Università di Catania
Dipartimento di Matematica e Informatica

Project - Internet Security

M6: Insecure Authorization

Sergio Mancini - 1000022352

Academic year 2022-2023

1 Introduction

The OWASP Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members.

This project concerns position 6 on the "OWASP Top 10 Mobile Risks 2016" list, entitled "M6: Insecure Authorization". This list identifies the top threats and vulnerabilities related to mobile applications.

2 Authentication vs Authorization

authentication is the validation of a user's identity before they can access the system, or their account. Some forms of authentication can be:

- **Basic authentication**: Enter your username and password to access your account.
- **Two-factor authentication**: In addition to entering the username and password, a PIN is required which is sent via message or email, or it can be requested via biometrics.

Figure 1: Authentication

authorization, on the other hand, is the process of providing someone with the ability to access a resource, or permission to perform certain operations. Some forms of permissions can be:

- **Role-based access control (RBAC)**: RBAC allows access to different tiers of information depending on user roles, some users will have higher levels of clearance than others, they'll be able to view data that others can't.
- **Attribute-based access control (ABAC)**: Abac, relies more heavily on a user's attributes to grant authorization. These attributes may include

a person's security clearance, the file's owner, the type of action desired (such as viewer vs. editor), and the location of an access attempt. When the request doesn't meet a company's approved characteristics, the system deny access.

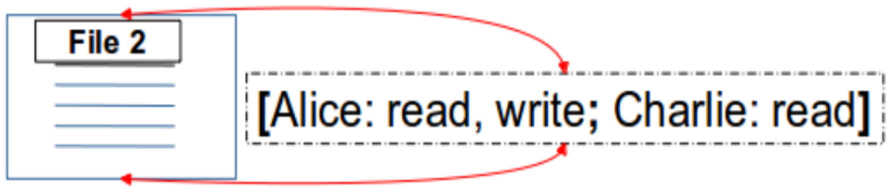


Figure 2: Example of authorization - ACL

The difference between authentication and authorization is clear: while authentication authenticates a person's identity before allowing successful login, authorization authorizes their ability to access a specific file, or resource.
Authentication requires the user to act, but authorization puts the onus on the server or site.

3 Dirty Pipe

Dirty Pipe (also known as CVE-2022-0847) is a very significant security vulnerability, discovered by Max Kellermann in April 2021, involving version 5.8 of the Linux kernel. This vulnerability affects the Linux kernel and its successful exploitation allows the attacker to perform a local privilege escalation and gain unauthorized access to sensitive data.
The vulnerability arises from the incorrect UNIX pipe handling which allowed the attackers to overwrite the files on the system with arbitrary data (modifying sensitive files potentially including root passwords and SUID binaries).

Kellermann explained: "To exploit this vulnerability, you need to: 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 (leaving the flag set in all struct pipe_buffer instances on the struct pipe_inode_info ring), splice data from the target file (opened with O_RDONLY) into the pipe from just before the target offset [and] write arbitrary data into the pipe".

This data will overwrite the cached file page instead of creating a new anonymous struct pipebuffer because PIPE_BUF_FLAG_CAN_MERGE is set.
The pipe flag "PIPE_BUF_FLAG_CAN_MERGE", signifies that the data buffer inside the pipe can be merged, this flag notifies the kernel that the changes

which are written to the page cache pointed to by the pipe shall be written back to the file that the page is sourced from.

3.1 What Are Pipes?

Pipes provide a unidirectional interprocess communication channel. A pipe has a read end and a write end. Data written to the write end of a pipe can be read from the read end of the pipe. Example of a pipe:

```
$ echo hello | wc -c
$ 6
```

The output of the first process, is passed into the pipe, which is later used by the second process as input.

3.2 What Is Page Cache?

The page cache memory is a form of cache memory used in operating systems to accelerate access to data stored on the hard disk or other storage devices. Its main function is to store frequently requested data pages in memory in order to reduce access times to the data.

When a program performs read or write operations on a storage device, such as a hard disk, the operating system can store a copy of the read or written data in the page cache.

A page that is modified in the cache and not yet updated in secondary memory (resulting in the two copies being different) is referred to as a ”dirty page”.
This is partly responsible for its resemblance in the vulnerability nickname ”Dirty Pipe”.

