



**臺北市立大學**  
UNIVERSITY OF TAIPEI

**Computer Science**

**Linux Information Security Auditing  
And Exploitation Analysis**

**Student ID: U10916024**

**Student: Cheng-Hao, Zhang**

**Advisor: Dong-Hwa Lu**

**May 2024**





## 摘要

由於本系伺服器長年缺少安全人員管理及維護，造成存在許多安全漏洞。為此，本專題透過多款 Open-source 的工具（如：Lynis, ClamAV 等）對伺服器進行安全性掃描，修補多項具有安全隱憂的 CVE；此外，本文亦將自動化更新、掃描等重複性任務，撰寫成為 Bash 腳本，並透過系統排程重複自動執行之。

經過安全性工具稽核後，本文針對兩個重要安全漏洞進行了深入分析，分別為 PwnKit(CVE-2021-4034) 和 Looney Tunables(CVE-2023-4911)。

PwnKit 漏洞影響一個廣泛應用的 Linux daemon 系統服務，即 polkit，用於管理 Unix-like 作業系統中的系統行程（Process）權限。漏洞的本質是其中的 pkexec 程式對於參數指標未經檢查就直接使用，造成對於環境變數的越界存取，從而可能被攻擊者濫用，提升系統權限。修補方式主要為對 pkexec 執行過程中的參數數量 argc 進行檢查，以防止越界讀寫攻擊。

Looney Tunables 漏洞存在於 GNU C Library 的動態載入器 ld.so 中，當處理 GLIBC\_TUNABLES 環境變數時出現了緩衝區溢位。這使得攻擊者可以在 SUID 程式中，有權限執行的二進位檔案中使用特制的 GLIBC\_TUNABLES 環境變數，執行具有提升權限的惡意程式碼。修補建議強調在處理 GLIBC\_TUNABLES 環境變數時，對於存放其副本的緩衝區實施邊界以及剩餘空間的檢查，以杜絕緩衝區溢位風險。

綜上所述，本報告深入探討了這兩個漏洞的利用邏輯、影響和可能的後果，並針對落點提出了有效的修補建議。以利提高伺服器系統的安全性，減少潛在攻擊風險。報告的結論中亦呼籲系統管理者和開發者們儘快更新至最新建議的修補版本，以確保系統在日後營運中排除安全風險；並且，開發者於進程式架構設計時，應該注重軟體品質以及安全性之控管，尤其是底層系統程式的記憶體、緩衝區管理。



## **abstract**

Due to the longstanding lack of security personnel management and maintenance in our department's servers, numerous security vulnerabilities exist. To address this issue, this project conducts security scans on the servers using various open-source tools such as Lynis, ClamAV, etc., to patch several CVEs that pose security concerns. Additionally, this paper automates repetitive tasks such as updates and scans, by scripting them in Bash and scheduling them to run automatically.

After conducting security tool audits, this paper analyzes two critical security vulnerabilities: PwnKit (CVE-2021-4034) and Looney Tunables (CVE-2023-4911).

The PwnKit vulnerability affects a widely used Linux daemon system service, polkit, which manages process permissions in Unix-like operating systems. The vulnerability lies in the unchecked use of parameter pointers by the pkexec program, leading to out-of-bounds access to environment variables, potentially exploitable by attackers to elevate system privileges. The remedy primarily involves checking the parameter count `argc` during pkexec execution to prevent buffer overflow attacks.

The Looney Tunables vulnerability resides in the dynamic loader `ld.so` of the GNU C Library, where a buffer overflow occurs when processing the `GLIBC_TUNABLES` environment variable. This allows attackers to execute malicious code with escalated privileges in SUID binaries using specially crafted `GLIBC_TUNABLES` environment variables. The suggested fix emphasizes boundary and remaining space checks on the buffer holding copies of `GLIBC_TUNABLES` to mitigate buffer overflow risks.

This report thoroughly examines these two vulnerabilities' exploitation logic, impacts, and potential consequences, providing effective patch recommendations. This aims to enhance the security of server systems and reduce potential attack risks. The report's conclusion urges system administrators and developers to promptly update to the latest recommended patch versions to mitigate security risks in future operations. Furthermore, developers should prioritize software quality and security controls, especially in memory and buffer management of underlying system programs during software architecture design.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Auditing</b>	<b>3</b>
2.1	yum update . . . . .	3
2.2	Lynis . . . . .	3
2.3	Chkrootkit . . . . .	6
2.4	ClamAV . . . . .	7
2.5	Linux Malware Detect . . . . .	7
2.6	Nessus Vulnerability Scanning . . . . .	8
<b>3</b>	<b>PwnKit (CVE-2021-4034)</b>	<b>11</b>
3.1	Polkit Introduction . . . . .	12
3.2	Analysis . . . . .	13
3.3	Exploitation . . . . .	15
3.4	Repetition . . . . .	17
3.5	Patch . . . . .	19
<b>4</b>	<b>Looney Tunables (CVE-2023-4911)</b>	<b>21</b>
4.1	GNU C Library dynamic loader . . . . .	21
4.2	Analysis . . . . .	21
4.3	Exploitation . . . . .	25
4.4	Patch . . . . .	30
	<b>Bibliography</b>	<b>35</b>



# List of Figures

2.1	Executing Lynis . . . . .	4
2.2	Lynis Update Check . . . . .	5
2.3	Lynis Audit Details . . . . .	5
3.1	CVE-2021-4034 CVSS Vector . . . . .	11
3.2	Program Stack . . . . .	14
3.3	Linux uname . . . . .	17
3.4	pkexec version . . . . .	17
3.5	Before exploit . . . . .	18
3.6	After exploit . . . . .	18
3.7	Throw an Exception . . . . .	19
4.1	Memory map of <code>id.so</code> Segments . . . . .	26

# List of Source Codes

3.1	<code>pkexec.c</code> <code>main()</code> , line 435-640 . . . . .	13
3.2	<code>pkexec.c</code> <code>main()</code> , line 649-702 . . . . .	15
3.3	<code>pkexec.c</code> , line 88-409 . . . . .	16
3.4	<code>pkexec.c</code> <code>main()</code> , line 493-499 . . . . .	19
4.1	<code>__tunables_init()</code> function . . . . .	22
4.2	<code>parse_tunables()</code> function . . . . .	22
4.3	<code>diff --git a/elf/dl-tunables.c b/elf/dl-tunables.c</code> . . . . .	31



# Chapter 1

## Introduction

The motivation behind this research stems from the prolonged neglect of server security measures, resulting in a landscape fraught with vulnerabilities and potential cyber threats. With numerous Common Vulnerabilities and Exposures (CVEs) persisting in server environments, the risks to critical services are heightened, considering their extensive audience. Furthermore, the escalating nature of cybersecurity threats underscores the urgency for proactive measures rather than reactionary fixes.

Therefore, this study aims to address these challenges through a multifaceted approach. By bolstering cybersecurity measures, educating stakeholders, and spreading awareness about effective security practices, this research seeks to fortify server infrastructure against potential attacks.

