ISSN: 2182-2069 / E-ISSN: 2182-2077

Effectiveness of MAC Systems based on LSM and their Security Policy Configuration for Protecting IoT Devices

Masato Miki[1], Toshihiro Yamauchi[2*], and Satoru Kobayashi[3]

[1]Graduate School of Natural Science and Technology, Okayama University, Japan.
pwuo7fgl@s.okayama-u.ac.jp, https://orcid.org/0009-0001-6383-653X

[2*]Faculty of Environmental, Life, Natural Science and Technology, Okayama University, Japan.
yamauchi@okayama-u.ac.jp, https://orcid.org/0000-0001-6226-5715

[3]Faculty of Environmental, Life, Natural Science and Technology, Okayama University, Japan.
sat@okayama-u.ac.jp, https://orcid.org/0000-0003-1017-0938

Received: March 30, 2024; Revised: May 29, 2024; Accepted: June 27, 2024; Published: August 30, 2024

Abstract

The number of attacks exploiting Internet of Things (IoT) devices has been increasing with the emergence of IoT malware targeting IoT devices. The use of IoT devices in a wide variety of situations has resulted in an urgent need to improve the security of the IoT devices themselves. However, the IoT devices themselves have low hardware performance and their operating systems and applications are not frequently updated, leaving many devices vulnerable to IoT malware attacks. Mandatory Access Control (MAC) systems based on Linux Security Modules (LSM), such as SELinux and AppArmor, can mitigate the impact of these attacks, even if software vulnerabilities are discovered and exploited. However, most IoT devices do not currently employ these systems. While existing approaches have examined on-board resources as one factor affecting the applicability of MAC systems, they are insufficient to address all relevant factors. In this paper, we report the factors that may prevent the deployment of LSM-based secure OS in IoT devices and the results of our evaluation of the effectiveness of LSM-based secure OS against IoT malware attacks. First, we comprehensively investigated the impact of each factor of IoT devices on the deployment of LSM-based secure OS. To improve the comprehensiveness of the factors affecting the deployment, we investigated the kernel version, CPU architecture, and BusyBox support. Next, we conducted an attack experiment that simulated the attack method of Mirai, a typical IoT malware, to investigate whether it is possible to protect against IoT malware. We also showed how to modify the security policy, and the cost of modifying it, for secure OSs that cannot prevent attacks from IoT malware with the default security policy. Finally, we report the results of our investigation into the impact of these factors in combination.

Keywords: Mandatory Access Control System, IoT Security, Linux Security Modules.

Journal of Internet Services and Information Security (JISIS), volume: 14, number: 3 (August), pp. 293-315.
DOI: *10.58346/JISIS.2024.I3.018*

*Corresponding author: Faculty of Environmental, Life, Natural Science and Technology, Okayama University, Japan.

1 Introduction

The number of attacks exploiting IoT devices has been increasing with the emergence of IoT malware targeting IoT devices. Moreover, attack methods that exploit software vulnerabilities in IoT devices are becoming more common. Therefore, improving the security of IoT devices is essential.

It is challenging for security measures to focus on specific attack methods to mitigate the impact of attacks exploiting the most recent vulnerabilities. Linux OS is widely used in IoT devices (Eclipse, 2019); thus, such devices cannot prevent attacks once an attacker gains root privilege. One possible approach is to restrict redundant file operations, which remains effective even if attackers have root privileges. To provide these security features, Linux mainline kernels support MAC systems, such as SELinux (SELinux Wiki contributors, 2017) and AppArmor (Ubuntu Wiki, 2021), based on LSM. These systems can restrict file operations—such as read, write, and execution—based on security policy rules. MAC systems are effective even for users with root privileges (Salve, 2020).

However, Akiyama et al., (Akiyama et al., 2023) reported that MAC systems are not actually deployed in most IoT devices. Previous studies have focused on resources on IoT devices as a factor preventing MAC systems from being deployed. However, they have not comprehensively investigated factors related to the actual security of IoT devices, such as protection against IoT malware (Aswathy et al., 2023). They also did not explore the impact of combinations of these factors. Revealing these factors is helpful in determining which MAC systems should be used in IoT devices (Iman et al., 2023). Therefore, these factors need to be clarified comprehensively. While previous studies did not clarify how the limitation on the actual security of IoT devices impacts on the effectiveness of MAC systems, our study investigated factors which related with current limitation or environment on the actual security of IoT devices comprehensively and overcame this gap.

Our research aims to conduct a thorough investigation into the factors influencing the deployment of MAC systems based on LSM in IoT devices. To address this objective, we analyzed three critical aspects to determine the frequency with which technical factors hinder the deployment of MAC systems on IoT devices: (1) their applicability to IoT devices, (2) their effectiveness in protecting against IoT malware, and (3) the cost associated with updating security policies.

To increase the comprehensiveness of the applicability factors, we investigated additional technical aspects such as the kernel version, CPU architecture, and BusyBox support. We chose the kernel version to investigate whether IoT devices install kernel versions which lack the support of MAC systems frequently (Surendar et al., 2024). We focused on CPU architecture to clarify whether MAC systems can be deployed on any CPU architecture because IoT devices uses various CPU architecture. We selected support for BusyBox to investigate whether MAC systems can distinguish each command executed by it and work properly since BusyBox is widely used in IoT devices while there exists a concern which it cannot be controlled properly because it is a single file. We also examined factors previously identified in studies, including memory consumption (Nakamura et al., 2015), file systems (Nakamura, 2006), and processing delays (Nakamura et al., 2015). Additionally, we conducted simulation experiments using the Mirai attack method with both default and customized settings to assess whether MAC systems can protect IoT devices from malware (Bobir et al., 2024). Moreover, we compared the rules added in the customized settings and analyzed the cost of updating security policies to protect IoT devices from Mirai. We investigated the impact of various combinations of these factors on the adoption of MAC systems. This paper extends a conference paper (Miki et al., 2023) with further details.

2 MAC Systems in IoT Devices

2.1 Overview of LSM-Based MAC Systems

A security framework (LSM) was implemented on Linux systems to facilitate the deployment of MAC systems. LSM comprises a set of hook functions for security checks within the Linux kernel, which are applied to users with root privileges as well.

This study investigates several prominent MAC systems, including SELinux (SELinux Wiki contributors, 2017), Smack (Schaufler, 2011b), TOMOYO Linux (TOMOYO Linux, n.d.c), and AppArmor (Ubuntu Wiki, 2021). These MAC systems are categorized into two types: label-based and path-name-based. Label-based MAC systems rely on extended attribute (xattr) support in file systems, using an attribute called 'label' for MAC purposes, which is stored in xattr. In contrast, path-name-based MAC systems do not require such support, as they implement MAC through the use of path names.

2.2 Advantages of MAC Systems in Securing IoT Devices

For instance, SELinux can protect Linux systems against attacks targeting vulnerabilities such as CVE-2019-13272, which is exploited for privilege escalation (Red Hat, 2023). The implementation of security measures in IoT devices is often constrained by limited onboard resources (Hossain et al., 2015). However, there are scenarios in which MAC systems can be effectively deployed on IoT devices by optimizing resource usage. For example, SELinux can be configured for use on a device with 64MB of memory by adjusting its settings to limit memory consumption to under 1MB (Nakamura et al., 2015). Additionally, it has been demonstrated that the processing delays associated with MAC systems do not significantly impair file operation performance in many instances (Zhang et al., 2021).

2.3 Problems of MAC Systems in the Protection of Actual IoT Devices

Akiyama et al., (Akiyama et al., 2023) found that, among an analysis of 893 GPL source codes corresponding to the firmware of actual IoT devices, SELinux was adopted in only six instances, and no other MAC systems were observed. Previous research has identified three potential barriers to the adoption of SELinux:

1. Excessive memory consumption with default settings (Nakamura et al., 2015).
2. Ineffectiveness of per-file MAC on file systems lacking xattr support (Nakamura, 2006).
3. Significant processing delays associated with default settings (Nakamura et al., 2015).

