International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 13, Number 2, pages 149–172, July 2023

Low Resource and Power Consumption and Improved Classification Accuracy for IoT Implementation of a Malware Detection Mechanism using Processor Information

Mutsuki Deguchi
University of Nagasaki
1-1-1, Manabino, Nagayo-cho, Nagasaki, Nagasaki, 851-2130, Japan

Masahiko Katoh
University of Nagasaki
1-1-1, Manabino, Nagayo-cho, Nagasaki, Nagasaki, 851-2130, Japan

Ryotaro Kobayashi
1-24-2, Nishi-shinjuku, Shinjuku, Tokyo, 160-0023, Japan

Received: February 14, 2023
Revised: April 28, 2023
Accepted: June 6, 2023
Communicated by Yasuyuki Nogami

Abstract

In recent years, the demand for Internet of Things (IoT) devices has increased dramatically. They are used in a variety of applications, such as sensors, in-vehicle terminals, control devices in factories, and medical equipment in hospitals. However, attackers are also increasingly targeting IoT devices, making it necessary to implement security countermeasures. However, IoT devices rarely have sufficient resources like high-end products such as PCs and servers. Therefore, we are developing a system that can detect malware with lower resources. In this study focusing on targeting the RISC-V architecture, we measure the classification accuracy of a detection mechanism using random forests and reduce the circuit scale and power consumption. We show that the detection mechanism can be made more accurate in classifying normal programs and malware and cost of the detection mechanism can be reduced.

Keywords: Machine Learning, Internet of Things, RISC-V, Malware Detection, Hardware Security

1 INTRODUCTION

In recent years, small-scale devices such as sensors, in-vehicle terminals, and medical devices have begun to be connected to the Internet. These devices form the Internet of Things (IoT), which facilitates the collection, analysis, and utilization of information from a wide range of sources. In addition, due in part to the influence of COVID-19, unprecedented new online-based work systems, such as remote conferencing using webcams, are also beginning to be established. The number of

[0] The conference version of this paper is published in the proceedings of The Tenth International Symposium on Computing and Networking (CANDAR 2022).

services utilizing such IoT devices is increasing every year, and the number of small-scale devices connected to the Internet is reported to be in the tens of billions [20]. However, while the convenience of IoT devices has improved as their number has expanded, there are still some devices that have been in use for a long time, have been out of service for a long time, have not been updated, or retain vulnerabilities in their firmware, etc., or whose password settings have not been changed since their initial state when deployed, etc. From a security perspective, some devices are exposed to external dangers. Therefore, attacks on IoT devices that take advantage of these vulnerabilities and inadequacies cannot be overlooked.

Malware such as Mirai and Bashlite infect vulnerable IoT devices, create botnets, and then cause malfunctions by simultaneously communicating with devices that provide services. Furthermore, since the source codes of these malwares are available on the Internet, many and more dangerous versions and variants of them have been reported [16].

Therefore, IoT devices must be equipped with adequate security measures to prevent infection from such malware. Various software-based methods have been proposed for common malware detection [11, 18, 23, 7]. However, IoT devices place high importance on long-term operation with low resources and are subject to strict constraints on resource usage. This makes application of these methods difficult in terms of resources.

Koike et al. and Deguchi et al. proposed a hardware-based malware detection mechanism (MDM) for devices that need to operate under such resource-constrained conditions. The MDM receives signals from each mechanism and register obtained during CPU operation, analyzes the currently running program using the processed data as features, and decides whether the program is a normal program or malware. In Koike et al.'s research, the MDM consists of three mechanisms: an access hit counter, a divider, and a classifier [15]. Furthermore, Deguchi et al. proposed a mechanism called a hit-rate table (HRTable) to solve the problems of the complicated circuit configuration of the divider, the increase in circuit scale, and the calculation speed of Koike's MDM, enabling a simple circuit configuration and fast calculation results. Each time the CPU runs one instruction, the divider and HRTable calculate the feature values and pass the calculation results to the classifier. The classifier determines whether an instruction is benign or malicious based on the obtained feature values and decides whether the program being executed is a normal program or a malicious program based on the rates of benign instructions and malicious instructions in a given number of instructions.

In this paper, we make the following two contributions toward improving the performance of the MDM:

- A method for improving the MDM's classification accuracy by adding features, while considering the effects on hardware resource consumption.
- A method to reduce the circuit scale and power consumption of the HRTable by access control for low-cost operation of the MDM.

This paper is composed of the following sections: Section 2 describes the differences from this paper's interim report; Section 3 introduces related works and discusses the MDM and HRTable in detail and their current issues; Section 4 proposes a method to address the issues in Section 3; Section 5 evaluates the method; Section 6 discusses the results of the evaluation of the method; and finally Section 7 concludes the paper.

2 RELATIONSHIP WITH PREVIOUS STUDIES

An interim report of this paper was presented at the CANDAR workshop in November 2022. In the workshop, we improved the classification accuracy and reduced the hardware scale and power consumption of HRTable. For the accuracy of classification, we are considering improving the accuracy by adding feature values to be used for classification. Furthermore, the hardware scale and power consumption of the HRTable are both reduced by improving some of the mechanisms in MDM. These proposed methods are described in Section 5.

This paper is the complete version of the interim report presented at CANDAR, with some sentences and all figures different [8]. Furthermore, the differences from the interim report include updates and additional elements. The differences between this paper and the CANDAR paper are as follows;

- Explanation
  - Addition: In order to clarify the purpose of the proposed mechanism, the number of references to related studies is increased and their differences are described (Section 3).
  - Addition: Addition of data collection techniques used for classification, processing techniques, and a list of programs used (Section 5.1.1 and Section 5.1.2).
  - Update: Modification of training parameters to implement in FPGA for evaluation (Section 5.1.3).
- Evaluation
  - Addition: In the classification accuracy measurement, we measured the accuracy when the feature values, Program Counter, were removed (Section 5.1.6 and Section 5.1.7).
  - Update: More detailed power consumption of HRTable was measured by reproducing the actual program execution trace in simulation (Section 5.2.2).
  - Addition: We measured the power consumption of the classifier, one of the mechanisms of MDM (Section 5.3).
  - Addition: We measured the hardware resource utilization and power consumption of the overall MDM. Furthermore, we added a survey on CPU performance for applying MDM to IoT devices (Section 5.4).
- Verification
  - Update: The evaluation environment for classification accuracy has been changed, and a new evaluation section has been added (Section 6.1).
  - Addition: Added discussion on evaluation of Section 5.3 and Section 5.4 (Section 6.3 and Section 6.4).

3 RELATED WORKS

Control-flow Enforcement Technology (CET) [2] and TrustZone [1] have been announced as hardware-based security measures. CET employs new technologies called shadow stack (SS) and indirect branch tracking (IBT), which detect attacks that exploit return and branch instructions, and pass an exception to the OS to alert it of an anomaly. Furthermore, ARM has developed TrustZone and trusted execution environment (TEE) technologies that classify computer resources into secure and non-secure areas and store important information in the secure areas, thereby providing a secure execution environment for small-scale devices. However, this mechanism is intended to protect data and is a different security measure from the one used in this research, which is intended to detect malware. There are a variety of hardware-based machine-learning and deep-learning methods. Kornaros focused on the effectiveness of machine-learning-based intrusion detection systems or mechanisms as a defense against cryptography and unknown threats [17]. Among others, Kornaros discussed the advantages, disadvantages, and potential of machine-learning and deep-learning-based security methodologies for IoT devices and other devices with limited hardware resources.

Furthermore, many studies have used the hardware performance counter (HPC) and machine learning to distinguish malicious programs from normal programs [13]. HPCs record events that occur in hardware and software, such as cache misses, branch prediction/misses, etc., and are conventionally used as debugging functions to monitor and improve hardware performance. Furthermore, data obtained from a HPC is considered to be difficult to tamper with from the outside, making it effective in detecting cyber-attacks.

Torres et al. classified multiple hardware events obtained from a HPC into normal and abnormal behaviors by using support vector machines (SVM) as a countermeasure against data-oriented attacks such as Heartbleed. The two-class SVM model has a detection accuracy of 92%, but the detection accuracy for unknown attacks needs to be improved [24].

Bahador et al. proposed an SVM model that uses a combination of singular value analysis and HPC. The model focuses on branch instruction predictions/mistakes among hardware events and has a detection rate of over 90% and a false positive rate of less than 1%, indicating a higher classification accuracy compared to other learning models. However, it is intended to be implemented in software and requires additional improvements for operation in the IoT [5].

Nomani et al. proposed a countermeasure against side-channel attacks using HPC and neural networks (NN) as the learning model. Here, the frequency of cache and floating-point-unit (FPU) resource sharing that occurs among multiple applications is reduced by an extended scheduling function using NN to suppress the possibility of side-channel attacks [21]. Similarly, Alam et al. presented countermeasures against side-channel attacks on microarchitectures. They built a three-stage detection system using a time-series approach to detect known and unknown attacks by examining the correlation between execution traces and cryptographic keys, thus reducing the false positive rate for normal traces [4].

Jyothi et al. proposed a framework that combines network traffic and application usage statistics with low-level statistics obtained from a HPC to mitigate damage to servers caused by DoS attacks. Here, the addition of HPC reduces the false positive rate to 0%, which could not be reduced by existing high-level statistics-only detection techniques [14].