Through vulnerability scanning, automation, and investigation of specific vulnerabilities like specific vulnerabilities such as Pwnkit (CVE-2021-4034) and Looney Tunables (CVE-2023-4911), which pose local privilege escalation risks in Polkit's `pkexec` and `glibc`'s `ld.so`, respectively. Comprehensive strategies will be developed to mitigate risks effectively and ensure the integrity and reliability of server systems.



# Chapter 2

## Auditing

### 2.1 yum update

#### auto-update

Switch to the root user's crontab:

```
sudo crontab -e
```

Add the following line to schedule monthly system updates, ensuring that it runs with sudo:

```
0 3 1 * * sudo /usr/bin/yum -y update
```

In this line:

- `0 3 1 * *` specifies the timing to run the update on the 1st day of every month at 3:00 AM.
- `sudo /usr/bin/yum -y update` is the command to update all installed packages using yum. The `-y` flag answers "yes" to any prompts that may come up during the update process.

### 2.2 Lynis

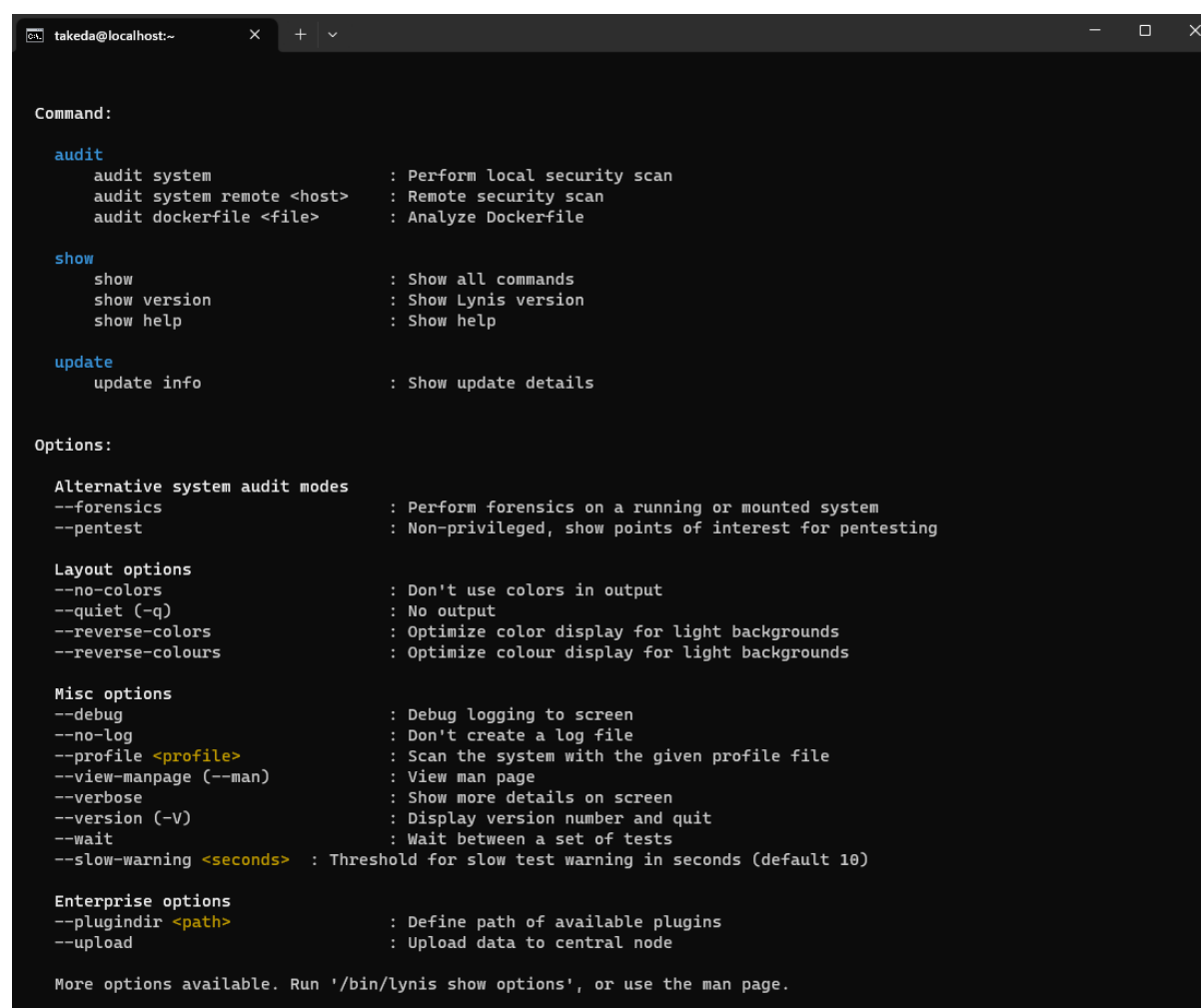
Lynis, an introduction Auditing, system hardening, compliance testing

#### Install

```
# Github
git clone https://github.com/CISOfy/lynis
cd lynis
chmod +x ./lynis
./lynis

# yum
yum install lynis -y
```

Figure 2.1 is the executing Lynis system, it shows the basic commands and the remaining options.



```
Command:
audit
  audit system          : Perform local security scan
  audit system remote <host> : Remote security scan
  audit dockerfile <file> : Analyze Dockerfile

show
  show          : Show all commands
  show version  : Show Lynis version
  show help     : Show help

update
  update info    : Show update details

Options:

Alternative system audit modes
--forensics      : Perform forensics on a running or mounted system
--pentest        : Non-privileged, show points of interest for pentesting

Layout options
--no-colors      : Don't use colors in output
--quiet (-q)     : No output
--reverse-colors : Optimize color display for light backgrounds
--reverse-colours : Optimize colour display for light backgrounds

Misc options
--debug          : Debug logging to screen
--no-log         : Don't create a log file
--profile <profile> : Scan the system with the given profile file
--view-manpage (--man) : View man page
--verbose        : Show more details on screen
--version (-V)   : Display version number and quit
--wait          : Wait between a set of tests
--slow-warning <seconds> : Threshold for slow test warning in seconds (default 10)

Enterprise options
--plugindir <path> : Define path of available plugins
--upload       : Upload data to central node

More options available. Run '/bin/lynis show options', or use the man page.
```

Figure 2.1: Executing Lynis

## Update info check

```
lynis update info
```

Figure 2.2 shows the Lynis status, we can check whether the version is up-to-date.

```
== Lynis ==

Version      : 3.0.8
Status       : Up-to-date
Release date : 2022-05-17
Project page : https://cisofy.com/lynis/
Source code  : https://github.com/CISOfy/lynis
Latest package : https://packages.cisofy.com/

2007-2021, CISOfy - https://cisofy.com/lynis/
```

Figure 2.2: Lynis Update Check

## Start the audit

```
sudo lynis audit system
```

Figure 2.3 shows the security scan details including the index, tests performed, and plugins enabled. The hardening index shows the score of auditing on the system.

```
Lynis security scan details:

Hardening index : 69 [#####          ]
Tests performed : 267
Plugins enabled  : 0
```

Figure 2.3: Lynis Audit Details

## Auto-scanning once a month

Open your terminal and run the following command to edit your crontab file:

```
crontab -e
```

Add the following line to schedule a monthly Lynis scan with a dynamic log file name that includes the month:

```
0 3 1 * * /usr/sbin/lynis audit system --cronjob > /var/log/lynis_$(
date +%Y%m).log
```



In this modified cron job:

- `0 3 1 * *` specifies the timing to run the Lynis scan on the 1st day of every month at 3:00 AM.
- `/usr/sbin/lynis audit system --cronjob` runs Lynis with the `--cronjob` flag to prevent Lynis from prompting for user input.
- `{/var/log/lynis_$(date +%Y%m).log}` redirects the output of the Lynis scan to a log file with a name that includes the current year and month (e.g., "lynis\_202309.log" for September 2023).

