Evaluating the Hardware Performance Counters of an Xtensa Virtual Prototype

Adebayo Omotosho
University of Central Lancashire
Preston, United Kingdom
aomotosho@uclan.ac.uk

Sirine Ilahi
University of Passau
Passau, Germany
sirine.ilahi@uni-passau.de

Ernesto Cristopher Villegas Castillo
Cadence Design Systems
Munich, Germany
ernesto@cadence.com

Christian Hammer
University of Passau
Passau, Germany
christian.hammer@uni-passau.de

Christian Sauer
Cadence Design Systems
Munich, Germany
sauerc@cadence.com

***Abstract*—Embedded systems' hardware and software stacks are becoming more complex requiring more development time, time to market, and cost, which contributes to delayed delivery of these silicon devices. A virtual prototype (VP) provides an embedded systems architecture simulator for application development and testing purposes. In this paper, we developed and present the first virtual prototype of the Xtensa LX7 microprocessor that evaluates the performance of its emulated hardware performance counters (HPCs) with those collected from an actual Xtensa LX7 hardware. Seven machine learning models were developed and trained to find the relationships between the two different datasets for the sample application of classifiying return-oriented programming (ROP) attacks. Our experiments show that the obtained micro-architectural characteristics on the VP are on average about 70% similar and thus permit early simulation capabilities for developers and testers.**

***Index Terms*—virtual prototype, xtensa, hardware performance counters, return oriented programming, machine learning**

I. Introduction

In recent years, embedded devices have become an integral part of the networked world and are used in a variety of applications such as household appliances, automobiles, and medical devices. Majorly, programs for these embedded systems are written in languages that leave memory management to the programmer, such as C/C++ [1]. This potential for memory vulnerabilities allows attackers to force a running program into executing illegal instructions, a process named control flow hijacking [2], [3].

The growing complexity and evasiveness of memory malware has led to the evolution of mitigation mechanisms, such as Control Flow Integrity (CFI) and Address Space Layout Randomization (ASLR), which were originally not targeted at embedded devices [4], [5]. CFI compares the software's execution to a predefined model computed based on the control flow graph. This involves a high-performance overhead which makes it unsuitable for firmware-only embedded devices. ASLR randomly arranges program segments in memory to prevent an attacker from predicting the addresses of program instructions. This approach is vulnerable to information leakage attacks that expose the memory layout and do not protect against memory attacks using ROP [6].

It is quite difficult to perform rapid testing of security features on embedded devices in the absence of a native environment [7], [8]. Increasing system complexity combined with shorter time-to-market has led to many challenges for system development and evaluation. A recent study by International Business Strategies shows that a three-month delay in time-to-market generally reduces chip manufacturers' revenue by about 30%, with the disadvantage being even greater in rapidly evolving markets such as mobile devices [9].

Virtual Prototypes (VPs) have been increasingly applied in hardware and software development, integration, and validation before silicon devices are ready [10]. Virtual prototyping techniques are widely explored and used by both industrial engineers and academic researchers, enabling early firmware development. VPs are software models developed according to the hardware specification. Such models simulate functional hardware behaviors and enable unmodified software execution on on *instruction set simulators* (ISSs). With virtual prototyping, software developers can develop and validate firmware without silicon hardware being available.

This paper evaluates the degree of the performance and capability of the micro-architectural characteristics of our developed VP for security testing, i.e., to detect code reuse attacks, in the absence of a native environment. The main contributions of this paper are:

- the development of a VP for simulating the Xtensa ISS and *hardware performance counters* (HPCs). This tool could be useful to embedded developers for virtually programming the Xtensa platform without the real hardware.
- the development of an *application programming interface* (API) for accessing Xtensa ISS elements such as cores,

979-8-3503-3277-3/23/$31.00 ©2023 IEEE

memories, and profiling summary counters.
- the first security evaluation of a VP and its native embedded platform using HPCs. Existing works on VP commonly focus on performance and power consumption validation, so a design space exploration could be performed [11]–[14].
- the development and evaluation of machine-learning algorithms to detect code reuse attacks via HPCs. Our evaluation demonstrates a sufficiently high similarity between the micro-architectural events of the VP and real hardware to perform early development and testing tasks, even for security purposes.

