to

RES and KERN monitors. In addition, processing of this data is very time and memory intensive. Therefore, the RES monitor is the most suitable data source to detect anomalies and classify ransomware while consuming resources in a reduced manner.

There are several areas where further investigation could prove beneficial in improving ransomware detection and classification. These include assessing the impact of network activity on the monitoring results and isolating those factors which negatively influence the process. Another future area of research is to identify the optimal balance between training data size/monitoring duration and ML/DL ransomware detection accuracy.

CRediT authorship contribution statement

Alberto Huertas Celdrán: Conceptualization, Methodology, Writing – Original draft preparation. **Pedro Miguel Sánchez Sánchez:** Data curation, Writing – Original draft preparation. **Jan von der Assen:** Visualization, Investigation. **Dennis Shushack:** Experiments, Validation. **Angel Luis Perales Gomez:** Software, Validation. **Gerome Bovet:** Data curation, Validation. **Gregorio Martínez Pérez:** Supervision, Writing – Reviewing and Editing. **Burkhard Stiller:** Supervision, Reviewing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work has been partially supported by *(a)* the Swiss Federal Office for Defense Procurement (armasuisse) with the DEFENDIS and CyberForce (CYD-C-2020003) projects and *(b)* the University of Zürich UZH.

References

Ahmed, Y.A., Huda, S., Al-rimy, B.A.S., Alharbi, N., Saeed, F., Ghaleb, F.A., Ali, I.M., 2022. A weighted minimum redundancy maximum relevance technique for ransomware early detection in industrial iot. Sustainability 14, 1231.

Alam, M., Sinha, S., Bhattacharya, S., Dutta, S., Mukhopadhyay, D., Chattopadhyay, A., 2020. Rapper: ransomware prevention via performance counters. Preprint. arXiv: 2004.01712.

Almashhadani, A.O., Kaiiali, M., Sezer, S., O'Kane, P., 2019. A multi-classifier network-based crypto ransomware detection system: a case study of locky ransomware. IEEE Access 7, 47053–47067.

Almomani, I., Qaddoura, R., Habib, M., Alsoghyer, S., Al Khayer, A., Aljarah, I., Faris, H., 2021. Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data. IEEE Access 9, 57674–57691.

Almousa, M., Basavaraju, S., Anwar, M., 2021. Api-based ransomware detection using machine learning-based threat detection models. In: 2021 18th International Conference on Privacy, Security and Trust (PST), IEEE, pp. 1–7.

Azmoodeh, A., Dehghantanha, A., Conti, M., Choo, K.K.R., 2018. Detecting crypto-ransomware in iot networks based on energy consumption footprint. J. Ambient Intell. Humaniz. Comput. 9, 1141–1152.

Bae, S.I., Lee, G.B., Im, E.G., 2020. Ransomware detection using machine learning algorithms. Concurr. Comput., Pract. Exp. 32, e5422.

Berrueta, E., Morato, D., Magaña, E., Izal, M., 2022. Crypto-ransomware detection using machine learning models in file-sharing network scenarios with encrypted traffic. Expert Syst. Appl. 209, 118299.

ElectroSense, 2023. Collaborative Spectrum Monitoring. https://electrosense.org/. (Accessed 24 March 2023).

Faghihi, F., Zulkernine, M., 2021. Ransomcare: data-centric detection and mitigation against smartphone crypto-ransomware. Comput. Netw. 191, 108011.

Gazet, A., 2010. Comparative analysis of various ransomware virii. J. Comput. Virol. 6, 77–90.

HashingVectorizer, 2023. HashingVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html. (Accessed 24 March 2023).

Huertas Celdran, A., Kreischer, J., Demirci, M., Leupp, J., Sanchez, P.M., Franco, M.F., Bovet, G., Perez, M.G., Stiller, B., 2023. A framework quantifying trustworthiness of supervised machine and deep learning models. In: SafeAI2023: The AAAI's Workshop on Artificial Intelligence Safety, pp. 2938–2948.

Huertas Celdrán, A., Sánchez Sánchez, P.M., Azorín Castillo, M., Bovet, G., Martínez Pérez, G., Stiller, B., 2023a. Intelligent and behavioral-based detection of malware in iot spectrum sensors. Int. J. Inf. Secur. 22, 541–561.

Huertas Celdrán, A., Sánchez Sánchez, P.M., Bovet, G., Martínez Pérez, G., Stiller, B., 2023b. Cyberspec: behavioral fingerprinting for intelligent attacks detection on crowdsensing spectrum sensors. IEEE Trans. Dependable Secure Comput.

Huertas Celdrán, A., Sánchez Sánchez, P.M., Scheid, E.J., Besken, T., Bovet, G., Martínez Pérez, G., Stiller, B., 2022. Policy-based and behavioral framework to detect ransomware affecting resource-constrained sensors. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, IEEE, pp. 1–7.

Imtiaz, S.I., ur Rehman, S., Javed, A.R., Jalil, Z., Liu, X., Alnumay, W.S., 2021. Deepamd: detection and identification of Android malware using high-efficient deep artificial neural network. Future Gener. Comput. Syst. 115, 844–856.

Lakshmanan, R., 2021. Wormable DarkRadiation Ransomware Targets Linux and Docker Instances. https://thehackernews.com/2021/06/wormable-darkradiation-ransomware.html. (Accessed 24 March 2023).

Meland, P.H., Bayoumy, Y.F.F., Sindre, G., 2020. The ransomware-as-a-service economy within the darknet. Comput. Secur. 92, 101762.

Patro, S., Sahu, K.K., 2015. Normalization: a preprocessing stage. Preprint. arXiv:1503.06462.

