CyberSpec: Behavioral Fingerprinting for Intelligent Attacks Detection on Crowdsensing Spectrum Sensors

Alberto Huertas Celdrán, Pedro Miguel Sánchez Sánchez, Gérôme Bovet, Gregorio Martínez Pérez, and Burkhard Stiller

Abstract—Integrated sensing and communication is a novel paradigm using crowdsensing spectrum sensors to help with the management of spectrum scarcity. However, well-known vulnerabilities of resource-constrained spectrum sensors and the possibility of being manipulated by users with physical access complicate their protection against spectrum sensing data falsification (SSDF) attacks. Most recent literature suggests using behavioral fingerprinting and Machine/Deep Learning (ML/DL) for improving similar cybersecurity issues. Nevertheless, the applicability of these techniques in resource-constrained devices, the impact of attacks affecting spectrum data integrity, and the performance and scalability of models suitable for heterogeneous sensors types are still open challenges. To improve limitations, this work presents seven SSDF attacks affecting spectrum sensors and introduces CyberSpec, an ML/DL-oriented framework using device behavioral fingerprinting to detect anomalies produced by SSDF attacks. CyberSpec has been implemented and validated in ElectroSense, a real crowdsensing RF monitoring platform where several configurations of the proposed SSDF attacks have been executed in different sensors. A pool of experiments with different unsupervised ML/DL-based models has demonstrated the suitability of CyberSpec detecting the previous attacks within an acceptable timeframe.

Index Terms—Anomaly detection, behavioral fingerprinting, crowdsensing spectrum sensors, ML/DL, SSDF attacks.

I. INTRODUCTION

The growth of wireless IoT devices is increasing the demand for radio frequency (RF) spectrum [1]. Consequently, interferences occurring in overcrowded RF bands degrade service quality due to the necessity of re-transmissions and the reduction of effective data rates. This situation promotes crowdsensing RF spectrum sensors as one of the most attractive and cheapest solutions for spectrum optimization. In contrast to static, expensive, and complex traditional RF platforms, those based on crowdsensing [2] consider low-cost spectrum sensors (with limitations in terms of CPU, RAM, and disk) that can be quickly deployed where needed to scan the spectrum. These platforms are also important when (i) intercepting illegal communications occurring in licensed bands; (ii) detecting cyberattacks, jamming or interfering critical transmissions; or (iii) classifying transmission standards and technologies.

All the previous tasks highly rely on sensed RF spectrum data, a critical asset that must be protected to ensure trusted services. This protection is challenging since Spectrum Sensing Data Falsification (SSDF) attacks are among the most well-known and impactful attack families affecting spectrum sensor [3]. Moreover, it becomes even more challenging in the crowdsensing paradigm, where sensors (i) present well-known and exploitable vulnerabilities, (ii) face hardware and software limitations to deploy cybersecurity mechanisms, and (iii) can be easily manipulated by users with physical access to them [4]. Traditionally, SSDF attacks have been detected by analyzing spectrum data and reaching a consensus between different sensors closely located [5]. However, these detection approaches present several limitations, such as (i) the necessity of redundant and trustworthy sensors, something not feasible in zero-trust scenarios [6], and (ii) the difficulty of detecting attacks sending outdated spectrum data or adding small perturbations. Therefore, novel approaches are needed to improve upon the drawbacks.

In this context, behavioral fingerprinting turned out to be one of the most promising alternatives to detect cyberattacks [7]. In particular, heterogeneous data sources, like hardware events, system calls, or resource usage, can be monitored to create device behavioral fingerprints. These fingerprints are evaluated to detect anomalies by different techniques, being Machine and Deep Learning (ML and DL) the most promising approaches nowadays. Therefore, fingerprinting and ML/DL could be a promising combination to detect SSDF attacks. However, this approach has not been studied in the literature, and the following challenges are open: (ch1) there is no work defining comprehensively how to manipulate spectrum data; (ch2) there is no solution measuring the performance and suitability of behavioral
fingerprinting in spectrum sensors with limited CPU, RAM, and disk; (ch3) data sources and events precisely modeling normal and under attack behaviors of spectrum sensors have not been studied; and (ch4) there is no analysis of the effectiveness, efficiency, and scalability of fingerprinting and ML/DL solutions applied to SSDF attacks.

To address these challenges, the main contributions of this work include:

- The definition and deployment of seven novel SSDF attacks (Repeat, Mimic, Confusion, Noise, Spoof, Freeze, and Delay) affecting spectrum (addressing ch1). These attacks are categorized into three main families focused on: (i) simulating non-existing communications, (ii) hiding cyberattacks or illegal transmissions, and (iii) performing both actions in parallel. All attacks are implemented by modifying the legitimate code of RF spectrum sensors.

- The design and implementation of CyberSpec, an ML/DL-oriented framework that uses behavioral fingerprinting to detect anomalies in resource-constrained spectrum sensors affected by SSDF attacks (addressing ch2). Events belonging to the device virtual memory, file system, CPU, network, scheduler, device drivers, and random number generation are monitored periodically to create behavior fingerprints. Then, anomaly detection algorithms are employed to detect deviations in the sensor behavior. The framework code is publicly available in [8].

- The deployment of CyberSpec on a real IoT-based crowd-sensing RF monitoring platform called ElectroSense [2]. Nine Raspberry Pis 3 and 4, acting as RF spectrum sensors, were infected with different configurations of the seven SSDF attacks to later measure the CyberSpec detection performance as well as the time and resource consumption (addressing ch3). Normal and under-attack behaviors of each sensor were monitored to create a dataset. The dataset is publicly available in [8].

- The creation of a pool of experiments to detect normal and under attack behaviors using ML/DL models per (i) individual sensors, (ii) families of sensors excluding a different number of devices from training, and (iii) combinations of families (addressing ch4). The obtained results show that five of the seven SSDF attacks are almost perfectly detected in the three experiments. Device-type and global ML/DL models provide similar detection performance to individual models. Finally, when the 15%, 33%, and 50% of devices are excluded from training, device-type models perform acceptably for most excluded devices (80-70% TPR and 100% TNR for five attacks).

The remainder of this article is organized as follows. Section II reviews attacks affecting spectrum data and behavior fingerprinting to detect different cyberattacks. While Section III introduces the basics of spectrum sensing, Section IV presents seven SSDF attacks affecting sensors. Section V introduces CyberSpec, an ML/DL-oriented framework detecting anomalies. Section VI outlines the deployment and performance of CyberSpec in ElectroSense. Finally, Section VII draws conclusions and next steps.

II. RELATED WORK

SSDF attacks affecting spectrum sensors, detection approaches to detect SSDF attacks, and device fingerprinting to detect cyberattacks are analyzed below. Table I compares the most representative aspects and performance of related work.

A. SSDF Attacks and Detection Mechanisms

SSDF attacks [9] send fake spectrum data to backend platforms to disrupt its services. [10] considered heterogeneous SSDF attacks to evaluate the effectiveness of the proposed consensus detection mechanisms. However, this work does not explain how SSDF attacks manipulate spectrum data. Additionally, the proposed detection mechanism analyzes spectrum data, but several sensors in the same area are needed to reach a consensus. In contrast, CyberSpec does not require additional sensors to detect SSDF attacks. The authors of [11] used a probabilistic SSDF attack model executed by malicious sensors. Furthermore, they proposed a detection mechanism based on historical spectrum data and M-ary quantized data. In [12], a Bayesian-inference-based sliding window trust model was proposed to detect various probabilistic SSDF attacks. However, the main limitation of these two works is that SSDF attacks repeating the same spectrum data or adding minor variations are not detectable. This issue does not affect CyberSpec since it analyzes sensors behaviors instead of variations in spectrum data. Finally, the authors of [5] reviewed existing detection mechanisms for SSDF attacks, such as reputation-based, neighborhood distance, abnormality detection, AI-based, genetic algorithms, and outlier detection. However, all these approaches analyze spectrum data, and some of them require multiple sensors and trusted scenarios. In contrast, CyberSpec proposes a novel approach considering the behavior of spectrum sensors where redundancy and trust are not needed.

In conclusion, there is a lack of work detailing how existing SSDF attacks manipulate the spectrum data, which is critical to proposing effective detection mechanisms. Regarding detection mechanisms, works focused on analyzing spectrum data are vulnerable to attacks sending outdated data or adding minor perturbations. Furthermore, most works rely on trust and consensus between sensors, and it cannot be assumed in some real scenarios.

B. Cyberattack Detection Through Behavioral Fingerprinting

The number of existing work applying behavioral fingerprinting in IoT devices from the host perspective is reduced, but one of them is HADES-IoT [13]. HADES-IoT is a host-based anomaly detection system for different Linux-based IoT devices that creates white lists of benign system calls. 100% accuracy is reported when evaluating the system with different IoT malware affecting seven different IoT devices. Also, in the field of fingerprinting and IoT, DAIMD [14] is a hybrid scheme that monitors memory, network, virtual file system, process, and system calls of devices to detect both well-known and zero-day attacks. A
convolution neural network (CNN) model is trained to binary classify samples with 99.8% accuracy.

System calls, resource usage, or Hardware Performance Counters (HPC) have been widely used to create behavioral fingerprints and detect cyberattacks affecting different devices. From the system calls perspective, VizMal [15] detects Android malware by creating color boxes that represent software execution time windows. The color of each box refers to the maliciousness level, and the size is the number of system calls executed during the time window. A Long Short-Term Memory (LSTM) Neural Network trained with samples labeled as malware and non-malware provided 9.8% FPR. VMGuard [16] is another ML-oriented security system that uses system calls to detect malware affecting Virtual Machines (VM) in cloud scenarios. VMGuard monitors the system calls of VMs to create a ‘Bag of n-grams (BonG)’ integrated with the Term Frequency-Inverse Document Frequency (TF-IDF) method. During the evaluation process, Random Forest classified different attacks with 93-99% accuracy.

Resource usage is another well-known and widely used data source to create behavioral fingerprints. In this context, the solution presented in [17] detects Denial-of-Service (DoS) attacks in cloud scenarios by considering CPU usage. CPU usage statistics of micro-services running on cloud architectures were monitored to later detect DoS attacks using auto-regressive statistical models. RADS [18] is another system that monitors resource usage to detect DDoS attacks in cloud data centers. RADS uses one class classification algorithm and time series analysis to obtain 90-95% accuracy when detecting anomalies produced by DDoS and cryptomining attacks. Finally, HPCs were considered in [19], where authors proposed an ML/DL-oriented solution that detects deviations from normal behaviors produced by malware affecting embedded systems. Experiments reported an accuracy higher than 95% with Hidden Markov Models (HMM) and LSTM Neural Networks.

In conclusion, most fingerprinting solutions are designed for powerful devices, being useless for those with limited CPU, RAM, and disk due to the amount of monitored data sources, consumption of detection techniques, and attacks nature. Despite HADES-IoT and DAIMD work for IoT, they do not evaluate their suitability or performance when detecting SSDF attacks in spectrum sensors (having a particular functionality that affects the device behavior). Finally, none of the analyzed works studied the detection performance of ML/DL models combining behaviors of different types of devices. Due to the previous facts, more work aligned with SSDF attacks affecting RF sensing platforms and ML/DL-based detection frameworks is needed.

### III. Crowdsensing Spectrum Sensors & Cybersecurity Issues

Crowdsensing spectrum sensors are devices with limited CPU, RAM and disk, such as a Raspberry Pi, equipped with Software-defined Radio (SDR) kits. Typically, spectrum sensors implement a process in charge of (i) dividing the spectrum into fixed-size segments, and (ii) scanning cyclically the segments composing the whole RF spectrum, from the lower frequency supported by the sensor to the higher. This RF monitoring process enables the collection of different data types per segment, such as Power Spectrum Data (PSD), indicating the spectrum occupancy of each particular segment. After that, the collected data is sent periodically to a central platform, in charge of processing and analyzing it to provide services in charge of (i) optimizing RF occupancy, (ii) intercepting illegal communications over licensed bands, and (iii) detecting cyberattacks jamming or interfering legitimate communications.

The trustworthiness of the previous services depends on the integrity of PSD collected by IoT spectrum sensors. However, the nature of crowdsensing RF monitoring platforms, relying on resource-constrained spectrum sensors vulnerable to cyberattacks and manipulations of dishonest users, makes spectrum sensors vulnerable to SSDF attacks. In this sense, external attackers could execute brute-forcing password attacks to exploit well-known vulnerabilities of network services such as SSH or Telnet. In addition, internal attackers with physical access to the device could manipulate its code. Both attackers could execute SSDF attacks with the following objectives [20].

- Misbehaving. The attacker breaks the rules established by the network or spectrum sensing platform, for example, illegal hiding transmissions.
Selfish. The attacker keeps network resources for his/her benefit, for example, simulating non-existent transmissions to prevent or block others.

Cheating. The attacker provides fake spectrum data to increase his/her quality of service (QoS).

Malicious. The attacker targets the spectrum data to jam or degrade the QoS of other nodes and the network efficiency.

In such a context, and as explained in Section II, there is a literature gap in terms of how to implement SSDF attacks performing these objectives or behaviors.

IV. SSDF ATTACKS

This section presents seven novel SSDF attacks that execute the four previous behaviors. The proposed SSDF attacks are categorized into three main families according to their PSD impact: Transmission Simulation, Transmission Hiding, and Hybrid. Transmission Simulation attacks modify the PSD of spectrum segments to simulate non-existing wireless transmissions. Transmission Hiding attacks make transmissions invisible for the platform. Therefore, both attack families can be used to implement misbehaving, selfish, cheating, and malicious behaviors. Finally, Hybrid attacks can cover all malicious behaviors by simulating fictitious transmissions and hiding illegal ones in parallel. Fig. 1 shows the attacks belonging to each family.

A. Hybrid SSDF Attacks

Attacks belonging to this family can be used either to simulate transmissions or hide them. The family is composed of the following three attacks: Repeat, Mimic, and Confusion.

Repeat attacks copy in a particular moment of time the PSD of a selected spectrum segment and continuously replicates it in the segments targeted by the attack. More in detail and as can be seen in Fig. 2, first, the attacker selects the RF segment that wants to be replicated (Source_seg) and the spectrum segments whose PSDs are going to be manipulated (SegA). Depending on the PSD values of Source_seg, the attack will hide or simulate transmissions. After that, it creates a file (File) to save PSD values of Source_seg. At this point, the RF spectrum is continuously scanned, as indicated in Section III. During the first scanning of the RF spectrum, the PSD values of Source_seg are stored in File (only once). In the next RF scanning cycles, the attack modifies the PSD values of the SegA with the File content.

Mimic attacks are an evolution of Repeat, being the creation of new PSD copies per RF spectrum the main difference of them. In particular, as seen in Fig. 3, this attack defines two sets of spectrum segments (SegS are the segments whose PSDs are going to be replicated, and SegA the replaced segments) and creates one empty file (File) per RF SegS. After that, the scanning process starts, and for each RF spectrum cycle the attack stores the PSD values of SegS in FileS. Once the files contain the PSD values of SegS, the attack substitutes the PSD of the SegA with the files content. The number of RF segments of SegS and SegA must be equal, and depending on the SegA occupancy the attack simulates or hides transmissions.

Confusion attacks pretend to exchange the occupancy levels of two or more RF segments. The attack can be used to hide a transmission, to simulate a non-existing one, or for both at the same time, creating confusion. Fig. 4 shows how the attack starts defining the sets of spectrum segments that are going to be exchanged (SegX and SegY), and creating two files (FileX and FileY) with PSD values. After that, the monitoring process initiates the sequential RF scanning. When the scanning process senses the occupancy of a segment belonging to the SegX, its PSD values are stored in FileX, and replaced with the content of FileY. Similarly, when the monitor senses an RF segment
belonging to SegY, it saves its PSD in FileY and replaces the segment occupancy with FileX content.

B. Transmission Simulation SSDF Attacks

Attacks of this family are in charge of modifying the RF spectrum occupancy data to simulate fake transmissions. This family is composed of two attacks: Noise and Spoof.

Noise attacks focus on adding random noise to the occupancy level of a set of RF spectrum segments affected by the attack. Fig. 5 shows the life-cycle of this attack, where the attacker starts defining the spectrum segments (SegA) that will be attacked and the intensity of the noise. After that, the RF monitoring process starts. When those segments affected are scanned, the attack generates random numbers and adds them to their PSD values. This process is periodically repeated in each RF scanning cycle for all segments affected.

Spoof attacks are an evolution of Mimic attacks. The main difference is that Spoof adds random noise to the occupancy level of source spectrum segments to replicate a transmission but adding differences to complicate its detection. As seen in Fig. 6, the attacker defines the sets of source and attacked spectrum segments (SegS and SegA), creates a file per spectrum segment belonging to SegS (FileS), and defines the intensity of random noise. After that, the RF monitoring process starts. If the current segment belongs to SegS, one random value per PSD is generated, added to the PSD of the current segment, and stored in FileS. When the segment is within the set of the attacked segments (SegA), the occupancy of the current segment is replaced with the content of FileS. This process is repeated for the whole RF spectrum and across time.

C. Transmission Hiding SSDF Attacks

Freeze and Delay are the attacks belonging to this family and pretend to hide illegal or unauthorized transmissions.

Freeze attacks copy the PSD values of one or more spectrum segments in a given moment and replicate them across the time. This type of attack hides any transmission starting after the screenshot generation and substitution. As seen in Fig. 7, the attacker begins defining the spectrum segments affected by the attack (SegA) and creates an empty file, where the PSD values will be stored. After that, the cyclic RF scanning starts. The first time (first scan) the monitor senses the occupancy of a segment belonging to SegA, its PSD values are stored in FileA. In successive iterations of the scanning process, when the monitor senses a segment belonging to SegA, its PSD values will be replaced with the PSDs previously saved in FileA for that particular segment.

Delay attacks are an evolution of Freeze, and their main goal is to send the RF monitoring platform obsolete or non-updated PSDs of affected spectrum segments. The main difference between Delay and Freeze is that Freeze always sends the same occupancy level for each affected segment, while Delay keeps a sliding time window to send different outdated PSD. In this sense, Fig. 8 shows that the attack starts defining the set of affected spectrum segments (SegA), the time of delay to provide...
the segments occupancy (DelA), and creates an empty file per affected segment (FileA). After that, the RF spectrum scanning starts. If the sensed spectrum segments belong to the set of SegA, its PSD values are stored in FileA. Later, it is checked if the time window of delay is over. If so, the segment PSD values are replaced with the oldest content of FileA, and the content is deleted from the file. In contrast, when DelA is not over, the attack stores the segment occupancy values in FileA, but the segment occupancy is not altered. It happens when the attack starts and until the selected delay is reached.

V. CYBERSPEC FRAMEWORK

The CyberSpec framework considers device behavioral fingerprinting to detect anomalies produced by SSDF attacks affecting resource-constrained spectrum monitors. For that, the following two modules compose the flexible framework:

- **Behavior Fingerprinting.** Monitor the internal behavior of the resource-constrained spectrum sensor from different data sources to create behavioral fingerprints.

- **Cyberattacks Detection.** Train unsupervised ML/DL anomaly detectors that consider device behavior fingerprints. It evaluates the trained models with the real-time behavior of sensors to detect anomalies produced by SSDF attacks.

Fig. 9 shows the two modules of the proposed framework as well as their components, which are explained in this section. CyberSpec can be deployed in a hybrid way, where the sensor hosts the Behavior Fingerprinting module and the Cyberattacks Detection module can be executed on sensors and servers.

A. Behavioral Fingerprinting

This module monitors the behavior of resource-constrained spectrum sensors in periodic and configurable time windows to create behavior fingerprints that are sent to the Cyberattacks Detection module. To achieve this functionality, the *Data Acquisition* component monitors the Linux kernel events shown in Table II and dealing with the usage of CPU, memory, network interface, file system, scheduler, device drives, or random number (step 1 in Fig. 9). The criteria to select the events relies on covering the most diverse and relevant hardware and software characteristics of resource-constrained spectrum sensors while ensuring the fingerprint stability across time. These events are monitored periodically using configurable time windows (details are provided in Section VI). In particular, the *Perf* Linux command is used for monitoring purposes. Once the events are measured, their values are saved in a behavioral vector containing a timestamp. Finally, the *Communication* component sends the behavioral vector to the Cyberattacks Detection module (step 2 in Fig. 9).

B. Cyberattacks Detection

The Cyberattacks Detection module considers unsupervised ML/DL models because the diversity of SSDF attacks and potential configurations are too high to assume that they could be modeled and considered for training. Therefore, only normal behavior would be available during training in a real scenario. In this sense, the nature of spectrum sensors, with a reduced set
TABLE II  
Features, Categorized Into Event Families, Monitored by CyberSpec to Detect Attacks on IoT Spectrum Sensors. The Final Selected Features are Shown in Blue

| Family         | Features                                                                 |
|----------------|-----------------------------------------------------------------------------|
| Network        | tcp:tcp_destroy_sock tcp:tcp_probe udp:udp_fail_queue_rcv_skb net:net_dev_queue | net:net_dev.stat ql:ql:ql:ql:dequeue skb:consume_skb skb:kfree_skb |
| Virtual Memory | writeback:writeback_global_dirty_state writeback:writeback_clean_inode_dirty_page | writeback:writeback_pages:written writeback:writeback_single_inode writeback:writeback:write_inode kmem:kmem_cache:free kmem:kmem_cache:alloc |
| Pile Systems   | jbd:jbd2:handle_start jbd:jbd2:commit block:block_bio_backmerge block:block_bio_remap | block:block:dirty_buffer block:block:getrq block:block:touch_buffer block:block:wning cachetables:cachetables_create cachetables:cachetables:lookup cachetables:cachetables:mark_active filmapmap:mm:filmap:map add_to_page_cache |
| Scheduler      | sched:sched_process_exec sched:sched_process_free sched:sched_process_wait sched:sched_switch | signal:signal:deliver signal:signal:generate task:task:recv task:task:nexttask cpus:cpus:cpus:cpus |
| CPU            | clk:clk:clk_rate rpm:rpm:resume | rpm:rpm:suspend ipi:ipi:raise |
| Device Drivers | irq:irq:handler:entry mmc:mmc:enable request:start | preempt:irq:enable gpio:gpio:enable dma:fence:mem healthcare_init |
| Random Numbers | random:get_random_bytes | random:random:pool:bytes:unlock random:random:read |

of normal behaviors, makes the assumption of monitoring them during training feasible. This module hosts two processes. The first is performed offline and consists of training unsupervised ML/DL models with the normal behavior of the spectrum sensor. Once it is done, an online process periodically evaluates the real-time behavior of the sensors with the trained models to detect anomalies produced by cyberattacks.

The offline process is executed before the online one and it is composed of the following four components: Dataset Generation, Data Curation, Algorithm Selection, and Model Training. The Dataset Generation periodically receives data vectors modeling the normal (benign) behavior of each resource-constrained spectrum sensor and creates a dataset for each device. The dataset creation task is configurable and its duration depends on the characteristics of spectrum sensor and platform where the framework is deployed. It is critical to ensure that no attacks affect the device during this period and contextual noise is minimized. Once the dataset is created, the Data Curation component performs several tasks. The first is to remove noisy vectors, features with constant values, features providing no relevant information, highly correlated features, and features with different data distributions across the time and between sensors. After performing these tasks, the resulting features are shown in Table II in blue. Finally, other important tasks of the Data Curation process are i) splitting the datasets into training (72%), validation (18%), and testing (10%), and ii) normalizing the values of the features. The data splitting distribution has followed the typical standards used in the literature for ML/DL projects pipeline. First, the original dataset is split into 90% for training and algorithm optimization, and 10% for testing. Then, the 90% is split into 80/20% for training and validation, resulting in the final 72% for training and 18% for validation over the original complete dataset. In addition, after evaluating other distributions in the range of training 70-80%, validation 15-20%, and testing (15-10%), the obtained results were similar to the ones shown in the next section. Once the final list of features is decided, the Algorithm Selection component selects several unsupervised ML/DL algorithms (explained in Section VI) to be trained by the framework offline process. After that, the Model Training component receives the selected algorithms and feeds them with the training set to build the models. Finally, the performance of each model is evaluated with the validation set, and the models obtaining the best scores are sent to the online process (see step 3 in Fig. 9).

The online process is periodically executed to detect cyber-attacks affecting the behavior of resource-constrained spectrum sensors. With that goal in mind, when the Model Evaluation component has the models it evaluates the received periodic behavior vectors (step 4 in Fig. 9). Each evaluation predicts if the current behavior vector is abnormal, which means that the spectrum sensor is infected.

VI. VALIDATION SCENARIO & EXPERIMENTS

ElectroSense [2] is the crowdsensing RF spectrum monitoring platform selected to validate the CyberSpec framework and to measure its performance when detecting anomalies produced by the SSDF attacks of Section IV.
As validation scenario, this work has deployed a set of Raspberry Pis equipped with SDR kits and the ElectroSense software sensing the RF spectrum. Sensors monitor the PSD values of frequency bands segments ranging from 20 MHz to 1.6 GHz in blocks of 2.4 MHz. This process takes about 50 s. The following Raspberry Pi acting as ElectroSense sensors have been considered:

- **Six Raspberry Pis 3 Model B** with Quad Core 1.2 GHz Broadcom BCM2837 64-bit CPU, and 1 GB of RAM.
- **Three Raspberry Pis 4** with Quad Core 1.5 GHz Broadcom BCM2711 64-bit CPU, and 2 GB of SDRAM.

These nine spectrum sensors are connected to the Internet through the next four local area networks, which are deployed in different geographical locations.

- **LAN_1**: One Raspberry Pi 3.
- **LAN_2**: One Raspberry Pi 4.
- **LAN_3**: Two Raspberry Pis 3 & one Raspberry Pi 4.
- **LAN_4**: Three Raspberry Pis 3 & one Raspberry Pi 4.

As indicated in Section V, the Behavior Fingerprinting module of CyberSpec is deployed on each sensor. In particular, the module monitors the sensor behavior in time windows of 50 s and requires 6.8 s to pre-process the events. 50 s is the minimum monitoring time needed to capture all SSDF attacks affecting any RF segment band since it is the time needed by the sensors to scan the spectrum. In addition, the Cyberattack Detection module is deployed on spectrum sensors or a server depending on the characteristics of the experiments presented in Section VI-A.

In such a scenario, the normal behavior of each Raspberry Pi was monitored for eight days, and a dataset per device was created. The monitoring duration depends on the dynamicity and functionality of the crowdsensing platform and sensors. ElectroSense is a platform where final users can access sensors to monitor spectrum data. In addition, the platform has some pre-defined spectrum data monitoring campaigns and sensor maintenance functions that are executed on random and periodic days. To deal with this variability, we decided to monitor one week (seven days) plus one additional day to check the events periodicity and fingerprint stability. In summary, according to our data analysis experiments, eight days is a reasonable time to model both random and periodic platform behaviors affecting the internal behaviors of sensors. After that, several configurations of the seven attacks defined in Section IV were sequentially executed in each Raspberry Pi for two hours, creating one dataset per attack configuration and device. Attacks configurations differ in the affected spectrum bandwidth to simulate and hide heterogeneous wireless communications standards. In particular, the selected configurations of attacks affect transmissions technologies such as WiFi (from 20 MHz to 160 MHz depending on the 802.11 version), Bluetooth (2 MHz), 3G (200 kHz), and SOS (20 kHz). At this point, it is worth mentioning that each Raspberry Pi has been infected with only one attack at the same time. The combination several attacks in parallel is not realistic because hybrid attacks already consider the execution of malicious behaviors in parallel (see Section IV). As a summary, the following datasets were created per sensor and are available in [8].

- One dataset with normal behavior (~192 h of monitoring and ~12173 samples).
- Forty-two datasets with malicious behavior (~2h of monitoring and ~120 samples each one). In detail, six datasets with a different bandwidth configuration (20 kHz, 200 kHz, 2 MHz, 20 MHz, 80 MHz, and 160 MHz) per attack (Noise, Spoof, Repeat, Confusion, Mimic, Freeze, and Delay).

### A. Experiments

The next three experiments evaluate the effectiveness and efficiency of CyberSpec when detecting the previous configurations of SSDF attacks (see Section IV). In addition, for each experiment, Table IV summarizes where the CyberSpec modules are deployed and the experiment setup.

- **Exp_1**: Performance analysis of global models per Raspberry Pi.
- **Exp_2**: Performance analysis of models per type of device, with different amounts of Raspberry Pis 3 used during training and evaluation.
- **Exp_3**: Performance analysis of individual models per Raspberry Pi.

To perform the three experiments, a common methodology was followed. It starts by choosing the datasets modeling the normal behavior of the selected Raspberry Pis (different for each experiment). After that, the pipeline indicated in Section V is followed to prepare the data for the ML/DL phase. Next, a set of unsupervised anomaly detection algorithms are selected. In particular, Autoencoder, Isolation Forest (IF), Copula-Based Outlier Detection (COPOD), Local Outlier Factor (LOF), and One-Class Support Vector Machine (OC-SVM) with a diverse set of hyperparameters are considered at this stage (see Table III). It is important to mention that these algorithms have been selected because of their good performance in similar cybersecurity detection scenarios and their heterogeneity in terms of implementation. The next step is to adjust the threshold to detect anomalies by using a well-known statistical approach. In this sense, the Interquartile Rule [21] is applied to find outliers. This technique calculates the quartiles and the interquartile range (IQR) in the scores of the training instances and defines two thresholds calculated as $Q1 - 1.5*IQR$ and $Q3 + 1.5*IQR$. Then, any score below or over these values is treated as an anomaly. In the case of the Autoencoder, the Mean Squared Error (MSE) of the input reconstruction (difference between model output and input in each feature) was used as a score, and in the case of LOF, IF, COPOD, and OC-SVM, the raw algorithm outputs were used. After that, a final selection of the hyperparameters and the threshold is made using the validation dataset (see Table III). At this point, it is important to mention that the hyperparameters selection for the three experiments has been made using the configuration of Exp_3. Another alternative is to perform a hyperparameter search per model of each experiment. Still, the minimum impact of different hyperparameters in models of Exp_3, the number of models (more than 100 trained across all experiments), and the similarity of data between experiments make individual hyperparameter optimization a non-essential
TABLE III
ANOMALY DETECTION ALGORITHMS AND HYPERPARAMETERS TESTED & SELECTED

| Algorithm     | Hyperparameters tested                      | Hyperparameters selected |
|---------------|---------------------------------------------|--------------------------|
| Autoencoder   | layers ∈ [1, 2, 3], neurons_layer ∈ [10, 60] (3 neuron steps) | layers = 1, neurons = 40 |
| LOF           | n_neighbors ∈ [3, 25] (one neighbor steps)   | n_neighbors = 15         |
| OC-SVM        | gamma ∈ [0.001, 0.01, 0.1, 1, 10, 100]      | kernel = \( rbf' \), gamma = 0.001 |
|               | kernel ∈ \{ 'rbf', 'linear', 'sigmoid', 'poly' \} |                           |
|               | degree ∈ [2, 3, 4, 5] (only poly kernel)    |                           |
| IF            | Number_of_trees ∈ [50, 1000] (50 tree steps) | Number_of_trees = 150    |
| COPOD         | -                                           | -                        |

TABLE IV
EXPERIMENTS CONFIGURATION

| Exp. | CyberSpec | Configuration |
|------|-----------|---------------|
| Exp_1| Monitor: RasPi Detection: RasPi | -One model per RasPi (7 RP3 & 3 RP4) |
| Exp_2| Monitor: RasPi Detection: Server | -One model combining 7/6/5/4/3/2 RP3 |
| Exp_3| Monitor: RasPi Detection: Server | -One model combining 7 RP3 & 3 RP4 |

task for solution performance evaluation. In addition, the main purpose of this work is not to find the best hyperparameters configuration per model. Finally, the detection performance of each algorithm is evaluated with the normal and the behavior under attack of each device (in each experiment, different data setups are used for training and testing). The metrics used to compare the detection performance are the TNR (True Negative Rate) and TPR (True Positive Rate). TNR indicates the number of non-anomalies found in the normal or benign behavior, while TPR is used to evaluate malicious behavior and provides the number of detected anomalies. With TPR and TNR it is possible to calculate the false positive and negative rates (FPR and FNR) as 1-TNR and 1-TPR, respectively. TPR and TNR are suitable metrics for this problem because each behavior is evaluated individually in a given dataset. Therefore, ML/DL models do not evaluate unbalanced datasets where TPR and TNR could provide biased conclusions. Due to the requirements of the experiments, the previous steps dealing with the Cyberattacks Detection module of CyberSpec have been deployed on each spectrum sensor for Exp_1, and a sever for Exp_2 and Exp_3.

1) Exp_1: Individual Models: This experiment employs the nine Raspberry Pis (six Raspberry Pis 3 and three Raspberry Pis 4) and their training datasets (modeling the 72% of their normal behaviors) to train the five selected algorithms (Autoencoder, IF, LOF, COPOD, and OC-SVM) per device (45 ML/DL models in total). After that, the individual models of each device have been evaluated with (i) the testing dataset (modeling the 10% of normal behavior) of that device, and (ii) the 42 datasets modeling the different configurations of the SSDF attacks affecting that device.

For each algorithm, Fig. 10 shows the TNR mean and standard deviation of each individual model algorithm when they are evaluated with normal behavior. As can be seen, IF is the model obtaining the best TNR (95%) and TPR (100%) for all configurations of Noise, Spoof, Confusion, Mimic and Delay. Despite IF is the model obtaining the best TNR, as can be seen, it is not able to detect some attacks like Noise or Spoof (detected by other models). Furthermore, Repeat and Freeze attacks deserve special consideration since they are not adequately detected until the affected bandwidth reaches 80 and 160 MHz (only by Autoencoder). Exploring more in detail these two attacks, it can be stated that the perf events monitoring the file system cannot detect the increment of reading operations. This is something interesting because it does not happen with writing operations. Because of that, Repeat and Freeze are not detected as well as Mimic.

![Average TNR performance of individual ML/DL models when detecting normal behavior.](image)

In conclusion, Autoencoder and OC-SVM are the two models showing the best detection of normal and SSDF attacks. In particular, all attacks are perfectly detected, less Repeat and Freeze. These two attacks read many files in the sensor, but this behavior does not impact the monitored events. However, when the affected bandwidth is sufficient (80 and 160 MHz), the attacks impact the rest of monitored events, and these variations are slightly detected. The impossibility of detecting attacks reading files is the main limitation of this work. In addition, having individual models per device might be cumbersome to maintain in dynamic platforms, such as ElectroSense, with a significant number of new devices per month.
2) **Exp.2: Device-Type Models:** This experiment evaluates the performance of models per device type (one model for Raspberry Pis 3 and another for Raspberry Pis 4). In particular, it analyzes the fact of having device-type models trained and evaluated with the normal behavior of all devices, and (ii) trained with different numbers of devices and evaluated with the rest.

Regarding the first setup, the training datasets (modeling the 72% of the normal behavior) of the 100% of Raspberry Pis 3 (six devices) are combined to feed the five selected algorithms per device family. After that, the testing datasets of the six Raspberry Pis 3 (modeling the 10% of their normal behavior) are concatenated and evaluated. This process is repeated for the three Raspberry Pis 4. Fig. 12 shows the TNR for each algorithm and device type (light green for Raspberry Pis 3 and blue for Raspberry Pis 4). As can be seen, for both families of devices, the performance is similar (being Autoencoder the one obtaining greater differences for both families with 0.85% and 0.93% TNR), showing that the proposed features and framework are suitable for Raspberry Pis 3 and 4. In addition, comparing these results with the obtained by the individual models of the previous experiment, the differences are minimal as well.

Regarding the detection of attacks, Fig. 13 shows the mean TPR of the Raspberry Pi 3 & 4 family models when detecting anomalies contained in the 42 malicious datasets per device (252 datasets of Raspberry Pis 3 and 126 of Raspberry Pis 4). These results are similar to those shown in Fig. 11. In particular, all configurations of Noise, Spoof, Confusion, Mimic, and Delay are detected by Autoencoder, LOF, and OC-SVM. These results show a promising path to have models per family of devices, reducing the number of models in the platform and, therefore, the maintenance cost.

Once the suitability of device type models is demonstrated, the second setup of this experiment evaluates the scalability of the proposed solution when new devices appear on the platform (unseen during the training phase). In this sense, different tests have been performed by excluding the 15%, 33%, 50%, 66%, and 85% of Raspberry Pis 3 (1, 2, 3, 4, and 5 devices, respectively) from training. All combinations of devices have been performed for each test to show robust results. In other words, and as an example, when one device is excluded from the training phase, all combinations (6 in total) have been considered to train the models. It is essential to mention that only Raspberry Pis 3 have been selected due to the number of available devices. As in the previous experiment, 72% of the normal behavior of the selected devices has been used for training and 100% of the normal behavior of excluded devices for evaluation. Fig. 14 shows the TNR mean and standard deviation of Autoencoder, LOF, and OC-SVM. Only these three algorithms are shown in the figure due to they are the ones obtaining 100% TPR for all configurations of Noise, Spoof, Confusion, Mimic, and Delay (in this experiment test and the previous ones).

As can be seen in Fig. 14, regardless of the number of excluded devices, OC-SVM is the model providing the best averaged TNR. In addition, and as expected, the higher the number of devices excluded from the training, the lower the TNR. This is due to the fact of having independent data with different distributions (also known as non-IID) per device. These different data distributions could be influenced by external factors such as network bandwidth, traffic, device temperature, or processes. In any case, OC-SVM obtains good TNR averages (80%, 75%, 65%, 55%, 45%).
Fig. 13. Average TPR performance of device-type ML/DL models when detecting SSDF attacks.

Fig. 14. Average TNR performance of device-type ML/DL models trained with different numbers of Raspberry Pis 3 when detecting normal behavior.

and 70%) when 85%, 66%, and 50% of devices are used only during training. Fig. 14 also shows that the OC-SVM models work very well for the majority of devices (high TNR mean) but bad for a few of them (high standard deviation). Furthermore, when more than 50% of devices are excluded from training, the detection performance of all models drops to 50%.

In conclusion, the detection results and scalability obtained by OC-SVM are promising for device type models even when 30% of evaluated devices are new. These results are slightly worse than the ones achieved in Exp_1 having one model per device, but still good. However, more sensors connected to the same and different networks are required to confirm the positive trend. In terms of limitations, as in the previous experiment, the main drawback is the low detection rate for Repeat and Freeze affecting small bandwidths. Additionally, this scenario does not protect the privacy of sensors fingerprints since models are trained and evaluated on a server.

3) Exp_3: Global Model: This last experiment consists of training a global model that combines the normal behavior of all Raspberry Pis 3 and 4. For that, it combines the nine datasets modeling the 72% of the devices normal behavior, and the samples of each dataset were balanced for the training process of the five selected algorithms. Finally, the resulting models were
evaluated with both normal (testing dataset of each device with 10% of samples) and under attack behaviors of all devices. 

Fig. 15 shows the TNR of global models (light green bars), models-type for Raspberry Pi 3 (blue bars) and Raspberry Pis 4 (orange bars), and individual models for a Raspberry Pi 4 (gray bars) when detecting normal behaviors. As shown, there are no significant differences in the TNR of the three experiments, indicating that models combining the normal behavior of Raspberry Pis 3 and 4 perform reasonably well when detecting normal behaviors. Regarding the detection of anomalies, the performance of the five ML/DL algorithms was evaluated. Still, for the sake of simplicity, Table V shows the TPR obtained by OC-SVM, the one providing the best ratio of TNR/TPR performance. It is crucial to mention that these results have been calculated by computing the average mean of the TPR score provided by each Raspberry Pi 3 and 4.

To explain predictions done by models, supervised algorithms such as Decision Trees and Random Forest can calculate the list of most relevant features. However, in unsupervised ML models, like the ones used in this work, this is not a straightforward task since there are no labels, and models do not present that functionality. However, this experiment has selected the datasets modeling all SSDF attacks affecting 2MHz bandwidth and calculated each feature correlation ratio (under normal and attack conditions). Below, the ten features with a higher correlation are presented.

- random:urandom_read
- jbd2:jbd2_start_commit
- writeback:writeback_written
- writeback:writeback_write_inode

4) Resource Consumption: Apart from the detection performance, it is also relevant to measure the consumption of resources of CyberSpec for each experiment. In this sense, for Exp_1, the monitoring and detection modules have been deployed on each sensor, and for Exp_2 and Exp_3 the monitoring is deployed on a sensor and the detection on a server. Table VI shows the consumption of resources per module of the CyberSpec framework for each experiment. The first two rows indicate the CPU, RAM, and storage consumed by the monitoring and training process of the detection module when they are deployed on a sensor (Exp_1). The third and fourth rows show the same info, but when the detection module is deployed on a server with Intel i7-5930K CPU @ 3.50GHz, 3 NVIDIA GTX1080 GPUs, and 96 GB RAM (Exp_2 and Exp_3). In terms of time, the monitoring takes 56.8 s and the detection less than 1 s on evaluating samples (the training time is not relevant because it can be done offline). As in Exp_1 the training and evaluation are directly done on the sensor, Table VII depicts the resource consumption of the different algorithms tested. As can be seen, OC-SVM not only has the best performance but also offers a good balance between train time (≈37 secs), test

---

**TABLE V**

| Attack   | 20 kHz | 200 kHz | 2 MHz | 8 MHz | 160 MHz |
|----------|--------|---------|-------|-------|--------|
| Mouse    | 61%    | 99%     | 100%  | 100%  | 100%   |
| Spool    | 99%    | 100%    | 100%  | 100%  | 100%   |
| Repeat   | 6%     | 26%     | 18%   | 26%   | 26%    |
| Confusion| 100%   | 100%    | 100%  | 100%  | 100%   |
| Mimic    | 100%   | 100%    | 100%  | 100%  | 100%   |
| Dereeze | 7%     | 24%     | 30%   | 28%   | 32%    |
| Delay    | 100%   | 100%    | 100%  | 100%  | 100%   |

---

**Table VI**

| Exp.            | Module          | CPU          | RAM      | Storage |
|-----------------|-----------------|--------------|----------|---------|
| All (sensor)    | Monitoring      | 0.5-2% 1 Core with 8% peaks | 900 kB     | 7.8 kB   |
| Exp_1 (sensor)  | Detection (OC-SVM) | 100% during ≈ 37 s (training) | 1.17 MB   | 1.05 MB  |
| Exp_2 (server)  | Detection (OC-SVM) | 100% during ≈ 229 s (training) | 3.5 MB     | 3.34 MB  |
| Exp_3 (server)  | Detection (OC-SVM) | 100% during ≈ 810 s (training) | 10.5 MB    | 10.46 MB |

---

**Table VII**

| Algorithm              | Train Time | Test Time | RAM | Storage |
|------------------------|------------|-----------|-----|---------|
| Autoencoder            | ≈ 152.43 s | ≈ 0.21 s   | 62.48 KB | 62.48 KB |
| IF                     | ≈ 6.31 s   | ≈ 0.27 s   | 1.23 MB  | 1.17 MB  |
| LOF                    | ≈ 8.97 s   | ≈ 0.021 s  | 4.7 MB   | 4.48 MB  |
| COPOD                  | ≈ 1.72 s   | ≈ 0.72 s   | 29.9 MB  | 28.3 MB  |
| OC-SVM                 | ≈ 37.17 s  | ≈ 0.001 s  | 1.17 MB  | 1.05 MB  |

- writewb:sb_clear_inode_writeback
- writewb:writeback_pages_written
- writewb:wbcs_writepage
- writewb:writeback_single_inode
- irq:irq_handler_entry
- writewb:global_dirty_state

In conclusion, a global model combining all Raspberry Pis is able to detect all configurations of Spoof, Confusion, Mimic, and Delay attacks. The Noise attack was ideally detected almost for all configurations, excluding the less impactful (20kHz). These results can be explained by looking at the most relevant features, where the generation of random numbers and writing operations are the most relevant features. Therefore, this approach is the most practical one due to the fact of having to maintain only one model while providing acceptable detection capabilities. Finally, the main limitation is that Repeat and Freeze are not adequately detected, as in the previous two experiments.
time of one single sample (0.001 secs), and memory/storage usage (≈1.1 MB). Therefore, combining detection performance and resource consumption, it is the most suitable model for resource-constrained spectrum sensors.

In summary, CyberSpec takes less than 60 s to detect attacks, and it consumes the CPU, RAM, and storage of the sensors in an acceptable and reduced manner.

VII. SUMMARY, CONCLUSIONS, AND FUTURE WORK

This work introduced seven SSDF attacks (Repeat, Mimic, Confusion, Noise, Spoof, Freeze, and Delay) affecting crowdsensing spectrum sensors. Additionally, it presented CyberSpec, an ML/DL-oriented framework that monitors internal events of spectrum sensors, such as network, virtual memory, files system, scheduler, system calls, CPU, device drivers, and random numbers, to detect anomalies produced by SSDF attacks. The performance of CyberSpec has been evaluated in a real crowdsensing spectrum monitoring platform called ElectroSense, where nine Raspberry Pis 3 and 4 acting as spectrum sensors were infected with the previous SSDF attacks. Three experiments with different unsupervised ML/DL models evaluated the CyberSpec detection capabilities. Individual ML/DL models per device, models per family of devices (excluding different % of devices from training), and global models combining all devices have provided a promising performance when detecting the normal behavior of six Raspberry Pis 3 and three Raspberry Pis 4. Five (Noise, Spoof, Confusion, Mimic, and Delay) of the seven analyzed attacks are almost perfectly detected (100% TPR) by Autoencoder, LOF, and OC-SVM in the three experiments. Having models combining normal behaviors of different devices does not reduce the detection performance significantly. Moreover, looking at the scalability of device-type models, when the 15%, 33%, and 50% of devices are excluded from training, they perform relatively well for most excluded devices (80-70% TPR and 100% TNR for five attacks). Repeat and Freeze are not correctly detected because CyberSpec does not monitor read file operations, and the behavior of these attacks relies on that. Finally, the CyberSpec framework needs less than 60 s to detect normal/under attack behavior and OC-SVM is the model showing the best trade-off between detection performance and consumption of time, CPU, RAM, and disk.

With the goal of answering the open challenges depicted in Section I, the main conclusions after performing the experiments and analyzing the results are (i) device fingerprinting is appropriate to detect precisely SSDF attacks; (ii) individual and global ML/DL models show a promising detection performance in scenarios with a reduced number of ElectroSense sensors, but more experiments are needed to evaluate its scalability; and (iii) additional behavioral events and features are required to properly detect SSDF attacks having a slight impact on the internal behavior of the ElectroSense sensors.

Future work will evaluate the performance of privacy-preserving mechanisms. Solutions like the proposed in this work, where global models are trained with the behavior of different devices, are not suitable for privacy-preserving scenarios where federated learning can be a solution to train models while preserving data sensitiveness. Furthermore, supervised algorithms will be investigated to classify cyberattacks in different families. Finally, adding new event families like system calls could improve the detection performance, especially in attacks such as Repeat and Freeze.

REFERENCES

[1] K. Riad, T. Huang, and L. Ke, “A dynamic and hierarchical access control for IoT in multi-authority cloud storage,” J. Netw. Comput. Appl., vol. 160, 2020, Art. no. 102633.
[2] S. Rajendran et al., “Electrosense: Open and big spectrum data,” IEEE Commun. Mag., vol. 56, no. 1, pp. 210–217, Jan. 2018.
[3] J. Li, Z. Feng, Z. Peng, and P. Zhang, “A survey of security issues in cognitive radio networks,” Chin. Commun., vol. 12, no. 3, pp. 132–150, 2015.
[4] F. A. Meneghelli, M. Calore, D. Zuccchetto, M. Polese, and A. Zanella, “IoT: Internet of Threats? A survey of practical security vulnerabilities in real IoT devices,” IEEE Internet Things J., vol. 6, no. 5, pp. 8182–8201, Oct. 2019.
[5] P. S. Chatterjee, “Systematic survey on SSDF attack and detection mechanism in cognitive wireless sensor network,” in Proc. Int. Conf. Intell. Technol., 2021, pp. 1–5.
[6] J. M. Jorduca Valero, P. M. Sánchez Sánchez, M. Gil Pérez, A. Huertas Celdrán, and G. Martínez Pérez, “Cutting-edge assets for trust in 5G and beyond: Requirements, state-of-the-art, trends & challenges,” ACM Comput. Surv., vol. 55, pp. 1–36, 2023.
[7] P. M. Sánchez Sánchez, J. M. Jorduca Valero, A. Huertas Celdrán, G. Barbuiya, M. Gil Pérez, and G. Martínez Pérez, “A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets,” IEEE Commun. Surv. Tut., vol. 23, no. 2, pp. 1048–1077, Second Quarter 2021.
[8] P. M. Sánchez Sánchez and A. Huertas Celdrán, “CyberSpec dataset and source code,” 2022. Accessed: Oct. 19, 2022. [Online]. Available: https://github.com/sxz0/CyberSpec.
[9] K. Yadav, S. Dhar Roy, and S. Kundu, “Defense against spectrum sensing data falsification attacker in cognitive radio networks,” Wirel. Pers. Commun., vol. 112, pp. 849–862, 2020.
[10] F. R. Yu, H. Tang, M. Huang, Z. Li, and P. C. Mason, “Defense against spectrum sensing data falsification attacks in mobile ad hoc networks with cognitive radios,” in Proc. IEEE Mil. Commun. Conf., 2009, pp. 1–7.
[11] H. Chen, M. Zou, L. Xie, and J. Li, “Cooperative spectrum sensing with M-Ary quantized data in cognitive radio networks under SSDF attacks,” IEEE Trans. Wireless Commun., vol. 16, no. 8, pp. 5244–5257, Aug. 2017.
[12] Y. Fu and Z. He, “Bayesian-inference-based sliding window trust model against probabilistic SSDF attack in cognitive radio networks,” IEEE Syst. J., vol. 14, no. 2, pp. 1764–1775, Jun. 2020.
[13] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, “HADES-IoT: A practical host-based anomaly detection system for IoT devices,” in Proc. ACM Asia Conf. Comput. Commun. Secur., 2019, pp. 479–484.
[14] J. Jeon, J. H. Park, and Y. Jeong, “Dynamic analysis for IoT malware detection with convolution neural network model,” IEEE Access, vol. 8, pp. 96899–96911, 2020.
[15] A. De Lorenzo, F. Martellini, E. Medvet, F. O. Mercaldo, and A. Santone, “Visualizing the outcome of dynamic analysis of Android malware with VizMAl,” J. Inf. Secur. Appl., vol. 50, 2020, Art. no. 102423.
[16] P. Mishra, V. Varadarajan, E. S. Pilli, and U. Tupakula, “VMGuard: A VMI-based security architecture for intrusion detection in cloud environment,” IEEE Trans. Cloud Comput., vol. 8, no. 3, pp. 957–971, Third Quarter 2020.
[17] R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, “Anomaly detection using resource behaviour analysis for autoscaling systems,” in Proc. IEEE 4th Conf. Netw. Softwarization Workshops, 2018, pp. 192–196.
[18] S. Barbuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos, “RADS: Real-time anomaly detection system for cloud data centres,” 2018, arXiv:1811.04481.
[19] K. Ott and R. Mahapatra, “Hardware performance counters for embedded software anomaly detection,” in Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervasive Intell. Comput., 2018, pp. 528–535.
[20] S. Alrabaee, M. Khasawneh, A. Agarwal, N. Goel, and M. Zaman, “Applications architectures and protocol design issues for cognitive radio networks: A survey,” *Int. J. Wirel. Mobile Comput.*, vol. 7, no. 5, pp. 415–427, 2014.

[21] H. P. Vinutha, B. Poornima, and B. M. Sagar, “Detection of outliers using interquartile range technique from intrusion dataset,” in *Proc. Inf. Decis. Sci.*, 2018, pp. 511–518.

Alberto Huertas Celdrán received the MSc and PhD degrees in computer science from the University of Murcia, Spain. He is a senior researcher with the Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH. His scientific interests include cybersecurity, machine and deep learning, continuous authentication, and computer networks. For more information, please visit: huertas@ifi.uzh.ch

Pedro Miguel Sánchez Sánchez received the MSc degree in computer science from the University of Murcia, Spain. He is currently working toward the PhD degree in computer science with the University of Murcia. His research interests focus on continuous authentication, networks, 5G, cybersecurity, and machine learning and deep learning. For more information, please visit: pedromiguel.sanchez@um.es

Gérôme Bovet received the PhD degree in networks and computer systems from Telecom ParisTech, France. He is the head of data science for the Swiss Department of Defense. His work focuses on machine and deep learning, with an emphasis on anomaly detection, adversarial and collaborative learning in IoT sensors. For more information, please visit: gerome.bovet@armasuisse.ch

Gregorio Martínez Pérez is a full professor with 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. For more information, please visit: gregorio@um.es

Burkhard Stiller received the MSc and PhD degrees from the University of Karlsruhe, Germany. He chairs the Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, as a full professor. His main research interests include fully decentralized systems, network and service management, IoT, and telecommunication economics. For more information, please visit: stiller@ifi.uzh.ch