A. Motivation

Embedded systems are increasingly becoming complex in terms of functionality and architectural resources available to meet performance and cost requirements. It is therefore the responsibility of the developer to make the right choices in terms of software and hardware components. Verification and validation are key to reducing time-to-market and improving product quality. To reduce the trade-offs between timing accuracy and simulation speed, processor prototyping has become a better choice in the design flow. VP has found its place in early firmware development and validation. However, VP can be explored for characterizing other dependability concepts such as reliability, safety, and security [15], [16]. For instance, hardware-level mitigation for embedded devices security threats is growing [17], hence, such exploration can provide feedback and general insight into the behaviors of the real processor, which can support developers in speeding up the development of security modules and features.

II. Preliminaries

A. Hardware performance counters

Modern microprocessors have special registers that store information about various micro-architectural events such as clock cycles, cache access, cache overwrites, etc. These registers, called hardware performance counters, can be used to profile program behavior and detect illegal program modifications [18], [19]. HPCs can fit into a range of processor platforms, from high-performance processors to low-power embedded processors, but the number of micro-architectural events that can be computed simultaneously is restricted. This is primarily due to the limited number of physical registers on the processor chip, which are costly to implement. Also, the number of available HPCs and hardware events depends on the processor model. For example, the ARM V8 Cortex-A53 core can compute 62 events with four HPC registers, while Xtensa LX7 can count 125 events with eight HPC registers [18]. In this paper, we use HPCs to collect execution traces of selected micro-architectural events of both the malicious and benign firmware programs running from the flash memory. This is then used for evaluating our developed VP and real Xtensa.

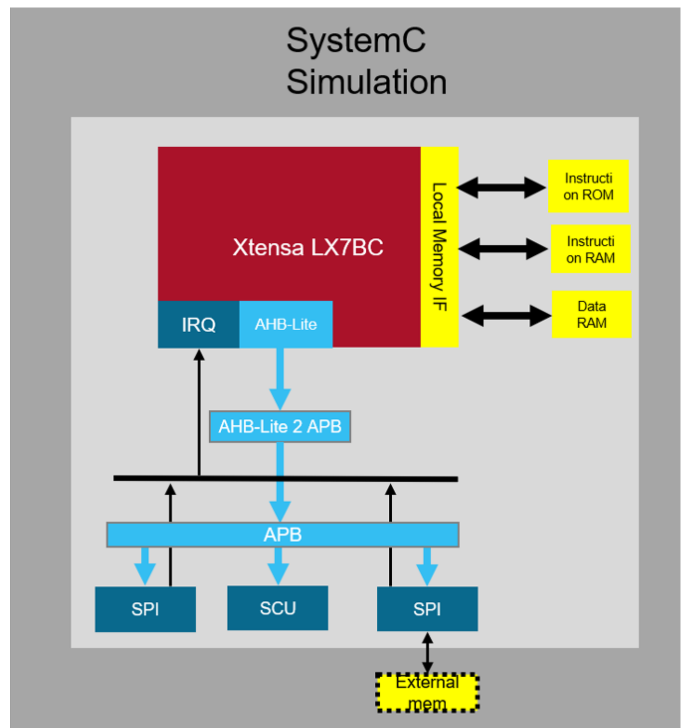


Fig. 1. VP configuration based on Xtensa LX7 ISS

B. Challenges of building our Xtensa VP

The Xtensa processor architecture is a 32-bit core, whose main feature is the ability to be configured and customized according to the customers' needs. Dedicated instructions can be added to the Xtensa architecture using its software development toolchain. Additionally, it provides support for HPCs, which can be configured and accessed inside the target code around the software that is required to be profiled. As stated in Section II-A, eight HPCs can be accessed and configured through *select* and *mask* registers that specify which events to count. However, a specific API that contains accessor functions to the HPCs is required.

