A Lightweight Intrusion Detection System against IoT Memory Corruption Attacks

Mohamed El Bouazzati*, Russell Tessier†, Philippe Tanguy* and Guy Gogniat*
* Univ. Bretagne-Sud, UMR 6285, Lab-STICC F-56100 Lorient, France
†Univ of Massachusetts, Department of Electrical and Computer Engineering, Amherst, MA, USA
* {mohamed.el-bouazzati, philippe.tanguy, guy.gogniat}@univ-ubs.fr
† tessier@umass.edu

Abstract—**Attacks against internet-of-things (IoT) end-devices represent a significant threat since their wireless communication capabilities provide a potential attack entry point. To address this threat, we demonstrate the use of hardware performance counters (HPCs) in a host-based intrusion detection system (HIDS). The counter-based monitors are customized to support IoT end-devices which use low data rate GHz and sub-GHz protocols. Our solution implements a hardware unit that performs data tracing for a 32-bit RISC-V based wireless connectivity unit. The unit can detect ongoing remote attacks in real time. We demonstrate the effectiveness of our system by detecting a packet injection exploit. Our FPGA implementation of HIDS has a logic overhead of about 6% and design frequency penalty of less than 1% for a RISC-V processor.**

Index Terms—**IoT Security, RISC-V, Network Processor Architecture, HIDS, LoRa, HPC.**

I. Introduction

Systems-on-chip (SoCs) [1] for IoT end-devices are typically deployed with security protections such as cryptography primitives, an update mechanism, and secure boot. SoCs with wireless capabilities, such as LoRaWAN and Bluetooth Low Energy (BLE), that are currently deployed in the field do not integrate dedicated hardware mechanisms to counteract remote attacks that exploit wireless communication [2], [3].

In this paper, we evaluate an implementation of a host-based intrusion detection system (HIDS) based on hardware performance counters (HPCs). These performance counters can detect wireless remote attacks related to LoRaWAN and Bluetooth protocols. We implement the lightweight HIDS in hardware with a tracer module to monitor data and a detector module to analyze and classify attacks.

The HIDS tracer uses a network processor which supports wireless connectivity. It monitors hardware HPCs that indicate the current activity status of the network processor. The HIDS detection module, located outside the network processor, uses a machine learning classification model to analyze cumulative HPC values. To assess our approach, the detection accuracy and resource utilization of the implemented host-based intrusion detection system are quantified.

The remainder of this paper is structured as follows. Section II provides an overview of the security issues associated with wireless IoT. Section III details previous intrusion detection system implementations. Section IV presents our HIDS concept and describes its deployment. Section V describes our experimental setup and implemented architecture, and Section VI quantitatively evaluates our HIDS.

II. Security Context

Attacks against IoT end-devices are a significant threat since they contain wireless communication capabilities that may be vulnerable to attacks. In this section, we list common vulnerabilities and attacks, present the considered threat model and give an overview of related work regarding countermeasures.

A. Vulnerabilities and Attacks

Recently several vulnerabilities were discovered in IoT device communication for selected protocols. *AMNESIA33* [4] exposed 33 new critical vulnerabilities found in open source TCP/IP stack based protocols *(uIP, FNET, picoTCP and Nut/Net)* used by millions of IoT devices supplied by over 150 vendors. *BLEEDINGBIT* [5] and *LoRaDawn* [6] are attacks related to memory corruption that exploit vulnerabilities in the packet parsers of the BLE and LoRaWAN software stacks. An attacker can exploit the lower layers of a protocol stack, such as the MAC layer or the physical layer, to perform denial-of-service (DoS), packet injection, man-in-the-middle (MITM) attacks, and remote code execution (RCE).

Attacks that exploit LoRaWAN MAC and physical layer vulnerabilities to perform DoS, packet injection and replay attacks have also been reported [3], [7], [8]. Similar attacks are possible for the BLE stack [9]–[12]. IoT protocols are often subject to packet injection based attacks resulting in exploits such as DoS, MITM, RCE and privilege escalation. Systems-on-chip (SoCs) for IoT devices [1] frequently have two or more supported protocols, resulting in a need for an independent protocol security mitigation solution. Our HPCs-based HIDS approach targets the detection of packet injection attacks exploiting memory corruption vulnerabilities within a network processor.