Although they clarified the aforementioned factors individually, they did not reveal the whole picture of factors that impact on the adoption of MAC systems on IoT devices. When a decision which related to the deployment of MAC systems is made without a whole picture of the factors, it can be biased or erroneous. Thus, it is improper to make decisions that are related to MAC system adoption using only these previous studies, and the whole factors that impact on the adoption of MAC systems on IoT devices in the current situation have to be clarified. Therefore, these factors and the impact of each factor on the adoption of MAC systems need to be investigated comprehensively.

However, previous studies have inadequately explored the factors influencing the adoption of MAC systems. For example, they have not thoroughly examined the impact of the software used in IoT

devices, such as the Linux kernel and BusyBox, nor have they considered the hardware characteristics of IoT devices, including the variability of CPU architectures. Furthermore, the effects of protection against real-world IoT malware and the costs associated with updating security policies have not been addressed.

3 Investigation on the Applicability to IoT Devices

To comprehensively investigate the factors influencing the adoption of LSM-based MAC systems for IoT devices, we investigated three key aspects: applicability to IoT devices, protectability against IoT malwares, and the cost for updating security policies. This section presents the results of a comprehensive investigation into the factors that impact the applicability of MAC systems for IoT devices.

3.1 Investigation Strategy

In addition to the impacts of memory consumption, file systems, and processing delays, which have been investigated in previous studies, we investigated the effects of several notable characteristics of IoT devices:

1. Older Linux kernels (Akiyama et al., 2023): Older versions of the Linux kernel do not officially support MAC systems. We investigate the prevalence of this issue in actual IoT devices.
2. Different CPU architectures: Previous studies have reported that the CPU architecture does not cause the prevention of using SELinux (Nakamura & Sameshima, 2008) or TOMOYO Linux (TOMOYO Linux, n.d.b). However, the findings from the former study (Nakamura & Sameshima, 2008) were published in 2008, and the impact on current MAC systems has not been reassessed since then. Therefore, we investigated whether CPU architecture might influence the applicability of contemporary MAC systems.
3. Installation of BusyBox: It has been reported that SELinux and AppArmor provide per-BusyBox MAC command controls (Nakamura, 2007). However, it is not yet known whether other MAC systems can manage each BusyBox command differently. Therefore, we investigated whether BusyBox commands can be protected based on their individual functions.

3.2 Investigation Methods and Results

3.2.1 Memory Consumption

The evaluation environment is listed in Table 1. We evaluated the memory consumption. SELinux, Smack, and AppArmor were applied with their default security policies. For TOMOYO Linux, we used a security policy based on MAC rules from execution history because the default security policy provided by TOMOYO Linux does not contain any MAC rules. Linux capabilities were enabled by default in this environment.

The procedure for measuring memory consumption is shown as follows.

1. We measured the memory use one minute after system login using the free command. The measurement is conducted five times for each system.
2. We calculated the average memory usage from these measurements.

3. We determined the memory consumption with the MAC system by subtracting the average memory usage with all MAC systems disabled from the average memory usage with the MAC system enabled.

Table 2 summarizes the evaluation results.

Table 1: Performance Evaluation Environment

CPU	BCM2837B0 (ARMv8) 1.4GHz
OS	Ubuntu Server 20.04.3 LTS (32bit)
Kernel Version	5.4.83-v7+

Table 2: Measured Memory Consumption (Unit: MB)

MAC System	**Entire System**	**Increase**
None	60.851	-
SELinux	112.573	51.722 (85.0%)
Smack	61.273	0.422 (0.7%)
TOMOYO Linux	62.213	1.363 (2.2%)
AppArmor	74.273	13.423 (22.1%)

We compared these results to those in Table 3, showing the memory capacities of IoT devices listed as available by OpenWrt (OpenWrt Project, 2023). Some devices do not have any information of memory capacities, which are excluded in the following analysis.

Table 3: Memory Capacity of IoT Devices Supported by OpenWrt as Available Devices

Memory Capacity	**Number**	**Memory Capacity**	**Number**
16MB	1	1,024MB-	56
32MB	51	2,048MB	18
64MB	169	4,096MB	12
128MB	175	More than 4GB	1
256MB	104	8,192MB	5
512MB	64	More than 8GB	1
		Total	657

We established four progressive criteria to evaluate whether the memory usage was acceptable for each MAC system:

Criterion 1 (C1): Whether memory usage is less than 1% of the onboard memory capacity (Whether an applied MAC system do not squeeze memory on any Linux distribution).

Criterion 2 (C2): Memory usage is less than 2% of the onboard memory capacity. This criterion evaluates whether the MAC system fits within the memory constraints of a Linux distribution under more relaxed conditions compared to C1.

Criterion 3 (C3): The total memory usage of the MAC system and the OpenWrt kernel (3,828 KB (OpenWrt Project, 2018)) must be less than the onboard memory capacity. This criterion evaluates whether the memory is not depleted solely by the OpenWrt kernel.

Criterion 4 (C4): The total memory usage of the entire system is less than the onboard memory capacity. Devices with a memory capacity of is less than 64 MB were excluded. This criterion assesses whether the combined memory usage of the MAC system and the entire Ubuntu Server does not deplete the available memory.

Table 4 describes the evaluation results of each MAC system based on these criteria. The results show that, aside from Smack, only a few services could meet the memory usage requirements specified in C1 and C2 for MAC systems. However, in fewer than one-third of all cases, the system's memory usage exceeded the onboard memory capacity, as observed in C3 and C4. Therefore, while memory consumption has a discernible impact on applicability, it should be noted that there are only a few instances where we cannot rely on MAC systems due to memory constraints.

Table 4: The rate of devices that tolerate MAC system memory usage on each criterion

MAC System	**C1**	**C2**	**C3**	**C4**
SELinux	0.91%	2.89%	92.09%	72.07%
Smack	92.09%	99.85%	100.00%	100.00%
TOMOYO Linux	39.73%	66.36%	100.00%	100.00%
AppArmor	5.63%	14.16%	99.85%	72.07%

3.2.2 File Systems

We demonstrated the availability of MAC systems with different file systems. For the demonstration, we use a simulated IoT device described in Table 5. The file systems we investigated are listed in Table 6. Cramfs, romfs, JFFS2, and SquashFS are widely used file systems, which cover all of the file systems used in the firmware analysis by Liu et al., (Liu et al., 2021). We also included tmpfs because its reported abuse by IoT malware (Alrawi et al., 2021).

Table 5: Basic Environment for Evaluating MAC Systems

CPU	BCM2837B0 (ARMv8) 1.4GHz
OS	Ubuntu Server 20.04.3 LTS (64bit) (Except for TOMOYO Linux) Ubuntu Server 20.04.3 LTS (32bit) (TOMOYO Linux)
Kernel Version	5.4.0-1047-raspi (Except for TOMOYO Linux) 5.4.83-v7+ (TOMOYO Linux)

Table 6: File Systems to be Examined

File System	***xattr* Support**	**Read/Write Permission**
Cramfs	No	Read-only
romfs	No	Read-only
JFFS2	Yes	Readable/Writable
SquashFS	Yes	Read-only
tmpfs	Yes	Readable/Writable

We examined the granularity of file-reading restrictions (either per-file or per-file-system) within each file system to assess the proper functioning of the MAC systems. To this end, we categorized the target files into two groups: files restricted from being read by the MAC systems (restricted files) and files permitted for reading (unrestricted files). A per-file MAC is deemed functional if processes can access unrestricted files while being denied access to restricted files. Conversely, if this condition is not met, we consider the per-file MAC to be non-functional. Table 7 presents the results.

Table 7: Control Granularity of MAC Systems in Each File System (Do Not Consider that on Processes)

MAC System	**Cramfs**	**romfs**	**JFFS2**	**SquashFS**	**tmpfs**
SELinux	per-FS	per-FS	per-file	per-file	per-file
Smack	per-FS	per-FS	per-file	per-file	per-file
TOMOYO Linux	per-file	per-file	per-file	per-file	per-file
AppArmor	per-file	per-file	per-file	per-file	per-file

Label-based MAC systems, such as SELinux and Smack, provide per-file-system (per-FS) MAC on file systems that lack xattr support, as detailed in Table 7. In contrast, pathname-based MAC systems, including TOMOYO Linux and AppArmor, offer per-file MAC across all file systems.