Save the file and exit the text editor. With this setup, each time the cron job runs, it will create a new log file with a name that includes the current year and month, ensuring that you have separate log files for each month.

## 2.3 Chkrootkit

### Chkrootkit - Linux Rootkit Scanner

Chkrootkit 是一個 open-source 的 rootkit 掃描工具，它能在 Unix 系統上進行檢查 rootkit。它有助於檢測隱藏的安全漏洞。Chkrootkit 包含一個 shell script 及一個 program，Shell script 將會檢查系統二進位文件以進行 rootkit 修改，而程式將會檢查各種安全問題。

```
$ sudo apt install chkrootkit
```

For CentOS systems, you need to download it using the following commands:

```
yum update
yum install wget gcc-c++ glibc-static
wget -c ftp://ftp.pangeia.com.br/pub/seg/pac/chkrootkit.tar.gz
tar -xzf chkrootkit.tar.gz
mkdir /usr/local/chkrootkit
mv chkrootkit-0.53/* /usr/local/chkrootkit
# (Check the version after extraction, adapt to 0.5X)
cd /usr/local/chkrootkit
```

```
make sense
```

Now you can run Chkrootkit:

```
sudo chkrootkit (Debian)
# OR
/usr/local/chkrootkit/chkrootkit (CentOS)
```

After running, you can see whether your server has any malware or rootkits in the report.

If you want to run it automatically every night and receive email notifications, you can use the following cron job to run it at 3 AM and send the report to your email address:

```
0 3 * * * /usr/sbin/chkrootkit 2>&1 | mail -s "chkrootkit Reports of
My Server" name@example.com
```

## 2.4 ClamAV

*ClamAV, short for “Clam AntiVirus,” is an open-source antivirus software toolkit designed to detect and combat various forms of malware, including viruses, trojans, worms, and other malicious software. ClamAV is widely used in personal and professional settings to provide an additional layer of security against malware threats.*

```
yum -y update
yum -y install clamav
#Installation and startup are straightforward.
freshclam
clamscan -r -i DIRECTORY
```

## 2.5 Linux Malware Detect

First, download LMD and extract:

```
wget http://www.rfxn.com/downloads/maldetect-current.tar.gz
tar -zxvf maldetect-current.tar.gz
```

Install as root. The current latest version is 1.6, and when installing, you need to change the directory name after extraction:

```
cd maldetect-1.6
./install.sh
```

Then create a symbolic link to the /bin directory for the maldet command:

```
ln -s /usr/local/maldetect/maldet /bin/maldet
hash -r
```

The default installation directory for LMD is in /usr/local/maldet/. The conf.maldet file is the LMD configuration file. Open LMD's configuration file:

```
vim /usr/local/maldetect/conf.maldet
```

Modify the following options:

- Set to 1 to enable email alerts:

```
email_alert="1"
```

- Set the email address to receive alerts:

```
email_addr="you@yourdomain.com"
```

- Use ClamAV for scanning:

```
scan_clamscan="1"
```

[Reference](<https://www.ltsplus.com/linux/centos-7-install-lmd-clam-antivirus>)

## 2.6 Nessus Vulnerability Scanning

Nessus detects vulnerabilities: HTTP TRACE / TRACK Methods Allowed CVSS V3.0: 5.3

HTTP TRACE/TRAC is commonly used for debugging. To verify if the system indeed has TRACE/TRACK functionality enabled, use telnet on port 80 of the web page:

```
telnet 127.0.0.1 80
```

Then enter the following:

```
TRACE / HTTP/1.1  
Host: localhost.localdomain
```

Press Enter twice.

If the system responds with:

```
HTTP/1.1 200 OK
```

It indicates that HTTP TRACE/TRACK methods are allowed. To disable, add `TraceEnable off` to the Apache configuration file `httpd.conf`:

```
vi /etc/httpd/conf/httpd.conf
```

Add the line:

```
TraceEnable off
```

Then restart Apache:

```
service httpd restart
```

Test again using telnet. The system should respond with:

```
HTTP/1.1 403 Forbidden
```



## Chapter 3

### PwnKit (CVE-2021-4034)

- keyword: Local Privilege Escalation, out-of-bounds read/write, Memory corruption, SUID-root program
- CVSS v. 3.1: **7.8 HIGH**, Figure 3.1 is the visual representation of the CVSS score for CVE-2021-4034.

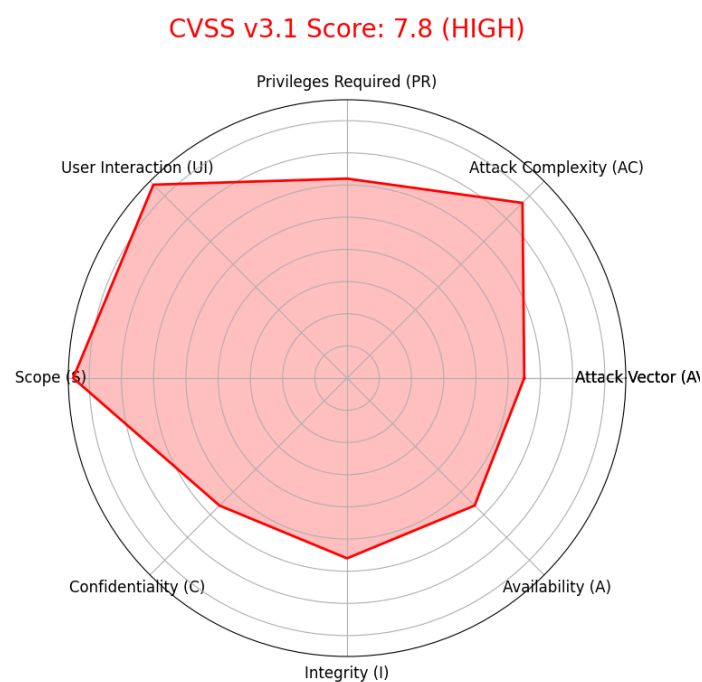


Figure 3.1: CVE-2021-4034 CVSS Vector

## 3.1 Polkit Introduction

**polkit** 是一個在 Unix like 作業系統中，用來控制系統 process 權限的一套工具。它提供非特權 processes 以一個有系統性的方式與特權 processes 進行溝通；也可以使用 **polkit** 裡面具有提升權限的指令 **pkexec**，來取得 root 特權 (與 **sudo** 不同，**polkit** 並沒有賦予完全的 root 權限)。

*Polkit (formerly PolicyKit) is a component for controlling system-wide privileges in Unix-like operating systems. It provides an organized way for non-privileged processes to communicate with privileged ones. It is also possible to use polkit to execute commands with elevated privileges using the command pkexec followed by the command intended to be executed (with root permission). [6]*

Researchers from *Qualys*[5] describe the vulnerability as a dream come true for attackers:

- **pkexec** is installed by default on various Linux distributions.
- The vulnerability has existed since May 2009 (commit c8c3d83, "Add a **pkexec**(1) command").
- Any non-privileged user can obtain full root privileges.
- The vulnerability can be exploited even if **polkit** itself is not running.