B. Mitigation

Hardware and software based security mechanisms that can detect IoT attacks have been developed [13], [14]. Generally, these mechanisms provide at least one of the three elements of the triad (C/I/A): confidentiality, integrity and availability. Software-based security mechanisms include secure boot and code instrumentation, while hardware-based mechanisms include information flow tracking. Davis et al. [13] use a hardware-based control flow integrity extension to prevent code re-use attacks, such as return-oriented programming (ROP), in Intel processors. Such mechanisms require compiler-level and architectural modifications, including changes inside the processor pipeline. Other countermeasures include memory protection by design with a safe programming language (e.g., RUST [14]). Saeed et al. [15] performed code instrumentation by adding tags to memory locations for every memory allocation, and using extra tag-checking instructions for all memory accesses to detect illegal accesses. Such mechanisms typically require memory layout changes, which result in memory overhead. Static analysis tools can be used to detect bugs at the compilation stage.

Intrusion and anomaly detection approaches were proposed for IoT environments to detect attacks using an attack signature [16] or by learning a pattern of legitimate system activities [17]. These solutions are divided into three parts: acquisition, analysis and alert. Acquisition uses hardware probes and/or software to collect information from the system. Collected information is then analyzed to identify in-progress attacks. If a malicious action is detected, the system notifies the user. Such mechanisms [18] have been shown to accurately detect attacks. However, the performance (detection rate) and area, code size and power consumption overheads remain challenging.

In this work, we target resource-constrained IoT end-devices that have an approximate power consumption of 1 to 2 Watts and a memory capacity of 100 to 300 kB [1]. In previous work [19], an end-device collects metrics and sends them to a server or a gateway for remote analysis and detection. Rather, in our approach, tracing and detection are both implemented on the IoT end-device.

III. Related Work

Our HIDS, detailed in Section IV, uses hardware metrics to prevent intrusions. Several research studies [19]–[22] have demonstrated the use of HIDS with hardware metrics to detect attacks. Bourdon et al. [19] describe an anomaly detection approach based on the analysis of data from hardware performance counters (HPCs). The data is used to identify compromised devices among a massive population of similar IoT devices. An accurate behavioral model of the device was built using hardware performance event values from an ARM Cortex A53 processor: memory cache, instructions, exceptions, prediction branches, and bus access. The values reflect the current state of the CPU. A kernel tracing module is installed in device software for monitoring. The tracer stores the HPC data in a local file every 5 seconds. The analysis and detection module deployed on the server receives a data file every 30 minutes.

The detection strategy was assessed with multiple attack scenarios on a population of $10,000$ devices. The simulated attacks mainly consider DoS and packet injection exploits. The detection accuracy of the approaches was determined using machine learning algorithms via two metrics: the true positive rate (TPR) and the false positive rate (FPR). Since the analysis and detection modules are remotely deployed on a server, the real-time detection of IoT device intrusions is quite challenging. The distributed implementation is needed due to the limited resources available on the IoT device (power, memory and CPU, etc.).

Our approach differs from the one noted above in two main ways. First, we detect ongoing remote packet injection attacks on the network processor of the wireless connectivity unit. Second, our monitoring and detection modules are implemented on the IoT platform and do not require remote detection using a server or gateway.

IV. Methodology

A. Threat Model

Fig. 1 shows the wireless IoT threat model. The victim is an IoT SoC with a CPU used for user applications and a wireless connectivity subsystem which uses one or more protocol stacks managed by the network processor. Since most of the vulnerabilities described in Section II are found in the physical and MAC layers, these layers of the IoT protocol stack form the attack surface. The attacker is remote and can use a software-defined radio (SDR) platform or a protocol-specific dongle to perform an attack. The attacker attempts to perform packet injection into the victim's wireless connectivity network processor to achieve one of the following exploits: DoS, MITM, RCE and privilege escalation.