Based on these results, we conclude that the absence of xattr support significantly impacts the applicability of MAC systems. A per-FS MAC protocol is unable to enforce distinct access control rules for individual files within a single file system. This limitation hinders the effectiveness of MAC systems in providing comprehensive protection against attacks. Consequently, SELinux and Smack offer constrained MAC functionality and are unsuitable for deployment on file systems without xattr support.

3.2.3 Processing Delays

We assessed the processing delay introduced by each MAC system in the environments described in Table 1. Initially, we configured LMbench 3.0-19 (LMbench) with 512 MB of memory and a CPU frequency of 1,397 MHz. LMbench was executed five times, and the processing delay was computed as the difference between the results obtained with and without each MAC system.

The evaluation results are summarized in Table 8. SELinux introduced an overhead ranging from approximately 16.0% to 40.2% for operations except read/write, where the overhead was relatively minimal. Smack exhibited substantial overhead in stat, open/close, and file creation operations, but showed relatively low overhead in other operations. TOMOYO Linux demonstrated overheads of approximately 123.3% and 59.6% for stat and open/close operations, respectively, while maintaining minimal overhead for read/write operations. AppArmor showed a sufficiently low measurement overhead. We observed errors except read/write, as these operations were unaffected by the LSM hooks.

Table 8: Comparison of Processing Delays in Each MAC System (Unit: μs)

MAC System	**stat**	**open/ close**	**read/ write**	**0Kfile Create**	**0Kfile Delete**	**10Kfile Create**	**10Kfile Delete**
None	4.00	11.1	0.81	48.6	35.3	101.9	66.2
	(-)	(-)	(-)	(-)	(-)	(-)	(-)
SELinux	4.64	13.3	0.82	68.2	37.8	122.0	64.4
	(16.0%)	(19.4%)	(1.7%)	(40.2%)	(7.0%)	(19.7%)	(-2.7%)
Smack	5.30	13.4	0.70	60.0	37.9	111.6	65.2
	(32.5%)	(20.9%)	(-13.1%)	(23.3%)	(7.3%)	(9.6%)	(-1.6%)
TOMOYO	8.94	17.7	0.71	61.4	40.9	109.8	67.9
Linux	(123.3%)	(59.6%)	(-11.9%)	(26.3%)	(15.7%)	(7.7%)	(2.6%)
AppArmor	3.84	10.3	0.82	48.6	34.2	96.9	60.8
	(-4.0%)	(-7.2%)	(1.7%)	(-0.1%)	(-3.1%)	(-4.9%)	(-8.2%)

Thus, we conclude that increased processing delays impact the applicability of the MAC systems, especially SELinux, Smack, and TOMOYO Linux. However, it remains unclear whether this impact is substantial enough to render them inapplicable.

3.2.4 Kernel Version

Table 9 outlines the mainline kernel versions integrated with each MAC system. To assess the support rates for these kernel versions, we compared the kernel versions installed on IoT devices with those listed in Table 9. Our analysis focused on firmware from IoT devices collected over a three-year period, from 2019 to 2021, to identify the kernel versions used in these devices.

Table 9: The Mainline Kernel Version with which each MAC System was Merged

MAC System	Kernel version
SELinux	2.6.0 (National Security Agency, n.d.)
Smack	2.6.25 (Corbet, 2008)
TOMOYO Linux	2.6.30 (TOMOYO Linux, n.d.a)
AppArmor	2.6.36 (AppArmor contributors, n.d.)

There were 408, 229, and 115 kernel versions in 2019, 2020, and 2021 from our tabulation result, respectively, excluding kernels with unknown versions. Fig. 1 shows the comparison results of collected kernel versions supporting each MAC system.

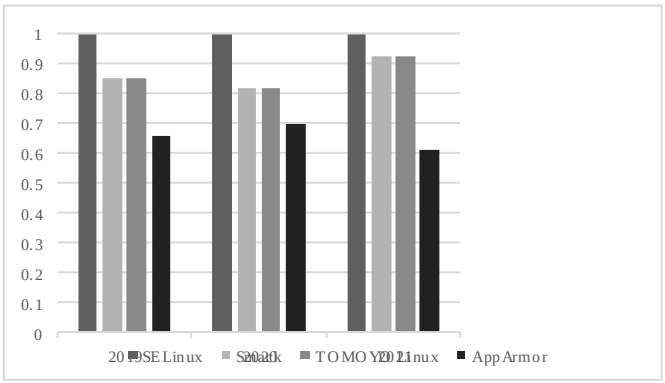

Figure 1: Kernel Support Rate for each MAC System

As shown in Figure 1, all the kernel versions support SELinux. Smack and TOMOYO Linux are supported in more than 80% of the collected kernel versions. In contrast, over 30% of the collected kernel versions do not support AppArmor. Additionally, less than 20% of the collected kernel versions lack the support for any pathname-based MAC systems.

Hence, we believe that the kernel version has a minimal impact on the applicability of MAC systems today, although the effect is not definitely determined. In addition, we expect this impact to decrease over time as IoT devices should adopt newer kernel versions.

3.2.5 CPU Architecture

We examined the impact of CPU architecture to the applicability of MAC systems using the method proposed by Nakamura and Sameshima (Nakamura & Sameshima, 2008). This method involved analyzing the arch directory of the kernel source code to evaluate the impact of CPU architecture. For our analysis, we used kernel version 5.4.83-v7+ source code.

Our findings confirm that MAC systems are available regardless of CPU architecture. Therefore, we conclude that CPU architecture has no impact on the applicability of MAC systems.

3.2.6 Support for BusyBox

We evaluated whether Smack and TOMOYO Linux can work correctly within the BusyBox environment, as shown in Table 5. For this analysis, we used BusyBox version 1.36.0, where SELinux and AppArmor are supported. To investigate the granularity of MAC (per-command or per-executable file), we restricted file reading on a per-command basis for each BusyBox command. We executed the cat, head, busybox cat, and busybox head commands to attempt to read the same file. As a result, only

the cat command successfully read the file, while the other commands failed. Based on this outcome, we conclude that a per-command MAC is functional in BusyBox when the cat command can read the file, and the other commands cannot. If this condition is not met, we consider the per-command MAC protocol to be non-functional in BusyBox.

Based on the results above, cat is the only command that successfully read the target file, while the other commands failed. This outcome demonstrates that the BusyBox support for SELinux and AppArmor (Nakamura, 2007) is also effective for the other two MAC systems (Smack and TOMOYO Linux).

4 Investigation on the Protection Against IoT Malware

4.1 Investigation Strategy and Methods

To enhance the comprehensiveness of our investigation into IoT malware protection, we evaluated whether MAC systems could defend against Mirai, a well-known IoT malware. Mirai compromises IoT devices via Telnet, creating botnets for Distributed Denial of Service (DDoS) attacks (Antonakakis, 2017). According to (Alrawi et al., 2021), Mirai represents more than half of all IoT malware currently in circulation. Therefore, focusing on this malware enables us to address threats that constitute a substantial portion of actual IoT malware incidents.

We investigated the protectability of MAC systems against Mirai using their default settings. Additionally, we investigated which rules need to be added to protect systems with MAC systems that cannot defend against attack simulations using default settings.

4.1.1 An Attack Experiment Simulating the Attack Method of Mirai

We conducted an attack simulation in the environment detailed in Table 10 to assess the effectiveness of MAC systems against Mirai attacks. This environment was set up on a network isolated from the Internet. We designated a single device as both attacker and victim, with the HTTP server on the attacker device simulating a program loader for Mirai (simulated malware). Meanwhile, the HTTP server on the victim device served as the target for the DoS attack originating from the IoT device.

