Received November 17, 2021, accepted December 6, 2021, date of publication December 10, 2021,
date of current version December 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3134256

A Real-Time Cache Side-Channel Attack Detection System on RISC-V Out-of-Order Processor

ANH-TIEN LE[1,2], (Graduate Student Member, IEEE), TRONG-THUC HOANG[1,3], (Graduate Student Member, IEEE), BA-ANH DAO[1,2], (Graduate Student Member, IEEE), AKIRA TSUKAMOTO[3], KUNIYASU SUZAKI[3,4], (Member, IEEE), AND CONG-KHA PHAM[1], (Member, IEEE)

[1]Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Chofu, Tokyo 182-8585, Japan
[2]Academy of Cryptography Techniques (ACT), Hanoi 12511, Vietnam
[3]National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan
[4]Technology Research Association of Secure IoT Edge Application Based on RISC-V Open Architecture (TRASIO), Tokyo 101-0022, Japan

Corresponding author: Anh-Tien Le (leanhtien@vlsilab.ee.uec.ac.jp)

This work was supported by the New Energy and Industrial Technology Development Organization (NEDO) under Project JPNP16007.

ABSTRACT Computer designers have included techniques such as speculative execution and caching to optimize speed and performance. Unfortunately, they could be exploited by the recently discovered cache-side channel attack, spectre. The purpose of this research is to resolve this problem on the open-source RISC-V architecture. Previously, software mitigation techniques and hardware modifications have been investigated on Intel or ARM system to address these issues. However, they are either difficult to implement or have resulted in a significant loss of performance. This work presents a real-time detection approach for cache side-channel attacks such as spectre on the RISC-V processor. By monitoring the CPU's cache activity with Hardware Performance Counters and analyzing the gathered data with a neural network, we extend previous researches in the field of cache side-channel identification. Since cache side-channels frequently result in a significantly altered cache usage pattern, the proposed multi-layer perceptron network can detect an attack event with an accuracy greater than 99% in our test environment with low performance overhead. This is the first time, to our knowledge, that a spectre attack on the RISC-V architecture has been detected in run-time via hardware events and machine learning.

INDEX TERMS RISC-V, spectre, side-channel attack, machine learning, cache memory, hardware performance counters, real-time, software, security.

I. INTRODUCTION

Micro-architectural analysis (MA) [1] has been introducing a new area of side-channels attacks. Unlike attacks based on analyzing theoretical cryptography algorithms, it successfully exploits the affect of processor configuration on the security of a cryptographic system. Micro-architectural attacks could take advantage of the executed code inside the device even if the attacker does not have complete possession of the physical machine and additional measurement equipment. With modern computers implementing various components to increase the execution performance, those units become convenient targets for MA. Spectre attack is a practical example in the era of micro-architectural analysis.

The associate editor coordinating the review of this manuscript and approving it for publication was Yongming Li.

Since 2018, the two most interesting vulnerabilities in the processor security field have been spectre [2] and meltdown [3]. They pose a threat to nearly every modern processors. The early spectre variants were inspired by cache timing and branch prediction. It broadens the scope of micro-architectural analysis research by profiting on a significant defect in the technology that implements speculative execution. Spectre intentionally identifies a target function that does not strictly adhere to a serialized in-order process but executes speculatively. It exploits mispredictions of speculative executions of those target functions to obtain sensitive information about victims. Both Intel and ARM processors are vulnerable to spectre-style attacks [4], [5]. This cache side-channel attack also targeted the open-source RISC-V architecture [6], [7].

Because of the threats from spectre, many researchers focus on finding the methodology to prevent this kind of side-channel attack [8]. Mainly, the favoured approach is modifying or strengthening the processor architecture of the out-of-order processor. SafeSpec [9] holds speculative refilled data for caches and translation look-aside buffers(TLBs) in shadow or user-invisible structures. Therefore, it could defence most attacks leaking through the i-cache or the TLBs. Reference [10] proposed a low-frequency re-configurable hardware for detecting cache side-channel attacks. Even the developer of RISC-V Berkeley out-of-order processor has improved the line-fill buffers to prevent the attacker processes from learning speculated behaviours from the L1 data-cache state in their newest version [11]. Some methods provide a complete modification from software, operating system (OS) to hardware to defence against spectre attack. SpectreGuard [12] focuses on bound check bypass attack type. It marks sensitive memory blocks by OS/Library API. They also prevent secret dependent speculative execution by delaying a speculative memory access instruction until the execution is safe. SpecCFI [13] prevents speculative control-flow from being hijacked. Control-flow integrity (CFI) information becomes commonly available for modern processors by hardware CFI extensions. They use the CFI data to apply CFI principles to limit illegal control-flow during speculative execution. The majority of the studies as mentioned above provide novel defence mechanisms with minimal performance loss. However, they necessitate complex customization of the previous processor's architecture, which is quite challenging to implement. Theoretically, some simple software-based methods have also been studied. By preventing speculative execution effects, fence and speculative load hardening (SLH) aim to strengthen vulnerable code that a spectre attack could exploit. These approaches, however, necessitate recompiling the attacked program and result in significant performance loss.

Given the limitations of software mitigation techniques, it is critical to implement real-time detection techniques for spectre attacks until these attacks are completely eliminated. This can be accomplished by customizing the hardware of forthcoming processors and developing a software mitigation method that does not degrade performance. On the basis of hardware performance counters and machine learning, we provide real-time detection methodologies for spectre. Our proposed detection technique is capable of detecting attacker activities with high accuracy in environments with particular RISC-V processors and not decreasing the execution performance.

To identify harmful attacks statically, typical security software scans suspect commands in binaries or traces in system log files. However, spectre provides no evidence in system event logs. As a result, static analysis is ineffective at detecting spectre. A recent study has demonstrated that malware can be discovered utilizing dynamic micro-architectural execution patterns gathered from current processors' existing hardware performance counter (HPC). This research [14] describes a machine learning-based monitoring technique for detecting various micro-architecture side-channel attacks. The proposed method involves monitoring low-level hardware events via the *Perf* API in Linux operating system and then by applying relevant machine learning models to obtain data. CloudRadar [15] analyzes counter value to deploy the cache side-channel detection on virtual machine OpenStack cloud system. Miao Yu *et al.* presents an algorithm based on hardware performance counter data for detecting code-reuse attack such as Return Oriented Programming (ROP) [16]. NumChecker [17] detects malicious alterations to a system call in the guest virtual machine by analyzing the number of particular hardware events that occur during the execution of the system call. There are also researches successfully using HPCs to detect spectre on common processors, especially Intel. Chiappetta *et al.* [18] develops a utility called *quickhpc* which based on the information gathered by *Perf* and *PAPI* tools. Mushtaq *et al.* [19] develops NightsWatch for detecting Prime+Probe attack using 12 different machine learning models. Bazm *et al.* [20] uses Intel Cache Monitoring Technology (CMT) to obtain performance hardware counter for the same purpose. They create a machine learning model using Gaussian anomaly detection method. Cho *et al.* [21] benefits from Intel Performance Counter Monitor (Intel PCM) and deep learning to detect Flush+Reload, Prime+Probe and even Flush+Flush attack scenarios. Reference [22] takes advantage of fully supported *Perf* engine to obtain performance counter data on Intel processors. They successfully detected spectre-v1 with high accuracy above 90%. Reference [23] designed an edge-based classifier and implemented it to validate the same approach on x86 and ARM-based SoC prototype architectures.

Nevertheless, none of these has been confirmed on the RISC-V processor because of the source nature of this architecture. Common HPC utilities such as *Perf* or *PAPI* are not fully supported on RISC-V architecture. Therefore, it is currently challenging for recording hardware performance value on RISC-V processor. We address the issue of detecting cache-side channel attacks on RISC-V system precisely and effectively in this study. The comparison of this work to prior researches is presented in Table 1. We are not relying on pre-existing tools, such as on earlier studies, but are instead developing our own hardware performance counter analysis tools. We present a technique for detecting attacks that leverage hardware speculation, branch prediction, and out-of-order execution. This validation would try to mitigate both spectre-v1 and spectre-v2 which used common cache timing attacks technique such as Flush+Reload and Prime+Probe. Additionally, we implemented our own cross-process spectre scenario and a variety of alternative spectre implementation techniques to expand the test. Our detection system would notify users of any of these threats. To identify spectre, we monitor the translation look-aside buffer and branch prediction related events pattern for all running processes using hardware performance counters. After that, we use machine learning models to identify malicious events.

TABLE 1. Comparison to related works.