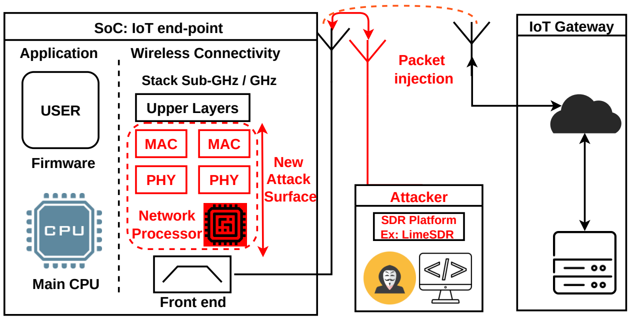

Fig. 1. Wireless communication threat model

B. Hardware Based HIDS Design

Fig. 2 shows the processing flow for our hardware-based host intrusion detection system. The software operating on the network processor processes the MAC layer of an IoT protocol stack. It parses packets that are received from the physical layer. Simultaneously, the HIDS, which is implemented in hardware, monitors hardware performance events using HPCs available on the network processor, as demonstrated in Fig. 3.

The cumulative values of the hardware events provide insight into network processor behavior during packet parsing. The produced values of the counters are used within the detection module to identify intrusions and attacks.

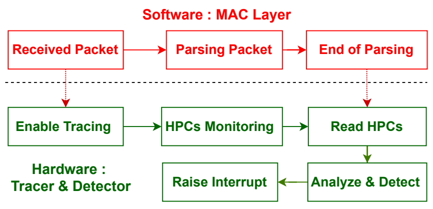

Fig. 2. Flow diagram of network packet processing, HPC monitoring and detection.

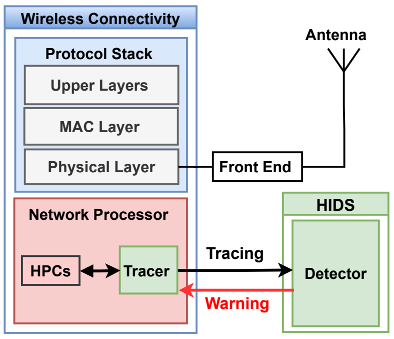

Fig. 3. Wireless connectivity and HIDS (Host Intrusion Detection System) block diagram

V. Experimental Setup

Our HIDS approach in Fig. 3 has been designed, implemented and evaluated. A complete system, including HIDS hardware, has been evaluated using cycle-accurate simulation with Verilator [23], a Verilog simulator. Following simulation, the entire system was implemented using Xilinx Vivado in an Artix-7 35T FPGA on a Digilent Arty 35T/100T board, and has been tested for the LoRa protocol stack with a simplified MAC layer using a physical testbed.

A. Simulation Testbed

The use of a simulation testbed to generate a large data set to train and build machine learning models has been a common approach in the field of machine learning. By simulating various scenarios and conditions, this approach can also be useful in reducing the cost and complexity of obtaining real-world data for training and testing.

Fig. 4 shows the SoC system built with LiTeX [24], a SoC builder. A RISC-V soft-core CV32E41P, developed by the Open Hardware Group [25], serves as the network processor for wireless connectivity. This small, 32-bit, in-order RISC-V core has a 4-stage pipeline that implements the RV32IM RISC-V instruction set architecture according to the RISC-V privileged specification [26]. Fig. 4 shows a minimal SoC including network processor (CV32E41P), RAM and UART, interconnected via a Wishbone bus.

The CV32E41P includes a set of 64-bit HPCs that are placed with the control and status registers (CSRs). The CV32E41P implements a clock cycle counter and a retired instruction counter, which are always activated, and 29 configurable event counters. The event counters can be accessed with a parameter of 0 and 2 through 29. The CV32E41P tracks a set of hardware events using the counters listed in Tab. I. These events are each assigned to an event counter using a CSR event selector.

TABLE I
List of hardware events monitored by the CV32E41P performance counters

Hardware Event	Description	Counter
CYCLES	Number of cycles	0
INSTR	Number of instructions retired	2
LD_STALL	Number of load use hazards	3
JMP_STALL	Number of jump register hazards	4
IMISS	Cycles waiting for instruction fetches	5
LD	Number of load instructions	6
ST	Number of store instructions	7
JUMP	Number of jumps (unconditional)	8
BRANCH	Number of branches (conditional)	9
BRANCH_TAKEN	Number of branches taken (conditional)	10
COMP_INSTR	Number of compressed instructions retired	11

The *Tracer* shown in the HIDS block diagram in Fig. 3 is directly connected to the CPU core. It captures behavioral information of a process running on the network processor by accumulating values of selected hardware events during network packet parsing. A developer can access the Tracer (called $HPMtracer$ in our implementation) to enable monitoring based on a predefined security policy.