Table 10: Attack Simulation Environment

Device	**Target IoT device**	**Attacker device and Victim server**
CPU	Same as Table 5	Intel® Core™ i7-10700K (x86_64) 3.8GHz
OS	Same as Table 5	Fedora 34
Kernel version	Same as Table 5	5.15.6-100.fc34.x86_64

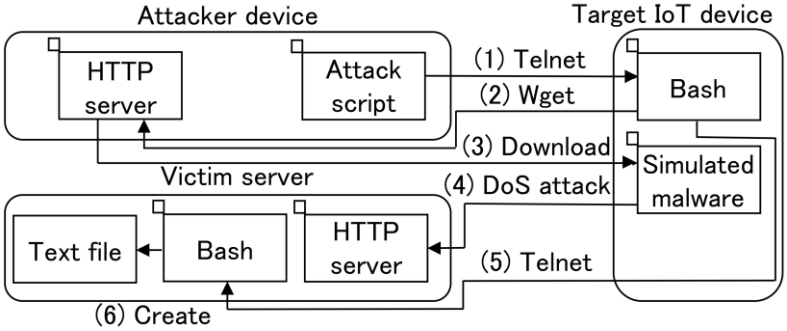

Figure 2: The Overview of Mirai Attack Simulation

Figure 2 illustrates the simulation flow based on Mirai's attack method (Gamblin, 2017). In this simulation, while Mirai spreads infection from device to device, we instead create a text file as a marker of a successful attack. This prevents the victim server from being continuously attacked. We evaluate what attacks in the flow each MAC system can thwart by the results of this simulation. The simulation proceeds as follows.

1. The attacker executes the attack script and accesses the target IoT device remotely via Telnet.
2. The attacker retrieves simulated malware from the loader using the wget command.
3. The attacker on the target IoT device downloads the simulated malware from the loader.
4. The attacker permits the execution of the simulated malware using Discretionary Access Control (DAC) and sends 10,000 HTTP packets to simulate a Denial of Service (DoS) attack on the victim server.
5. The attacker uses Telnet to gain access to the victim server from the target IoT device.
6. Finally, the attacker creates a text file to indicate the successful execution of the attack.

4.1.2 Security Policy Settings

Previous studies have not established whether MAC systems can effectively protect against Mirai using either the default security policies provided by Linux distributions or customized security policies. To address this gap, we conducted the simulation detailed in Subsection 4.1.1, employing both the default security policy of Ubuntu Server and a customized security policy to evaluate their effectiveness in mitigating these attacks.

Methods for defining access control rules are classified into allow listing and deny listing. Allow listing allows only the accesses defined in the rule sets, while all other accesses are denied. This method must cover all combinations of processes, files, and operations, and requires the verification of each covered combination. Consequently, it incurs significant costs to update security policies. Moreover, costs increase when almost all accesses are allowed in the default settings. In contrast, deny listing restricts only accesses defined in rule sets, permitting all other accesses. This method incurs fewer updating costs than allows listing because it is easier to identify which accesses to restrict. Additionally, the costs are relatively low, even if almost all accesses are allowed in the default settings, because costs are sensitive to the number of rules defined in the security policies to be updated. Because IoT devices have constraints on development costs, deny listing is more likely to be applied than allow listing from the perspective of updating costs. As a result, deny listing is more likely to yield practical results regarding development costs. Therefore, we attempted to apply deny listing when updating security policies to protect against attacks. In our investigation, we customized security policies using the following customization strategies. We investigated the rules added and evaluated the protection against the attacks for each strategy.

Customization Strategy 1: This strategy denies the use or execution of Telnet daemons, Telnet clients, wget, and chmod without specifying the access conditions. This measure addresses cases in which these commands cannot be uninstalled even though they are not used for proper system operation.

Customization Strategy 2: This strategy denies the use or execution of a Telnet client, wget, or chmod by attackers who log in remotely through the Telnet daemon. This measure addresses situations where the use of these commands by local users and remote login via Telnet is legitimate, but protection is required against attackers who use Telnet remotely to execute these commands.

Customization Strategy 3: This strategy prevents attackers who remotely log in through the Telnet daemon from modifying file permissions (chmod) of executing any files downloaded via wget. This measure is intended for situations where the use of these three commands by local users and remote login via Telnet is legitimate, but the system needs protection against the execution of downloaded malware by attackers who log in remotely using Telnet.

4.2 Investigation Results

We measured at what stage MAC systems mitigate the attack, as illustrated in Fig. 2. With default settings, none of the attack stages were successfully thwarted by Smack, TOMOYO Linux, or AppArmor. Consequently, we conducted additional experiments using customized settings based on the strategies outlined in Subsection 4.1.2. Table 11 presents the protection results from the simulations described in Section 4.1. Note that the protection against remote login also prevents both DoS attacks and infections. The success or failure of DoS and infection protection are not related each other.

Table 11: Results of the Simulated Mirai Attack

MAC System		**Default Setting**			**Customized Setting**	
	Remote Login	**DoS**	**Infection**	**Remote Login**	**DoS**	**Infection**
SELinux	Success	(Unobserved)	(Unobserved)	(Not Tested)	(Not Tested)	(Not Tested)
Smack	Failure	Failure	Failure	Success	Partially Failure	Success
TOMOYO Linux	Failure	Failure	Failure	Success	Success	Success
AppArmor	Failure	Failure	Failure	Success	Success	Success

4.2.1 Protection Based on Default Settings

SELinux successfully thwarted each stage of the attack, as it effectively blocked remote login attempts via Telnet, as illustrated in (1) of Fig. 2. In contrast, the other three MAC systems (Smack, TOMOYO Linux, and AppArmor) did not prevent any stages of the attack. Smack assigned nearly all processes and files with the same label, '_' (floor), which allowed unrestricted access between them. TOMOYO Linux operated without a default security policy and used initial settings that lacked access controls. In AppArmor, the absence of profiles for the attacker's programs meant that these programs could perform any operations, as programs without profiles were permitted to execute without restrictions.

Based on these findings, SELinux is expected to provide effective protection against actual IoT malware with its default settings. Conversely, Smack, TOMOYO Linux, and AppArmor demonstrate limited effectiveness under their default configurations and require customization to effectively mitigate attacks.

4.2.2 Protection Based on Customized Settings

We also conducted the attack simulation using customized security policies, following the customization strategies described in Subsection 4.1.2 on Smack, TOMOYO Linux, and AppArmor. Customization Strategy 1 denies the use or execution of Telnet daemons, Telnet clients, wget, and chmod without specifying access conditions. Customization Strategy 2 denies the use or execution of

the Telnet client, wget, or chmod by attackers who log in remotely via the Telnet daemon. Customization Strategy 3 prohibits attackers who log in remotely via the Telnet daemon from modifying the file permissions (chmod) of any file downloaded using wget and from executing them. Tables 12, 13, and 14 present the simulation results for the security policies following Customization Strategies 1, 2, and 3, respectively.

Customization Strategy 1: In Smack, we assigned label "^" (hat) to the executable files of applicable programs as the object label, leveraging the fact that, by default, label "_" is not allowed to access to label "^." This prevented the execution or reading of these programs and thwarted remote login (Telnet daemon), DoS (wget and chmod), and infection (Telnet client). However, it cannot prevent any attack stage if an attacker successfully logged into the system as the root user. While the execution of these attacks could be thwarted when the label "^" was assigned to the executable files of wget and the Telnet client as the subject label, these attacks were thwarted not owing to the denial of execution or reading of the executable files, but rather due to protocol errors during communication. Logs detailing the causes of program failure are not output, but we consider the cause is a lack of write permissions during communication. In addition, the execution of chmod by users with root privileges is not hindered. In Smack, CAP_MAC_ADMIN capability in Linux grants processes the permission to modify the labels of processes or files (Schaufler, 2011a). Thus, CAP_MAC_ADMIN likely allows users with root privileges to bypass MAC and other security restrictions.

