Easy Path

VM Setup and How to Exploit It

Firstly, we set up an FTP server with the possibility to log in as an anonymous user. In particular, FTP is used to upload files on an Apache server. Due to the fact that there are no checks on the type of file you can upload, it's possible to upload a PHP Web Shell on the server and to run it in order to gain access to the server. To make PHP scripts executable, we installed php7.4 on the system and enabled it on Apache (a2enmod php).
Since we modified the configuration files of Apache to make a specific user run the Apache server, the attacker will get access to the server as this user (i.e., not www data).
Now, by typing " `sudo -l` ", the attacker would notice he is able to run a sudo command ("find", which is a binary) without the need for a password, since we edited the sudoers file to allow him execute this command. By searching on GTFOBins website, he would discover the command he should execute to open a root shell without any password needed.

Why is it realistic?

Requiring users to enter a password every time they run a sudo command can be inconvenient, especially for frequently used commands, such as the "find" command. Allowing some commands to be executed without a password can improve productivity.

Intermediate Path

VM Setup and How to Exploit It

We set up a WordPress site (latest version) that runs on an Apache server. Then, there is a MySQL DataBase to manage the website data. For this path, we decided to introduce a vulnerability with a known exploit: CVE-2023-3460. It consists of an outdated plugin, called "Ultimate Member", which manages the users registration and login pages, directly interacting with the DB. During the registration phase, by easily adding a parameter in the request body ("`&wp_càpabilities[administrator]=1`"), using the BurpSuite proxy feature to intercept the request, we can register a new user as admin of the WordPress website.

Then, we created an image upload page usable only by admin users, which weakly checks the content type of the header of the request. In fact, it is possible for the attacker to upload a PHP web shell, bypassing the filter by intercepting the request with BurpSuite and changing the content-type header of the request to "image/jpeg". Since the path where uploaded files are stored (".../wp-content/uploads/2024/04/") is accessible from the web server, the attacker could execute the Web Shell and gain access to the server as "www-data".

Then, we activated an NFS server (nfs-kernel-server) and modified the /etc/exports file to disable "root_squashing" on "/tmp*". The attacker could use this feature to upload a C script on the shared directory. The script should contain something like "`int main() { setgid(0); setuid(0); system("/bin/bash"); return 0; }`", which changes the current user to root. The C file must be compiled from the victim machine to avoid library incompatibilities with Kali Linux. The executable file should have
root:root owner and all permissions allowed, including the SUID bit, and this is
possible from the attacker machine. Finally, running the script from the victim
machine, the attacker gains root privileges.

Why is it realistic?

Not updating services, or WordPress plugins in this case, are common weaknesses that often lead to vulnerabilities.

The image upload page is realistic because it’s based on a poorly written filter, which is easily exploitable.

Disabling root squashing on NFS can be beneficial in environments where system administrators need elevated privileges to manage shared files.

Hard Path

VM Setup and How to Exploit It

We set up an Apache server with some simple html pages and one php page, installed php7.4-fpm on the system, and enabled it on Apache. We edited the php.ini file, specifying “/var/lib/php/sessions” as “session.save_path” and gave 777 permissions to “/var/lib/php/sessions”, in order to make it accessible and executable from the web server. We also changed this directory owner to the user that runs the Apache Server. The php web page contains this PHP script:

```php
<?php
session_start();
if(isset($_GET['page'])){
   $_SESSION['page'] = $_GET['page'];
   echo "You're currently in" . $_GET["page"];
   include($_GET['page']);
}
?>
```