The *Software* components shown in Fig. 4 include the firmware executed by the network processor. The software executes specific parts of a full IoT protocol stack including the MAC layer of the IoT protocol stack shown in Fig. 3. The firmware controls the *HPMtracer* using three signals: *HPM Reset*, *HPM Enable* and *HPM Stop*.

Network Traffic Generator and *Machine Learning Processing* operations, displayed in Fig. 4, are performed offline using Python for data set generation and processing. Additional details regarding attack scenarios and machine learning processing are presented in the two next subsections.

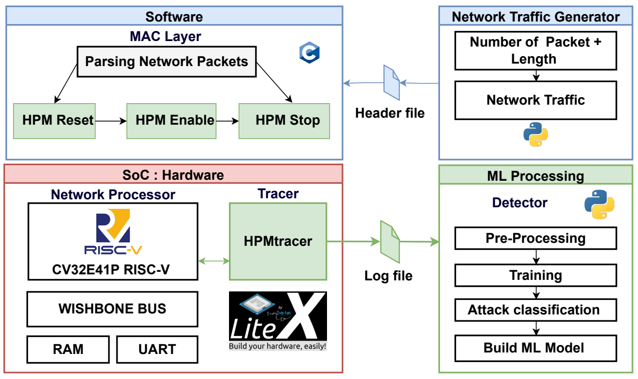

Fig. 4. Wireless communications testbed including network processor.

B. Attack Scenarios

We consider packet injection attacks based on software vulnerabilities. The attacks focus on memory corruption resulting in buffer overflow in the stack or the heap.

S1: Stack Overflow Exploit: The LoRaDawn *CVE-2020-11068* vulnerability can cause a buffer overflow in the LoRaWAN protocol stack due to a lack of buffer size checks.

The exploitation of this vulnerability may result in a remote denial of service for the LoRaWAN node. This vulnerability was reproduced by declaring a 10- or 23-byte reception buffer in the *packet parser* software. The reception buffer receives the data contained in the network packet. Using the *packet generator* module, we generated 1,000,000 network packets with 5- to 10-byte packet sizes. These packets fit within the allocated reception buffer size. We also generated 1,000,000 network packets with sizes between 13 and 23 bytes. These packets caused buffer overflows as shown in Tab. II.

S2: Heap Overflow Exploit: A heap overflow vulnerability based on *CVE-2022-0204* in Bluez, an open source Bluetooth stack, was also evaluated. The vulnerability is found in GATT protocol implementations. We reproduced the vulnerability by declaring a reception buffer in the heap using a malloc() statement in the packet parser software. We followed the same methodology as the stack overflow scenario using packet sizes shown in Tab. II.

TABLE II
ATTACKS SCENARIOS : THE PHYSICAL BUFFER SIZE IS 10 OR 23 BYTES. LARGER PACKETS RESULT IN A BUFFER OVERFLOW.

Attack Scenarios		**Buffer Size**	
Packet Type	**Traffic Size**	**Stack**	**Heap**
Legitimate	$5-10$ *bytes*	10 *bytes*	10 *bytes*
S1: Stack Overflow	$13-23$ *bytes*	10 *bytes*	23 *bytes*
S2: Heap Overflow	$13-23$ *bytes*	23 *bytes*	10 *bytes*

C. Generated Data Set

The data set used in this study consisted of $3,000,000$ samples, collected from the simulation testbed in Fig. 4, covering the scenarios outlined in Tab. II. We used 11 features for our dataset, which are a collection of cumulative hardware event values that were then fed into a machine learning classifier for training. Fig. 5 plots the behavior of two events among 11 events listed in Tab. I : *BRANCH_STALL*, *LD_STALL*. The figure profiles events for 3,000,000 network packet samples in terms of stack overflow, heap overflow and legitimate packet processing. The results shown in green reflect the normal behavior of the network processor with software in the MAC layer handling legitimate packets without attack. The red and yellow results illustrate the network processor behavior during stack overflow and heap overflow attacks. The magnitudes of HPCs increase during buffer overflow conditions compared to conditions caused by legitimate packet processing.