Liu et al. proposed a software-based detection module using Intel Processor Trace (IPT) as an alternative to HPC [19]. IPT is implemented for microarchitectures and collects traces with little overhead through special instructions from the OS. Liu et al. examined the control flow of executables and libraries in two stages, fast-pass and slow-pass, and considered countermeasures against return-oriented programming (ROP).

Pattee et al. proposed a mechanism to complete the flow of hardware event collection, conversion of events to metadata for use in discrimination, and malware detection on the hardware, similar to our proposed MDM [22].

Basu et al. collected data from cache lines using an embedded trace buffer (ETB) and created learning models in software [6]. The ETB does not affect the processor and enables traces to be collected with low latency compared to HPC. Furthermore, while HPC traces are mainly traces of programs running on virtual machines, ETB traces are collected from programs running on real machines, so hardware events are collected that are very close to real-world environments.

In contrast to the above studies, which are positive about malware detection by HPC, Zhou et al. pointed out that the lower-level information obtained from HPC is not powerful enough to classify higher-level activities such as software behavior [26]. However, their experimental environment assumes Windows, which has different behavior compared to small-scale devices (i.e., Linux with minimal a configuration or no OS installed) that are the focus of our proposed MDM protection.

Similar to these related studies, Koike et al. proposed an MDM that synchronizes with the processor on large-scale integration (LSI) and uses processor information obtained during program execution for malware detection [15]. Deguchi et al. proposed the HRTable to increase the feasibility of hardware implementation of the MDM proposed by Koike et al. on small devices [9]. In the following sections, we provide an overview of the MDM and the mechanisms that compose it, and then describe the issues faced by the MDM, which we attempt to improve in this paper.

3.1 Overview of the MDM

The MDM proposed by Koike et al. is being considered for implementation on LSIs, where electronic signals from small devices are bypassed and used as feature values for malware detection. Caches are frequently used to speed up the process when CPUs execute programs. In creating learning models using machine learning, the cache hit rate is one of several signals obtained during CPU operation, and its contribution rate is high, so using this value is expected to yield high classification accuracy [15]. However, the cache hit rates are usually not calculated by the mechanism on the LSI. To calculate the cache hit rates, the number of cache accesses and hits must be totaled in software and the hit rates at that point in time must be calculated. However, in software-based systems, the overhead for calculating hit rates, which is a feature value, is large, while it is also possible that damage to the device has already shifted to a serious state at the time of detection. Therefore, the MDM has a mechanism to calculate the cache hit rates, which enables faster malware detection without requiring software processing. Furthermore, when considering the operation of the MDM in small-scale environments, reducing the dimensionality of feature values is necessary for lower resource consumption, since the space required to mount the mechanism for calculating the feature values can be omitted.

Considering the above, the MDM proposed by Koike et al. consists of three mechanisms: an access hit counter, dividers, and a classifier, and converts the data into the target feature values through processing by each of the mechanisms. Furthermore, Figure 1 shows the configuration of the MDM.

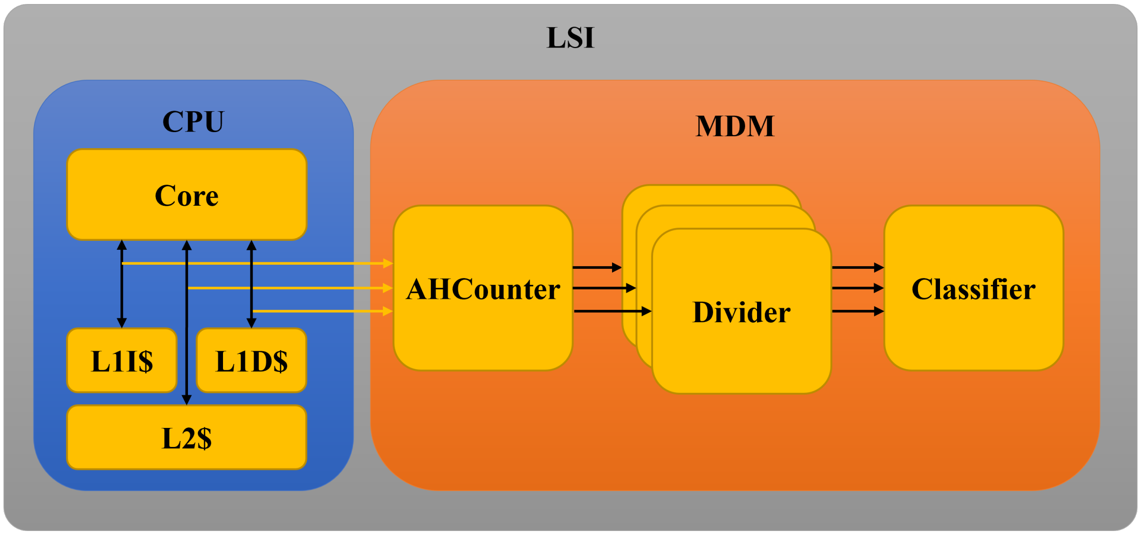


Figure 1: Proposed method by Koike et al.

Next, we describe the details of each mechanism of the MDM.

3.1.1 Access-Hit Counter

The access-hit counter bypasses the access signal sent by the CPU when it requests data from the cache and bypasses the hit signal sent by the cache when there exists data corresponding to the memory address requested by the CPU. The mechanism adds 1 to the number of accesses when the access signal is asserted and adds 1 to the number of hits when the hit signal is asserted. This operation is repeated for each instruction. There are three caches: the L1 instruction cache and L1 data cache, which can respond faster, and the L2 cache, which has a similar response speed and records the number of accesses and hits. The data counted here are passed to the dividers, which are the next process.

3.1.2 Divider

The divider calculates the cache hit rate from the number of accesses and hits sent from the access hit counter. The hit rate is the number of hits divided by the number of accesses, and shows the percentage of target data in the cache. Furthermore, since there are three caches, this mechanism is also equipped with three dividers to output the hit rate for the three caches at the same time. The calculated hit rates are passed to the classifier for the next processing.

3.1.3 Classifier

The classifier uses the hit rate obtained from the dividers as feature values for classification. This mechanism utilizes an internal random forest mechanism to perform the final program classification. The random forest mechanism is created using the random forest algorithm, which is a machine-learning algorithm, and its training model is relocated on the hardware. Compared to several other algorithms, random-forest-based algorithms have been shown to achieve high accuracy [25, 12]. The classifier outputs classification results by performing two internal processes. First, it passes the feature values passed for each instruction through the random forest, it classifies the instruction as either benign or malicious, and calculates the predicted malicious instruction (PMI) rate. Second, a threshold is set for the PMI rate calculated in the first process and if the rate exceeds the threshold in any instruction interval, then the program running on the CPU at that time is classified as malware. The MDM uses these three mechanisms to constantly monitor the CPU, classify instructions, and detect malware.

3.2 Overview of the HRTable

In response to Koike et al.'s proposed MDM, Deguchi et al. proposed a new mechanism called the HRTable and are considering its implementation in the MDM. In this section, we describe the issues with the divider, which is a component of Koike et al.'s MDM, and the HRTable proposed as an alternative [9].

The construction of a mechanism to perform division in hardware, including the MDM, is an issue that needs to be addressed in terms of circuit scale and processing speed. The scale of circuitry required for division dramatically increases as the bit width of the divisor and dividend increases. Furthermore, multiple clocks are required to calculate a single result. This is a bottleneck process when calculating and classifying the hit rate for each instruction in MDM.

To solve this problem of the divider, Deguchi et al. proposed a mechanism for calculating hit rates quickly by using a table-type mechanism that retains hit rate calculation results in advance (Table 1). This mechanism is called the HRTable, and instead of the dividers, the HRTable takes as entries the pairs of access and hit counts obtained from the access-hit counter and passes the hit rates recorded in the corresponding locations to the classifier. Compared to the divider, the HRTable does not require arithmetic processing, thus enabling faster response times and reduced power consumption. Furthermore, the scale of the circuit of the HRTable depends on the bit width of the number of accesses and hits, which are the entry points, and the bit width of each stored hit rate. To reduce the circuit scale of the HRTable, Deguchi et al. reduced the bit widths of the entry values and feature values. Furthermore, the classification using the hit rates obtained after the reduction confirmed that the PMI rates in the executed programs were clearly separated between normal programs and malware and that the final classification accuracy was maintained compared to that before the reduction. To reduce the hardware resources of HRTable, entries that have a hit rate of 100% (e.g., when A=127, H=127) are retained as 0%. This is to reduce the resources required to hold the most significant bit due to the digit increase that occurs when the hit rate goes from 99% to 100%. Furthermore, since the cache rarely reaches 100% during operation, the impact during machine learning is negligible.

Figure 2 shows the HRTable described in this section and implemented in the MDM. Furthermore, in order to expand the range of devices that can be equipped with the MDM, the required feature values were changed from an L2 cache hit rate to a program counter (PC). This proposal enables us to consider implementing the MDM in smaller devices that are not equipped with an L2 cache [10].

Table 1: Configuration of the HRTable.

