Abertay University

Web Penetration Test of Astley’s Store

Joe Crichton

CMP319: Web Application Penetration Testing

BSc Ethical Hacking Year 3

2022/23

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

.

Abstract

Businesses operating online can be a target of cybercrime, from phishing attacks to organised, methodical remote attacks on a business's website or other services trying to exploit vulnerabilities. Penetration testing is a valuable way of testing the security of these websites, services or machines from remote attacks by simulating attacks that malicious actors could perform on these systems and reporting potential countermeasures to vulnerabilities. This report contains the results of such a test against a client's web application, Astley's Store, on http://192.168.1.10 of a virtual network.

The OWASP WSTG was used as a basis for selecting which phases penetration testing the web application would be subject to. These phases covered the scanning, enumeration, and exploitation of the web application to test for vulnerabilities and their potential exploits. Significant information leakage about the web applications systems was found, with findings aiding testing and exploitation in later stages. A dangerous HTTP method was found to be allowed. User accounts and sessions were found to be highly insecure. XSS, SQL Injection, CSRF and Clickjacking attacks were evaluated to be all possible. The web application was also identified to communicate with clients only using HTTP.

The web application exhibited many vulnerabilities, predominantly showing user session and account insecurity. It would be extremely easy for an attacker to obtain user account details and potentially extract personal information and payment details. Business operations of the web application could be disrupted, causing monetary damage to the business. Further studies should be undertaken to assess the cookies sent to the client and to understand capability of the SQL injection and admin account vulnerabilities.

.

Contents

.

.

.

1 Introduction

1.1 Background

Cybercrime is a constant threat to companies operating online, with 4 in every 10 businesses reporting an identified cyber-attack in March 2022 over the previous year (Gov.uk, 2022). Of those attacks, 26% were identified as a more sophisticated attack- beyond a social engineering phishing attempt. A third of the businesses reported the attack having an impact on the business.

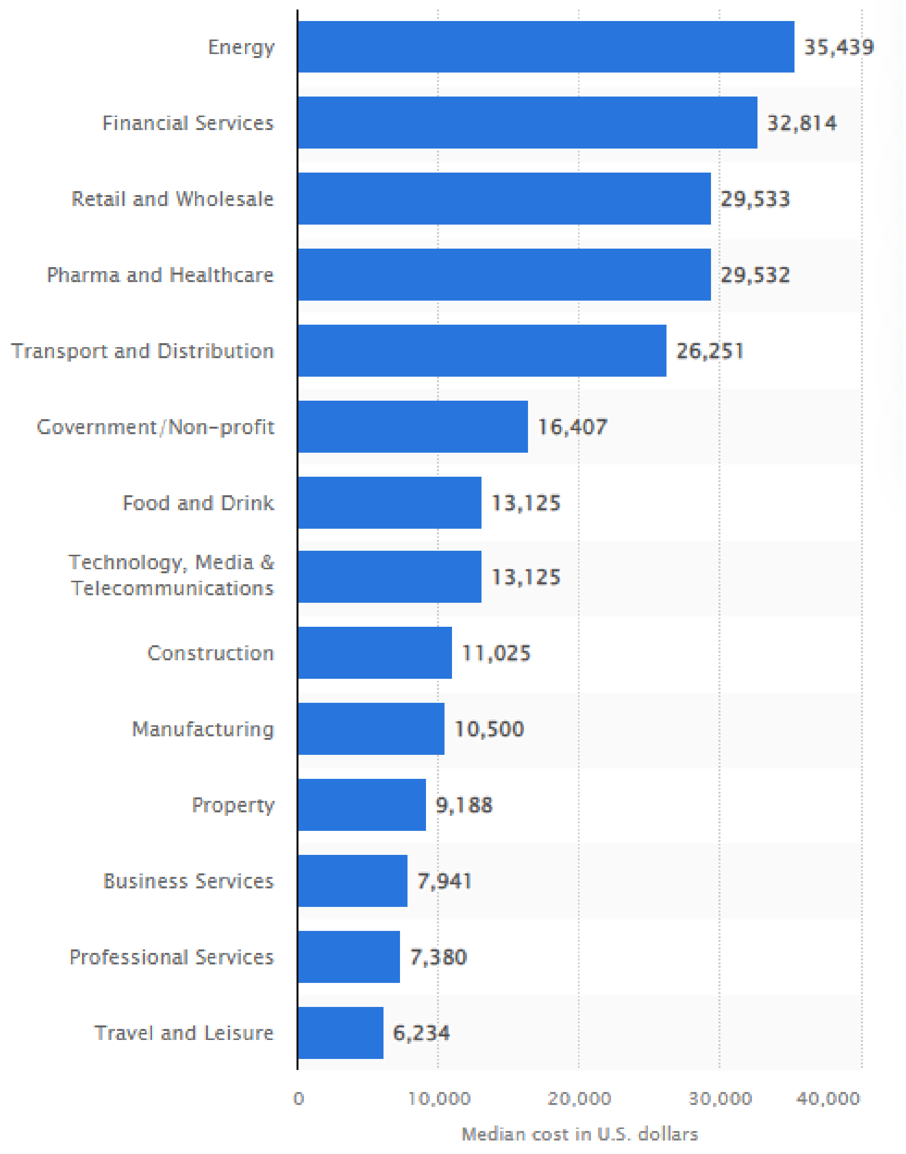


Figure 1-1: Average cost of cyber incidents to organizations in the UK as of 2021, by industry (Statista, 2021)

As shown above in figure 1, the recorded cost of cyber incidents in the UK ranged from $6,000-$35,000 in 2021. It is clear that cyber-crime is a modern-day threat and can range from inconvenient to highly damaging to the operations of businesses or the privacy of individuals. These damages are incurred from business disruption, data theft or even monetary theft, with varying motivation. (IBM, no date.) Data of the business, customer, or anything left on an individual's website can all be a target. Despite this threat, only 17% of UK businesses undertook a “cyber vulnerability audit”- which includes penetration testing or scanning (ISACA, 2016).

