National College of Ireland

Evaluating LSM-Based MAC Policies in Kernel Space for IoT Device Protection

MSc Research Project
Cloud Computing

Alexander Mamani
Student ID: 23329823

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

Evaluating LSM-Based MAC Policies in Kernel Space for IoT Device Protection

Alexander Mamani
23329823

Abstract

Environments with resource constraints, such as IoT devices, face a significant footprint challenge in implementing security policies. Previous research concluded that traditional Mandatory Access Control (MAC) systems like SELinux and AppArmor introduced considerable overhead, since these are executed in user space. Therefore, there is a need to optimize such security policy enforcement. Solutions like tuning embedded MAC into IoT devices were also tackled by other research, but complex modifications were involved in the kernel, and interoperability across different IoT devices was not guaranteed. To address such need, the effectiveness of enforcing security policies in kernel space was evaluated by using the Extended Berkeley Packet Filters (eBPF), while minimizing both the resource footprint and modifications in the kernel. Experiments based on the Mirai attack were carried out to assess the effectiveness of these security policies under real-world conditions against IoT malware, successfully preventing Remote Login, DoS, and Infection of other IoT devices. Results showed that LSM BPF has an acceptable memory consumption of only 5.2% and better adoption on devices with only 128 MB of RAM compared to AppArmor. LSM BPF also enforced a per-file control over common file systems used in IoT, demonstrating no dependency on extended attributes like SELinux or Smack. Finally, analysis of OpenWrt firmware demonstrated a rapid adoption of kernel version 5.7+ (required for LSM BPF) in IoT devices from 2022 to 2024.

1 Introduction

The fast adoption of IoT in multiple sectors have led to companies to deploy thousands of devices under limited hardware and software constraints (Illakya et al., 2024). These devices run lightweight Linux-based distributions and these are expected to run continuously for days in hostile environments wherein saving resources like power, memory, CPU consumption, and so on are crucial. So, their limited computational power and memory capacity makes them unable to support traditional security mechanisms like in conventional cloud services like AWS EC2 instances, or Azure virtual servers. As a result, multiple types of malware have been released to take control of such devices, the most famous one was Mirai, being able to infect 500,00 devices at its peak (De Donno et al., 2018).

IoT security concern was also addressed by Miki et al. (2023) and Miki et al. (2024), evaluating multiple traditional MAC Systems ,and finding that most IoT devices do not implement such MAC Systems due to its considerable overhead. Nakamura et al. (2015) formalized the concept of Embedded SELinux, achieving to reduce the resource consumption, but complex modification in the kernel was required. However, with the Linux kernel 5.7 release, LSM functionality was extended to allow attaching eBPF programs in kernel space (Song and Li, 2024).

Since Linux 5.7 enables eBPF programs to attach to LSM hooks in kernel space, this work addresses the question: *Can security policies be enforced effectively in kernel space to protect against IoT malware while maintaining high availability, low overhead in IoT devices?* To answer it,

this work evaluated the effectiveness of LSM-based security policies enforcement by implementing small eBPF programs in kernel space and attaching them into LSM hooks; It supports policy definition and modification without requiring system restart or kernel compilation.

The objectives of this research are to evaluate the applicability and performance of LSM BPF in IoT environments by comparing standard metrics like memory consumption and processing delay; examine kernel and file system adoption used in IoT firmware like OpenWrt. Finally, evaluate the effectiveness against IoT malware like Mirai. This research contributes to the computer science field by demonstrating that LSM BPF is a viable alternative for enforcing security policies in IoT devices.

This research is organised as follows: Section 2 discusses related work and MAC state; Section 3 describes the methodology to carry out and assess the implementation; Section 4 describes the architecture; Section 5 shows the implementation of the proposed approach; Section 6 presents experimental setup and results; Section 7 concludes the work and outlines future research directions.

2 Related Work

Linux Security Modules have been the subject of numerous studies aiming to harden security on servers, personal computers, and others, but there are limited studies in the IoT field; owing to high-resource servers are not affected by the MAC system footprint, as a limited IoT Device is. To better understand the context of this study, To understand the context of this study, existing works on LSM-based security policy enforcement were reviewed, focusing on traditional MACs, embedded systems in IoT, protection against malware, and eBPF-based approaches.

2.1 Traditional MAC Systems for IoT

Miki et al. (2023) performed an evaluation wherein compares four MAC systems (SELinux, AppArmor, TOMOYO, and Smack) on IoT environments. Such evaluation was focused on evaluating default policy effectiveness, malware resilience specifically against Mirai (Alrawi et al., 2021), system compatibility, and resource consumption. While SELinux offered a strong security out-of-the-box by managing to block Mirai attacks, it also introduced the highest resource overhead, reaching up to 85% of the memory in test scenarios, making it unsuitable for limited IoT hardware. AppArmor and TOMOYO show better efficiency, but require highly customised policy development to achieve the same protection. Smack failed because it was unable to prevent malware propagation due to limitations in its label and capability model.

Guo et al. (2013) proposed MPdroid, a hybrid solution combining Smack (kernel-level MAC) with role-based access control in the Android framework. MPdroid reached out to block malware behaviours (unauthorised SMS and data access) through dynamic post-installation role assignment. However, MPdroid has a strong dependency on the platform, and it was tightly integrated with Android's internal system services, making it unsuitable for general-purpose or embedded Linux distributions.

These findings are consistent with earlier efforts by Nakamura and Sameshima (2008) and Nakamura et al. (2015), reducing SELinux's memory consumption using SEEdit and SPDL. achieving to reduction of up to 90%, SELinux still required several megabytes of RAM and significant CPU usage for policy evaluation; unsuitable for typical IoT devices running minimal Linux distributions like BusyBox. This understanding of the past reinforces that high resource overhead is still a critical limitation, despite years of optimisation.

Despite previous research, no solution provides low-overhead, dynamic, and portable MAC enforcement suitable for kernel-based IoT platforms. This shows a study gap in security enforcement within the kernel space.

2.2 Embedded Optimization

SELinux's lack of suitability for consumer electronics (CE) and embedded devices was addressed by Nakamura and Sameshima (2008) through extensive kernel optimisations, and by the creation of SEEdit, a simplified policy editor. The Simplified Policy Description Language (SPDL) was used by SEEdit to allow more concise and readable policy definitions compared to traditional SELinux syntax. SELinux's memory footprint was reduced up to 90% from (5.3 MB to 465 KB), and the policy file size was decreased from 2.3 MB to 211 KB as a result of the tuning. This allowed SELinux to run on hardware with as little as 64 MB of RAM and 32 MB of flash, which was common in CE and IoT devices at that time.

Building on this, the concept of Embedded SELinux was formalised by Nakamura et al. (2015). The kernel and policy were further tuned, and patches were contributed to BusyBox, libselinux, and SELinux userland utilities to improve compatibility with minimal Linux distributions. Improvements in table allocation based on policy size was introduced, managing to reduce runtime overhead and enhance scalability. Reduction in syscall latency of up to 50% was shown through benchmarking, and policy enforcement was sustained without performance degradation under typical embedded workloads.

Despite the optimisation achieved, the feasibility of deploying SELinux on embedded devices is too complex and does not guarantee interoperability due to kernel tuning. This limitation leaves a gap in the need for an interoperable MAC system capable of running among Linux-based IoT devices.

2.3 Protection Against IoT Malware

Despite the increasing deployment of traditional MAC systems like SELinux, AppArmor, TOMOYO, and others in general-purpose computing, IoT devices remain unprotected against modern IoT malware. Alrawi et al. (2021) and De Donno et al. (2018) have reveled that 1) A large of IoT firmware lacks integrated security policies; 2) Malware based on infection such as Mirai, Hajime, and Reaper exploit trivial misconfiguration and weak or absent access controls; 3) specific IoT behaviors such as minimal update cycles, default credentials, and open telnet/SSH ports render traditional host-based MAC enforcement ineffective or absent altogether.