Research	Year	Target	Tool	Performance counter	Detection model
[18]	2016	Intel	Perf & PAPI	TOTAL_INSTRUCTIONS TOTAL_CYCLE LLC_ACCESS LLC_MISS	Correlation-based Neural Network
[19]	2018	Intel	PAPI	TOTAL_INSTRUCTIONS TOTAL_CYCLE CACHE_ACCESS CACHE_MISS	Linear Discriminant Analysis Logistic Regression Support Vector Machine
[20]	2018	Intel	Intel CMT	LLC_MISS LLC_REFERENCES iTLB_MISS iTLB_ACCESS	Gaussian
[21]	2019	Intel	Intel PCM	CACHE_MISS RETIRED_BRANCH	Single-layer perceptron
[22]	2021	Intel	Perf	LLC_MISS LLC_REFERENCES BRANCHES BRANCH_MISPREDICTION	Logistric Regression Support Vector Machine Neural Network
[23]	2021	ARM & x86	ARM PMU & Perf	BRANCH_INSTRUCTIONS L1_CACHE SPECULATIVE_LOAD/STORE	Support Vector Machine Random forest
This work	2021	RISC-V	Proposed tool	INSTRUCTION_RETIRED TOTAL_CYCLE BRANCH_MISPREDICTION iTLB_MISS	Multi-layer perceptron Logistic Regression Support Vector Machine

The contributions of our work are three-fold:

1) We propose a detection system for spectre attacks using hardware performance counters.
2) We find effective hardware performance counters for the detection of spectre attacks and implement our detection technique under realistic RISC-V system.
3) We implement practical RISC-V applications data-set to gather hardware counters data. We provide experimental results and discussion on detection accuracy with different machine learning models.

II. BACKGROUND KNOWLEDGE

A. BERKELEY OUT-OF-ORDER PROCESSOR

RISC-V has been developed in recent years as a new instruction set architecture that is freely available under open, non-restrictive licenses. It is intended to promote computer architecture research and education and garner assistance from significant enterprises, such as chip and device manufacturers. The RISC-V ecosystem offers a new level of innovation in processor architecture, which is critical for achieving the required performance and power efficiency improvements over the next decade. One of the RISC-V's benefits is its simplicity; it is far smaller than other commercial general-purpose instruction set architecture (ISAs) but retains considerable detail. Additionally, it supports applications, operating system kernels, and hardware implementations that use both 32-bit and 64-bit address spaces. As a result, both in-order and out-of-order processors based on the RISC-V instruction set have been explored and constructed.

The out-of-order processor implements a dynamic rescheduling mechanism that makes optimal use of the chip's resources in order to boost the computer's performance. The University of California, Berkeley, has been developing and maintaining "Berkeley Out-of-Order Machine (BOOM)", an open-source synthesizable parameterized out-of-order RISC-V processor [24].

The micro-architecture of the BOOM contains several fundamentals that create an ideal environment for speculative execution attacks. As an out-of-order processor, BOOM uses the optimization method to achieve maximum utilization of execution units available in a CPU. A processor having out-of-order execution functionality does not wait for the instructions to complete their execution in sequential order. Preceding instructions start executing if all necessary operands and functional units are available without waiting for the previous instructions to complete their execution. Regardless, visible effects persist from those erroneous steps that side-channel attacks could exploit to obtain private information. Correspondingly, the BOOM processor includes some convenient characteristics for the spectre attack.

1) SPECULATIVE EXECUTION

Unlike out-of-order execution, speculation attempts to finish subsequent computation; nonetheless, speculation is meant to benefit programs in executing faster when the control flow or data dependencies are uncertain. Probably the most significant sort of speculative execution is branch prediction. Predicting whether a conditional or indirect branch will occur is challenging because it could require more time. If the prediction is true, retired instructions are re-initialized; if the prediction is incorrect, a (theoretically) re-misprediction, speculatively executed instruction is re-initialized. The branch predictor unit (BCU) is a micro-architecture component that permits speculative forecasting of unconditional and conditional jumps, as well as function return and return pathways.

2) CACHING

The term ''implicit caching'' refers to the caching of memory resources, such as data or instructions that are not immediately retrieved or accessed. Implicit caching may occur in current processors as a result of ''aggressive prefetching, branch prediction, and TLB miss handling''. For example, improperly predicted branching will result in the fetching and execution of instructions, as well as data memory reads and writes, that the program does not intend. Spectre attack targets on cache effect of the out-of-order processor to calculate the secret key from the target procedure.

B. CACHE TIMING ATTACKS

By examining the time difference between cache hits and cache misses, cache side-channels infer the victim's memory access habits. This attack approach is called cache-timing attacked. When data is searched up in the cache but is not discovered because the cache does not include the item being sought, this is referred to as a cache miss. When a processor searches for an item in a cache, the item is saved, and the processor can satisfy the query, resulting in a performance improvement. This is known as a cache hit. Prime+Probe and Flush+Reload are both standard approaches for observing sensitive program execution trances left in the CPU caches. They would monitor cache miss and hit events and analyze them to interpret the target software's sensitive information. These techniques are significant components for performing spectre attack.

1) FLUSH+RELOAD ATTACK

The Flush+Reload attack approach [25] is based on observing how the processor's memory behaves while programs are executed. When the CPU requires already cached data (cache hit), the performance is significantly faster than when the CPU must reach main memory to retrieve the data. To execute the Flush+Reload approach, the adversary must share particular physical memory pages with the victim. Often, the Flush+Reload assault is presented in three stages. To begin, the adversary clears the shared memory address from the memory cache. It is the L1 cache in the BOOM core in our suggested environment. The spy then waits for the victim function to execute to gain access to the sensitive data. Finally, it reloads to determine which piece loads faster than the rest. As a result, it is the final element accessed by the victim process, which contains the value that the attacker wishes to retrieve. Because most security systems rely on conditional and branch instructions to authenticate authorization to access sensitive data, this type of attack could endanger a wide number of programs that store sensitive user data. This technique has the advantages of being less noisy and maintaining a high accuracy.

2) PRIME+PROBE ATTACK

The prerequisite condition for performing a Flush+Reload attack is that the malware must share particular physical memory pages with the target. In comparison, Prime+Probe [26] is another popular cache-timing attack strategy that does not require the victim to share memory pages. At its most basic level, the attacker begins by completely priming the cache sets which they want to monitor with a well-chosen eviction set. When the victim generates memory references, its accesses act as a substitute for some of the attacker's cache lines. The attacker can then re-visit the eviction set; if this access results in a cache miss, the adversary can determine that the victim has accessed that cache set, resulting in the replacement of his data. The most widely used way for carrying out a prime and probe attack is systematically accessing all cache lines in the same cache set from the eviction set while timing the overall access time. This approach, priming and probing, are combined into a single access pattern, allowing for efficient monitoring of a single cache set. Despite the fact that the attacker does not need to share a memory with the victim, this strategy produces more noisy and inaccurate results than the Flush+Reload approach.

C. HARDWARE PERFORMANCE COUNTERS

Modern processors are integrated hardware counters that measure specific events occurring at the machine level, such as the number of cycles and instructions executed by an application, its corresponding cache memory behaviour, and off-chip memory procedure. While the information collected by performance counters are frequently simple, when combined properly, it could present a useful report about a system's activity and therefore serve as a valuable debugging mechanism. Typically, performance counters on the hardware are used to verify and improve the performance of the software. The first step in optimizing a program is to ascertain its performance on the target system. Similar counting capabilities are provided by different computer processor manufacturers, such as Performance Counter Monitor (PCM) in Intel processor and Performance Monitoring Unit (PMU) in ARM processor.

HPC metrics may be obtained via a simple command-line interface using profiling programs such as *Perf*. *Perf* is integrated within Linux operating system to be a performance analyzing tool. It supports both Intel and ARM processor. *Perf* can be read in one of two ways: through counting or through sampling. At the conclusion of a counting mode program, the occurrences of events are simply collected and reported on standard output. It performs sampling on numerous metrics based on the occurrence of a certain number of events. Unfortunately, RISC-V is currently not fully supported with a thorough performance counter monitor tool such as *Perf*. As a result, assessing HPC on a RISC-V machine would be challenging.

D. NEURAL NETWORKS

Artificial Intelligence (AI) has made tremendous strides in recent decades, progressing from object recognition and detection algorithms to software and hardware with amazing execution capabilities. Artificial neural networks (ANNs) are machine learning models inspired by the human brain's structure. Because neural networks excel at recognizing patterns in large amounts of data, they have been shown to be extremely effective at performing classification challenges. Classification's objective is to specify the category to which a given input belongs. When detecting spectre attacks, the input is collected HPC data from several processes, and the output is classified as benign or malicious.

Numerous artificial neurons are arranged in multiple layers to form neural networks. Each neuron has a value and is connected to neurons in the subsequent layer via weighted connections. A feed-forward network is the simplest network architecture. The feed-forward network depicted has an input and output layer as well as hidden layers. Each node in the output layer is represent the probabilities of the input belong to the corresponding category. To forecast a value, the network takes inputs from the input layer, feeds them to the hidden layer, and then outputs the prediction at the output layer. The projected category is the output node with the highest value. The layers in this example are fully connected, i.e, each neuron in one layer is connected to every neuron in the preceding layer. To forecast a value given an input, each neuron adds the weighted values obtained from all connected neurons in the previous layer and then sends the result through an activation function.