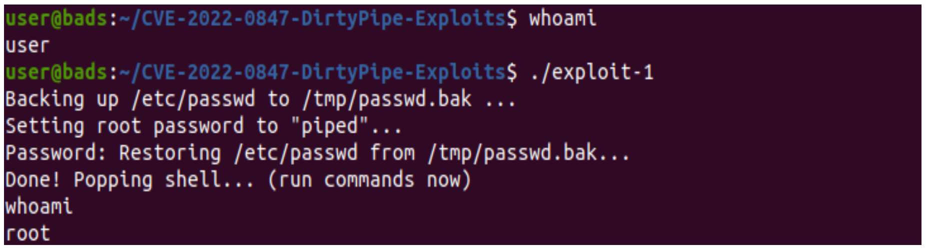
3.3 Exploit 1: Explanation

The first exploit does modify the **/etc/passwd** file, placing a password of ”piped” as the root password, and drop an elevated root shell. Before doing this, it will create a backup named ”passwd.bak” and after exiting the shell it will restore the backup.
This vulnerability is caused due to an uninitialized pipe buffer flag variable, which essentially overwrites any file content in the page cache even if the file is not permitted to be written or if it's a read-only file, this because the page cache is always writable by the kernel, and writing to a pipe never checks any permission because whenever you use a pipe to redirect output, the Linux kernel already assumes that the user using the pipe already has permissions.

The fact that writing to a pipe never checks any permissions, allows the attackers to perform a privilege escalation by essentially overwriting the data in the read-only files and injecting code or data from an unprivileged process into a privileged process.

3.3.1 Summary of Exploit 1

1. Backs up the /etc/passwd file to /tmp/passwd.bak
2. Set the root account password to ”piped” using the date variable that contains the new password string.
3. Opens the read-only input file using the open function and verifies the specified offset.
4. Prepare a pipe where all the ”bufs” in the pipe_inode_info ring have the PIPE_BUF_FLAG_CAN_MERGE flag set using the prepare_pipe function.
5. Uses the splice function to move a byte from the input file into the pipe, adding a reference to the page cache. This exploits the lack of initialization of ”flags” in the copy_page_to_iter_pipe function.
6. Uses the write function to write the specified data to the pipe. Because of the PIPE_BUF_FLAG_CAN_MERGE flag, data is written directly to the page cache instead of creating new pipe buffers.
7. Executes the execv command to run a shell script that restores the /etc/passwd file from the backup copy, allowing shell access with root privileges.



```
user@bads:~/CVE-2022-0847-DirtyPipe-Exploits$ whoami
user
user@bads:~/CVE-2022-0847-DirtyPipe-Exploits$ ./exploit-1
Backing up /etc/passwd to /tmp/passwd.bak ...
Setting root password to "piped"...
Password: Restoring /etc/passwd from /tmp/passwd.bak...
Done! Popping shell... (run commands now)
whoami
root
```

Figure 3: User drop shell - Exploit 1

3.4 Exploit 2: Explanation

The second exploit can be used to inject and overwrite data in read-only SUID process memory that run as root.
This Dirty Pipe hijacks a SUID binary to drop a root shell.

There are two limitations of this exploit:

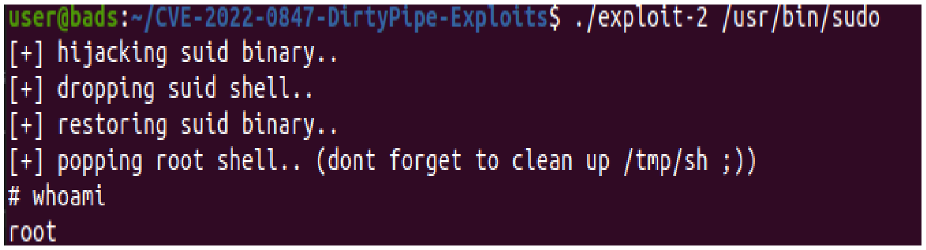
1. The offset cannot be on a page boundary, because it needs to write one byte before the offset, to add a reference to this page to the pipe.

2. The write cannot cross a page boundary

To find all the SUID processes that run as root, you can use this command that prints them on the screen:

```
$ find / -perm -4000 2>/dev/null
```

After choosing the process, for example "/usr/bin/sudo", simply run the second exploit and pass this process as the second argument. After that a shell will be launched.



```
user@bads:~/CVE-2022-0847-DirtyPipe-Exploits$ ./exploit-2 /usr/bin/sudo
[+] hijacking suid binary..
[+] dropping suid shell..
[+] restoring suid binary..
[+] popping root shell.. (dont forget to clean up /tmp/sh ;))
# whoami
root
```

Figure 4: User drop shell - Exploit 2

3.4.1 Summary of Exploit 2

1. Defines an array of "elfcode" bytes that represents a small ELF file. This ELF file is created in the next part of the code and is used to achieve the execution of a shell with root privileges.

