International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 12, Number 2, pages 253–269, July 2022

Evaluation of implementability in 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 15, 2022
Accepted: April 4, 2022
Communicated by Toru Nakanishi

Abstract

Currently, software implementation is the mainstream approach for anti-malware measures. However, software-based anti-malware measures are difficult to implement in Internet of Things devices with limited hardware resources. To solve this problem, a malware detection mechanism that can be realized with only hardware has been proposed. The hardware mechanism consists of three elements: an access-hit counter, dividers, and a classifier. The classifier is generated by a random forest and uses processor information as feature values. To reduce the hardware scale, a Hit Rate Table (HRTable) is introduced in place of the dividers. We propose methods of reducing the scale of hardware resources and synchronizing the CPU and the malware detection mechanism. This paper implements the proposed mechanism in hardware, simulates it while considering the delay caused by input/output to the HRTable, and evaluates the hardware scale of the proposed mechanism combined with RISC-V on a field-programmable gate array (FPGA).

Keywords: Machine learning, Internet of Things, RISC-V, Malware detection, Hardware security

1 INTRODUCTION

The incidence of cyber-attacks targeting Internet of Things (IoT) devices has been on the rise in recent years and many attacks by malware such as Mirai have been identified to date [16, 14]. One of the reasons for the increase in attacks is the increasing popularity of IoT devices themselves. Another factor is that IoT devices on the market are still vulnerable to attack [9]. Therefore, anti-malware measures for IoT devices should be taken as soon as possible.

[0]The conference version of this paper is published in the proceedings of The Ninth International Symposium on Computing and Networking (CANDAR 2021).

One of the reasons for the lack of anti-malware measures in IoT devices is that many anti-malware measures are designed to be implemented in software, whereas IoT devices have fewer hardware resources than personal computers and server devices. To realize anti-malware measures that can be implemented even in IoT devices with scarce hardware resources, a malware detection mechanism (MDM) based on hardware implementation was proposed by Koike et al. [13]. The MDM consists of three hardware elements: an access-hit counter, three dividers, and a classifier. It is proposed to operate on Large-Scale Integration (LSI). A hardware element called the Hit Rate Table (hereinafter referred to as the HRTable) was introduced to the proposed MDM and evaluated [13, 12]. It is a mechanism that calculates the information obtained from the CPU (processor information) instead of the divider in the MDM. In the classifier of the MDM, a random forest structure that uses processor information as a feature is implemented in hardware. The classifier detects whether an instruction is a normal instruction or an abnormal instruction when it is executed when it is coupled with the CPU. Furthermore, if the number of abnormal instructions exceeds a certain number within the number of instructions executed in a given period, then this mechanism judges the program being executed to be malware.

Since the MDM uses processor information inside the CPU, it is necessary to synchronize the operations of the CPU and the MDM and to properly calculate the features required for instruction classification. In addition, varying the parameters of the number of decision trees in the random forest and the maximum depth of the decision trees in the classifier causes differences in hardware resource utilization. Resource utilization varies depending on the bit width of the feature value that is imputed to the classifier.

The following are the two contributions of this paper:

- A method for calculating the bit width of features, calculating the parameters of random forest decision trees, and reducing the scale of CPUs needed to implement the MDM, which can be implemented on small LSIs such as those used in IoT devices.

- In the synchronization of CPUs connected to MDMs, the HRTable is realized with consistent data input and output, taking into account the delay in implementation.

In the following, Section 2 describes related research and Section 3 gives an overview of the MDM and issues. Section 4 describes the proposed method, Section 5 describes the measurement results and evaluation, Section 6 discusses the results, and Section 7 concludes this paper.

2 RELATED WORKS

An example of a hardware security mechanism is the Trust-Zone security extension implemented in ARM, which virtually divides the ARM execution environment into a normal world and a secure world and handles important data in the secure world [10]. This mechanism is designed to protect data, and its purpose is different from that of the MDM.

In addition, Intel has announced a processor equipped with Control-Flow Enforcement Technology (CET), which is a chip-level security function [8]. CET can be used for attacks that exploit return and branch instructions, which are difficult to detect and prevent in software. CET is a function that detects the behavior of these attacks in hardware and protects the system by stopping the program execution.

Koike et al. [13] proposed an MDM that runs in parallel with the CPU on LSI and uses processor information to detect malware. Similar work has been done on malware detection using register aggregates called hardware performance counters (HPC), which classify applications as either malware or benign [15]. Deguchi et al. [12] proposed the HRTable as a countermeasure to the problem of hardware utilization of the divider among the hardware elements of the MDM proposed by Koike et al. The next section first provides an overview of the MDM and its issues and an overview of the HRTable before evaluating the hardware implementation of the MDM using the HRTable.

3 OVERVIEW OF MDM AND ITS ISSUES

This section gives an overview of the MDM proposed by Koike et al. [13] and the HRTable proposed by Deguchi et al. [12].

3.1 Overview of the MDM

First, the method proposed by Koike et al. [13] is shown in Fig. 1, where the processor information from the CPU is inputted to the access-hit counter, and statistics such as hit rates obtained from the L1 instruction cache (L1I$), L1 data cache (L1D$), and L2 cache (L2$) are calculated using a divider. The statistics are then inputted to the classifier to determine whether the program being executed is malware or not. The details of the MDM are described in Fig. 2. The first step is to calculate the total number of accesses and the total number of hits in each cache using the access-hit counter. The counter takes as input the access signals (L1I_access, L1D_access, and L2_access) generated when the CPU core accesses the cache and the hit signals (L1I_hit, L1D_hit, and L2_hit) generated when data corresponding to the address exist. The number of accesses (L1I_access_num, L1D_access_num, and L2_access_num) and the number of hits (L1I_hit_num, L1D_hit_num, and L2_hit_num) for each cache are outputted by adding the values whenever a signal is obtained from each cache. In the second process, a counter is created. In the second step, the cache hit rates (L1I_hit_rate, L1D_hit_rate, and L2_hit_rate) are calculated from the number of accesses and hits obtained by the counters using a divider. Three dividers are provided to calculate the hit rates for each cache in parallel. The third step is to classify the instructions using a classifier and to determine whether the program being executed is normal or malware. For the classification of programs, the hit rate obtained from the divider is used as the feature value and the random forest in the classifier is used to classify each instruction. If the number of instructions classified as malignant by the random forest exceeds a threshold value in an instruction section, then the classifier judges that the program executed in that instruction section is malware.