The activation function of a node in artificial neural networks determines the output of that node given input or group of inputs. Normally, the activation function returns a value in the range of 0 to 1. There are some activation functions frequently considered for usage in hidden layers; they are as follows: Rectified Linear Activation (ReLU), Logistic (Sigmoid) and Hyperbolic Tangent (Tanh). The activation function's objective is to introduce non-linearity into a neuron's output. Each neural connection's weight must be adjusted to determine the optimal mapping between input and output values.

III. PRELIMINARIES

A. TARGETED DEVICE

In this work, we used an out-of-order RISC-V processor as in Table 2. The system consisted of a Berkeley out-of-order processor, BOOM. The processor implemented the RV64GC extension instruction set, which contains Integer, Multiplication, Atomic, Floating Point, Double, and Compress instructions. The implementation procedure was generated using the rocket-chip generator [27], [28]. The BOOM core is a 64-bit two-wide configuration, which means that we can write up to two instructions into the "issue queue" in the processor pipeline.

TABLE 2. Processor configuration.

	BOOM
iTLB	1 set, 32-entry
dTLB	1 set, 8-entry
I-Cache	block size: 64B, size: 16384B
D-Cache	block size: 64B, size: 16384B

We prepared and tested the experimental environment by practical degrees, from software simulation using Verilator to implementing it on a physical FPGA, which is the closest thing to a real RISC-V chip. The used FPGA is the Xilinx Virtex-7 VC707 FPGA board. The VC707 board provides a hardware environment for evaluating processor design. It comes with 1GB of DDR3 SODIMM memory, a PCI Express® interface, and two UART interfaces. On FPGA, the processor is booted into a light Linux operating system. This processor is then used to experiment with the spectre attack replication and to develop the real-time detection system.

B. CACHE SIDE-CHANNEL ATTACK

As BOOM contains some appropriate fundamentals for speculative execution attacks. We have re-implemented the first two variants of the spectre attack: bound check bypass attack and branch target injection attack. After that, we also developed a new cross-process scenario of spectre on the RISC-V platform. These cache side-channel attacks would then experiment on the proposed processor. The attack's goal is to obtain sensitive information stored in a memory region that only the victim program can access. We assume that the adversary could execute any programs or processes that he developed on the host machine. The purpose is to replicate the spectre's problem in RISC-V system before verifying the proposed detection method. Additionally, when replicating the spectre attacks on RISC-V environment, we have also tried different parameters (e.g. training round…) to find out which minimum conditions for the attacker to exploit the system successfully. Those conditions are also applied when we verify our detection system.

In the spectre-v1, Bound Check Bypass, the conditional branch becomes the target, often called a spectre gadget. Typically, the condition branch would be executed in sequential order. The inside code block normally only be executed after the condition, for example, $x < size$ is examined. However, for the processor containing speculation execution, like the proposed processor, the system would try to predict the result of the conditional check, which could be true most of the time. Then the private information of the

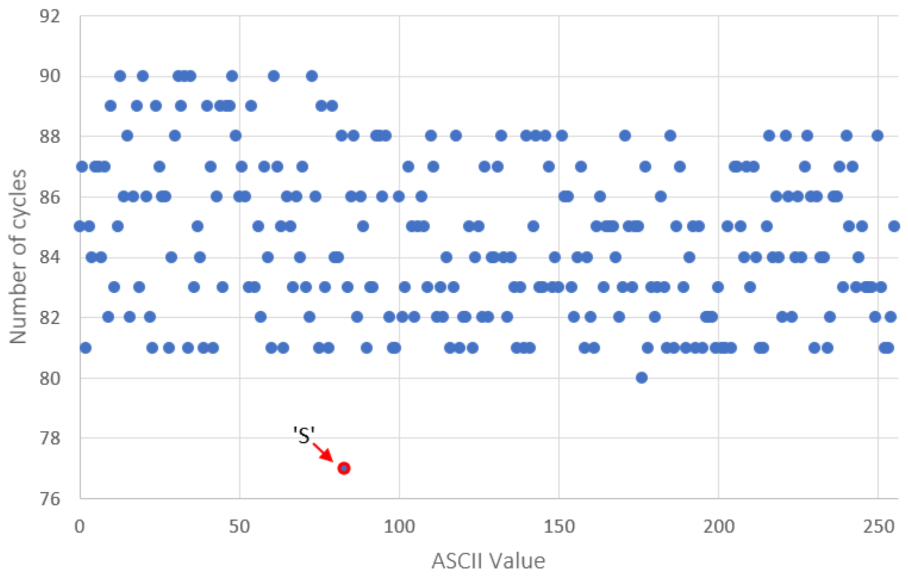


FIGURE 1. Cache access time of 256 ASCII values.

```
function victim(){
    if(x < a_size)
        data = a[x];
}
```

LISTING 1. Spectre gadget.

inside code block could be loaded into the memory before the examination of the condition in the bound check.

In this attack, the adversary points to the address space of the victim function and reads the victim data in the conditional branches. The condition branch in the victim function would be run and trained repeatedly. Then, during the attack round, the attacker gives a value to the victim that fails the bounds check in the victim. The cache effect would appear, and the attacker would use the Flush+Reload [25] method to achieve the secret value. In each attack round, the target is to achieve one secret byte. The secret value could be 256 different results that correspond to the ASCII table's 256 values. The value will be collected as a guess for the secret byte if its access time is less than the number of cycles required to reach the cache hit threshold and other 255 values. For example, in Figure 1, the value 83 corresponding to character 'S' has the lowest cache access time after performing the attack, so 'S' is the secret byte.

In variant 2 of the spectre attack, Branch Target Injection, the attacker targets the branch target buffet to guide the transient execution to a misprediction branch target. Direct branches happen when the destination of the branch can be known from the address declared in the instruction. Indirect branches, on the other hand, happen when the destination of the branch is not contained in the instruction itself. During run-time, the indirect branch predictor would look up the Branch History Buffer for the branching history to predict the destination of the function and immediately start to speculative execute the function. The adversary could find a program similar to the target program to manipulate indirect branch predictor. Returning to the attack scenario, after (mis)training the indirect branch predictor, this attack exploits the *jalr* instruction which uses speculative execution to predict the destination. Therefore, during the attack round, the *jalr* jumps to the private function instead of the other function. Therefore, the sensitive data is loaded and could be achieved by Flush+Reload similar to variant 1.

Most previous spectre proof-of-concept demonstrations, particularly on RISC-V, used the same-process or same-address-space scenario [2], [4]. To further the spectre research on the RISC-V platform, we proposed a cross-process scheme that separates the adversary and victim. Assume that the attacker does not have administrative privileges but has the ability to execute any code on the machine (controlled by the untrusted host). The attacker attempts to obtain data from another process. In particular, the victim would play as a server, and the attacker is a remote client. To leak data, an attacker must send specially controlled inputs to the target, which are then forwarded to a spectre-vulnerable function. The attacker must first map the target binary into its own address space (for example, using *mmap()*). The attacker will then share all read-only and non-dirty pages with the victim as in Figure 2.

The attacker must then locate two memory locations in the victim binary that satisfy the requirements of the Flush+Reload technique:

1) The offset in the operand file used to perform the spectre condition to flush it. For instance, in the bound check bypass scenario, this would be the offset of *a_size*. This is used to force the CPU to rely on the Pattern History Table (PHT), as the adversary has to fetch the operand from DRAM before resolving it.

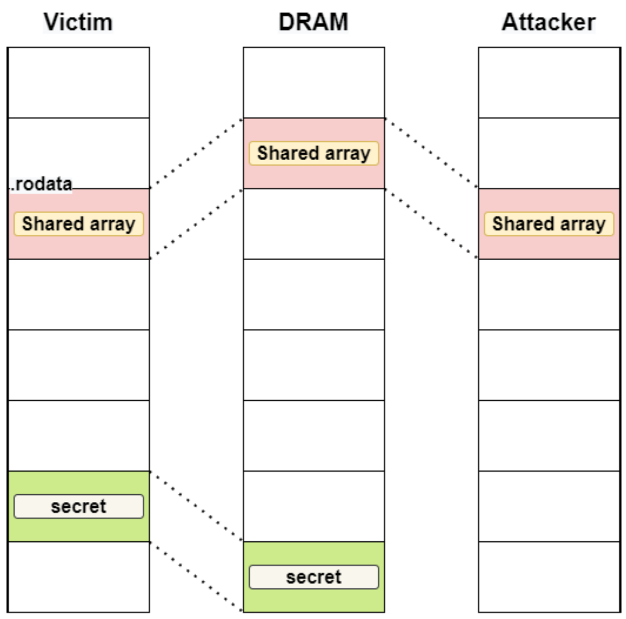


FIGURE 2. Mapping to victim binary & shared buffer.