The highest HPC values in Fig. 5 result from network processor software execution based on the flow diagram shown in Fig. 2. This processing employs two reception buffers, one allocated on the stack and the other dynamically allocated on the heap. Both buffers store each incoming network packet. When a network packet exceeds the buffer size and a stack or heap buffer overflow occurs, increased event counter values occur.

In Fig. 5, counter values *BRANCH_TAKEN* and *LD_STALL* count the number of branch instructions that are taken and the number of load instructions that are stalled. Our attack scenarios involve writing more data to a buffer than it can handle. The extra data results in the alteration of CPU behavior to include unexpected branches and large delays in data retrieval from memory. Increased *BRANCH_TAKEN* and *LD_STALL* counter values can be used to identify potential attacks against the MAC layer of a protocol stack.

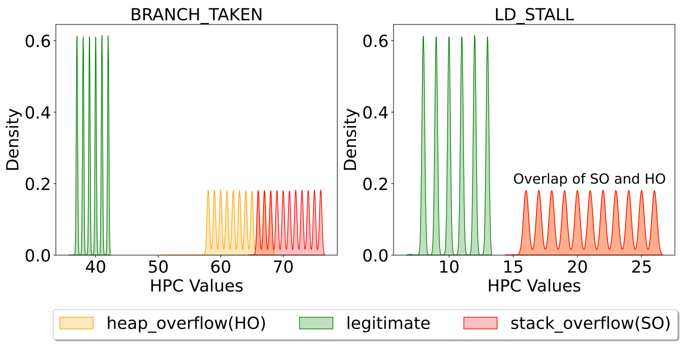

Fig. 5. Density area of cumulative values of hardware events LD_STALL, BRANCH_TAKEN in attack scenarios

D. Machine Learning-Processed Data Set

To generate classifier results for subsequent real-time processing, the training data set of 11 hardware events listed in Tab. I and collected in simulation was processed offline. A decision tree machine learning classifier, a supervised machine learning approach in which training samples provide decisions in a structured tree model, was used. The classifier differentiates between three categories of accumulated hardware values (heap overflow, stack overflow and legitimate). Once training was complete, the classifier was implemented in hardware using a SystemVerilog implementation. Decision tree models are suitable for FPGA implementation given their limited hardware overhead and suitable classification speed [27], [28]. Fig. 6 illustrates a decision tree model block diagram produced with our data set. The model splits the generated data set from Fig. 5 into three classes: *(legitimate, stack_overflow, heap_overflow)*. In this case *BRANCH_TAKEN* and *LD_STALL* HPC values form a set selected by the decision classifier among the 11 hardware events. Indeed during the learning phase *BRANCH_TAKEN* and *LD_STALL* HPC values demonstrate their high capacity to detect attacks. As we target a low complexity solution for IoT end-devices, using a minimal set of HPCs values is important. 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 process the received packets and to store the received data in buffers. A demonstrator was built based on these two events which represent values needed to construct an efficient and low complexity detector.

E. FPGA Implementation

The FPGA design shown in Fig. 3 includes three components: the soft-core *CV32E41P* RISC-V processor imported from GitHub [25] and the two modules of the HIDS: the *HPMtracer* and the *Detector*, including the decision tree model. The design operates at a maximum clock frequency

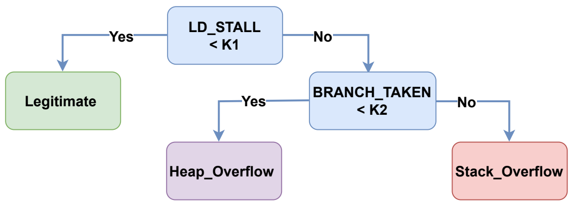

Fig. 6. Generated decision tree classifier model

of 65 MHz. The HPMtracer was implemented as a finite state machine controlled by the CV32E41P network processor. Monitoring is enabled when a new network packet is received and processed by the *CV324E41P*. The processor can enable/disable the *HPMtracer* by overwriting the corresponding bit in the *mcountinhibit* register.