Penetration testing is a good practice for evaluating the security of IT infrastructure- hosts, servers, and networks. This is where a penetration tester finds vulnerabilities and then exploits them to simulate and evaluate what damage could be done to a system from the found vulnerabilities. (Cisco, no date). Doing so offers feedback to the system owner, a business or individual, with the intention that the feedback can be used to improve the security of the system and so prevent data breaches or other damage to the system. The intention of a web application test is no different from that of any other penetration test type- to test and evaluate the security of the system so the vulnerabilities can be fixed (Digital Defense, no date). However, the system in question for this type of testing can be complex, make use of multiple applications or servers and may be vulnerable client or server side. This is why a web application test must be conducted rather than a broad and general penetration test, as specific frameworks are used to test vulnerabilities that may be present on web applications.

This type of penetration testing provides an in-depth review of vulnerabilities and the associated exploits of a web application, and any countermeasures that can be deployed to patch these vulnerabilities. While antimalware services or other automated tools can be installed on the web application host machine, specific vulnerabilities present in the web application will not be directly fixed in the way that a professional effort, guided by feedback from a web application penetration test, could. While a comprehensive vulnerability scanner like Nessus can be used to quickly cover the applications attack surface (Tenable, no date), a review of how much damage could be done by exploiting these vulnerabilities and, as before, a review of how to solve these vulnerabilities is most valuable to the business or individual hosting the web application.

This report will provide aims for the test and move on to outline the specific framework used as a methodology with appropriate tools used. The procedure will then be described step-by-step, with the results presented and a discussion based on what was found. References for any citations can be found in the references section of this report.

1.2 Aims

This report aims to identify and evaluate the security and vulnerabilities present in the client's web application, Astley’s Store, with the address http://192.168.1.10/ on a virtual network via a penetration test. Discovered vulnerabilities may be tested by exploitation to measure the severity of the vulnerability. This means that for each vulnerability found, the sub aims are to:

- Evaluate the severity of the vulnerability and its implications; and if applicable, its exploit(s)
- Present potential countermeasures for the vulnerability where necessary

The OWASP Web Application Security Testing framework will be used as a methodology for testing the web application. The framework also specifies which scanning and exploitation tools will be used. The steps this methodology follows, and tools can be found in section 2.1.

Note that the host machine for the web application was out of scope for the web application penetration test.

2 PROCEDURE AND RESULTS

2.1 METHODOLOGY

The OWASP Web Security Testing Guide v4.2 (OWASP, no date. A) was used as the basis for testing the web application. The OWASP WSTG was chosen as it is a comprehensive, up-to-date cybersecurity testing resource that can be used by developers and security professionals alike (PacketLabs, 2021). This framework would provide industry standardised results and ensure a thorough step-by-step approach to testing the web application for vulnerabilities.

Parts of the Web Application Security Testing step of the guide were used as the methodology. This step encompasses many sub-steps, which are grouped into the phases in the table below and have been aligned with the section number in this report.

Section	Phase
2.3	Information Gathering
2.4	Configuration and Deployment Management Testing
2.5	Identity Management Testing
2.6	Authentication Testing
2.7	Session Management Testing
2.8	Input Validation Testing
2.9	Business Logic Testing
2.10	Client-side Testing

Table 1-1: OWASP WTSG Web Application Security Testing Phases Used

Each phase was split into logical steps which the web application was subject to, which are covered in the appropriate section of the procedure.

Phases and steps encompassing the testing of technologies not present on the web application were also excluded. Steps that gave no result that confirmed the lack of, or presented, a countermeasure to a vulnerability were additionally excluded from this report.

The investigator was supplied with a standard user account on Astley's Store- a Mr. Steve Brown with the login credentials hacklab@hacklab.com and hacklab for email and password, respectively. The web application was known to be on 192.168.1.10 of the virtual network. This was the only information supplied to the investigator, along with the fact the web application was "functional", but concern existed that there were vulnerabilities present that could be used to exploit the web application. The investigator was tasked with performing a web penetration test with this information.

This means the penetration test would be black box- where "internal structure or design" (Pedamkar,P. no date) of the web application is unknown. Therefore, as the procedure progressed, it was not possible to execute some steps- e.g. testing moderator privileges, as the tester had not been supplied with moderator login information. These invalid steps have also been excluded from this report.

The investigator used a Kali Linux (2022.4 release) virtual machine to test the virtual web application. Kali Linux is an open-source, Debian-based Linux distribution made for penetration testing, providing common tools for this task (KALI, 2022A). This was ideal for performing the web application penetration test on Astley's Store.

The operating system and tools used in the web application penetration test procedure are included in the table below.

Tool	Version	Description
cURL	7.85.0	Command line tool for internet protocol interaction (Sulcas, A. 2020), used to analyse web application HTTP responses
Nmap	7.93	Network scanner (NMAP, no date) used to determine open ports and services on the web application
Firefox (view-source)	102.5.0	Web browser, with view source function used for manual inspection of web applications code
Burp Suite Community	2022.9.6	Web proxy (Portswigger, no date) used to analyse and modify HTTP traffic from web application
DIRB	2.22	Dictionary based web server scanner (KALI, 2022B) used to scan web application
Nikto	2.1.6	Web server and CGI scanner (KALI, 2022C) used to scan web application
SQLMap	1.6.11	Tool to scan and exploit SQL injection vulnerabilities (KALI, 2022D) used to scan and exploit SQL vulnerabilities in the web application
Mousepad	0.5.10	Default tool on Kali Linux for text editing (Perez, Y. no date)
Cyberchef	9.55.0	Web application tool, used in this test to decode strings

Table 1-2: Tools used in Web Application Test of Astley's Store

The OWASP WSTG and the tools listed above gave structure to the procedure and allowed for comprehensive testing of the web application.