Furthermore, **pkexec** is a **sudo**-like, SUID (SetUID)-root tool with the following declarations:

### NAME

**pkexec** - Execute a **command** as another user

### SYNOPSIS

**pkexec** [--version] [--disable-internal-agent] [--**help**]

**pkexec** [--user username] PROGRAM [ARGUMENTS...]

### DESCRIPTION

**pkexec** allows an authorized user to execute PROGRAM as another user.

If PROGRAM is not specified, the default shell will be run.  
If username is not specified, **then** the program will be executed as the administrative super user, root.

## 3.2 Analysis

要分析此漏洞，我們要觀察 source code—pkexec.c (ver. 0.120)[3]，先從 main() 函數著手，Listing 3.1 列出了 main() 函數的一部分：

```
435 main (int argc, char *argv[])
436 {
...
534     for (n = 1; n < (guint) argc; n++)
535     {
...
568     }
...
610     path = g_strdup (argv[n]);
...
629     if (path[0] != '/')
630     {
...
632         s = g_find_program_in_path (path);
...
639         argv[n] = path = s;
640     }
```

Listing 3.1: pkexec.c main(), line 435-640

Terms explanation:

- **int argc**: argument count, 意即參數的個數；根據 C99 規範 “The value of argc shall be nonnegative.” argc 的值應為非負整數。
- **char \*argv[]**: 參數的值。

此程式乍看之下並無任何問題，但在些許非正常使用情況下將造成漏洞威脅。



正常情況下，`argc` 的值至少為 1，因為 argument list 最少包括程式自身名稱；但不幸的是，若使用 `execve()` 進行系統呼叫的時候，傳遞給它的值為空的 i.e. `NULL`，此時的 `argc` 為 0，`argv[0]` 為 `NULL`，並且：

- Line 534: 整數 `n` 將永遠是 1。
- Line 610: 指標 `path` 將因 `argv[1]` 造成越界讀取。
- Line 639: 指標 `s` 將越界寫入到 `argv[1]`。

那實際越界存取到的這個 `argv[1]` 究竟是什麼位置呢？要解答這個問題，必須先了解呼叫 `execve()` 的時候，kernel 做了甚麼動作。當使用它新增了一個程式時，kernel 會將參數、環境變數字串以及指標們（`argv`、`envp`）複製到此程式的堆疊結尾；圖例如 Figure 3.2，模擬出程式堆疊的連續狀態以及存放內容：

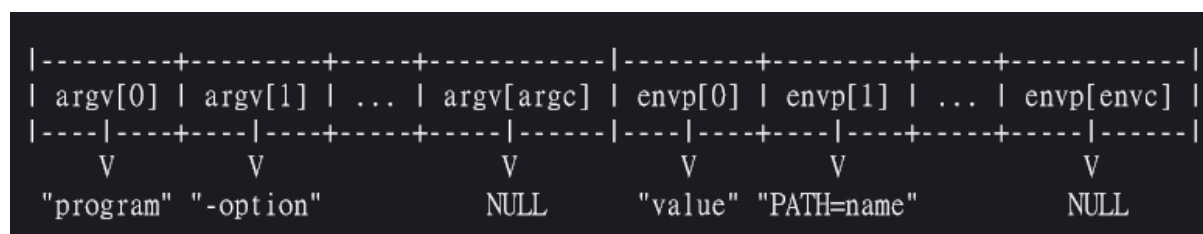


Figure 3.2: Program Stack

因為 `argv` 及 `envp` 指標在記憶體中是連續的，可以輕易地觀察到，若 `argc` 是 0，這個越界 `argv[1]` 其實是 `envp[0]`，這個指標指向的第一個環境變數 `value`。

- 在第 610 行，從 `argv[1]`（即 `envp[0]`）中讀取的程式路徑超出了範圍，並指向 `"value"`；
- 在第 632 行，這個路徑 `"value"` 被傳遞給 `g_find_program_in_path()`（因為 `"value"` 不以斜線開頭，所以在第 629 行）；
- `g_find_program_in_path()` 在 `PATH` 環境變數的目錄中搜尋名為 `"value"` 的可執行檔；如果找到這樣的可執行檔，則將其完整路徑返回到 `pkexec` 的 `main()` 函數（line 632）；
- 並且在第 639 行，將這個完整路徑寫出到 `argv[1]`（即 `envp[0]`），從而覆蓋第一個環境變數。

更精確地說：

- 如果 `PATH` 環境變數是 `"PATH=name"`，而且十分剛好的這個目錄 `name` 存在在當前工作目錄，又更剛好的裡面有個可執行檔案 `value`，然而此指向這個字串 `"name/value"` 會被越界寫入到 `envp[0]`。
- `"PATH=name"` 可替換成 `"PATH=name="` 亦成立
- 換言之，此類的越界寫入允許重新引入這些不安全的環境變數到 `pkexec` 中（例如，`LD_PRELOAD`）；此類不安全的變數通常在呼叫 `main()` 會被 `ld.so` 從 SUID 程式中移除。
- 此外，值得注意的是 `polkit` 也支援非 Linux 作業系統，如：Solaris、BSD；然而在 OpenBSD 中，因為其 kernel 不接受 `argc` 為 0 的 `execve()`，此漏洞不可被利用。

### 3.3 Exploitation

Listing 3.2 列出了 `main()` 裡面對於環境變數的處理。

至於要利用何種不安全的環境變數呢？這裡的選項是很有限的，因為在越界寫入後（at line 639）不久後，`pkexec` 會完全地清除了他的環境變數（at line 702）。

```
639     argv[n] = path = s;
...
657     for (n = 0; environment_variables_to_save[n] != NULL; n++)
658     {
659         const gchar *key = environment_variables_to_save[n];
...
662         value = g_getenv (key);
...
670         if (!validate_environment_variable (key, value))
...
675     }
...
702     if (clearenv () != 0)
```

Listing 3.2: `pkexec.c main()`, line 649-702

Qualys[5] 指出：

為了印出錯誤訊息到 `stderr`，`pkexec` 呼叫了 GLib 的 `g_printerr()`；

*the GLib is a [GNOME](https://www.gnome.org/) library, not the GNU C Library,  
aka glibc [1]*

例如：`validate_environment_variable()` 和 `log_message()` 呼叫了 `g_printerr()` (在第 126 和 408-409 行)：

Listing 3.3 列出了 `pkexec.c` 裡面處理 `stderr` 的過程，呼叫了 `g_printerr()`。

```
88 log_message (gint      level,
89              gboolean print_to_stderr,
90              const    gchar *format,
91              ...)
92 {
...
125  if (print_to_stderr)
126      g_printerr ("%s\n", s);
...
383 validate_environment_variable (const gchar *key,
384                                const gchar *value)
385 {
...
406      log_message (LOG_CRIT, TRUE,
407                  "The value for the SHELL variable was
    not found the /etc/shells file");
408      g_printerr ("\n"
409                  "This incident has been reported.\n");
```