Xtensa tools provide an *Xtensa SystemC* (XTSC) package that supports *transaction-level modeling* (TLM) of Xtensa cores for the development of *system on a chip* (SoC) VP. An Xtensa core model is a wrapper around the Xtensa ISS that can be used to test, debug, and profile software applications before being factored on a chip or synthesized in a field programmable gate array. The ISS also supports performance analysis based on the accumulation of statistics from the *instruction set architecture* (ISA) level to micro-architectural properties. At the end of a simulation run, a summary containing profile information such as committed instructions, instruction fetches, taken branches, loads, stores, cache misses, exceptions, etc. can be obtained. Unfortunately, these values might differ from the profile information obtained in real hardware, due to the lack of hardware details presented in the ISS. For instance, it is assumed that every instruction will take a single cycle to be executed.

```
addi a1, a1, -16 # prologue
s32i.n a0, a1, 0 # prologue
#gadget_1
#gadget_2
...
#gadget_n
l32i.n a0, a1, 0 # epilogue
addi a1, a1, 16 # epilogue
ret.n
```

Listing 1. Call0 ABI function structure

However, neither the API nor the HPCs are modeled inside the ISS. To overcome this problem, additional modeling was carried out using methods of the diverse component classes of the XTSC library. Additionally, a new API was developed to access internal information of the ISS elements (i.e. cores, memories, and profiling summary counters) using a *simcall* function that allows communication between the Xtensa target program and a Lua script running in a SystemC thread of the simulator. In this thread, XTSC library functions can be executed to access the component's real-time information in order to return their values. One of these functions accesses ISS profile summary counters such as Cycles, Instructions, TakenBranches, CacheReads, CacheRefills, CacheWrites (for Instruction and Data memories), and Data CacheCastouts. Furthermore, other elements' counters can be accessed, such as the number of blocking/non-blocking READ and WRITE commands performed over instantiated data memories of the ISS. The developed VP in this paper is shown in Fig. 1 and consists of the following components:

- an Xtensa LX7 SystemC/TLM2.0 model that wraps the ISS running the applications,
- an interconnection module based on TLM2.0, that serves as communication bus with other peripherals,
- an SPI model for booting the Xtensa LX7 from an external memory, and
- an SPI model in master mode for communication with other processors.

III. Evaluation

A. Implemented Xtensa Application Binary Interface

Our Xtensa hardware configuration uses the CALL0 *application binary interface* (ABI), in which function epilogues and prologues are of the form illustrated in Listing 1. The Xtensa ISA is a *load-store* architecture that, rather than featuring a `pop` opcode, leverages load instructions (`l32i`) to move values into registers. The *.n* suffix is Xtensa processors' code density feature that instructs Xtensa compiler/assembler to optionally use *16-bit narrow instructions*. Fig. 2 shows a simplified example of how code reuse attacks work on the Xtensa architecture. `l32i` loads the attacker's values and gadget addresses from the stack at offsets 4, 8, and 16 from the stack pointer register *a1*, into registers *a2*, *a3*, and *a4*, respectively. This task is performed by three gadgets *@g1*, *@g2*, and *@g3*, ending with `ret` instructions, which, as a consequence of the attacker's stack modifications, effectively call one another in succession. On successful execution, 200

