Microprocessors and Microsystems 120 (2026) 105236

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

SHAX: Evaluation of SVM hardware accelerator for detecting and preventing ROP on Xtensa

Adebayo Omotosho [a,*], Sirine Ilahi [b], Ernesto Cristopher Villegas Castillo [c], Christian Hammer [b], Hans-Martin Bluethgen [c]

[a] School of Business, Computing, and Social Sciences, University of Gloucestershire, The Park, Cheltenham, GL50 2RH, Gloucestershire, United Kingdom
[b] Faculty of Computer Science and Mathematics, University of Passau, Innstrasse 33, Passau, 94032, Bavaria, Germany
[c] Cadence Design Systems, Mozartstrasse 2, Feldkirchen, 85622, Bavaria, Germany

ARTICLE INFO

Keywords:
Security
Return-oriented programming
Support vector machines
Field programmable gate array
Xtensa Reliability
Fault injection
Hardware accelerator
Instruction set simulator

ABSTRACT

Return-oriented programming (ROP) chains together sequences of instructions residing in executable pages of the memory to compromise a program's control flow. On *embedded systems*, ROP detection is intricate as such devices lack the resources to directly run sophisticated software-based detection techniques, as these are memory and CPU-intensive.

However, a *Field Programmable Gate Array* (FPGA) can enhance the capabilities of an embedded device to handle resource-intensive tasks. Hence, this paper presents the first performance evaluation of a Support Vector Machine (SVM) hardware accelerator for automatic ROP classification on Xtensa-embedded devices using hardware performance counters (HPCs).

In addition to meeting security requirements, modern cyber–physical systems must exhibit high reliability against hardware failures to ensure correct functionality. To assess the reliability level of our proposed SVM architecture, we perform simulation-based fault injection at the RT-level. To improve the efficiency of this evaluation, we utilize a hybrid virtual prototype that integrates the RT-level model of the SVM accelerator with the Tensilica LX7 Instruction Set Simulator. This setup enables early-stage reliability assessment, helping to identify vulnerabilities and reduce the need for extensive fault injection campaigns during later stages of the design process.

Our evaluation results show that an SVM accelerator targeting an FPGA device can detect and prevent ROP attacks on an embedded processor with high accuracy in real time. In addition, we explore the most vulnerable locations of our SVM design to permanent faults, enabling the exploration of safety mechanisms that increase fault coverage in future works.

1. Introduction

Return-oriented programming (ROP) is one of the most widely used techniques for maliciously reusing code and creating attacks from benign instructions. ROP attacks are particular injection attacks that reuse instructions already in an application's executable memory rather than injecting new instructions into memory. Among other data ROP injects the memory addresses of existing short instruction sequences (called *gadgets*) into a location that determines the instruction pointer at some point in the (near) future. The most common approach passes a maliciously crafted payload (containing addresses of gadgets) to a vulnerable function in the target application, which (due to a buffer overflow) overwrites adjacent memory locations containing that function's return address. This scenario results in memory corruption that upon return of that function forces the application to redirect control flow to the attacker's selected gadgets, creating a chain of attack whereby each gadget terminates with an instruction that diverts control flow to the next (like *return*). As gadgets in an application's code base are usually Turing complete, an attacker can virtually perform any computational task with a sequence of gadgets [1]. To date, injection attacks are among the top most critical security concerns according to the Open Web Application Security Project [2]. Moreover, this attack is hazardous as it can stealthily breach confidentiality (e.g., leaking of sensitive data from memory), integrity (e.g., alter program execution, modifying data or control flow), and availability (e.g., hijack execution to crash the program) [3].

Existing standard mitigation techniques such as *address space layout randomization* (ASLR) and *control flow integrity* (CFI) have been helpful

* Corresponding author.
E-mail address: bomotosho@glos.ac.uk (A. Omotosho).

https://doi.org/10.1016/j.micpro.2025.105236
Received 22 November 2024; Received in revised form 17 October 2025; Accepted 1 December 2025
Available online 4 December 2025

in severing memory corruption on regular PCs but cannot be directly used on embedded devices because these countermeasures are generally resource-intensive protection schemes [4,5]. An alternative method to strengthen the security of embedded devices is to delegate resource-demanding mitigation detection tasks to specialized hardware such as a field programmable gate array (FPGA) [6]. Hardware accelerators using FPGAs are reconfigurable and commonly used to parallelize resource-intensive tasks and provide high throughput while consuming low energy/power compared to traditional general-purpose processors (CPUs) [7,8]. This advantage makes FPGAs useful for empowering resource-constrained embedded devices.

Another critical aspect that modern Cyber-Physical Systems (CPS) must address, alongside security, is reliability, which ensures robustness against hardware faults and guarantees correct functionality, especially in mission-critical applications. These faults often arise from external influences and environmental conditions. Given the high complexity of such systems, assessing fault-tolerance in architectures like Machine Learning architectures typically requires costly simulation-based fault injection campaigns to uncover vulnerabilities that may necessitate design modifications to enhance system robustness [9]. As this process is inherently iterative and demands extensive verification, current methodologies are increasingly shifting toward virtual prototyping. This enables early fault-tolerance evaluation, significantly optimizing time-consuming design and verification cycles while ensuring the final product meets reliability specifications [10]. In this work, we propose a hybrid virtual prototype model for early exploration of vulnerabilities in an RT-level SVM implementation. Our approach predicts reliability metrics such as fault coverage for permanent fault types (e.g., *stuck-at-0/1*). It is important to note that our simulation-based fault injection is solely intended for reliability assessment and is unrelated to ROP attack injection, which pertains to security. Both security and reliability are essential properties commonly evaluated during the industrial development cycles of modern CPS, such as those in the automotive domain, to achieve cybersecurity certification [11] and functional safety compliance [12].

Machine learning algorithms are efficient at finding patterns in data, and an FPGA's parallel architecture makes it suitable for machine learning applications [6]. Most importantly, SVMs are considered by experts as an excellent binary classifier because they are effective in high-dimensional spaces and robust to overfitting, making them well-suited for a wide range of classification tasks, including those with complex decision boundaries and large feature sets [13]. Likewise, hardware performance counters (HPCs) are well-known special-purpose registers initially designed for debugging hardware during development [14,15]. However, developers repurpose them to measure the micro-architectural changes that reflect a program's behavior at a low cost [16,17]. Hence, we present the application of the first hardware acceleration of support vector machines (SVM) on an FPGA that automatically detects and prevents ROP attacks using HPCs on an Xtensa embedded processor.

1.1. Motivation

Memory corruption attacks leading to code reuse are among the most dangerous threats to embedded systems and IoT devices. Due to the limited computational capabilities of these devices, it is impractical to directly deploy well-known attack mitigations such as ASLR and CFI because these techniques require substantial resources, such as memory management units, and impose high runtime overhead [5,18,19].

Our mitigation strategy exploits the Xtensa processor, a reconfigurable processor exclusively designed for embedded devices. This processor is commonly present in low-cost Wi-Fi microchips, including those used in the ESP8266, ESP32, LX6, and LX7. One of the several recent motivating examples of a memory attack on Xtensa was on MediaTek audio DSP which is present in 37% of all smartphones and IoT devices worldwide [20]. The CVE-2021-0661, CVE-2021-0662, and CVE-2021-0663 [21–23] were all due to out-of-bounds-write due to incorrect bounds checking exploited by a ROP attack. Hence, buffer overflow is still one of the most exploited memory access bugs for ROP attacks in embedded firmware [24,25].

Also, hardware-assisted defense techniques are increasingly gaining popularity in research. Despite CFI being computationally expensive on embedded devices [24], hardware-assisted techniques using CFI-based accelerators have been the focus of extensive research [24, 26–28]. However, implementing CFI techniques in embedded systems remains particularly challenging due to their limited processing power, memory, and energy resources. These constraints make it challenging to deploy CFI techniques that require significant computational overhead [29]. Moreover, the necessity for debug symbols further complicates the implementation process [28]. On the other hand, hardware performance counters (HPCs) are a proven alternative for detecting illegal program modification, especially on the x86 architecture [30, 31]. Therefore, this paper presents an evaluation of a hardware-assisted approach using an SVM hardware accelerator with HPCs to detect and prevent ROP attacks in real time and assess its reliability, aiming to determine the adequacy of this approach.