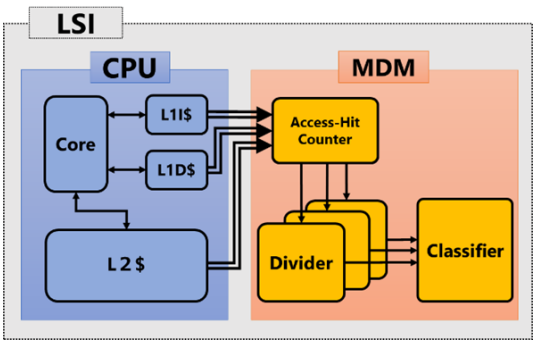

Figure 1: The method proposed by Koike et al.

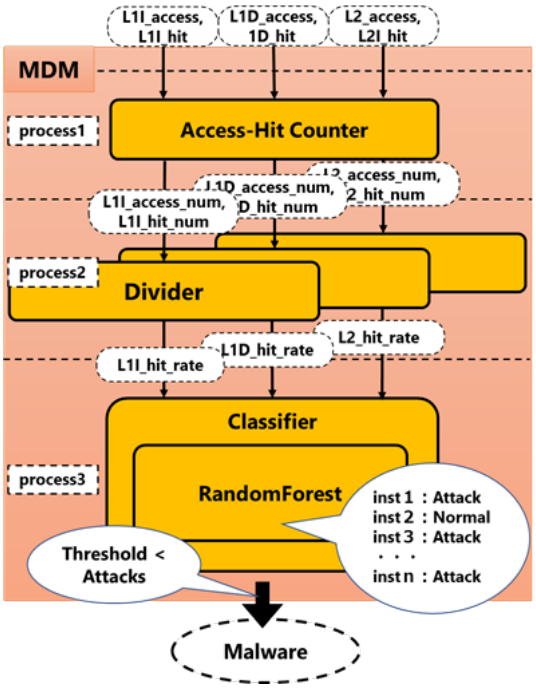

Figure 2: Flowchart for malware detection.

3.2 Overview of HRTable

Deguchi et al. [12] proposed a faster method for calculating hit rates with fewer hardware resources by introducing the HRTable as a replacement for the divider used by Koike et al. The HRTable holds the calculated division results in the form of a table, and when a binary value consisting of the number of accesses and the number of hits is inputted, it searches for the hit rate from the table and outputs it. The output hit rate is inputted to the classifier as a feature value. Deguchi et al. reduced the hardware resource utilization of the HRTable by reducing the address width corresponding to the input values and the bit width of each entry. Figure 3 shows the MDM configuration with the HRTable instead of dividers, focusing on the flow of malware detection from processor information obtained from the CPU. The CPU and the MDM operate in parallel and are independent of each other. The double line in the flowchart indicates the connection between the CPU and the MDM, and the access and hit signals generated by the interaction between the core and each cache are outputted to the access-hit counter in the MDM.

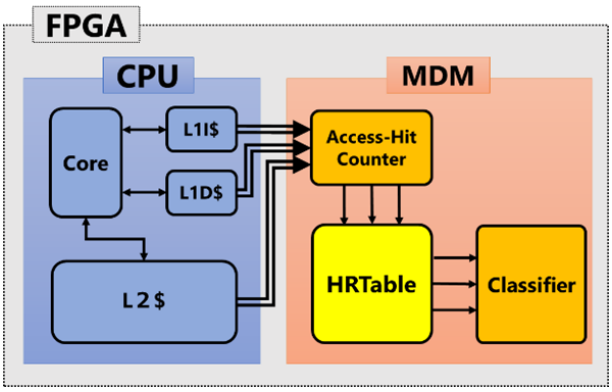

Figure 3: MDM configuration with HRTable.

3.3 Issues of the MDM and HRTable

The MDM has issues that need to be addressed in the hardware implementation. The divider used in the second process requires multiple cycles to calculate the hit rate. Figure 4 shows the number of cycles that elapse from the input of a value to the output of a divider created with Vivado, a development tool from Xilinx [5]. The number of cycles increases as the number of divisors, the number of dividends, and the bit width of the output value increase. Reducing the number of cycles until calculation of the hit rate is complete increases the complexity of the divider and the scale of the circuit, making it unsuitable for implementation in small-scale environments. Furthermore, the relationship between the classification accuracy of the classifier and the hardware scale has not yet been investigated [13, 12]. In addition, the timing of the signal processing of the HRTable and the hardware scale of the simultaneous implementation of the CPU and the MDM with the HRTable have not been explored. Details of these issues are described next.

3.3.1 Issues of the scale of the classifier and overall hardware

The hardware scale of the MDM classifier varies depending on the number of decision trees in the random forest, the depth of the trees, and the bit width of the three features. The bit width of a feature is 16 bits. However, although the accuracy of the random forest held by the classifier has been verified, the hardware scale of the classifier has not been shown. For the CPU, the Rocket Chip, which is a processor based on RISC-V [11], is adopted. However, the hardware resource utilization of the entire system connected to the Rocket Chip has not been measured and still needs to be verified.

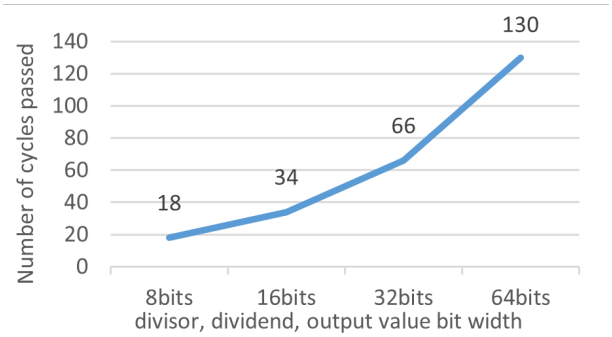

Figure 4: Number of cycles in the division process.