2.2 OVERVIEW OF PROCEDURE

The investigator proceeded with the penetration test by following the OWASP WSTG phases outlined in section 2.1. The results of each step of each phase were recorded and then the investigator proceeded with the next step, until the investigation had concluded. The purpose of each phase is described below:

Phase	Purpose
Information Gathering	Scan and enumerate the web application so further testing can continue. Identify application entry points and find web application structure.
Configuration and Deployment Management Testing	Test web application communication methods, information leakage and handling in files. Enumerate admin interfaces.
Identity Management Testing	Test security of account handling upon creation and role permissions.
Authentication Testing	Test for account authentication weaknesses and sensitive information change weaknesses.
Session Management Testing	Test for session handling with cookies or exposed variables and exploits for these elements.
Input Validation Testing	Test for vulnerabilities where user input can be supplied, and vulnerabilities with HTTP handling.
Business Logic Testing	Test logical disruption of the operation of the online business.
Client-Side Testing	Test for vulnerabilities from client interaction with web application.

Table 2-1: Purpose of OWASP WTSG Phases

The tools described in section 2.1 were used as appropriate- the OWASP WSTG described what software and tools were needed to perform the testing for that step, or what capabilities were required to perform the testing for that step.

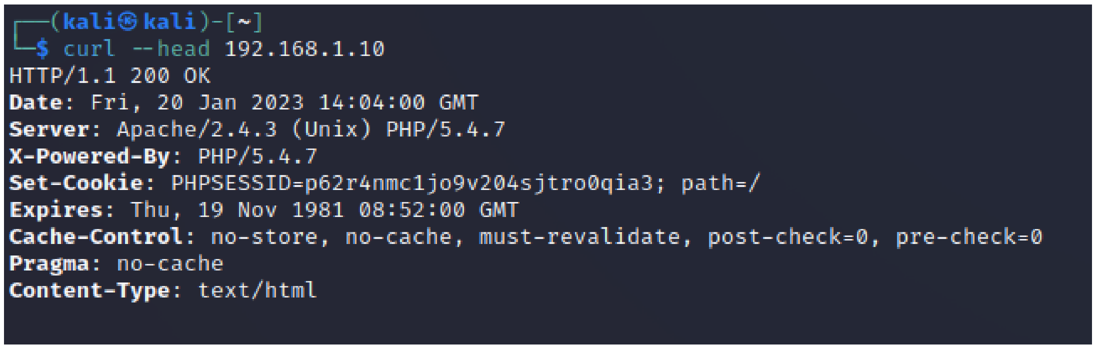
2.3 Information Gathering

2.3.1 Fingerprint Web Server

The version and type of web server needed to be determined to allow for further progress in discovering any vulnerabilities. This was done by banner grabbing with the cURL tool and the command:

curl –head 192.168.1.10

Below in figure 2-1 is the HTTP response from the web application revealing the version and type of web server.



```
┌──(kali㉿kali)-[~]
└─$ curl --head 192.168.1.10
HTTP/1.1 200 OK
Date: Fri, 20 Jan 2023 14:04:00 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: PHPSESSID=p62r4nmc1jo9v204sjtro0qia3; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Type: text/html
```

Figure 2-1: HTTP Response Header

The web server was found to be Apache 2.4.3.

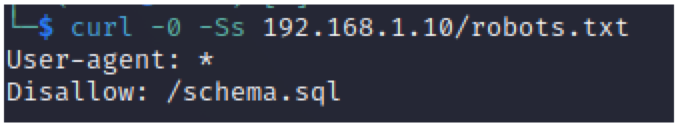
2.3.2 Review Webserver Metafiles for Information Leakage

This test attempted to discover metadata files for and information about the web applications paths or other information of interest.

Robots.txt was found with the command:

Curl –0 –Ss 192.168.1.10/robots.txt

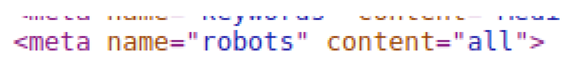
as shown below in figure 2-2.



```
└─$ curl -0 -Ss 192.168.1.10/robots.txt
User-agent: *
Disallow: /schema.sql
```

Figure 2-2: Contents of robots.txt

Meta tags were then retrieved in the HEAD section of index.php by viewing the HTTP response with Burp Suite. This is shown below in figure 2-3.



```
<meta name="robots" content="all">
```

Figure 2-3: meta Tags in Response Head

This test found that robots.txt contained information pertaining to a file that may contain information which would help find a vulnerability.

2.3.3 Enumerate Applications on Web Server

This step's purpose was to discover any applications hosted on the web server. Applications and their respective vulnerabilities can be leveraged once the application is known. Therefore, the webserver was scanned for open ports and services using nmap with the command:

Nmap –Pn –sT –sV –p0-65535 192.168.1.10

To determine the version of any service (-sV) with a TCP scan (-sT) only (-Pn) on ports 0-65535 (-p). This gave the result shown below in table 3-1 (the nmap output can be found in Appendix A).