Table 12: Simulation Results on Security Policies Following Customization Strategy 1

MAC System	Remote Login	DoS	Infection
Smack	Success	Partially Failure	Success
TOMOYO Linux	Success	Success	Success
AppArmor	Success	Success	Success

Table 13: Simulation Results on Security Policies Following Customization Strategy 2

MAC System	Remote Login	DoS	Infection
Smack	-	Partially Failure	Success
TOMOYO Linux	-	Success	Success
AppArmor	-	Success	Success

Table 14: Simulation Results on Security Policies Following Customization Strategy 3

MAC System	Remote Login	DoS	Infection
Smack	-	Partially Failure	-
TOMOYO Linux	-	Success	-
AppArmor	-	Success	-

TOMOYO Linux assigns domains to all executed programs which include all previously executed path names, although these domains are created with minimal permissions. Additionally, access control is disabled by default. Therefore, most permissions for these programs can be disabled by enabling access control for the applicable domains based on the execution history of the attack simulation. As a result, it prevents remote login, DoS (wget), and infection, similar to Smack, owing to the failure of operations such as obtaining environment variables of shell or file attributes. These results remained consistent even when the attacker gained root privileges. Furthermore, it thwarted the use of chmod (shown in (4) of Fig. 2), resulting in the failure of the DoS attack.

We created profiles using the aa-genprof command on the executable files of the applicable programs in AppArmor. The profiles created had minimal permissions, because only execution in the memory map and reading of the program itself are explicitly allowed. Consequently, the Telnet

daemon failed to communicate with the attacker's device and thwarted the remote login. The system also successfully protected against the DoS attack, owing to the failure to create a file for the simulated malware via wget or modify the file permissions using chmod. Furthermore, the Telnet client failed to read a necessary configuration file, and the MAC system thwarted the infection. Even when the attacker had root privileges, every attack stage was thwarted, similar to TOMOYO Linux.

Customization Strategy 2: We defined labels for each applicable program on Smack, using these labels as subjects and objects for their executable files. The new labels implemented the following rules:

1. Label "_" can access all newly defined labels.
2. All labels, except those assigned as objects for the applicable programs, can read and execute all newly defined labels.
3. Labels assigned as subjects for applicable programs can access label "_" and all newly defined labels, provided no other rules prohibit this access.
4. Labels used as subjects for Telnet daemons are prohibited from accessing labels assigned as objects for wget, chmod, and Telnet clients. (The label assigned as subject for the Telnet daemon is inherited by a bash shell, which is automatically provided to an attacker who logs in remotely.)
5. All other accesses follow the default settings.

After defining the new labels, we covered all combinations of these labels with the default labels ("_," "^", "*" (star), "?" (huh), "@" (web)) and assigned all permissions (rwxat) to each combination. When a combination of subject, object, and operation requires denial in the covered combinations, the corresponding description or an unnecessary permission was deleted. Consequently, the results obtained for DoS and infection were consistent using Customization Strategy 1.

In TOMOYO Linux, we implemented rules that permit all accesses except those prohibited on control target operations. We used groups that allowed all operations except those related to control targets for the applicable domains. For the bash domain, which is automatically provided to an attacker logging in remotely, we added two specific rules to allow all operations except the execution of the wget, chmod, and Telnet clients. We defined this group (group 1) to permit all operations, excluding those related to the control targets, with any option. We created 12 groups using various options, including file permissions and IP addresses. Although we used wildcards to reduce the number of rules, describing all path names except those for wget, chmod, and Telnet clients required 15 lines of rules owing to descriptions excluding particular path names. Additionally, there were instances where proper system operations were prevented owing to MAC activation on domains derived from bash without specific rules because domains would transition to wildcard-based domains by default. Therefore, we need to cover domains that include wildcards and configure them to prevent the activation of MAC and domain transitions. In the covered domains, MAC can be disabled by adding use_profile 0, on a single line. As a result, bash failed to execute wget, chmod, and Telnet clients, and the MAC system thwarted both the DoS and infection attacks.

In AppArmor, we first added rules that permit all operations to the Telnet daemon profile and then added rules on accesses that should be denied. AppArmor has expressions to specify the types of operations to restrict, including capability, network, and file. These expressions allow all operations of the applicable types by adding a rule-like file. We added rules with the aforementioned expressions to the 12 operations supported by the experimental environment and allowed all operations. Subsequently, we added rules of the form, such as audit deny pathname x, on the path names of

execution files of applicable programs and restricted their execution. As a result of this customization, we obtained the same results as those for TOMOYO Linux.

Customization Strategy 3: None of these MAC systems can identify files downloaded by attackers who log in remotely using Telnet via the wget directory. Therefore, we configured security policies in Smack so that wget inherits its labels as a subject for the files it creates. We customized its settings to enable the following rules based on the customization strategy:

1. Label assigned to wget as subjects can access the label "@" in addition.
2. Label "@" can access labels assigned as subjects to wget in addition.
3. Labels assigned as subjects to Telnet daemons can access labels assigned as objects for wget, chmod, and Telnet client, but lose permission to access labels assigned as subjects to wget.
4. Labels assigned as subjects to chmod lose permission to access labels assigned as subjects to wget.

Additionally, the communication destination needs to be labeled with label "@" using echo IP address @ > /sys/fs/smackfs/netlabel for the proper operation of wget. Consequently, the DoS attack was thwarted owing to the failure of chmod to read the simulated malware. The DoS attack was also thwarted because the bash failed to read or execute the malware. However, these protections become ineffective if an attacker logs in remotely as the root user, similar to results with other customization strategies.

In TOMOYO Linux, we set up a directory for wget to download files (download directory). We added rules to prohibit the bash from being used by an attacker who logs in remotely via Telnet from downloading files outside this directory using wget and executing files in it. Moreover, we added rules to prevent chmod from accessing files in the download directory. A domain corresponding to wget uses a group (Group 2) that allows all operations except file writing and creation. We added three lines of rules that limit file writing and creation are limited to the download directory. Additionally, a domain corresponding to chmod uses a group (Group 3) that allows all operations except the modification of file permissions. We added two lines of rules that allow the modification of file permissions only outside of the download directory. A domain corresponding to bash uses Group 1. We added four lines of rules that allow the execution of wget and chmod, and access to all path names, except for the executable files of these commands and inside the download directory. The download directory can be described with four lines of rules, whereas all path names outside the download directory require 16 lines of additional rules. In addition, it required 23 lines of rules to describe all path names, excluding those inside the download directory and the executable files of wget and chmod. As a result, we confirmed that wget failed to download the simulated malware outside the download directory. Furthermore, we confirmed that chmod could not modify file permissions, and the execution of the downloaded malware within the download directory was blocked. Consequently, TOMOYO Linux thwarted the DoS attack.

We added rules for AppArmor using the same strategy as in TOMOYO Linux. Similar to Customization Strategy 2, we added rules that allowed all operations at the beginning of the profile of Telnet daemon. The execution of download files in the download directory by the Telnet daemon can be restricted using the rule audit deny download_directory/** x. To add rules for wget and chmod in the profile of the Telnet daemon as child processes, we added rules with the format path name rwlkmcx. In the profiles of wget and chmod, as child processes of the Telnet daemon written using the aforementioned method, we added rules that allow all operations. In the wget, all files outside the download directory were covered by listing files in the root directory and all path names with different

characters in the same position (except for the first character /) as those in the download directory, because it cannot describe all path names other than specific ones. In chmod, the download directory can be written to with a single line using the aforementioned rule. Therefore, the download of the simulated malware failed because of the failure of mknod when the malware was downloaded outside the download directory. It also prevented chmod from modifying the file permissions of the malware downloaded in the download directory. Moreover, the bash cannot execute malware in the download directory. Consequently, it succeeded in protecting against DoS.

From these results, we confirm that TOMOYO Linux and AppArmor are effective against attacks by actual IoT malware when they have properly customized security policies. We showed that these MAC systems are effective in protecting against IoT malware even when rules are installed based on a strategy that denies the execution or use of programs exploited by attackers without specifying any conditions. It is also clarified that a strategy which prohibits the execution of programs used by attackers when their parent process is an entry-point program for attackers, such as the Telnet daemon, is effective. Moreover, we confirmed that these MAC systems can restrict operations to files created by programs executed by attackers using a defined download directory. However, there are cases in which some attacks succeed under any strategy owing to MAC bypass by root privileges in Smack. Therefore, Smack is less likely to protect systems compared to other MAC systems when an attacker gains root privileges or logs in remotely as a root user.

