Diwall: A Lightweight Host Intrusion Detection System against Jamming and Packet Injection Attacks

MOHAMED EL BOUAZZATI, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, France
PHILIPPE TANGUY, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, France
GUY GOGNIAT, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, France
RUSSELL TESSIER, Department of Electrical and Computer Engineering, University of Massachusetts, USA

The rapid growth of Internet of Things (IoTs) applications in various sectors has led to a significant increase in the number of IoT devices. This has led to the deployment of numerous IoT protocols to provide greater connectivity. However, this extensive adoption has also left them vulnerable to attack. In particular, attacks targeting wireless communication capabilities represent a significant threat. Such attacks exploit various vulnerabilities in the wireless connectivity unit, compromising its security. To counter this threat, this paper proposes a Host Intrusion Detection System (HIDS) for detecting wireless attacks. Its components are customized to support IoT end-devices using low-GHz and sub-GHz data rate protocols. The HIDS deploys a hardware tracer to monitor microarchitecture and network metrics using hardware performance counters (HPCs). It performs monitoring of network and microarchitecture metrics for a 32-bit RISC-V based wireless connectivity unit. The HIDS uses analysis and classification of monitored data for detecting memory corruption and jamming attacks. We evaluate the effectiveness of the HIDS in detecting packet injection and jamming attacks. Our FPGA implementation of HIDS has a logic overhead of about 14.30% and 22.89% of flip flops (FFs) and lookup tables (LUTs), respectively, compared to the CV32E40P baseline on an Arty A7 100T board. The design frequency and code size penalties are less than 1% for a RISC-V processor with a LoRaWAN protocol stack.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; **Hardware security implementation**; • **Hardware → Reconfigurable logic and FPGAs**.

Additional Key Words and Phrases: HIDS, RISC-V, LoRaWAN, Jamming, Packet Injection, HPCs, EWMA, FPGA

ACM Reference Format:
Mohamed EL BOUAZZATI, Philippe TANGUY, Guy GOGNIAT, and Russell TESSIER. 2024. Diwall: A Lightweight Host Intrusion Detection System against Jamming and Packet Injection Attacks. *ACM Trans. Embedd. Comput. Syst.* 1, 1 (February 2024), 29 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The Internet of Things (IoTs) has experienced exceptional growth in recent decades and has become an integral part of the world. This has brought significant transformations to various aspects of our lives, enabling continuous data monitoring and real-time remote system control. IoT applications span across all sectors, such as healthcare, smart homes, smart cities, agriculture, and industrial automation. Associated consumer products are being

Authors' addresses: Mohamed EL BOUAZZATI, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, Lorient, France, F-56100, mohamed.el-bouazzati@univ-ubs.fr; Philippe TANGUY, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, Lorient, France, F-56100, philippe.tanguy@univ-ubs.fr; Guy GOGNIAT, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, Lorient, France, F-56100, guy.gogniat@univ-ubs.fr; Russell TESSIER, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA, tessier@umass.edu.

1539-9087/2024/2-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

increasingly integrated into daily routines. Indeed, the spread of IoT devices has been remarkable, with billions now in use globally. For instance, the global IoT market reached 16.7 billion devices and is projected to grow to 29.7 billion by 2027 [1]. However, along with this growth, there are increasing challenges related to security and data privacy. Cyberattacks targeting IoT devices have surged significantly, with millions of incidents reported annually. In 2016, the cybersecurity landscape was significantly affected by the Mirai botnet attack [2]. This attack searches for open telnet ports in numerous IoT devices often insecure, like digital cameras. Attackers successfully attempted to gain access using default passwords. Then, they constructed a botnet and unleashed Distributed Denial of Service (DDoS) attacks. In 2022 alone, there were over 112 million IoT-targeted attacks worldwide, a stark increase compared to the approximately 32 million detected cases in 2018 [36]. Addressing the security challenges is crucial for ensuring the stability of the current and the future growth. The IoT environment's topology is meaningful, as highlighted in Figure 1a, spanning from the top where cloud servers manage vast amounts of data, to the bottom housing IoT devices transmitting their data to gateways in the middle. Centering on IoT devices and their wireless connections to gateways, these physical components interact with the physical world. They employ IoT protocols to establish wireless connections with gateways, as illustrated in the lower part of Figure 1a. A subset of commonly employed IoT devices, referred to as IoT end-devices, possess limited resources in terms of performance, computing capabilities, memory capacity, and power consumption.

Embedded systems in IoT devices now have built-in communication capabilities. Generally, a dedicated unit is employed to implement the IoT protocol stack, frequently based on open-source stacks on the market. This unit provides the primary pathway for data to enter the IoT network, representing a crucial entry point that is susceptible to threats. It hosts various vulnerabilities that malicious parties can exploit to launch cyberattacks within the IoT network. As an illustrative example, Forescout Research Labs uncovered 33 vulnerabilities, collectively referred to as AMNESIA:33 [22]. The discovered vulnerabilities are primarily related to memory corruption. They pose a significant risk to millions of IoT devices and affect four open-source TCP/IP IoT stacks. These vulnerabilities allow the attackers to compromise devices, gain remote control, execute malicious code, initiate denial-of-service (DoS) attacks, and steal data. The attackers can also access the IoT network where these devices are deployed. The emergence of high-performance, low-cost platforms and associated software [24] has also expanded the opportunities for attackers and simplified their access to the lower layers of communication networks.

The potential economic repercussions of these vulnerabilities are considerable, given that open-source IoT stacks are widely utilized in their products by numerous vendors. Addressing these vulnerabilities requires the deployment of patches across millions of IoT devices. However, a challenging aspect of this situation is that some IoT devices cannot be patched due to technical limitations.

To counter these security issues, embedded systems within IoT end-devices have been enhanced with various security mechanisms and countermeasures such as protecting physical access to devices, securing data privacy with cryptographic encryption, and enhancing communication security. Systems-on-Chip (SoCs) in IoT end-devices include security mechanisms against cyberattacks, such as cryptographic accelerators for data encryption, secure boot procedures, and over-the-air (OTA) update mechanisms [19, 37, 39]. These are crucial for addressing IoT device security challenges. However, the same devices lack monitoring and detection mechanisms that can track system metrics and analyze behavior to identify malicious activities. We argue that combining monitoring and detection with existing protection and update measures can enhance the resilience of IoT devices to security challenges.

In this paper, we provide a lightweight monitoring and detection method for detecting ongoing wireless attacks. The implemented security mechanism extends the work presented in [7], with important modifications targeting multi-level monitoring, and a new class of attack. To the best of our knowledge, prior research has predominantly focused on network-based IDS for IoT devices. These systems are typically based on a single-layer metric, which limits detection to specific wireless network attacks [15, 44]. Host-based IDS are frequently implemented at

the gateway level, taking advantage of the substantial resources available for handling complex detection tasks, often employing deep learning techniques. Other IDS solutions are designed around application-specific features, which limits their scalability to support different or multi-protocol environments [14, 33, 43, 45]. Therefore, there is a pressing need for lightweight, onboard strategies to monitor IoT devices and detect ongoing potential threats.

The main contributions of this work are as follows:

- The methodology and the associated experimental framework for simulating, emulating, and implementing wireless attacks, such as jamming and packet injection. This approach also includes dataset generation for microarchitectural and network metrics, which are essential for constructing a HIDS tailored for IoT end-devices.
- A lightweight HIDS called *Diwall*. It was designed to detect packet injection and jamming attacks by monitoring microarchitectural events and Received Signal Strength Indicator (RSSI) metadata within the LoRaWAN stack.

In our first contribution, we introduce our methodology and the associated framework to study packet injection and jamming attacks that target IoT protocol stacks. This framework generates extensive datasets for both simulated and real-world scenarios. We delve into the behavior of microarchitectural events during memory stack and heap buffer overflows, focusing on packet injection attacks. Additionally, we examine network metadata metrics such as RSSI and Signal-to-Noise Ratio (SNR) to identify jamming attacks. Our research encompasses machine learning and statistical techniques applied to the generated datasets. Importantly, this comprehensive framework is adaptable to various IoT protocol stacks, and instruction set architecture (ISA). This adaptability is possible because monitored microarchitectural and RSSI metadata are present in modern CPUs and IoT protocol stacks. Within this methodology, we employ a simplified MAC layer that can be substituted with more complex MAC layers that we demonstrated using the LoRaWAN stack. Similarly, the LoRa physical (PHY) layer can be replaced with technologies such as Bluetooth or Zigbee to investigate the same wireless attacks that pose threats to the aforementioned IoT stacks.

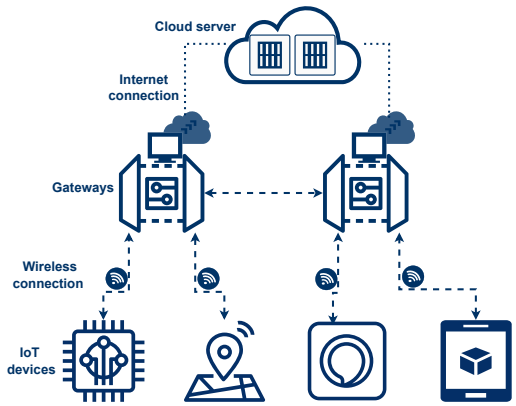
In our second contribution, we introduce *Diwall*, a hardware-based lightweight HIDS that requires minimal software instructions for configuration and control. It uses microarchitectural event analysis to identify packet injection attacks that exploit buffer overflow. It also utilizes RSSI-based metrics to detect jamming attacks. We implemented *Diwall* as a compact component within a RISC-V processor on an FPGA board. In real-world scenarios, we evaluated the detection rates and achieved high levels of approximately 99.94% accuracy. Moreover, *Diwall* requires a minimal FPGA area overhead of approximately 14.30% of LUTs and 22.89% of FFs compared to the CV32E40P baseline on an Arty-A7 100T board, and does not negatively impact system performance.

This paper is organized as follows: Section 2 provides background information regarding the security of SoCs with built-in wireless connectivity. Section 3 describes the proposed security mechanism approach and architecture. The methodology employed to assess the effectiveness of the proposed security mechanism is outlined in Section 4. In Section 5, the obtained results are presented along with a discussion of the limitations. Finally, the paper concludes with Section 6, summarizing key findings and providing suggestions for future work.

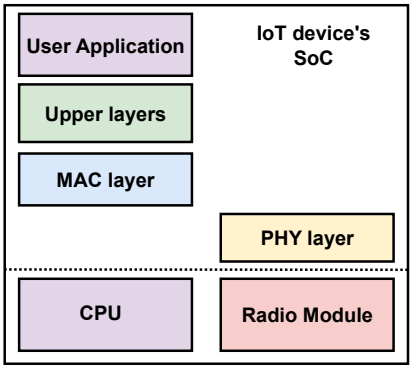
2 BACKGROUND

2.1 SoC with Built-in Wireless Connectivity

Many chip manufacturers have designed SoCs for IoT devices with built-in wireless capabilities [19, 37, 39]. These IoT SoCs integrate a range of network connectivity technologies, including commonly used IoT protocols such as LoRaWAN, Bluetooth, and ZigBee. The built-in radio ensures dedicated wireless network access, solving compatibility issues. They also reduce implementation costs, energy requirements, and complexity compared to traditional SoCs, which require expensive hardware for wireless connectivity.



(a) IoT environment topology



(b) IoT device's SoC block diagram with a hardware radio module and a CPU executing the network stack layers

Fig. 1. IoT use case overview

The ESP32-H2 [39] is a SoC developed by Espressif Systems and is part of the ESP32 series. It embeds a single-core, 32-bit RISC-V processor. The wireless connectivity unit supports the coexistence of radio protocols in the 2.4 GHz band, each protocol using a dedicated baseband Application-Specific Integrated Circuit (ASIC). Increasing flexibility and adaptability for wireless connectivity would involve integrating a dedicated network CPU. The STM32WL55CC [37] is a STMicroelectronics SoC, which is part of the STM32WL series. It combines a single-core Arm Cortex-M4 processor as the main CPU and a single-core Arm Cortex-M0+ dedicated to sub-GHz wireless connectivity. Various sub-GHz IoT stacks and proprietary protocols are supported, such as LoRaWAN and Sigfox. The CC1352R [19], produced by Texas Instruments, features a multiband device specifically tailored for IoT devices and proprietary protocols in the sub-GHz and 2.4 GHz bands. It incorporates a dual-core Arm Cortex-M4F/M0, supporting various protocol operations such as Thread, Zigbee, and IEEE 802.15. Wireless connectivity is managed by the single-core Arm Cortex-M0 and has the ability to manage several protocols simultaneously. In SoC architecture, wireless connectivity systems typically include a network processor dedicated to the PHY and MAC layer management. Without loss of generality, in this paper we focus on the SoC block diagram highlighted in Figure 1b, where the physical layer is implemented on a separate radio module, while the remaining stack, including the MAC layer and upper layers (network and application layers), is executed by the SoC's main CPU.