Poudyal, S., Dasgupta, D., 2021. Analysis of crypto-ransomware using ml-based multi-level profiling. IEEE Access 9, 122532–122547.

RAASNet, 2021. RAASNet. https://github.com/Toyhack/RAASNet. (Accessed 24 March 2023).

Rajendran, S., Calvo-Palomino, R., Fuchs, M., Van den Bergh, B., Cordobes, H., Giustiniano, D., Pollin, S., Lenders, V., 2018. Electrosense: open and big spectrum data. IEEE Commun. Mag. 56, 210–217. https://doi.org/10.1109/MCOM.2017.1700200.

Ramesh, G., Menen, A., 2020. Automated dynamic approach for detecting ransomware using finite-state machine. Decis. Support Syst. 138, 113400.

Ransomware-PoC, 2023. Ransomware-PoC. https://github.com/jimmy-ly00/Ransomware-PoC. (Accessed 24 March 2023).

Ransomware Spotlight, 2023a. Hive. https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-hive. (Accessed 24 March 2023).

Ransomware Spotlight, 2023b. LockBit. https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-lockbit. (Accessed 24 March 2023).

Rey, V., Sánchez Sánchez, P.M., Huertas Celdrán, A., Bovet, G., 2022. Federated learning for malware detection in iot devices. Comput. Netw. 204, 108693.

Rhode, M., Burnap, P., Jones, K., 2018. Early-stage malware prediction using recurrent neural networks. Comput. Secur. 77, 578–594.

Sánchez Sánchez, P.M., Huertas Celdrán, A., Bovet, G., Martínez Pérez, G., Stiller, B., 2023. Specforce: a framework to secure iot spectrum sensors in the Internet of battlefield things. IEEE Commun. Mag. 61, 174–180. https://doi.org/10.1109/MCOM.001.2200349.

Sánchez Sánchez, P.M., Huertas Celdrán, A., Schenk, T., Iten, A.L.B., Bovet, G., Martínez Pérez, G., Stiller, B., 2022. Studying the robustness of anti-adversarial federated learning models detecting cyberattacks in iot spectrum sensors. IEEE Trans. Dependable Secure Comput.

Sánchez Sánchez, P.M., Jorquera Valero, J.M., Huertas Celdrán, A., Bovet, G., Gil Pérez, M., Martínez Pérez, G., 2021. A survey on device behavior fingerprinting: data sources, techniques, application scenarios, and datasets. IEEE Commun. Surv. Tutor. 23, 1048–1077.

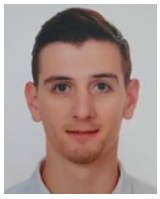
Sharma, S., Krishna, C.R., Kumar, R., 2021. Ransomdroid: forensic analysis and detection of Android ransomware using unsupervised machine learning technique. Forensic Sci. Int., Digit. Investig. 37, 301168.

Shushack, D., 2023a. Data analysis application flask. https://github.com/dennisshushack/BA_Thesis_Flask. (Accessed 24 March 2023).

Shushack, D., 2023b. Monitor controller repository. https://github.com/dennisshushack/BA_Thesis_PI. (Accessed 24 March 2023).

Thierer, A., Castillo, A., 2015. Projecting the Growth and Economic Impact of the Internet of Things. George Mason University, Mercatus Center. June 15.

Alberto Huertas Celdrán is a senior researcher at the Communication Systems Group CSG, Department of Informatics Ifl, University of Zurich UZH. He received the MSc and PhD degrees in Computer Science from the University of Murcia, Spain. His scientific interests include cybersecurity, machine and deep learning, continuous authentication, and computer networks. Contact him at huertas@ifi.uzh.ch.



Pedro M. Sánchez Sánchez is pursuing his PhD in computer science at the University of Murcia. He received the MSc degree in Computer Science from the University of Murcia, Spain. His research interests focus on continuous authentication, networks, 5G, cybersecurity, and machine learning and deep learning. Contact him at pedromigel.sanchez@um.es.

Jan von der Assen received his MSc degree in Informatics from the University of Zurich, Switzerland. Currently, he is pursuing his Doctoral Degree under the supervision of Prof. Dr. Burkhard Stiller at the Communication Systems Group, University of Zurich. His research interest lies at the intersection between risk management and the mitigation of cyber threats. Contact him at vonderasssen@ifi.uzh.ch.

Dennis Shushack received his Bachelor degree in Computer Science from the University of Zurich, Switzerland. His research interests focus on cybersecurity, IoT, and machine learning and deep learning. Contact him at dennis.shushack@uzh.ch.

Ángel Luis Perales Gómez is a postdoctoral researcher in the Department of Computer Engineering at the University of Murcia. He received his M.S. and Ph.D. degrees in Computer Science from the University of Murcia, Spain. His research interests include deep learning, machine learning, interpretability, robustness, and cybersecurity of industrial control systems. Contact him at angelluis.perales@um.es.

Gérôme Bovet is the head of data science for the Swiss Department of Defense. He received his PhD in networks and computer systems from Telecom ParisTech, France. His work focuses on Machine and Deep Learning, with an emphasis on anomaly detection, adversarial and collaborative learning in IoT sensors. Contact him at gerome.bovet@armasuisse.ch.

Gregorio Martínez Pérez is Full Professor in the Department of Information and Communications Engineering of the University of Murcia, Spain. His scientific activity is mainly devoted to cybersecurity and networking, where he has published 160+ papers. Contact him at gregorio@um.es.

Burkhard Stiller chairs the Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH, as a Full Professor. He received the MSc and PhD degrees from the University of Karlsruhe, Germany. His main research interests include fully decentralized systems, network and service management, IoT, and telecommunication economics. Contact him at stiller@ifi.uzh.ch.