5 Investigation on the Cost of Updating Security Policies

Section 4 does not clarify the factors that can impact the deployment of MAC systems on IoT devices from the perspective of the costs associated with updating security policies. Thus, this section presents results that comprehensively investigate the security policies customized in Section 4 from this perspective.

5.1 Investigation Strategy

This analysis compares the security policy rules added in Section 4 and the work required for customization to investigate the cost of updating security policies for protecting systems against actual IoT malware.

We investigated the number of rules that need to be added to protect systems against actual IoT malware in each MAC system. We also examined the work required to update the security policies. Based on these aspects, we conducted a comprehensive analysis of the factors that impact the cost of updating security policies.

5.2 Investigation Methods

We confirm the number of lines and rules that indicate the combination of subject, object, and operation (rules on access permissions) are added to the security policies based on each customization strategy, using this as a quantitative criterion for the number of rules in the security policies. Lines not directly related to access permissions, such as the configuration of domain transitions, are excluded from the tabulation of rules on access permissions. In TOMOYO Linux, we tabulated rules on access permissions by extracting newly defined groups and excluding those related to operational options. We tabulated a rule written in the form of use_profile as one line of rules on access permissions, because it is used for enabling or disabling MAC. In AppArmor, we tabulated a rule written in the form of a file as one line of rule, excluding rules related to domain transitions in the profiles of child processes.

We summarize the investigation results on required works, such as the coverage of possible accesses and the availability of deny listings for each MAC system as qualitative criteria for the cost of customization. Based on these results, we discuss the potential updating costs associated with each MAC system.

5.3 Investigation Results

5.3.1 Comparison of the Number of Rules Added to Security Policies

Table 15 shows the number of lines and rules added or updated in the customized security policies. In Customization Strategy 1, Smack did not add new rules but only modified labels. TOMOYO Linux modified the value of use_profile and updated four rules. AppArmor had a relatively large number of lines added; however, its updating cost was low because the rules were automatically generated by its tool. In Customization Strategy 2, the number of added rules for AppArmor remained relatively stable, while those for Smack and TOMOYO Linux increased significantly. In Customization Strategy 3, the number of added rules increased significantly for AppArmor, but it did not exceed that of Smack. In Smack, the increment in the added rules in this strategy was only two compared to Customization Strategy 2. The number of rules added for TOMOYO Linux increased more significantly compared to Customization Strategy 2.

Table 15: The Numbers of Lines/Rules Added or Updated in Customized Security Policies

MAC System	**Customization Strategy 1**	**Customization Strategy 2**	**Customization Strategy 3**
Smack	0/0	53/53	55/55
TOMOYO Linux	4/4	100/41	177/107
AppArmor	36/4	25/15	74/51

From the results, the updating cost is generally lower for any MAC system when Customization Strategy 1 is adopted. However, Smack and TOMOYO Linux may incur higher updating costs owing to the significant increase in rules when adding them to specify fine access conditions, such as in Customization Strategies 2 and 3. In contrast, AppArmor incurs a lower updating cost compared to TOMOYO Linux, owing to fewer rules required.

5.3.2 Comparison of Required Works for Customization and Cost for Each Work

Table 16 lists the work required for security policy customization for each MAC system. Table 17 lists the availability of the deny listings. Smack can use a deny listing when applying Customization Strategy 1, because it does not require additional rules for restricting programs. However, when applying Customization Strategies 2 or 3, it must use and allow listing to define rules covering label combinations, owing to the need to add rules and the absence of syntax for denying specified accesses. Additionally, increasing the number of newly defined labels can rapidly increase the number of label combinations and rules. TOMOYO Linux can easily create security policies using deny listing when it applies Customization Strategy 1 because its tool enables MAC restrictions for each program. In contrast, when using Customization Strategies 2 or 3, it must cover and describe rules that allow all accesses on restriction targets and accesses that should not be restricted because it does not prepare syntax for denying specified accesses. When describing rules, it must be considered that TOMOYO Linux struggles with describing all path names while excluding specified ones. Additionally, domains to be restricted must be included in the security policies for any customization policy. Therefore,

attack simulation is required before customization, and the domains must be identified from the collected logs. AppArmor can easily allow all accesses using file syntax. When it applying Customization Strategies 1 or 2, it can easily create security policies based on deny listings using denial syntax. However, it lacks syntax for describing all path names while excluding specified ones on a single line, as shown in Customization Strategy 3.

Table 16: Required Works for Security Policy Customization

MAC System	**Overall Coverage of Accesses**	**Description to Allow All Accesses**	**Log Retrieval at Attack Simulation**
Smack	Required	Not Required	Not Required
TOMOYO Linux	Required	Not Required	Required
AppArmor	Not Required	Required	Not Required

Table 17: Availability of Deny Listing in the Customization of Security Policies

MAC System	**Available**	**Not Available**
Smack	Specifying Programs to Be Restricted	Customizing Rules for Programs to Be Restricted
TOMOYO Linux	Specifying Programs to Be Restricted	Customizing Rules for Programs to Be Restricted
AppArmor	Specifying Programs to Be Restricted, Customizing Rules for Programs to Be Restricted	-

In summary, we have confirmed that Smack and TOMOYO Linux can only use deny listings in limited cases. When deny listings cannot be used, they must cover and verify all possible accesses. Consequently, this results in a higher cost for creating rules. Additionally, it is necessary to describe the rules created in security policies according to the syntax of each MAC system, owing to the need to verify label combinations or the challenge of describing all path names while excluding specified path names. Hence, we conclude that the overall cost of customizing security policies for Smack and TOMOYO Linux is high, except in cases such as Customization Strategy 1. However, AppArmor can create rules using deny listings in most cases, except for the description of all path names, while excluding specified path names. Therefore, it does not require the coverage and verification of overall accesses and can explain how security policies control accesses by simply confirming the rules in the deny list. Hence, we conclude that the cost of updating security policies on AppArmor is significantly lower compared to other MAC systems. Furthermore, AppArmor has the potential to implement a syntax for describing all path names while excluding specified path names (Ubuntu Wiki, 2021). When it becomes able to describe all path names while excluding specified ones in one line, owing to this syntax, the cost of updating security policies is expected to be even lower.

6 Discussion on the Impact of Combinations of Factors

This section presents the results on the impact of the combinations of factors affecting MAC system adoption, as discussed in Sections 3, 4, and 5.

6.1 Combinations on the Applicability for IoT Devices

It is challenging to update security policies during operation due to the prevalence of read-only file systems on many IoT devices (Liu et al., 2021). Additionally, updating IoT device software is costly, as automatic updates are not yet widespread (Akiyama et al., 2023). Consequently, updating security policies for IoT devices can be expensive. Moreover, MAC systems are more prone to misconfiguration compared to other security features and may interfere with proper system operation due to the inherent complexities and difficulties associated with their configuration.

When security policies cannot be easily updated, systems that are affected by MAC systems may also face difficulties in being updated. This can lead to situations where MAC systems cannot be deployed to avoid these risks. Therefore, these issues are likely to occur in IoT devices.

6.2 Combinations on the Protection Against IoT Malware

IoT vendors frequently develop IoT devices using software components (e.g., SDKs and ODM devices) provided by chip manufacturers, as illustrated in Figure (Akiyama et al., 2023). Due to various technical or contractual constraints, there are instances where these components must be employed with their default settings (Akiyama et al., 2023). Consequently, under such constraints, only the default security settings of each MAC system can be utilized. However, as detailed in Subsection 4.2.1, MAC systems, with the exception of SELinux, are unable to effectively mitigate Mirai attacks when using their default configurations.

Hence, MAC systems can be ineffective for protection when vendors cannot customize the default security policies owing to these constraints in IoT devices. Therefore, we assume that there are cases in which IoT vendors cannot adopt them because of this problem. These cases arise unless IoT vendors have the freedom to update the software components provided by chip vendors.

6.3 Combination on the Cost to Update Security Policies

In some cases, members of the same family of IoT malware may exploit different vulnerabilities or attack methods. When security policies are customized using deny listings, the rules added to protect against specified vulnerabilities may not always thwart attacks using other vulnerabilities or attack methods. Additionally, attackers may evade MAC controls by using methods such as downloading malware using curl instead of wget when the former is omitted from access control. Hence, denying a listing requires adding rules for each vulnerability, attack method, and command used by attackers. However, IoT vendors must manage an enormous number of vulnerabilities when adding rules that focus on specific vulnerabilities. More than 20,000 CVEs have been reported each year since 2021 (cve.org, n.d.).