3.3.2 Issues of delay in HRTable input and output

The HRTable has one input port and one output port, and the access and hit signals from each cache are processed in turn by the access-hit counter every cycle (Fig. 5). Therefore, the MDM requires three cycles before the three features are passed to the classifier. In addition, to obtain three features per cycle, the HRTable must operate three times faster than the CPU frequency (Fig. 6), which increases the power consumption. However, in the study [12], only hardware scale reduction is discussed and the delay is not considered.

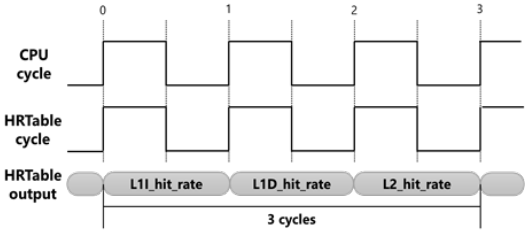

Figure 5: HRTable output (three CPU cycles).

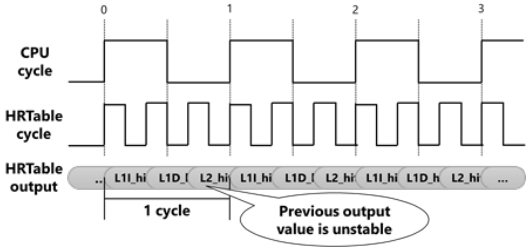

Figure 6: HRTable output (one CPU cycle).

4 PROPOSED METHOD

This section describes the proposed method. Since this paper assumes that the Rocket Chip is used as a small-scale implementation of the CPU, the connection method between the MDM and the Rocket Chip using the HRTable is first explained. Next, a circuit reduction method for the classifier is proposed in Section 3.3.1 and a cycle reduction method for the HRTable is proposed in Section 3.3.2.

4.1 Method for reducing the circuit scale of a classifier

4.1.1 Overview of Rocket Chip

The Rocket Chip, which is connected to the MDM in this research, has a framework called the Rocket tile, which consists of a Rocket core with a RISC-V instruction set architecture, a page table walker (PTW), an L1 instruction cache (L1I$), and an L1 data cache (L1D$). The Rocket Tile also consists of the SystemBus, which interfaces to each subsystem; the MemoryBus, which contains the memory system configuration including the L2 cache (L2$); the ControlBus, which contains a boot loader called during interrupts and system resets; the PeripheryBus, which provides connections

to peripheral devices; and the FrontBus, which handles read/write requests to the memory system (Fig. 7). It is also possible to use Boom cores instead of Rocket cores or to change the configuration to run multiple cores in parallel [3, 7].

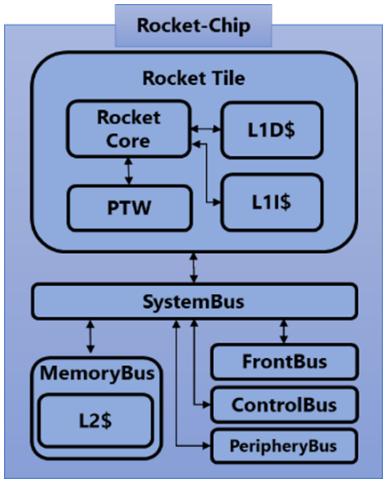

Figure 7: Overview of Rocket Chip.

4.1.2 Reduce hardware scale of MDM and CPU

This study proposes a method that does not use the L2 cache to further reduce the circuitry compared to the standard Rocket Chip configuration. It uses the program counter (PC) with the next highest contribution rate after the L2 hit rate, which is described in the paper by Koike et al. Figure 8 compares the classification accuracy of the random forest when the L1 instruction hit rate, L1 data hit rate, and PC are used as features in machine learning and when only two features, the L1 instruction hit rate and L1 data hit rate, are used. In this case, the PC is 64 bits, and the L1 instruction hit rate and L1 data hit rate are 16 bits each. The horizontal axis shows the name of the malware for which the accuracy was measured and the vertical axis (Score) shows the ratio of the number of instructions classified as attacks to the total number of instructions for each malware. For some malware, the score value tends to be very low when the hit rate alone is used. Table 1 recalculates the contribution rates of the features obtained when a random forest is created by focusing on three features for machine learning: L1 instructions, data cache hit rate, and the PC. For the MDM, this study focuses on the configuration of the classifier and the random forest and checks how much the hardware scale changes as each parameter changes, and then check the classification accuracy of the classifier when each parameter is applied.

4.1.3 Bit width reduction for program counter

We describe a method to reduce the bit width of the PC for using the PC as a new feature values in machine learning. The PC used for learning and classification holds the address of the next instruction to be executed by the program. There are two types of addresses, physical and virtual, and a RISC-V instruction set such as Rocket-Chip provides address translation mechanisms such as Sv32/32, Sv39/64, and Sv48/64 [17]. The Rocket-Chip used in this research is implemented with the Sv39/64 mechanism, which converts a 39-bit virtual address into a 64-bit physical address. When the virtual address is extended to 64 bits, 40-64 bits must be the same value as the 39 bit. Therefore, when the feature values have a bit width of 64 bits, the PC uses the sign-extended virtual address. RISCV on QEMU, which is used as training data, implements Sv48/64 and performs sign expansion when referring to the same address as above.

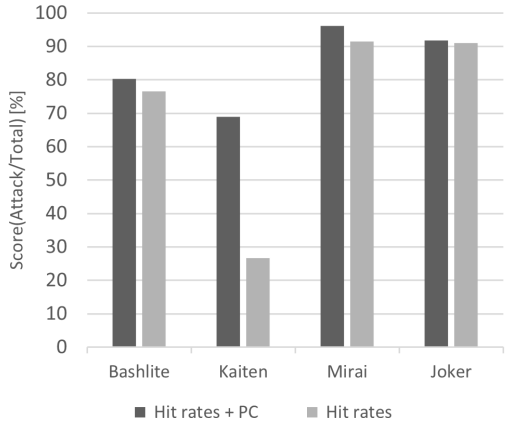

Figure 8: Instruction classification accuracy of random forests.