		0x0				0x7f	0x80				0xff
	A H	**0**	**1**	**2**	**...**	**127**	**128**	**...**	**253**	**254**	**255**
0x0	**0**	**0**	**0**	**0**	**...**	**0**	**0**	**...**	**0**	**0**	**0**
	1	**-**	**0**	**1/2**	**...**	**1/127**	**1/128**	**...**	**1/253**	**1/254**	**1/255**
	2	**-**		**0**	**...**	**2/127**	**2/128**	**...**	**2/253**	**2/254**	**2/255**
	...	**-**	**-**	**-**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
	126	**-**	**-**	**-**	**-**	**126/127**	**126/128**	**...**	**126/253**	**126/254**	**126/255**
0x7f	**127**	**-**	**-**	**-**	**-**	**0**	**127/128**	**...**	**127/253**	**127/254**	**127/255**
0x80	**128**	**-**	**-**	**-**	**-**	**-**	**0**	**...**	**128/253**	**128/254**	**128/255**
	...	**-**	**-**	**-**	**-**	**-**	**-**	**...**	**...**	**...**	**...**
	253	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**0**	**253/254**	**253/255**
	254	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**0**	**254/255**
0xff	**255**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**0**

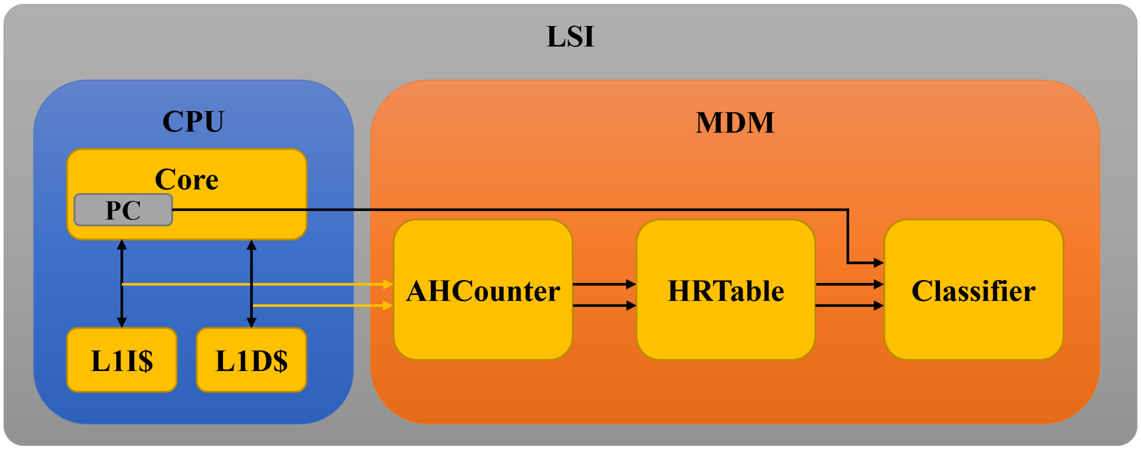


Figure 2: MDM configuration with HRTable.

3.3 Issues of the MDM and HRTable

The implementation of the HRTable proposed by Deguchi et al. in the MDM proposed by Koike et al. has resulted in faster feature values calculation and reduced circuit scale. However, there are some issues that need to be addressed in the proposed mechanism. The first is the problem of false positives in the classifier, and the second is the problem of power consumption due to the structure of the HRTable. In the following sections, we will discuss the details of these two issues in order to pursue and solve them.

3.3.1 Misdetection by the classifier

The MDM describes the use of a random forest contained within a classifier to identify the behavior of the instructions being executed. Figure 3 shows the PMI rates measured for each program in the random forest created based on the method proposed by Deguchi et al. In Figure 3, the five programs on the left are malware; the rest are normal programs. The learning model used in the measurements is based on RISC-V, is built on field-programmable gate arrays (FPGA), and uses processor information obtained when running a small Linux OS embedded in the FPGA, such as PC, L1 instruction cache hit rate, L1 data instruction cache hit rate. RISC-V is a RISC-based instruction set architecture (ISA) developed at the University of California, Berkeley. It allows users to add their own custom instructions. Furthermore, since it is an open license specification, various examples of its use in organizations and companies have been published in recent years.

The results of the learning model created based on data obtained from the environment as described above indicated cases where the detection rate of malicious instructions was low for malware (low PMI rate) and cases where the percentage of malicious instructions predicted for normal programs was high (high PMI rate), making it difficult to set appropriate thresholds. These cases may cause false positives (FP) for normal programs (For example, if the PMI rate threshold is set at 30%, the jq command will be false detected.) and false negatives (FN) for undetected malware, leading to unexpected anomalies during normal system operation and malware activities that operate behind the scenes both being overlooked. Therefore, countermeasures are needed to address the issue of classification accuracy.

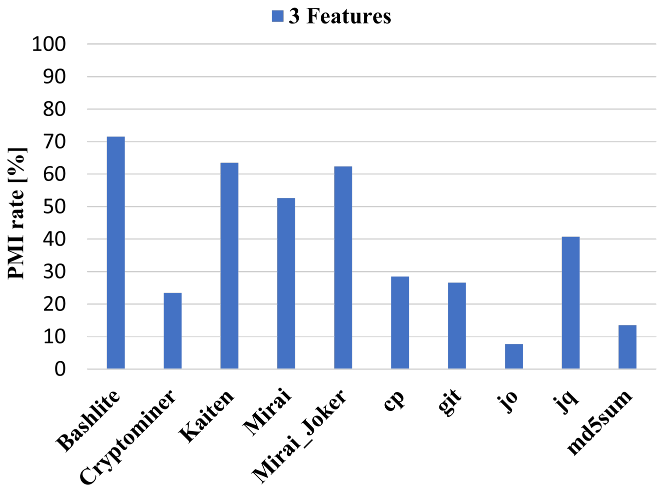


Figure 3: PMI rate when 3 feature values used.

3.3.2 Circuit scale and power consumption of the HRTable

The MDM in this research is considered to be implemented as a security countermeasure mechanism for small devices classified as IoT devices and, therefore, each mechanism in the MDM should be resource- and power-efficient. In the proposal by Deguchi et al., the entry points to the HRTable are derived according to the data flow shown in Figure 4. For example, when the access and hit signals sent from the CPU are received (a_sig == 1, h_sig == 1), the number of accesses (a_cnt) and hits (h_cnt) are counted up. Then, if the number of accesses takes a value greater than the specified value ($2^8 \leq$ a_cnt), the number of accesses and the number of hits are reduced to the range of entry points by shifting them by 1 bit. Thereafter, the count will also be based on the number of accesses and hits after the shift. However, rounding errors are not taken into account in the calculation of the entry points and the hit rate obtained from these incorrect entry points is also inaccurate. Therefore, we present an alternative to the above method. Furthermore, the following section describes the possibility of reducing the circuit scale of the HRTable, power consumption issues related to the reference frequency, and countermeasures for the proposed method.

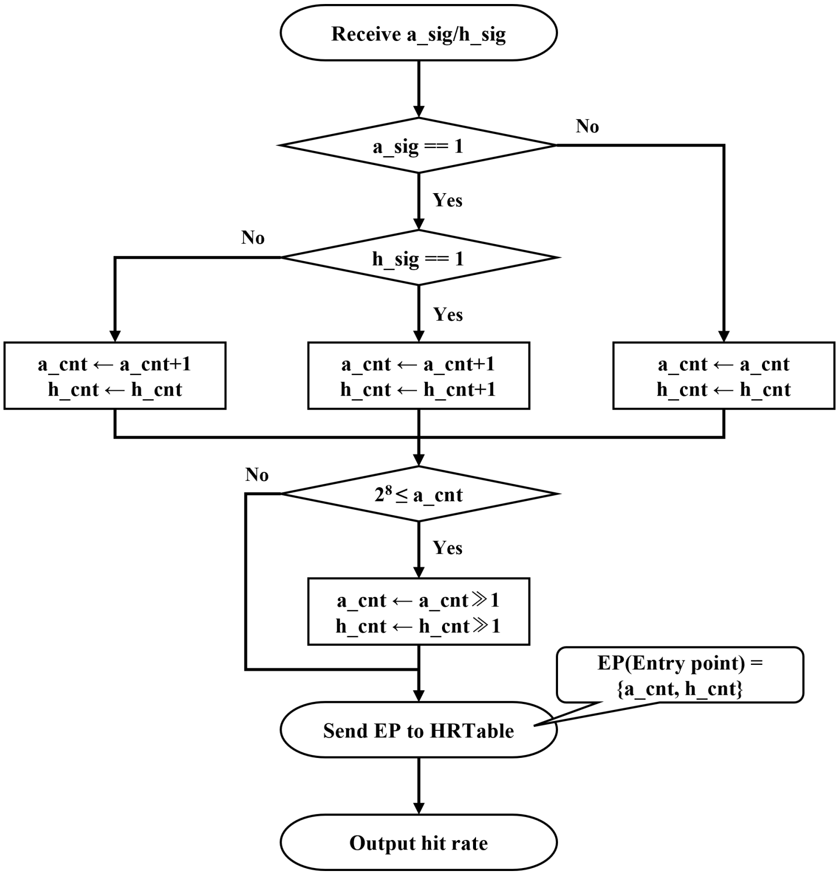


Figure 4: Flowchart for calculation of entry points with errors.