Therefore, the cost of updating security policies is likely to be unacceptably large, even when vendors use deny listings to cover overall vulnerabilities and implement measures against them. Additionally, this cost may become even higher when IoT vendors consider covering attack methods or commands to be restricted, verifying whether the added rules can protect systems and managing constraints on resource consumption. We presume that this problem results in more cases where MAC systems are not adopted.

7 Related Work

7.1 Applicability of MAC Systems Focusing Resources

Numerous prior studies have investigated the applicability of MAC systems derived from resource consumption. Nakamura et al., (Nakamura et al., 2015) demonstrated that the default settings of SELinux resulted in memory usage and processing delays that were deemed unsuitable for IoT devices. However, their study did not evaluate the resource consumption of other MAC systems in the context of IoT devices. Zhang et al., (Zhang et al., 2021) assessed the file operation processing delays of major MAC systems within a PC environment, but did not explore their impact on IoT devices. In contrast, our research includes a performance analysis of all prominent MAC systems within an IoT device simulation environment.

Additionally, research has explored the influence of file systems on MAC operations, which is another critical factor in determining the applicability of MAC systems. Vogel and Steinke (Vogel & Steinke, 2010) demonstrated that SELinux could be enabled on JFFS2 by patching the kernel prior to official xattr support, but their study did not address other file systems commonly used in IoT devices. In our work, we expanded on this by demonstrating the capability of per-file MAC systems to function effectively across a broader range of file systems.

7.2 Protecting Systems Using MAC Systems

Bugiel et al., (Bugiel et al., 2013) focused on defense mechanisms against attacks by enhancing SELinux for Android, demonstrating that their improved version of SELinux could defend against attacks exploiting real vulnerabilities. However, their research has two key limitations. Firstly, they did not examine the protectability of MAC systems only with default security policy settings. Secondly, their study did not address whether MAC systems could offer protection through simple security policy configurations, as opposed to more extensive extensions. In contrast, our research involved attack simulations based on the Mirai attack method, a representative IoT malware, to evaluate whether each MAC system could protect against such simulated attacks using either default or customized settings.

There exist previous studies that create security policies using logs and show which rules are effective for protection. Zhu & Gehrmann (Zhu & Gehrmann, 2021) designed LiCSec to generate profiles for AppArmor. Zhu et al., (Zhu et al., 2023) improved LiCSec for container environment and described which access denial rule contributed to protecting systems on thwarted vulnerabilities. However, these studies did not clarify how to describe equivalent rules in other MAC systems. We investigated which rules should be added to protect against Mirai on Smack, TOMOYO Linux, and AppArmor targeting protection for IoT devices.

Other studies built security policies using allow listing. Jiang et al., (Jiang et al., 2023) showed specific and detailed procedure for customizing security policies on SELinux. On the other hand, they did not evaluate which rules are effective for protection in the procedure. Dunlap et al., (Dunlap, 2022) clarified excessive or lacking permissions on the security policies of snap and flatpak by manual application tests. However, this work targeted repositories for desktop systems and its investigation results cannot be applied to IoT devices. A literature (Farrow, 2018) described how Netflix created security policies on SELinux using numerous logs. This method may not be applied easily because it assumes the use of huge logs. In this study, we provided a method to customize security policies using deny listing but without numerous logs on Smack, TOMOYO Linux, and AppArmor to protect against

Mirai on IoT devices and showed whether deny listing is effective from the aspects of protection and updating cost.

7.3 Cost to Update or Operate Security Policies on MAC Systems

Akiyama et al., (Akiyama et al., 2023) showed that verification cost is one of the challenges for the deployment of security features through a vendor interview as a previous study on the operation cost of MAC systems. However, they did not clarify how much it costs to verify security policies. Additionally, they did not sufficiently investigate what work is the cause of the large cost to customize and operate security policies. We revealed how many lines and rules are added in each MAC system and customization strategy to protect IoT devices against Mirai. Moreover, we compared required works for policy customization and the availability of deny listing.

There exists a study that designed a programmable notation for security policies to decrease the updating cost of security policies and improve explanatory of security policies. Belair et al., designed a MAC system based on LSM as SNAPPY (Belair et al., 2021) and showed that their MAC system can measure against vulnerabilities by adding a few to a dozen lines of rules. However, it needs the deployment of eBPF and their own LSM, which is not merged to the mainline kernel. In this study, we focused on deny listing as a method to both reduce costs to update security policies and improve explanatory.

8 Conclusion

In this study, we present the findings of our investigation into the factors that prevent the introduction of LSM-based secure OSs into IoT devices, as well as an analysis of the ability of LSM-based secure OSs to protect IoT devices from IoT malware attacks. In our investigation of the factors that prevent the introduction of LSM-based secure OSs into IoT devices, we conducted a comprehensive analysis of the following factors: kernel version, CPU architecture, BusyBox support, memory consumption, and file system support. Furthermore, we evaluated the influence of the combination of multiple factors on the feasibility of implementation.

Furthermore, we investigated whether it is feasible to protect against attack simulations using the Mirai attack method by examining the capabilities of the default security policies of each secure OS. In the case of secure OSs that are unable to prevent attacks using the default security policies, we presented methods for updating the security policies and the associated costs, as well as strategies for utilizing LSM-based secure OSs to provide protection for IoT devices.

As a result, we revealed that MAC systems cannot be applied in certain cases, primarily because of resource consumption, file systems, and security policy settings. Additionally, we clarified that the cost of fixing security policies is larger in Smack and TOMOYO Linux than in AppArmor, particularly in terms of the number of added rules and the required work.

Moreover, we investigated cases in which combinations of these factors could impact on the deployment of MAC systems in IoT devices. First, we identified that more cases occur where MAC systems are neither adopted to avoid the risk of improper operation nor customized, because their settings cannot be updated during operation owing to technical reasons in many IoT devices. Second, there could be cases in which MAC systems, except for SELinux, fail to effectively protect IoT devices because they rely solely on the default security policy settings, constrained by the development practices within the IoT supply chain. Finally, we consider cases where, even with a deny listing, the

cost of fixing security policies may be unreasonable in IoT devices because MAC systems need to add rules whenever new vulnerabilities are identified.

In summary, this study clarified the range of factors that can impact the adoption of MAC systems in IoT devices. Consequently, we expect IoT vendors to more easily determine which MAC systems they can adopt for developing IoT devices based on the findings of this study. We also anticipate that they will be able to take more appropriate measures when adopting MAC systems. Additionally, this study revealed that there are issues, such as the specifications of MAC systems or the software installed on IoT devices, which IoT vendors cannot resolve independently. Based on our findings, we expect the developers of MAC systems and software used in IoT devices to modify their software to facilitate the deployment of MAC systems. In particular, we expect AppArmor developers to implement syntax that allows for the description of overall path names, while excluding specified path names in a single line, leading to more cases where the fixing cost of security policies on AppArmor is acceptable.

Acknowledgments

This research was partially supported by JST, PRESTO Grant Number JPMJPR1938, Japan, and by JSPS Grants-in-Aid for Scientific Research, JP 23K24848. We also extend our gratitude to those who contributed to this work. Specifically, Ryota Yoshimoto and Shugo Shiraishi were instrumental in collecting the firmware of IoT devices analyzed in Section 3.2.4.

References

[1] Akiyama, M., Shiraishi, S., Fukumoto, A., Yoshimoto, R., Shioji, E., & Yamauchi, T. (2023). Seeing is not always believing: Insights on IoT manufacturing from firmware composition analysis and vendor survey. *Computers & Security*, *133*, 103389. https://doi.org/10.1016/j.cose.2023.103389

[2] Alrawi, O., Lever, C., Valakuzhy, K., Snow, K., Monrose, F., & Antonakakis, M. (2021). The Circle of life: A {large-scale} study of the {IoT} malware lifecycle. *In 30th USENIX Security Symposium (USENIX Security 21)*, 3505-3522.