Threat model: Our targets are embedded devices with firmware that runs directly from flash memory. We assume the firmware is written in C and contains memory vulnerabilities that a buffer overflow can exploit through itself. Our toolchain can detect arbitrary code reuse resulting from return-oriented programming attacks by feeding the HPCs of a running program to a hardware-accelerated SVM running on an FPGA. The accelerator parallelizes the task, making detecting attacks in real time possible. In our prototype, an output signal is sent via a blinking LED when an attack is detected. This indicator can be fed into other intelligent decision support systems to decide whether to halt or continue the execution of suspicious processes.

The key contributions of our paper are:

- we evaluate SHAX, leveraging a custom SVM hardware accelerator [32], for *real-time* detection and prevention of ROP attacks in flash memory programs using HPCs.
- we develop a hybrid virtual prototype consisting of a Xtensa Instruction Set Simulator (ISS) and SVM's RTL hardware accelerator for improving the simulation performance and providing a flexible design test-bench, which requires a complex setup on an FPGA platform.
- we use Cadence®Xcelium™Fault Simulator (XFS) tool for fault injection to carry out a fault-tolerances evaluation of SHAX by injecting permanent faults (i.e., Stuck-at 0/1) at different levels (e.g., RTL structure and port interfaces) in the virtual prototype to measure the model's resilience to permanent faults.
- we obtain an average real-time attack detection accuracy of 80% for programs running from the flash memory. We also achieve a hardware model fault coverage that varies between 65% (*on full RTL*) to 100% (*on AXI data*). Furthermore, our online detection incurs a very moderate performance overhead.

1.2. Research questions

This research focuses on the performance evaluation of our SVM-based hardware accelerator for detecting ROP attacks. Hence, our investigation is guided by the following research questions.

- **RQ1:** What detection accuracy can our accelerator achieve for real-time ROP attacks?
- **RQ2:** How does the performance of the hardware kernel compared to software implementation?
- **RQ3:** What is the reliability level of a hardware accelerator when exposed to fault scenarios induced by environmental disturbances?

Table 1
Comparison with other hardware-assisted memory anomaly detection approaches.

Author	Hardware method	FPGA	Accuracy	Overhead	Operating systems	Reliability evaluation
[35]	CFI	Yes	–	–	Linux	–
[28]	CFI	–	–	–	–	–
[24]	CFI	Yes	–	9.77%	Linux	–
[27]	PUF	–	–	0.95%	–	–
[34]	CFG	Yes	–	–	–	–
Our paper	HPC	Yes	86%	4.41%	–	Yes

The remainder of this paper is organized as follows: Section 2 discusses related work on anomaly detection accelerators, ROP on Xtensa, hardware performance counters, and virtual prototypes, and compares our work to recent research. Section 3 describes the design and implementation strategies used in this paper. Section 4 presents the evaluation results and discusses them in relation to the research questions. Finally, Section 5 provides a summary of the paper.

2. Related work

Anomaly Detection Accelerators: There are limited works on accelerator approaches for intrusion detection or prevention, and the majority of the works we found focus mainly on CFI. A comparative study by De Clercq et al. [26] shows that hardware-based CFI has been extensively studied. However, this approach typically requires extending a processor's instruction set architecture and modifying the compiler [27]. For instance, Davi et al. [28] proposed a hardware-assisted CFI for ROP detection using a per-function label approach and a state model where active labels, currently executing functions, are maintained in a dedicated memory area. Zhang et al. [27] introduced a hardware-assisted CFI that employs encrypted Hamming distances matching and linear encryption/decryption operations based on Physical Unclonable Function (PUF). Their tool achieved a 0.95% runtime overhead. Oh et al. [24] proposed *ActiMon*, a code reuse attack monitor implemented on system-on-chip FPGA, where both the host CPU and FPGA are on a single platform. ROP detection in *ActiMon* was based on a list of active functions (invoked but not yet returned) using CFI enforcement techniques such as branch regulation and shadow stack, incurring a 9.77% performance overhead on the host CPU. Mohsin et al. [33] also developed an FPGA-based accelerator for K-Nearest Neighbor classification, demonstrating that machine learning algorithms can be accelerated and the performance can be faster than a software implementation. This approach obtained over *100X* speeds up applications running on the FPGA. Given the computational expense of CFI on embedded devices, our paper is the first to explore hardware-assisted detection of ROP on an FPGA using an HPC alternative. Rahmatian et al. [34] use a control flow graph (CFG) with FPGA to detect control flow violations caused by attacks such as buffer overflows, rather than implementing full control flow integrity; however, the system's detection capability was not evaluated.

On other non-ROP hardware-assisted detection models, RD-FAXID [36] targets ransomware detection with FPGA-accelerated XGBoost using HPCs from data collected on Linux and Windows OS, achieving detection accuracies between 78.31% and 99.52%. APPARENT [37] reported accuracies between 42% and 92.7% for anomaly detection using HPCs. LightBench [38] is a hardware- and software-aware trusted execution platform for intelligent malware detection at the edge. It also uses HPCs in TinyML-compatible pipelines and achieved 60.86% and 97.7% detection accuracy for SVM. Makrani et al. [39] investigated accelerated machine learning for on-device hardware-assisted cybersecurity in edge platforms using HPC data and achieved a detection accuracy of 82% with a runtime overhead of 15%.

ROP on Xtensa: Compared to ARM and x86 architecture, ROP attacks on Xtensa have only been rudimentarily investigated and attacks on its Call0 ABI and windowed register ABI are still being researched [40–43]. In the only practical study on ROP detection on Xtensa using HPCs, eight events were used, and an average offline model accuracy of 79% was reported [43]. In contrast, our accelerator uses only four HPCs and achieves an improved average detection accuracy of 86% online and 95% offline on similar benchmark applications. In addition, we used a hardware-friendly kernel that is faster than the time-consuming standard RBF kernel.

Hardware Performance Counters: HPCs are special registers found in almost all modern microprocessors that can measure architectural events at CPU level at full hardware speed and with minimal overhead [15]. In the x86 architecture, HPCs have proven to profile abnormal execution of short instruction sequences (gadgets in ROP) with high precision [16]. However, the number of available HPC registers and events differs between processors, irrespective of whether they belong to the same family. Likewise, the number of HPC registers determines the number of events that can be simultaneously measured. For instance, modern Intel processors, ARM Cortex-A5, ARM Cortex-A8, SPARC T4, and Xtensa LX7, have 2, 4, 4, 4, and 8 HPC registers, respectively, which means that only a tiny fraction of the total number of events can be measured simultaneously. Weaver et al. [15] studied the consistency of HPC events across various x86 implementations, revealing approximately 1.07% differences. On x86, Pfaff et al. [44] proposed a Linux kernel patch that uses four HPC events to predict ROP attacks on x86 systems. Unlike our paper, their approach requires kernel modification to detect ROP. Malone et al. [30] used five HPC events, chosen based on predefined criteria, to perform program integrity checks on x86 running Linux. Our technique differs as it does not require an OS, thereby not incurring additional OS-level HPC noise. Singh et al. [45] used sixteen HPCs collected from a virtual machine with Intel's VTune for rootkit detection on Windows 7. In contrast, our study collected HPCs on a bare-metal Xtensa processor, employed four HPC events, and implemented a machine learning and hardware-driven detection approach.

Virtual Prototypes: Virtual prototypes can serve as a platform for software development, architectural modeling, and system implementation, providing a detailed simulation as a final prototype [46]. For example, security testing on virtual prototypes for networked embedded systems was performed by Mahmoodi et al. [47], who proposed to use them at different levels of simulation to bridge the gaps between the already designed and developed system modules and the final prototypes. Pieper et al. [48] also presented an approach for early and accurate security analysis of embedded binaries using SystemC-based virtual prototyping. We targeted this approach to detect security policy violations at runtime. In addition, Omotosho et al. [49] developed and demonstrated the Xtensa ISS, which evaluates the performance of its emulated HPCs with those from a real Xtensa hardware and whether they accurately simulate the realistic values exemplified via detecting code reuse attacks. Their experimental results show that the micro-architectural characteristics obtained with a virtual prototype, on average, are about 70% similar, providing developers and testers with early simulation opportunities. In our paper, we use virtual prototypes to investigate the fault-tolerance of SHAX by injecting faults into the accelerator. Table 1 summarizes the difference between our work and other hardware-assisted anomaly detection papers.