This gives the possibility to perform a PHP session file attack. Indeed, the attacker should add to the index.php URL something like: " `.../index.php?page=<?php if(isset($_REQUEST["cmd"])){ echo "<pre>"; $cmd = ($_REQUEST["cmd"]); system($cmd); echo "</pre>"; die; }?>` ", in order to write a Web Shell script inside the current PHP session file assigned to the attacker. Then, by inspecting the web page he can find the session file. By navigating to the path where its session file is stored, he can execute commands on the victim server by typing “ `...index.php?page=/var/lib/php/sessions/sess_#########&cmd=…` ” and get access to the server.
The attacker has got access to a specific user of the system. In its home directory, we placed a vulnerable C file already compiled (for 32-bit) and not editable. The executable file has the owner root:root and, as permissions, it is executable and hasthe SUID bit.
The C file:

```c
#include <stdio.h>
#include <string.h>
void permission_check() {
   char input[40];
   gets(input);
   if(strstr(input,"admin"))
       printf("Successful check \n");
   else
       printf("Error during check, try again \n");
}
// WARNING: Obsolete function: do not use
void login_func() {
   printf("---Login---");
   setuid(1002);
   setgid(1002);
}
void main() {
   printf("Welcome to the system. Please enter your credentials: ");
   permission_check();
}
```

To exploit this vulnerable file (with NX enabled), it is necessary to perform a “ret2libc” attack after having invoked the “login_func()”, which is a secret function since it is not called by the main function. To do this, the attacker needs to find the exact offset to overflow the buffer and overwrite the return address with a malicious payload, gaining control of the program's execution flow. Then, he needs to find the address of the “login_func”, in order to invoke it. Now, the attacker can perform the “ret2libc” attack, building an exploit.py script like the following:

```python
#!/usr/bin/env python3
from pwn import *
p=process('./login')
padding = b'A'*52

libc_base_address=0xf7dcd000
system=p32(libc_base_address+0x41780)
ret=p32(0x0)
shell=p32(libc_base_address+0x18e363)
log_func=p32(0x080492a1)
p.sendline(padding+log_func+system+ret+shell)
print(p.clean)
p.interactive()
```

By exploiting this vulnerability, the attacker will perform lateral movement becoming another non-sudoer user. This user is thought to be the manager of the WordPress DB and in its home directory, there is a file containing its MySQL DB credentials. By using them to access the DB, it’s possible to see the list of users logged in the WordPress website and to notice that there is a certain user logged with the same username as a user of the victim server. The attacker should try to use its WordPress password, found inside the DB, to log in inside the victim server (and it would work!). An obstacle is that passwords are hashed and must be decrypted with tools like John The Ripper. In particular, we set a password that is located inside the rockyou.txt file provided from John The Ripper, in order to make it decryptable in a short amount of time.
Gaining the foothold of this last user, the attacker would discover to be a sudoer and he would gain root privileges.

Why is it realistic?

Misconfigurations related to file permissions and ownership are common problems, especially for low-experienced server administrators. Moreover, it’s possible that developers leave work-in-progress web pages available, where they may introduce vulnerabilities like allowing user-controlled input (`$_GET['page']`) without proper sanitization or commands like “echo "You're currently in" `. $_GET["page"];`”, that are only useful in the development phase.

The buffer overflow-vulnerable C file is realistic because it is believed to be a work-in-progress script for performing logins. One way to change the current user is by modifying the ID. This particular function is insecure and unused in the “main”, but it may have been left there due to laziness or distraction.

The user who leaves his database credentials in his home directory is realistic because it's common to have files containing this kind of information, even though they should be protected. Often, users are lazy and avoid taking precautions in this sense. Another weakness related to distraction and laziness is the one that concerns the user that uses the same credentials to log in both the WordPress website and the victim server. It’s also pretty common to use weak passwords that could be forced using the right tools.

Additional Considerations for All Paths

1. We created a service that is executed at boot, which consists of saving the current IP address of the machine inside the MySQL database of WordPress. This was done because WordPress stores the IP address of the initial website configuration in the database, so it would not have been accessible otherwise.
2. We cleared the history of every user on the system.
3. Every file or directory with 777 permissions is justified by misconfiguration.

4. Each website held by Apache listens on a different port, properly set in the configuration files.

Note: for any issue, these are the credentials of one of the root accounts of the system:

User: `bale_gareth_1`
Password: `#Bale_Golf:1!`

Members:

- Biondi Federico
- Casciani Leonardo
- Di Paola Riccardo