[3] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., ... & Zhou, Y. (2017). Understanding the mirai botnet. *In 26th USENIX security symposium (USENIX Security 17)*, 1093-1110.

[4] AppArmor contributors. (n.d.). AppArmor. 2023. https://apparmor.net/

[5] Aswathy, R.H., Srithar, S., Roslin Dayana, K., Padmavathi., & Suresh, P. (2023). MIAS: An IoT based Multiphase Identity Authentication Server for Enabling Secure Communication. *Journal of Internet Services and Information Security*, *13*(3), 114-126.

[6] Belair, M., Laniepce, S., & Menaud, M.J. (2021). SNAPPY: Programmable Kernel-Level Policies for Containers. *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*, 1636–1645.

[7] Bobir, A.O., Askariy, M., Otabek, Y.Y., Nodir, R.K., Rakhima, A., Zukhra, Z.Y., Sherzod, A.A. (2024). Utilizing Deep Learning and the Internet of Things to Monitor the Health of Aquatic Ecosystems to Conserve Biodiversity. *Natural and Engineering Sciences*, *9*(1), 72-83.

[8] Bugiel, S., Heuser, S., & Sadeghi A. R. (2013). Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. *In 22nd USENIX Security Symposium (USENIX Security 13)*, 131–146.

[9] Corbet, J. (2008). More stuff for 2.6.25, LWN.net. https://lwn.net/Articles/267849/

[10] cve.org. (n.d.). Metrics | CVE. 2024. https://www.cve.org/About/Metrics

[11] Dunlap, T., Enck, W., & Reaves, B. (2022). A Study of Application Sandbox Policies in Linux. *In Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, 19-30.
[12] Eclipse. (2019). IoT Developer Survey 2019 Results. https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2019.pdf
[13] Farrow, R. (2018). Interview with Travis McPeak. *Login Usenix Mag.*, *43*(2), 10–12.
[14] Gamblin, J. (2017). jgamblin/Mirai-Source-Code: Leaked Mirai Source Code for Research/IoC Development Purposes. https://github.com/jgamblin/Mirai-Source-Code
[15] Hossain, M. M., Fotouhi, M., & Hasan, R. (2015) Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things. *In IEEE World Congress on Services*, 21–28.
[16] Iman, M.B., Qusay, A.A., Inass, S.H., & Refed, A.J. (2023). Mobile-computer Vision Model with Deep Learning for Testing Classification and Status of Flowers Images by using IoTs Devices. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *14*(1), 82-94.
[17] Jiang, C., Wu, S., Wu, G., Yang, C., Cai, L., & Zhong, F. (2023). Application Research of Security Policy in the Linux Operating System. *Proceedings of the 2022 4th International Conference on Robotics, Intelligent Control and Artificial Intelligence (RICAI '22)*, 638–641.
[18] Liu, K., Yang, M., Ling, Z., Yan, H., Zhang, Y., Fu, X., & Zhao, W. (2021) On Manually Reverse Engineering Communication Protocols of Linux-based IoT Systems. *IEEE Internet of Things Journal*, *8*(8), 6815-6827.
[19] Miki, M., Yamauchi, T., & Kobayashi, S. (2023) Evaluation of Effectiveness of MAC Systems Based on LSM for Protecting IoT Devices. *In Proceedings of 11th International Symposium on Computing and Networking (CANDAR2023)*, 161–167.
[20] Nakamura, Y. (2006). Specification of Simplified Policy Description Language (SPDL) Ver 2.1. https://seedit.sourceforge.net/doc/2.1/spdl_spec.pdf
[21] Nakamura, Y. (2007). Domain assignment support for SELinux/AppArmor/LIDS. http://lists.busybox.net/pipermail/busybox/2007-August/062635.html
[22] Nakamura, Y., & Sameshima, Y. (2008). SELinux for Consumer Electronics Devices. *In Ottawa Linux Symposium*, 125–134.
[23] Nakamura, Y., Sameshima, Y., & Yamauchi, T. (2015). Reducing Resource Consumption of SELinux for Embedded Systems with Contributions to Open-Source Ecosystems. *Journal of Information Processing*, *23*(5), 664–672.
[24] National Security Agency. (n.d.). Home | Open Source @ NSA. 2023. https://code.nsa.gov/
[25] OpenWrt Project. (2018). [OpenWrt Wiki] OpenWrt File System Hierarchy / Memory Usage. https://openwrt.org/docs/techref/file_system
[26] OpenWrt Project. (2023). [OpenWrt Wiki] Table of Hardware: Full details. https://openwrt.org/toh/views/toh_extended_all
[27] Red Hat. (2023). SELinux as a security pillar of an operating system - Real-world benefits and examples. https://access.redhat.com/articles/6964380
[28] Salve, V. (2020). Inside the Linux security module (LSM). *In Embedded Linux Conference North America 2020 (ELC2020)*.
[29] Schaufler, C. (2011a). The Smack Project - Description from the Linux source tree. http://schaufler-ca.com/description_from_the_linux_source_tree
[30] Schaufler, C. (2011b). The Smack Project - Home. http://schaufler-ca.com
[31] SELinux Wiki contributors. (2017). SELinux Wiki. http://selinuxproject.org/page/Main_Page
[32] Surendar, A., Saravanakumar, V., Sindhu, S., & Arvinth, N. (2024). A Bibliometric Study of Publication - Citations in a Range of Journal Articles. *Indian Journal of Information Sources and Services*, *14*(2), 97–103. https://doi.org/10.51983/ijiss-2024.14.2.14
[33] TOMOYO Linux. (n.d.a). (2024). TOMOYO Linux 2.6.x: The Official Guide: Chapter 2. https://tomoyo.sourceforge.net/2.6/chapter-2.html.en

[34] TOMOYO Linux. (n.d.b). (2024). TOMOYO Linux Documentation. https://tomoyo.sourceforge.net/documentation.html.en
[35] TOMOYO Linux. (n.d.c). (2024). TOMOYO Linux Home Page. https://tomoyo.sourceforge.net/index.html.en
[36] Ubuntu Wiki. (2021). AppArmor - Ubuntu Wiki. https://wiki.ubuntu.com/AppArmor
[37] Vogel, B., & Steinke, B. (2010). Using selinux security enforcement in linux-based embedded devices. In *1st International ICST Conference on Mobile Wireless Middleware, Operating Systems and Applications*. http://dx.doi.org/10.4108/ICST.MOBILWARE2008.2927.
[38] Zhang, W., Liu, P., & Jaeger, T. (2021). Analyzing the Overhead of File Protection by Linux Security Modules. *In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*, 393–406.
[39] Zhu, H., & Gehrmann, C. (2021). Lic-Sec: an enhanced AppArmor Docker security profile generator. *Journal of Information Security and Applications*, *61*, 102924. https://doi.org/10.1016/j.jisa.2021.102924.
[40] Zhu, H., Gehrmann, C., & Roth, P. (2023). Access security policy generation for containers as a cloud service. *SN Computer Science*, *4*(6), 748. https://doi.org/10.1007/s42979-023-02186-1.

Authors Biography

Masato Miki, received his B.E., and M.E. degrees from Okayama University, Japan in 2022, 2024, respectively. His research interests include computer security.

Toshihiro Yamauchi, received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu University, Japan, in 1998, 2000, and 2002, respectively. In 2002, he became a Research Associate with the Faculty of Information Science and Electrical Engineering, Kyushu University. In 2005, he became an Associate Professor with the Graduate School of Natural Science and Technology, Okayama University. Since 2021, he has been serving as a Professor with Okayama University. His research interests include operating systems and computer security. He is a member of IPSJ, IEICE, ACM, IEEE, and USENIX.

Satoru Kobayashi, received the Ph.D. degree in information science and technology from the University of Tokyo, Japan, in 2018. Since 2022, he has been an Assistant Professor at Faculty of Environmental, Life, Natural Science and Technology, Okayama University. His research interests include network management and data mining. He is a member of IPSJ, IEICE, ACM, and IEEE.