2.2 Security Threats

SoCs in IoT devices can have multiple weak points targetable by attacks. One entry point could stem from the application processor executing a malicious process that attempts to carry out illegitimate actions, such as launching a DDoS attack [2]. In such attacks, numerous IoT devices are exploited to overwhelm servers collectively. The second potential entry point relates to the vulnerabilities present in the lower layers of the protocol stacks. This paper primarily focuses on this entry point, specifically, the wireless connectivity of SoCs. Several vulnerabilities in IoT devices have recently been identified. These vulnerabilities have been found in the communication elements of the IoT devices, and affect several protocols. These include TCP/IP, LoRaWAN, ZigBee, and BLE. Numerous vulnerabilities are created during the implementation of the protocol standards. However, others result from the standard itself.

Forescout Research Labs conducted a security study on seven open-source TCP/IP stacks [22]. They found 33 new critical vulnerabilities in four stacks: uIP, FNET, picoTCP, and Nut/Net. These stacks are heavily used

in millions of IoT devices and by more than 150 vendors. Many of these vulnerabilities are traced back to poor software development practices. The exploitation of these vulnerabilities (with e.g., buffer overflow), referred to as AMNESIA:33, can result in exploits such as remote control execution and privilege escalation reached by attackers. These risks include compromised IoT devices, execution of malicious code, DoS attacks, and sensitive information theft. The authors of AMNESIA:33 acknowledge that identifying and rectifying these vulnerabilities pose a significant challenge to the security community. In response, they proposed mitigation measures, like implementing solutions that provide granular visibility of IoT devices through network communication monitoring. They also recommend isolating IoT devices or network segments that are vulnerable to these threats to manage potential risks.

A group of critical vulnerabilities, known as BLEEDINGBIT [5], affect BLE chips. They impact Wi-Fi access points from Cisco, Meraki, and Aruba solutions, which utilize Texas Instruments (TI) BLE chips. BLEEDINGBIT allows an unauthenticated attacker to execute code remotely on targeted chips. The first vulnerability, CVE-2018-16986, stems from a masking error within the Bluetooth specification, persisting from version 4.2 to version 5.0. The updated version enables the use of larger advertising packets, increasing the packet length from 37 bytes to 255 bytes. To accommodate this change, the packet header in the new specification has been reorganized, shifting the packet length field from 6 bits to 8 bits. This expansion is achieved by allocating 2 bits previously reserved for future use (RFU). The slight difference in the two specifications may have triggered memory corruption bugs when parsing BLE advertising packets. Developers might neglect to mask out RFU bits in the packet header length field, assuming they are always transmitted as zeros as per the specification. The second vulnerability, CVE-2018-7080, was discovered in the Over-the-Air (OTA) firmware download (OAD) feature of TI's BLE stack SDK. The Aruba access point utilizes this functionality on TI CC2540 chips for remote firmware updates. Although this feature is disabled by default on commercialized devices, it appears to be enabled in the 300 Series. An attacker with the appropriate credentials could exploit this vulnerability to gain access to an affected IoT device, replace the BLE chip's firmware with malicious code, and subsequently gain control of the device. This introduces a new attack vector on Aruba access points, potentially enabling an unauthenticated, over-the-air attack that compromises a secure Wi-Fi network.

The Tencent Blade team discovered two vulnerabilities in LoRaWAN, CVE-2020-11068 and CVE-2020-4060 [41]. These vulnerabilities can allow a remote DoS on LoRaWAN end-devices and potentially enable a remote code execution (RCE) exploit on the LoRaWAN gateway under certain conditions. CVE-2020-11068 was found in the LoRaWAN end-device stack implementation, specifically in versions below V4.4.4 of LoRaMac-node. This issue arises during the OTA Activation (OTAA) process and involves a reception buffer overflow due to the lack of packet size check. In contrast, CVE-2020-4060 is a use-after-free (UAF) vulnerability. This leads to memory corruption on 32-bit machines and was found in the LoRaWAN Gateway implementation, specifically the LoRa Basics Station. This station uses a Configuration and Update Server (CUPS) protocol to check for updates. The UAF issue occurs when the signature length of a message from a CUPS server exceeds 2 GB.

Multiple vulnerabilities have been reported in the implementation of numerous IoT protocol stacks, as well as in their specifications. In [4, 47], various attacks that exploit these vulnerabilities are discussed. They can affect service availability, data integrity, and confidentiality. Attackers can exploit the protocol stack's lower layer vulnerabilities, at the MAC layer or at the PHY layer for instance. They can thus introduce malicious network packets to target victims or jam their communication channels, resulting in e.g., packet injection, jamming, DoS or man-in-the-middle (MITM) attacks. Table 1 summarizes a comparison of common representative attacks on LoRaWAN, a sub-GHz protocol, and on Bluetooth, BLE, and ZigBee, three 2.4 GHz protocols. The comparison is based on the targeted protocols, exploited and targeted layers in the IoT protocol stack, as well as the used vulnerabilities and their results. From this table, we observe that several IoT protocol stacks are susceptible to attacks and vulnerabilities that exploit and target multiple layers of the protocol stack. This includes the exploitation of vulnerabilities hosted on the MAC layer and PHY layer, with attacks also targeting upper layers

Table 1. Security State-of-the-Art of IoT Low Data-Rates Protocols LoRaWAN, ZigBee, and BLE Reported with the Targeted Layer as T and the Exploited Layer (if any) as E.

Attack	Protocol	PHY	MAC	Upper Layers	Vulnerability	Results
Wazabee [12]	ZigBee	E	E/T	T	BLE API	DoS, injection
Selective Jamming [4]	LoRaWAN	E	E/T	T	Header plain-text	DoS, Wormhole
Spoofing [16]	LoRaWAN	E	E/T	-	Authentication	DoS
Rescuing [6]	LoRaWAN	-	T	T	Protocol weakness	Replay, DoS
InjectBLE [11]	BLE	E	E/T	T	Pairing	MITM, Sniffing
Downgrade [47]	BLE	-	-	T	Design flaws	DoS, MITM
Injection-free [34]	BLE	-	-	E/T	Limited Bounding	DoS, MITM
Downgrade [3]	BT/BLE	-	E/T	E/T	Insecure BLE clock	MITM

related to protocol application. This underscores the importance of implementing security measures across all layers of IoT protocol stacks.

2.3 Security Countermeasures

Many academic and industrial studies have been conducted on IoT security mechanisms to counter vulnerabilities and their exploitation, as discussed in the previous section. Security mechanisms typically address one or more elements of the Confidentiality, Integrity, and Availability (CIA) triad. The proposed mechanisms can be software- or hardware-based, or can involve hardware-accelerated software. Security functions are provided by most chip manufacturers for their SoCs, typically using modules that provide similar security functions. ESP32-H2, STM32WL55CC, and CC1352R [19, 37, 39] incorporate hardware security features, such as cryptographic acceleration (AES, SHA, RSA, and TRNG). Furthermore, integrated hardware enables code authentication for a wide range of security services. These include secure boots, secure firmware update mechanisms, memory encryption/decryption, and protocol stack key generation. These ensure integrity and provide important protection mechanisms against unauthorized access. IoT SoC manufacturers commonly provide protection and update mechanisms. However, monitoring and detection mechanisms such as intrusion detection systems are usually not included.

Intrusion and anomaly detection approaches for IoT environments have been proposed in the literature [14, 40, 46]. They detect attacks using a signature list or by learning the legitimate behavior of a system. This solution comprises three main modules: acquisition, analysis, and alertness. Probes, in hardware or software, collect the system metrics during the acquisition part. This information is then analyzed to identify ongoing attacks. If a malicious action is detected, an alert warns the user. These mechanisms are highly accurate in detecting attacks. However, the detection rate performance and overhead, such as silicon area, code size, or power consumption, is still challenging. Utilizing lightweight detection algorithms can assist resource-constrained IoT devices to overcome these limitations. Moreover, relocating data analysis and detection algorithms to a server or a gateway can be beneficial, allowing the end-device to focus solely on collecting metrics.

2.4 Related Work

An HIDS [14][46][9] incorporates a combination of software and hardware probes embedded within the system. The HIDS monitors not only the network activity but also additional hardware and runtime metrics on IoT devices.

The *Physical layer Intrusion Detection System (PHY-IDS)* was proposed in [45]. It is an RSSI-based IDS framework which identifies body movement spoofing attacks on wearable devices. First, the system builds a legitimate behavior model using RSSI time-series data features. This model is then used to spot frames that deviate from the regular wireless signal patterns of legitimate wearable devices. The *PHY-IDS* can be located in a hub, such as a smartwatch or a smartphone. Experimental results demonstrate that *PHY-IDS* has an average detection accuracy of approximately 99.8% for naive attacks. However, comprehensive knowledge and advanced learning capabilities are still required to counter the most sophisticated spoofing attacks. Other research uses MAC layer metrics, such as packet headers, to detect DoS and jamming attacks. In their work, the authors of [21] proposed *Demo*, an IDS framework tailored for a 6LoWPAN-supported IoT environment. However, *Demo* can only be integrated into the gateway and requires additional probes to sniff packets from the IoT devices. Raza et al. [30] focused on analyzing packet headers to detect spoofing attacks in 6LoWPAN networks. This was achieved using *SVELTE*, an HIDS integrated inside a mini-firewall for IoT devices. *SVELTE* employs a hybrid detection method based on both signatures and anomalies, as well as a hybrid placement strategy. With eight nodes per 6LoWPAN network, it achieved a detection rate of approximately 90%. However, as the number of nodes increases, the detection accuracy decreases. For example, with 32 nodes, the detection rate dropped to 70%.

Hessel et al. [14] used metrics from both the MAC and the PHY layers. For example, *Passban* IDS uses anomaly-based detection with RSSI and packet headers to identify malicious packet injections in Linux-based IoT gateway devices. It is capable of detecting various types of malicious traffic, such as port scanning, HTTP and SSH brute-force attacks, as well as SYN flooding. However, the assessment of *Passban* IDS has a considerable impact on the IoT gateway: it incurs a 6% memory overhead, increases the CPU usage by 30%, and reduces the network speed by 7%. This suggests that it is not suitable for embedding in smaller devices. This may be related to the exclusive use of software metrics, while certain hardware metrics may also allow implementing an HIDS that can detect attacks, particularly those that compromise integrity.

In [8], Bourdon et al. proposed an HPC-based anomaly detection system to detect packet injection and DoS attacks. They selected seven HPC registers that reflect the CPU state (memory cache, instructions, exceptions, prediction branches, bus access) to create an application-independent mechanism for detecting malware. A model of legitimate device behavior was generated using machine-learning algorithms and data from HPC registers. Moreover, a hybrid placement strategy was employed for the IDS. A kernel module, acting as a tracer, was also installed on each device. This tracer records time-series data for each HPC register in a local file. This local file is then transmitted to the server using FTP communication to be further analyzed using machine-learning algorithms. The detection rate accuracy was measured using two metrics: true positive rate (TPR) and false positive rate (FPR). Overall, with a maximum of 1% compromised devices, the TPR was approximately 80% and the FPR was less than 1%. However, the detection rate declined when 5% of the total number of devices were compromised, suggesting that identifying intrusions in the case of widespread attacks poses a considerable challenge.

In [10], Cayre introduced a framework for creating embedded detection software for devices using a BLE protocol. This detection module is built on the instrumentation of the MAC layer (Link layer) within the BLE stack. The detection method relies on specific rules applied to network metrics. These rules involve setting threshold values for the Cyclic Redundancy Check (CRC) validity number and the advertising interval. The CRC thresholds are particularly useful for identifying jamming attacks, whereas the advertising interval time helps to detect other types of attacks related to the BLE protocol, including MITM attacks. Several experiments have demonstrated that this method can be used to identify existing BLE attacks. Using the instrumentation of stack code combined with a rule-based approach may allow integrating detection methods directly into the existing stacks. However, this approach presents several challenges. First, rule-based detection methods can sometimes produce higher values for false positive and false negative rates. Attackers may even exploit the known rules to bypass them during more complex attacks. Second, adding extra code for stack instrumentation can lead to performance issues,

including increased execution times and memory costs. These limitations can be particularly problematic for IoT devices with constrained resources. Finally, implementing this approach requires a thorough understanding of the wireless communication protocol stack. For instance, specific knowledge is required regarding where to insert additional functions for the monitoring and analysis modules. Overall, this study introduces an innovative but complex method to enhance the security within the BLE protocol.

Due to the absence of embedded lightweight HIDS in prior research, we present a detailed comparison in Table 2. This comparison highlights how *Diwall* aligns with a selected group of existing HIDS studies [8, 10, 14, 21, 30, 45]. Notably, only [10]offers a rule-based embedded HIDS that utilizes protocol stack instrumentation. However, a comparison with our work is not directly feasible, as our approach involves a hardware implementation and employs different IoT protocols.

For each HIDS, we identify its type and placement, the context including protocols and attacks it can detect, and finally, the metrics (network and hardware) utilized by each solution. The key features of our proposed approach, *Diwall*, are listed for comparison. More details are provided in Section 3.