Although Miki et al. (2024) tried to evaluate the classic MACs in the context of IoT malware, each of those did not succeed in implementing real-time mitigation in IoT ecosystems without suffering resource degradation. These results support the necessity of lightweight, dynamic, and changeable MAC policies on the basis of the limitations of embedded systems.

2.4 Current State of eBPF-based MAC for IoT

Findlay et al. (2020) introduced bpfbox, a mature eBPF-based MAC prototype, it is designed to enforce fine-grained access control policies directly within the Linux kernel. Rather than using a general-purpose policy language, bpfbox adopts a rule-based format to express per-executable permissions for file system and network operations, which implements a capability-based model, granting the minimal set of operations such as read, write, exec, bind, or connect. Additionally, bpfbox supports advanced semantics like #[taint] and #[transition] to manage limited forms of privilege propagation and dynamic execution control. These policies are compiled into eBPF bytecode and attached to LSM hooks like file open, inode permission, bprm check security, providing in-kernel runtime enforcement with minimal performance overhead. Likewise, Brimhall et al. (2023) presented BPF/KRSI , a more context-sensitive logic than SELinux, which achieved better performance in key microbenchmarks based on the same programmable LSM infrastructure.

SNAPPY, in a similar manner, proposes a programmable kernel-level MAC mechanism tailored for Linux containers (Bélair et al., 2021). It allows defining policies, which are compiled into eBPF, and loaded into the kernel, in a syntax accessible to the user space. SNAPPY links

policies with containers based on process metadata and cgroup identifiers, enabling container-aware security enforcement. It monitors key system operations through LSM hooks such as task creation, file open, and bprm check security, allowing fine-grained monitoring and control of container behaviour. Despite SNAPPY being demonstrated to offer better flexibility and lower overhead than traditional mechanisms like SELinux or AppArmor, the system is explicitly targeted at containerised workloads, making it not suitable for embedded or IoT scenarios.

While these systems show improvements, there is still a gap in how they can be used in limited IoT environments like OpenWRT or BusyBox. Key resource limitations of IoT devices, such as CPU, memory, and the challenges of policy deployment, were not considered. Most importantly, these systems were not tested against real IoT malware or the types of attacks that happen on compromised IoT devices, clearly showing a gap in testing and adaptation for IoT use.

Table 1: MAC for IoT Comparison Table

Research	Tool	Traditional MACs							IoT Malware Protection					eBPF MACs					
		IoT-Oriented	Live Updates	Policy Granularity	Overhead Measured	xattr/FS Constraints	Kernel Changes Needed	Portability	AppArmor	SELinux	TOMOYO	Real Malware Used	MAC Gaps	Defense Effectiveness	Runtime Protection	Mitigation Tested	LSM BPF Used	IoT-Oriented	Overhead Measured	Policy Customization	Hook Selection	Ease of Deployment
Nakamura and Sameshima (2008)†		●	-	◐	●	◐	●	●	-	●	-	-	◐	◐	◐	-	-	-	-	-	-	-
Nakamura et al. (2015)†		●	-	◐	●	●	●	●	-	●	-	-	◐	◐	-	-	-	-	-	-	-	-
Bugiel et al. (2013)†*	FlaskDroid	-	◐	●	●	-	●	◐	-	◐	◐	-	◐	◐	◐	-	-	-	-	-	-	-
Zhu and Gehrmann (2021)†*	Lic-Sec	-	◐	●	●	-	-	●	●	-	-	-	●	●	●	◐	-	-	-	-	-	-
Loukidis-Andreou et al. (2018)†*	Doc-Sec	-	●	●	●	-	-	◐	●	-	-	-	◐	●	●	●	-	-	-	-	-	-
Guo et al. (2013)†*	MPdroid	◐	●	●	●	●	●	-	-	◐	-	●	◐	●	●	●	-	-	-	-	-	-
Miki et al. (2023) †		●	-	◐	●	●	-	◐	●	●	●	●	●	●	●	◐	●	-	-	-	-	-	-
Miki et al. (2024)†		●	-	◐	●	●	-	●	●	●	●	◐	◐	●	●	●	●	-	-	-	-	-	-
Bessler et al. (2024)†		●	-	●	◐	●	◐	◐	-	◐	●	-	●	●	●	●	-	-	-	-	-	-	-
Zhang et al. (2021)†		-	-	◐	●	◐	-	-	-	●	●	◐	-	-	-	-	-	-	-	-	-	-	-
Akiyama et al. (2023)†		●	-	-	-	◐	-	-	◐	◐	◐	-	◐	-	-	-	-	-	-	-	-	-	-
Ko et al. (2019)†		●	●	◐	●	◐	◐	-	◐	●	-	-	-	-	-	-	-	-	-	-	-	-	-
Findlay et al. (2020)†*	Bpfbox	◐	-	-	●	-	-	-	◐	◐	-	-	◐	-	-	-	●	-	●	◐	●	◐
Bélair et al. (2021)†*	SNAPPY	-	-	-	-	-	-	-	-	-	-	-	◐	-	◐	-	●	-	●	●	●	◐
Brimhall et al. (2023)†*	BPF/KRSI	-	-	●	●	-	◐	●	◐	-	●	-	◐	◐	◐	-	●	●	-	●	●	◐
Alrawi et al. (2021)†		●	-	-	-	-	-	-	-	-	-	-	●	◐	-	●	●	-	-	-	-	-	-
De Donno et al. (2018)†		●	-	-	-	-	-	-	-	-	-	-	●	◐	-	◐	◐	-	-	-	-	-	-

● = provides property; ◐ = partially provides property; - = does not provide property;
†has academic publication; *end-user tool available

2.5 Research Gap

This study addressed a key gap in the enforcement of security policies in kernel space instead of traditional user space to reduce overhead in IoT devices. While embedded solutions offer optimised approaches, these require extensive modifications in the kernel and do not guarantee interoperability across other IoT devices. Previous eBPF-based approaches have shown promise, but these were neither evaluated in IoT environments nor tested against IoT malware. This research evaluated the effectiveness of eBPF technology in IoT environments by attaching small programs to LSM hooks in the kernel space, instead of modifying and rebuilding the kernel source,

while keeping a low overhead compared to traditional MAC System and protecting against IoT malware.

3 Methodology

In order to evaluate the effectiveness of security policies based on LSM BPF for protecting IoT devices and make it comparable with other traditional MAC systems, two main strategies were defined: evaluating their applicability to IoT devices and their protection against real malware. Both strategies were used by Miki et al. (2024) and Miki et al. (2023), giving a solid benchmark discussed in section 6.

First, individual factors were evaluated to measure their impact on the system, and then a combination of both was assessed for a more comprehensive and realistic evaluation. To make the results precise and comparable with Miki et al. (2024), hardware with the same characteristics was used in both applicability and protection against IoT malware strategy.

Additionally, in protection against IoT malware strategy, a simulation of a real-world Mirai attack was performed in a controlled environment; therefore, a group of virtual machines were used. It is worth mentioning that these virtual machines are a complement to the second strategy and do not influence the results due to LSM BPF security policies were not executed in them.

3.1 Strategy for Investigating Applicability

In this section, key metrics for the applicability of LSM BPF in constrained devices are explained and justified for the evaluation, such as: the memory consumption involved in using LSM BPF; the most common file systems used and how to assess LSM BPF effectiveness on them; the cost of using LSM BPF and its impact on basic operating system functions; and kernel compatibility.