2) The offset in the read-only/non-dirty memory location file is used to leak the read bytes in a spectre attack. This would be the offset of the victim's shared array with the attacker.

After this, the attacker will compute the virtual addresses for both memory locations in its own address space by simply adding the offsets to the address to which the binary was mapped. The attacker now recognizes that the target physically resides at those two addresses. The adversary then flushes the memory location used to store the bytes leaked from the cache hierarchy. Because the leak will occur in this zone, it is crucial to keep it clean to prevent false positives. After that, valid inputs would be sent to the victim to train, conditioning the PHT to accept valid values. Before sending the attack input, the attacker must flush the value used to compute from the cache hierarchy. A spectre-vulnerable conditional jump will initiate the leak. Otherwise, the value will almost certainly be cached, and the CPU will avoid performing speculative execution on the target state. The attacker can then trigger the spectre attack by sending a rogue input that speculatively reads and caches the secret. Finally, to retrieve the hidden byte, the adversary must determine the access time to the shared memory location where it is stored. The real value of the secret is decided by the index that has the fastest access time.

C. PREVIOUS SOFTWARE MITIGATION APPROACH

Prior hardware-based solutions on defending against spectre required complex customization on hardware with time-consuming effort. Software-based mitigation is used as a simple and quick patch for this problem. There is some basic method that focuses on strengthening spectre target on victim application to prevent these kinds of cache side-channel attack.

1) FENCE INSTRUCTION

To cope with the spectre variants v1 and v2, both Intel and AMD have introduced microcode update patches that present new fence instruction [29]. This instruction would insert a memory barrier to prevent speculative execution in proper places. Before the previous instruction is executed, the latter instruction will be stopped from executing speculatively.

The RISC-V ISA contains a relaxed memory model, with the *fence* instruction used to establish ordering constraints. To prevent the cache attack on the RISC-V processor, we have inserted the fence instruction into the victim function to prevent speculative execution from happening. This allows the program to protect the private key from being faulty-loaded into the memory and stop the adversary. The fence instruction could be used in both Bound Check Bypass and Branch Target Injection scenarios to defend against spectre.

2) SPECULATIVE LOAD HARDENING

Despite preventing the spectre, the *fence* instruction has entirely blocked the speculative mechanizes and significantly decreased the system's performance. Speculative load hardening (SLH) [30] proposed a sufficient approach, especially with the conditional branch. SLH assured that the program would load a safe constant instead of user-controlled data in the misspeculated execution. To enable SLH to work, platforms need to provide a sufficient set of instructions for tracking and masking that are not subject to prediction. Using a global predicate state *predicate_state*, which is all-ones if a misprediction has occurred, or all-zeroes if one has not. The effect is that an address can never be caused to point to a user-controlled secret via misprediction. Because in the event of misprediction, no variables will be user-controlled at all, and no sensitive information could be exploited by spectre.

3) VALIDATING SOFTWARE MITIGATION METHODS ON RISC-V

We have re-compiled the victim program in the spectre scenario and tested the above mitigation methods to validate the proposed defence mechanism in the RISC-V processor. The experiment was done on the VC707 FPGA board with the proposed RISC-V processor, which was previously vulnerable to the spectre attack. The software-based shield will be able to stop spectre attacks from targeting vulnerable code.

Though these software-based defence mechanisms proved that they could be implemented on RISC-V and could stop the spectre attack, they require recompiling the program with a trade-off in significant performance loss. Figure 3 checked the performance of a function containing a ''spectre gadget'' conditional branch. We executed the function in a loop in which each iteration contains nine times of normal run and one time that the attacker tries to exploit the function. Here, normally the function takes about 210 cycles for execution, but when we apply the fence mitigation method, the

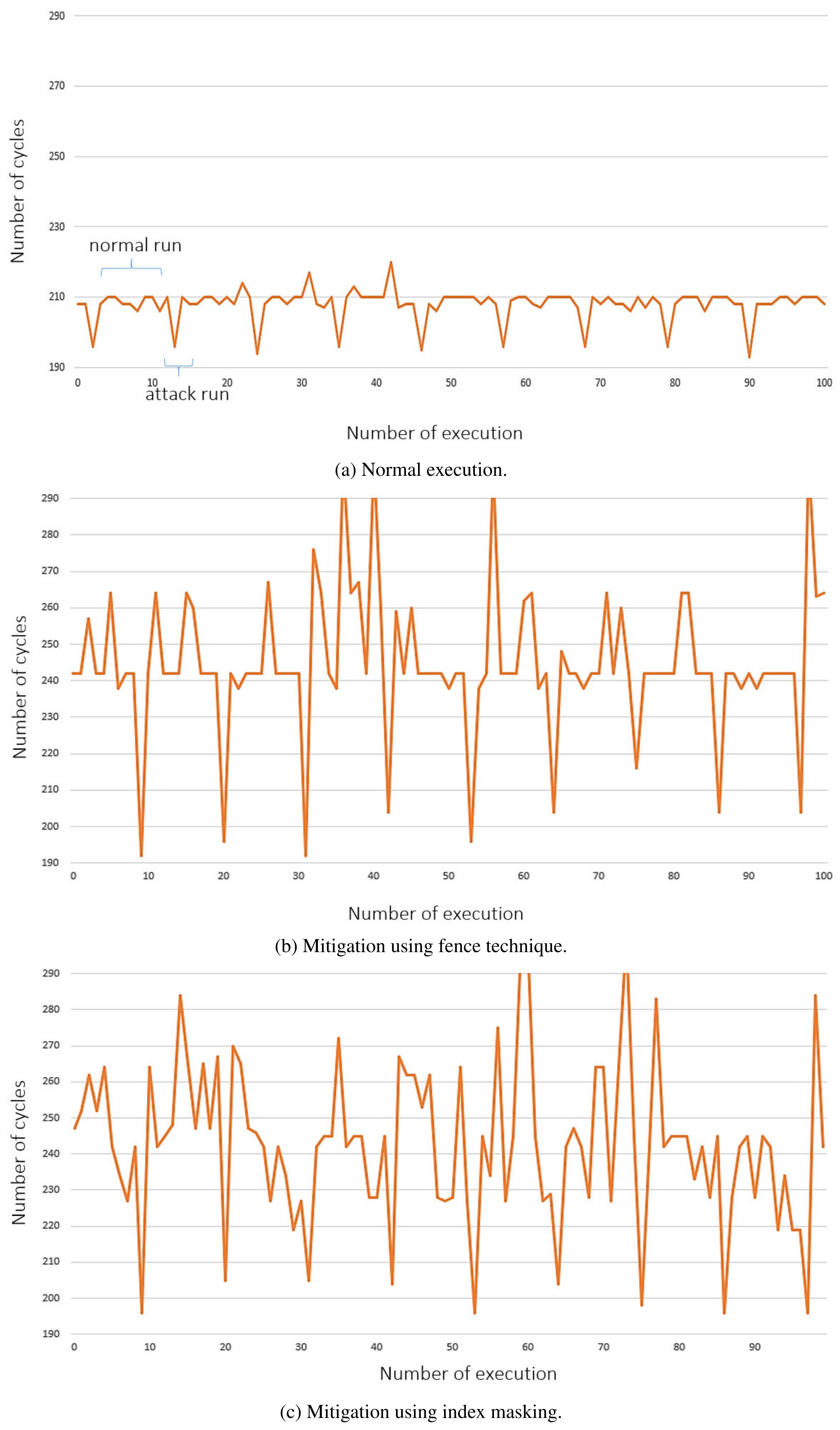
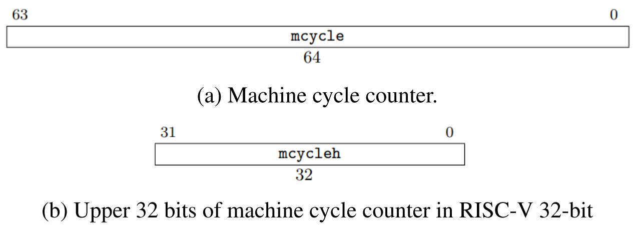


FIGURE 3. **Performance of a function with/without software-based defense mechanisms.**

execution time could increase by about 15%∼43%. Similarly, when using index masking or speculative load hardening, the average performance overhead is about 10%∼38%. Therefore, we need to suggest a different solution that is still



(a) Machine cycle counter.

(b) Upper 32 bits of machine cycle counter in RISC-V 32-bit

FIGURE 4. Example of RISC-V counter.

simple as those software methods but without increasing the performance overhead.

D. RISC-V HARDWARE PERFORMANCE MONITOR

While Intel and ARM offer proprietary performance monitoring solutions that allow software developers to get the most out of their hardware, RISC-V is still reliant on custom/vendor-specific solutions, with no complete support for common performance monitoring software tools such *PAPI* or *Oprofile*. RISC-V, in particular, is also still not fully supported by the Linux kernel monitoring tool *Perf*.