Listing 3.3: `pkexec.c`, line 88-409

`g_printerr()` 通常會輸出 UTF-8 的錯誤訊息，但是若他的環境變數 `CHARSET` 不是 UTF-8，他也可以輸出其他的字元集。這裡的 `CHARSET` 與安全無關，他不是不安全的環境變數之一。

為了轉換錯誤訊息從 UTF-8 到其他字元集，`pkexec` 將會呼叫 `glibc` 當中的 `iconv_open()`。

要將一種字元集的訊息轉換為另一種字元集，會使用 `iconv_open()` 函數來執行相關的共享 library 來進行轉換操作。通常，這個函數會使用預設設定檔（通常在 `/usr/lib/gconv/gconv-modules`）中的設定，包括來源字元集、目的字元集以及 library name。

不過，也可以透過設定環境變數 `GCONV_PATH` 來強制 `iconv_open()` 使用其他設定檔案。需要注意的是，`GCONV_PATH` 屬於一個「不安全」的環境變數，因為它有潛在的風險，可能會執行任意的 library。因此，在 SUID 程式中，系統會自動刪除 `GCONV_PATH` 變數，以確保安全性。

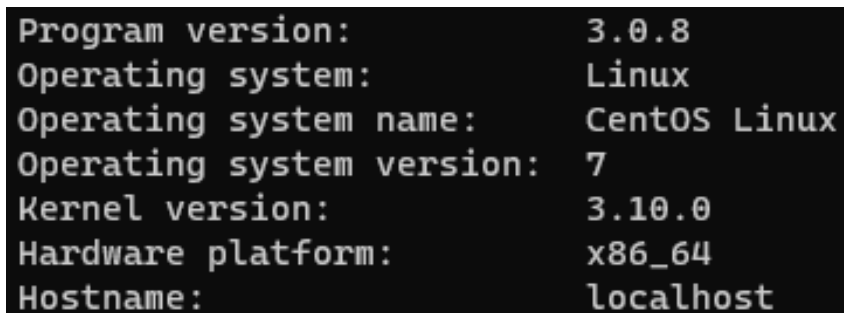
不幸的是，CVE-2021-4034 允許重新引入（re-introduce）`GCONV_PATH` 進入 `pkexec` 當中，並執行共享函式庫，就像 root 一樣。

但此漏洞利用技術會在 log file 當中留下 "The value for the SHELL variable was not found the /etc/shells file" (line 406, 407)。

## 3.4 Repetition

- Linux version: Linux

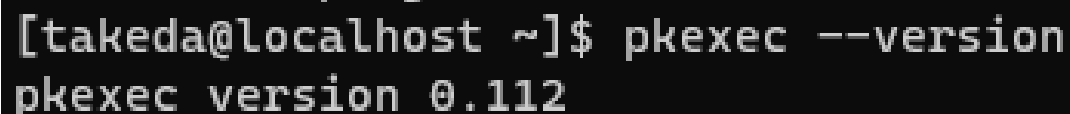
```
localhost.localdomain 3.10.0-1160.21.1.el7.x86_64 #1 SMP  
Tue Mar 16 18:28:22 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```



```
Program version:      3.0.8  
Operating system:    Linux  
Operating system name: CentOS Linux  
Operating system version: 7  
Kernel version:      3.10.0  
Hardware platform:    x86_64  
Hostname:             localhost
```

Figure 3.3: Linux uname

- pkexec version: 0.112



```
[takeda@localhost ~]$ pkexec --version  
pkexec version 0.112
```

Figure 3.4: pkexec version

- 提權前狀況

```
[takeda@localhost ~]$ id
uid=1883(takeda) gid=1884(takeda) groups=1884(takeda),10(wheel)
```

Figure 3.5: Before exploit

- 執行破解程式後

```
[takeda@localhost polkit-pkexec-ex]$ ./exploit
Current User before execute exploit
hacker@victim$whoami: takeda
Exploit written by @luijait (0x6c75696a616974)
[+] Enjoy your root if exploit was completed succesfully
[root@localhost polkit-pkexec-ex]# id
uid=0(root) gid=0(root) groups=0(root),10(wheel),1884(takeda)
[root@localhost polkit-pkexec-ex]# |
```

Figure 3.6: After exploit

## 3.5 Patch

由於此 CVE 風險分數高達 7.8，修補十分迅速，邏輯也十分簡單明瞭。[4]

```
/*  
 * If 'pkexec' is called THIS wrong, someone's probably evil-doing.  
 * Don't be nice, just bail out.  
 */  
if (argc<1)  
{  
    exit(127);  
}
```

Listing 3.4: `pkexec.c` `main()`, line 493-499

Listing 3.4 秀出了 Patch 的解決方案，你沒有看錯，就是加上一個 `if` 進行 `argc` 的判斷，並且拋出 `exit(127)`，如此暴力、簡單就能化解高風險漏洞；因此程式設計師對於變數、指標的變化必須要精確的掌控，以免發生此種越界讀寫的安全性問題。

Figure 3.7顯示，修補過後執行 exploit，會直接觸發 `if`，拋出 `exit()`：

```
[u10916024@localhost test]$ ./exploit  
Current User before execute exploit  
hacker@victim$whoami: u10916024  
pkexec --version |  
    --help |  
    --disable-internal-agent |  
    [--user username] PROGRAM [ARGUMENTS...]  
  
See the pkexec manual page for more details.  
  
Report bugs to: http://lists.freedesktop.org/mailman/listinfo/polkit-devel  
polkit home page: <http://www.freedesktop.org/wiki/Software/polkit>
```

Figure 3.7: Throw an Exception



# Chapter 4

## Looney Tunables (CVE-2023-4911)

- keyword: keywords: glibc, Heap-based buffer overflow, SUID permission
- CVSS v. 3.1: **7.8 HIGH**
- date: 2023/10/03

*A buffer overflow was discovered in the GNU C Library's dynamic loader ld.so while processing the GLIBC\_TUNABLES environment variable. This issue could allow a local attacker to use maliciously crafted GLIBC\_TUNABLES environment variables when launching binaries with SUID permission to execute code with elevated privileges. [2]*

### 4.1 GNU C Library dynamic loader

The GNU C Library 的動態載入器是"find[s] and load[s] the shared objects (shared libraries) needed by a program, prepare[s] the program to run, and then run[s] it" (man **ld.so**). 動態載入程式對安全性極為敏感，因為當本機使用者執行 SUID 程式、SGID 程式或具有功能的程式時，其程式碼會以提升的權限執行。從歷史上看，環境變數（例如 **LD\_PRELOAD**、**LD\_AUDIT** 和 **LD\_LIBRARY\_PATH**）的處理一直是動態載入器中多數漏洞來源。

### 4.2 Analysis

Listing 4.1 是 `__tunables_init()` 的 function 內容。



在執行的一開始，`ld.so` 呼叫 `__tunables_init()` (位於第 279 行)，走訪環境變數 `envp` 尋找 `GLIBC_TUNABLES` 變數 (第 282 行)。對於每個找到的 `GLIBC_TUNABLES`，它會複製此變數 (第 284 行)，呼叫 `parse_tunables()` 來處理和清理此 `new_env` (第 286 行)，最後將原始的替換為這個清理過的 `new_env` (第 288 行)：