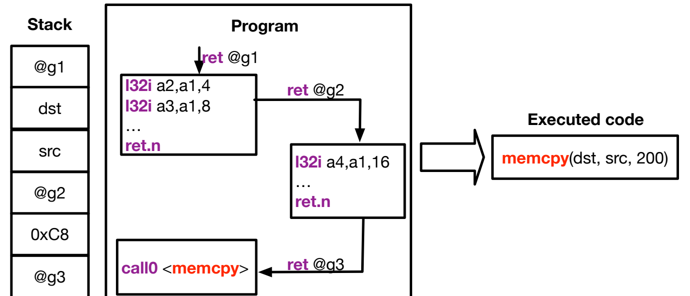


Fig. 2. Xtensa ROP

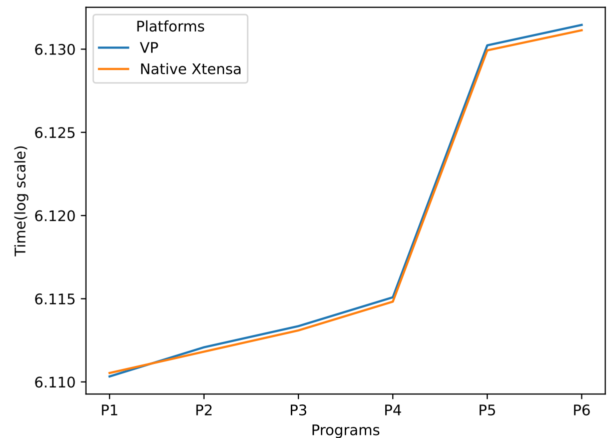


Fig. 3. Running time

bytes of input are copied from the *src* to the *dst* address. This process is generally possible after overwriting a vulnerable function's return address stored in register *a0*, which crashes the device, or hijacks control flow and forces the execution of a series of attack gadgets. One major difference between this and the *ARM* or *x86* architectures is that the return address appears immediately after a buffer variable, thus there is no need for another four bytes for the frame pointer. The payloads used in our experiment include not only gadget addresses but also values for parameters passed to firmware functions.

B. Evaluation programs

Vulnerable functions were inserted into our instrumented embedded programs through which code reuse attacks using ROP were initiated. Technically, our attacks exploit unsafe C functions like `memcpy` in Xtensa firmware (just as in Xtensa MediaTek audio DSP in 2021 [20]) to corrupt memory and launch ROP.

We modified and recompiled embedded program examples from the MiBench benchmark [21], [22]. MiBench contains looped implementations of algorithms (for the ARM architecture) with applications in areas such as networking, security, telecommunications, etc. They were developed in C using the *xt-cc* compiler. This compiler uses the GNU preprocessor, assembler, and linker while providing superior and smaller

TABLE I
IMPLEMENTED XTENSA HPCS

Label	HPC	Description
F1	COMMITTED_INSN	instructions committed
F2	BRANCH_PENALTY	Branch penalty cycles
F5	CYCLES	Count cycles
F12	D_STORE_U1	Data memory store instruction
F15	D_LOAD_U1	Data memory load instruction

TABLE II
MODELS PERFORMACE ON EVALUATION SET

Model	Precision		Recall	
	VP	Native Xtensa	VP	Native Xtensa
RFC	0.93	0.88	0.90	0.96
SGDC	0.90	0.74	0.68	0.58
SVC(RBF)	0.90	0.90	0.69	0.97
LSVC	0.93	0.72	0.77	0.51
LR	0.95	0.70	0.74	0.53
OSVM	0.24	0.50	0.41	0.39
SVM(HFK)	0.97	0.90	0.97	0.93

compiled code. The running time of the six variations of the breadth-first search *BFS* [22], in which ROP chains of length 1...6 (P1...P6) were exploited, is shown in Fig. 3. The running time tends to increase with the ROP chain's length, while the VP incurred a very low runtime overhead of 0.00046% for these six programs.

C. Model development

We could not emulate all eight HPCs used in [22] because, e.g., the pipeline HW structure is not modeled in the ISS, such that their respective counters could not be accessed. Furthermore, these counters can not be accessed from XTSC objects as it was for the *xtsc_memory* and and *xtsc_core* object case. Hence, only five HPC events that are available in XTSC serve as input into our machine-learning models. The emulated HPC events on the VP are shown in Table I.