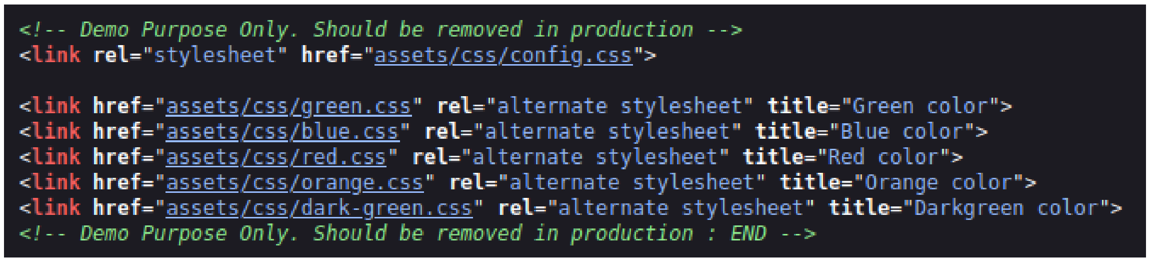
Port	Service	Version
21	ftp	ProFTPD 1.3.4a
80	http	Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
3306	mysql	MySQL

Table 3-1: Applications on Web Server

Applications found on the web server can have vulnerabilities, which can be exploited.

2.3.4 Review Webpage Content for Information Leakage

The web application may include comments and other information in the source code of the webserver files that allows for exploitation. The web applications source code was inspected page by page with the view-source function of the Firefox browser. Shown below is the only comments section that was found to be out with the normal function of the web application.



```
<!-- Demo Purpose Only. Should be removed in production -->
<link rel="stylesheet" href="assets/css/config.css">

<link href="assets/css/green.css" rel="alternate stylesheet" title="Green color">
<link href="assets/css/blue.css" rel="alternate stylesheet" title="Blue color">
<link href="assets/css/red.css" rel="alternate stylesheet" title="Red color">
<link href="assets/css/orange.css" rel="alternate stylesheet" title="Orange color">
<link href="assets/css/dark-green.css" rel="alternate stylesheet" title="Darkgreen color">
<!-- Demo Purpose Only. Should be removed in production : END -->
```

Figure 2-4: Webpage Comment

This comment block was found on all the .php web pages of the application, with the comment: “Demo Purposes Only. Should be Removed in production”.

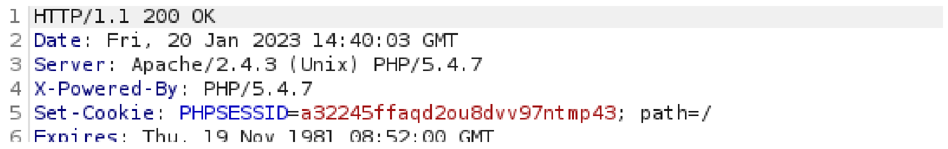
This is the only webpage content leakage.

2.3.5 Identify Application Entry Points

This test aimed to map the attack surface of the web application, which would help the investigator focus on probable weaknesses. This was done by noting parameters in POST requests and any other parameters of note, and queries submitted in the URL of web pages.

Documenting the responses was done with Burp Suite.

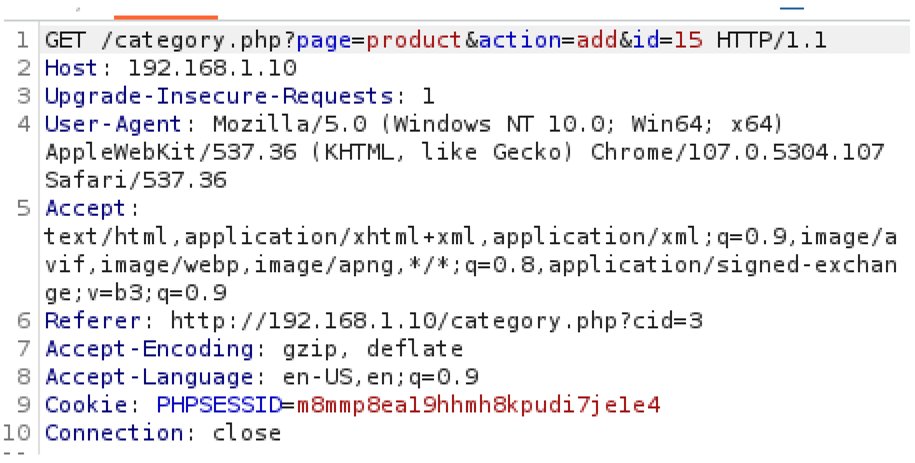
Browsing to any web page for the first time gave the response parameter shown below in figure 2-5: Set-Cookie: PHPSESSID=X, where X is an obfuscated string.



```
HTTP/1.1 200 OK
Date: Fri, 20 Jan 2023 14:40:03 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: PHPSESSID=a32245ffaqd2ou8dvv97ntmp43; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
```