```
269 void
270 __tunables_init (char **envp)
271 {
272     char *envname = NULL;
273     char *envval = NULL;
274     size_t len = 0;
275     char **prev_envp = envp;
276     ...
279     while ((envp = get_next_env (envp, &envname, &len, &envval,
280                                     &prev_envp)) != NULL)
281     {
282         if (tunable_is_name ("GLIBC_TUNABLES", envname))
283         {
284             char *new_env = tunables_strdup (envname);
285             if (new_env != NULL)
286                 parse_tunables (new_env + len + 1, envval);
287             /* Place the updated envval. */
288             *prev_envp = new_env;
289             continue;
290         }
```

Listing 4.1: `__tunables_init()` function

Listing 4.2 是 `parse_tunables()` function，在此處理 `tunables` 的 parsing。

`parse_tunables()` 的第一個參數 `tunestr` 指向 `GLIBC_TUNABLES` 的即將被清理複製值，而第二個參數 `valstring` 指向原本在堆疊中的 `GLIBC_TUNABLES` 環境變數。為了清理 `GLIBC_TUNABLES` 的複製值，形式為 `"tunable1=aaa:tunable2=bbb"`，`parse_tunables()` 會從 `tunestr` 中刪除所有危險的 `tunables(SXID_ERASEtunables)`，但保留 `SXID_IGNORE` 和 `NONE` `tunables` (位於第 221-235 行)：

```
162 static void
```

```
163 parse_tunables (char *tunestr, char *valstring)
164 {
165     ...
166     char *p = tunestr;
167     size_t off = 0;
168
169     while (true)
170     {
171         char *name = p;
172         size_t len = 0;
173
174         while (p[len] != '=' && p[len] != ':' && p[len] != '\0')
175             len++;
176
177         if (p[len] == '\0')
178         {
179             if (__libc_enable_secure)
180                 tunestr[off] = '\0';
181             return;
182         }
183
184         if (p[len] == ':')
185         {
186             p += len + 1;
187             continue;
188         }
189
190         p += len + 1;
191
192         char *value = &valstring[p - tunestr];
```

```
199     len = 0;
200
201     while (p[len] != ':' && p[len] != '\0')
202         len++;
203
204
205     for (size_t i = 0; i < sizeof (tunable_list) / sizeof (
        tunable_t); i++)
206     {
207         tunable_t *cur = &tunable_list[i];
208
209         if (tunable_is_name (cur->name, name))
210             {
211                 ...
212
213                 if (__libc_enable_secure)
214                     {
215                         if (cur->security_level !=
TUNABLE_SECLEVEL_SXID_ERASE)
216                             {
217                                 if (off > 0)
218                                     tunestr[off++] = ':';
219
220                                 const char *n = cur->name;
221
222                                 while (*n != '\0')
223                                     tunestr[off++] = *n++;
224
225                                 tunestr[off++] = '=';
226
227                                 for (size_t j = 0; j < len; j++)
228                                     tunestr[off++] = value[j];
229                             }
230
231                         if (cur->security_level !=
```

```
TUNABLE_SECLEVEL_NONE)
236             break;
237         }
238
239         value[len] = '\0';
240         tunable_initialize (cur, value);
241         break;
242     }
243 }
244
245     if (p[len] != '\0')
246         p += len + 1;
247 }
248 }
```

Listing 4.2: `parse_tunables()` function

不幸的是，如果 `GLIBC_TUNABLES` 環境變數的形式是 `"tunable1=tunable2=AAA"`（其中 `"tunable1"` 和 `"tunable2"` 是 `SXID_IGNORE` tunables，例如 `"glibc.malloc.mxfast"`），則：

在 `parse_tunables()` 中的 `while (true)` 的第一次迭代中，`"tunable1=tunable2=AAA"` 被整個複製到 `tunestr`（在第 221-235 行），從而填滿了 `tunestr`；因為在第 203-204 行沒有找到 `':'`，所以在第 247-248 行，`p` 沒有增加；因此 `p` 仍然指向 `"tunable1"` 的值，即 `"tunable2=AAA"`；

在 `parse_tunables()` 中的 `while (true)` 的第二次迭代中，`"tunable2=AAA"` 被添加（彷彿它是第二個 tunable）到 `tunestr`，此時的 buffer 已經滿了，從而溢出了 `tunestr`。

## 4.3 Exploitation

這個漏洞是一個直接的緩衝區溢位，但是為了實現任意程式碼執行，應該覆寫什麼呢？此溢出的緩衝區是由 `tunables_strdup()` 在第 284 行分配的，這是 `strdup()` 的重新實作，它使用了 `ld.so` 的 `__minimal_malloc()` 而不是 `glibc` 的 `malloc()`，而 `glibc` 的 `malloc()` 沒有被初始化。這個 `__minimal_malloc()` 實現只是簡單地呼叫

`mmap()` 來從核心獲取更多記憶體。

那麼問題來了，在 `mmap` 區域中，應該覆寫哪些可寫分頁呢？據此只有兩個選擇，因為這個緩衝區溢出發生在 `ld.so` 的執行的最開始。

`ld.so` 本身的可讀寫 ELF 段，實際上這個可讀寫段的前幾頁是 `ld.so` 的 RELRO 段，但它們還沒有使用 `mprotect()` 設為唯讀：

Figure 4.1 是這些記憶體的映射，並標示出需要覆寫的頁面。藍色矩形表示唯讀段 (`r-p`)；綠色矩形表示可執行段 (`r-xp`)；紅色矩形表示可寫段 (`rw-p`)；虛線框標示需要覆寫的可寫段 (`rw-p`)。

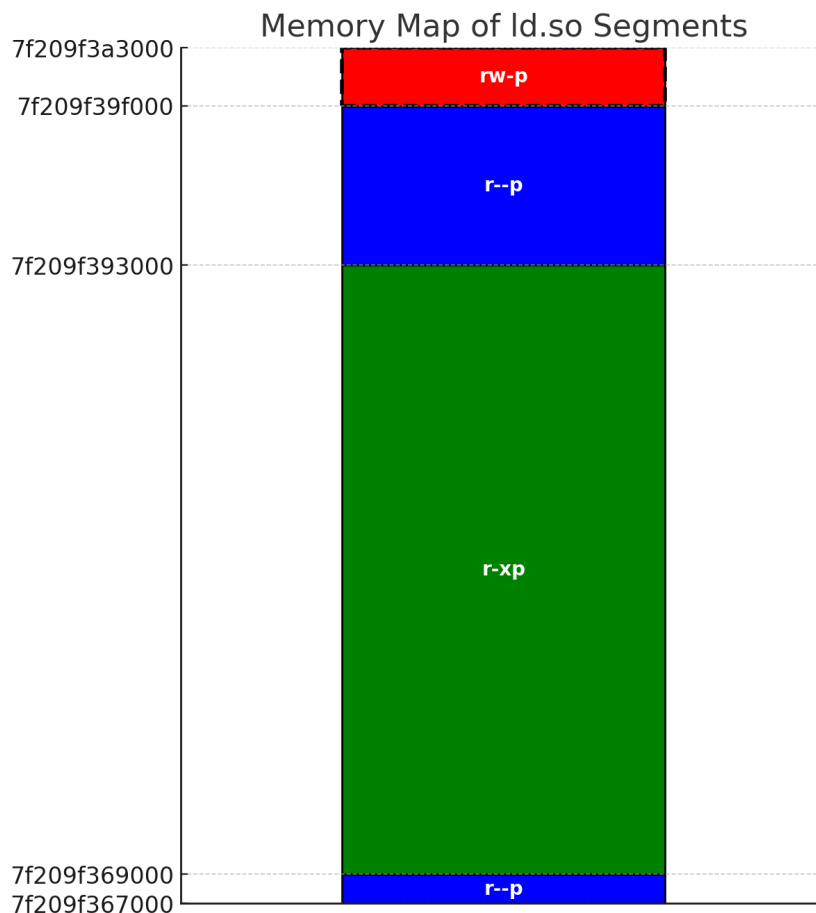


Figure 4.1: Memory map of `ld.so` Segments

然而，檢查的所有 Linux 發行版中，`ld.so` 的可讀寫段下方的未映射空間最多只有一頁，但是 `ld.so` 的 `__minimal_malloc()` 總是分配至少兩頁（為了減少 `mmap` 呼叫次數而多分配一頁）。換句話說，溢出的緩衝區不能立即分配到 `ld.so` 的可讀寫段下方，因此無法覆寫該段。

此處唯一的選擇是覆寫由 `tunables_strdup()` 本身分配的 `mmap()` 分頁：因為 `__tunables_init()` 可以處理多個 `GLIBC_TUNABLES` 環境變數，且由於 Linux 核心的

`mmap()` 是從上向下分配的，在此可以 `mmap()` 第一個 `GLIBC_TUNABLES`（而不溢出），然後 `mmap()` 第二個 `GLIBC_TUNABLES`（直接在第一個下方），再溢出它，從而覆寫第一個 `GLIBC_TUNABLES`。下述是兩個可能可行之方法：

1. 用一個完全不同的環境變數（例如 `LD_PRELOAD` 或 `LD_LIBRARY_PATH`）替換這第一個 `GLIBC_TUNABLES`，但這些危險的變數在稍後被 `ld.so` 在 `process_envvars()` 中移除，因此這樣的替換將是無意義的；
2. 或者，用一個包含危險 `SXID_ERASE` 可調參數的 `GLIBC_TUNABLES` 替換第一個，這些參數在 `parse_tunables()` 中先前被刪除。此法似乎很有希望，但利用這樣的替換需要一個 SUID-root 程式，該程式以 `setuid(0)` 並以保留的環境（以 root 身份處理危險的 `GLIBC_TUNABLES`，但不啟用 `__libc_enable_secure`）執行另一個程式。

不幸的是，在 Linux 上，本文不知道這樣的 SUID-root 程式（在 OpenBSD 上，`/usr/bin/chpass` 以 `setuid(0)` 執行，並 `execv()` 另一個程式 `/usr/sbin/pwd_mkdb`，在 CVE-2019-19726 中被利用）。

此時，情況看起來相當絕望，但是 `ld.so` 的 `_dl_new_object()` 中的一個註釋引起了本文關注（在第 105 行）：