Analysis of HPC cumulative values is performed by the *Detector* at the end of processing each network packet. The *HPMtracer* reads the HPC counters one at a time and enables the *Detector*. The *Detector* requires two clock cycles to decide whether the network packet is malicious or legitimate. At end of the analysis, the *Detector* alerts the *HPMtracer* if an attack is detected. If so, an exception is raised by the network processor.

F. LoRa Stack Testbed with HIDS

Following simulation, further experiments were conducted to test and validate the HIDS units using a physical testbed. As shown in Fig. 7, this use case, which includes the LoRa physical layer (PHY) and a simplified medium access control (MAC) layer, was used. LoRa wireless communication operates in the sub-GHz frequency band (868 MHz). Our SoC (Sec. V-A) was adapted to support the LoRa protocol stack by including the board support package (BSP) required for the LoRa SX1276 transceiver. The BPC also includes the HIDS elements (Sec. V-E) and a Litescope debugging unit. We used a microcontroller with a LoRa SX1276 transceiver to perform the attack scenarios listed in Tab. II.

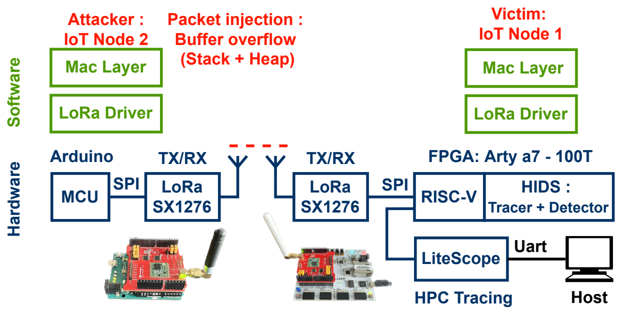

Fig. 7. LoRa Testbed with HIDS on Arty A7 FPGA board

VI. Results

In this section, we discuss the detection accuracy, resource utilization, and performance results of our security mechanism.

A. Detection Accuracy

The LoRa study described in the previous section analyzed $400,000$ network packets. The network traffic used in this evaluation consisted of $200,000$ legitimate (benign) packets and $200,000$ packets that were subject to stack and heap buffer overflows. The packet loss rate (PLR) in the testbed was approximately 2.72%, and a total of $389,097$ packets were received out of the $400,000$ packets sent. The PLR value was considered to be within an acceptable range for LoRaWAN networks. The loss rate occurred since the testbed is indoors and consists of two closed devices (20 cm), the spreading factor was set to $SF7$ and the time between sent packets was minimal ($\sim 250\ ms$). Tab. III reports true positive and negative and false positive and negative values and rates for the decision tree model in Fig. 6 using hardware implementation. The overall detection accuracy achieved is 99.98%. The cumulative hardware events data shown in Fig. 5 supports this high level of detection accuracy, as it allows for an efficient classification between legitimate and malicious network packets. A detection accuracy of 100% is not achieved with our decision tree model since we launched the worst case of our attack scenarios, where the sizes of legitimate and malicious packets were close (10 and 13 bytes, respectively).

TABLE III
HARDWARE DECISION TREE IMPLEMENTATION EVALUATION METRICS

LoRa based FPGA Testbed			
True Positives	**False Positives**	**True Neg.**	**False Neg.**
$195,704$	13	$193,327$	53
False Negative Rate (FNR):		0.027%	
False Positive Rate (FPR):		0.013%	
Detection Accuracy :		99.98%	

B. Area and Performance Overhead

In Tab. IV we report area in lookup tables (LUTs) and flip flops (FF) and maximum frequency on the Arty-A7 35T FPGA (XC7A35TICSG324-1L). The results were generated using Xilinx Vivado v2020.2. Three versions of the network processor were evaluated (V1, V2, and V3).

TABLE IV
IMPLEMENTATION RESOURCE UTILIZATION AND MAXIMUM FREQUENCY FOR THREE VERSIONS OF THE NETWORK PROCESSOR AND HIDS.