4 PROPOSED METHOD

In Section 3, we discussed issues of the conventional MDM, such as the risk of false positives in the classifier and the possibilities of reducing the circuit scale and power consumption of the HRTable. In this section, we describe efforts to improve the classification accuracy of the classifier and to reduce the circuit scale and power consumption in the HRTable.

4.1 Use of hit rates segmented by CPU mode

The random forest mechanism of conventional classifiers is based on three feature values: PC, L1 instruction cache hit rate, and L1 data cache hit rate. When a cache holds data at an address, it also holds data at surrounding addresses and manages them in arbitrary blocks. Furthermore, CPU operation modes are categorized into user area (user mode), and kernel area (privileged mode). Since the addresses holding instructions to be executed in each area are roughly classified, access to separate blocks is considered when the CPU accesses the instruction cache. A program with a certain level of functionality goes back and forth between user mode and privileged mode, so it is expected that more detailed training data can be obtained by measuring the transition of cache hit rates from these two perspectives and adding them as feature values. Furthermore, in the data cache, we measure the data cache hit rate for each mode of operation by categorizing the cases in which the instructions executed in each mode make access requests.

4.2 Reduction of circuit scale and power consumption of the HRTable

We explained in Section 3 that the entry points of the HRTable are generated by the access-hit counter. In this section, we propose a conversion circuit that approximates the entry points calculated by the access hit counter more accurately. We show that the proposed method reduces the circuit scale and power consumption of the HRTable.

4.2.1 Calculation of entry points by the access-hit counter

As shown in Figure 4, the mechanism of halving the counter value by shifting is intended to prevent deviations from the steady state of the hit rate caused by zero resetting during overflow. However, this mechanism is mathematically imprecise, and Figure 5 is proposed to address this issue. Figure 5 shows the newly proposed conversion method for calculating entry points, which more accurately represent the values of accesses and hits. The terms “a_cnt_shr” and “h_cnt_shr” in the figure shift the number of accesses and hits held in “a_cnt” and “h_cnt”, respectively (2^7(0x80) $\leq$ a_cnt_shr, h_cnt_shr $\leq$ 2^8-1(0xff)). The number of accesses and hits are always increasing, therefore, the variation of the hit rates with the change in the lower bits over time becomes minute. Therefore, by leaving the upper bits, which are largely responsible for variations in hit rates, the calculated hit rate can be approximated to a more accurate value when compared to the data flow shown in Figure 4.

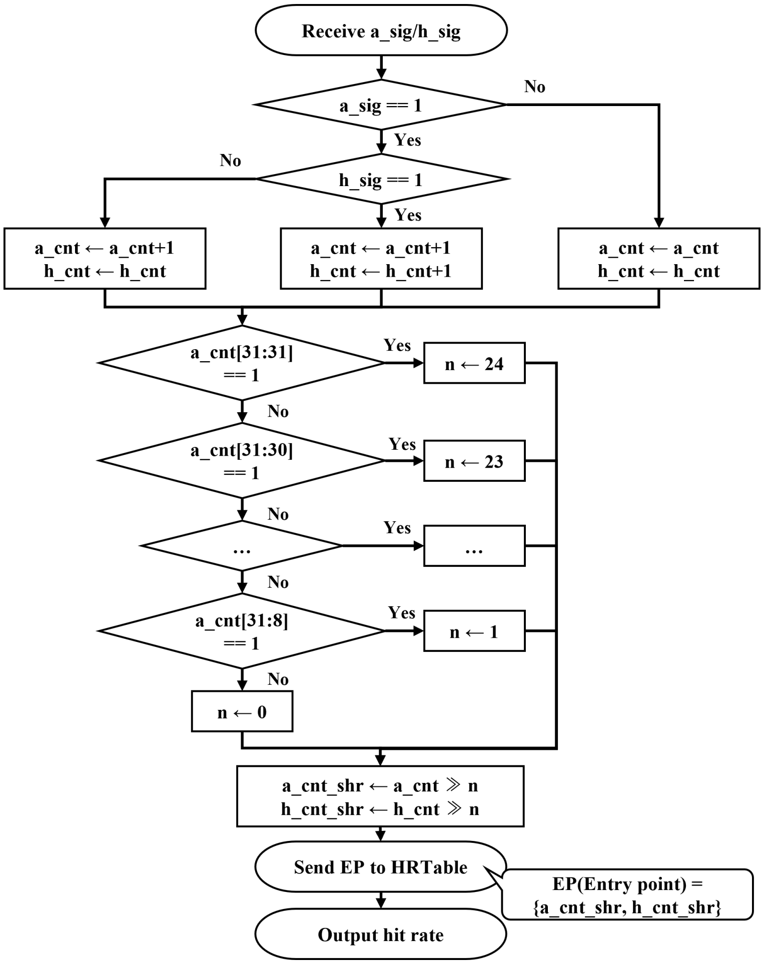


Figure 5: Flowchart to calculate entry points.

4.2.2 Reduction of circuit scale for the HRTable

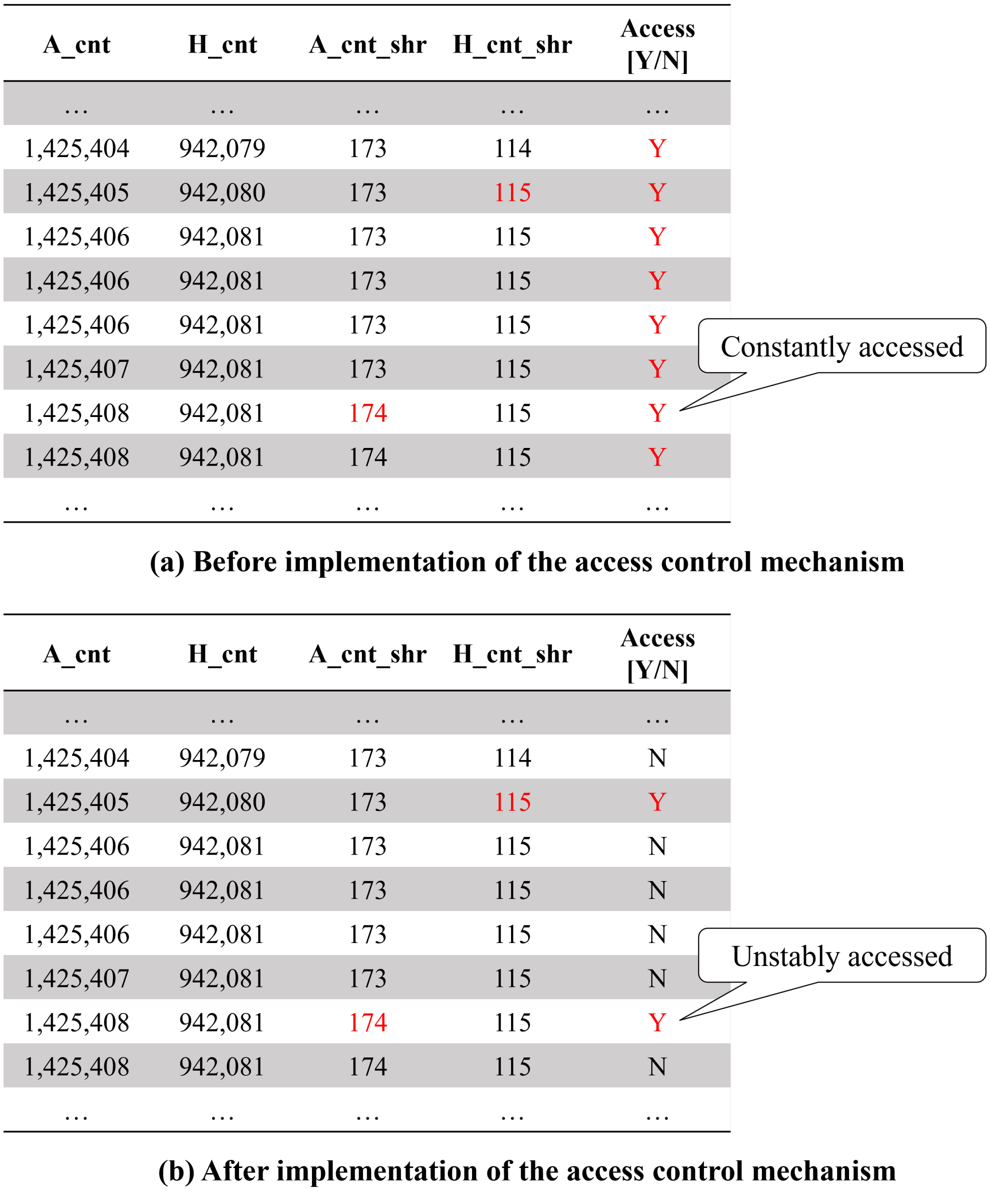
In the new method of calculating the entry points using the access-hit counter, there are two types of registers: two 32-bit registers, a_cnt, h_cnt, that hold the number of accesses and hits before the shift, respectively, and two 8-bit registers, a_cnt_shr and h_cnt_shr, that hold the number of accesses and hits reduced to 8 bits in width after the shift, respectively, using four registers to obtain a single hit rate. The number of accesses and hits held in the 32-bit registers is always increasing in the ranges $0 \leq a\_cnt\ and\ h\_cnt \leq 2^{32} - 1$ (0xffffffff). Furthermore, in the 8-bit registers that hold the value after the shift, the number of accesses and hits that hold the value after the shift increases in the ranges $0 \leq a\_cnt\ and\ h\_cnt \leq 2^8 - 1$ (0xff) during the 255 increments after the count starts. We focus on the 8-bit register (a_cnt_shr) that holds the number of accesses after the shift. After 256 increments, a_cnt_shr always shows 1 in the MSB, and its value range converges to 2^7 (0x80) $\leq$ a_cnt $\leq 2^8 - 1$ (0xff). Furthermore, Table 2 shows the shaded area in the HRTable where circuit reduction can be expected. When a_cnt is greater than 256, a_cnt_shr cannot be less than 2^7, therefore, HRTable references are concentrated in the right half of the figure (0x80 $\leq a\_cnt \leq$ 0xff and $0 \leq h\_cnt \leq$ 0xff). Since 32-bit registers can hold up to 2^{32} increments, the number of references to the left half of the HRTable ($0 \leq a\_cnt\_shr \leq$ 0x7f and $0 \leq h\_cnt\_shr \leq$ 0xff) is negligible for increases up to 256 times. Therefore, the left half of the HRTable can be removed to reduce the circuit scale by approximately 1/2th.