Table 1: Feature importance.

Features	Contribution rates (%)
Program Counter	37.79
L1 Instruction Hit rate	39.88
L1 Data Hit rate	22.33

Next, select the bit regions to be reduced. In order to maintain the classification accuracy, we leave the bit regions where a large number of feature value patterns can be collected. Therefore, we divided the acquired 64-bit wide PC into the upper 32 bits and the lower 32 bits and measured the number of patterns at each address. Table 2 shows the measurement results. Among the number of instructions used as training data, only a few patterns of address changes in the upper 32 bits were observed. This suggests that when reducing the bit width of the PC, it is effective to leave the lower bits to maintain the classification accuracy. Therefore, as a method to reduce the bit width of PC, we start from the upper bits. Since RISC-V consists of 32-bit fixed-length instructions and 16-bit compressed instructions, the value of the least significant bit of the PC is always zero. Therefore, the bit width of the PC used for learning and judgment is the value obtained by deleting the least significant bit and adding one bit to the most significant bit.

Table 2: Number of patterns in instructions.

Num of instructions	100812
High Address [63:32]	5
Low Address [31:0]	43799

4.2 Cycle Reduction Methods for HRTable

In the MDM with the HRTable, all three feature values inputted to the classifier need to be converted in the HRTable, and three cycles are required to align the feature values of one instruction processed by the CPU. In this research, the number of cycles required to classify one instruction is reduced by using the PC as a feature that does not require conversion in the HRTable.

Figure 9 shows the configuration of the Rocket tile and the MDM. In this figure, the Rocket tile has the same configuration as shown in Figure 7 and the MDM is connected to the Rocket tile by signal lines. Figure 9 illustrates the operation from the acquisition of processor information from the Rocket tile to the MDM to the classification. First, the PC is inputted to the L1 instruction cache, and the instruction is fetched. At this time, the PC output from the Rocket core is directly inputted to the classifier as processor information. Next, for the hit rate, the access-hit counter takes the access signals (L1I_access and L1D_access) and hit signals (L1I_hit and L1D_hit) from the two caches as inputs and count the number of accesses and hits in each cache. These signals are asserted using the request signal (req), which is outputted when the Rocket core accesses each cache, and the response signal (resp), which is returned by the cache to the core upon a hit, respectively. Next,

the counter converts the number of accesses and hits in the two caches into the address for the L1 instruction cache (L1I_addr), and the address for the L1 data cache (L1D_addr), and outputs them alternately to the HRTable. In the HRTable, the L1 instruction hit rate (L1_hit_rate) and L1 data hit rate (L1D_hit_rate) corresponding to each address are alternately outputted and inputted to the classifier. Thus, three feature values can be inputted into the classifier in two cycles (Fig. 10).

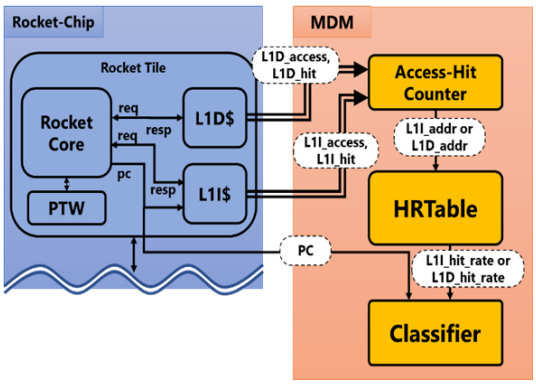

Figure 9: The process from processor information acquisition to classification.

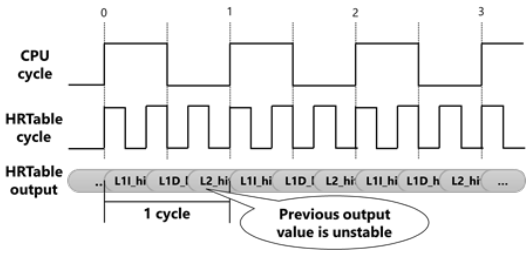

Figure 10: HRTable output (two CPU cycles).

4.3 Acquisition of feature values using DMA

Koike et al. [13] and Deguchi et al. [12] obtained the feature data for training and classification from a virtual environment. Deguchi et al. used a RISC-V image for the virtual environment. However, the Rocket-Chip implemented on the Zedboard uses the Sv39/64 virtual address system, which may differ from the feature data obtained from the virtual environment. Therefore, we use Direct Memory Access (DMA) on the Zedboard to obtain feature values directly from the target device.

5 EVALUATION METHOD AND MEASUREMENT RESULTS

This section implements the proposed method described in Section 4 and evaluates the hardware scale of the classifier, the hardware scale with the CPU and the MDM connected, and the reduction of the number of cycles of the HRTable. For evaluation of the hardware scale, it is compared with the number of available hardware resources of the Zynq-7000 series "Zedboard (xc7z020clg484-1)", which is a FPGA to be implemented. The measured Rocket Chip and the MDM are also implemented on the Zedboard, and it is confirmed that Linux for RISC-V runs on the Rocket Chip [2].

5.1 Evaluation of hardware scale of the classifier

5.1.1 Evaluation method

The degree of hardware resource reduction of the classifier is measured by reducing the bit width of features. Furthermore, the change in hardware resource utilization when increasing the number and depth of decision trees while reducing the bit width is measured and used as an indicator for weight reduction. The measurements will be performed in a simulation environment where Vivado completes the logic synthesis, placement, and wiring processes.