In the first RISC-V privileged specification, the ISA implemented three fixed counters: Time, Cycle, and Retired Instructions (instret). The corresponding machine Control and Status Register (CSR) are *mcycle*, *mtime*, *minstret*. This installation enabled baseline performance monitoring of a RISC-V implementation, calculating the average number of instructions for each clock cycle. The counter, for example, *mcycle* has 64 bits value on RISC-V 32-bit, 64-bit and 128-bits system as in Figure 4. On RISC 32-bit system, the value of *mcycle* would be only the low 32 bits. On the other hand, the bits at the position 63 to 32 could be read by the counter *mcycleh*.

Currently, RISC-V ISA specification [31] supports for 32 performance monitor registers in total. The Hardware Performance Monitor (HPM) counters, numbered *hpmcounter3* to *hpmcounter31* as in Table 3. The first three counters are still time, cycle and instret. Those HPCs can be individually modified by setting an event identification in the associated hpmevent registers, a set of XLEN-bit registers.

For now, the RISC-V HPM has been substantially simple and not fully supported by standard performance analysis tools such as *Perf*. Despite this, the RISC-V HPM standard is a versatile general performance monitoring solution, and its open-source nature provides a high degree of implementation flexibility. Given the RISC-V privileged specification condition, we developed our custom monitor system to record the performance counter on the out-of-order RISC-V processor BOOM. This method would be used as a quick and effective solution for the spectre problems in the RISC-V processor before a complex hardware-based method is presented.

To provide access to the counters of the processor, we need to be able to access the chosen CSRs. Therefore, the programmer would have the ability to directly read the setting or counter values using the RISC-V ISA's software structures. For developing the RISC-V spectre detection system, we need

TABLE 3. RISC-V default hardware performance counter value.

Counter Event	Counter parameter
Cycle	mcycle
Time	mtime
Instructions Retired	minstret
Exception taken	mhpmcounter3
Integer Load instruction retired	mhpmcounter4
Integer Store instruction retired	mhpmcounter5
Atomic memory operation retired	mhpmcounter6
System instruction retired	mhpmcounter7
Integer arithmetic instruction retired	mhpmcounter8
Conditional branch retired	mhpmcounter9
JAL instruction retired	mhpmcounter10
JALR instruction retired	mhpmcounter11
Integer multiplication instruction retired	mhpmcounter12
Integer division instruction retired	mhpmcounter13
Floating-point load & store instructions retired	mhpmcounter14
Floating-point other instructions retired	mhpmcounter15
Load-use interlock	mhpmcounter16
Long-latency interlock	mhpmcounter17
CSR read interlock	mhpmcounter18
Instruction cache/ITIM busy	mhpmcounter19
Data cache/DTIM busy	mhpmcounter20
Branch direction misprediction	mhpmcounter21
Branch/jump target misprediction	mhpmcounter22
Pipeline flush from CSR write	mhpmcounter23
Pipeline flush from other event	mhpmcounter24
Integer multiplication interlock	mhpmcounter25
Floating-point interlock	mhpmcounter26
Instruction cache miss	mhpmcounter27
Data cache miss or memory-mapped I/O access	mhpmcounter28
Data cache writeback	mhpmcounter29
Instruction TLB miss	mhpmcounter30
Data TLB miss	mhpmcounter31

to encounter three parts as in Figure 5. As illustrated, we have to configure from processor configuration to Linux kernel to take advantage of the current RISC-V ISA for our detection system.

To begin, we must incorporate the appropriate counter into the CPU setup. As BOOM is a highly customizable open-source project, we can conveniently add or remove desired Hardware Performance Events (HPEs) by modifying the event set and specific events in the project's Scala code.

Second, the Linux kernel needs to be modified as well. To initialize the required performance counter, we change the machine initialization process configuration during the operating system boot phase. Additionally, there is a single 32-bit counter enable register *mcounteren*. Each bit in this register is associated with one of the 32 event counters discussed previously. This register solely regulates the counter registers' visibility and allows the user to view the HPC value. Additionally, we must match the events to the counters. Each *mhpmevent#* would correspond to a unique counter register.

Finally, a utility has been developed to read the counter value from software level and analyze the data.

IV. PROPOSED DESIGN

A. DETECTION PROCEDURE

The BOOM processor runs a light Linux operating system on an FPGA as in Figure 6. As a result, we could create

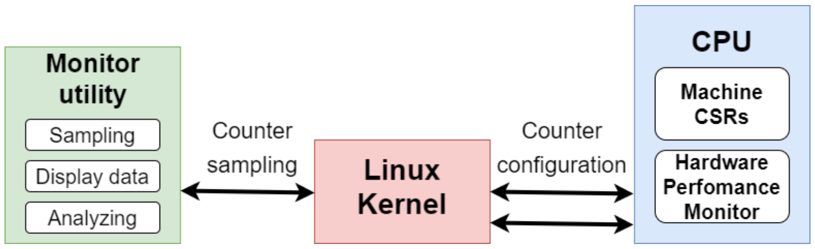


FIGURE 5. HPM Monitor software structure.

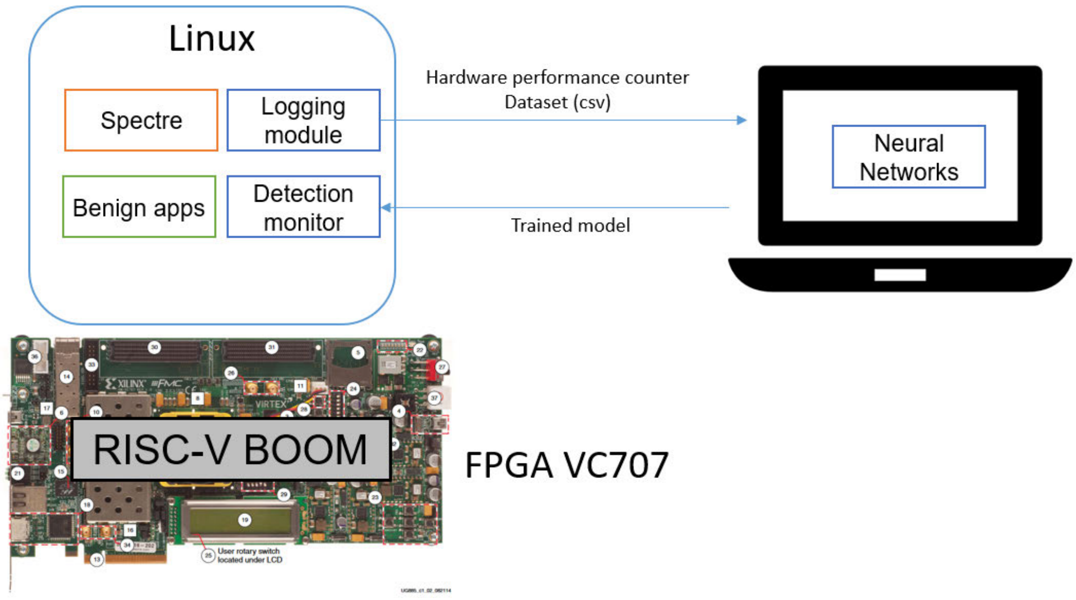


FIGURE 6. System model.

a real-time monitor to retrieve the hardware performance counter and detect a cache side-channel attack. The services would run in the system's background. We ensure that each test case executes only sample applications and the logging service on the core. Additionally, we created an idle test case in which only the logging service was executed for the dataset. Our detecting procedure is divided into three stages:

- *Collecting Data Sets:* We profile spectre and other benign processes in Attack and Non-Attack scenarios during this phase. At a sampling rate of 100ms, we collect performance counter values. The system would execute only one sample application and the logging service during each sampling period to exclude noise. We independently monitor appropriate hardware counters (mentioned in section III.D) for profiling, directly relevant to distinguishing benign programs from malicious processes. The collected dataset was saved in a .csv file.
- *Detection Model Training:* We train the machine models employed in the spectre detection technique during the detection model training phase. We use a data set of 20,000 samples to train the machine learning models. We blended data samples from benign and malicious processes and trained the models on labelled data. As we develop the neural network model using C, the model could be trained on either a regular Intel or ARM processor or our proposed RISC-V system, which contains a RISC-V GNU toolchain.
- *Profiling at Runtime:* We connected the trained model to the monitor services. At runtime, we monitor hardware events. We also choose a sampling time interval of 100ms to incur the slightest detection overhead. Also, because this phrase uses a pre-trained model, it would be an immediate computation. Therefore, the attack could be detected within around 100ms. When the system detects a cache side-channel attack, the services notify the user.