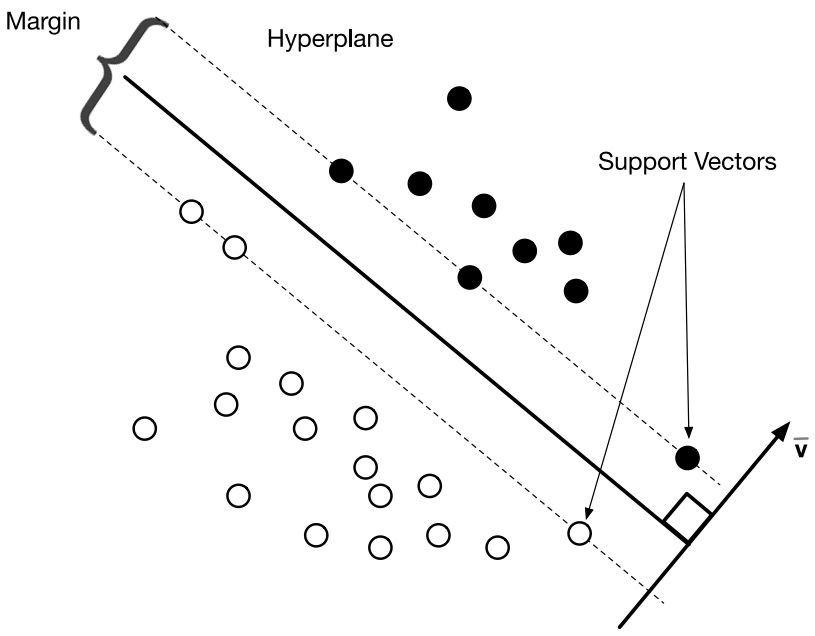


Fig. 1. SVM Classification.

3. Planning and execution methods

3.1. Support vector machine

SVM binary classification works by finding the optimal hyperplane that maximizes the margin between two classes in a multidimensional feature space as shown in Fig. 1. Given a labeled dataset (x_i, y_i), where x_i represents the feature vector and $y_i \in -1, +1$ represents the class labels. SVM aims to find a hyperplane $\bar{v} \cdot x + b = 0$ such that the data points from both classes are separated with the largest possible margin. The margin is the distance between the hyperplane and the nearest data points from either class, called support vectors. A perpendicular vector, $\bar{v}$ to the hyperplane can be used to characterize a hyperplane such that the location of an unknown vector $\bar{u}$ can be determined using: $\bar{v} \cdot \bar{u} + b \geq 0$ such that if $\bar{v} \cdot \bar{u} \geq -b$, then the unknown vector $\bar{u}$ is one the right side of the margin otherwise it is on the left.

$\bar{v}$ vector is usually computed as

$$\bar{v} = \sum_i \alpha_i y_i \bar{x}_i \tag{1}$$

Where α_i is the Lagrange multiplier from the training data, x_i denotes the ith training sample, and y_i the associated label (−1: negative samples, +1: positive samples), $\bar{u}$ is an unknown vector to be classified and b is the function bias.

SVM supports different types of kernel functions, but the Radial Basis Function (RBF) kernel also known as Gaussian kernel is the most popular non-linear kernel used in SVM. This kernel is effective for handling non-linearly separable data and is widely used in classification and regression tasks. The RBF kernel is defined as:

$$F(\overline{x_i}, \bar{u}) = e^{-\gamma \|\overline{x_i} - \bar{u}\|^2} \tag{2}$$

where γ controls the influence of individual data points, a larger γ value leads to a more complex decision boundary, while a smaller γ smoothens it. F is a function computing the dot product of $\overline{x_i}, \overline{u_i}$ in a nonlinear space, and it is the main function or kernel to be substituted.

Our experiment is based on a scalable SVM hardware architecture provided [32,50]. *Hardware-friendly kernel* is a lightweight version of the standard RBF kernel that can be realized in hardware without multipliers; only the decision function of the SVM is implemented in the accelerator. Given the general kernel decision function:

$$\sum_i \alpha_i y_i F(\overline{x_i}, \bar{u}) + b \geq 0 \tag{3}$$

By replacing the kernel function of the standard RBF kernel function ($e^{-\gamma \|\overline{x_i} - \bar{u}\|^2}$) with the hardware-friendly kernel function $2^{-\gamma \|\overline{x_i} - \bar{u}\|_1}$ the decision function becomes:

$$\sum_{i=1}^{n} \beta_i y_i 2^{-\gamma \|\overline{x_i} - \bar{u}\|_1} + b \geq 0 \tag{4}$$

The β is just a scaled version of α expressed using a precision bits, the Lagrange multiplier resulting from solving the quadratic optimization problem during the training. $\|.\|_1$ is the Manhattan norm, substituting the Euclidian norm from the RBF kernel.

For design validation, on sklearn breast cancer dataset the implementation runs at 175MHz and takes 11.8μ s to process 4096 support vectors with 15 features and 12 bit precision. It requires 32 blocks of random access memory (BRAM) and around 4000 lookup tables within 1325 slices. Also, the model will need 2063 clock cycles to calculate the result, which approximates about 11.8μ s per classification result. The power consumption is about 1.86 W. Our ROP applications use maxium of 1397 support vectors and 8 features and the measured latency provides a valid upper-bound estimate of system performance.

Fig. 2 shows the block diagram of the SVM. Each macrocell is responsible for calculating a part of the support vectors and calculates either exclusively positive or negative support vectors in order to save the sign. By adjusting the number of macrocells, models of different size can be realized. Fig. 3 shows how a macrocell accommodate different model parameters. An AXI bus is used instead of the SPI interface, as this allows a higher throughput to be achieved and Xtensa does not provide an SPI interface. The support vectors used in the model are stored in the BRAM memories. The computation required for the hardware-friendly kernel is divided into two steps. First, the input vector and support vectors are normalized. This is done iteratively in the normalization block. The number of cycles required is determined by the number of features in the model. A corresponding number of normalization calculations are performed simultaneously so that the subsequent CORDIC pipeline can process a new normalized vector in each clock cycle. The transition is made by a shift register that can be written to in parallel by all normalization blocks and delivers a new

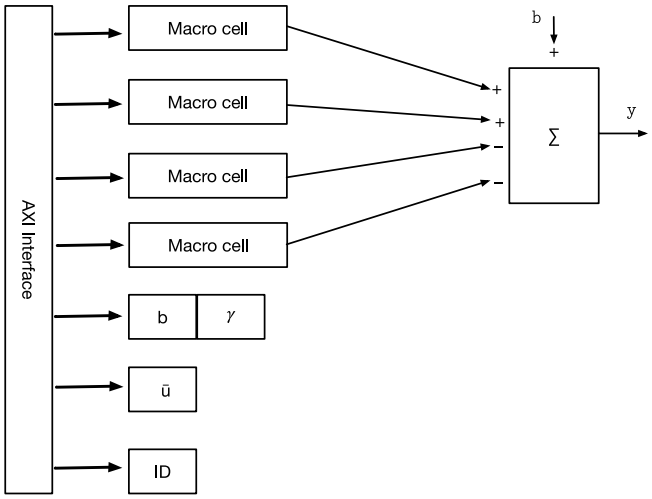


Fig. 2. Block diagram of the SVM architecture.

value to the pipeline in each clock cycle. The CORDIC pipeline performs an iterative calculation of the hardware-friendly kernel. The length of the CORDIC pipeline is limited by the accuracy (number of bits) of the features. At the end, all results are summed up and propagated to the outside. Outside of the macrocell, the individual intermediate results of the macrocells are summed up again with the corresponding sign to obtain the overall result (i.e y in Fig. 2)

3.2. How ROP attacks work on xtensa

Fig. 4 is a simplified example of how our designed ROP attacks work on Xtensa. This example copies 200 bytes of payload from *src* to *dst*, similar to the CVE-2021-0661 exploitation in MediaTek DSP. For ROP attacks to work normally, we must be cautious so that the payload skips the function epilogue (usually the first two instructions). This example uses three gadgets @g1 ...@g3; @g1 loads the source `src` and destination `dst` addresses and then returns to @g2. Gadget @g2 loads the data size (`0xC8`) to be copied and returned to gadget @g3. @g3 jumps to the address of `memcpy` which uses the three attacker-supplied parameters, a2 ... a4. Usually, `ret` will find the corrupt return addresses the payload has overwritten immediately below the buffer, as frame pointers are optional. Also, this described approach only works for the Xtensa Call0 application binary interface.