The main types of hardware resources are LUT (Look Up Table) and FF (Flip Flop) and the hardware scale of the classifier is checked by measuring the numbers of these two hardware elements used [6, 4]. Two of the three parameters—the number of trees, the depth of the trees, and the bit

width of the features—are fixed and the degree of variation of two of these that affects the value of the other is examined. The fixed values for each parameter are the number of trees, 5; the maximum depth of the tree, 3; and the bit width, 16 bits. The measurement patterns for each of the three parameters are shown below. In pattern 1, the bit width of the feature is varied, referring to the bit reduction method proposed by Deguchi et al. [12]. In patterns 2 and 3, the number of decision trees and the maximum depth of the trees, respectively, are varied. Since a large increase in the range of variation leads to overlearning and an increase in the time required to create a random forest, the parameters of pattern 1 are increased by 5 [1].

1. Number of trees: 5; Maximum depth: 3; Feature bit width n = 64, 32, 16, 8.

2. Number of trees: n = 20,15,10,5; Maximum depth: 3; Feature bit width: 16.

3. Number of trees: 5; Maximum depth: n = 18, 13, 8, 3; Feature bit width: 16.

5.1.2 Measurement results

Tables 3, 4, and 5 show the results of varying each parameter, respectively, on the measured hardware resource utilization. Figures 11 and 12 show the changes in each parameter of LUT and FF. First, Table 3 shows the number and percentage of resources used in the Zedboard when the bit width of the features is reduced. As the width is reduced, the number of LUTs and FFs used decreases. There is a significant difference between the 32-bit and 64-bit versions. Table 4 shows the results of varying the number of decision trees; there are gradual increases and decreases in both LUTs and FFs between 5 and 20 trees. Table 5 shows the results of varying the maximum depth of the decision trees. Compared to the results in Tables 3 and 4, the number of LUTs used as a large range of increase and decrease, while the number of FFs used remains constant.

Table 3: Number of trees: 5; Maximum depth: 3; Feature bit width: n.

Bit width	i. 64bit	ii. 32bit	iii. 16bit	iv. 8bit
LUT	722(1.36)	135(0.25)	61(0.11)	21(0.04)
FF	199(0.19)	103(0.10)	55(0.05)	30(0.03)

Table 4: Number of trees: n; Maximum depth: 3; Feature bit width: 16.

Bit width	i. 20	ii. 15	iii. 10	iv. 5
LUT	124(0.23)	115(0.22)	92(0.17)	61(0.11)
FF	70(0.07)	65(0.06)	60(0.06)	55(0.05)

Table 5: Number of trees: 5; Maximum depth: n; Feature bit width: 16.

Bit width	i. 18	ii. 13	iii. 8	iv. 3
LUT	1208(2.27)	1082(2.03)	469(0.88)	61(0.11)
FF	55(0.05)	55(0.05)	55(0.05)	55(0.05)

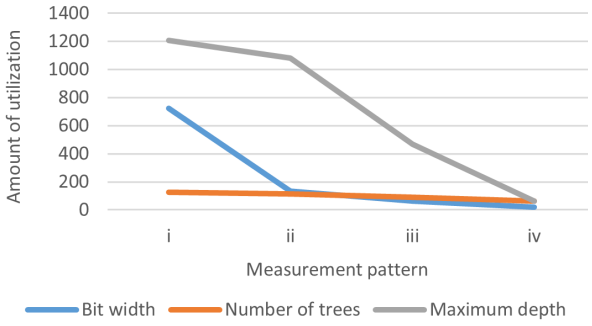

Figure 11: Resource utilization (LUT).

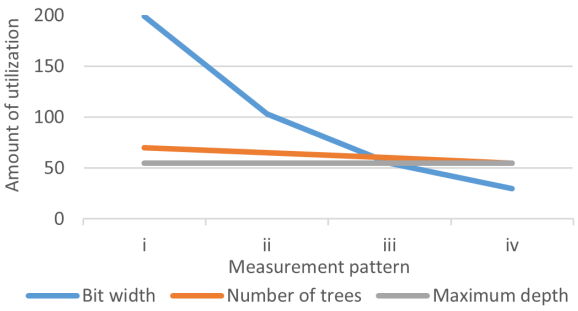

Figure 12: Resource utilization (FF).

5.2 Evaluation of overall hardware scale

5.2.1 Evaluation method

Evaluation method: The Rocket Chip is connected to the MDM as described in Section 4.1.2. As well as the classifier, the hardware resource utilization of the circuit, which has already been placed and wired, is measured and compared with the available hardware resources of the Zedboard.

The structure of the HRTable and the classifier of the MDM is as follows:

- HRTable:
  - Address width: 16 bits
  - Data width (feature value): 16 bits
- Classifier:
  - Feature data width: 16 bits
  - Random forest
    - Maximum depth: 3
    - Number of trees: 5

5.2.2 Measurement results

Table 6 shows the hardware resource utilization when the Rocket Chip and the MDM are connected. The measurements show that each hardware resource is kept within the resource range maintained by the Zedboard. Figure 13 shows the utilization rates of the Rocket Chip and the MDM separately.

Table 6: Results of measurement of hardware scale.

Resource	Utilization	Available	Utilization Rate (%)
LUT	32,942	53,200	61.92
LUTRAM	1,123	17,400	6.45
FF	16,885	106,400	15.87
BRAM	54	140	38.57
DSP	15	220	6.82
IO	9	200	4.50
BUFG	1	32	3.13
MMCM	1	4	25.00

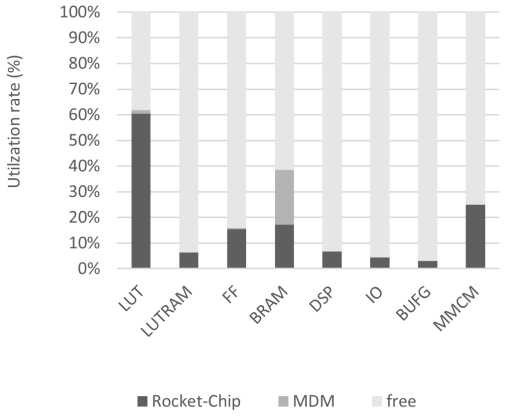

Figure 13: Utilization rates of Rocket-Chip and MDM.