Figure 2-5: PHPSESSID Parameter

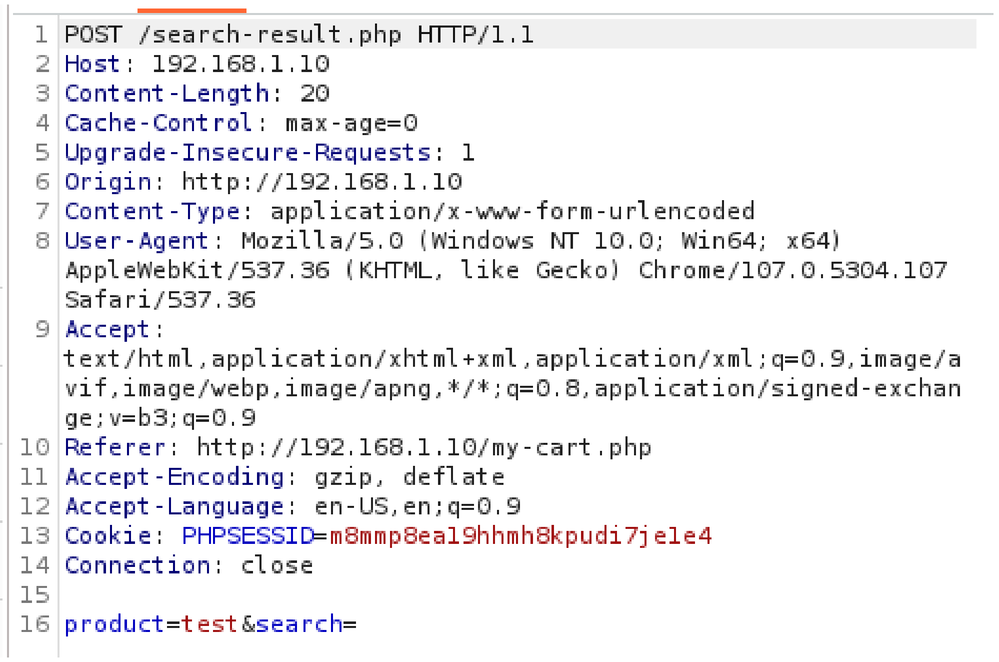
Adding an arbitrary product from the store to the cart passed some queries as part of a GET request, as shown below in figure 2-6.



```
GET /category.php?page=product&action=add&id=15 HTTP/1.1
Host: 192.168.1.10
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/a
vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchan
ge;v=b3;q=0.9
Referer: http://192.168.1.10/category.php?cid=3
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=m8mmp8eal9hhmh8kpudi7jele4
Connection: close
```

Figure 2-6: GET Request Query

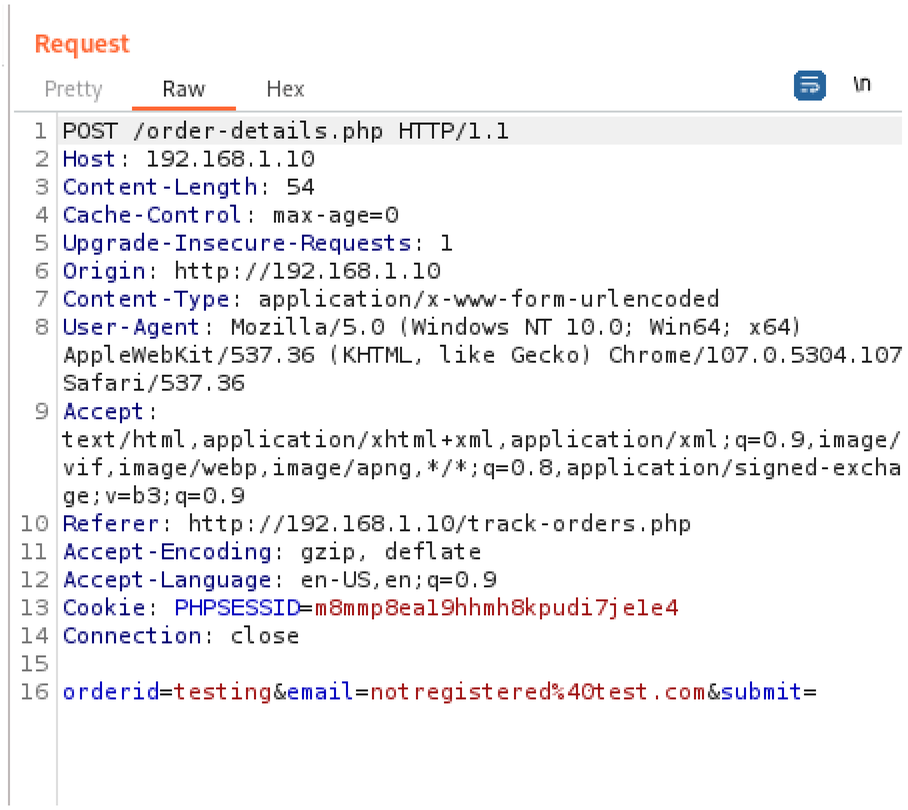
Searching for a product using the search bar of the website passed some parameters in a POST request as shown below in figure 2-7; the product=&search=.



```
POST /search-result.php HTTP/1.1
Host: 192.168.1.10
Content-Length: 20
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.10
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/a
vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchan
ge;v=b3;q=0.9
Referer: http://192.168.1.10/my-cart.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=m8mmp8ea19hhmh8kpudi7je1e4
Connection: close

product=test&search=
```