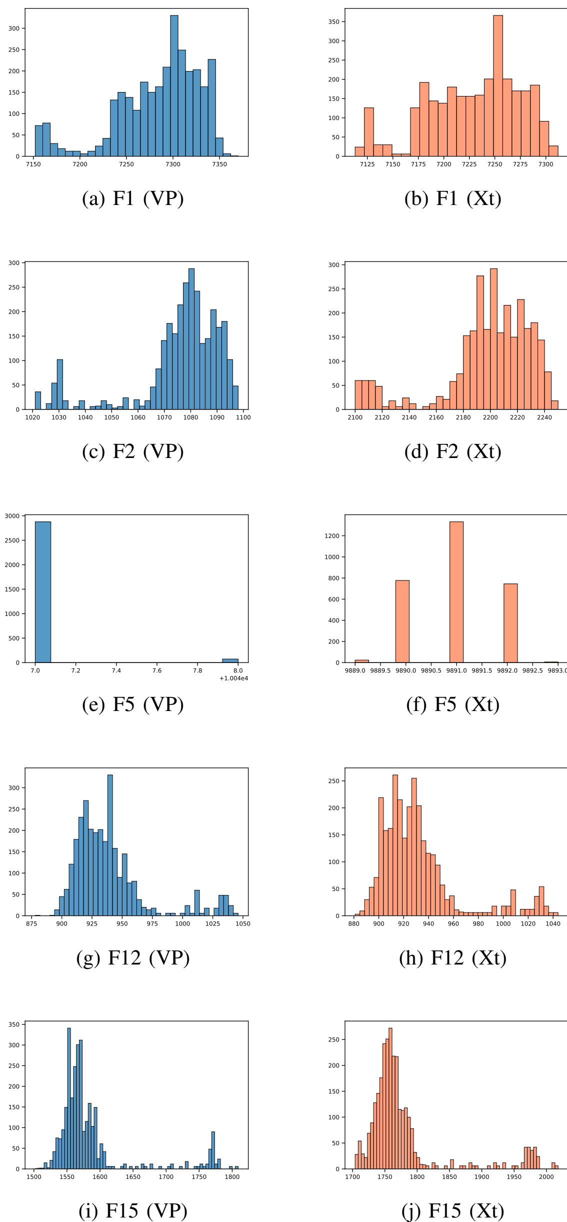
We trained seven ROP classifiers consisting of a Random Forest (RFC), SGD Classifier (SGDC), Support Vector with radial basis function kernel (SVC), Linear SVC (LSVC), Logistic Regression (LR), One Class Support Vector Machine (OSVM), and SVC (with a hardware-friendly kernel [23]) for each platform. The training set has the shape (5835, 7), 80% of the data was used for training and 20% for evaluation. The benchmark dataset that was used for the validation of our models has shape (2638, 7). The two additional features that were dropped were *#ROP* and *ROP label*. Ten-fold cross-validation was performed on all the models and the top three were RF, SVC (HFK), and SVC (RBF) with average accuracies of 0.96%, 0.97%, and 0.91% respectively. Although accuracy metrics alone can be misleading for classifiers it gives initial insight into the models' capabilities.

D. Research questions

The model evaluation will be based on two research questions:
[RQ1]: *Are HPCs on the VP comparable to real hardware?* This will be answered by verifying if there are similarities in the patterns of our five implemented HPC events on both the VP and native Xtensa.
[RQ2]: *To what extent can we predict ROP with VP HPC events?* The goal of the paper is not just to obtain high classification precision or recall but to have an insight into how well we can correctly predict ROP execution as an instance of a non-trivial testing task to be simulated by the VP. The classification metrics such as precision – a measure of the accuracy of positive predictions – and recall – a measure of the percentage of positive classes that are correctly classified – will be computed on both the evaluation set and benchmark programs whose HPCs were collected from real Xtensa.