	HIDS elements			**Overhead**		**Freq**
	HPCs	**Tracer**	**Detector**	**LUT**	**FF**	**MHz**
V1	✓ (1)	-	-	4636(+00%)	1237(+00%)	65.86(+00%)
V2	✓ (2)	-	-	4802(+3.58%)	1318(+6.54%)	65.35(−0.77%)
V3	✓ (2)	✓	✓	4932(+6.38%)	1318(+6.54%)	65.47(−0.59%)

- **V1:** Version 1 is the baseline version of the processor. There is no dedicated security protection related to our threat model. *NUM_MHPMCOUNTERS=1*.
- **V2:** Version 2 of the processor includes two hardware performance counters (*BRANCH_TAKEN*, *LD_STALL*) *NUM_MHPMCOUNTERS=2*.
- **V3:** Version 3 includes HPMtracer and Detector for our HIDS with two hardware performance counters *NUM_MHPMCOUNTERS=2*.

Our hardware HIDS design used two hardware performance counters on the CV32E41P to achieve effective results. For V2, the counter overhead is 6.5% in flip-flops and 6.4% in LUTs compared to the baseline V1. The extra flip flops were mostly present in the two counters. The HIDS design (V3) units do not affect the design's performance (compared to V1). The maximum frequency is largely unchanged at around 65 MHz (Tab. IV). The results demonstrate that our HIDS is an efficient hardware-based solution in the context of resource-constrained IoT end-devices and that attacks can be detected at the network processor level.

VII. Conclusion and Future Work

This paper describes and demonstrates a host-based detection module that uses hardware events profiling at the network processor level. Addressing security directly at the network processor level is an interesting approach as it allows an early detection of attacks that use the communication link to compromise IoT end-devices. The hardware implementation benefits from large data set decision tree classification trained using a machine learning algorithm. This classification detects packet injection in the LoRa protocol stack in our case study. Our system is able to detect stack and heap overflow attacks in real time using a low complexity FPGA implementation. An FPGA allows for easy and efficient updates through reconfigurable hardware, eliminating the need to reprogram the entire system, which is important when new attack scenarios arise. The FPGA implementation exhibits detection accuracy at 99.98%. An area overhead of 6.4%, 6.5% of LUTs/FFs and a maximum clock frequency of 65 MHz is preserved. Although we employed the RISC-V ISA in our testbed for open-source purposes, our HIDS units approach is adaptable to other existing ISA, as most modern processors incorporate HPCs. In future work, our HIDS will incorporate additional metric layers to enable detection of a wider range of attack classes.

References

[1] T. Instruments, "Simplelink™ multi-band cc1352r wireless mcu launchpad™ development kit," 2019. [Online]. Available: https://www.ti.com/product/LAUNCHXL-CC1352R1/part-details/LAUNCHXL-CC1352R1

[2] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sun, and E. Kurniawan, "SweynTooth: Unleashing mayhem over bluetooth low energy," *Proceedings of the 2020 USENIX Annual Technical Conference, ATC 2020*, pp. 911–925, 2020.

[3] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, "Exploring the security vulnerabilities of LoRa," *2017 3rd IEEE International Conference on Cybernetics, CYBCONF 2017 - Proceedings*, no. December, 2017.

[4] F. R. Labs, "Amnesia:33, how tcp/ip stacks breed critical vulnerabilities in iot, ot and it devices," 2020. [Online]. Available: https://www.forescout.com/research-labs/amnesia33/

[5] "Bleedingbit," 2019. [Online]. Available: https://www.armis.com/research/bleedingbit/

[6] "Loradawn - multiple lorawan security vulnerabilities," 2020. [Online]. Available: https://blade.tencent.com/en/advisories/loradawn/

[7] F. Hessel, L. Almon, and F. Álvarez, "ChirpOTLE: A framework for practical LoRaWAN security evaluation," *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 306–316, 2020.

[8] G. Avoine and L. Ferreira, "Rescuing LoRaWAN 1.0," in *Financial Cryptography and Data Security: 22nd International Conference, FC 2018*, Nieuwpoort, Curaçao, Feb. 2018.

[9] R. Cayre, F. Galtier, G. Auriol, V. Nicomette, M. Kaâniche, and G. Marconato, "Injectable: Injecting malicious traffic into established bluetooth low energy connections," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2021, pp. 388–399.