2. Prepare a pipe where all the "bufs" in the pipe_inode_info ring have the PIPE_BUF_FLAG_CAN_MERGE flag set using the prepare pipe function.

3. The "hax" function takes as input the name of a file, an offset, the data and their length. The function opens the specified file as read-only.

4. A "splice" is performed that moves a byte before the offset specified in the original file in the pipe. This adds a reference to the page cache, but since the "copy_page_to_iter_pipe()" function does not initialize the "flags" flag, the "PIPE_BUF_FLAG_CAN_MERGE" flag remains set.

5. A "write" is performed on the pipe that overwrites the contents of the file with the provided data. Because of the "PIPE_BUF_FLAG_CAN_MERGE" flag set earlier, this operation writes directly to the page cache instead of creating a new "pipe_buffer".

4 Crack /etc/shadow

/etc/shadow is a text file that contains information about the system's users' passwords. It is owned by user root and group shadow.
This file contains one entry per line, each representing a user account, typically the first line describes the root user, followed by the system and normal user accounts. New entries are appended at the end of the file.

By using one of the two previous exploits to obtain a root shell on a compromised system, you can perform additional unauthorized actions, such as installing additional tools such as "John the Ripper"[1] to crack passwords in the "/etc/shadow" file.

To try to crack password hashes you need to run these commands:

1. Perform one of the two exploits.
2. Install John the Ripper

   ```
   $ apt-get install john
   ```

3. Copy the /etc/passwd and /etc/shadow

   ```
   $ cp /etc/passwd passwd.txt
   $ cp /etc/shadow shadow.txt
   ```

4. Combine the contents of the /etc/passwd and /etc/shadow files and write them to a file

   ```
   $ unshadow passwd.txt shadow.txt > john-input.txt
   ```

5. `$ john --format=crypt john-input.txt`

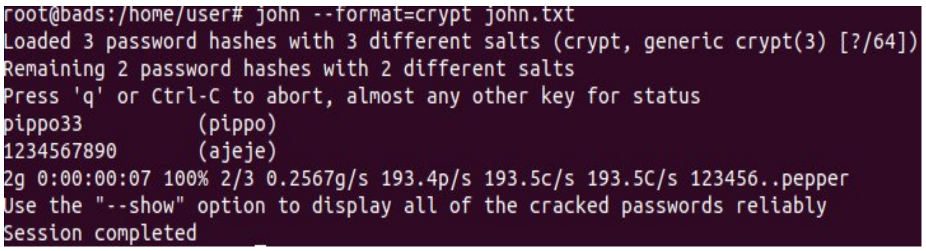
```
user:$6$AiN6kGo/IqC8ovEV$piOhACq...:19536:0:99999:7:::
```

The entry above contains information about the user "user" password (is truncated for better readability):

- "user": This is the username associated with the password hash.
- $6$: This part of the hash indicates the hashing algorithm used. In this specific case, the SHA-512 algorithm for password encryption.

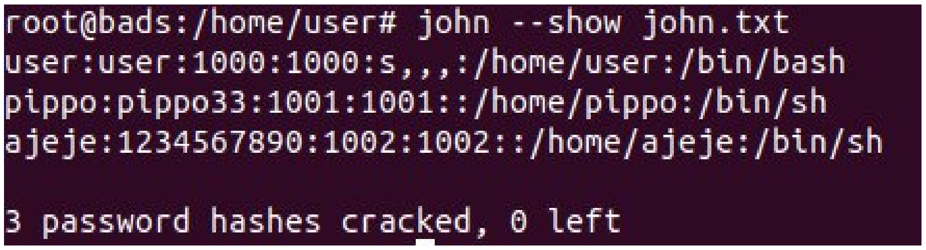
[1] John the Ripper is a powerful password cracking program, widely used by security experts and system administrators to test password strength. It uses different techniques, such as brute force and dictionary.