Table 2: Reduction of circuit scale for the HRTable considering frequency of reference.

		0x0				0x7f	0x80				0xff
	H A	0	1	2	…	127	128	…	253	254	255
0x0	0	▨	▨	▨	▨	▨	0	…	0	0	0
	1	▨	▨	▨	▨	▨	1/128	…	1/253	1/254	1/255
	2	▨	▨	▨	▨	▨	2/128	…	2/253	2/254	2/255
	…	▨	▨	▨	▨	▨	…	…	…	…	…
	126	▨	▨	▨	▨	▨	126/128	…	126/253	126/254	126/255
0x7f	127	▨	▨	▨	▨	▨	127/128	…	127/253	127/254	127/255
0x80	128	▨	▨	▨	▨	▨	0	…	128/253	128/254	128/255
	…	▨	▨	▨	▨	▨	-	…	…	…	…
	253	▨	▨	▨	▨	▨	-	-	0	253/254	253/255
	254	▨	▨	▨	▨	▨	-	-	-	0	254/255
0xff	255	▨	▨	▨	▨	▨	-	-	-	-	0

4.2.3 Access control and reduction of power consumption of HRTable

In the MDM, the HRTable is used to calculate the cache hit rates, which are feature values. In the case of adding additional feature values to improve classification accuracy as proposed in this paper, it is necessary to calculate six feature values from a single instruction: total instruction cache hit rate, total data cache hit rate, kernel instruction cache hit rate, kernel data cache hit rate, user instruction cache hit rate, and user data cache hit rate. Furthermore, when measuring the cache hit rates, the variation of the hit rate generally decreases as the number of cache hits increases over time. Figure 6 shows the reference requests to HRTable at a certain time when enough time has elapsed. The number of accesses and hits are sufficiently large that no change appears in the value after the shift. If the entry value is the same as the previous reference, then the output value, the hit rate, will also be the same(Figure 6(a)). Adding an access control mechanism to the HRTable that takes this property into account in the circuit reduces the conventional power consumption(Figure 6(b)).



A_cnt	H_cnt	A_cnt_shr	H_cnt_shr	Access [Y/N]
…	…	…	…	…
1,425,404	942,079	173	114	Y
1,425,405	942,080	173	115	Y
1,425,406	942,081	173	115	Y
1,425,406	942,081	173	115	Y
1,425,406	942,081	173	115	Y
1,425,407	942,081	173	115	Y
1,425,408	942,081	174	115	Y
1,425,408	942,081	174	115	Y
…	…	…	…	…

Constantly accessed

(a) Before implementation of the access control mechanism

A_cnt	H_cnt	A_cnt_shr	H_cnt_shr	Access [Y/N]
…	…	…	…	…
1,425,404	942,079	173	114	N
1,425,405	942,080	173	115	Y
1,425,406	942,081	173	115	N
1,425,406	942,081	173	115	N
1,425,406	942,081	173	115	N
1,425,407	942,081	173	115	N
1,425,408	942,081	174	115	Y
1,425,408	942,081	174	115	N
…	…	…	…	…

Unstably accessed

(b) After implementation of the access control mechanism

Figure 6: Improved frequency of access to the HRTable.

5 EVALUATION METHOD AND MEASUREMENT RESULT

In Section 4, we described our efforts to improve the proposed MDM. The first is a proposal to add feature values to improve the classification accuracy of the classifier. The second is a proposal to improve and reduce the circuit scale in the process from the calculation of entry points using the access hit counter to the calculation of hit rates using the HRTable, and to reduce the power consumption of the process. As only the HRTable is measured, the incremental circuitry and power consumption of the other mechanisms described in this proposal are not considered.

5.1 Evaluation of the effects of adding features on classification accuracy

5.1.1 Collecting traces

To perform learning and testing, Direct Memory Access (DMA) is built on Zedboard to collect processor information obtained from RISC-V (Figure 7). DMA can read and write the collected data directly to the memory on Zedboard. At this time, a specific range of memory addresses is specified for DMA. Furthermore, since the DMA process operates independently from the processor (RISC-V), the processor information can be collected at high speed without affecting the data to be debugged or the processor. The collected processor information is used for machine learning in the Evaluation PC, which generates a random forest model after training and converts it to Verilog code for placement on the FPGA. This model is then built in the MDM's classifier. Next, we explain how the processor information is processed for training.

The collected processor information is processed into a trace that holds the feature values shown in Table 3 for each instruction. First, the PC directly uses the value of the program counter obtained from the RISC-V CPU. The hit rate is divided into six types: total instruction/data cache hit rate, kernel instruction/data cache hit rate, and user instruction/data cache hit rate. The total number of accesses and the total number of hits are obtained from the access and hit signals of the processor information, respectively, and the hit rate is calculated for each elapsed instruction. This trace is executed six times for each program and distinguished as "program_number" (e.g., apache_1, apache_2, ..., apache_6). The number indicates the number of times the program is executed. Each time a program is executed, the environment is initialized so that it is not affected by the previous execution, and then the program is executed again.

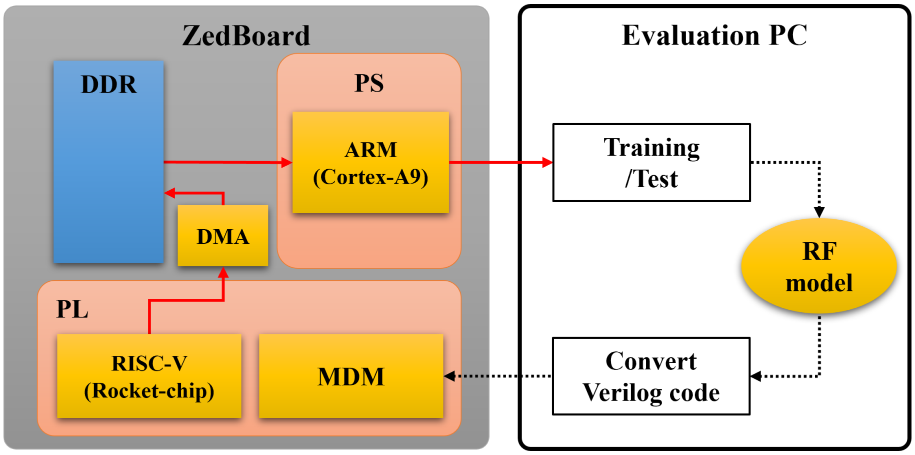


Figure 7: Processor information collection using DMA.

5.1.2 Categorizing traces

The normal programs and malware for which traces were collected are shown in Table 4 and Table 5, respectively. For the normal programs, we divide the programs into OS command and NOS command categories. OS commands are programs that are included by default even in small Linux configurations and are frequently used. NOS commands are programs that are not considered to be used by default like OS commands, and must be installed by the users themselves. Regarding the variation of input values, especially for normal programs, it is basic that the values of arguments are variable, therefore it is necessary to input various arguments when collecting traces. In this

Table 3: Feature values to be used.

Features	Description
PC	Program Counter Value
Total ICache Hit Rate	Calculated from cumulative instruction cache accesses and hits.
Total DCache Hit Rate	Calculated from cumulative data cache accesses and hits.
Kernel ICache Hit Rate	Instruction cache hit rate when PC shows kernel area.
Kernel DCache Hit Rate	Calculated from the number of hits and accesses to DCache during kernel mode execution
User ICache Hit Rate	Instruction cache hit rate when PC shows user area
User DCache Hit Rate	Calculated from the number of accesses and hits to DCache during user mode execution

study, however, the input values are fixed for both normal programs and malware. This is because malware basically has fixed input values at runtime, so the conditions need to be the same, and we considered it necessary to first verify the classification function at this stage. The variety of input values for normal programs is a future issue.