[10] Y.Zhang, J.Weng, R.Dey, Y.Jin, Z.Lin, and X.Fu, "Breaking secure pairing of bluetooth low energy using downgrade attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 37–54.

[11] A. C. Santos, J. L. Filho, Á. Í. Silva, V. Nigam, and I. E. Fonseca, "BLE injection-free attack: a novel attack on bluetooth low energy devices," *Journal of Ambient Intelligence and Humanized Computing*, 2019.

[12] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy," *ACM Transactions on Privacy and Security*, vol. 23, no. 3, 2020.

[13] L. Davi, M. Hanreich, D. Paul, A. R. Sadeghi, P. Koeberl, D. Sullivan, O. Arias, and Y. Jin, "HAFIX: Hardware-Assisted Flow Integrity eXtension," *Proceedings - Design Automation Conference*, vol. 2015-July, 2015.

[14] N. D. Matsakis and F. S. Klock II, "The rust language," in *ACM SIGAda Ada Letters*, vol. 34, no. 3. ACM, 2014, pp. 103–104.

[15] A. Saeed, A. Ahmadinia, A. Javed, and H. Larijani, "Intelligent intrusion detection in low-power IoTs," *ACM Transactions on Internet Technology*, vol. 16, no. 4, 2016.

[16] B. F. L. M. Sousa, N. C. Soeiro, Z. Abdelouahab, W. F. Ribeiro, and D. C. P. Ribeiro, "An intrusion detection system for denial of service attack detection in internet of things," *ACM International Conference Proceeding Series*, 2017.

[17] W. Yan, S. Hylamia, T. Voigt, and C. Rohner, "PHY-IDS: A physical-layer spoofing attack detection system for wearable devices," *WearSys 2020 - Proceedings of the 6th ACM Workshop on Wearable Systems and Applications, Part of MobiSys 2020*, pp. 1–6, 2020.

[18] A. Tabassum, A. Erbad, and M. Guizani, "A survey on recent approaches in intrusion detection system in IoTs," *2019 15th International Wireless Communications and Mobile Computing Conference, IWCMC 2019*, pp. 1190–1197, 2019.

[19] M. Bourdon, P.-F. Gimenez, E. Alata, M. Kaaniche, V. Migliore, V. Nicomette, and Y. Laarouchi, "Hardware-performance-counters-based anomaly detection in massively deployed smart industrial devices," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, 2020, pp. 1–8.

[20] R. Gassais, N. Ezzati-Jivan, J. M. Fernandez, D. Aloise, and M. R. Dagenais, "Multi-level host-based intrusion detection system for Internet of things," *Journal of Cloud Computing*, vol. 9, no. 1, 2020.

[21] M. B. Bahador, M. Abadi, and A. Tajoddin, "HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition," *Proceedings of the 4th International Conference on Computer and Knowledge Engineering, ICCKE 2014*, pp. 703–708, 2014.

[22] A. P. Kuruvila, S. Karmakar, and K. Basu, "Time series-based malware detection using hardware performance counters," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 102–112.

[23] W. Snyder. [Online]. Available: https://veripool.org/verilator/documentation/

[24] Enjoy-Digital, "Enjoy-digital/litex: Build your hardware, easily!" [Online]. Available: https://github.com/enjoy-digital/litex

[25] "Openhw group cv32e41p user manual." [Online]. Available: https://docs.openhwgroup.org/projects/openhw-group-cv32e41p/index.html

[26] A. Waterman, Y. Lee, D. Patterson, and K. Asanovi, "The RISC-V Instruction Set Manual v2.1," *2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings*, vol. I, pp. 1–32, 2012.

[27] A. Elkanishy, D. T. Rivera, P. M. Furth, A. H. A. Badawy, Y. Aly, and C. P. Michael, "FPGA-Accelerated Decision Tree Classifier for Real-Time Supervision of Bluetooth SoC," *2019 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2019*, no. 1, 2019.

[28] R. Choudhury, S. R. Ahamed, and P. Guha, "Efficient Hardware Implementation of Decision Tree Training Accelerator," *Proceedings - 2020 6th IEEE International Symposium on Smart Electronic Systems, iSES 2020*, pp. 212–215, 2020.