```
56 struct link_map *
57 _dl_new_object (char *realname, const char *libname, int type,
58                struct link_map *loader, int mode, Lmid_t nsid)
59 {
60     ..
61
62     struct link_map *new;
63     struct libname_list *newname;
64     ..
65
66     new = (struct link_map *) calloc (sizeof (*new) + audit_space
67                                     + sizeof (struct link_map *)
68                                     + sizeof (*newname) +
69                                     libname_len, 1);
70     if (new == NULL)
71         return NULL;
72 }
```

```

98  new->l_real = new;
99  new->l_symbolic_searchlist.r_list = (struct link_map **) ((char
    *) (new + 1)
100                                     +
    audit_space);
101
102  new->l_libname = newname
103      = (struct libname_list *) (new->l_symbolic_searchlist.r_list
    + 1);
104  newname->name = (char *) memcpy (newname + 1, libname,
    libname_len);
105  /* newname->next = NULL;      We use calloc therefore not
    necessary. */

```

`ld.so` 使用 `calloc()` 為這個 `link_map` 結構分配記憶體，因此並未顯示初始化其各個成員為零；這是一個合理的優化。正如之前提到的，這裡的 `calloc()` 不是 `glibc` 的 `calloc()`，而是 `ld.so` 的 `__minimal_calloc()`，它呼叫 `__minimal_malloc()` 而不是將返回的記憶體顯示初始化為零；這也是一個合理的優化，因為就所有目的而言，`__minimal_malloc()` 總是回傳一個由 `kernel` 初始化為零的乾淨的 `mmap()` 記憶體區塊。

不幸的是，`parse_tunables()` 中的緩衝區溢出允許用非零字串覆寫乾淨的 `mmap()` 記憶體，從而覆寫即將分配的 `link_map` 結構的指標為非 `NULL` 值。這使得完全打破了 `ld.so` 的邏輯，該邏輯假定這些指標是 `NULL`。

本文首先嘗試通過覆寫 `link_map` 結構的 `l_next` 和 `l_prev` 指標（`link_map` 結構的雙向鏈結 `map`）來利用這個緩衝區溢出，但由於在 `setup_vdso()` 中出現了兩個 `assert()` 失敗，這立即導致了 `ld.so` 的中止（在此檢查的所有發行版都編譯了帶有 `assert()` 的 `glibc`，因此也是 `ld.so`）：