Next, the obtained traces are classified into training and test datasets. Each dataset is grouped by trace numbering. For example, {Dataset_1: apache_1, cp_1, Mirai_1, ...}, {Dataset_2: apache_2, cp_2, Mirai_2, ...}, ..., {Dataset_6: apache_6, ...}. Next, we train on each dataset. The training models are trained on (a)DataSet_1, (b)DataSet_2, ..., (f)DataSet_6. For each training model, we use the untrained dataset as a test dataset and calculate the average PMI rate.

Table 4: Normal programs used.

	Normal Program (version)
OS command	cat, cp, gzip, ls, md5sum, seq, sort, strings (BusyBox ver. 1.29.2)
NOS command	apache(2.4.34), cvs(1.12.13), git(2.16.24), jo(1.1), jq(1.5)

Table 5: Malware used and its hash value.

Malware	MD5 hash
Bashlite	ec287752d8ee41d74a27d83ebf781f0b
Cryptominer	e0364e3ce5072988d0459dba836f7ec9
Kaiten	7a473fb7dc8055c576c25d0434731301
Mirai	35a30703f7dac234ddebc6e80a732476
Mirai_Joker	9717e012f372d2dca3dd69c1f71de0b9

5.1.3 Selection of learning parameters

A random forest model is created by training on each dataset. The parameters used to create the random forest model are reduced to the range that is feasible for implementation on an FPGA. First, we set the bit width of the feature values to be trained to 16 bits. Table 6 shows the binary size of each training model when training with different parameters (trained on Dataset_1). Here, “bw” is the bit width, “nt” is the number of decision trees in the random forest, and “md” is the maximum depth of the decision tree. The model shown in italics is the excess hardware elements retained by the FPGA during hardware implementation. Furthermore, the models underlined are those for which timing constraints were violated during implementation, owing to the depth of the decision tree. Therefore, among the learning models that can be implemented, we evaluate a random forest model with bw:16, nt:10, md:10 (bold body), which has a well-balanced parameter distribution.

5.1.4 Evaluation method 1

The classification accuracies before and after the addition of feature values are compared. The number of features before the addition of the features (PC, Total ICache Hit Rate, and Total DCache Hit Rate), which caused problems with classification accuracy, was 3, while the number of

Table 6: Sizes of the learning models (RF: random forest).

RF model	Size[Byte]	RF model	Size[Byte]
bw16_nt005_md05	21K	bw16_nt050_md05	190K
bw16_nt005_md10	168K	*bw16_nt050_md10*	1.7M
bw16_nt005_md15	746K	*bw16_nt050_md15*	7.0M
bw16_nt005_md20	1.5M	*bw16_nt050_md20*	17M
bw16_nt010_md05	41K	bw16_nt100_md05	385K
bw16_nt010_md10	**326K**	*bw16_nt100_md10*	3.3M
bw16_nt010_md15	1.4M	*bw16_nt100_md15*	14M
bw16_nt010_md20	3.4M	*bw16_nt100_md20*	32M

features after the addition of the features was 7 (all of Table 3). By comparing these results, we can confirm whether segmentation of limited processor information is effective in improving accuracy.

5.1.5 Evaluation result 1

Figure 8 shows a comparison of PMI rates before and after implementation of the proposed method to improve the classification accuracy of programs. For the normal programs, we show the top five programs whose PMI rates were calculated to be relatively high among the normal programs before the proposed method was implemented(For the other normal programs, the PMI rate ranged between 0.00-2.17%). In most of the measurements, the PMI rate of malware traces increased after the addition of feature values. However, the PMI rate of "Cryptominer" tends to be lower than that of other malware. Next, for normal programs, the PMI rates of "cp", "jq", and "md5sum", which were calculated to be high before the proposed method was implemented and could cause false positives, were suppressed after the proposed method was implemented.

5.1.6 Evaluation method 2

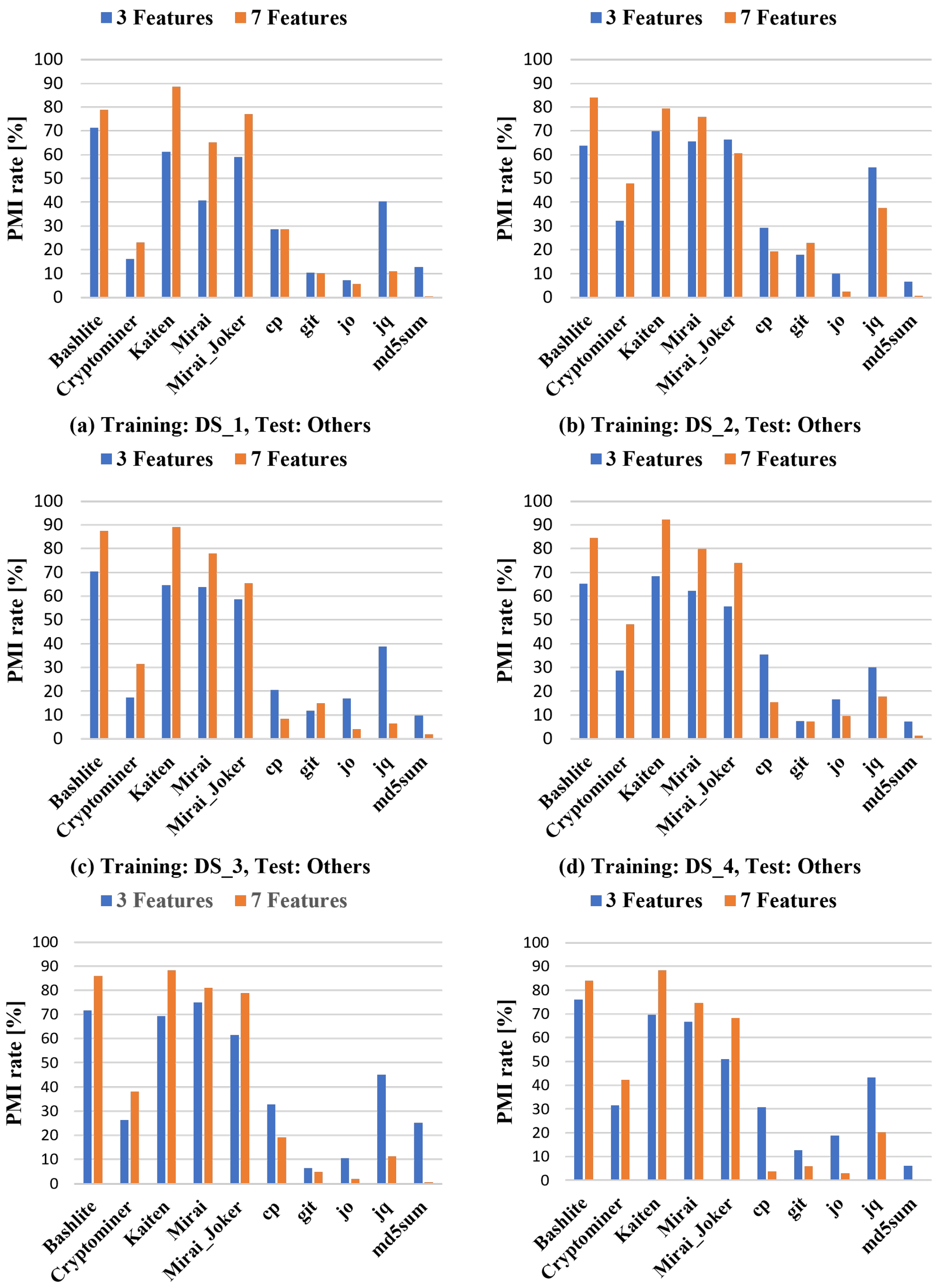
Next, a random forest model using non-PC feature values from the 7 feature values is generated to measure the PMI rate. This is a measure to reduce the number of accesses to the classifier when implementing MDM. Currently, MDM calculates feature values for each instruction and classifies whether the instruction is benign or not. However, it is obviously the case that the classification result for the same set of feature values will be the same as for the previous one unless the implemented random forest model is changed. Therefore, access to the classifier can be stopped until a change appears in the feature values. However, since the PC among the seven feature values changes with each instruction, the set of feature values is constantly changing. Therefore, a random forest model is generated from the 6 feature values without the PC and compared with the 7 feature values.

5.1.7 Evaluation result 2

Figure 9 compares the results of the random forest models using 7 and 6 feature values. Overall, there is little negative impact from unplugging the PCs. Furthermore, for some malware, the PMI rate has improved. The lowest PMI rate for the malware with 7 feature values was 23.15% and that for the malware with 6 feature values was 25.89%. Furthermore, the maximum PMI rate of the normal program was 37.62% for the 7 feature values and 23.32% for the 6 feature values, indicating that it was suppressed.

5.2 Circuit scale and power consumption of HRTable

In this section, we investigate the circuit scale and its effect on power consumption in the HRTable discussed in Section 4.2. The measurement environment is Vivado, an integrated development tool provided by Xilinx. Furthermore, the target FPGA is Xilinx's XC7Z020CLG484-1, and the circuit



(a) Training: DS_1, Test: Others

(b) Training: DS_2, Test: Others

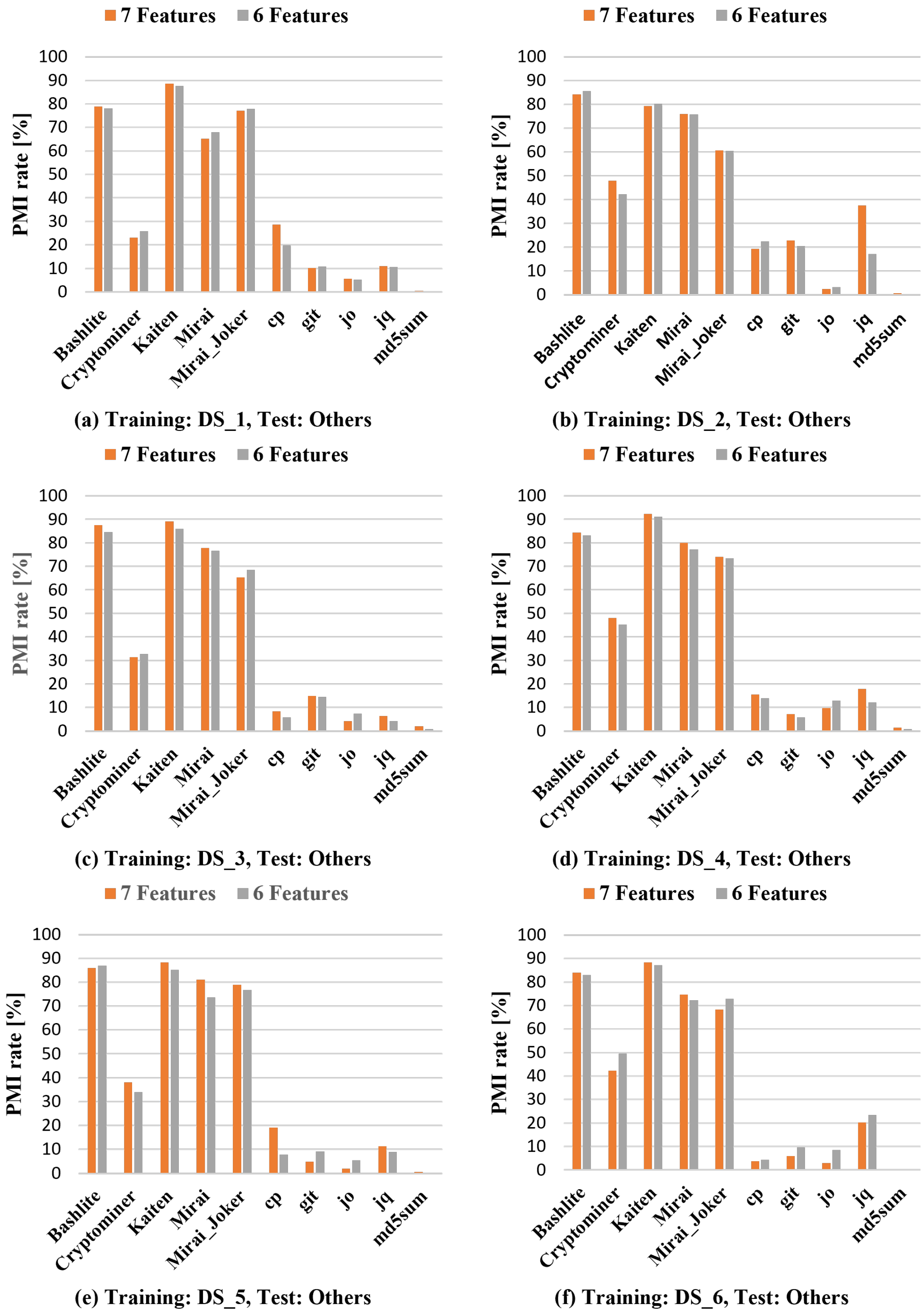
(c) Training: DS_3, Test: Others

(d) Training: DS_4, Test: Others

(e) Training: DS_5, Test: Others

(f) Training: DS_6, Test: Others

Figure 8: Percentage of instructions predicted malware. (3 features vs 7 features)



(a) Training: DS_1, Test: Others

(b) Training: DS_2, Test: Others

(c) Training: DS_3, Test: Others

(d) Training: DS_4, Test: Others

(e) Training: DS_5, Test: Others

(f) Training: DS_6, Test: Others

Figure 9: Percentage of instructions predicted malware. (7 features vs 6 features)

design, logic synthesis, and implementation are performed in Vivado. The measured values are calculated from the optimized circuit design at the implementation stage.

5.2.1 Evaluation of circuit scale

Section 4.2.1 and Section 4.2.2 describe a method for reducing the circuit scale of the HRTable. Table 7 shows the circuit scale of the HRTable before and after the implementation of this method. Therefore, it was confirmed that the HRTable used about twice as many FPGA resources compared to the after implementation of the control logic.

Table 7: HRTable main hardware usage.

Category	Resource usage (LUT)	Resource usage (Block RAM)
Before	59 (0.11%)	30 (21.4%)
After	28 (0.05%)	15 (10.7%)

5.2.2 Evaluation of power consumption

In Section 4.2.3, we described that power consumption reduction can be expected by implementing access control to the HRTable. In Table 8, we show the results of power consumption measurements using traces obtained during program execution on the FPGA. The measurement results were obtained by sending processor information to MDM every cycle in a simulation environment and processing 500,000 instructions. The Before column in Table 8 shows the state before the proposed method, in which the HRTable is frequently accessed because the access control circuit has not been implemented. The after figure shows that the proposed method reduces power consumption by more than 90% compared to the before state.

Table 8: Power consumption of HRTable.

Category	Power consumption (mw)
Before	5.06
After	0.23

5.3 Power consumption of the classifier

The number of feature values was reduced from seven to six by removing PCs from the feature values used for discrimination (Section 5.1.6 and Section 5.1.7). Furthermore, it was mentioned earlier that the PCs usually change with each instruction, so the feature values constantly change. Here, we measure the reduction in power consumption by reducing the number of classifications by the classifier, and by suppressing the variation in feature values by excluding the PC. In the measurement, processor information is sent to MDM every cycle in the simulation environment, and an estimate is made up to the point at which 500,000 instructions have been classified.

Table 9 shows the measured power consumption of the classifier for 7 and 6 feature values. The power consumption of the classifiers is reduced by excluding the PC. Furthermore, we showed in Section 5.1.7 that excluding the PC in the classifier is effective in improving the PMI rate.

Table 9: Power consumption of the classifier.

Category	Power consumption (mw)
7 Features	1.82
6 Features	1.36

6 DISCUSSION

This section discusses the evaluation results in Section 5.

6.1 Adding feature values to the classifier

When adding feature values, the difference in PMI rates between normal programs and malware became larger, allowing a clearer separation between the programs. To make a final judgment on the programs, we assume that if the threshold of 30% is exceeded, then the program is malware. Tables 10, 11, and 12 show the final judgment results for each learning model at the 30% threshold. By adding feature values, we were able to improve the malware detection rate (TP/TP+FN), which indicates the percentage of malware detected, and we were able to reduce the false positive rate (FP/TP+FP), which indicates the percentage of false positive detections of normal programs. Furthermore, by removing the PC from the feature values and reducing the number of features to six, the abnormal increase in the PMI rate of the normal program, which was caused by the PC, was suppressed, and the false positive rate was reduced to 0% for all training models.

Furthermore, although this measurement dealt only with the learning effect of a single dataset, it may be possible to test multiple datasets simultaneously to mutually compensate for the range of classification difficulties of each dataset.

Furthermore, MDM assumes an environment where a single program is executed. However, in actual operation, it will be a multi-programming environment where multiple programs run simultaneously, which is considered a future issue. When programs are executed simultaneously, they are executed one by one, switching at any time by context switches. At this time, a sufficient number of instructions are executed for classification, therefore, MDM is considered to be able to detect malware. However, since experiments considering a strict multi-programming environment have not been conducted in this study, it is considered that the conduct of research is an issue for the future.

Table 10: Classification accuracy on 3 feature values [threshold: 30%].

	(a)	(b)	(c)	(d)	(e)	(f)
TP	4	5	4	4	4	3
FP	1	1	1	2	2	2
FN	1	0	1	1	1	0
TN	12	12	12	11	11	11
Malware detection rate (%)	80%	100%	80%	80%	80%	100%
False positive rate (%)	20%	17%	20%	33%	33%	29%

Table 11: Classification accuracy on 7 feature values [threshold: 30%].

	(a)	(b)	(c)	(d)	(e)	(f)
TP	4	5	5	5	5	5
FP	0	1	0	0	0	0
FN	1	0	0	0	0	0
TN	13	12	13	13	13	13
Malware detection rate (%)	80%	100%	100%	100%	100%	100%
False positive rate (%)	0%	17%	0%	0%	0%	0%

Table 12: Classification accuracy on 6 feature values [threshold: 30%].