B. HARDWARE PERFORMANCE COUNTER AS FEATURES

As described in Section III-B, most attackers use spectre in one of two ways. In the first variation, the attacker mis-trains branch instructions to speculatively execute unprivileged instructions, whereas, in the second variant, the attacker targets the indirect branch. Based on the features selection criteria, we choose performance counters linked to both

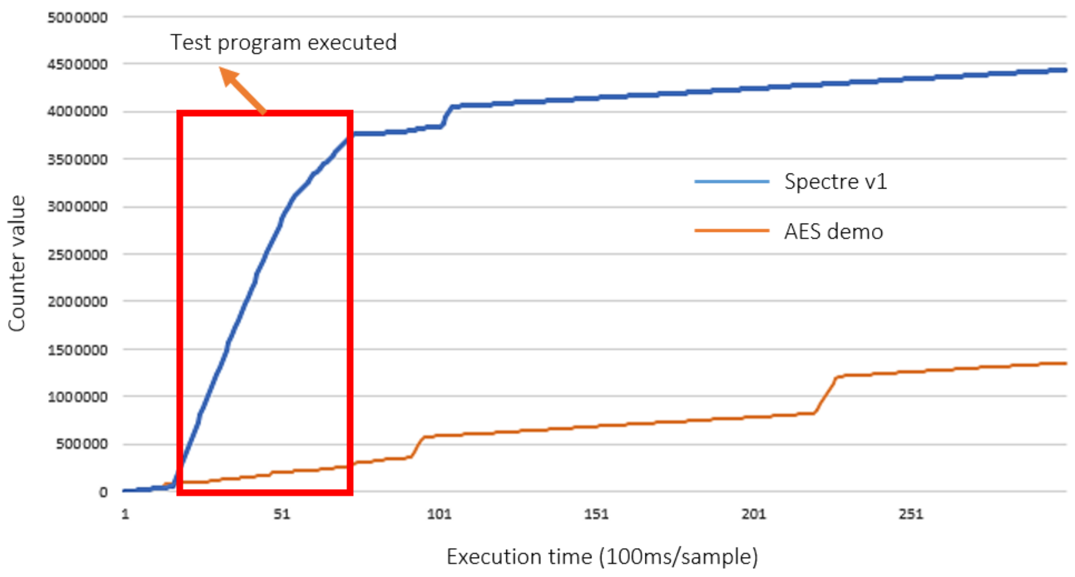


FIGURE 7. Branch misprediction counter.

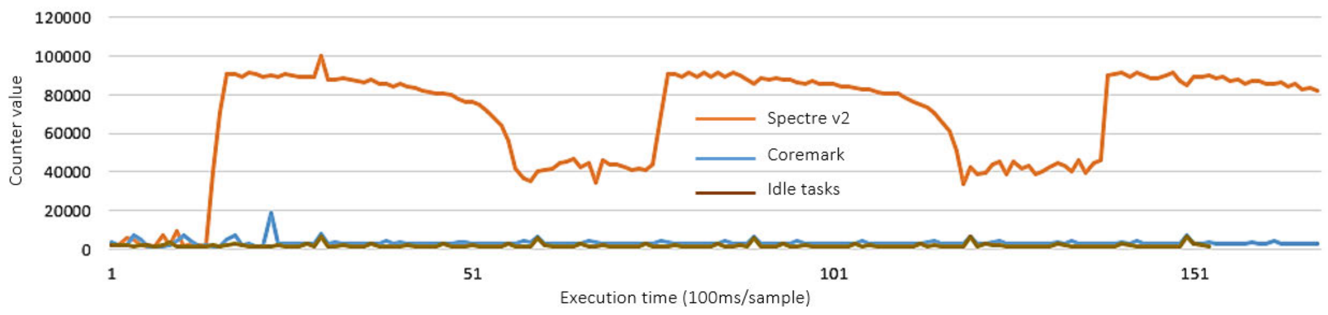


FIGURE 8. Instruction TLB miss counter.

variants of spectre attacks, which are Branch misprediction and iTLB miss.

To validate our selected counter, we proposed experiment that compared the differences in counter's value when the spectre attack was performed to the value when the usual program was executed as shown in Figure 7 and 8. As illustrated in the Figure 7, the branch misprediction counter increases significantly when spectre-v1 occurs in comparison to performing a standard AES calculation. The Figure 8 shows how the instruction TLB miss counter changes over time. The three scenarios depict the execution of spectre-v2, coremark, and idle jobs, respectively. Obviously, the value of spectre-v2 is significantly greater than the values in the other two cases. As a result, these two counters are a good match for identifying spectre attack events.

In addition to TLB and branch-related events, we choose the total number of Retired instruction events and cycle because it demonstrates the workload that a particular process places on the CPU to generate related cache misses, cache accesses, and branch miss predictions. Because the spectre attack consists of a shorter loop that conducts branch mispredictions and cache accesses regularly. Spectre attacks are likely to have a more significant rate of cache misses and branch mispredictions in relation with the total number of executed instructions than other benign processes. Table 4 shows four counters used for detecting cache side-channel attacks in RISC-V.

TABLE 4. Selected counter event.

Counter Event	RISC-V Counter parameter
Cycle	mcycle
Instructions Retired	minstret
Branch misprediction	mhpmcounter21
TLB miss	mhpmcounter30

C. MULTILAYER PERCEPTRON NETWORK

When it comes to classification tasks, artificial neural networks have proven to be extremely effective. Classification's purpose is to specify which category a given input belongs to. In the case of detecting spectre attacks, the input is collected HPC data from several processes, and the output is divided into two categories: benign and dangerous. It is feasible to reliably classify complex data without manually constructed

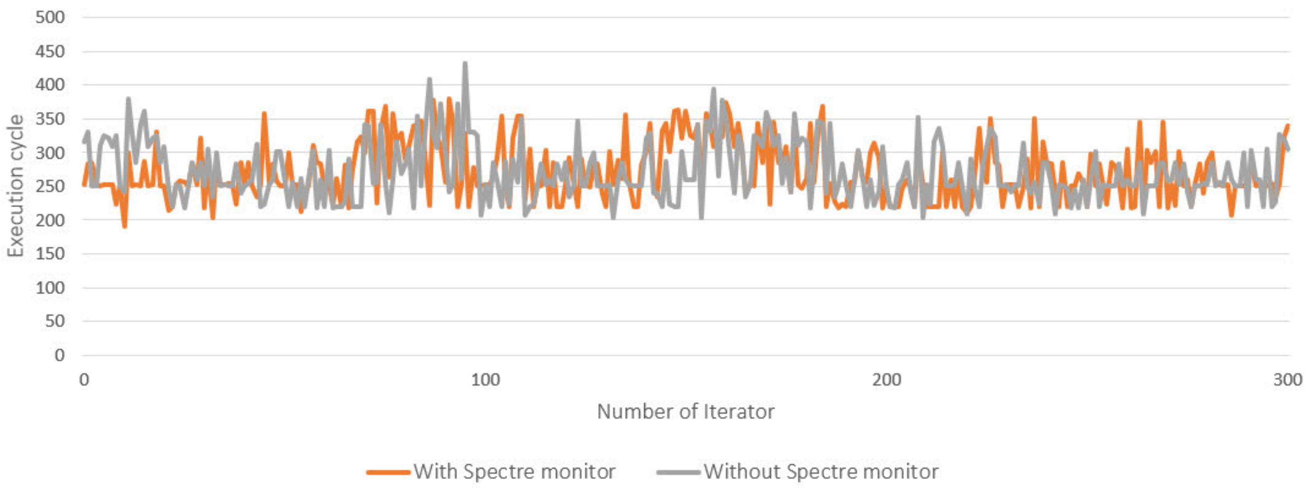


FIGURE 9. Execution time of with/without detection system.

features due to neural networks' ability to learn nonlinear relations of the input set.

Because our dataset contains solely labeled counter values, a simple neutral network is sufficient for developing an appropriate detection model. As a result, we choose a multi-layer perceptron network implementation. The multi-layer perceptron was formerly thought to provide a nonlinear mapping between an input vector and its associated output vector. It commonly provides high performance on detection system [32]. Its advantage is that it maintains the same accuracy ratio even when the dataset is smaller.

This neural network comprises four input neurons, one for each of the four HPCs collected. It has three fully linked hidden layers with 3, 5, and 7 neurons each and one output neuron. The output neuron uses a Rectified Linear activation function as in listing 2 because it is easier to train and commonly give a better performance. While we experimented with several network designs, this one produced the best results with our data set without triggering over-fitting.

```
f(x) = max(0,x)
```

LISTING 2. Rectified linear activation function (ReLU).

V. EXPERIMENT SETUP AND RESULTS

A. DATA COLLECTION

To train the supervised learning model, we created a data set comprised of HPC data collected from various benign processes and malicious spectre implementations. These data points are classified as either benign or malicious. Instead of accumulating readings from the entire CPU, we employ performance counters attached to each process. This separation enables the model to classify each process as benign or malicious. Based on the neural network's predictions, a detection system can then take action per process. For example, the system can notify the user when an application behaves suspiciously or kill a malicious process.

TABLE 5. Process dataset.