Figure 2-7: Search POST Parameters

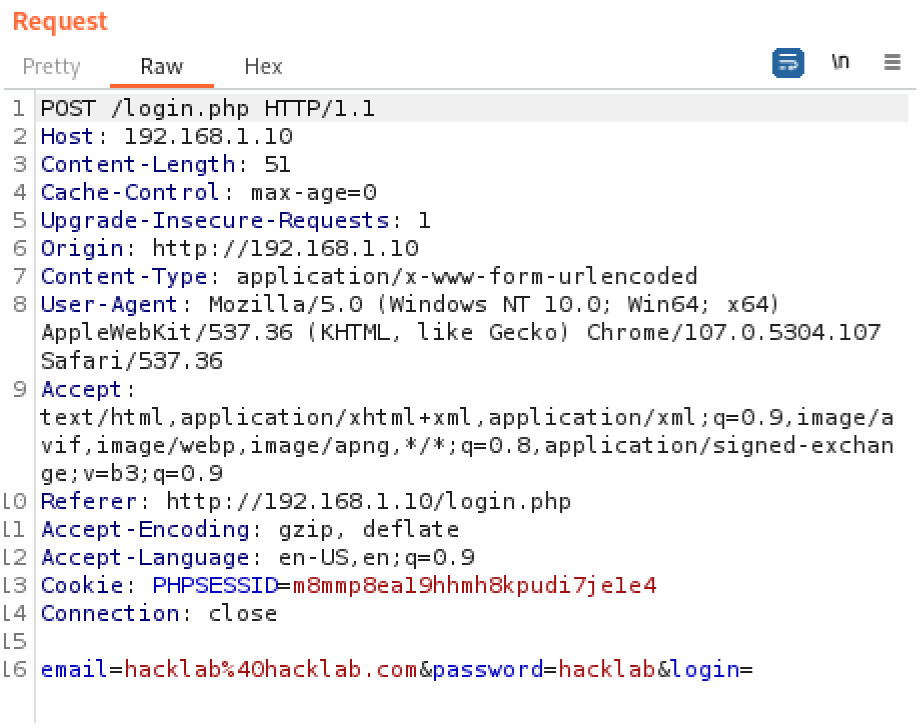
Attempting to track an order passed parameters in a POST request, as shown below in figure 2-8.



```
POST /order-details.php HTTP/1.1
Host: 192.168.1.10
Content-Length: 54
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.10
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/
vif,image/webp,image/apng,*/*;q=0.8,application/signed-excha
ge;v=b3;q=0.9
Referer: http://192.168.1.10/track-orders.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=m8mmp8ea19hhmh8kpudi7je1e4
Connection: close

orderid=testing&email=notregistered%40test.com&submit=
```

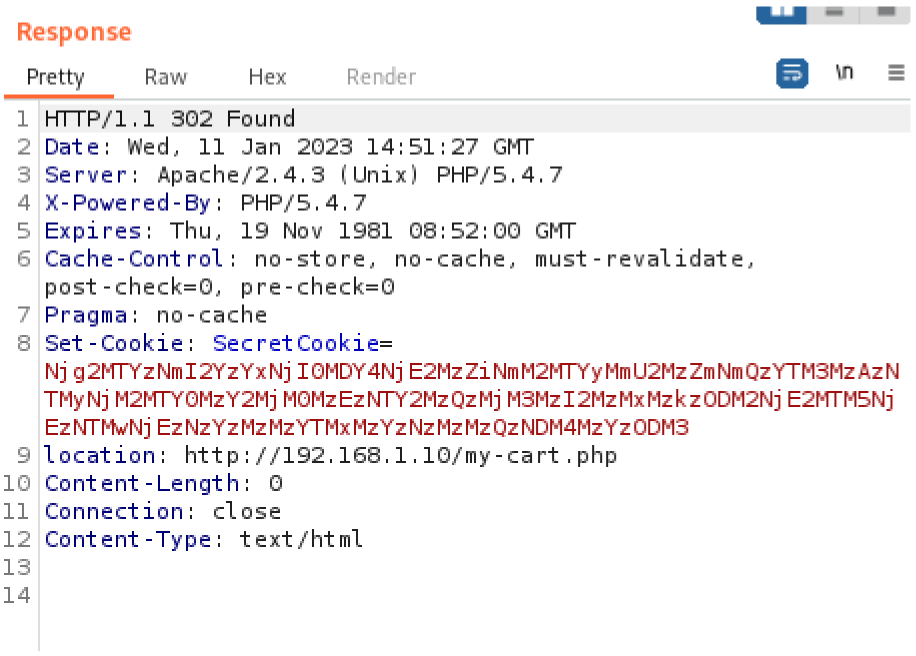
Figure 2-8: Tracking POST Parameters

Logging in as the supplied user passed the details as parameters in a POST request to the web server and received the Set-Cookie directive in response. The POST request is shown below in figure 2-9, with email=&password=&login=. The response is shown in figure 2-10 with the Set-Cookie: SecretCookie=.



```
POST /login.php HTTP/1.1
Host: 192.168.1.10
Content-Length: 51
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.10
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/a
vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchan
ge;v=b3;q=0.9
Referer: http://192.168.1.10/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=m8mmp8ea19hhmh8kpudi7jele4
Connection: close

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

Figure 2-9: Login POST Parameters