Table 2. Diwall and Related Works for HIDS in IoT. With RSSI (Received Signal Strength Indicator), adI (Advertising Interval), MITM (Man-in-the-middle), CRC (Cyclic Redundancy Check), HPC (Hardware Performance Counter)

HIDS			**Context**		**Metrics**	
Ref.	**Detection**	**Placement**	**Protocols**	**Attacks**	**Network**	**Hardware**
[45]	Anomaly	Gateway	IEEE 802.15.4	Spoofing	RSSI	-
[21]	Signature	Gateway	6LoWPAN	Flooding, Jamming	Packet	-
[14]	Signature	Gateway	BLE, WiFI	HTTP/SSH brute force, …	Packet	-
[30]	Hyprid	Hybrid	6LoWPAN	Routing, Spoofing,…	Packet	-
[8]	Anomaly	Hybrid	TCP/IP	SSH brute force, DDoS, …	-	HPC
[10]	Rule	Node	BLE	Jamming, MITM	adI, CRC	-
This work	*Anomaly*	*Node*	*LoRaWAN*	*Jamming, Injection*	*RSSI*	*HPC*

On one hand, software-based IDS solutions are commonly implemented in wireless network environments. Their popularity stems from their flexibility and ability to be easily redesigned, making them a preferred choice in many situations. However, this approach has some drawbacks, since implementing additional software can lead to performance overheads. It may also introduce new security risks owing to the potential software vulnerabilities. On the other hand, hardware-based IDS solutions offer advantages such as lower performance overhead with reduced potential for additional security risks. However, hardware-based IDS solutions require higher design times, and thus are not as commonly used as their software-based counterparts.

The majority of the discussed HIDS proposed here utilizes only one metric level, as highlighted in the Metrics column of Table 2. However, attacks and vulnerabilities are present in multiple locations of the IoT protocol stack, including their specification and implementation. As previously discussed in Subsection 2.2, the attack tree is found to be extensive, as it could start by exploiting vulnerabilities from the PHY layer or the MAC layer and conclude by targeting upper layers. In this paper, we present *Diwall*, a hardware-based IDS that leverages existing HPCs on a network processor as hardware probes. A comparative examination of our approach with several notable methodologies found in the literature has been presented in Table 2. *Diwall* approach sets itself apart from the existing systems in several essential aspects. Firstly, it is designed expressly for IoT SoCs with restricted resources, eliminating the need for remote detection through a gateway or a server. Secondly, instead of relying heavily on software-based solutions, such as traditional strategies, our system is primarily implemented in hardware, with only a minimal number of control and configuration instructions in software. Thirdly, our

approach adopts multi-level metrics, both hardware and network metrics to determine the normal operation of the IoT SoC's wireless connectivity, diverging from the common practice of utilizing a single metric in most related studies. Indeed, we utilized built-in CPU HPCs as monitoring probes, thereby avoiding extra area and performance overhead. We also propose an extension to the network processor architecture by incorporating dedicated HPCs designed for the specific purpose of monitoring network metrics. Finally, *Diwall* can detect remote packet injection and jamming attacks that target the network processor of a SoC.

This paper expands upon our prior research [7], which concentrated on a single-level monitoring and detection mechanism. The previous security system only integrated microarchitectural metrics to identify memory corruption attacks. Furthermore, in this work, the HIDS operates at multiple levels, integrating additional metrics derived from network data to identify jamming attacks. The previously proposed HIDS [7], which aimed to detect memory corruption attacks, was initially conceptualized and evaluated using the LoRa PHY layer. This extension involves the physical implementation and assessment of the HIDS for a complete open-source LoRaWAN IoT protocol stack.

3 DIWALL: A HOST-BASED INTRUSION DETECTION SYSTEM

3.1 Threat Model

The inherent vulnerabilities in lower layers of numerous IoT protocol stacks can be exploited by attackers to launch a range of attacks such as jamming and packet injection. A packet injection attack occurs when an unauthorized entity introduces malicious or false packets into a communication channel, aiming to disrupt network functionality or compromise device roles. This technique has been observed across various IoT protocols, where attackers exploit vulnerabilities in lower-layer specifications or implementations. Jamming is a severe threat to IoT devices, where attackers intentionally interfere with wireless signals to disrupt communication. It can be continuous or triggered by specific conditions, such as preamble detection [25]. Protocols like BLE, ZigBee, and LoRaWAN are vulnerable to selective jamming, where specific devices are targeted by decoding their MAC headers [4, 18, 29].

These basic attacks can serve as the starting point for more sophisticated ones, such as DoS or MITM. The victim IoT SoC is composed as shown in Figure 2. It has a wireless connectivity subsystem that supports one or several protocol stacks handled by the network processor. In addition, it has a CPU that oversees the execution of user applications and upper protocol stack layers. The attack surface of the IoT protocol stack mainly includes its PHY and MAC layers because these layers provide the first entry point for the attacker and are most sensitive to vulnerabilities. The remote attacker can perform the attack on the victim's IoT system by using either an SDR platform or a protocol-specific dongle. An attack vector of the attacker includes packet injection and jamming, targeting the victim's wireless connectivity network processor. When conducting such attacks, the attacker attempts to compromise the system.

Our research work focuses on the wireless IoT attacker model, as illustrated in Figure 2. In this figure, a victim IoT end-point initiates communication with the IoT gateway. The victim's wireless connectivity possesses vulnerabilities on its attack surface, encompassing the MAC and PHY layers. Positioned between the communication, the attacker can inject a packet or jam the victim's communication channel. *Diwall* addresses this threat model by monitoring metrics at both the network and microarchitecture levels. This monitoring provides insights into the activity of the network processor and the traffic surrounding the victim.

3.2 Approach

The wireless connectivity of an IoT SoC device can be depicted using the highlighted block diagram in Figure 3a. This representation is divided into hardware and software components. The hardware wireless connectivity

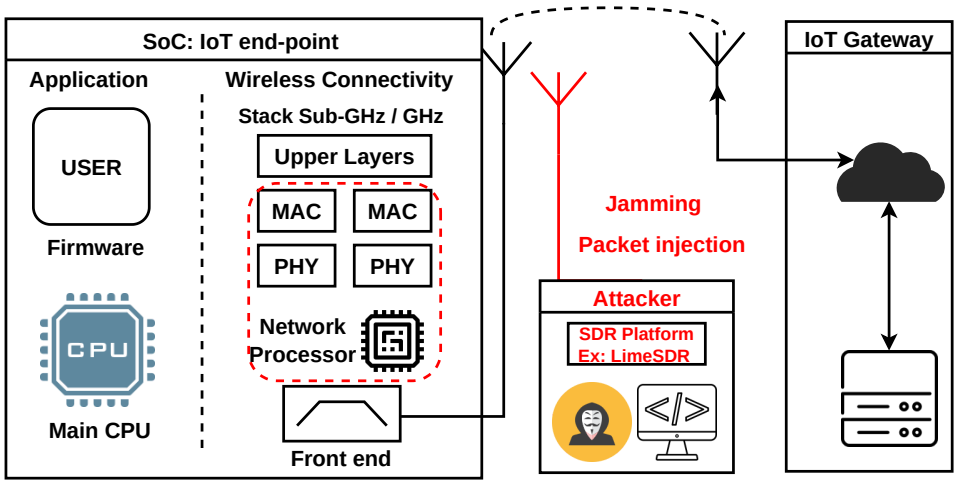


Fig. 2. IoT Wireless Connectivity with the considered threat model. The attack surface is represented by red dashes.

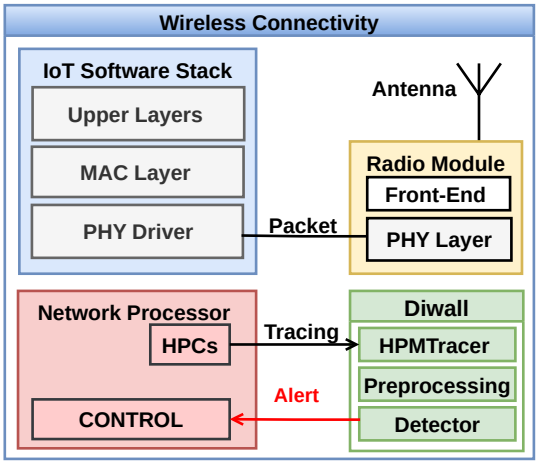
encompasses the radio module, which includes the PHY layer, front-end, and antenna, along with a network processor responsible for managing the radio module and the IoT software stack. The software wireless connectivity comprises layers of the IoT software stack that operate on the network processor. It involves a PHY driver for handling the PHY layer, a MAC layer responsible for processing incoming packets from the PHY layer, and upper layers specific to individual applications for each IoT protocol stack. In the present study, we focus solely on software implementations of the MAC layer, while acknowledging that MAC layers can also be implemented using dedicated hardware.

We propose to secure wireless connectivity with a module named *Diwall* for attack detection. It directly monitors the network processor, performs data analysis, and raises alerts. Figure 3a includes a summarized view of the proposed module.

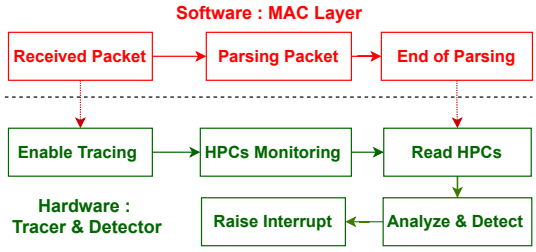
Diwall relies on three main hardware units, coupled to wireless connectivity components. The upper unit in Figure 3a is the HPMtracer, a hardware tracer used to monitor the HPC data of the network processor. The unit below is a preprocessing unit to prepare the data before the detection. The last, the Detector, is a hardware unit used for data analysis and decision-making, based on a prebuilt legitimate model.

Multi-level metrics are targeted, and both network and microarchitecture levels are tracked. The network metrics are stored in a new dedicated HPC that we propose to add to the network processor. Microarchitectural events occur when a network packet is parsed by the MAC layer, and they are monitored by pre-existant HPCs. The proposed module is connected to the network processor via two signal blocks: Tracing and Alert. The Tracing signal establishes a link between HPCs in the network processor and the HPMtracer. This signal includes essential control commands for enabling or disabling the monitoring from software. The Alert signal is used to convey the decisions made by *Diwall*. An alert can be generated by the Detector after the analysis of HPC data. This alert is closely tied to the network processor control and managing interrupts, according to a user-defined security policy. This policy determines the actions to be taken following attack detection.

The processing flow for *Diwall* is shown in Figure 3b. The red top part of the figure represents the network processor software running the MAC layer of the IoT protocol stack and analyzing the incoming packets from the PHY layer. The first row of the green bottom part is run simultaneously with the MAC processing, i.e. *Diwall* hardware monitors the metrics using the HPMtracer. Then, the metrics are processed using a dedicated Detector



(a) Wireless Connectivity and *Diwall* Block Diagram



(b) Flow diagram of Network Packet processing, HPC Monitoring and Detection by *Diwall*

Fig. 3. Diwall Approach

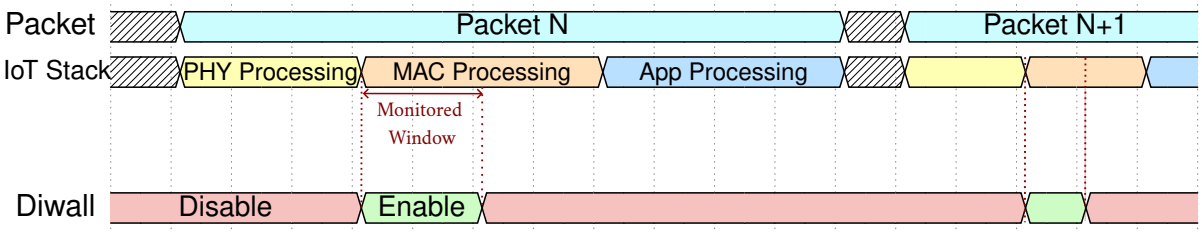


Fig. 4. Example : Diwall Chronogram with IoT Stack

model for identifying specific attacks. The cumulative values of microarchitectural events provide an indicator of the network processor's behavior during packet parsing. The value of the network metrics, such as RSSI, provides an indicator of the network traffic received. The output value is then compared with a threshold, and an alert can be raised.

Figure 4 illustrates a chronology example of the IoT stack activities during network packet handling and *Diwall* operations. A network packet (in light blue) undergoes several stages within an IoT stack. Upon successful reception in the PHY layer (in yellow), the packet is transferred to the MAC Layer (in orange). Here, the packet undergoes MAC processing, which can include cryptographic security and preprocessing of the upper layers. After MAC processing, the application layer (in blue) uses the data extracted from the packet, as indicated in the chronogram. The *Diwall* approach monitors the microarchitectural metrics and RSSI network metadata for a subwindow (in green, entitled "Enable") of the MAC processing window. Its activation is confined to this window, as highlighted in the chronogram. For both packets N and N+1, *Diwall* is exclusively enabled during the monitored window. It is disabled during the other segments of the IoT stack.

3.3 Targeted Metrics

In this section, the two types of metrics used for *Diwall* detection are detailed and discussed. The first metric encompasses microarchitectural events, related to the general activity of the network processor. The second metric uses network metadata, specifically related to the PHY layer.

Microarchitectural events: HPCs offer developers access to detailed low-level information about the processor's microarchitecture and memory access. These resources can serve various usages, e.g., benchmarking, debugging, and security enforcing. Several studies have used the capabilities of HPCs to develop robust security mechanisms, particularly for IDSs [20, 26]. Das et al. [13] identified various challenges, pitfalls, and risks associated with the use of HPCs in security defenses. These challenges are related to issues such as noise measurements, the non-deterministic nature of microarchitectural events, and the sampling method used. Identified issues are particularly present in complex systems that are based on operating systems (OSes). In these systems, multiple applications and processes run concurrently, which can impact HPC event monitoring and lead to over- or under-counting of cumulative values. In addition, adversaries can take advantage of a vulnerable process to intentionally manipulate HPCs. However, in our work, we consider HPCs within the framework of a bare-metal application as we use a network processor. In this context, while noise may still originate from peripherals like hardware interrupts or timers, the running application has more control over the hardware, without OS interference. This level of control serves to mitigate issues arising from noise sources during the execution process. In our case, we use a network processor dedicated to processing network packets. It has fewer peripherals than an application processor, and interrupts occur only upon packet reception, which is infrequent, occurring just a few times per day in our context. This infrequency, coupled with the IDS's brief execution time of fewer than 10 CPU clock cycles, minimizes any potential noise. Furthermore, instead of sampling, we employ polling. This alternative method involves instrumenting the code within the software and reading the HPCs at the end of the monitored window.

Network metadata: Metadata are available in the lower layers of IoT protocol stacks, mainly at the PHY layer and the MAC Layer. These metadata have been used for performance monitoring and wireless communication management. Various PHY layer metadata have also been used for security mechanisms [15, 44]. In our approach, we chose to study the RSSI metric, which is frequently employed in wireless communications. It provides information regarding the strength and quality of a signal during transmission or reception. Our decision to incorporate this metric aims to ensure that *Diwall* remains independent of any specific IoT protocol stack. Given the fact that the RSSI feature is already embedded in many receivers for the most common IoT protocols, it improves the scalability and flexibility of our approach, enabling compatibility with various protocols.

3.4 Architecture

This section aims to present the implementation of *Diwall*. It was implemented on an Arty A7 100T FPGA board, along with a RISC-V CPU network processor. We extended the RISC-V CV32E40P [27] pipeline as shown in Figure 5. This figure represents a new variant of CV32E40P, now used as a network processor integrating our proposed module. In this architecture, *Diwall*'s signals and data are connected to the CV32E40P's Control Status Register (CSR) to monitor microarchitectural events and network metrics. Once it analyzes data metrics and detects malicious behavior, it raises an alert signal, which is propagated to the CV32E40P's interrupt controller. The proposed approach for detecting wireless attacks could be integrated as a small hardware component within the CPU architecture, similar to an FPU or a TRNG. This choice offers quicker access to the HPCs and reduces potential performance impact. Figure 5 shows the architectural details of *Diwall* components. It comprises the same hardware components presented in Section 3 in Figure 3a: HPMtracer, Preprocessing, and Detector units.

All highlighted units were designed using a finite state machine (FSM). Each unit in *Diwall* has independent FSM states and controls. Only enable, disable, and data signals are propagated between the units.

The HPMtracer tracks the HPCs and sends them to preprocessing, where the exponentially weighted moving average (EWMA) of the RSSI network metric is calculated. The microarchitectural event data are directly transferred without any preprocessing. These data are then sent to the Detector unit for comparison with the thresholds. Once the Detector finishes its analysis, it activates the end signal to reset the values stored in the HPMtracer and puts our HIDS in an idle state.

More information about each hardware unit and *Diwall* configuration is provided in the following.

HPMtracer: Hardware Tracer. The primary hardware component of *Diwall* is the HPMtracer. This hardware tracer monitors data from the network processor. Its operations are managed by the software running on the network processor. The CV32E40P network processor is equipped with HPCs tailored to monitor the microarchitecture and network data. Incidentally, it incorporates CSR into its execution stage pipeline. This CSR houses HPCs such as `HPC1` and `HPC2`, which track selected microarchitectural events, as determined by our Decision Tree model presented in the following sections. These events can be configured and adjusted by users using the software. In addition, we configure a dedicated Network HPC named `NwHPC`, a 64-bit register, divided into two 32-bit parts, serving as registers to monitor IoT stack metadata.

CSR also features signals essential for controlling the HPMtracer, represented as `csr_data` and `csr_addr`. *Diwall* is activated and deactivated when `csr_addr` is set to `0x320` and `csr_data` has values of `0x0` and `0xF`. This action consists of writing into a dedicated register called `mcountinhibit`, which controls the enabling or disabling of HPCs. If the enabling condition is met, the HPMtracer forwards the HPCs' current values to the subsequent stage for analysis and decision-making. In this architecture, the present values of the HPCs are denoted as `HPC1_v`, `HPC2_v` and `NwHPC_v` with `v` corresponding to the value.

Preprocessing. The preprocessing unit prepares the data for further analysis. The goal here is to sanitize the data before it is analyzed by the Detector. In our research, we specifically analyzed and implemented EWMA preprocessing for RSSI metadata in the LoRa and LoRaWAN protocol stack. If required, further preprocessing methods for the metrics would also be incorporated into this block.

In this scenario, preprocessing receives three pieces of data from the HPMtracer: `HPC1_v`, `HPC2_v`, and `NwHPC_v`. `NwHPC_v` represents the RSSI value coded on 32 bits. `HPC1_v` and `HPC2_v` are the cumulative values of the selected microarchitectural events. They are used directly in our Detector, and simply go through the unit. In *Diwall*,

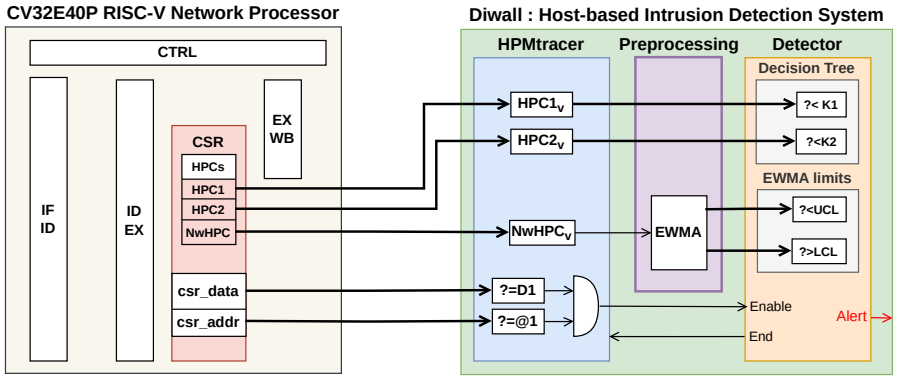


Fig. 5. Block diagram illustrating the architecture of Diwall incorporating CV32E40P Processor

the Detector works on the EWMA of the RSSI value provided in NwHPC_v. This function 1 has been tweaked to reduce the overhead. EWMA involves multiplications, and the direct implementation of multipliers in hardware would increase the area overhead for *Diwall*. Thus, we replaced multiplications by right-shifting the binary representation. Indeed, multiplying the RSSI by a parameter such as 0.25 is equivalent to dividing by 4, which can be implemented in hardware by right-shifting the RSSI binary representation by two bits. The EWMA hardware is implemented with an FSM consisting of three states: an IDLE state, a state for calculating the necessary right-shifting values for the equation and updating the output (the EWMA of the RSSI), and a state to send data to the Detector.

Detector: Decision Tree and EWMA Control Limits. The second module, the Detector, is responsible for data analysis and decision-making based on the data monitored by the HPMtracer and handled by the preprocessing unit. The Detector comprises two modules: (1) a Decision Tree model for detecting packet injection attacks and (2) an EWMA control limits for jamming attacks detection.

A Decision Tree model tailored to detect memory corruption based on packet injection attacks is generated offline. This model is based on the monitored values from HPC1 and HPC2, and two thresholds, named K1 and K2. These thresholds represent the maximum cumulative microarchitectural events considered legitimate within a monitored window in the MAC processing of an IoT protocol stack. The monitored window refers to a section of the program, rather than the entire MAC processing software. In our study on the LoRa and the LoRaWAN stack, we monitor the section of code responsible for receiving and parsing a frame. If a frame is received, *Diwall* starts monitoring these microarchitectural events and compares them with the thresholds. The Detector module issues an alert when the data exceeds the thresholds set by the Decision Tree model. If the network processor exerts significant effort during frame parsing, it is reflected in the values of HPC1 and HPC2. In our findings, buffer overflow in the stack and heap leads to higher HPC1 and HPC2 values, deviating from the model defined by K1 and K2.

Another module within our Detector, designed to detect jamming attacks, exploits the EWMA of the RSSI. The calculated EWMA of the NwHPC_v value is compared with predefined control limits, denoted as Upper Control Limit (UCL) and Lower Control Limit (LCL). If the network metadata drifts outside these predefined limits, the Detector module raises an alert.

The combination of the three units, the HPMtracer, Preprocessing and Detector modules, in the *Diwall* architecture allows efficient detection of both jamming and packet injection attacks. This is achieved within a brief time frame, taking less than 10 clock cycles to operate the entire architecture. This efficiency stems from our method of reading HPCs only at the end of the monitoring phase. Moreover, *Diwall* does not utilize local counters. Instead, it relies on registers to temporarily store data for a few cycles, thus preventing preemption of the network processing.

Diwall Configuration: To enhance the flexibility of the *Diwall* approach, we enable parameter configuration from the software. HPC1, HPC2, and NwHPC are configured using software running on the network processor. The configuration of HPC1 and HPC2 is achieved using dedicated software functions, which are implemented in the assembly code of the RISC-V processor. These functions write *Diwall* parameters directly to the mapped CSR registers. They are provided in a software library called *LibDiwall*. In the CV32E40P architecture, HPC1 and HPC2 are represented by mhpmcounterX, where X represents the counter ID, ranging from 3 to 31. These counters are mapped to the CSR addresses from 0xB03 to 0xB1F. Specifically, we use mhpmcounter3 and mhpmcounter4 to monitor HPC1 and HPC2, with addresses 0xB03 and 0xB04.

Assigning microarchitectural events to mhpmcounterX is accomplished through software using the event selector CSR, which is denoted as mhpmeventX. Each event selector corresponds to a counter ID, with X ranging from 3 to 31. These event selectors are located within the address range of 0x323 to 0x33F. Microarchitectural events are identified with an ID from 0 to 15, and we associate this ID with the desired mhpmeventX, which, in turn,

assigns the event to mhpmcounterX. For example, if mhpmevent3 is linked to the ID of LD_STALL (represented by ID 2), setting the second bit in mhpmevent3 to 1 results in mhpmevent3 being set to 0x4, which will affect LD_STALL to mhpmcounter3. The relationship between mhpmeventX and mhpmcounterX is important. If mhpmevent3 is set to 1, it implies that it will be counted by the HPC mhpmcounter3. Both mhpmeventX and mhpmcounterX are mapped in the CSR of the RISC-V architecture and can be accessed for read and write operations using the CSR instructions provided: csrr for read and csrw for write. We utilize the csrw instruction to select the desired counter ID by setting the address of the event selector to the microarchitectural event ID. This instruction also allows us to reset the counters to 0. During the training phase, we used the csrr instruction to read the HPC values for dataset generation.

Following the HPCs configuration, a dedicated register for enabling and disabling the HPCs is mcountinhibit, located at CSR address 0x320. Each HPC can be individually enabled or disabled by modifying the corresponding bit in the mcountinhibit address. For instance, to enable mhpmcounter3, the bit 3 is set to 0; to disable it, the bit 3 is set to 1. To monitor the network metadata using NwHPC, we used mhpmcounter5. The configuration is specified for the RSSI and declared in the software as a 32-bit value at address 0xB05. We reset the counter using the csrw write instruction to set this register to 0, and then write the received RSSI value to the same address to track it using *Diwall*.

K1, K2, LCL, and UCL are the parameters of *Diwall*'s Detector, representing the generated model of the Decision Tree and the EWMA control limit thresholds. In this version of *Diwall*, these parameters are configured directly in the HDL code, which, on the FPGA, requires new bitstream generation. To increase flexibility, several approaches are expected to allow software configurations for K1, K2, LCL, and UCL. The first configuration uses SoC-CSR-dedicated registers. The SoC includes its own external CSR, which follows the same methodology as the RISC-V network processor for configurable HPCs. Each register in the SoC has its address and data accessible through the provided software instructions. Memory-mapped registers on the SoC's CSR can be associated with each *Diwall* parameter, enabling user configuration and updates to the Detector's models. In the LiteX framework, functions such as csr_write_simple(reg_data, reg_@) and reg_data = csr_read_simple(reg_@) are provided to manage memory-mapped hardware registers. These functions can be used to configure and update *Diwall* parameters.

However, an alternative and more efficient option for configuration is to use dedicated registers implemented within the network processor's CSR. The RISC-V CSR includes additional registers, dedicated to *Diwall*, containing its parameters. Although not all CSR registers are implemented in CV32E40P, they can be extended to include the necessary registers for the configuration. The RISC-V ISA already provides instructions for CSR handling in software. Thus, added dedicated configuration registers do not require ISA extensions ; they use the provided CSR instructions for configuration, specifically, csrr and csrw.

The proposed approaches of using CSR from the SoC or the network processor in configuration would allow dynamic adjustments to *Diwall* thresholds and settings directly from the software, eliminating the need for bitstream regeneration and significantly improving the system's adaptability. This extension is left for future work.

4 EXPERIMENT AND METHODOLOGY

In this section, we describe our experimental setup and the methodology to build datasets to configure the IDS. To simplify the explanation and description of our method, we have taken only one MAC layer as an example, which we will refer to as the *simplified MAC layer*.

4.1 Experimental Environment and Evaluation Setup

Our testbed comprises three LoRa IoT devices, as illustrated in Figure 6: IoT nodes 1 and 2 for establishing a LoRa network and an attacker responsible for executing wireless attacks. This configuration ensures a comprehensive assessment of the proficiency of *Diwall* in detecting wireless attacks in a representative communication setting.

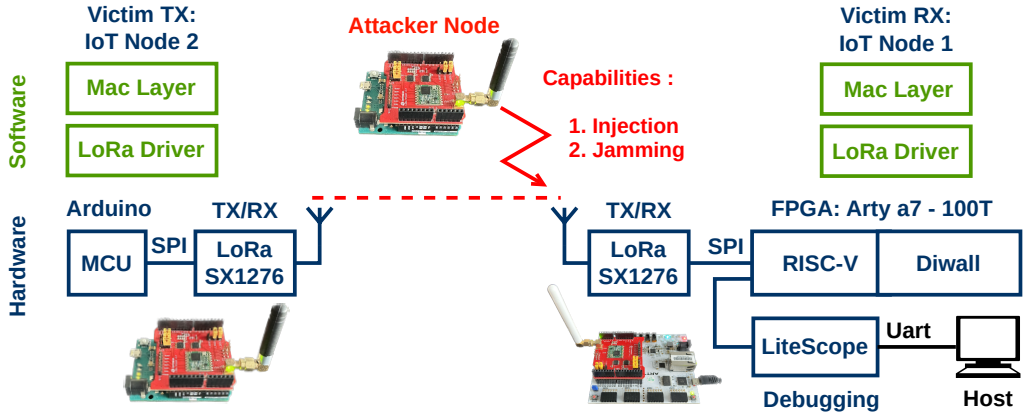


Fig. 6. Evaluation of *Diwall* on FPGA Arty A7 100T Board within LoRa Testbed

IoT node 1 (Victim): This custom-built IoT end-device acts as a potential victim. It integrates the necessary hardware for both the LoRa PHY layer and the simplified MAC layer, which is further tailored for compatibility with the proposed *Diwall*. The simplified MAC layer parses LoRa frames and stores them in a reception buffer. It is instrumented with *Diwall* instructions for configuration and control. The configuration includes selecting the microarchitectural events `LD_STALL` and `BRANCH_TAKEN`, assigned to `HPC1` and `HPC2`, and RSSI to `NwHPC`. A *Diwall*-enabled instruction is placed at the beginning of parsing LoRa frames, and a disable instruction is placed at the end of parsing. For every received packet, *Diwall* verifies the data value of the microarchitectural events and the RSSI value in the hardware. These values are compared with the parameter thresholds established by an embedded Decision Tree model and EWMA control limits, as detailed in Subsection 4.3. An alert is issued if the frame conflicts with the established security policy. To collect results on detection rates, a debugger called "LiteScope" gathers details within registers and counters inside *Diwall* and the RISC-V core. A dedicated counter is placed inside *Diwall*'s Detector unit, counting the alerts raised. "LiteScope" facilitates reading this counter, which logs the total number of alerts triggered. We used a MAC layer packet parser implemented in C. It represents a simplified MAC layer for the IoT protocol stack. To monitor the network-packet parsing window, *Diwall* configuration and control instructions were integrated in software. We instrumented the Reception Function as represented in Algorithm 1. *Diwall* is controlled using three signals: HPM_Reset, HPM_Enable, and HPM_Stop. With this instrumentation, HPMtracer tracks microarchitectural events and RSSI metadata while storing the payload in the MAC reception buffer. During this evaluation, extra IoT node 2 is used to establish a LoRa communication link with the victim (IoT node 1).

Attacker. Located between the two IoT nodes (IoT nodes 1 and 2), the attacker has the ability to inject and jam signals. We use an ARM-based Arduino board and a Dragino LoRa Shield, which utilizes an SX1276 LoRa radio module. To manage the LoRa radio module, we use RadioLib, a versatile wireless communication library for Arduino. The programs provided by the RadioLib library have been customized to inject substantial network

Algorithm 1 Reception Function in simplified MAC Layer

```
1: procedure ReceptionFunction(payload)          ▷ Triggered by radio module interrupt
2:     ...
3:     Call HPM_Reset()                          ▷ Reset HPCs to 0
4:     Call HPM_Enable()                         ▷ Select/Enable HPCs and enable Diwall
5:     Recepetion_Buffer ← payload               ▷ Getting MAC payload
6:     Metadata ← metadata                       ▷ Getting Metadata value such as RSSI
7:     Call HPM_Stop()                           ▷ Stop HPCs, read and analyze by Diwall
8:     ...
9: end procedure
```

traffic, including a specified range of packet sizes, and to activate both jamming modes: triggered jamming and continuous jamming.

Evaluation of Packet Injection Attack. In the packet injection scenario, IoT node 1 establishes a typical communication link by sending legitimate traffic to IoT node 2. Meanwhile, an attacker positioned between the two nodes competes with IoT node 2 and injects oversized malicious packets into IoT node 1. While IoT node 1 is equipped with a 10-byte reception buffer, it lacks a software-based size check for the incoming frames. The attacker capitalizes on this by sending packets that, although in line with the MAC layer protocol, exceed this 10-byte buffer, aiming to disrupt IoT node 1's operation. To detect this, *Diwall* system continuously monitors microarchitectural data for incoming frames, raising an alert for any packet that breaches the buffer limit.

For this evaluation, we have sent 4×10^5 packets. The interval between consecutive packets is determined by the LoRa parameter known as the spreading factor (SF). LoRa devices utilize a range of SF values to balance data rate and communication range, typically ranging from 7 to 12. Higher SF values extend the range at the expense of data rate, while lower values provide higher data rates but shorter ranges. The SF in the LoRa PHY layer affects the Time on Air (ToA): higher SF values result in a longer ToA, while lower SF values lead to a shorter ToA. The LoRa PHY layer was configured with an SF7 value. An interval of approximately 100 ms between each consecutive packet and the central frequency was within the 868.1 MHz LoRa channel. Selecting SF7 was aimed at reducing the time between packets. However, opting for a different SF does not affect the packet injection study.

Evaluation of Jamming Attack. In this study, an attacker transmits random frames, potentially aligning them with the buffer size of the victim's receiver. The primary goal of a jammer is to occupy the victim's channel.

Our testbed consists of three LoRa devices:

(1) A transmitter (TX victim) and a receiver (RX victim) are positioned within 10 $meters$ of each other.
(2) A jammer (attacker) located close to the RX device is approximately $1 - 2$ $meters$ distant.

We investigate two jamming methodologies:

(1) **Continuous Jamming**: Here, the attacker persistently transmits random frames, aiming to disrupt IoT node 2 during the communication with the victim.
(2) **Trigger Jamming**: In this approach, the attacker eavesdrops on the channel, springing into action upon the detection of a preamble from IoT node 2. Upon such detection, the channel is interfered with.

Even with channel interference, the victim remains capable of receiving frames and discerning the RSSI value of the frame. *Diwall* verifies the RSSI value of each acquired network packet. It calculates the EWMA and compares it with the defined EWMA control limits. If a packet's RSSI diverges outside the EWMA limits, an alert is triggered, indicating a potential jammer proximity. For both jamming strategies, *Diwall* detection efficacy was gauged over

4, 000 frames. Our LoRa trials use a central frequency of about 868.3 MHz, and employ an SF12. Choosing SF12 provides an ideal ToA for jammers to succeed in their attack.

4.2 Detection of Packet injection Attacks

For a more in-depth understanding of the relationship between the microarchitectural features and the behavior of the network processor during network-packet parsing, we use supervised machine learning classification algorithms. In this study, microarchitectural features are grouped into three network packet classes (legitimate, heap overflow and stack overflow).

Our primary research objective was to identify ongoing packet injection attacks using microarchitectural metrics. Certain microarchitectural features are more pertinent for detection using machine learning classifiers. For this purpose, a supervised learning method that employs training samples to construct decisions within a tree model was considered.

For this study on detecting packet injection attacks using microarchitectural metrics, a dataset was generated by simulating the experimental setup detailed in Subsection 4.1. Buffer overflow exploits, as summarized in Table 3, are applied to the monitored window in the simplified MAC layer presented in Algorithm 1. Through simulation, a total of 3,000,000 packets were parsed by the simplified MAC layer, while simultaneously monitoring microarchitectural events from the network processor.

Table 3. Attacks Scenarios: The buffer size is 10 or 23 bytes. Larger Packets result in a Buffer Overflow.

Attack Scenarios			**Reception Buffer Size**	
Packet Type	**Traffic size**	**Packet length**	**Stack**	**Heap**
S0: Legitimate	$1,000,000$	$5-10$ $bytes$	10 $bytes$	10 $bytes$
S1: Stack Overflow	$1,000,000$	$13-23$ $bytes$	10 $bytes$	23 $bytes$
S2: Heap Overflow	$1,000,000$	$13-23$ $bytes$	23 $bytes$	10 $bytes$

We employed Scikit-Learn, a widely used open-source machine learning library in Python, to train and evaluate multiple classifiers on our labeled dataset of 11 microarchitectural features. Each classifier's performance was measured using standard metrics such as accuracy, precision, recall, and F1-Score. Initial results indicated that several algorithms consistently achieved near-optimal classification, with approximately 99% accuracy across all metrics. Given the resource limitations of IoT end-devices, we selected the Decision Tree classifier for its simplicity and ease of implementation in hardware.

To reduce the area overhead within the network processor, only two HPCs from the microarchitecture are used. Figure 7 illustrates the comparative behavior of the two features, `BRANCH_TAKEN` and `LD_STALL` selected by the Decision Tree. These HPCs measure the number of branch instructions and the number of delayed load instructions.

Buffer overflow results in the alteration of the CPU behavior to include unexpected branches and large delays in data retrieval from memory. The Decision Tree selects the increased `BRANCH_TAKEN` and `LD_STALL` counter values from the 10 microarchitectural metrics to identify potential attacks against the MAC layer of a protocol stack. Figure 8 illustrates the Decision Tree model block diagram produced using our dataset. The model splits the generated dataset from Figure 7 into three classes: legitimate, stack_overflow, and heap_overflow. Indeed, during the learning phase, the `BRANCH_TAKEN` and `LD_STALL` HPC values demonstrated a high capacity to detect attacks. These values are used for decision-making based on thresholds `K1` and `K2` determined from the training data: `BRANCH_TAKEN` < 65.5 and `LD_STALL` < 14. The values are directly related to the code used to parse the received network packets and to store the received data in the buffers. The value of K2=65.5 reflects the separation

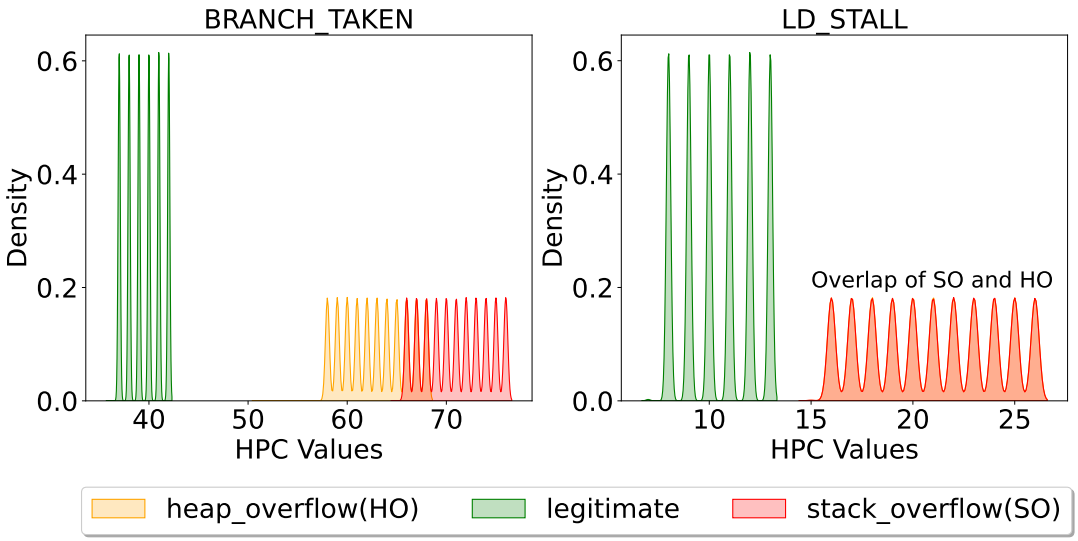


Fig. 7. Selected Microarchitectural Metrics for Decision Tree Model

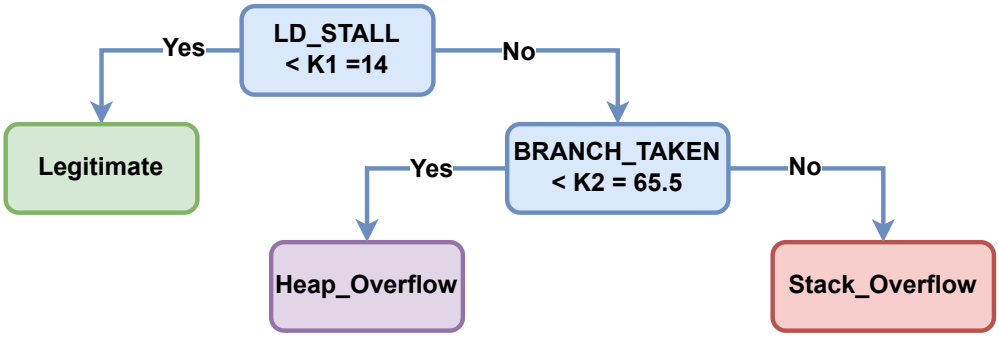


Fig. 8. Generated Decision Tree Classifier Model

between the heap overflow and stack overflow classes. As shown in Figure 7, there is an overlap in HPCs values between these two classes. To address this, we adopt a binary classification approach (attack vs. no attack), which reduces false alarms in scenarios involving multi-class attacks. In our system, alerts are based directly on the legitimate class, as the selected decision tree is lightweight with a limited depth, designed to minimize overhead.

There are several options for Decision Tree implementation for IoT end-devices. These options include hardware, software, and a combination of both. A hardware design can be developed for our system using an FSM and hardware accelerators written in hardware description language. Software implementations can be crafted using the C programming language. Decision Tree models can be efficiently implemented in FPGA with limited hardware and can achieve satisfactory classification speed. As a result, we created a hardware-based implementation.

4.3 Detection of Jamming attacks

Several strategies for countering jamming attacks have been proposed in the literature, demonstrating their effectiveness in detecting such attacks in wireless sensor networks and IoT architectures [32, 42]. Many of these strategies use metadata from the PHY and MAC layers as features for jamming countermeasures [31, 38]. Ruotsalainen et al. [32] highlighted numerous such mechanisms. These include radio fingerprinting, an authentication technique that identifies devices by analyzing the characteristics of the received signals, and a novel technique of key generation based on RSSI values, which extracts secret keys from these values. Machine learning and deep learning techniques are frequently incorporated into these mechanisms, owing to their superior performance in recognition and classification tasks. In [42], Testi et al. used a neural-network-based machine learning approach to propose a novel jamming detection and classification algorithm. This approach employs features such as Packet Delivery Rate (PDR), SNR mean and variance, SNR Power Spectral Density (PSD), and cross-correlation. However, although the use of deep learning techniques, such as neural networks, in jamming detection provides high accuracy and effective feature performance, it also requires significant computational resources. This approach is more suited to the upper levels of IoT architecture, such as gateways or servers, where voluminous resources are available, and all features are accessible.

EWMA, a statistical method, is a valuable tool for detecting small deviations in statistical data, and it has a relatively low computational complexity. Its effectiveness in identifying jamming attacks while requiring a minimal number of metadata features has been proven. Compared to neural-network-based methods, this methodology requires significantly fewer computational resources. In their work, Osanaiye et al. [28] proposed deploying EWMA on the cluster head to identify attacks on member nodes and on the base stations to detect attacks on the cluster heads. The implemented EWMA is capable of detecting anomalous changes in the intensity of a jamming attack event by utilizing the packet inter-arrival feature of the packets received from the sensor nodes. In another study [23], Bolivar proposed the use of EWMA for jamming detection in LoRaWAN networks. He employed RSSI and packet inter-arrival time (IAT) as datasets for model training and evaluation using both large-scale simulation datasets and small-scale real-world datasets. He compared his approach with a Recurrent Neural Network (RNN) machine learning model, demonstrating higher detection rates. He reported a true positive rate of approximately 90% for the statistical model and 98% for the neural network machine learning model. This demonstrates that although the EWMA method has a slightly lower accuracy, it offers a competitive alternative with less computational complexity.

The EWMA formula is given in Equation 1 with λ the smoothing constant determining the depth of the EWMA.

$$\mathrm{EWMA}_0 = \mathrm{RSSI}_0 \; \mathrm{EWMA}_n = \lambda \mathrm{RSSI}_n + (1 - \lambda)\mathrm{EWMA}_{n-1} \tag{1}$$

Our goal was to focus on resource-limited IoT devices. Unlike previous works that have implemented EWMA in software for gateways, we chose a hardware-based implementation of EWMA. This strategy aims to identify acceptable RSSI values for the network metric, with any values beyond this indicating a statistical anomaly. An anomaly is identified as a significant deviation from normal RSSI behavior, detected using EWMA. Since an attacker must be in proximity for jamming to be effective, this results in higher RSSI values. During periods of triggered and continuous jamming, the EWMA of RSSI value increased significantly, as illustrated in Figure 9. This increase indicated a jamming attack. Normal RSSI behavior is characterized by values staying within predefined thresholds based on previous data from non-attack conditions, reflecting packets received from a legitimate gateway. While simply applying thresholds to RSSI values may result in numerous false positives due to natural variations in the communication channel, incorporating EWMA improves the accuracy of anomaly detection.

We customized our network processor (CV32E40P) to include an `NwHPC` that monitored the RSSI of each incoming packet. Within *Diwall*, we implemented the EWMA algorithm in a hardware block to analyze the RSSI values and make decisions. For the hardware EWMA, we assigned a value of $\lambda = 0.25$. Previous research [28] has suggested that this value should be between 0.2 and 0.5. To detect jamming attacks, we conducted a profiling

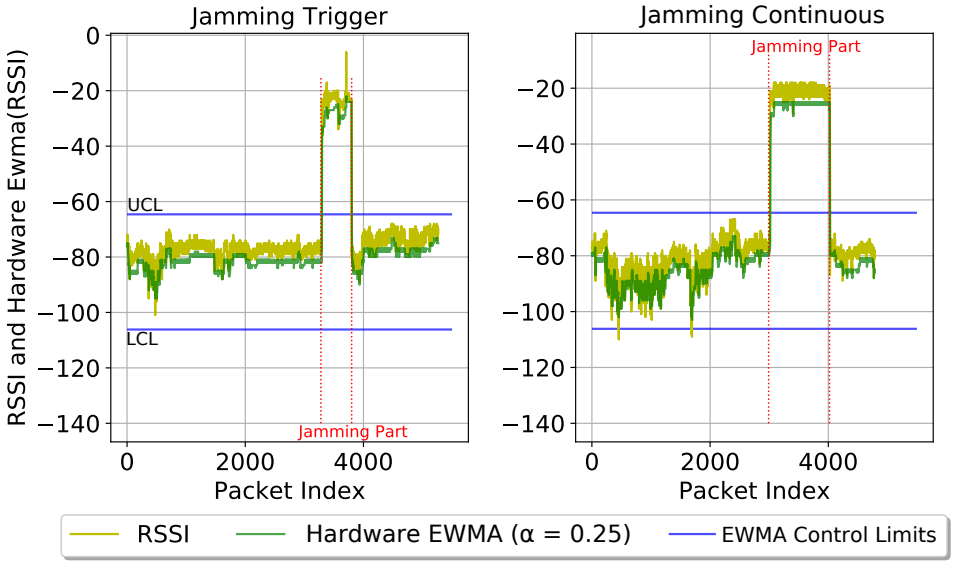


Fig. 9. RSSI and Hardware based EWMA during Legitimate Traffic and Jamming Attacks

phase to capture the legitimate behavior of RSSI dataset in a controlled, attack-free environment between the gateway and an IoT node. During this phase, we calculated the EWMA control limits and identified the UCL ($UCL = -65$) and LCL ($LCL = -106$) thresholds. These thresholds represent the legitimate upper and lower limits of the EWMA values. The LCL and UCL values were calculated as:

$$\text{UCL} = EWMA_0 + f \cdot \sigma_{ewma} \; \text{LCL} = EWMA_0 - f \cdot \sigma_{ewma} \tag{2}$$

where EWMA_0 is the target value, f is a tuning value usually set at 3-sigma control limits and σ_{ewma} is the standard deviation of historical EWMA values estimated as:

$$\sigma^2_{ewma} = \sigma^2_{rssi} \cdot \left(\frac{\lambda}{2 - \lambda} \right) \tag{3}$$

where σ_{rssi} is the standard deviation of historical RSSI values.

EWMA, a statistical technique, efficiently detects small shifts in time-series data owing to its low complexity and requires updates only for newly observed data. It effectively combines current and historical data, facilitating the quick detection of small shifts. The computational complexity of EWMA is minimal, requiring only a single update per packet based on the RSSI, which ensures low overhead.

Setting the value $\lambda = 0.25$ in Equation 1 simplifies the implementation of EWMA. This is because multiplying by 0.25 is equivalent to dividing the value by 4, which can be approximated by right-shifting the binary representation of the RSSI by 2 bits. The same strategy can be applied to previous $(1 - \lambda)\text{EWMA}_{n-1}$ values. For instance, multiplying by 0.75 can be represented by adding 0.25 and 0.5, which can be approximated by right-shifting by 2 and 4 bits, respectively.

5 RESULTS AND DISCUSSION

In this section, we detail the experimental results of *Diwall*'s effectiveness in a full IoT setup, including detection accuracy, FPGA resource usage, and performance relative to a baseline network processor. Additionally, a discussion of the proposed module results and limits is provided.

5.1 *Diwall* Detection Rate

To evaluate and validate the detection effectiveness of *Diwall*'s model parameters generated in Subsections 4.2 and 4.3, we conducted a test campaign using the physical testbed depicted in Figure 6. The environmental conditions for assessing jamming and packet injection are outlined in Subsection 4.1. In this evaluation, several performance metrics are utilized, including accuracy (ACC), which measures the correctness of predictions by calculating the ratio of correct predictions to total predictions. It is defined as:

$$\text{Accuracy (ACC)} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4}$$

where True Positives (TP) refer to instances where malicious packets are correctly identified, while True Negatives (TN) represent the correct identification of legitimate packets. False Positives (FP) occur when the system incorrectly flags legitimate packets as malicious, leading to false alerts. Conversely, False Negatives (FN) arise when malicious packets go undetected, being incorrectly classified as legitimate. False Negative Rate (FNR) and False Positive Rate (FPR) are also calculated to provide insights into the model's strengths and weaknesses.

Table 4 summarizes the test campaign and performance metrics related to the achieved detection rate, where our proposed module is integrated into a simplified MAC layer and in a LoRaWAN network.

Table 4. Evaluation of *Diwall* Detection Rates for Packet Injection and jamming Attacks in LoRa and LoRaWAN networks. With PI (Packet Injection), JT (Jamming Trigger), JC (Jamming Continuous), and PLR (Packet Loss Rate).

MAC Layer	Attacks	FP	FN	TP	TN	FNR	FPR	ACC	PLR
Simplified MAC	**PI**	53	13	193,327	195,704	0.007%	0.027%	99.98%	2.72%
	JT	0	1	402	1000	0.25%	0%	99.92%	59.7%
	JC	0	1	413	1000	0.24%	0%	99.92%	58.7%
LoRaWAN	**PI**	0	1	5589	5630	0.017%	0%	99.99%	N/A
	JT + JC	0	2	6335	5520	0.031%	0%	99.98%	N/A

LoRa and Simplified MAC Layer. In this evaluation, we assessed the generated Detector's models. For the Decision Tree, the parameters are `HPC1=BRANCH_TAKEN` with $< K1 = 65.5$ and `HPC2=LD_STALL` with $< K2 = 14$. The EWMA control limits are defined with UCL ($UCL = -65$) and LCL ($LCL = -106$). During a packet injection attack, the network traffic comprises 200, 000 benign packets and an equal number of packets subjected to stack and heap buffer overflow. During the experiment, 400, 000 packets were sent and 389, 097 were successfully received, resulting in a packet loss rate (PLR) of approximately 2.72% as reported in the last column of Table 4. This PLR was deemed acceptable for LoRa networks. The reduced PLR can be attributed to the indoor location of the testbed. The detection accuracy for packet injection was significant at 99.98%. These results are obtained with numbers of FP of approximately 53 (0.027%) and FN of approximately 13 (0.007%), both of which are notably low. This high accuracy is backed by the distinct behavior of microarchitectural HPCs, enabling effective differentiation between legitimate packets and those affected by buffer overflow. Complete 100% accuracy was not realized with our Decision Tree model. This was owing to our testing under the most challenging scenario, where the sizes of benign and malicious packets were very similar, being 10 and 13 bytes, respectively.

For the jamming attack assessment, we examined both the triggered and continuous detection rates. To create an ideal environment for the jammer, we used a long interval of approximately 1 second between transmitted packets. The LoRa transmitter is positioned 8 meters from the victim, with the jammer situated close to the victim. In the experiment, the LoRa transmitter dispatches approximately 2, 000 legitimate packets to a victim. Concurrently, the jammer sends an equivalent traffic volume split between the two jamming types. The observed PLR is 59%, which is typical for jamming scenarios. This high PLR occurs because most data packets are lost or corrupted owing to interference caused by the jamming signal. The combined detection rate achieved by *Diwall* for both jamming methods is approximately 99.92%. This accuracy is accompanied by the absence of FP and negligible FN of approximately 1 (0.24%). While our evaluation is based on received packets, implementing a security policy on *Diwall* to verify the RSSI value even if there are no received packets could potentially elevate the detection rate to 100%.

LoRa and LoRaWAN. After successfully studying a simplified MAC layer and achieving promising detection rates, we extended the *Diwall* methodology to the LoRaWAN protocol using a LoRaMac-node stack [35]. In this integration, we targeted a code section of parsing LoRaWAN packets in periodic uplink application that was instrumented with *libDiwall* functions. After training, new parameters were generated for the Decision Tree model, with `K1=59` and `K2=595`, where `HPC1` represents `LD_STALL` and `HPC2` represents `INSTR`, indicating the number of instructions executed by the network processor during the parsing of the LoRaWAN MAC packet. The EWMA control limits remained unchanged with UCL ($UCL = -65$) and LCL ($LCL = -106$), as we utilized the same physical layer and the same distances as previously between devices. During the evaluation, some modifications were applied to the LoRa PHY layer of the victim device to facilitate successful injection and jamming. The reception period continuously on SF7 maintained open for consistent frame reception. Only the first channel, 868.1 MHz, of the LoRaWAN stack was employed to intensify our jamming efforts and enhance the probability of successful attacks. The last two rows of Table 4 provides a summary of key metrics, including TP, TN, FP, FN, and accuracy rates, obtained during packet injection and jamming attacks on the LoRaWAN network. Our observations for both packet injection and jamming attacks revealed detection accuracy rates that approached 99.98%. This high accuracy is notable because of the absence of FP and a minimal FN of approximately 1 (0.017%) to 2 (0.031%); every legitimate frame is recognized as a true negative, and the majority of alerts are raised. This demonstrates that *Diwall* does not generate false alerts during normal end-device operations or adherence to its policy. Of the 23, 000 frames analyzed during the experiment, only three malicious network packets were incorrectly identified as false negatives. This result further highlights the efficiency of the proposed approach. This indicates that *Diwall* successfully identifies jamming and packet injection attacks in LoRaWAN networks, while maintaining a minimal rate of false negatives.

5.2 FPGA Resources, Performance Overhead and Code Size

In this section, we delve into the implementation cost obtained on the FPGA board. In Table 5, we present the area metrics of *Diwall* and the network processor in terms of lookup tables (LUTs), flips flops (FFs), and the maximum frequency for an FPGA (XC7A100TICSG324-1) deployed on an Arty-A7 100T board. All overhead percentages for *Diwall* provided in the table are calculated relative to the CPU of the network processor, rather than the total available FPGA resources These results are obtained using the Xilinx Vivado v2020.2 tool. We evaluate three variants of the network processor:

- **V1:** This is the RISC-V baseline version, devoid of any *Diwall* component. By default, only one HPC is enabled for generic use and not affected to any microarchitectural event. This serves as a comparison benchmark for V2 and V3.
- **V1':** V1 integrated with the HPCs: (`HPC1`, `HPC2`). This version illustrates the effect of activating 2 HPCs on the RISC-V CPU.

Table 5. Resource Utilization and Maximum Frequency of Implementation for 5 Versions of the Network Processor with and without *Diwall*

Network Processor			Overheads relative to CPU		Freq
CV32E41P	**HPCs**	***Diwall***	**LUT**	**FF**	**MHz**
V1 (Base)	1 (Default)	✗	4676 (+00%)	2136 (+00%)	65.69
V1'	2	✗	4777 (+2.16%)	2217 (+3.79%)	65.60
V1"	3	✗	4897 (+4.73%)	2298 (+7, 58%)	65.62
V2 ([7])	2	✓	5105 (+9.17%)	2352 (+10.11%)	65.50
V3 (This work)	3	✓	5345 (+14.30%)	2625 (+22.89%)	65.07

- **V1":** V1 integrated with the HPCs: (`HPC1`, `HPC2`), and `NwHPC`. This version illustrates the effect of activating three HPCs on the RISC-V CPU.
- **V2:** It builds on V1' by incorporating the HPMtracer and Detector for detecting packet injection attacks. This highlights the overhead introduced by the previous version of *Diwall* [7] in contrast to the V1 baseline.
- **V3:** It builds on V1" by incorporating the HPMtracer, EWMA preprocessing, and Detector for detecting jamming and packet injection attacks. This highlights the overhead introduced by *Diwall* in contrast to the V1 baseline.

For *Diwall* architecture in V3, CV32E41P employs three HPCs. Two counters track microarchitectural events (`HPC1` and `HPC2`), whereas a dedicated register-based HPC `NwHPC` measures the network metric RSSI.

Comparison between V3 and V1" with V1:

- The integration of the three HPCs incurs an overhead of 7.58% in FFs and 4.73% in LUTs.
- *Diwall* itself contributes to an area overhead of approximately +22.83% in FFs and 14.30% in LUTs.
- The additional FFs and LUTs in *Diwall* are predominantly associated with the three counters.

When comparing V1" and V1' with V1, we observe that an increase in the number of HPCs can lead to a significant increase in overhead. Designers should consider this tradeoff between the number of features used in a dataset and the detection rate, alongside the overhead of HPCs.

When examining the performance implications of *Diwall* design units V3 and V2 regarding V1:

- There is no substantial impact on design performance.
- The maximum frequency consistently hovers at approximately 65 MHz.

We compare the code size percentages in two scenarios: one with and one without the *Diwall* integration overhead. The proposed solution doesn't impact the code size of a LoRaMAC-node. Integrating *LibDiwall* into an IoT protocol stack for configuration and control adds only 74 extra bytes to the code size. In the context of a LoRaMAC-node [35] implementation in the RISC-V BSP, this represents a 0.07% increase in code size. This evaluation was conducted on a mid-range FPGA, but *Diwall* can also be implemented on smaller FPGAs suitable for IoT devices.

The outcomes highlighted in this subsection emphasize that *Diwall* is a resource-efficient hardware solution, particularly for IoT end-devices with limited resources. It effectively detects both packet injection and jamming attacks at the network processor level.

5.3 Discussion

In this paper, we initially evaluated a hardware-based HIDS named *Diwall* with a LoRa PHY layer using a simplified MAC layer, and then with a LoRaWAN network. The simplified MAC layer is instrumented with

the necessary instructions for controlling the detection mechanism from software. In our evaluation, we tested jamming and packet injection attacks. In this context, we achieved an overall detection accuracy of approximately 99.94%. Statistically, out of 10,000 LoRa packets, only six are incorrectly predicted. These mispredictions are largely attributed to packets not being received due to adverse channel conditions and a high PLR, particularly during jamming attacks.

We opted for the simplified MAC to enable rapid prototyping. By focusing on MAC processing post-PHY layer, we were able to more effectively identify interesting metrics in a shorter time frame. Consequently, the parameters generated for *Diwall*'s Detector are not universally applicable, and may vary under different conditions. This results in an oversimplification of the synthetic dataset. This may lead to a loss of precision due to overfitting to the synthetic data when scaling up.

Several limitations can be enumerated. **Limitation (l_a)**: the values generated for `K1` and `K2`, as well as the microarchitectural events selection, such as `LD_STALL` and `BRANCH_TAKEN`, result from the usage of the simplified MAC layer. These results can be modified if a different monitored MAC layer window is defined.
Limitation (l_b): There is a potential vulnerability regarding adaptive attackers who could try to evade detection by crafting attacks that avoid triggering the monitored HPCs. For instance, attackers might design exploits that minimize `LD_STALL` and `BRANCH_TAKEN`, thereby flying under the radar. However, since the proposed method tracks cumulative HPC values rather than a sequence of actions, any modification to the program workflow, such as forging packets to mimic legitimate behavior or reusing existing code gadgets, leads to higher HPC values, which *Diwall* detects as an attack. Similarly, for a jamming attack to be successful, the attacker must be in close proximity to the victim or use a higher transmit power transmission [17], leading to higher RSSI values. While an attacker could attempt to mimic normal RSSI values by positioning themselves near the legitimate gateway, this does not guarantee effective jamming. The success of jamming largely depends on the attacker's closeness to the target or use of a higher transmit power transmission [17], which typically results in higher than normal RSSI values.

Limitation (l_c): changes in RSSI can occur for legitimate behavior of the gateway, and can affect the generation of UCL and LCL, as RSSI values vary based on location. Usually, jammers provide higher RSSI value because of their proximity to the victim node, which will deviate up to the UCL. The jammer may use lower RSSI values to avoid detection but needs to transmit higher RSSI values to effectively jam legitimate signals. An adjustment mechanism of *Diwall*'s parameters, should overcome these limitations.

Several IoT protocol designers utilize the LoRa PHY layer as a foundation and develop custom MAC layers to fulfill their requirements. Developers can leverage the proposed framework for simulation and training to enhance the security of their MAC layer. To illustrate a potential use case of *Diwall*, we tested our framework in LoRaWAN networks, a widely used MAC layer for LoRa in IoT end-devices. We equipped the LoRaMAC-node, an open-source IoT protocol stack for LoRaWAN, with the module detailed in Section 3 to detect packet injection and jamming attacks. Within the MAC layer stack, various software parts can be monitored using our approach. We specifically selected a window within the LoRaMAC-node software that corresponds to the reception function of LoRa frames from the PHY layer. The selected software window is instrumented with instructions provided by *LibDiwall*. During the execution of this software window, microarchitectural events and RSSI values are monitored, as explained in Subsection 3.2. We introduced modifications to the monitored window to create vulnerabilities for packet injection attacks. After training, both `K1` and `K2` values were adjusted, and a new microarchitectural event `INSTR` was selected. It refers to the number of retired instructions and was monitored using `HPC2`. These parameter changes result from the monitored software window change, as discussed earlier in this section about the limitation (l_a). Indeed, the LoRaMAC-node stack is substantially more complex than the previous simplified MAC layer. The UCL and LCL values remained the same, as the experimental setup was not modified. The overall detection accuracy achieved in this experiment was approximately 99.98%. There was no drop in performance compared to the synthetic simulation utilized in the simplified MAC layer. Effectively, it

means that instead of relying solely on software patches to mitigate buffer overflow vulnerabilities, *Diwall* can monitor the network processor behavior and distinguish between legitimate and malicious activities.

Some limitations remain and could be addressed in future work. **Limitation (l_d)**: first, it is vital to carefully choose the monitored window because buffer overflow can occur in different parts of MAC layer software stack. Notably, buffer overflows can produce excessive values of microarchitectural events. **Limitation (l_e)**: second, patches introduced by IoT stacks developers to the monitored window may impact the previously generated decision parameters, requiring retraining.

To address these challenges, we could extend our approach to incorporate a hardware module dedicated to updating decision parameters. New parameters can be regenerated periodically or based on a new selected monitored window, which may increase reconfigurability and allow overcoming limitations (l_d) and (l_e).

6 CONCLUSION AND FUTURE WORK

In this paper, we present an embedded, lightweight strategy for monitoring IoT devices and detecting wireless attacks. While previous research has primarily focused on network-based IDS for IoT, often limited by single-layer metrics, gateway-level host-based IDS utilize resource-intensive algorithms for analysis and detection. We outline the hybrid hardware and software implementation of the *Diwall* detection module and associated mechanism. An experimental evaluation for a synthetic case using simulated packets and a more realistic case using a LoRaMAC node have been detailed. The security strategy employed includes microarchitecture and network metrics monitoring at the network processor level, with a focus on wireless attack detection. The Detector module analyzes the EWMA value RSSI to detect jamming attacks and a dedicated HPC, added to the RISC-V processor, monitors RSSI metadata in hardware. Furthermore, a Decision Tree model consumes microarchitectural HPC data to identify packet injection attacks which attempt to exploit memory vulnerabilities. *Diwall* was implemented on an FPGA and integrated within a RISC-V processor. The initial evaluation methodology focused on a streamlined MAC layer tasked with parsing network packets received from the LoRa PHY layer. The module and associated mechanism underwent evaluation utilizing a comprehensive open-source IoT protocol stack, encompassing both LoRa PHY and LoRaWAN MAC layers.

The detection rates in the LoRaWAN context reached 99.98%. Thus, *Diwall* demonstrates the ability to promptly detect packet injection that exploit memory vulnerabilities, as well as real-time recognition of jamming attacks. The latter has been achieved using a low-complexity FPGA implementation. The FPGA eliminates the need for a complete system reprogramming when new attack scenarios emerge. The FPGA implementation introduces an area overhead of approximately 14.30%, consuming 22.89% of LUTs and FFs compared to the CV32E40P baseline on an Arty A7 100T board, without affecting the maximum clock frequency of 65 MHz. The implementation represents a negligible increase in code size of 0.07% regarding the LoRaMAC-node software, corresponding to 74 bytes of code memory.

This work could be extended with potential security and flexibility enhancements, by (1) evaluating security in the context of a more complex threat model involving an attacker's perception of *Diwall* presence and (2) investigating more advanced attack scenarios. One improvement to address these points would be to use a more complex model, like a combination of a decision tree and an isolation forest, for anomaly detection and classification. Another envisioned enhancement involves the development of a *Diwall* version capable of complete configuration using the CPU control status registers (CSRs), thereby enhancing the adaptability of its parameters.

In this work, we focused on remote attackers, but future efforts will extend the threat model to include attackers with physical access to IoT devices, such as those performing fault injection and side-channel attacks. Additionally, poisoning attacks will be considered, particularly when employing online training in hardware, to prevent attackers from manipulating decision thresholds established during training. The IDS could support post-quantum cryptographic schemes, such as secure key exchange and digital signatures, to enhance resilience against

emerging quantum threats. This integration ensures secure communication with the gateway and authenticates alerts generated by the IDS, particularly when transmitting alerts for further analysis, thereby mitigating potential disruptions across the entire IoT environment. Another potential avenue for future work involves developing our IDS as an IP hardware component for integration with various SoC architectures through hardware/software co-design. Finally, the energy consumption of our proposed IDS approach will be studied to ensure efficient operation in resource-constrained environments.

All related materials, including datasets and code, are available open source on GitHub [1]

ACKNOWLEDGMENTS

This research was funded by a PhD grant from the *Conseil régional de Bretagne*, France, and the *Université Bretagne Sud, Lorient, France*. In addition, part of the research was also funded by grant CNS-1902532 from the National Science Foundation in the United States, and by the GdR IASIS in France.

REFERENCES

[1] IoT Analytics. 2024. *Number of Connected IoT Devices.* Retrieved September 17, 2024 from https://iot-analytics.com/number-connected-iot-devices/

[2] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17)*. Vancouver, BC, Canada, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Trans. Priv. Secur.* 23, 3, Article 14 (Jul 2020), 28 pages. https://doi.org/10.1145/3394497

[4] Emekcan Aras, Nicolas Small, Gowri Sankar Ramachandran, Stéphane Delbruel, Wouter Joosen, and Danny Hughes. 2017. Selective Jamming of LoRaWAN Using Commodity Hardware. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* (Melbourne, VIC, Australia) *(MobiQuitous 2017)*. New York, NY, USA, 363–372. https://doi.org/10.1145/3144457.3144478

[5] Armis. 2019. *BLEEDINGBIT Vulnerability Analysis.* Retrieved September 19, 2024 from https://www.armis.com/research/bleedingbit/

[6] Gildas Avoine and Loïc Ferreira. 2018. Rescuing LoRaWAN 1.0. In *Proceedings of Springer Financial Cryptography and Data Security*. Berlin, Heidelberg, 253–271. https://doi.org/10.1007/978-3-662-58387-6_14

[7] Mohamed El Bouazzati, Russell Tessier, Philippe Tanguy, and Guy Gogniat. 2023. A Lightweight Intrusion Detection System against IoT Memory Corruption Attacks. In *Proceedings of the 26th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)* (Tallinn, Estonia). 118–123. https://doi.org/10.1109/DDECS57882.2023.10139718

[8] Malcolm Bourdon, Pierre-François Gimenez, Eric Alata, Mohamed Kaaniche, Vincent Migliore, Vincent Nicomette, and Youssef Laarouchi. 2020. Hardware-Performance-Counters-based anomaly detection in massively deployed smart industrial devices. In *Proceedings of the 19th IEEE International Symposium on Network Computing and Applications (NCA)* (Cambridge, MA, USA). 1–8. https://doi.org/10.1109/NCA51143.2020.9306726

[9] Ismail Butun, Salvatore D. Morgera, and Ravi Sankar. 2014. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Communications Surveys & Tutorials* 16, 1 (2014), 266–282. https://doi.org/10.1109/SURV.2013.050113.00191

[10] Romain Cayre. 2022. *Offensive and defensive approaches for wireless communication protocols security in IoT*. Ph. D. Dissertation. INSA Toulouse, Toulouse, France. https://tel.archives-ouvertes.fr/tel-03841305

[11] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. 2021. InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections. In *Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (Taipei, Taiwan). 388–399. https://doi.org/10.1109/DSN48987.2021.00050

[12] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. 2021. WazaBee: attacking Zigbee networks by diverting Bluetooth Low Energy chips. In *Proceedings of 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (Taipei, Taiwan). 376–387. https://doi.org/10.1109/DSN48987.2021.00049

[13] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. 2019. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (SP)* (San Francisco, CA, USA). 20–38. https://doi.org/10.1109/SP.2019.00021

[1] https://github.com/mohamedElbouazzati/Diwall-framework

[14] Mojtaba Eskandari, Zaffar Haider Janjua, Massimo Vecchio, and Fabio Antonelli. 2020. Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices. *IEEE Internet of Things Journal* 7, 8 (2020), 6882–6897. https://doi.org/10.1109/JIOT.2020.2970501

[15] Pierre-Francois Gimenez, Jonathan Roux, Eric Alata, Guillaume Auriol, Mohamed Kaaniche, and Vincent Nicomette. 2021. RIDS: Radio Intrusion Detection and Diagnosis System for Wireless Communications in Smart Environment. *ACM Trans. Cyber-Phys. Syst.* 5, 3, Article 24 (Apr 2021), 1 pages. https://doi.org/10.1145/3441458

[16] Frank Hessel, Lars Almon, and Flor Álvarez. 2020. ChirpOTLE: A Framework for Practical LoRaWAN Security Evaluation. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Linz, Austria) *(WiSec '20)*. ACM, New York, NY, USA, 306–316. https://doi.org/10.1145/3395351.3399423

[17] Ningning Hou, Xianjin Xia, and Yuanqing Zheng. 2023. Jamming of LoRa PHY and Countermeasure. *ACM Trans. Sen. Netw.* 19, 4, Article 80 (May 2023), 27 pages. https://doi.org/10.1145/3583137

[18] Chin-Ya Huang, Ching-Wei Lin, Ray-Guang Cheng, Shanchieh Jay Yang, and Shiann-Tsong Sheu. 2019. Experimental Evaluation of Jamming Threat in LoRaWAN. In *Proceedings of the 89th IEEE Vehicular Technology Conference (VTC2019-Spring)* (Kuala Lumpur, Malaysia). 1–6. https://doi.org/10.1109/VTCSpring.2019.8746374

[19] Texas Instruments. 2023. *CC1352R: Multi-Band Wireless SoC.* Retrieved September 17, 2024 from https://www.ti.com/product/CC1352R

[20] Sai Praveen Kadiyala, Pranav Jadhav, Siew-Kei Lam, and Thambipillai Srikanthan. 2020. Hardware Performance Counter-Based Fine-Grained Malware Detection. *ACM Trans. Embed. Comput. Syst.* 19, 5, Article 38 (Sep 2020), 17 pages. https://doi.org/10.1145/3403943

[21] Prabhakaran Kasinathan, Gianfranco Costamagna, Hussein Khaleel, Claudio Pastrone, and Maurizio A. Spirito. 2013. DEMO: An IDS Framework for Internet of Things Empowered by 6LoWPAN. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. New York, NY, USA, 1337–1340. https://doi.org/10.1145/2508859.2512494

[22] Forescout Research Labs. 2020. *AMNESIA:33, How Embedded TCP/IP Stacks Breed Critical Vulnerabilities.* Retrieved September 17, 2024 from https://i.blackhat.com/eu-20/Wednesday/eu-20-dosSantos-How-Embedded-TCPIP-Stacks-Breed-Critical-Vulnerabilities-wp.pdf

[23] Ivan Marino Martinez Bolivar. 2021. *Jamming on LoRaWAN Networks : from modelling to detection.* Ph. D. Dissertation. Institut National des Sciences Appliquées de Rennes, Rennes, France. https://theses.hal.science/tel-03196484

[24] Lime Microsystems. 2017. *LimeSDR mini RX & TX 10MHz - 3.5GHz Full-Duplex.* Retrieved September 17, 2024 from https://www.passion-radio.fr/emetteur-sdr/limesdr-mini-667.html

[25] Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou. 2009. A survey on jamming attacks and countermeasures in WSNs. *IEEE Communications Surveys & Tutorials* 11, 4 (2009), 42–56. https://doi.org/10.1109/SURV.2009.090404

[26] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, and Guy Gogniat. 2018. NIGHTs-WATCH: a cache-based side-channel intrusion detector using hardware performance counters. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy* (Los Angeles, California) *(HASP '18)*. New York, NY, USA, 1–8. https://doi.org/10.1145/3214292.3214293

[27] OpenHWGroup. 2024. *CV32E40P User Manual.* Retrieved September 19, 2024 from https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/latest/

[28] Opeyemi Osanaiye, Attahiru Alfa, and Gerhard Hancke. 2018. A Statistical Approach to Detect Jamming Attacks in Wireless Sensor Networks. *Sensors* 18, 6 (May 2018), 1691. https://doi.org/10.3390/s18061691

[29] Hossein Pirayesh and Huacheng Zeng. 2022. Jamming Attacks and Anti-Jamming Strategies in Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 24, 2 (2022), 767–809. https://doi.org/10.1109/COMST.2022.3159185

[30] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks* 11, 8 (2013), 2661–2674. https://doi.org/10.1016/j.adhoc.2013.04.014

[31] Henri Ruotsalainen. 2022. Reactive Jamming Detection for LoRaWAN Based on Meta-Data Differencing. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '22)*. New York, NY, USA, Article 98, 8 pages. https://doi.org/10.1145/3538969.3543805

[32] Henri Ruotsalainen, Guanxiong Shen, Junqing Zhang, and Radek Fujdiak. 2022. LoRaWAN Physical Layer-Based Attacks and Countermeasures, A Review. *Sensors* 22, 9 (Apr 2022), 3127. https://doi.org/10.3390/s22093127

[33] Amani K. Samha, Nidhi Malik, Deepak Sharma, Kavitha S, and Papiya Dutta. 2023. Intrusion Detection System Using Hybrid Convolutional Neural Network. *Mobile Networks and Applications* (2023). https://doi.org/10.1007/s11036-023-02223-6

[34] Aellison C. T. Santos, José L. Soares Filho, Ávilla Í. S. Silva, Vivek Nigam, and Iguatemi E. Fonseca. 2023. BLE injection-free attack: a novel attack on bluetooth low energy devices. *Journal of Ambient Intelligence and Humanized Computing* 14, 5 (2023), 5749–5759. https://doi.org/10.1007/s12652-019-01502-z

[35] Semtech. 2013. *LoRaMac-Node Repository.* Retrieved August 30, 2023 from https://github.com/Lora-net/LoRaMac-node

[36] Statista. 2023. *Worldwide Annual Internet of Things Attacks.* Retrieved September 17, 2024 from https://www.statista.com/statistics/1377569/worldwide-annual-internet-of-things-attacks/

[37] STMicroelectronics. 2023. *STM32WL55CC: Wireless System-on-Chip (SoC).* Retrieved September 17, 2024 from https://www.st.com/en/microcontrollers-microprocessors/stm32wl55cc.html

[38] Mario Strasser, Boris Danev, and Srdjan Čapkun. 2010. Detection of Reactive Jamming in Sensor Networks. *ACM Trans. Sen. Netw.* 7, 2, Article 16 (sep 2010), 29 pages. https://doi.org/10.1145/1824766.1824772

[39] Espressif Systems. 2023. *ESP32-H2: Wi-Fi and Bluetooth LE SoC*. Retrieved September 17, 2024 from https://www.espressif.com/en/products/socs/esp32-h2

[40] Aliya Tabassum, Aiman Erbad, and Mohsen Guizani. 2019. A Survey on Recent Approaches in Intrusion Detection System in IoTs. In *Proceedings of the 15th IEEE International Wireless Communications & Mobile Computing Conference (IWCMC)* (Tangier, Morocco). 1190–1197. https://doi.org/10.1109/IWCMC.2019.8766455

[41] Tencent Blade team. 2020. *LoRaDawn*. Retrieved September 19, 2024 from https://github.com/Lora-net/LoRaMac-node/security

[42] Enrico Testi, Luca Arcangeloni, and Andrea Giorgetti. 2023. Machine Learning-Based Jamming Detection and Classification in Wireless Networks. In *Proceedings of the 2023 ACM Workshop on Wireless Security and Machine Learning* (Guildford, United Kingdom) *(WiseML'23)*. New York, NY, USA, 39–44. https://doi.org/10.1145/3586209.3591395

[43] Zhongru Wang, Xinzhou Xie, Lei Chen, Shouyou Song, and Zhongjie Wang. 2023. Intrusion Detection and Network Information Security Based on Deep Learning Algorithm in Urban Rail Transit Management System. *IEEE Transactions on Intelligent Transportation Systems* 24, 2 (Feb. 2023), 2135–2143. https://doi.org/10.1109/TITS.2021.3127681

[44] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer, and Dongyan Xu. 2020. BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks. In *Proceedings of the 23rd USENIX International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)* (San Sebastian). 397–411. https://www.usenix.org/conference/raid2020/presentation/wu

[45] Wenqing Yan, Sam Hylamia, Thiemo Voigt, and Christian Rohner. 2020. PHY-IDS: A Physical-Layer Spoofing Attack Detection System for Wearable Devices. In *Proceedings of the 6th ACM Workshop on Wearable Systems and Applications* (Toronto, Ontario, Canada) *(WearSys '20)*. New York, NY, USA, 1–6. https://doi.org/10.1145/3396870.3400010

[46] Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. 2017. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications* 84 (2017), 25–37. https://doi.org/10.1016/j.jnca.2017.02.009

[47] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *Proceedings of the 29th USENIX Conference on Security Symposium (SEC'20)*. USA, Article 3, 18 pages.

Received 14 February 2024; revised 12 March 2009; accepted 5 June 2009