Type	Program	Number of sample
Malicious	Bound Check Bypass	1980
	Indirect Branch Injection	1723
	spectre cross-process	2150
	Prime+probe attack [35]	1684
	spectre-image [36]	2614
Benign	Idle tasks	2992
	Auto (basic math, bitcount, qsort...)	1545
	Network (dijkstra...)	1600
	Security (blowfish, aes, sha...)	2048
	Coremark	1904

To obtain performance counter data, we create an HPCs monitor. We take the measurement every 100ms to avoid degrading the performance too much. We implement and compile our own RISC-V application dataset, which includes five spectre attack scenarios, benchmark processes Mibench [33] and Coremark [34]. The total number of samples collected is around 20.000 records as specified in Table 5.

B. PERFORMANCE OVERHEAD

We perform experiments with a sampling rate of 100ms to extract performance counter values. Increasing sampling rate increase the detection speed. However, it will increase the detection overhead. Our detection mechanism offers collecting samples at a suitable frequency for being able to detect cache side-channel attacks relatively early with minimal performance overhead.

To validate the performance, we benchmark the victim program in two scenarios: with and without the detection system running. We run the scenario 300 times and compare the average execution time of two different scenarios. As illustrated in Figure 9, there would be only a little difference in the execution time of the target application in the two scenarios. The average overhead for performance is only 2.25 percent.

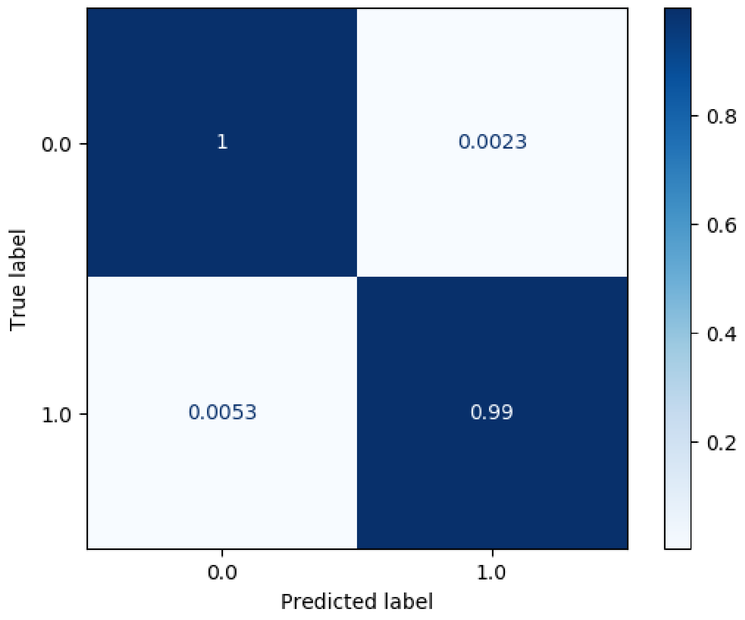


Fig. 10. Confusion matrix (normalize over the true (rows)).

TABLE 6. False prediction results.

Type	Program	False prediction (%)
Malicious	Bound Check Bypass	0
	Indirect Branch Injection	0
	spectre cross-process	**1.3**
	Prime+probe attack	0
	spectre-image	0
Benign	Idle tasks	0
	Auto (basic math, bitcount, qsort...)	0
	Network (dijkstra...)	0
	Security (blowfish, aes, sha...)	**0.6**
	Coremark	0

TABLE 7. Detection results comparison between different model.

Model	Accuracy (%)	FP(%)	FN(%)
MLP (this work)	99.61	0.27	0.12
LogisticRegression	99.04	0.74	0.22
SVC	94.19	5.7	0.11
Linear SVC	94.83	5.1	0.07

C. DETECTION RESULTS

We collected data to measure the performance of our detection system neural network after training it with the data set described in Section III.C. We split up 20% of our data set that was not used for training to validate our trained network with data that has not been used. Figure 10 shows the results of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) (FN). While the accuracy is very high, there are still a few false positives and false negatives. We have detailed the results to determine which samples were incorrectly predicted as in Table 6. The false positives happen because their sample is from both a cross-process attack set and a stand-alone sample from the victim process. As for the false negatives, which are mispredicted benign processes as an attack, are samples from the cryptography algorithm calculation process, as our attacks normally target cryptography operations.

Due to dataset limitations, we also validate the proposed neutral network using k-fold cross-validation to ensure accuracy. The k-fold cross-validation approach is a widely used technique for assessing the performance of a machine learning algorithm on a limited dataset. We simulate our case using leave-one-out cross-validation (LOOCV), a computationally expensive cross-validation version with k=N and N equaling the total number of examples in the training dataset. For each sample in the training set, an example is provided to be used as the test evaluation dataset on its own. The mean classification accuracy for various k values may then be compared to the mean classification result for the same dataset obtained using LOOCV. Figure 11 shows a line plot comparing the mean accuracy scores to the LOOCV result, with the min and max of each result distribution indicated by error bars. The results indicate that for this model on this dataset, most k values scenarios under-perform when compared to the ideal case. The average accuracy of cross-validation method is nearly 99.6%.

Table 7 shows experimental results of cache side-channel detection using four machine learning models. We have also implemented and trained our dataset on Logistic Regression and Support Vector Machine models. The work is done using python and scikit-learn library. The results show that our proposed multi-layer perceptron recorded the highest accuracy. Logistic Regression could also use as a suitable for analyzing the performance counter value.

We also compare our results with prior researches in Table 8. With our proposed tool and neural network model, we could still keep the high accuracy on detecting cache

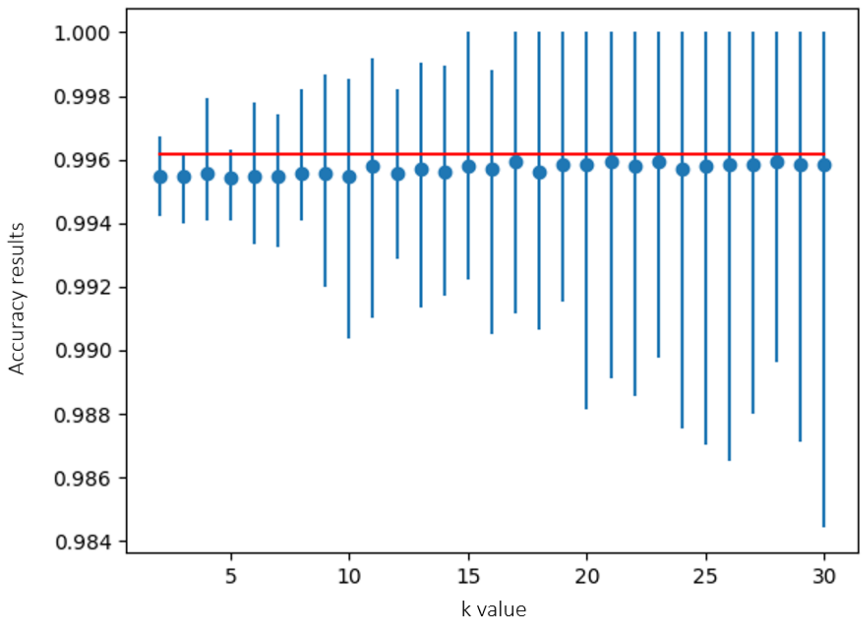


Fig. 11. Mean accuracy for cross-validation k-values. (Blue: Each case error bar, Red: Ideal case result).

TABLE 8. Results comparison with prior researches.

Research	Target	Tools	Accuracy
[18]	Intel	Perf & PAPI	F1-score: 0.98
[19]	Intel	PAPI	95-99%
[20]	Intel	Intel CMT	F1-score: 0.67
[21]	Intel	Intel PCM	95%
[22]	Intel	Perf	90-99%
[23]	ARM & x86	ARM PMU & Perf	99%
This work	RISC-V	Proposed tool	94-99.61% F1-score: 0.9961

side-channel attacks on RISC-V processors as other studies on Intel or ARM processor use pre-existing tools.

VI. RESEARCH LIMITATIONS AND FURTHER DISCUSSION

While our detection service has an excellent detection rate, it does not address harmful processes once they are found. They would only notify the user when an attack occurred. However, we may increase the viability of detection services by coupling them with an application that manages these harmful processes. To solve these challenges of wrongly detection cases, our detection service forwards the id of processes that are expected to be harmful to an application that determines what to do with them. Instead of terminating the process, one possibility is to pause it and pass the decision to the user. This would ensure that no processes are terminated as a result of incorrect prediction in the event of FP or FN. While it is the role of a mitigation strategy to manage the findings of a detection service, combining mitigation and detection techniques can result in improved detection results and a reduction in FP and FN.

Additionally, due to the limitations of RISC-V hardware performance tools, we struggle on conducting a thorough analysis of the effect of cache side-channel attacks on other performance counters. We believe that by combining the characteristics of each counter provided by the RISC-V architecture, the system could identify more types of cache attacks in addition to the spectre attacks stated above.

VII. CONCLUSION

This paper proposes monitoring micro-architectural feature deviations to detect spectre attacks, which exploit vulnerabilities in modern processors hardware design such as speculative execution and cache side channels. On the RISC-V architecture, we implement a run-time detection system for spectre attacks. The characteristics are derived from various RISC-V applications' hardware performance counters values, both malicious and benign. The experimental results of a trained multi-layer perceptron classifier show a good detection accuracy of more than 99%. Also, the performance overhead is only 2%, which is the same with other hardware mitigation methods. Because comprehensive spectre mitigation is difficult, it is now more practical to detect such attacks directly and in real-time. We believe that our presented detection system will detect other spectre variants and any other attack that exploits branch predictors, out-of-order execution, and cache-based side-channel attacks on the RISC-V architecture.

REFERENCES

[1] O. Acıçmez and C. C. K. Koç, *Microarchitectural Attacks Countermeasures*. Boston, MA, USA: Springer, 2009, pp. 475–504, doi: 10.1007/978-0-387-71817-0_18.

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1–19.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proc. 27th USENIX Conf. Secur. Symp.*, 2018, pp. 973–990.

[4] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *Proc. 28th USENIX Secur. Symp. (USENIX Security)*, Santa Clara, CA, USA, Aug. 2019, pp. 249–266. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/canella

[5] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices," IACR Cryptol. ePrint Arch., Tech. Rep. 2016/980, 2016.

[6] A.-T. Le, B.-A. Dao, K. Suzaki, and C.-K. Pham, "Experiment on replication of side channel attack via cache of RISC-V Berkeley out-of-order machine (BOOM) implemented on FPGA," in *Proc. 4th Workshop Comput. Architecture Res. RISC-V*, 2020, pp. 1–4.

[7] A. Gonzalez, "Replicating and mitigating spectre attacks on a open source RISC-V microarchitecture," in *Proc. 3rd Workshop Comput. Archit. Res. (RISC-V)*, Phoenix, AZ, USA, Jun. 2019.

[8] C. Su and Q. Zeng, "Survey of CPU cache-based side-channel attacks: Systematic analysis, security models, and countermeasures," *Secur. Commun. Netw.*, vol. 2021, pp. 1–15, Jun. 2021, doi: 10.1155/2021/5559552.

[9] K. N. Khasawneh, E. M. Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "SafeSpec: Banishing the spectre of a meltdown with leakage-free speculation," in *Proc. 56th Annu. Design Automat. Conf.*, Jun. 2019, pp. 1–6.

[10] Y. Mao, V. Migliore, and V. Nicomette, "REHAD: Using low-frequency reconfigurable hardware for cache side-channel attacks detection," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Sep. 2020, pp. 704–709.

[11] J. Zhao, "Sonicboom: The 3rd generation Berkeley out-of-order machine," in *Proc. 4th Workshop Comput. Archit. Res. (RISC-V), Virtual Workshop*, May 2020.

[12] J. Fustos, F. Farshchi, and H. Yun, "Spectreguard: An efficient data-centric defense mechanism against spectre attacks," in *Proc. 56th Annu. Design Automat. Conf.*, ser. DAC '19. New York, NY, USA, 2019, pp. 1–6, doi: 10.1145/3316781.3317914.

[13] E. M. Koruyeh, S. H. A. Shirazi, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "SpecCFI: Mitigating spectre attacks using CFI informed speculation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 39–53.

[14] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya. *Performance Counters to Rescue: A Machine Learning Based Safeguard Against Micro-Architectural Side-Channel-Attacks*. Cryptology ePrint Archive, Report 2017/564, 2017. [Online]. Available: https://ia.cr/2017/564

[15] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *Proc. 19th Int. Symp. RAID*. Paris, France: Springer, 2016.

[16] M. Yu, B. Halak, and M. Zwolinski, "Using hardware performance counters to detect control hijacking attacks," in *Proc. IEEE 4th Int. Verification Secur. Workshop (IVSW)*, Jul. 2019, pp. 1–6.

[17] X. Wang and R. Karri, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 3, pp. 485–498, Aug. 2016.

[18] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, vol. 49, pp. 1162–1174, Dec. 2016.

[19] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "NIGHTs-WATCH: A cache-based side-channel intrusion detector using hardware performance counters," in *Proc. 7th Int. Workshop Hardw. Architectural Support Secur. Privacy*, New York, NY, USA, Jun. 2018, pp. 1–8, doi: 10.1145/3214292.3214293.

[20] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through Intel cache monitoring technology and hardware performance counters," in *Proc. 3rd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, 2018, pp. 7–12.

[21] J. Cho, T. Kim, T. Kim, and Y. Shin, "Real-time detection on cache side channel attacks using performance counter monitor," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2019, pp. 175–177.

[22] C. Li and J.-L. Gaudiot, "Detecting spectre attacks using hardware performance counters," *IEEE Trans. Comput.*, early access, May 21, 2021, doi: 10.1109/TC.2021.3082471.

[23] R. Oshana, M. A. Thornton, E. C. Larson, and X. Roumegue, "Real-time edge processing detection of malicious attacks using machine learning and processor core events," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2021, pp. 1–8.

[24] K. Asanović, D. A. Patterson, and C. Celio, "The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," Dept. EECS, Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-167, 2015.

[25] Y. Yarom and K. E. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proc. USENIX Secur. Symp.*, 2014, pp. 719–732.

[26] D. Wang, Z. Qian, N. Abu-Ghazaleh, and S. V. Krishnamurthy, "PAPP: Prefetcher-aware prime and probe side-channel attack," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[27] C. Duran, T.-T. Hoang, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Tee boot procedure with crypto-accelerators in RISC-V processors," in *Proc. 4th Workshop Comput. Archit. Res. (RISC-V), Virtual Workshop*, May 2020.

[28] T.-T. Hoang, C. Duran, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Cryptographic accelerators for trusted execution environment in RISC-V processors," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–4.

[29] *Intel Analysis of Speculative Execution Side Channels*, Intel, Mountain View, CA, USA, 2018.

[30] C. Carruth. (2019). *Speculative Load Hardening*. [Online]. Available: https://llvm.org/docs/SpeculativeLoadHardening.html

[31] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovic. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.9*. EECS Department, University of California, Berkeley. Jul. 2016. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[32] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Raheem, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[33] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization (WWC-4)*, Dec. 2001, pp. 3–14.

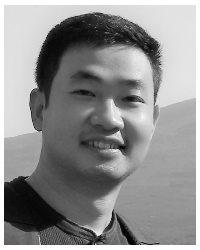
[34] S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," Embedded Microprocessor Benchmark Consortium, USA, Tech. Rep., 2012. [Online]. Available: https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf

[35] C. Trippel, D. Lustig, and M. Martonosi, "MeltdownPrime and SpectrePrime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols," 2018, *arXiv:1802.03802*.

[36] S. Yadav. (2020). *SPECTRE ATTACK Variant 1 on Images*. [Online]. Available: https://github.com/yadav-sachin/spectre-attack-image

ANH-TIEN LE (Graduate Student Member, IEEE) received the M.S. degree in information systems from the Hanoi University of Science and Technology, Vietnam, in 2019. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. He is also a Lecturer with the Academy of Cryptography Techniques (ACT), Hanoi, Vietnam.

TRONG-THUC HOANG (Graduate Student Member, IEEE) received the B.Sc. degree in electronics and telecommunications and the M.S. degree in microelectronics from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2012 and 2017, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. He is also a Research Assistant with the National Institute of Advanced Industrial Science and Technology (AIST), Tokyo.



BA-ANH DAO (Graduate Student Member, IEEE) received the B.Sc. degree in electronics and telecommunications and the M.S. degree in microelectronics from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2014 and 2019, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. He is also a Research Assistant with the Academy of Cryptography Techniques (ACT), Hanoi.

KUNIYASU SUZAKI (Member, IEEE) received the B.E. and M.E. degrees in computer science from the Tokyo University of Agriculture and Technology and the Ph.D. degree in computer science from The University of Tokyo, Tokyo, Japan. He is currently a Senior Researcher with the National Institute of Advanced Industrial Science and Technology (AIST) and a Researcher with the Technology Research Association of Secure IoT Edge Applications Based on the RISC-V Open Architecture (TRASIO). His research interests include security on CPUs, operating systems, and hypervisors.

AKIRA TSUKAMOTO received the M.S. degree in computer science from Columbia University, New York, NY, USA. He currently works with the National Institute of Advanced Industrial Science and Technology (AIST). He has worked on products based on Cell/B.E. and ARM. His main research interests include software engineering on a network, operating systems and system security, and he is enthusiastic regarding any kind of technical development.

CONG-KHA PHAM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Sophia University, Tokyo, Japan. He is currently a Professor with the Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Tokyo. His research interests include the design of analog and digital systems using integrated circuits.