```
HTTP/1.1 302 Found
Date: Wed, 11 Jan 2023 14:51:27 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate,
post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: SecretCookie=
Njg2MTYzNmI2YzYxNjI0MDY4NjE2MzZiNmM2MTYyMmU2MzZmNmQzYTM3MzAzN
TMyNjM2MTY0MzY2MjM0MzEzNTY2MzQzMjM3MzI2MzMxMzkzODM2NjE2MTM5Nj
EzNTMwNjEzNzYzMzMzYTMxMzYzNzMzMzQzNDM4MzYzODM3
location: http://192.168.1.10/my-cart.php
Content-Length: 0
Connection: close
Content-Type: text/html


```

Figure 2-10: Login Response

Attempting to login to the admin panel found at 192.168.1.10/admin/index.php passed the login details as parameters in a POST request in the same fashion as a user login to the store.

Over the course of this step to identify application entry points, several potential points were located.

2.3.6 Map Execution Paths Through Application

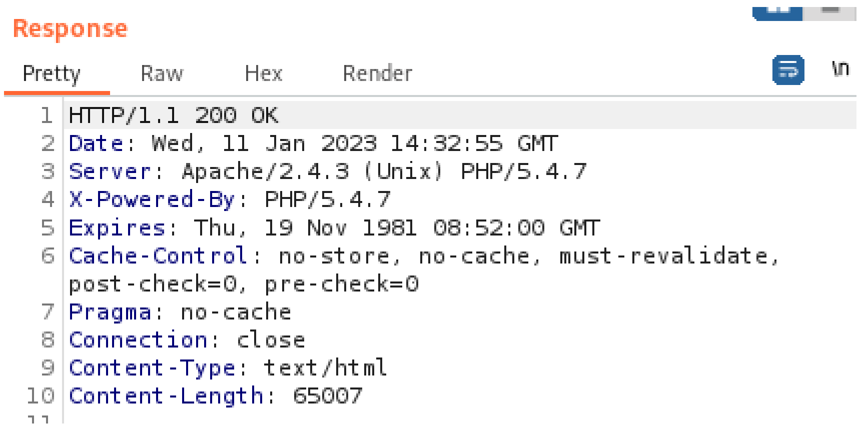
This part of the test aimed to determine the overall structure of the web application. The website was mapped using Burp Suite's Site Map feature, the results of which can be found in Appendix B.

An understanding of the structure of the web application can lead to vulnerability discovery.

2.3.7 Fingerprint Web Application Framework

This step's purpose was to determine the framework that the web server was dependent on. This was done by analysing response headers in Burp Suite and response attributes.

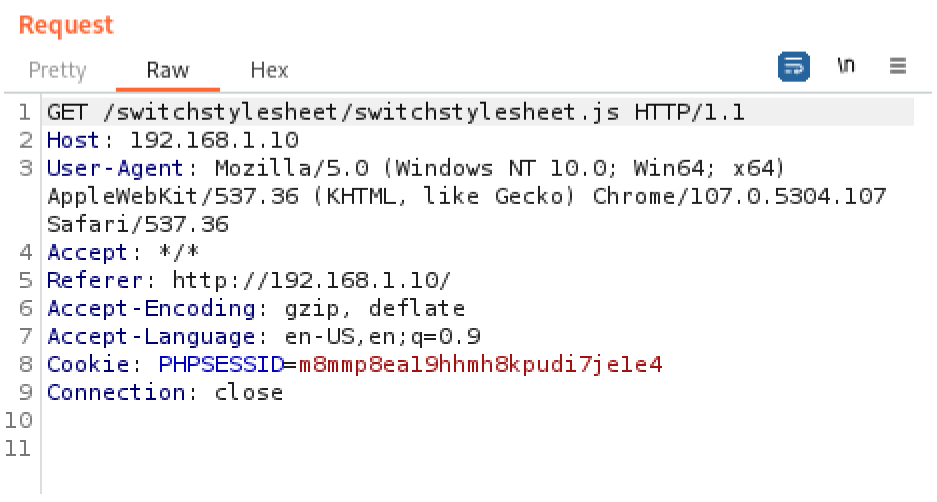
The web application was found to be running PHP/4.5.7 as found in the response headers from the website (one is shown below in figure 2-11).



```
HTTP/1.1 200 OK
Date: Wed, 11 Jan 2023 14:32:55 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate,
post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html
Content-Length: 65007
```

Figure 2-11: Response Header with PHP/5.4.7

Further evidence to show the web server was running a version of PHP. This was found by analysing response attributes, namely the Set-Cooke directive, and the resulting cookie which was named PHPSESSID=x. This was inferred to stand for PHP Session ID, indicating the web application was running a version of PHP. This is shown below in figure 2-12.