	(a)	(b)	(c)	(d)	(e)	(f)
TP	4	5	5	5	5	5
FP	0	0	0	0	0	0
FN	1	0	0	0	0	0
TN	13	13	13	13	13	13
Malware detection rate (%)	80%	100%	100%	100%	100%	100%
False positive rate (%)	0%	0%	0%	0%	0%	0%

6.2 Reduction circuit scale and power consumption of the HRTable

By creating an access control circuit in MDM, in Section 5.2, we were able to confirm that the circuit scale and power consumption of the HRTable could be reduced. In this measurement, we assumed that the CPU executes one instruction per cycle. However, since some instructions in real-world CPUs require multiple cycles to execute one instruction, more accurate power consumption estimation may be made possible by taking this into account. Furthermore, the implementation of the HRTable left some areas of the circuit not accessed to simplify the circuit architecture up to the calculation of the entry point of the access-hit counter. Therefore, it is expected that the size of the HRTable can be further reduced by improving the circuit architecture of the entry point calculation in the access-hit counter.

6.3 Hardware implementation

6.3.1 FPGA implementation of MDM

Table 13 shows the breakdown of power consumption for the MDM as a whole and for each mechanism. Case 1 shows the power consumption of the MDM before the implementation of the access control mechanism and the 6 feature values proposed in this paper. Case 2 shows the power consumption of the MDM when the access control mechanism is implemented. Case 3 shows the implementation of the access control mechanism and the classifier with feature values changed from 7 to 6 at the same time. The measurement results were obtained by sending processor information to the MDM every cycle in the simulation environment and processing 500,000 instructions. The Leaf Cells listed in the table serve as wires that connect each mechanism in the MDM. By implementing an access control method and the classifier with reduced feature values, the overall power consumption of the MDM can be reduced to approximately 65%. Furthermore, the power consumption of the Rocket-chip was 156.00 mw. These results are default measurements taken on Vivado, which is different from the measurement method of MDM. This is because it is difficult to develop CPU-specific test benches such as OS and applications on Vivado.

Furthermore, Figure 10 shows the results of FPGA implementation for the MDM and Rocket-chip in Case 3. The hardware usage of the Rocket-chip and peripheral devices accounts for most of the total hardware usage, while the MDM side is kept to a small amount, leaving a surplus in resource usage in Zedboard. Therefore, it is possible to add or improve the functionality of the MDM.

Table 13: Power consumption of MDM (mw).

Module	Case1	Case2	Case3
Leaf Cells	10.54	10.41	9.93
AHCounter	2.61	3.47	3.18
HRTable	5.06	0.23	0.23
Classifier	4.27	1.82	1.36
MDM	22.48	15.93	14.70

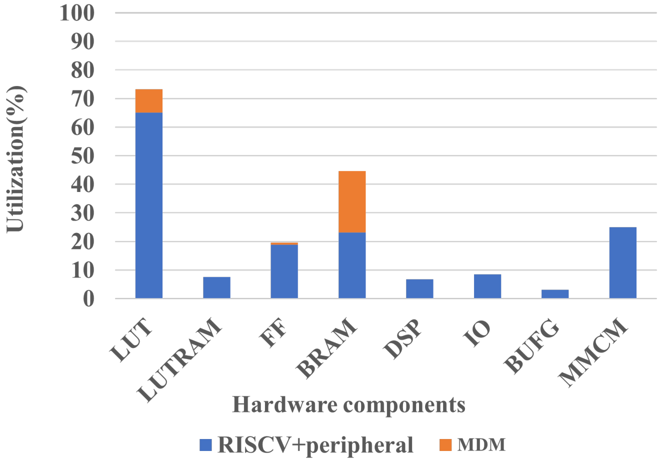


Figure 10: Hardware implementation results.

6.3.2 Survey of CPUs for application of MDM

Currently, CPUs embedded in IoT devices vary widely in performance (instruction set, number of cores, clock frequency, number of pipelines, cache/memory size, etc.) depending on the application of the device [3]. Finally, we survey CPUs installed in IoT devices currently used in various fields and examine whether they have the requirements to take advantage of the MDM. In this paper, we focus only on whether or not the CPU is equipped with a cache as a requirement. Table 14 shows an example of CPU configurations used in IoT devices. CPUs categorized as low-end are low-cost, compact, and often have a clock frequency of 100 MHz or lower, which supports a wide range of IoT devices. Mid-range products have clock frequencies under 1 GHz and are used in industrial and home-appliance applications. High-end products are those with clock frequencies above 1 GHz.

Among the three categories, most low-end products do not have a cache. This is due to the relatively low frequency of CPUs, which means that the benefit from the cache is limited. Therefore, to use the MDM, the performance of CPUs in the mid-range and high-end categories is necessary.

7 CONCLUSION

In this paper, we evaluated the malware detection mechanism, a mechanism for detecting malware based on the behavior information of small Internet of Things devices. We improved its classification

Table 14: Composition of CPUs in IoT devices.

	CPU Family	Frequency	Cache
Low-end	ARM Cortex-M	48 - 200 MHz	No
	RL78	32 MHz	No
	Intel Quark D	32 MHz	No
Mid-range	ARM Cortex-A7	1 GHz	Yes
	RX	240 MHz	Yes
	Intel Quark X	400 MHz	Yes
High-end	ARM Cortex-A15	1.9 GHz	Yes
	ARM Cortex-R	1.4 - 1.8 MHz	Yes

accuracy and reduced its cost by reducing its circuit scale and power consumption. In terms of classification accuracy, by setting an appropriate threshold based on the predicted malicious instruction rate, we were able to achieve a malware detection rate of 100% and a false positive rate of 0% for the final program classification.

Furthermore, we implemented an access control circuit by focusing on the access-hit counter and hit-rate table (HRTable). This enabled the reduction of cells with low reference frequency and reduced the circuit scale of the HRTable by approximately half. Furthermore, we confirmed that applying the access flag can reduce the power consumption caused by the HRTable by more than 90%. By reducing the number of feature values, the classifier reduces the frequency of data updates, leading to a reduction in power consumption.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 19K11968 and 20K11818.

References

[1] Arm ltd, technologies trustzone for cortex-m, 2021. `https://www.arm.com/en/technologies/trustzone-for-cortex-m`.

[2] Intel corp, intel security features and technologies related to transient execution attacks, 2021. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/related-intel-security-features-technologies.html`.

[3] T. Adegbija, A. Rogacs, C. Patel, and A. G. Ross. Microprocessor optimizations for the internet of things: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):7–20, 2018.

[4] 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, Paper 2017/564, 2017. `https://eprint.iacr.org/2017/564`.

[5] M. B. Bahador, M. Abadi, and A. Tajoddin. Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 703–708, 2014.

[6] K. Basu, R. Elnaggar, K. Chakrabarty, and R. Karri. Preempt: Preempting malware by examining embedded processor traces. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[7] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*, pages 113–120, 2013.

[8] M. Deguchi, M. Kato, and R. Kobayashi. Evaluation of low-cost operation of a malware detection mechanism using processor information targeting the iot. *The 10th International Workshop on Information and Communication Security (WICS 2022)*, pages 1–7, 11 2022.

[9] M. Deguchi, M. Katoh, and R. Kobayashi. Preliminary evaluation for fpga implementation of malware detection mechanism using processor information. In *Computer Security Symposium 2020 Proceedings*, pages 404–409, 2020.

[10] M. Deguchi, M. Katoh, and R. Kobayashi. Evaluation of implementability in a malware detection mechanism using processor information. In *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 313–319, 2021.

[11] A. Elhadi, M. Maarof, and A. H. Osman. Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9(3):283–288, 2012.

[12] M. Elnawawy, A. Sagahyroon, and T. Shanableh. Fpga-based network traffic classification using machine learning. *IEEE Access*, 8:175637–175650, 2020.

[13] J. C. Foreman. A survey of cyber security countermeasures using hardware performance counters, 2018. `arXiv:1807.10868`.

[14] V. Jyothi, X. Wang, S. K. Addepalli, and R. Karri. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID 2016)*, pages 587–588, 2016.

[15] K. Koike, R. Kobayashi, and M. Katoh. Iot-oriented high-efficient anti-malware hardware focusing on time series metadata extractable from inside a processor core. *International Journal of Information Security*, 21(4):757–775, 2022.

[16] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[17] G. Kornaros. Hardware-assisted machine learning in resource-constrained iot environments for security: Review and future prospective. *IEEE Access*, 10:58603–58622, 2022.

[18] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[19] Y. Liu, P. Shi, X. Wang, H. Chen, B. Zang, and H. Guan. Transparent and efficient cfi enforcement with intel processor trace. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 529–540, 2017.

[20] Q. D. Ngo, H. T. Nguyen, V. H. Le, and D. H. Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286, 2020.

[21] J. Nomani and J. Szefer. Predicting program phases and defending against side-channel attacks using hardware performance counters. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '15. Association for Computing Machinery, 2015.

[22] J. Pattee, S. M. Anik, and B. K. Lee. Performance monitoring counter based intelligent malware detection and design alternatives. *IEEE Access*, 10:28685–28692, 2022.

[23] M. Sewak, S. K. Sahay, and H. Rathore. Comparison of deep learning and the classical machine learning algorithm for the malware detection. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 293–296, 2018.

[24] G. Torres and C. Liu. Can data-only exploits be detected at runtime using hardware events? a case study of the heartbleed vulnerability. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, HASP 2016. Association for Computing Machinery, 2016.

[25] G. I. Trouli and G. Kornaros. Automotive virtual in-sensor analytics for securing vehicular communication. *IEEE Design & Test*, 37(3):91–98, 2020.

[26] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi. Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 457–468. Association for Computing Machinery, 2018.