E. Discussion

In this section, we discuss the results of our findings based on the research questions.
RQ1: *Are HPCs on the VP comparable to real hardware?*
Fig. 6 depicts the histograms showing the one-to-one comparison of five HPCs emulated on our VP versus the same counters on native Xtensa (Xt). These five HPC events are briefly described Table I. For the same set of embedded programs the simulated hardware events induce shapes similar to the real hardware's, potentially with a different value range. Notably, `F5` has fewer ranges in the VP. Nevertheless, these HPC events look reasonably similar and promising an approximation.
RQ2: *To what extent can we predict ROP with VP simulated hardware events?*
Seven machine learning models for classifying ROP exploitations were trained based on the five simulated HPC events in order to validate the feasibility of using the VP to obtain early security testing feedback through HPCs. Table II shows the performance of the model on the 20% evaluation set, and again, RF, SVC (RBF), and SVC (HFK) outperform the other four algorithms on both platforms. We further validated the model on ten embedded benchmark programs. The accuracy, precision, and recall of this additional evaluation are shown in Fig. 5, 6 and 7. For programs where we have ROP classification scores for native Xtensa, the accuracy, precision, and recall are about 80%, 90%, and 70%, respectively, sufficiently close to the VP. The evaluation further shows a reason why accuracy is not a reliable metric for classifiers. For example, the program *Prim* was not correctly detected. We briefly investigated this, the benchmarks each have different ROP length exploitations. Interestingly, ROP chains of *length = 2* were exploited in *Prim*, the same as in *DFS*. As it has been shown that the complexity of a program and not just the ROP chain length affects ROP classification, e.g., DFS is $O(V+E)$ and Prim, $O(E \log V)$. This program was correctly detected by a precision and recall score of over 80% with the eight HPC model in previous work [22]. Given the limitations of HPCs in the VP, it cannot be the aim of this paper to match the previous results exactly. Rather we intended to find out if the VP is an accurate simulation of native Xtensa with sufficient predictive power. Nevertheless, our future work will further investigate improvements on the overall performance of the VP.


(a) F1 (VP) (b) F1 (Xt)

(c) F2 (VP) (d) F2 (Xt)

(e) F5 (VP) (f) F5 (Xt)

(g) F12 (VP) (h) F12 (Xt)

(i) F15 (VP) (j) F15 (Xt)

Fig. 4. Comparison of the shape of the HPCs on VP vs. Xt hardware

F. Threats to Validity

The following outlines potential threats to the validity of our experimental results:

- The benchmark and training programs used may not reflect representative behaviors of embedded programs running from the flash memory. For example, we do not consider internet connectivity and remote exploitations. However, the programs in our evaluation have similarly been used for validating embedded processors [21], [22].
- Only a few HPCs are available on our VP, therefore, the simulation capabilities are limited. It may be possible in the future to emulate further hardware events. However, our VP achieved its goal of giving early insights and being reasonably comparable to native Xtensa.

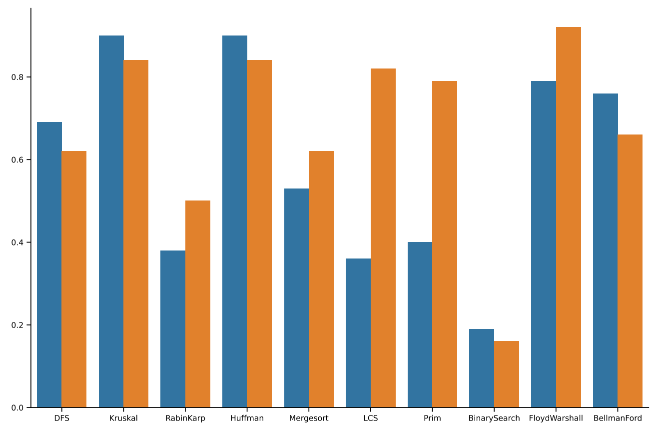


Fig. 5. Accuracy (VP in blue, Native Xtensa in orange)

IV. Related Work

The challenges of emulating embedded systems on chips are discussed in [12]. It was stated that although VP can provide early testing, market-entry, and other development insights, simulated platforms hardly outperform the real hardware or software. Our work does not focus on VP response time but tests the ability to produce comparable micro-architectural behavior in the VP and actual Xtensa hardware.

A VP of RISC-V instruction set extensions was presented in [24]. The proposed VP is a lightweight alternative approach to identifying application specific extensions. It aims at discovering an appropriate sequence of instructions that could be replaced by a custom RISC-V instruction. Our paper is not focused on instruction extension but rather on emulating Xtensa ISS and HPCs.

Similarly, [13] also worked on a RISC-V-based VP, which was also only evaluated based on timing and system-level features such as power, complex software, and hardware interactions. Our paper differs in that we model and focus on using low-level characteristics of the Xtensa VP to detect code reuse attacks.

Virtual prototyping of embedded software and its physical environment was done in [11]. Their simulation framework was developed using SystemC open-source tools. Unlike our paper, which focuses on HPC behavior, [11] primarily focuses on simulation speed.

V. Conclusion

A VP of an embedded processor can be extremely useful in testing and development when the native platform is not available due to reasons such as high cost, delayed delivery, etc. In this paper, we have developed and evaluated the Xtensa instruction set simulator VP to investigate if the HPC events reflect the behavior of a native environment, and whether they sufficiently accurately simulate the realistic values exemplified via detecting code reuse attacks. Micro-architectural events were collected from both the VP and native Xtensa and seven

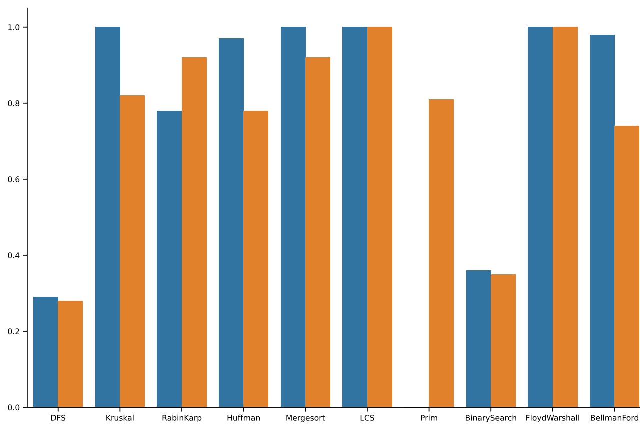


Fig. 6. Precision

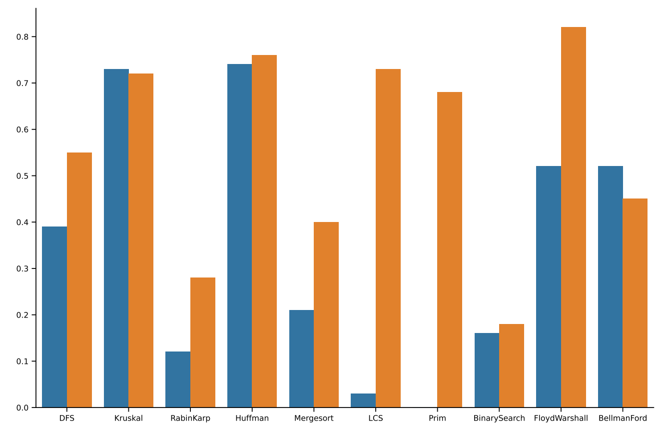


Fig. 7. Recall

VP in blue, Native Xtensa in orange

machine-learning models were trained. Even though we did not expect the two test platforms to produce identical results, unsurprisingly the native environment outperforms the VP. Nevertheless, we thoroughly evaluated the models and our finding shows that about 80% of the accuracy, 90% of the precision, and 70% of the recall of the samples detected in the native environment are comparably close to the VP. Although there is room for further improvement, we believe that our developed toolset could help embedded developers in early testing and development.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) under research grant number 01IS18065D.

References

[1] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in c/c++ and java source code," *Procedia Computer Science*, vol. 171, pp. 2023–2029, 2020.

[2] X. Cheng, H. Wang, J. Hua, G. Xu, and Y. Sui, "Deepwukong: Statically detecting software vulnerabilities using deep graph neural network," *ACM TOSEM*, vol. 30, no. 3, pp. 1–33, 2021.

[3] J. Xu, D. Mu, P. Chen, X. Xing, P. Wang, and P. Liu, "Credal: Towards locating a memory corruption vulnerability with your core dump," in *Proceedings of the 2016 ACM SIGSAC CCS*, 2016, pp. 529–540.

[4] F. Yan and K. Wang, "Leakage is prohibited: Memory protection extensions protected address space randomization," *Tsinghua Science and Technology*, vol. 24, no. 5, pp. 546–556, 2019.

[5] R. Denis-Courmont, H. Liljestrand, C. Chinea, and J.-E. Ekberg, "Camouflage: Hardware-assisted cfi for the arm linux kernel," in *2020 57th ACM/IEEE DAC*. IEEE, 2020, pp. 1–6.

[6] X. Wang and J. Backer, "Sigdrop: Signature-based rop detection using hardware performance counters," *arXiv preprint arXiv:1609.02667*, 2016.

[7] N. Corteggiani, G. Camurati, and A. Francillon, "Inception:{System-Wide} security testing of {Real-World} embedded systems software," in *27th USENIX*, 2018, pp. 309–326.

[8] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM TECS*, vol. 3, no. 3, pp. 461–491, 2004.

[9] S. Mu, G. Pan, Z. Tian, and J. Feng, "Asurvey of virtual prototyping techniques for system development and validation," *International Journal of Computer Science & Engineering Survey (IJCSES)*, pp. 19–26, 2015.

[10] S. Neumeyer, K. Exner, S. Kind, H. Hayka, and R. Stark, "Virtual prototyping and validation of cpps within a new software framework," *Computation*, vol. 5, no. 1, p. 10, 2017.

[11] W. Mueller, M. Becker, A. Elfeky, and A. DiPasquale, "Virtual prototyping of cyber-physical systems," in *17th Asia and South Pacific Design Automation Conference*. IEEE, 2012, pp. 219–226.

[12] Y. Ni, W. S. Mong, and J. Zhu, "On virtual prototyping of embedded system-on-chips," in *2011 9th IEEE ASIC*. IEEE, 2011, pp. 1106–1109.

[13] V. Herdt, D. Große, P. Pieper, and R. Drechsler, "Risc-v based virtual prototype: An extensible and configurable platform for the system-level," *Journal of Systems Architecture*, vol. 109, p. 101756, 2020.

[14] S. T. Sliper, W. Wang, N. Nikoleris, A. S. Weddell, and G. V. Merrett, "Fused: Closed-loop performance and energy simulation of embedded systems," in *2020 IEEE ISPASS*. IEEE, 2020, pp. 263–272.

[15] A. Miele, "A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems," *Microprocessors and Microsystems*, vol. 38, no. 6, pp. 567–580, 2014.

[16] B.-A. Tabacaru, M. Chaari, W. Ecker, T. Kruse, and C. Novello, "Speeding up safety verification by fault abstraction and simulation to transaction level," in *2016 IFIP/IEEE VLSI-SoC*. IEEE, 2016, pp. 1–6.

[17] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *2017 54th ACM/EDAC/IEEE DAC*. IEEE, 2017, pp. 1–6.

[18] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *2015 IEEE/ACM ICCAD*. IEEE, 2015, pp. 544–551.

[19] C. Malone, M. Zahran, and 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.

[20] S. Makkaveev, "Looking for vulnerabilities in mediatek audio dsp," Available at https://research.checkpoint.com/2021/looking-for-vulnerabilities-in-mediatek-audio-dsp/ (2023/01/02).

[21] MiBench, "Mibench version 1.0- embedded benchmark suite," Available at https://vhosts.eecs.umich.edu/mibench/ (2023/01/02).

[22] A. Omotosho, G. B. Welearegai, and 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.

[23] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-forward support vector machine without multipliers," *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1328–1331, 2006.

[24] M. Funck, V. Herdt, and R. Drechsler, "Virtual prototype driven design, implementation and evaluation of risc-v instruction set extensions," in *2022 25th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2022, pp. 14–19.