```
GET /switchstylesheet/switchstylesheet.js HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept: */*
Referer: http://192.168.1.10/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=m8mmp8ea19hhmh8kpudi7j e1e4
Connection: close


```

Figure 2-12: PHPSESSID Cookie

This test found the server to be running a version of PHP.

2.3.8 Map Application Architecture

The last step in mapping the web application structure was to determine the different components comprising the web application e.g. servers, proxies, and database servers.

From the nmap scan found in Appendix A and the site map in Appendix B, it was determined that the web application consisted of a single server hosting a web application with a back-end database.

2.4 CONFIGURATION AND DEPLOYMENT MANAGEMENT TESTING

2.4.1 Test Application Platform Configuration

Default files and applications present in a web application, if not removed once moving to production, can sometimes be the cause of vulnerabilities. This step of testing was to determine if any such files existed in the web application.

Performing a vulnerability scan of the web server would enumerate the existence of these files. Nikto was used to scan for known files and directories present in the web application.

Part of the scan is shown in Appendix A, the text of which is shown below for clarity:

+ 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.
+ 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.

Table 4-1: Nikto Known Directories Scan Result

These vulnerabilities found were already evaluated for severity by Nikto but presented a security risk.

2.4.2 Review Old Backup and Unreferenced Files for Sensitive Information

Web applications can have unreferenced or unused files that can be used to gain more information about the target. Automated backups renamed files or inclusion files are all part of this. These files can be used to better tailor an attack against a web application or contain functions that can be used to attack the web application.

The investigator was only able to find the unreferenced file schema.sql, included in the robots.txt file. It could be accessed by browsing to 192.168.1.10/schema.sql. This file describes the structure of the tables in the database for this web application, found in Appendix C.

This would present a vulnerability if SQL injection was possible.

2.4.3 Enumerate Infrastructure and Application Admin Interfaces

This part of the test was conducted to find any admin interfaces, and test for privilege escalation and bypassing of the interface. The admin interface was found at 192.168.1.10/admin/index.php by analysing results of the site map. The interface is shown below in figure 3-1.

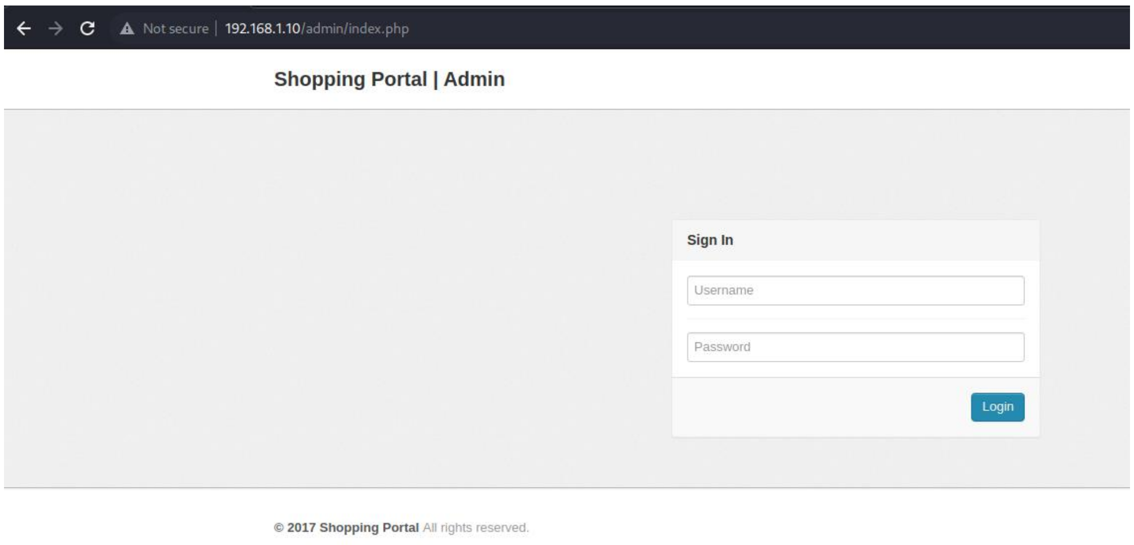

Not secure
192.168.1.10/admin/index.php
Shopping Portal | Admin
Sign In
Username
Password
Login
© 2017 Shopping Portal All rights reserved.

Figure 3-1: Admin Interface

No privilege escalation or capability to bypass this interface was found.

2.4.4 Test HTTP Methods

HTTP by default allows for several requests for interaction with a web application, some of which should be denied ensuring security of the web application. HTTP methods can also be used for XST (cross-site tracing) attacks, and arbitrary HTTP methods can also be a security vulnerability.

First, an nmap scan was used to determine which HTTP methods are allowed on the server. This was done with the command:

Nmap –p 80 –script http-methods –script-args http-methods.url-path='/index/php' 192.168.1.10

-p 80 specifies port 80 as that was the port of the http server, --script specifies a script to be used: in this case, the http-methods script, which needs the path to a webpage as an argument.

The results of this scan found that the supported methods were:

- GET
- HEAD
- POST
- OPTIONS

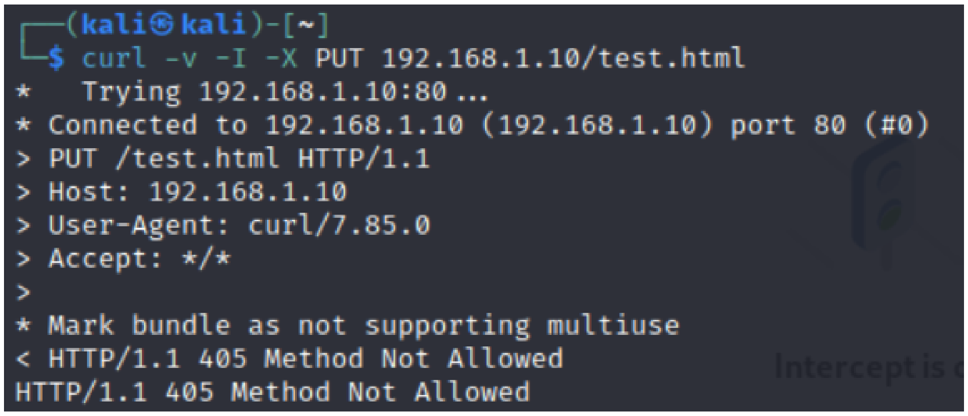
A screenshot of this full scan can be found in Appendix A.

To further corroborate results, cURL was used to request the non-existent /test.html with a PUT request (the PUT method can be used to place content on a web application remotely). The command used was:

```
curl -v -I -X PUT 192.168.1.10/test.html
```

Where –v specifies verbose, -I specifies only the header of the response be shown, and –X allows for HTTP methods other than GET (this includes arbitrary methods).

This is shown below in figure 3-2.

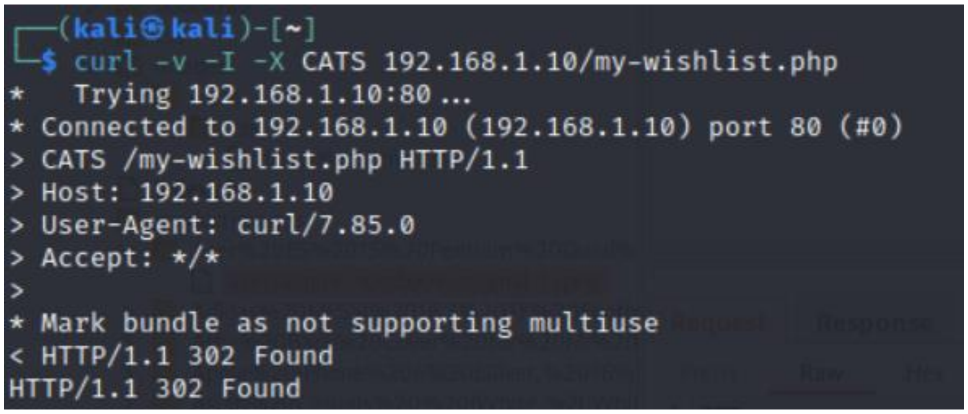


```
┌──(kali㉿kali)-[~]
└─$ curl -v -I -X PUT 192.168.1.10/test.html
*   Trying 192.168.1.10:80 ...
* Connected to 192.168.1.10 (192.168.1.10) port 80 (#0)
> PUT /test.html HTTP/1.1
> