National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Alexander Mamani
Student ID: 23329823

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

Configuration Manual

Alexander Mamani
23329823

1 Environment Configuration

In this section, the LSM BPF configuration is addressed by providing details about the hardware used for this work , the parameters needed to compile the kernel, and, finally, the tools and dependencies needed to enforce the security policies based on LSM BPF.

1.1 Hardware Requirement

The hardware used for this work, either for applicability or protection, is described in Table 1. A basic configuration and boot of the Raspberry Pi OS image is discussed on its own website (Raspberry Pi Foundation, 2025a).

Table 1: IoT Environment for Performance Evaluation

CPU	BCM2837B0 64bit (ARMv8) 1.4 GHz
Memory	1024 Mb
OS	Raspberry Pi OS Lite - Debian version 12 (64bit)
Kernel version	6.12.34+rpt-rpi-v8

1.2 Kernel Configuration

Once the Raspberry Pi OS is installed, certain kernel-parameter configurations are needed. First, the essential dependencies for kernel compilation are installed. Then the kernel from the Raspberry Pi repository is cloned; after entering the kernel directory, a variable is defined to identify the Raspberry Pi version. Next, the kernel configuration file is generated, and the graphical interface for kernel configuration is executed (Raspberry Pi Foundation, 2025b)

```
device$ sudo apt install bc bison flex libssl-dev make dwarves libelf-dev numba
     pytest bcc bpfcc-tools raspberrypi-kernel-headers
device$ git clone --depth=1 --branch rpi-6.12.y https://github.com/raspberrypi/
    linux
device$ cd linux
device$ KERNEL=kernel8
device# make bcm2711_defconfig
device# make menuconfig
```

It is important to enable the following parameters in the kernel to support eBPF in the OS.

- CONFIG_BPF=y
- CONFIG_BPF_SYSCALL=y
- CONFIG_BPF_JIT=y
- CONFIG_TRACEPOINTS=y
- CONFIG_BPF_LSM=y
- CONFIG_DEBUG_INFO=y
- CONFIG_DEBUG_INFO_BTF=y
- CONFIG_LSM="bpf,apparmor"

By pressing the "/" key, an input box appears that allows you to find each parameter and list its dependencies.

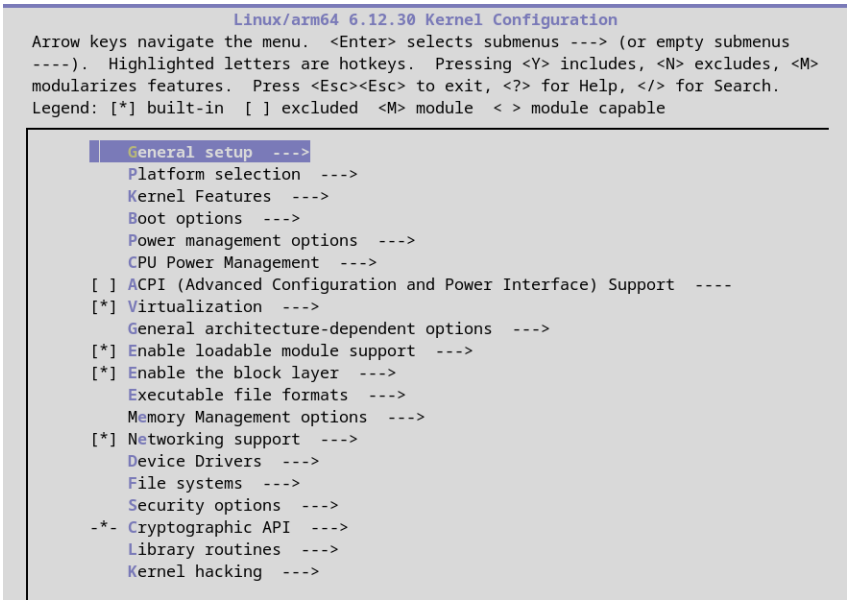

Figure 1: Graphical interface for adjust kernel parameters used for the `.config` file before to compile the kernel

After the kernel configuration parameters are set, compilation begins by executing the command `make`. With the following commands, the old kernel version is replaced by the new one.

```
device$ make -j6 Image.gz modules dtbs
device$ sudo make -j6 modules_install
device$ sudo cp /boot/firmware/$KERNEL.img /boot/firmware/$KERNEL-backup.img
device$ sudo cp arch/arm64/boot/Image.gz /boot/firmware/$KERNEL.img
device$ sudo cp arch/arm64/boot/dts/broadcom/*.dtb /boot/firmware/
device$ sudo cp arch/arm64/boot/dts/overlays/*.dtb* /boot/firmware/overlays/
device$ sudo cp arch/arm64/boot/dts/overlays/README /boot/firmware/overlays/
device$ sudo reboot
```

1.3 Software Requirements

As an important dependency for eBPF, the BPF Compiler Collection (BCC) is installed by executing the following commands (iovisor Project, 2025). First, the source code is cloned from the official repository, then BCC is compiled for the ARM architecture, and finally it is installed. In Table 2 the version of BCC is listed.

```
device$ sudo apt install -y zip bison cmake flex git libedit-dev libllvm18 llvm
    -18-dev libclang-18-dev python3 zlib1g-dev libelf-dev libbpfcc-dev libfl-
    dev python3-setuptools liblzma-dev libdebuginfod-dev arping netperf iperf
    libpolly-18-dev gcc-multilib clang llvm libbpf-dev libelf1
device$ git clone https://github.com/iovisor/bcc.git
device$ mkdir bcc/build; cd bcc/build
device$ cmake ..
device$ make
device$ sudo make install
device$ cmake -DPYTHON_CMD=python3 .. # build python3 binding
device$ pushd src/python/
device$ make
device$ sudo make install
device$ popd
```

Table 2: Development and Evaluation Tools

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, 2025).
eunomia-bpf	0.3.4	Lightweight eBPF development framework (eunomia-bpf, 2025).
lmbench	3.0-a9	lmbench benchmarking suite (Staelin and McVoy, 2009).
AppArmor	4.0	Traditional Mandatory Access Control (Canonical Group Ltd, 2025).

To develop the security policies based on LSM BPF, Eunomia-bpf was used; this compiler and runtime framework allowed attaching eBPF programs to kernel space in a user-friendly way. First, the Rust compiler is installed as a dependency of Eunomia-bpf, then the source code is cloned from its official repository, after which it is compiled and installed for the ARM architecture (eunomia-bpf, 2025).

```
device$ curl https://sh.rustup.rs -sSf | sh -s
device$ export PATH="$HOME/.cargo/bin:$PATH"
device$ git clone https://github.com/eunomia-bpf/eunomia-bpf.git
device$ cd eunomia-bpf
device$ make bpf-loader-rs
device$ make ecli
device$ make wasm-runtime
device@ make ecc
```

2 Mirai Attack Configuration

The Mirai Attack was deploy in an isolated local network, with virtual machines that replace physical servers or cloud instances. In Table 3 is shown the infrastructure necessary to deploy it.

Table 3: VM Environment for Mirai Attack Evaluation

Attribute	**Target IoT device**	**CNC Server**	**DNS Server**
CPU	Same as Table 1	2 vCPU (x86_64)	1 vCPU (x86_64)
Memory	Same as Table 1	2048 Mb	1024 Mb
OS	Same as Table 1	Ubuntu Server 24.04	Ubuntu Server 24.04
Kernel version	Same as Table 1	6.8.0-53-generic	6.8.0-53-generic

For ethical considerations, the step-by-step deployment of the Mirai malware will be skipped; however, the source code is available in a public repository for academic purposes (Gamblin, 2016)

3 About Applicability Evaluation

The following directory tree illustrates the research project artifact. At the first level, the protection directory stores the complete LSM BPF–based security policies used for the evaluation against Mirai, as discussed in Sections IV, V, and VI of the research document. The next level stores the LSM programs and the scripts used to generate the metrics for each applicability factor, such as memory consumption, file-system support, processing delay, and kernel-version analysis.

```
.
 |___ protection
 |      |___ DoS/
 |      |___ Infection/
 |      |___ Remote_Login/
 |
 |___ applicability
        |___ apparmor_config/
        |___ memory_consumption/
        |___ file_systems/
        |___ processing_delay/
```

3.1 AppArmor Configuration

The following commands illustrate the installation and configuration of an AppArmor profile; it was used to enforce the processing-delay and memory-consumption evaluations (Canonical Group Ltd, 2025), version of AppArmor is listed in Table 2. First, the AppArmor tools were installed, then an AppArmor profile was loaded from the artifact and parsed to verify its correctness. Thus, enforce mode was activated, allowing the blocking of operations that violated any profile rules. Finally, these modifications were confirmed by restarting AppArmor and querying its status.

```
devide$ sudo apt install apparmor-utils
device$ systemctl status apparmor
device$ sudo cp applicability/apparmor_config/usr.lib.lmbench /etc/apparmor.d/
    usr.lib.lmbench
device$ sudo apparmor_parser -r /etc/apparmor.d/usr.lib.lmbench
device$ systemctl restart apparmor
device$ sudo aa-enforce /etc/apparmor.d/usr.lib.lmbench
device$ apparmor_status | grep -i lmbench
```

```
# AppArmor must be stopped when the LSM BPF evaluation is running.
device$ systemctl stop apparmor
```

3.2 Memory consumption

The following commands shows how to execute the memory consumption evaluation. The execution of this script were done 10 times, for each time the OS were rebooted. The results of this evaluation are documented in the Google Sheets document linked below. Memory consumption Sheets

```
device$ cd applicability/memory_consumption/
device$ bash script.sh
```

3.3 File Systems

To evaluate the LSM BPF file systems support, tools like `cramfsprogs`, `squashfs-tools`, `mtd-utils`, and `genromfs` were installed to mount such file systems. Thus, the mount of all file system were carried out by and automated script.

```
device$ sudo apt-get update
device$ sudo apt-get install -y cramfsprogs squashfs-tools mtd-utils genromfs
device$ cd applicability/file_systems/
device$ sudo bash script_mount_filesystem.sh
```

As a next step, the security policies based on LSM BPF were loaded.

```
device$ cd applicability/file_systems/file_open_s__file_systems/
device$ sudo bash script.sh
```

And finally, each file system were accessed by denying access to restricted files, and allowing access to unrestricted files.

```
device$ cat /mnt/tmpfs/restricted.txt
cat: /mnt/tmpfs/restricted.txt: Operation not permitted

device$ cat /mnt/tmpfs/unrestricted.txt
Hi from unrestricted tmpfs
```

3.4 Processing Delays

To avoid the extra resource consumption typically introduced by the Eunomia-bpf framework, the security policies based on LSM BPF were loaded natively by: dumping the kernel's BTF data (`vmlinux.h`) with `bpftool`; compiling restricted C with `clang` for the ARMv7 architecture; generating a skeleton header from the compiled eBPF object; compiling and linking the skeleton header into a binary; and, finally, loading the LSM BPF program.

```
device$ cd applicability/processing_delay/lsm_processing_latency
# run LSM BPF program in a natively way to not incurr extra latency in
    evaluation
device$ sudo make run
```

The following commands illustrates the installation and execution of the processing delay evaluation with the lmbech tool (Staelin and McVoy, 2009).

```
device$ sudo apt-get install lmbench
device$ cd applicability/processing_delay/
device$ sudo bash script.sh
```

Results of the evaluation either LSM BPF or AppArmor are rerecorded are available in the Google Sheets liked below Processing Delay Sheets.

3.5 Kernel Version

Firmware version along OpenWrt devices were gathered from (OpenWrt Project, 2025), an public and open source dataset with all supported devices for OpenWrt. Analisys of kernel adoption is documented in Google Sheets liked below Supported Kernels Sheet

4 About Protection Evaluation

The following subsection shows how to execute the security policies based on LSM BPF described in research document. The output interpretation of each security policy is detailed in section V, and the design in section IV.

4.1 Prevent DoS

```
device$ cd protection/DoS/restric_files_download_by_wget
device$ sudo bash script.sh

device$ cd protection/DoS/wget_client_socket
device$ sudo bash script.sh
```

4.2 Prevent Remote Login

```
device$ cd protection/Remote_Login/telnet_service
device$ sudo bash script.sh
```

4.3 Prevent Infection

```
device$ cd protection/Infection/telnet_client
device$ sudo bash script.sh
```

5 Presentation

A demonstration of this research is available at the YouTube link below. YouTube presentation

References

Canonical Group Ltd (2025). AppArmor. Ubuntu Server Documentation. Last updated 23 July 2025. Available at: https://documentation.ubuntu.com/server/how-to/security/apparmor/ [Accessed 4 August 2025].

eunomia-bpf (2025). Build eCLI — building the Eunomia BPF project from source. Published 10 February 2025. Available at: https://eunomia.dev/en/eunomia-bpf/setup/build/#build-ecli [Accessed 3 August 2025].

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

iovisor Project (2025). BPF compiler collection (BCC). Version 0.35.0. Available at: https://github.com/iovisor/bcc [Accessed 3 August 2025].

OpenWrt Project (2025). Table of hardware — custom column view. Disponible en: https://toh.openwrt.org/ [Accedido el 3 de agosto de 2025].

Raspberry Pi Foundation (2025a). Raspberry Pi documentation — install OS. Available at: https://www.raspberrypi.com/documentation/computers/getting-started.html#installing-the-operating-system [Accessed 20 July 2025].

Raspberry Pi Foundation (2025b). Raspberry Pi documentation — Linux kernel. Available at: https://www.raspberrypi.com/documentation/computers/linux_kernel.html [Accessed 3 August 2025].

Staelin, C. and McVoy, L. (2009). *lmbench(8) — system benchmarks*, Ubuntu Manpage Repository. Version 3.0-a9-1. Available at: https://manpages.ubuntu.com/manpages/trusty/lmbench.8.html [Accessed 3 August 2025].