- ”AiN6kGo/IqC8ovEV”: This is the part of the hash that represents the encrypted password.
- ”piOhACq”...: This part of the hash represents salt, which is a random value added to the password before applying the hashing algorithm.
- ”19536”: This value represents the number of iterations of the hashing algorithm. The higher the number of iterations, the longer it will take to verify the password, making a brute force attack more difficult.
- ”0”: This value represents the day the password was last changed. In this case the date is unavailable or has not been recorded.
- ”99999”: This value represents the minimum number of days before the password can be changed.
- ”7”: This value represents the maximum number of days after which the password must be changed.
- ”:::” This part represent additional parameters.



```
root@bads:/home/user# john --format=crypt john.txt
Loaded 3 password hashes with 3 different salts (crypt, generic crypt(3) [?/64])
Remaining 2 password hashes with 2 different salts
Press 'q' or Ctrl-C to abort, almost any other key for status
pippo33          (pippo)
1234567890       (ajeje)
2g 0:00:00:07 100% 2/3 0.2567g/s 193.4p/s 193.5c/s 193.5C/s 123456..pepper
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Figure 5: John Crack Password



```
root@bads:/home/user# john --show john.txt
user:user:1000:1000:s,,,:/home/user:/bin/bash
pippo:pippo33:1001:1001::/home/pippo:/bin/sh
ajeje:1234567890:1002:1002::/home/ajeje:/bin/sh

3 password hashes cracked, 0 left
```

Figure 6: John Show Password

5 How To Run The Exploits

The first step should be to verify the kernel version on the system, because starting from the Linux kernel version 5.8 and later are vulnerable to these exploits. However, certain version have received fixes to address the vulnerabilities, for example:

- 5.16.11
- 5.15.25
- 5.10.102

If the kernel version is not vulnerable, you must downgrade to an earlier version.

Once you have confirmed whether your kernel version is vulnerable, proceed to compile and execute one of the two provided scripts, with:

```
$ ./compile.sh
$ ./exploit-1
```

Or

```
$ ./exploit-2 <SUID process memory that run as root>
```

6 Impact On Authorization

The use of a Dirty Pipe by a user who may not be included in the list of sudoers and who manages to obtain a root shell can have a significant impact on the authorization of a system.

This means that an unauthorized user can aquire root privileges and gaining complete control of the system and the ability to perform any operation, including even the most dangerous ones.

Users who don't have root privileges should not be able to perform commands or operations reserved for those users. However, if a user manages to obtain a root shell, these permission checks are bypassed.

7 Mitigations

To effectively mitigate the use of a Dirty Pipe and protect the system from unauthorized access with root privileges, one of the key measures is to kepp the system constantly updated to the latest version avaible.

Instead, to mitigate the problem of cracking the passwords contained in "/etc/shadow", you must use more complex passwords and not present in the wordlists, to avoid a possible dictionary attack. Software updates include security patches that fix known vulnerabilities.

8 Conclusion

In conclusion, the provided text focusing on the sixth position in the "OWASP Top 10 Mobile Risks 2016" list, which is titled "M6: Insecure Authorization." The distinction between authentication and authorization is explained. The text then delves into the Dirty Pipe vulnerability (CVE-2022-0847) discovered in the Linux kernel, which allows local privilege escalation and unauthorized access to sensitive data.

The Dirty Pipe vulnerability arises from incorrect UNIX pipe handling, enabling attackers to overwrite files on the system with arbitrary data, potentially compromising root passwords and SUID binaries. The text describes the two provided exploits, explaining their mechanisms and potential consequences.

Exploit 1 focuses on modifying the "/etc/passwd" file, changing the root password and dropping an elevated root shell. The lack of permissions checks during writing to a pipe allows attackers to escalate privileges and overwrite data in read-only files.

Exploit 2 targets read-only SUID process memory running as root, hijacking a SUID binary to drop a root shell. The limitations of this exploit are outlined, and instructions are given for identifying SUID processes and executing the exploit.

Furthermore, the text briefly mentions the possibility of cracking password hashes stored in the "/etc/shadow" file using tools like "John the Ripper" after obtaining a root shell through one of the exploits.

Overall, the text provides an overview of the Insecure Authorization vulnerability, highlights the Dirty Pipe vulnerability and its exploitation methods, and briefly touches on password cracking as a potential consequence of unauthorized access.

9 Bibliography

- ”M6: Insecure Authorization”:
  https://tinyurl.com/49573h74
- ”CVE-2022-0847”:
  https://jfrog.com/blog/dirtypipe-cve-2022-0847-the-new-dirtycow/
- ”Max Kellerman”
  https://dirtypipe.cm4all.com/
- ”Dirty-Pipe-Explained”:
  https://www.hackthebox.com/blog/Dirty-Pipe-Explained-CVE-2022-0847#introduction
- ”Exploit”:
  https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits
- ”John The Ripper”:
  https://www.openwall.com/john/
- ”/etc/shadow”
  https://linuxize.com/post/etc-shadow-file/