5.3 Evaluation of classification accuracy for each parameter

5.3.1 Evaluation method

In this section, we confirm the effect of reducing the circuit size on classification accuracy. For each random forest of parameters with the same conditions as in Section 5.1, we send test data of malware that is processed under the same conditions as each parameter and evaluate the classification accuracy. For malware, features are collected twice for each execution and categorized into two types: for training and testing. In addition, when reducing the bit width of the features to be trained and classified, the bit width of the PC is reduced using the method described in Section 4.1.3. For each hit rate, we use the method used in the study by Deguchi et al. The malware to be measured in this study is a DDoS-type malware, which is generally assumed to be an attack target for IoT devices.

5.3.2 Measurement results

Figures 14, 15, and 16 show the classification accuracy of the random forest within each classifier evaluated in Section 5.1. Figures 14, 15, and 16 show the measurement results when the bit width, the number of decision trees, and the depth of the decision trees are varied. Some malware showed almost no change in accuracy under each condition, while others showed a significant change. In this measurement, the smallest difference between the maximum and minimum values of each parameter was 0.176%, and the largest difference was 18.604%.

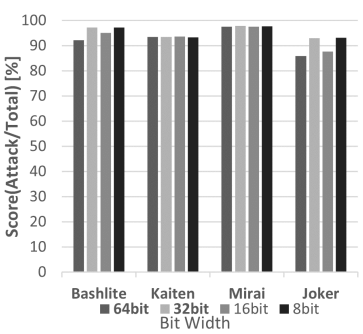

Figure 14: Classification accuracy when changing bit width.

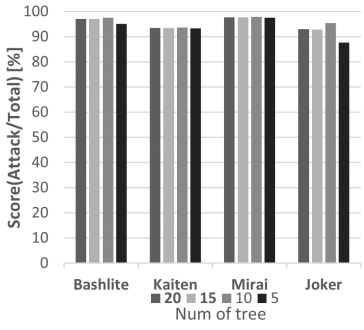

Figure 15: Classification accuracy when changing the number of decision trees.

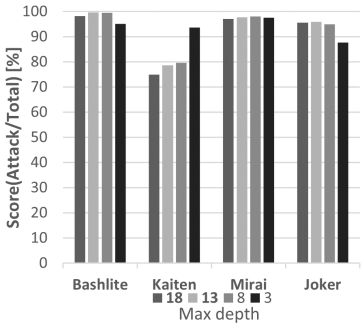

Figure 16: Classification accuracy at maximum depth change.

5.4 Evaluation of HRTable for reducing the number of cycles

5.4.1 Evaluation of delay in HRTable

5.4.1.1 Evaluation method To confirm whether it is possible to implement the MDM on the Zedboard, considering the delay caused by the implementation of the HRTable, a test MDM is constructed that retains only the access-hit counter functions and simulations are performed using development tools. We use Vivado's "Post-Implementation Functional Simulation" and "Post-Implementation Timing Simulation" as the simulation environments for measurement. First, in Functional Simulation, the logic synthesis and the design after placement and wiring are simulated to check the correspondence between input and output values. Next, in Timing Simulation, the above design is simulated with signal processing delay taken into account. The results of the two simulations are compared and it is confirmed that the input and output of the HRTable function are properly calculated.

The test MDM receives randomly generated access signals (L1I_access and L1D_access) and hit signals (L1I_hit and L1D_hit) every cycle from the test bench of the upper-level module and generates an L1 instruction address and an L1 data address (L1I_addr or L1D_addr) that are alternately sent to the HRTable. Finally, the HRTable returns the hit rate (L1I_hit_rate or L1D_hit_rate) corresponding to the address to the test bench (Fig. 17). One cycle is 20 ns, which is equal to the 50 MHz clock frequency of the Rocket Chip that is implemented in conjunction with the MDM. The operating frequency of the Rocket Chip can be calculated from the following (1):

$$Rocket\ Chip\ Clock\ rate\ (MHz) = \frac{1000}{Zynq\_CLK\_Period} * \frac{RC\_CLK\_Mult}{RC\_CLK\_Divide} \quad (1)$$

The parameters in (1) are:

- Zynq_CLK_Period: 10.0,
- RC_CLK_Mult: 10.0, and
- RC_CLK_Divide: 20.0.

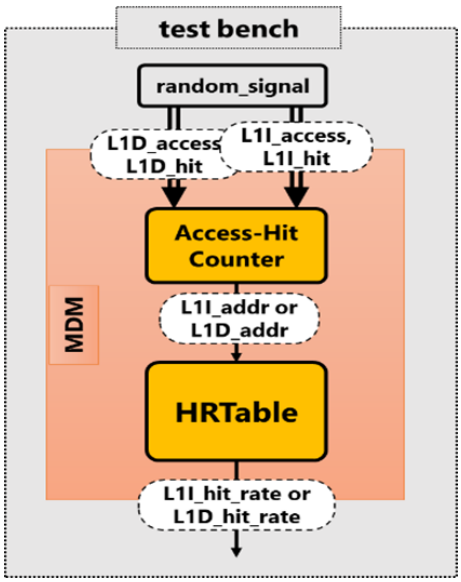

Figure 17: Testbench configuration.

5.4.1.2 Measurement results The waveform results obtained during the simulation are shown in Figures 18 and 19. First, in Figure 18, the HRTable returns the output value (L1I or L1D hit rates) corresponding to the obtained input value (L1I_addr or L1D_addr) with a delay of one clock cycle. The output value is the result of dividing the upper 8 bits of the input value by the lower 8 bits. For example, for the input value 1c33(16), we multiply 1c / 33 = 0.5490(10) by 1,000 and output 5,490(10). Next, Figure 19 shows a simulation of Figure 18 with an additional signal processing delay. There is a delay of approximately 10 ns until the input/output values in the HRTable and the registers are fixed in response to the rising edge of the clock (CPU cycle), but the output values are stable at the falling edge of the clock, and the input values and output values correspond.

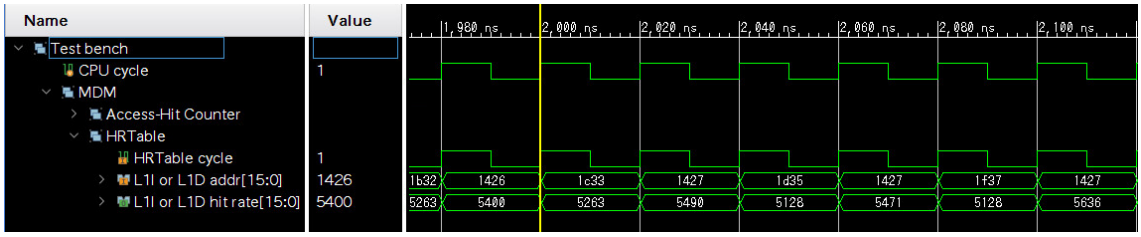

Figure 18: Functional Simulation waveform.

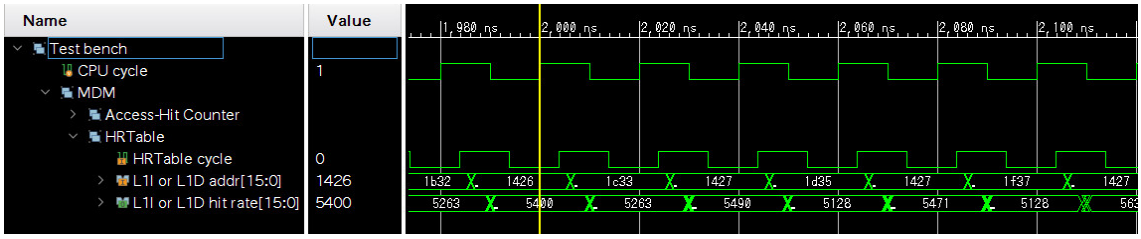

Figure 19: Timing Simulation waveform.

5.4.2 Evaluation of delay of the overall circuit

5.4.2.1 Evaluation method In Section 5.1, we created a classifier that consists of several different parameters. In Section 5.4.1, we reduced the number of cycles in the HRTable and confirmed that the delay was within an acceptable range. In this section, we implement the malware detection mechanism by combining the reduced number of cycles of the HRTable and each classifier and performing timing analysis in the circuit with the Rocket-Chip and the malware detection mechanism combined to check for timing violations. The state of the delay in the circuit is expressed as the difference (WNS, WHS, defined below) between the setup time, the hold time, and the analysis result by Vivado.

The setup time is the minimum time in which the data to be received must be fixed prior to the rising edge of the clock, and the Worst Negative Slack (WNS) is the value that has the smallest difference from the Vivado analysis results in the circuit. The hold time is the minimum time in which the data received after the rising edge of the clock must be held unchanged, and the value with the smallest difference from the Vivado analysis result in the circuit is called Worst Hold Slack (WHS). A negative value for WNS and WHS indicates that there is an unacceptable delay in the circuit.

5.4.2.2 Measurement result Table 7 shows the results of the timing analysis of the overall circuit for each parameter. Although there seems to be no regularity in the values of WNS and WHS for each parameter, we can not confirm any combination of parameters that resulted in negative analysis results for WNS and WHS. Therefore, it can be said that the overall circuit using the classifier created with the parameters evaluated in this section satisfies the timing constraints.

Table 7: Timing analysis results

Parameters	WNS	WHS
bw: 8, nt: 5, md: 3	0.250	0.035
bw:16, nt: 5, md: 3	0.181	0.030
bw:32, nt: 5, md: 3	0.389	0.019
bw:64, nt: 5, md: 3	0.412	0.014
bw:16, nt:10, md: 3	0.143	0.032
bw:16, nt:15, md: 3	0.015	0.037
bw:16, nt:20, md: 3	0.727	0.033
bw:16, nt: 5, md: 8	0.002	0.020
bw:16, nt: 5, md:13	0.193	0.036
bw:16, nt: 5, md:18	0.305	0.025

5.5 Acquisition of feature values using DMA

5.5.1 Evaluation method

Linux is run on the Rocket-Chip to obtain the feature values. Since the number of instructions of a program obtained on the virtual environment is approximately 10,000 we measure whether 10,000 instructions can be obtained at a time by DMA. For the program to be executed, we use the same malware as the one obtained in the virtual environment, but we restrict the function to access the Internet. The bit width of the PC is assumed to be 64 bits (8bytes).

5.5.2 Measurement result

The PC was acquired from the Rocket-Chip using DMA. In this measurement, we were able to obtain up to 12,000 instructions of 64-bit data. However, when transferring a larger size by DMA, Linux on the Rocket-Chip crashed, so it is necessary to consider other countermeasures.

Next, we measured the number of patterns of PC changes obtained by executing malware on the actual machine, and the results are shown in Table 8. In the acquired data, there were some addresses that had the same address consecutively and some addresses that were accessed repeatedly at regular intervals.

Table 8: Number of patterns on the PC when executing malware

Number of data	12500
Number of patterns	2353

6 DISCUSSIONS

In this section, we discuss the evaluation results obtained in Section 5.

6.1 Hardware scale of the classifier

In Section 5.1, the effect of each parameter on the hardware resource utilization of the classifier was explored by varying the configuration of the bit width of the features, the number of decision trees, and the maximum depth of the decision trees. The gradual increase in resource utilization caused by increasing the number of decision trees can be attributed to the number of features. Each node of a decision tree selects one of the input features as a condition for branching to a lower node and decides whether the feature exceeds a certain value or not. However, since the classifier has only three features, nodes with the same branching condition are formed in other decision trees. In addition, Vivado is equipped with an optimization function, and branch circuits with overlapping conditions

are combined into a single circuit with the same nodes mentioned above through optimization. Therefore, it can be assumed that the number of resources used is also reduced to a minimum. Next, the fact that a large change in the number of resources used was observed at the maximum depth suggests that, as the nodes become deeper, branching conditions that are not used at shallower nodes appear, and this is reflected in the number of resources used. For each classifier, the hardware resource utilization of the Zedboard is a very small percentage, suggesting that it is possible to increase the scale of the decision tree structure to improve detection accuracy.

6.2 Overall hardware scale

In Section 5.2, it was confirmed that the hardware scale of the combined circuit of the Rocket Chip and the MDM was within the amounts of each hardware resource in the Zedboard. The hardware resource utilization of the Rocket Chip exceeds most of the total utilization, but the MDM is kept to a small amount, and there is a surplus in the resource utilization in the Zedboard (Fig. 13). Therefore, it is possible to add or improve the functions of the MDM. In particular, the amount of FF used is relatively small, which is because the types of features in the classifier and the number of bits in each feature are limited.

6.3 Evaluation of classification accuracy for each parameter

We measured the classification accuracy against the learned malware by changing the parameters for creating the classifier. In this measurement, we confirmed that the classification accuracy was high throughout and that the accuracy could be maintained even when the circuit scale was small.

However, the classification accuracy against unlearned malware and the false positive rate against ordinary programs have not been measured yet. Therefore, if the above verification shows a decrease in the classification accuracy or an increase in the false positive rate, then it is necessary to adjust the parameters. In addition, since the malware is limited to the DDoS type, it is also necessary to learn programs with different behaviors, such as those that perform malicious mining.

Furthermore, the proposed mechanism uses a program counter and cache hit rates, and is considered to be effective and versatile for any CPU equipped with a cache.

6.4 Reducing the number of cycles in HRTable

The measurement results for the HRTable in Section 5.4 show that the delay in the HRTable is within one clock cycle of the Rocket Chip. Therefore, it is expected that the throughput in the calculation of the cache hit rate can be reduced by using this HRTable when it is implemented on the Zedboard. In addition, we conducted timing analysis of the Rocket-Chip and the overall malware detection mechanism using an HRTable with a reduced number of cycles and confirmed that all the classifiers we created satisfied the timing constraints.

6.5 Acquisition of feature values using DMA

We used DMA to transfer data directly from the Zedboard. As future work, we need to train using the features obtained from the DMA and verify whether there is a difference in the training between using the data obtained in the virtual environment and that obtained the DMA. If there is no difference, then the hardware resources of the DMA itself can be reduced by using the data in the virtual environment. If there is a difference, then the data needs to be acquired from the DMA, so the DMA needs to be implemented when obtaining data for machine learning. However, it is difficult for the resources of the IoT device to perform malware detection on the software of the IoT device using the data transferred from the DMA. In addition, since the time required for data transfer by DMA is added to the time required for detection, the detection speed is slower than that of a hardware-based malware detection mechanism. Therefore, it is desirable to use DMA only for creating a classifier.

7 CONCLUSION

In this paper, we proposed and evaluated the implementation of a malware detection mechanism (MDM) and CPU on a small LSI using the Hit Rate Table (HRTable) proposed by Deguchi et al. [5]. For the classifier, the variation of hardware resource utilization in the classifier was evaluated by adjusting the bit width of features, the number of decision trees constituting a random forest, and the maximum depth. Next, we evaluated the classification accuracy of the classifiers whose hardware resources were measured and found that they showed high accuracy in each parameter. By connecting the reduced MDM to the connected Rocket Chip and measuring the hardware scale, we confirmed the feasibility of implementation on the Zedboard. In the HRTable, the hit rate for one instruction was obtained in a total of two cycles by using the program counter (PC) instead of the processor information obtained from the L2 cache. In addition, timing analysis was performed to confirm that the timing constraints were satisfied overall in the circuit.

Next, a data tracing method on hardware using DMA was evaluated, and it was shown that the number of instructions required for learning could be obtained.

In the future, we will compare the feature values acquired by the DMA with those acquired in the virtual environment, and verify whether it is necessary to create a classifier again. In addition, we will run the CPU with the MDM on the Zedboard, and we will evaluate whether it is possible to distinguish between normal programs and malware.

References

[1] In depth: parameter tuning for random forest, 2017. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d.

[2] riscvarchive/riscv-buildroot, 2018. https://github.com/riscvarchive/riscv-buildroot/tree/riscv-start.

[3] ucb-bar/fpga-zynq, 2018. https://github.com/ucb-bar/fpga-zynq.

[4] Vivado design suite 7 series fpga and zynq-7000 soc libraries guide, 2018. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug953-vivado-7series-libraries.pdf.

[5] Vivado design suite user guide, 2018. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug973-vivado-release-notes-install-license.pdf.

[6] Zynq-7000 soc data sheet: overview, 2018. https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[7] 3.1 rocket chip, 2019. https://chipyard.readthedocs.io/en/latest/Generators/Rocket-Chip.html.

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

[9] Iot security issues in 2021: a business perspective, 2021. https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/magazine/internet-threats.

[10] Security ip: Trustzone, 2021. https://developer.arm.com/ip-products/security-ip/trustzone.

[11] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, and David Biancolin. The rocket chip generator. Technical report, Electrical Engineering and Computer Sciences at University of California at Berkeley, 2016.

[12] Mutsuki Deguchi, Masahiko Katoh, and Ryotaro Kobayashi. Preliminary evaluation for fpga implementation of malware detection mechanism using processor information. *Computer Security Symposium 2020 Proceedings*, pages 404–409, 2020.

[13] Kazuki Koike, Ryotaro Kobayashi, and Masahiko Katoh. Reduction of classifier size and acceleration of classification algorithm in malware detection mechanism using processor information. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 339–345, 2019.

[14] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[15] Abraham Peedikayil Kuruvila, Shamik Kundu, and Kanad Basu. Analyzing the efficiency of machine learning classifiers in hardware-based malware detectors. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 452–457, 2020.

[16] Quoc-Dung Ngo, Huy-Trung Nguyen, Van-Hoang Le, and Doan-Hieu Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286, 2020.

[17] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David Patterson, and Krste Asanović. The risc-v instruction set manual volume ii: Privileged architecture version 1.7. Technical report, Electrical Engineering and Computer Sciences at University of California at Berkeley, 2015.