```

96 assert (l->l_next == NULL);
97 assert (l->l_prev == main_map);

```

本文後來意識到 `link_map` 結構中有更多指標沒有明確初始化為 `NULL`；特別是在指標陣列 `l_info[]` 中指向 `Elf64_Dyn` 結構的指標。其中，`l_info[DT_RPATH]`，即 Library search path，立即引起本文注意：如果覆寫此指標並控制它指向的位置和內容，那麼可以迫使 `ld.so` 信任擁有的目錄，因此從這個目錄載入自己的 `libc.so.6` 或

LD\_PRELOAD 函式庫，並執行任意程式碼（如果透過 SUID-root 程式運行 `ld.so`）。

覆寫的 `l_info[DT_RPATH]` 應該指向哪裡？

這個問題的簡單答案是：Stack，更確切地說是 Stack 中的環境字串。在 Linux 上，Stack 在一個 16GB 的區域內進行隨機化，而環境字串最多可以占用 6MB (`_STK_LIM / 4 * 3`，在 kernel's `bprm_stack_limits()` 中)：在 16GB/6MB = 2730 次嘗試後，此處有很大的機會猜測到環境字串的地址，在 exploit 中，始終將 `l_info[DT_RPATH]` 覆寫為 `0x7ffdfbf010`，這是隨機化 Stack 區域的中心。在測試中，這種暴力方法在 Debian 上大約需要 30 秒，在 Ubuntu 和 Fedora 上需要約 5 分鐘。

覆寫的 `l_info[DT_RPATH]` 應該存放什麼？

換句話說，在 6MB 的環境字串中，`l_info[DT_RPATH]` 是一個指向小型（16B）`Elf64_Dyn` 結構的指標：

- 一個 `int64_t d_tag`，應該是 `DT_RPATH`（15），但實際上這個值在任何地方都沒有被檢查，因此可以在這裡儲存任何東西；
- 一個 `uint64_t d_val`，它是正在執行的 SUID-root 程式的 ELF 字串表的偏移量（此偏移量參考的字串就是 Library search path 本身）。

在 exploit 中，只是將 6MB 的環境字串填充為 `0xfffffffffffff8`（-8），因為在大多數 SUID-root 程式的字串表的偏移 -8B 的地方，出現了字串 `"\x08"`：這迫使 `ld.so` 信任一個名為 `"\x08"` 的相對目錄（在當前的工作目錄中），因此允許從這個目錄中載入和執行自己的 `libc.so.6` 或 `LD_PRELOAD`，以 root 權限運行。

本文中描述的攻擊方法適用於幾乎所有 Linux 上預設安裝的 SUID-root 程式；其中一些例外情況包括：

- 所有發行版上的 `sudo`，因為它指定了自己的 ELF RUNPATH（`/usr/libexec/sudo`），這會覆蓋 `l_info[DT_RPATH]`；
- 在 Fedora 上的 `chage` 和 `passwd`，因為它們受到特殊的 SELinux 規則保護；
- 在 Ubuntu 上的 `snap-confine`，因為它受到特殊的 AppArmor 規則保護。



雖然 glibc 2.34 受到這個緩衝區溢出的影響，但其 `tunables_strdup()` 使用 `__sbrk()`，而不是 `__minimal_malloc()`（它在 glibc 2.35 中由 commit `b05fae` 引入，`"elf: Use the minimal malloc on tunables_strdup"`）；尚未調查 glibc 2.34 是否是可利用的。

## 4.4 Patch

*elf: Ignore GLIBC\_TUNABLES for setuid/setgid binaries*

The tunable privilege levels were a retrofit to try and keep the malloc tunable environment variables' behavior unchanged across security boundaries. However, CVE-2023-4911 shows how tricky can be tunable parsing in a security-sensitive environment.

Not only parsing, but the malloc tunable essentially changes some semantics on setuid/setgid processes. Although it is not a direct security issue, allowing users to change setuid/setgid semantics is not a good security practice, and requires extra code and analysis to check if each tunable is safe to use on all security boundaries.

It also means that security opt-in features, like aarch64 MTE, would need to be explicit enabled by an administrator with a wrapper script or with a possible future system-wide tunable setting.

Co-authored-by: Siddhesh Poyarekar <siddhesh@sourceware.org>

Reviewed-by: DJ Delorie <dj@redhat.com> [7]

這個 commit 修改了 `dl-tunables.c` 檔案，主要是針對 `parse_tunables()` 函數進行了修改。以下是主要的更改：

### 1. 函數簽名和參數：

- 原函數簽名: `parse_tunables(char *tunestr, char *valstring)`
- 修改後的簽名: `parse_tunables(char *valstring)`

該函數之前接受兩個指向字串的指標 `tunestr` 和 `valstring`，現在只接受 `valstring`。這意味著在呼叫這個函數之前，可能對 `tunestr` 進行了一些處理，或者 `valstring` 可能是 `tunestr` 的一部分。

## 2. 迴圈結構修改：

- 將原來的 `while (true)` 修改成 `while (!done)`.

新的補丁使用了一個 `while` 迴圈，並且透過 `done` 旗標變數來確定何時退出迴圈。

3. 對 `value` 的處理：

- 原來使用 `len` 變數來維護 `value` 的長度，現在直接使用指標 `p`。

修改後，直接使用指標 `p` 來追蹤 `value` 的開始，而不再使用 `len` 變數。這可能使得程式碼更加簡潔，且容易維護。

## 4. 結束條件的更改：

- 原本是 `if (p[len] == '\0')`，更改成 `if (*p == '\0')`。

改變了結束條件的寫法，使用了指標 `p`，使得條件更符合實際的邏輯。

5. 將 `':'` 替換成 `'\0'`：

- 在迴圈內，將 `':'` 替換成 `'\0'`。

這樣的修改可能是為了更好地處理字串，確保它們以 `'\0'` 結束，以便後續的處理。

Listing 4.3 是將兩份程式碼進行 `diff -git a/elf/dl-tunables.c b/elf/dl-tunables`，分析更新過後的程式碼刪減與差異。

```
+parse_tunables (char *valstring)
{
- if (tunestr == NULL || *tunestr == '\0')
+ if (valstring == NULL || *valstring == '\0')
    return;

- char *p = tunestr;
- size_t off = 0;
+ char *p = valstring;
+ bool done = false;
```

```
- while (true)
+ while (!done)
    {
        char *name = p;
-        size_t len = 0;

        /* First, find where the name ends. */
-        while (p[len] != '=' && p[len] != ':' && p[len] != '\0')
-            len++;
+        while (*p != '=' && *p != ':' && *p != '\0')
+            p++;

        /* If we reach the end of the string before getting a valid
        name-value
           pair, bail out. */
-        if (p[len] == '\0')
+        if (*p == '\0')
            break;

        /* We did not find a valid name-value pair before
        encountering the
           colon. */
-        if (p[len] == ':')
+        if (*p == ':')
        {
-            p += len + 1;
+            p++;
            continue;
        }

-        p += len + 1;
+        /* Skip the '='. */
+        p++;
```

```
-    /* Take the value from the valstring since we need to NULL
-    terminate it. */
-    char *value = &valstring[p - tunestr];
-    len = 0;
+    const char *value = p;
-
-    while (p[len] != ':' && p[len] != '\0')
-        len++;
+    while (*p != ':' && *p != '\0')
+        p++;
+
+    if (*p == '\0')
+        done = true;
+    else
+        *p++ = '\0';
```

Listing 4.3: diff --git a/elf/dl-tunables.c b/elf/dl-tunables.c



# Reference

- [1] GNOME. *GNOME*. URL: <https://www.gnome.org/>.
- [2] Red Hat. *CVE-2023-4911*. Oct. 2023. URL: <https://access.redhat.com/security/cve/cve-2023-4911>.
- [3] polkit. *pkexec.c*. Sept. 2023. URL: <https://gitlab.freedesktop.org/polkit/polkit/-/blob/0.120/src/programs/pkexec.c>.
- [4] polkit. *pkexec.c v.121*. Sept. 2023. URL: [https://gitlab.freedesktop.org/polkit/polkit/-/blob/121/src/programs/pkexec.c?ref\\_type=tags](https://gitlab.freedesktop.org/polkit/polkit/-/blob/121/src/programs/pkexec.c?ref_type=tags).
- [5] Qualys. *Qualys Security Advisory*. URL: <https://www.qualys.com/2022/01/25/cve-2021-4034/pwnkit.txt>.
- [6] wikipedia. *Polkit*. Nov. 2023. URL: <https://en.wikipedia.org/wiki/Polkit>.
- [7] Adhemerval Zanella. Nov. 2023. URL: <https://sourceware.org/git/?p=glibc.git;a=commit;h=9c96c87d60eafa4d78406e606e92b42bd4b570ad>.