3.3. Test applications and model training

The hardware setup consists of a Xilinx Zynq XCZ7020-1CLG484C7[1] System on Chip (SoC) module mounted on a TE0703-068[2] carrier board. We implemented the SoC design using the Vivado Design Suite HL WebPACK Edition. Both the accelerator and Xtensa configuration are integrated on the Xilinx FPGA. Our performance evaluation monitors ten embedded programs, derived from MiBenchmark [51] and Omotosho et al. [43]. The applications are: BFS, Kruskal (KRU), RabinKarp (RKP), Huffman (HUF), Mergesort (MST), LCS, Prim (PRM), BinaryS (BIS), FloydWarshall (FLW), BellmanFord (BFD). These programs are also recently used in Xu et al. [52] for ROP analysis. The programs are written in C language and we recompiled them with *xt-xcc*, an optimized C Compiler for the Xtensa processor, targeting code optimization and size reduction [53]. Each application is executed multiple times on the hardware under both ROP and normal conditions, and the microarchitectural characteristics of the programs are collected.

Table 2
Xtensa HPCs.

Label	HPC events	Description
H1	PIPELINE_INTERLOCKS	Pipeline interlocks cycles
H2	D_STORE_U1	Data memory store instruction
H3	D_LOAD_U1	Data memory load instruction
H4	D_TLB	Data TLB accesses
H5	EXR	Exceptions and pipeline replays
H6	COMMITTED_INSN	Instructions committed
H7	BRANCH_PENALTY	Branch penalty cycles
H8	CYCLES	Count cycles

A dataset containing 6558 instances of HPC data records with eight HPC events as features is used to train the SVM models using the hardware-friendly kernel. The binary classifier labels are defined as -1 (non-ROP) and 1 (ROP). The model was trained in Python, and the resulting custom parameters were exported to the hardware to configure it for classification. The full set of HPC events, H1...H8, is shown in Table 2

These HPC events are relevant for ROP detection. For example, H1 (PIPELINE_INTERLOCKS) can indicate ROP attacks, as the frequent execution of short gadgets often causes processor pipeline stalls. H2 (D_STORE_U1) is relevant, as ROP gadgets typically perform few memory write operations, resulting in abnormal store patterns. H3 (D_LOAD_U1) is relevant, as ROP gadgets usually execute minimal memory reads, producing abnormal load patterns. H6 (COMMITTED_INSN) can reveal ROP attacks, as the execution of short gadgets often reduces the typical instruction commit patterns.

Fig. 5 illustrates the high-level blocks of tasks performed by SHAX. The SVM accelerator, a crucial component of SHAX, automatically makes a prediction based on the HPCs of a running program. Each application runs infinitely in a loop and can perform malicious tasks with gadgets without crashing. This approach was intentionally taken to assess whether a subtle change to a running program could be detected. The HPCs are periodically collected every 10,000 cycles and fed into the SVM for real-time detection and prevention. This prediction is a key step in detecting potential ROP attacks. Additionally, we developed C language modules to perform onboard data preprocessing tasks. The processed HPC data is fed into the hardware-friendly kernel SVM, which then performs the classification as ROP or benign. In our prototype, an output signal is sent via a blinking LED when an attack is detected, indicated by output 1. When the output is −1 (benign), the program keeps running, and this process continues throughout the execution of the program. This indicator can be fed into other intelligent decision support systems to decide whether to halt or continue the execution of suspicious processes.

3.4. Virtual prototype implementation

In order to efficiently simulate embedded software applications related to the SVM, we implemented a hybrid virtual prototype, shown in Fig. 6, whose components are the following:

- Xtensa LX7 ISS implementing HPCs required for ROP attack detection.
- Xtensa LX7 ISS connected to the TLM2.0 router via a bidirectional Initiator/Receiver TLM2.0 sockets binding.
- SystemC-to-RTL transactor enabling the conversion of a TLM2.0 general payload from the TLM2.0 router to the Advanced eXtensible Interface (AXI) protocol RT-level signals received by the SVM model.

[1] https://wiki.trenz-electronic.de/display/PD/TE0720+Resources
[2] https://wiki.trenz-electronic.de/display/PD/TE0703+Resources

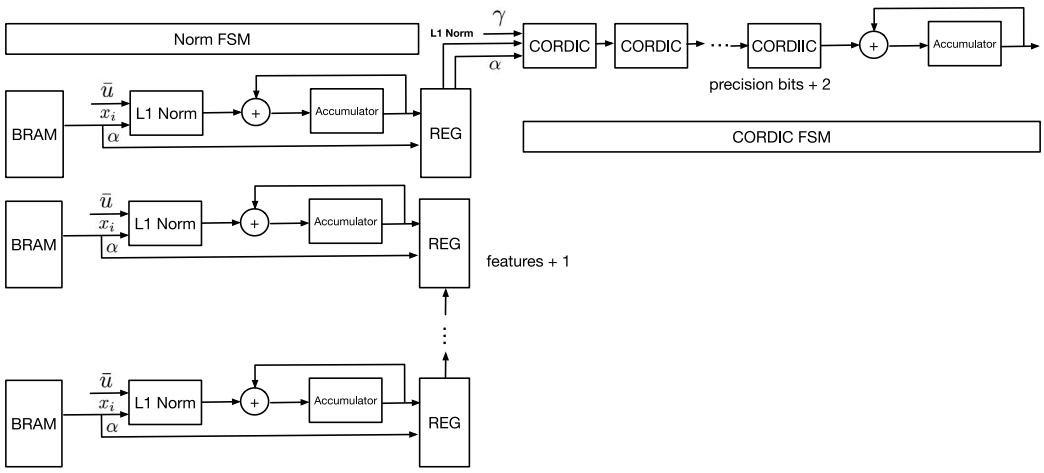


Fig. 3. Block diagram of a macrocell of the SVM design.

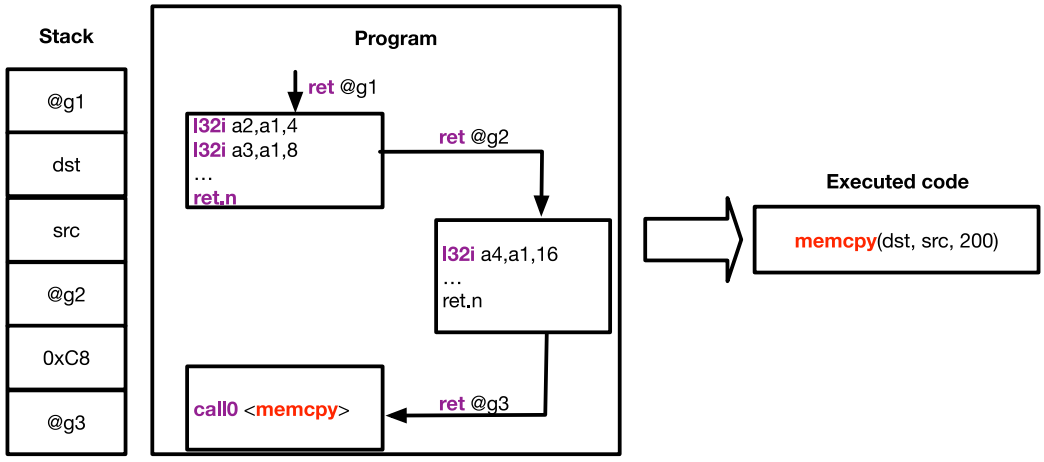


Fig. 4. Xtensa ROP [49].

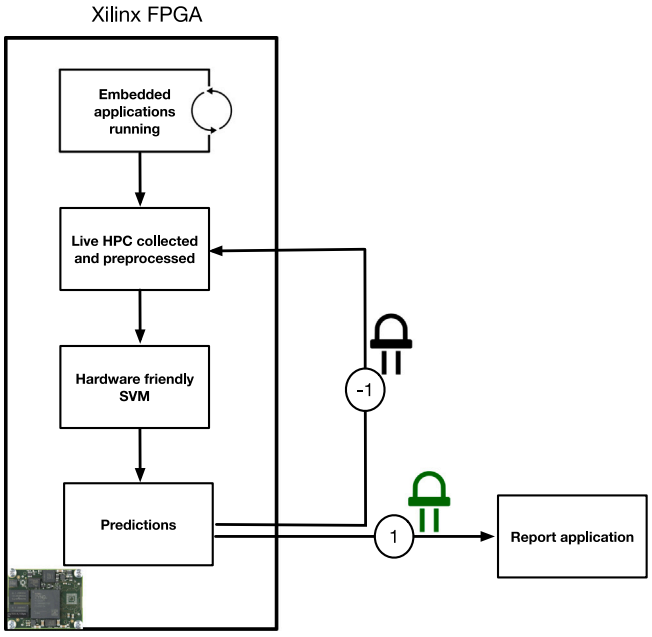


Fig. 5. SHAX.

- RTL-to-SystemC transactor enabling the SVM interruption signal (`intr_req`) conversion from RT-level `wire` to `sc_signal` received by Xtensa LX7 ISS port. Both transactors are encapsulated in the same `sc_module` class (i.e, green block of Fig. 6).
- SVM RTL accelerator connected to both transactors through AXI ports and interrupt signal.

In order to simulate the SystemC/TLM2.0 kernel [54] together with the SVM RTL, we created a SystemC wrapper around the RTL2SystemC transactor and the SVM RTL. This wrapper inherits a class from Cadence's Xcelium©SystemC extension library to associate RTL modules with a SystemC design. Therefore, both simulation kernels will synchronize the TLM2.0 transactions and the *intr_req*.

The virtual prototype testbench generates the clock signal `clk` at 250MHz along with the corresponding reset signal for the SVM hardware. The Xtensa ISS executes the embedded software responsible for configuring the SVM RTL model. This software trains the SVM by supplying the required parameters, which are subsequently stored in the internal BRAMs.

3.5. Fault injection tool and setup

Our experiments' selected fault model types are Stuck-at 0 (SA0) and Stuck-at 1 faults (SA1). Since permanent faults are commonly present on SVM architectures, we only considered them. Transient faults will be considered in future works.

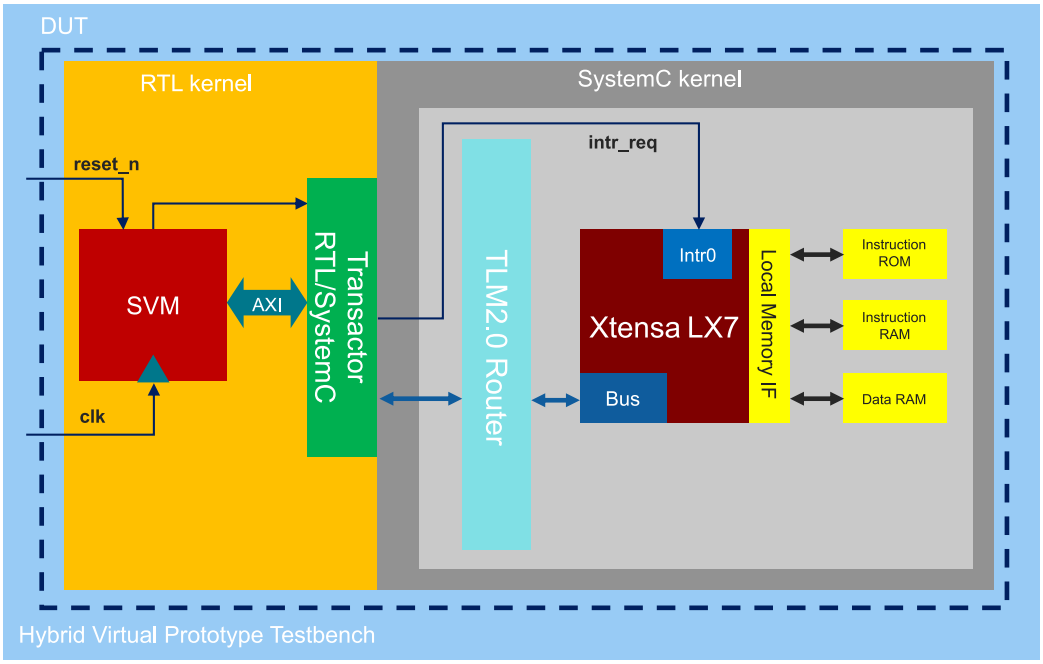


Fig. 6. Hybrid virtual prototype architecture composed of SVM RTL and Xtensa LX7BC ISS.

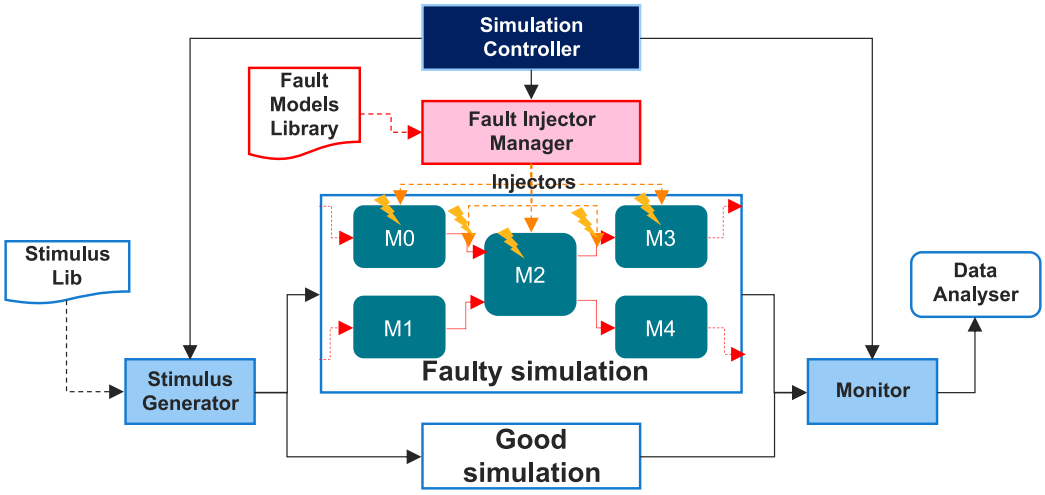


Fig. 7. Fault injection methodology.

We applied the fault injection methodology shown in Fig. 7, where all faulty simulations are compared to a good simulation to determine the effect of the injected fault in the virtual prototype. We use Cadence Xcelium©Fault Simulator (XFS) to perform the fault injection in all possible locations of the SVM architecture.

To observe the fault effect in the SVM architecture delivered service, we set strobes (also known as monitors) at the SVM's output service, i.e., AXI data port and the SVM interrupt request. The main vulnerabilities in the SVM architecture were identified by classifying the fault injection locations into four experimental sets:

- AXI address: whose fault targets are all bits of `axi_addr` port
- AXI data: whose fault targets are all bits of `axi_data` port
- SVM port interface: all interface ports from SVM RTL top hierarchy
- Full FI campaign: all possible fault targets inside SVM RTL

3.6. Evaluation metrics

3.6.1. Accuracy

To answer research questions RQ1 and RQ2, we evaluate our accelerator based on the prediction accuracy of the SVM classifier. Mathematically, the *Accuracy* formula in Eq. (5) computes the classifier's overall performance as the ratio of correctly predicted instances to the total number of instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

In Eq. (5), TP, FP, TN, and FN refer to true positive, false positive, true negative, and false negative, respectively.

3.6.2. Fault coverage

The third research question (RQ3) is addressed by employing our FI tool to introduce faults into the RTL structure of the SVM executing the ROP attack detection workload. Fault coverage is then computed

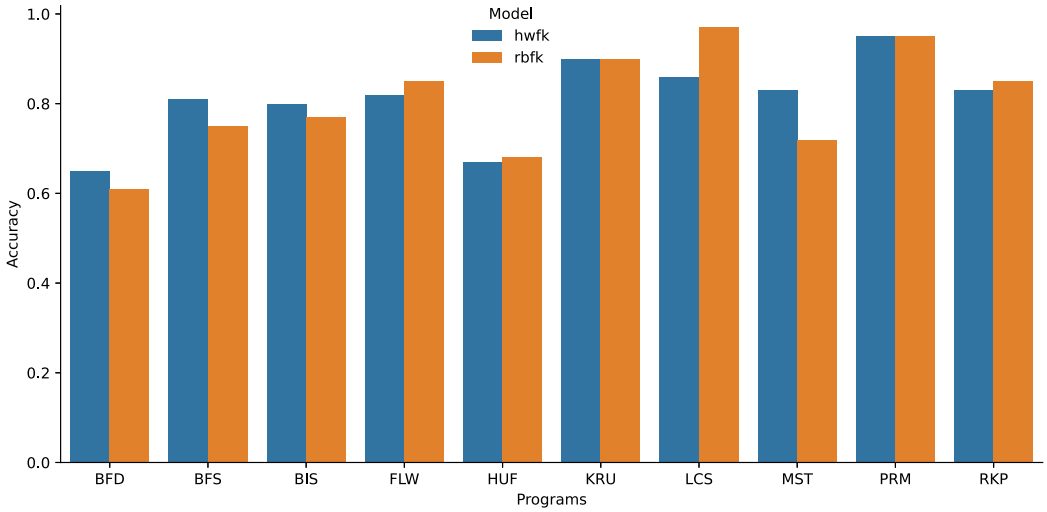


Fig. 8. Offline accuracies of hwfk and rbfk.

to quantify the proportion of injected faults that result in observable deviations in the output service provided by the target architecture. A detailed discussion of the Fault Coverage (FC) formula and its computation is presented in Section 4.3. Our injection targets are the AXI address, AXI data, top port interface, and the full RTL.

4. Discussion

This section answers the research questions highlighted in Section 1.2.

4.1. RQ1: what detection accuracy can our accelerator achieve for real-time ROP attacks?

Table 3 shows the results of the live detection and prevention of ROP attacks by SHAX using the hardware-friendly kernel. The online experiment uses a trained model (*m1*) with ≈ 95% accuracy that uses four HPC events (H1 ... H4) from Table 2.

Two additional models were trained, one using the same H1 ... H4 as the main model but with adjusted model parameters and number of support vectors (*m2*), and another using 8 HPCs, H1 ... H8 (*m3*), mainly for fault coverage assessment, as we are unsure if the model complexity affects SHAX's reliability.

During online detection, only three false alarms (false positives) were raised for one application *KRU* and none for the others, meaning that most positive attacks can be correctly prevented at runtime. When we investigated the results further, the online accuracies differed from the offline classification results. This is because, unlike in offline detection where event statistics appear to be deterministic, online event readings are not. Nevertheless, our automation detects attacks with significant confidence. However, our method incurs an average runtime overhead of ≈ 4.41%.

4.2. RQ2: how does the performance of the hardware kernel compared to software implementation?

Using identical model parameters, the hardware-friendly kernel has performance comparable to the RBF kernel in terms of detection accuracy, as shown in Fig. 8, where *hwfk* and *rbfk* denote the hardware-friendly kernel and the RBF kernel, respectively. The results in Fig. 8 are based on offline or pre-recorded counter profiles of ROP and benign executions. Across the 10 applications, the hardware-friendly kernel matches or outperforms the RBF kernel in 60% of the applications. Only in one application (LCS) does the RBF kernel (97%) clearly outperform the hardware-friendly kernel (86%). Even so, the performance of the approximate model remains acceptable.

Table 3
Real-time detection of ROP by SHAX.

Appl.	#False alarms	Detection accuracy	Execution (*per iteration*)	Slowdown
BFD	0	72.88%	478 977.41	1.69%
BFS	0	90.20%	132 186.356	1.31%
BIS	0	96.00%	853 750.21	7.41%
FLW	0	72.46%	143 088.681	4.35%
HUF	0	84.53%	103 429.314	3.09%
KRU	3	76.14%	116 863.702	4.55%
LCS	0	90.32%	875 212.72	6.45%
MST	0	89.74%	916 194.75	5.13%
PRM	0	93.65%	859 841.49	3.17%
RKP	0	93.10%	864 584.60	6.90%

4.3. RQ3: what is the reliability level of a hardware accelerator when exposed to fault scenarios induced by environmental disturbances?

This question is addressed via the SVM model fault coverage measurement using fault injection. The obtained results are divided into four types *Detected (D)*, and *Undetected (U)*, across three SVM model configurations.

The output type *Detected* (D) represents all faults propagated to the strobes at the SVM output service with a different logical value from the good simulation. The *Undetected* (UD) type represents all the faults whose effects cannot be perceived at the selected strobes since they got the same logical value as in the good simulation. *Potentially Detected* (PD) type represents all faults that propagate to the strobes, yet their values are undefined or *do not care* "X", in contrast to the *good simulation*. Finally, the *Not Injected* (NI) type represents the faults that were skipped to injected due to simulation crashes (e.g., failure on license checkouts, process failures, etc.). These two faults can be ignored as they do not contribute to our analysis. We calculate the FC using Eq. (6).

$$FC = \frac{D}{D + UD} \tag{6}$$

The configurations used by the three models *m1*, *m2*, and *m3* are identified by *Config 1*, *Config 2*, and *Config 3* respectively. The *Config 1* model utilizes 582 support vectors and four features; *Config 2* employs 1284 support vectors with the same number of features; and *Config 3* incorporates 1397 support vectors and eight features. These configurations serve to validate the FC of the SVM-based model across varying structural complexities. Specifically, the FC of the RTL structure is 65.63%, 66.54%, and 65.66% for *Config 1*, *Config 2*, and *Config 3*,

Table 4
Fault injection results.

Injection targets	Config 1	Config 2	Config 3
AXI address	D(84); UD(14); FC(85.7%)	D(20); UD(14); FC(58.8%)	D(82); UD(15); FC(84.5%)
AXI data	D(64); UD(0); FC(100.0%)	D(68); UD(0); FC(100.0%)	D(67); UD(2); FC(97.1%)
Top port interface	D(239); PD(2); UD(53); FC(81.8%)	D(243); PD(2); UD(42); FC(85.2%)	D(240); PD(3); UD(55); FC(81.3%)
Full RTL	D(1753); PD(8); UD(918); NI(1); FC(65.6%)	D(1782); PD(8); UD(896); NI(2); FC(66.5%)	D(1752); UD(916); NI(12); FC(65.7%)

respectively. These results suggest that all three configurations demonstrate comparable robustness against permanent faults. Moreover, the data indicates that over 50% of the RT-level model's locations are vulnerable to faults, underscoring the importance of further analysis to identify critical hierarchical components that influence the SVM's reliability in mission mode. The results of the fault injection campaign are summarized in Table 4. The most resilient target is the AXI data bus, with FC values exceeding 97% across all configurations. The second most robust component is the SVM port interface, with FC ranging between 81% and 85%. The AXI address bus ranks third, showing the lowest FC (59%) in *Config 2*, while *Configs 1* and 3 maintain FC values around 85%, consistent with the previous components. Overall, the average FC across all configurations is approximately 65%. These FC levels are considered safety-critical, especially in domains such as automotive and aerospace, where system reliability is paramount. By utilizing the virtual prototype as a design space exploration framework, it becomes feasible to integrate hardware- or software-based safety mechanisms to improve fault resilience—without the need for a complex FPGA-based validation setup. A practical recommendation is to incorporate a parity checker on the AXI interface, which can significantly enhance fault detection capabilities and thereby increase overall system reliability. The AXI interface is particularly crucial, as it serves as the primary conduit for transferring training data to the SVM. Ensuring its integrity is essential for maintaining dependable operation in mission-critical environments.

4.4. Threats to validity

The following are the threats to the validity of our findings:

- HPC statistical measurements may differ between microprocessors. Since our experiments are performed solely on the Xtensa LX7, the HPC measurements presented in this paper are not guaranteed to yield the same performance results on other architectures.
- We performed the accelerated SVM model validation using benchmark programs with an average program opcode and size of about 11357 and 0.3MB, respectively. In reality, embedded firmware might be more complex. Nevertheless, our classification result shows the feasibility of accelerated SVM in detecting and preventing ROP on programs running from flash memory.

5. Conclusion

This paper presents an improvement in hardware-assisted anomaly detection by investigating a non-CFI-based approach, demonstrating the application of well-known application profiling characteristics (using HPCs) for detecting and preventing ROP attacks on embedded devices in real time. We show the capability of *SHAX*, an FPGA-based SVM accelerator, to automatically detect and classify ROP attacks based on microarchitectural characteristics recorded by performance counters and triggered through a hardware interrupt. Experimental results demonstrate the feasibility of this alternative hardware-assisted security method for embedded systems, achieving an average performance overhead of just 4.41%. Fault-tolerance assessment of the SVM hardware model was conducted using Cadence XFS, with faults injected at various interfaces, resulting in fault coverage ranging from 65% to 100%. Although achieving 100% fault coverage is often infeasible, maximizing coverage minimizes risks and boosts confidence in deployed systems, particularly in critical applications. While the current study focuses on Xtensa, future work will extend the evaluation to other CPU architectures to better examine the method's generalizability.

Declaration of competing interest

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

Acknowledgments

The authors would like to thank Marcin Aftowicz of IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany, for his contributions to the design of the accelerator.

Data availability

Data will be made available on request.

References

[1] A. Homescu, M. Stewart, P. Larsen, S. Brunthaler, M. Franz, Microgadgets: Size does matter in turing-complete return-oriented programming, WOOT 12 (2012) 64–76.
[2] OWASP, OWASP top ten, 2024, URL https://owasp.org/www-project-top-ten/ (Accessed: 2024-11-19).
[3] V. Kiriansky, C. Waldspurger, Speculative buffer overflows: Attacks and defenses, 2018, arXiv preprint arXiv:1807.03757.
[4] V. Lefils, G. Grimaud, J. Iguchi-Cartigny, EE-CFI: Externalized control flow integrity for embedded devices, in: International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Springer, 2017, pp. 321–329.
[5] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, G. Tsudik, C-FLAT: control-flow attestation for embedded systems software, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 743–754.
[6] R. Chen, T. Wu, Y. Zheng, M. Ling, MLoF: Machine learning accelerators for the low-cost FPGA platforms, Appl. Sci. 12 (1) (2021) 89.
[7] A. Shawahna, S.M. Sait, A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, Ieee Access 7 (2018) 7823–7859.
[8] K. Sano, S. Yamamoto, FPGA-based scalable and power-efficient fluid simulation using floating-point DSP blocks, IEEE Trans. Parallel Distrib. Syst. 28 (10) (2017) 2823–2837.
[9] F.R. da Rosa, L. Ost, R. Reis, Fault injection framework using virtual platforms, in: Design of Fault-Tolerant Systems, Springer, 2020, pp. 47–67, http://dx.doi.org/10.1007/978-3-030-55704-1_3, URL https://link.springer.com/chapter/10.1007/978-3-030-55704-1_3.
[10] F.V. de Freitas, M.V.M. Gomes, I. Winkler, Benefits and challenges of virtual-reality-based industrial usability testing and design reviews: A patents landscape and literature review, Appl. Sci. 12 (3) (2022) 1755.
[11] ISO, S.A.E. International, Road Vehicles - Cybersecurity Engineering, SAE International, 2021, ISO/SAE 21434:2021. http://dx.doi.org/10.4271/ISO/SAE21434.

[12] International Organization for Standardization, Road Vehicles – Functional Safety, ISO, Geneva, Switzerland, 2018, Second Edition of ISO 26262. URL https://www.iso.org/publication/PUB200262.html.

[13] M. Mohammadi, T.A. Rashid, S.H.T. Karim, A.H.M. Aldalwie, Q.T. Tho, M. Bidaki, A.M. Rahmani, M. Hosseinzadeh, A comprehensive survey and taxonomy of the SVM-based intrusion detection systems, J. Netw. Comput. Appl. 178 (2021) 102983.

[14] J. Arulraj, P.-C. Chang, G. Jin, S. Lu, Production-run software failure diagnosis via hardware performance counters, Acm Sigplan Not. 48 (4) (2013) 101–112.

[15] V.M. Weaver, S.A. McKee, Can hardware performance counters be trusted? in: 2008 IEEE International Symposium on Workload Characterization, IEEE, 2008, pp. 141–150.

[16] L. Uhsadel, A. Georges, I. Verbauwhede, Exploiting hardware performance counters, in: 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, IEEE, 2008, pp. 59–67.

[17] S.P. Kadiyala, A. Kartheek, T. Truong-Huu, Program behavior analysis and clustering using performance counters, in: Proceedings of the 2020 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security, 2020, pp. 1–9.

[18] T. Kawada, S. Honda, Y. Matsubara, H. Takada, TZmCFI: RTOS-aware control-flow integrity using trustzone for Armv8-M, Int. J. Parallel Program. 49 (2) (2021) 216–236.

[19] T. Mishra, T. Chantem, R. Gerdes, Survey of control-flow integrity techniques for real-time embedded systems, ACM Trans. Embed. Comput. Syst. (TECS) 21 (4) (2022) 1–32.

[20] T. Alsop, Smartphone application processor (AP)/system-on-chip (SoC) vendor shipment share worldwide in 2020 and 2021, 2023, URL https://www.statista.com/statistics/796887/smartphone-system-on-chip-market-share-by-vendor-worldwide/ (Accessed: 2024-11-19).

[21] MITRE, CVE-2021-0661, 2021, URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-0661 (Accessed: 2024-11-19).

[22] MITRE, CVE-2021-0662, 2021, URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-0662 (Accessed: 2024-11-19).

[23] MITRE, CVE-2021-0663, 2021, URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-0663 (Accessed: 2024-11-19).

[24] H. Oh, M. Yang, Y. Cho, Y. Paek, Actimon: Unified JOP and ROP detection with active function lists on an SoC FPGA, IEEE Access 7 (2019) 186517–186528.

[25] A. Bhattacharyya, A. Sánchez, E.M. Koruyeh, N. Abu-Ghazaleh, C. Song, M. Payer, {SpecROP}: Speculative exploitation of {ROP} chains, in: 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), 2020, pp. 1–16.

[26] R. De Clercq, I. Verbauwhede, A survey of hardware-based control flow integrity (CFI), 2017, arXiv preprint arXiv:1706.07257.

[27] J. Zhang, B. Qi, Z. Qin, G. Qu, HCIC: Hardware-assisted control-flow integrity checking, IEEE Internet Things J. 6 (1) (2018) 458–471.

[28] L. Davi, P. Koeberl, A.-R. Sadeghi, Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation, in: 2014 51st ACM/EDAC/IEEE Design Automation Conference, DAC, IEEE, 2014, pp. 1–6.

[29] G. Christou, G. Vasiliadis, E. Athanasopoulos, S. Ioannidis, Hard edges: Hardware-based control-flow integrity for embedded devices, in: A. Orailoglu, M. Jung, M. Reichenbach (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation, Springer International Publishing, Cham, ISBN: 978-3-031-04580-6, 2022, pp. 275–287, http://dx.doi.org/10.1007/978-3-031-04580-6_18.

[30] C. Malone, M. Zahran, R. Karri, Are hardware performance counters a cost effective way for integrity checking of programs, in: Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing, 2011, pp. 71–76.

[31] X. Wang, R. Karri, Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 35 (3) (2015) 485–498.

[32] M. Aftowicz, K. Lehniger, P. Langendoerfer, Scalable FPGA hardware accelerator for SVM inference, in: 2022 11th Mediterranean Conference on Embedded Computing, MECO, IEEE, 2022, pp. 1–4.

[33] M.A. Mohsin, D.G. Perera, An FPGA-based hardware accelerator for K-nearest neighbor classification for machine learning on mobile devices, in: Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, 2018, pp. 1–7.

[34] M. Rahmatian, H. Kooti, I.G. Harris, E. Bozorgzadeh, Hardware-assisted detection of malicious software in embedded systems, IEEE Embed. Syst. Lett. 4 (4) (2012) 94–97.

[35] C. Li, D. Srinivasan, T. Reindl, Hardware-assisted malware detection for embedded systems in smart grid, in: 2015 IEEE Innovative Smart Grid Technologies-Asia (ISGT ASIA), IEEE, 2015, pp. 1–6.

[36] A. Gajjar, P. Kashyap, A. Aysu, P. Franzon, Y. Choi, C. Cheng, G. Pedretti, J. Ignowski, RD-FAXID: Ransomware detection with FPGA-accelerated XGBoost, ACM Trans. Reconfigurable Technol. Syst. 17 (4) (2024) 1–33.

[37] C. Pal, S. Saha, X. Zhai, G. Howells, K.D. McDonald-Maier, APPARENT: AI-powered platform anomaly detection in edge computing, IEEE Trans. Sustain. Comput. (2025).

[38] N. Miao, H. Sayadi, K.S. Kondapalli, M. Eslamimehr, H. Homayoun, LightBench: A hardware-aware trusted execution platform for intelligent malware detection at the edge, in: Proceedings of the Great Lakes Symposium on VLSI 2025, 2025, pp. 954–961.

[39] H.M. Makrani, Z. He, S. Rafatirad, H. Sayadi, Accelerated machine learning for on-device hardware-assisted cybersecurity in edge platforms, in: 2022 23rd International Symposium on Quality Electronic Design, ISQED, IEEE, 2022, pp. 77–83.

[40] K. Lehniger, M.J. Aftowicz, P. Langendorfer, Z. Dyka, Challenges of return-oriented-programming on the xtensa hardware architecture, in: 2020 23rd Euromicro Conference on Digital System Design, DSD, IEEE, 2020, pp. 154–158.

[41] B. Amatov, K. Lehniger, P. Langendörfer, Return-oriented programming gadget catalog for the Xtensa architecture, in: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops), IEEE, 2022, pp. 655–660.

[42] Z. Xu, J. Zhang, S. Ai, C. Liang, L. Liu, Y. Li, Offensive and defensive countermeasure technology of return-oriented programming, in: 2021 IEEE International Conferences on Internet of Things (IThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), IEEE, 2021, pp. 224–228.

[43] A. Omotosho, G.B. Welearegai, C. Hammer, Detecting return-oriented programming on firmware-only embedded devices using hardware performance counters, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, 2022, pp. 510–519.

[44] D. Pfaff, S. Hack, C. Hammer, Learning how to prevent return-oriented programming efficiently, in: International Symposium on Engineering Secure Software and Systems, Springer, 2015, pp. 68–85.

[45] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, I. Cervesato, On the detection of kernel-level rootkits using hardware performance counters, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 483–493.

[46] V. Herdt, D. Große, H.M. Le, R. Drechsler, Early concolic testing of embedded binaries with virtual prototypes: A RISC-V case study, in: Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.

[47] Y. Mahmoodi, S. Reiter, A. Viehl, O. Bringmann, Security analysis of embedded systems using virtual prototyping, in: Proceedings of the the 7th International Conference on Cyber-Technologies and Cyber-Systems, CYBER, IARIA, 2022, pp. 1–8.

[48] P. Pieper, V. Herdt, D. Große, R. Drechsler, Dynamic information flow tracking for embedded binaries using systemc-based virtual prototypes, in: 2020 57th ACM/IEEE Design Automation Conference, DAC, IEEE, 2020, pp. 1–6.

[49] A. Omotosho, S. Iiahi, E.C.V. Castillo, C. Hammer, C. Sauer, Evaluating the hardware performance counters of an xtensa virtual prototype, in: 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS, IEEE, 2023, pp. 1–6.

[50] D. Anguita, S. Pischiutta, S. Ridella, D. Sterpi, Feed-forward support vector machine without multipliers, IEEE Trans. Neural Netw. 17 (5) (2006) 1328–1331.

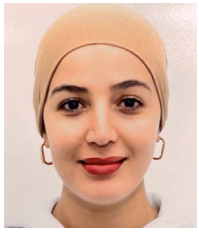
[51] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538), IEEE, 2001, pp. 3–14.

[52] J. Xu, D. Abraham, I.G. Harris, Run-time ROP attack detection on embedded devices using side channel power analysis, IEEE Embed. Syst. Lett. 16 (4) (2024) 377–380.

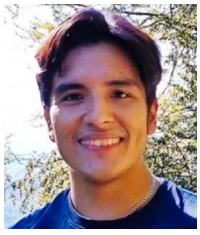
[53] Cadence Design Systems, Xtensa® Software Development Toolkit User's Guide, 2018, URL https://ip.cadence.com/swdev.

[54] J. Aynsley, OSCI TLM-2.0 Language Reference Manual, JA32, Open SystemC Initiative, 2009.

Dr. Adebayo Omotosho is a Senior Lecturer in Computing at the School of Business, Computing, and Social Sciences at the University of Gloucestershire. Previously, he was a lecturer in Cyber Security at the School of Engineering and Computing at the University of Central Lancashire. He earned his doctorate at Ladoke Akintola University of Technology and completed his postdoctoral research at the University of Potsdam and the University of Passau. His research focus is on embedded systems and Internet of Things security.



Sirine Ilahi is a Scientific Employee and Ph.D. candidate at the University of Passau, Germany, in the Chair of Software Engineering I. Previously, she held a similar role at the University of Potsdam, Germany. She completed her B.Sc. thesis at the Higher Institute of Biotechnology of Sfax, Tunisia, and her M.Sc. thesis at Karlsruher Institut für Technologie (KIT), Germany. Her research focuses on embedded systems security, with a particular emphasis on developing methods to enhance the security and resilience of embedded devices through software engineering practices. Her broader interests include vulnerability detection and leveraging machine learning techniques to strengthen the security of embedded systems.



Ernesto Cristopher Villegas Castillo received his B.Sc. in 2011 from the Pontifical Catholic University of Peru (PUCP) and his M.Sc. in 2014 from the University of Sao Paulo (USP). He is part of the system design enablement team at Cadence Design Systems Inc. and PhD. candidate by the Institute of Embedded Systems/Real-Time Systems at Ulm University. He is a former Digital Design Engineer at NXP Semiconductors, working with SoC RTL integration and verification, formal design tools, constraint definition, synthesis for netlist generation, and static timing analysis. Currently, he is working on research projects related to functional safety, artificial intelligence, and machine learning methodologies for Design Space Exploration, as well as virtual prototyping.

Professor Christian Hammer holds the Chair of Software Engineering 1 at the University of Passau. One of his main research interests is language-based software security, with a particular focus on the security of mobile and web applications, Internet of Things devices, and concurrent systems. After earning his doctorate at the Karlsruhe Institute of Technology (KIT), he conducted research at IBM Research and Purdue University. He then became a professor at Utah State University, Saarland University, and Potsdam University.

Dr. Hans-Martin Bluethgen leads the system design enablement team at Cadence Design Systems Inc., providing front-end services on SoC architecture development, large-scale virtual prototypes, Functional Safety, and design and verification. Previously, he was a research engineer and project manager at Infineon Technologies' mobile communications department, leading a multi-site software-defined radio project that involved virtual prototyping, embedded software development, and integrated circuit design. From 2001 to 2003, he was a visiting industrial fellow at the Berkeley Wireless Research Center (BWRC), Berkeley, USA. He holds a Dr.-Ing. degree from RWTH Aachen and a Dipl.-Wirt. Ing. degree from Hagen University.