3.1.1 Memory Consumption

To measure the memory consumption of LSM BPF in IoT devices, a snapshot of the entire operating system was taken ten times in an interval of one minute after login by the command `free` to measure the memory consumption. Then the average of total measurements was calculated, and finally, the memory consumption of the LSM BPF was calculated by taking the difference between the average memory usage when the LSM BPF was loaded into the kernel and when it was not loaded.

Then, four criteria were defined to determine whether the proposed security policies can be deployed in IoT devices using OpenWrt firmware in Table 2. Such criteria were:

(1) C1: Whether the proposed security policies' memory usage is less than 1% of the onboard memory capacity.

(2) C2: Whether the proposed security policies' memory usage is less than 2% of the onboard memory capacity.

(3) C3: Whether the sum of the proposed security policies' memory usage and the OpenWrt kernel memory usage (3828 KB) is less than the onboard memory capacity.

(4) C4: Whether the proposed security policies' memory usage of an entire system is less than the onboard memory capacity (limited to devices with 64MB or more memory capacity)

Table 2: Memory capacity of IoT devices supported by OpenWrt as available devices on OpenWrt Project (2024)

Memory Capacity	**Number**	**Memory Capacity**	**Number**
16MB	1	512MB	109
32MB	34	1024MB	97
64MB	183	2048MB	33
128MB	203	4096MB	34
256MB	158	More than 4GB	19
		Total	**871**

3.1.2 File Systems

The effectiveness of LSM BPF was evaluated for multiple types of IoT file systems. Table 3 shows the most common file systems for IoT. Some of them support `xattr`, a feature relevant for some traditional MAC systems because it helps them enforce file-reading restrictions. This feature was also relevant for the LSM BPF availability comparison. The Read/Write permission is also listed due to some file systems only support read-only. Liu et al. (2021) analysed multiple IoT firmware and reported that Cramfs, romfs, JFFS2 and SquashFS are the most used; on the other hand, Alrawi et al. (2021) reported that `tmpfs` is the most targeted file system. The last column shows whether LSM BPF is supported in such file systems.

Table 3: Object File Systems of investigation

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

The granularity of file-reading restrictions was also assessed on each file system by creating a restricted and unrestricted file, then attempting to read them. This technique was also used by Miki et al. (2023) and Miki et al. (2024), and it allowed for measuring whether a traditional MAC applies a restriction based on `per-file` or `per-FS`(per file system). This approach was also used to determine whether LSM BPF could distinguish between `per-file` and `per-FS` restrictions.

3.1.3 Processing Delays

To measure the impact of having LSM BPF policies loaded in kernel hooks, processing delays of basic operations like stat, `open/close`, `read/write`, `create`, and `delete` files were evaluated in a Raspberry Pi environment by following the following steps:

(1) LMbench was configured with specific memory usage and a CPU frequency.

(2) LMbench was ran twenty times.

(3) The processing delay of the LSM BPF programs was calculated from the difference between the average of the measurements and that of the case in which all the LSM BPF programs were unloaded.

3.1.4 Kernel Version

To evaluate kernel-level support for traditional MAC systems and LSM BPF in IoT environments, the mainline kernel versions in which these mechanisms were merged were first identified and are listed in Table 4. These versions were compared with the kernel versions present in OpenWrt firmware images of IoT devices. The firmware was obtained from a dataset collected over a three-year period from 2022–2024 (OpenWrt Project, 2024). This comparison was used to estimate the support rate of each mechanism across real-world deployments

Table 4: The Mainline Kernel Version Merged with traditional MAC System and LSM BPF

MAC System	**Kernel Version**
LSM BPF	5.7
SELinux	2.6.0
Smack	2.6.25
TOMOYO Linux	2.6.30
AppArmor	2.6.36

3.2 Strategy for Investigating Protection

An attack simulation was conducted in an isolated environment to investigate whether LSM BPF systems can protect against Mirai (Gamblin, 2016). The environment was isolated from the internet. Two roles, attacker device and victim server, were assigned to a single device. The HTTP server on the attacker device was configured to act as the loader of the program simulating Mirai (simulated malware). The HTTP server on the victim server was set to function as the target in a DoS attack from the simulated IoT device.

3.2.1 Protection with Default or Customized Settings

Modern Linux server distributions have eBPF enabled by default, primarily because service managers like `systemd` use eBPF for auditing, networking, and firewall functionalities. LSM BPF is also enabled by default in some of these systems. Therefore, a simulation with default and customised security policies was conducted against Mirai to compare their effectiveness.

4 Design Specification

The design specifications of the proposed security policy enforcement mechanism based on LSM BPF are defined in this section. It begins by outlining the Mirai malware workflow to provide a better understanding of its behaviour relevant to implementing security policies. Based on this analysis, a tailored architecture was established, with making focus on real use cases aimed to protect IoT devices as part of the methodology discussed in section 3.2. An isolated-controlled evaluation of it is discussed in section 5. Such security policies include: 1) preventing Denial-of-Service (DoS) attacks by restricting file permissions and execution related to malware downloaded by `wget` program; 2) blocking device infection by denying the use of `Telnet client`; 3) mitigating remote access by preventing connections to `Telnet service`. Critical LSM Hooks are also identified, which are used to attach LSM BPF programs that enforce runtime access control directly in the kernel.

4.1 Mirai Attack

A deeper understanding of the Mirai attack was tackled by De Donno et al. (2018), wherein the main components such as the Command-and-Control (CNC) server, Mirai bot, loader server, and

reporting server are discussed. Figure 1 illustrates the overall flow of the attack. An explanation in detail of its flow is the following:

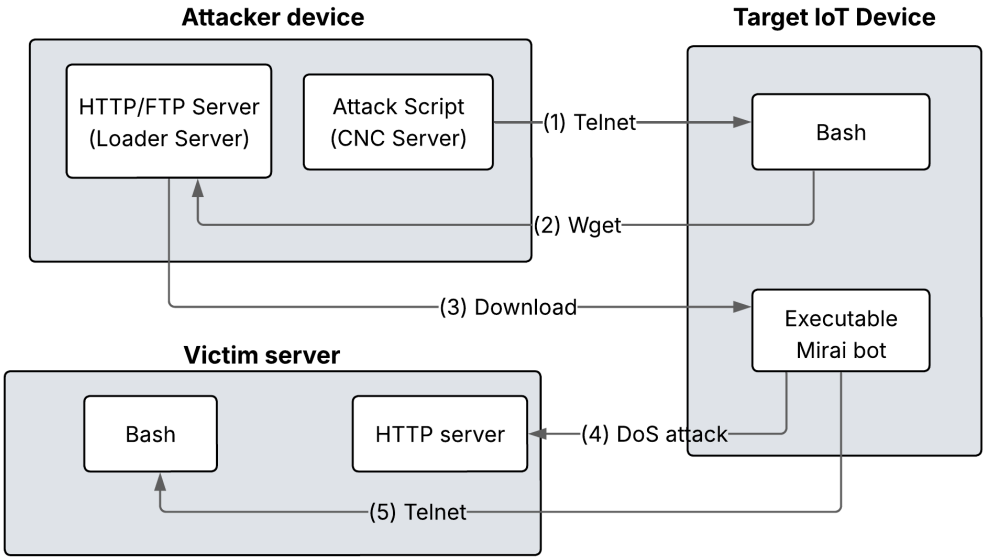

Figure 1: The flow of attack simulation of Mirai

(1) An attack script is run by the attacker device (CNC server), which remotely logs into the target IoT device using the Telnet service.

(2) Commands are issued by the attacker to get the malware from the loader server.

(3) The executable Mirai bot is downloaded from an HTTP/FTP server (the loader) onto the target IoT device.

(4) A DoS attack is carried out using HTTP packets by the executable Mirai bot against the victim server; this attack is controlled by the CNC server through the Mirai bot.

(5) An infiltration is also performed by the executable Mirai bot as part of the infection process, targeting other IoT devices via the Telnet client.

(6) The victim server is infected and becomes a new Mirai bot within the botnet.

4.2 Security Policies Enforcement Architecture

The proposed architecture used to enforce security policies within the kernel is described in this section. LSM BPF is considered an extension of the LSM framework, and it was merged into the kernel version 5.7 (The Linux Kernel Documentation, 2024). Unlike traditional MAC, these are executed in kernel space as part of the LSM hooks. Such LSM Hooks are points wherein logic is applied and are triggered by system calls interacting with kernel resources such as files, sockets, or binaries. Figure 2 illustrates such integration.

The process starts with the definition of a security policy, which is written in restricted C, and then it is compiled into eBPF bytecode. Before to be loaded into LSM hooks, the bytecode passes through the eBPF verifier and Just-In-Time(JIT) compiler, ensuring safe memory access, removing bounded loops, and others. Once verified, the security policy is attached to specific LSM hooks like `file_open`, `bprm_check_security`, `inode_setattr`, and others (Linux Kernel Project, 2023), depending on the purpose of the policy. At this point, it can inspect kernel state and use

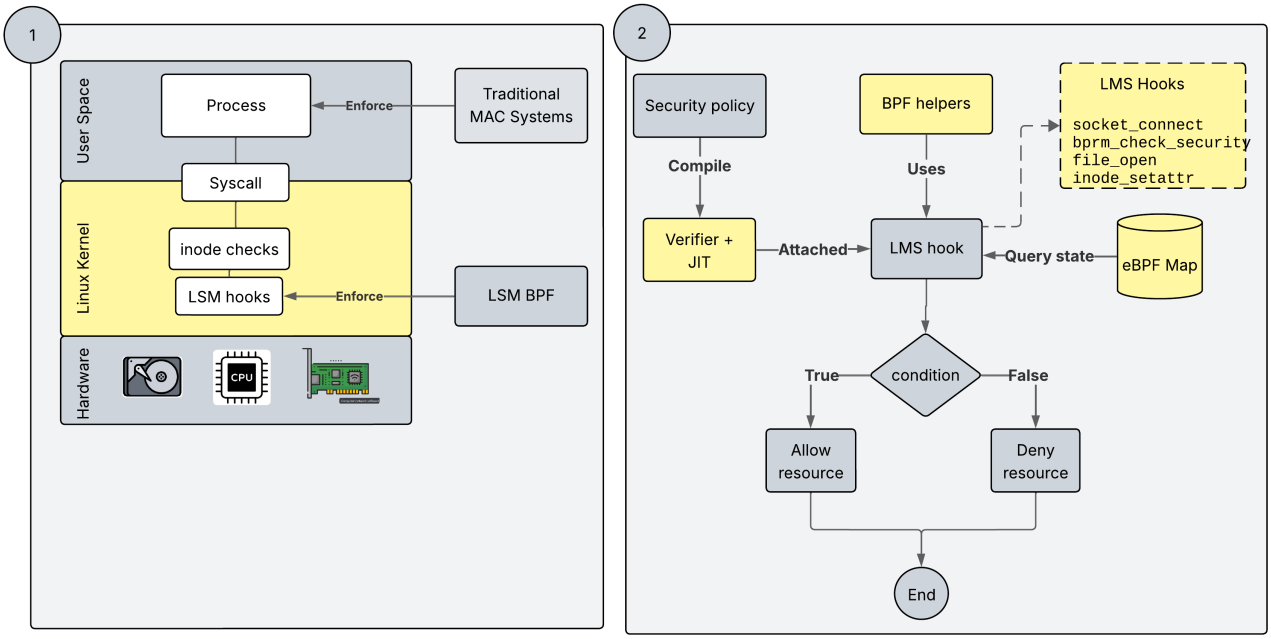

Figure 2: Illustration where Security Policies are enforced and Security Policies Architecture

safe function calls BPF helpers to interact with kernel objects or external policy state stored in eBPF maps. Based on this context, it evaluates a conditional expression which determines whether allowed or denied to the resource. This approach allows enforcement of dynamic security policies like blocking execution of files created by specific processes or preventing permission changes on marked files. The second part of Figure 2 further illustrates the security policies creation.

To reduce the memory footprint due to the attachment of security policies, certain considerations were addressed by Liu et al. (2024), thus, such considerations were taken into account during the architecture design, like:

(1) Multiple hooks were gathered in the same eBPF program since it saves memory and improves performance.

(2) The number and size of BPF maps were limited to reduce memory consumption; only relevant enforcement data, like inode identifiers, were stored. Setting fixed-size maps helped to avoid the growth of the memory.

(3) Usage of paths was avoided to reduce the overhead caused by the BPF MAP `TYPE_HASH` type. Instead, inode numbers were stored.

(4) Additional logics and stack allocation were avoided to enhance the verification success and reduce CPU consumption.

4.3 Security Policies Against Mirai

As part of the methodology described in Section 3.2, the security policy enforcement architecture was evaluated through the design of specific policies aimed at mitigating the Mirai attack. Three critical access points were identified wherein security policies enforcement can be applied to prevent malicious actions from Mirai malware, such as gaining access, propagating infection, or launching DoS attacks. For each access point, the corresponding LSM hooks were identified, and the enforcement workflow is described in detail.

4.3.1 Prevent DoS Attack

After a thorough analysis of the Mirai attack in Section 4.1, wget was identified as a key access point for the attacker to perform a DoS attack, as it is the tool used to download the Mirai botnet, modify its permissions, execute it, and launch thousands of requests to the victim server.

The action sequence taken by the proposed security policies architecture to mitigate Mirai DoS attack are illustrated in Figure 3, In this diagram, interactions between architecture components and key access points like bash, wget, LSM hooks, the Mirai malware, and eBPF maps are modeled. Such policies are divided in four main policy blocks

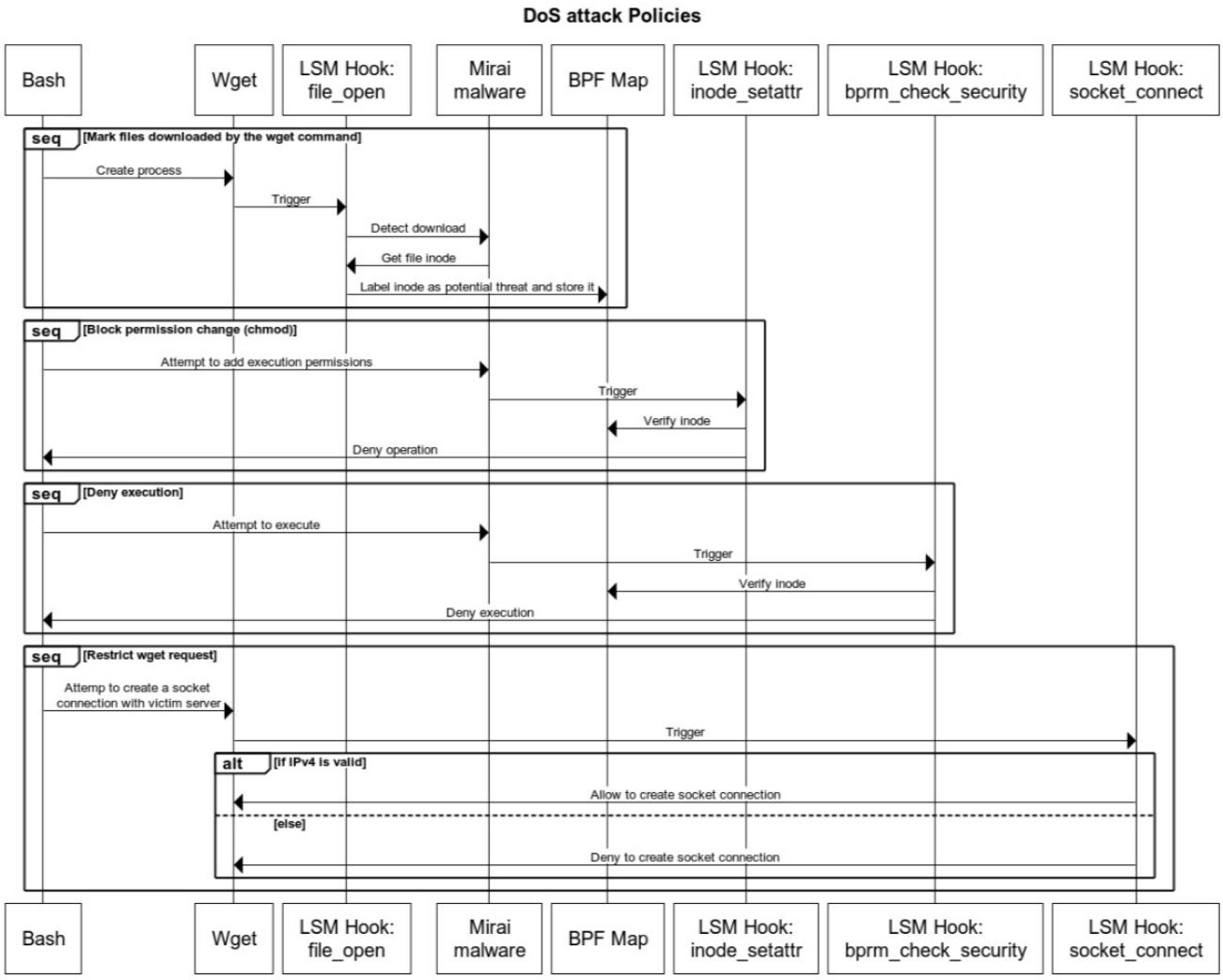

Figure 3: Sequence diagram illustrating the enforcement of DoS attack prevention policies using in LSM hooks

(1) Files downloaded by the `wget` command were intercepted in the `file_open` LSM hook and marked as a potential threat by storing their inodes in BPF Maps. This strategy helped to identify such files for future enforcement policies.

(2) Modifications in file attributes, such as execution permissions, were intercepted in the `inode_setattr` LSM hook by comparing their inodes with potential inodes stored in BPF Maps. Execution modification was denied if the inode was in the BPF Maps; otherwise, the action was allowed.

(3) Execution attempts were intercepted in the `bprm_check_security` LSM hook, and execution was denied if the inode was found in the BPF Maps.

(4) A list of trusted IPv4 addresses was created, and when a socket creation was attempted, it was intercepted in the `socket_connect` LSM hook, allowing the socket creation if and only if the destination IP was in the trusted list.

4.3.2 Prevent Remote Login

To prevent unauthorised remote access, `Telnet service` was identified as a key access point. Multiple security policies were implemented to block its execution, access to its configuration files, and outbound connections over port 23

(1) Execution attempts were intercepted in `bprm_check_security` LSM hook. Whether the path matched `/usr/sbin/inetutils-inetd`, the execution of such file was denied. This prevented the Telnet daemon from starting.

(2) File access attempts were intercepted in `file_open` LSM hook. whether the path matched Telnet configuration files or directories, such as `/etc/inetd.conf` or `/etc/inetd.d/`, the access to those files was denied. This prevented malicious modification in Telnet service files.

(3) Outgoing connections over port 23 were intercepted and denied in `socket_connect` LSM hook. This prevented any attacker from starting the Telnet daemon.

4.3.3 Prevent Infection

To prevent infection to other IoT devices, the Telnet client was identified as a key access point, as it is used to attempt brute-force attacks and connect with the victim device. See Figure 1.

Attempts to execute any file were intercepted in the `bprm_check_security` LSM hook. A comparison between its path and the `Telnet client` path was performed. If they matched, the execution of such a file was denied. Multiple possible paths were considered to increase the success rate.

5 Implementation

This section presents the practical implementation of the proposed architecture described in Section 4. It explains the development environment and tools used to enforce the security policies into kernel through the LSM interface (Smalley et al., 2024). This section also describes in details how each policy was implemented using specific LSM hooks, and includes the results obtained during testing, with the corresponding system logs.

5.1 Development Environment

Although no changes into the kernel code were required, some parameters in kernel settings had to be configured in advance to make everything work correctly, such parameters are:

- `CONFIG_BPF=y`
- `CONFIG_BPF_SYSCALL=y`
- `CONFIG_BPF_LSM=y`
- `CONFIG_DEBUG_INFO_BTF=y`
- `CONFIG_LSM="bpf,apparmor"`

To implement the eBPF-based security policies, eBPF programs were written in restricted C, which complies with the constraints enforced by the eBPF verifier in the Linux kernel. The set of tools used for compilation and deployment is listed in Table 5

Table 5: Development Tools for LSM BPF Integration

Tool	Version	Description
LLVM & Clang	18.1.3+	Tool for compiling restricted-C code to BPF bytecode.
BCC (libbpf)	0.35.0+	Toolkit for writing and loading eBPF programs (iovisor Project, 2024).
eunomia-bpf	0.3.4	Lightweight eBPF development framework (Eunomia eBPF, 2023).

The generated BPF bytecode was then loaded into the kernel and attached to specific LSM hooks to enforce the designed security policies.

5.2 Mirai Environment

A simple deployment infrastructure was chosen based on the original repository in github (Gamblin, 2016), as the goal was not to evaluate Mirai's internal components, but rather to make use of the effectiveness of its attack. The entire infrastructure was deployed in virtual machines, which were connected through a local network with no internet access. To allow the Mirai botnet running on the Raspberry Pi to resolve the CNC domain, a local DNS server was set up. An Apache HTTP server was also deployed to make the Mirai botnet available to the Raspberry PI.

5.3 Policy Enforcement outputs

After attaching the security policies in the kernel, its effectiveness was evaluated by performing critical tasks that Mirai attack could perform during its life cycle. The `/sys/kernel/debug/tracing/trace pipe` file was read to appreciate the security policies outputs.

5.3.1 Prevent DoS Attack

After applying the security policies to prevent the DoS attack, as explained in Section 4.3.1, the Mirai bot was downloaded from the Mirai loader server. This action triggered the first policy, which stored the bot's inode. This event can be seen in the file `/sys/kernel/debug/tracing/trace_pipe`. After trying to change the file permissions, the second policy was activated and blocked the action. Later, when the file was executed, the operation was denied, but this time by the Operating System's Discretionary Access Control(DAC) due to the execution permission lack, not by the proposed security policies.

```
device$ wget http://cnc.mirai.local/bins/bins.sh
device$ ls bins.sh
device$ 1835058 -rw-rw-r-- 1 device device 330 Jun 26 17:48 bins.sh
device$ chmod +x bins.sh
device$ chmod: changing permissions of 'bins.sh': Operation not permitted
device$ ./bins.sh
device$ bash: line 11: ./bins.sh: Permission denied
```

```
device$ sudo cat /sys/kernel/debug/tracing/trace_pipe
device$   wget-2214 bpf_trace_printk: Marked inode 1835058 as created by wget
device$   <...>-2222 bpf_trace_printk: Blocking chmod on inode 1835058 (wget)
```

After disabling the security policy responsible for blocking permission modifications, execution permissions were granted to the bot and execution attempt was made. The security policies successfully prevented the malware from being executed.

```
device$ wget http://cnc.mirai.local/bins/bins.sh
device$ chmod +x bins.sh
device$ ./bins.sh
device$ -bash: ./bins.sh: /bin/bash: bad interpreter: Operation not permitted
```

```
device$ sudo cat /sys/kernel/debug/tracing/trace_pipe
device$   wget-2214 bpf_trace_printk: Marked inode 1835058 as created by wget
device$   <...>-3666 bpf_trace_printk: Blocking execution of inode 1835058 (wget)
```

After defining a list of trusted IP addresses, such as the DNS server with IP address `150.50.0.150` and an HTTP Server with the domain `allow.server.mirai.local` domain and IP address `150.50.0.102`, the effectiveness of the policy to only allow downloads from trusted sources was tested. Downloading files from `150.50.0.102` was allowed, while connections to the Mirai loader with IP address `150.50.0.100` were blocked. The DNS server (`150.50.0.150`) was also included in the trusted list to allow domain resolution for `cnc.mirai.local` and `allow.server.mirai.local`.

```
device$ wget http://cnc.mirai.local/bins/bins.sh
device$ Resolving cnc.mirai.local (cnc.mirai.local)... 150.50.0.100
Connecting to cnc.mirai.local (cnc.mirai.local)|150.50.0.100|:80... failed: Operation
    not permitted.

device$ wget http://allow.server.mirai.local
device$ HTTP request sent, awaiting response... 200 OK
```

```
device$ sudo cat /sys/kernel/debug/tracing/trace_pipe
device$   wget-3811 bpf_trace_printk: socket_connect: blocking connection to
    150.50.0.100
device$   wget-3775 bpf_trace_printk: socket_connect: found connect to 150.50.0.102
```

5.3.2 Prevent Remote Login

As explained in Section 4.3.2, the effectiveness of the security policy to prevent Telnet remote login connections was tested. Basic operations like, reading and writing configuration files from `Telnet service` were blocked by denying the access to open them. Then, execution attempt was carried out over `/usr/sbin/inetutils-inetd` daemon, which was successfully denied. Finally, using `Netcat`, TCP connections on ports 23 and 80 were created and then successfully blocked, thereby denying the Telnet service.

```
device$ cat /etc/inetd.conf
cat: /etc/inetd.conf: Operation not permitted

device$ /usr/sbin/inetutils-inetd --foreground $INETD_OPTS
-bash: /usr/sbin/inetutils-inetd: Operation not permitted

device$ nc -v cnc.mirai.local 23
nc: connect to cnc.mirai.local (150.50.0.100) port 23 (tcp) failed: Operation not
    permitted
device$ nc -v allow.server.mirai.local 80
Connection to allow.server.mirai.local (150.50.0.102) 80 port [tcp/http] succeeded!
```

```
device$ sudo cat /sys/kernel/debug/tracing/trace_pipe
device$ <...>-5224 bpf_trace_printk: file_open: denying open of /etc/inetd.conf
device$ <...>-5425 bpf_trace_printk: bprm_check_security: denying execute of /usr/sbin/
    inetutils-inetd
device$ nc-5661 bpf_trace_printk: socket_connect: Blocking connection in port 23
```

5.3.3 Prevent Infection

Finally, the malware infection described in Section 4.3.3 was successfully prevented by denying the execution of the `Telnet client` binary located at `/usr/bin/telnet`.

```
device$ telnet cnc.mirai.local
-bash: /usr/bin/telnet: Operation not permitted
```

```
device$ sudo cat /sys/kernel/debug/tracing/trace_pipe
<...>-1477 bpf_trace_printk: bprm_check_security: denying execution of /usr/bin/telnet
```

6 Evaluation

This evaluation extends the work of Miki et al. (2024) and Miki et al. (2023), which focused only on traditional MAC systems in IoT devices. Some tables were updated to include the LSM BPF approach. The file system applicability table was extended, and the protection table for default and customised settings against Mirai malware was revised. The kernel version table was updated from 2019–2021 to 2022–2024, because LSM BPF was released in 2020 with the kernel version 5.7 and had low adoption at first. New measurements were made on the target hardware, and results were compared with AppArmor in terms of processing delay and memory usage. These updates make the evaluation reflect the current state of LSM BPF for IoT protection. The next subsections describe the results obtained from the experiments and provide an interpretation of the results

6.1 Memory consumption

Once performed the evaluation of memory consumption discussed in section 3.1.1, the results shows that proposed security policies based on LSM BPF has a smaller memory impact than AppArmor, adding only 4.6 MB (5.22%) to the entire system with 88.2MB, while AppArmor needs an extra of 12.5MB(14.17%), this is illustrated in Table 6

Table 6: Measured Memory Consumption

MAC System	**Entire System (MB)**	**Increase (MB)**
None	88.2	-
AppArmor	100.7	12.5(14.17%)
LSM BPF	92.8	4.6 (5.22 %)

When these two increases were compared with the RAM of 871 devices in the official database of OpenWrt on Table 2, it was found that the proposed security policies based on LSM BPF meet the strictest efficiency rule (C1) when memory consumption must be less than 1 % of physical RAM on 33.5 % of devices. and the C2 rule, when memory consumption must be less than 2 % of physical RAM on 51.7 %, almost three times more than AppArmor, with 9.9 % and 21.01 %. Both approaches, AppArmor and LSM BPF, satisfied rule C3, demonstrating no obstacles in running all devices and C4 in all devices over 64MB of memory. Table 7 illustrates the results for the criterion.

Table 7: The rate of devices tolerate memory usage of LSM BPF and AppArmor on each criterion

MAC System	**C1**	**C2**	**C3**	**C4**
AppArmor	9.87%	21.01%	99.89%	100%
LSM BPF	33.52%	51.66%	100%	100%

These findings demonstrated that security policies based on LSM BPF are lighter and a more suitable choice for constrained routers with less than 128MB of memory, while AppArmor is still practical for modern devices with more RAM resources.

6.2 File Systems

The evaluation results in Table 8 showed that security policies based on LSM BPF can apply `per-file` control on all tested file systems, including those that do not support extended attributes, such as Cramfs and romfs. This means that its enforcement capability does not depend on specific file system features, unlike SELinux and TOMOYO (label-based systems), which rely on extended attributes to apply `per-file` control; otherwise, they are forced to operate at the `per-FS` level.

Table 8: Control level of different MAC systems on various file systems, showing how LSM BPF keeps per-file control even without extended attribute support. Results adapted from Miki et al. (2023) and Miki et al. (2024)

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	per-file	per-file	per-file	per-file	per-file
Linux	per-file	per-file	per-file	per-file	per-file
AppArmor	per-file	per-file	per-file	per-file	per-file
LSM BPF	per-file	per-file	per-file	per-file	per-file

These findings confirm that the proposed security policy based on LSM BPF is suitable for most IoT devices, providing consistent policy enforcement regardless of the file system type.

6.3 Processing Delays

Once the latency evaluation was performed by creating an AppArmor profile, and loading security policies based on LSM BPF using `socket_connect`, `inode_setattr`, `bprm_check_security`, `file_open` and accessing to BPF MAP, the results are shown in Table 9.

Table 9: Evaluation results of processing delays (unit: µs)

MAC	stat	open/close	read/write	0Kfile Create	0Kfile Delete	10Kfile Create	10Kfile Delete
None	2.37 (-)	3.61 (-)	0.29 (-)	18.65 (-)	12.35 (-)	31 (-)	19.88 (-)
AppArmor	3.84 (61.66%)	4.20 (16.29%)	0.32 (9.53%)	20.49 (9.88%)	12.93 (-4.66%)	33.27 (7.32%)	21.08(-6.05%)
LSM BPF	2.42 (1.99%)	3.74 (3.66%)	0.29 (-0.50%)	19.20 (2.97%)	12.57 (1.79%)	32.75 (5.64%)	21.32 (7.26%)

Results demonstrated that all security policies added not more than 7.26 % latency; most of them, like `stat`, `open/close` and 0 KB `create/delete` operation, remained below 3.66 %. It was also shown that the proposed security policies based on LSM BPF did not slow down the critical read/write path operation (-0.5 %), showing that its hooks did not interfere with minimal I/O. Ideal for constrained hardware with less than 128 MB.

In contrast, traditional MACs like AppArmor introduced a considerable overhead, especially in file system operations. This is because its security policy is based on path, and this has to deal most with metadata, so it affects the `stat` calls with 61.66 %, while LLSM BPF remained only on 1.99 %.

In conclusion, results have shown that BPF hooks run only when they really need to and do not slow down basic I/O operations, being suitable for constrained hardware with 128MB.

6.4 Kernel Version

After performing the analysis of kernel adoption over OpenWrt firmware (OpenWrt Project, 2024), it was concluded that the use of LSM BPF has increased over the last few years as more devices

have started using newer Linux kernels. In 2022, only about 32.5% of devices supported LSM BPF, but this number grew to around 56% in 2023, and then it reached 75% in 2024. On the other hand, traditional MACs like SELinux, Smack, TOMOYO Linux, and AppArmor have been fully supported since 2023. These results show that LSM BPF is becoming more common as kernel versions improve.

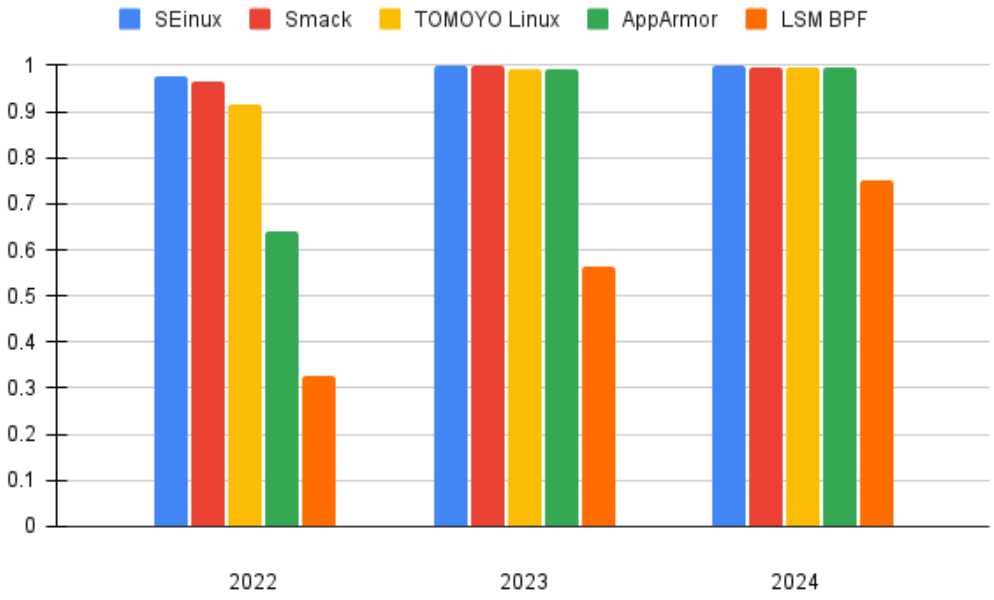

Figure 4: The rate of kernels since the merging of traditional MAC system and LSM BPF on the OpenWrt firmware analysis from 2022 to 2024

6.5 Protection with Default and Customized settings

The evaluation results in Table 10 show that, despite eBPF being enabled and it is used in modern OS versions, these was unable to prevent any stage of the simulated Mirai attack. This confirms that LSM BPF default configurations are not sufficient to address IoT-specific threats.

Table 10: Protection results of the attack simulated Mirai with traditional MAC and LSM, adapted from Miki et al. (2023) and Miki et al. (2024)

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
LSM BPF	Failure	Failure	Failure	Success	Success	Success

On the other hand, when proposed security policies based on LSM BPF were customised, this achieved complete mitigation of all three attack stages(Remote Login, DoS, and infection), demonstrating its capability to enforce targeted and effective protections.

6.6 Discussion

The results demonstrated that LSM BPF provides a strong and flexible protection for IoT devices when security policies are customised. In the evaluation, default settings failed to block any stage of the Mirai attack, showing that default settings are not enough for IoT threats. With

tailored policies, LSM BPF was completely excellent in denying Remote Login, DoS, and Infection by attaching rules to key points such as `file_open`, `inode_setattr`, `bprm_check_security`, and `socket_connect`, stopping the attack early.

Compared to the traditional MAC systems evaluated by Miki et al. (2023) and Miki et al. (2024), security policy based on LSM BF offers similar protection when they are tuned, but with important advantages for IoT. It does not depend on extended attribute support, so it can still enforce per-file control on lightweight file systems like Cramfs and romfs.

These results indicate that LSM BPF can be successfully applied in heterogeneous environmental conditions of IoT in which devices are characterised by limited resources and different configurations. At the same time, there are some limitations. LSM BPF requires a modern Linux kernel (version 5.7 or newer), and writing policies demands knowledge of eBPF programming, which may slow adoption in less technical environments.

In addition, the availability of specific functions from eBPF helpers varies between kernel versions. Knowledge of the LSM interface is also required, and not all LSM hooks are available in LSM BPF. Furthermore, some hook parameters in LSM BPF are restricted by the compiler. Finally, high-level tools to debug at the kernel level are also lacking; thus, deployment heavily relies on manual or custom scripts at present.

Another concern is the CPU and memory usage caused by the BPF compiler when a BPF program is being built. Although this compile-time cost does not affect LSM-BPF performance once the program is loaded into the kernel, it can complicate the deployment of such policies. Craun et al. (2023) address this by decoupling verification and just-in-time compilation from the device using a cross-compiler for BPF, but this approach increases the complexity of deploying LSM-BPF policies.

7 Conclusion and Future Work

In conclusion, this work has set out to answer the question: Can security policies be enforced effectively in kernel space to protect against IoT malware while maintaining high availability, low overhead in IoT devices? To answer such a question, an LSM BPF-based architecture was designed and implemented, identifying key enforcement hooks and integrating them through the LSM interface. The evaluation measured the applicability of LSM BPF through processing delay, memory consumption, and file system support, and protection effectiveness against the Mirai malware. The results confirm that the research question can be answered positively: default settings failed to block any stage of the Mirai attack, but customised policies successfully prevented Remote Login, DoS, and Infection stages without significant performance overhead. In particular, memory consumption remained below 5.2%, and LSM BPF demonstrated better adoption in devices with only 128 MB of RAM compared to AppArmor. LSM BPF also enforced per-file control on file systems without using xattr, making it suitable for constrainted IoT platforms.

These findings makes a LSM BPF as a viable and effective approach to IoT protection, capable of mitigating malware threats. However, its adoption depends on using a Linux kernel (5.7 or newer), demanding expertise in eBPF and the LSM interface, and addressing the lack of high-level management tools. Future work should create easy-to-use policy tools in the user space, ready-made policy sets for common IoT threats, and connect LSM BPF with automatic detection systems for flexible and scalable protection. With these improvements, LSM BPF could become a practical solution for securing the next generation of IoT devices.

References

Akiyama, M., Shiraishi, S., Fukumoto, A., Yoshimoto, R., Shioji, E. and Yamauchi, T. (2023). 'Seeingis 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 [Accessed 27 May 2025].

Alrawi, O., Lever, C., Valakuzhy, K., Court, R., Snow, K., Monrose, F. and Antonakakis, M. (2021). 'The circle of life: A large-scale study of the IoT malware lifecycle', *Proceedings of the 30th USENIX Security Symposium (USENIX Security '21)*, USENIX Association, pp. 3505–3522. Available at: https://www.usenix.org/conference/usenixsecurity21/presentation/alrawi-circle [Accessed 25 May 2025].

Bélair, M., Laniepce, S. and Menaud, J.-M. (2021). 'SNAPPY: Programmable kernel-level policies for containers', *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*, Association for Computing Machinery, New York, NY, USA, pp. 1636–1645. https://doi.org/10.1145/3412841.3442037.

Bessler, M., Sangster, P., Upadrashta, R. and O'Connor, T. J. (2024). 'Hardening the Internet of Things: Toward designing access control for resource-constrained IoT devices', *Proceedings of the 17th Cyber Security Experimentation and Test Workshop (CSET '24)*, Association for Computing Machinery, New York, NY, USA, pp. 1–7. https://doi.org/10.1145/3675741.3675744.

Brimhall, B., Garrard, J., De La Garza, C. and Coffman, J. (2023). 'A comparative analysis of linux mandatory access control policy enforcement mechanisms', *Proceedings of the 16th European Workshop on System Security (EuroSec '23)*, Association for Computing Machinery, pp. 1–7. https://doi.org/10.1145/3578357.3589454.

Bugiel, S., Heuser, S. and Sadeghi, A.-R. (2013). 'Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies', *Proceedings of the 22nd USENIX Security Symposium (USENIX Security '13)*, USENIX Association, Washington, D.C., pp. 131–146. Available at: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bugiel [Accessed 21 May 2025].

Craun, M., Oswald, A. and Williams, D. (2023). 'Enabling eBPF on embedded systems through decoupled verification', *Proceedings of the 1st Workshop on eBPF and Kernel Extensions (eBPF '23)*, Association for Computing Machinery, New York, NY, USA, pp. 63–69. https://doi.org/10.1145/3609021.3609299.

De Donno, M., Dragoni, N., Giaretta, A. and Spognardi, A. (2018). 'DDoS-capable IoT malwares: Comparative analysis and Mirai investigation', *Security and Communication Networks* **2018**: 1–30. Article ID 7178164. https://doi.org/10.1155/2018/7178164.

Eunomia eBPF (2023). 'eBPF hello world — eunomia eBPF tutorial'. Available at: https://eunomia.dev/en/tutorials/1-helloworld/ [Accessed 25 July 2025].

Findlay, W., Somayaji, A. and Barrera, D. (2020). 'bpfbox: Simple precise process confinement with eBPF', *Proceedings of the 2020 ACM SIGSAC Cloud Computing Security Workshop (CCSW '20)*, Association for Computing Machinery, New York, NY, USA, pp. 91–103. https://doi.org/10.1145/3411495.3421358.

Gamblin, J. (2016). 'Mirai source code: leaked for research/IoC development purposes (github repository)'. Available at: https://github.com/jgamblin/Mirai-Source-Code [Accessed 4 July 2025].

Guo, T., Zhang, P., Liang, H. and Shao, S. (2013). 'Enforcing multiple security policies for Android system', *Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation*, Atlantis Press, pp. 165–169. Available at: https://doi.org/10.2991/3ca-13.2013.42 [Accessed 1 June 2025].

Illakya, T., Keerthana, B., Murugan, K., Venkatesh, P., Manikandan, M. and Maran, K. (2024). 'The role of the Internet of Things in the telecom sector', *Proceedings of the 2024 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, IEEE, pp. 1–5. https://doi.org/10.1109/IC3IoT60841.2024.10550390.

iovisor Project (2024). 'BCC — installation guide'. Available at: https://github.com/iovisor/bcc/blob/master/INSTALL.md [Accessed 25 July 2025].

Ko, J.-Y., Lee, S.-G. and Lee, C.-H. (2019). 'Real-time mandatory access control on SELinux for the Internet of Things', *Proceedings of the 2019 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, pp. 1–6. https://doi.org/10.1109/ICCE.2019.8662112.

Linux Kernel Project (2023). 'Linux Security Module (LSM) hooks — lsm_hooks.h'. Available at: https://elixir.bootlin.com/linux/v6.12/source/include/linux/security.h [Accessed 10 April 2025].

Liu, C., Tak, B. and Wang, L. (2024). 'Understanding performance of eBPF maps', *Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions (eBPF '24)*, Association for Computing Machinery, New York, NY, USA, pp. 9–15. https://doi.org/10.1145/3672197.3673430.

Liu, K., Yang, M., Ling, Z., Yan, H., Zhang, Y., Fu, X. and Zhao, W. (2021). 'On manually reverse engineering communication protocols of Linux-based IoT systems', *IEEE Internet of Things Journal* **8**(8): 6815–6827. https://doi.org/10.1109/JIOT.2020.3036232.

Loukidis-Andreou, F., Giannakopoulos, I., Doka, K. and Koziris, N. (2018). 'DockerSec: A fully automated container security enhancement mechanism', *Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 1561–1564. https://doi.org/10.1109/ICDCS.2018.00169.

Miki, M., Yamauchi, T. and Kobayashi, S. (2023). 'Evaluation of effectiveness of MAC systems based on LSM for protecting IoT devices', *Proceedings of the 2023 Eleventh International Symposium on Computing and Networking (CANDAR)*, IEEE, pp. 161–167. https://doi.org/10.1109/CANDAR60563.2023.00029.

Miki, M., Yamauchi, T. and Kobayashi, S. (2024). 'Effectiveness of MAC systems based on LSM and their security policy configuration for protecting IoT devices', *Journal of Internet Services and Information Security* **14**: 293–315. Available at: https://api.semanticscholar.org/CorpusID:272633061 [Accessed 9 April 2025].

Nakamura, Y. and Sameshima, Y. (2008). 'SELinux for consumer electronics devices', *Proceedings of the Linux Symposium, Volume Two*, Linux Symposium, Ottawa, Ontario, Canada, pp. 125–134. Available at: https://landley.net/kdocs/ols/2008/ols2008v2-pages-125-134.pdf [Accessed 7 April 2025].

Nakamura, Y., Sameshima, Y. and Yamauchi, T. (2015). 'Reducing resource consumption of SELinux for embedded systems with contributions to open-source ecosystems', *Journal of Information Processing* **23**: 664–672. https://doi.org/10.2197/ipsjjip.23.664.

OpenWrt Project (2024). 'Table of hardware - full details'. Available at: https://openwrt.org/toh/views/toh_extended_all [Accessed 11 July 2025].

Smalley, S., Fraser, T. and Vance, C. (2024). 'Linux Security Modules: General security hooks for Linux'. Available at: https://docs.kernel.org/security/lsm.html [Accessed 10 April 2025].

Song, L. and Li, J. (2024). 'eBPF: Pioneering kernel programmability and system observability – past, present, and future insights', *2024 3rd International Conference on Artificial Intelligence and Computer Information Technology (AICIT)*, IEEE, pp. 1–10. https://doi.org/10.1109/AICIT62434.2024.10730620.

The Linux Kernel Documentation (2024). 'LSM BPF programs'. Available at: https://docs.kernel.org/bpf/prog_lsm.html [Accessed 10 April 2025].

Zhang, W., Liu, P. and Jaeger, T. (2021). 'Analyzing the overhead of file protection by Linux Security Modules', *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*, Association for Computing Machinery, New York, NY, USA, pp. 393–406. https://doi.org/10.1145/3433210.3453078.

Zhu, H. and Gehrmann, C. (2021). 'LicSec: An enhanced AppArmor Docker security profile generator', *Journal of Information Security and Applications* **61**: 102924. https://doi.org/10.1016/j.jisa.2021.102924 [Accessed 02 April 2025].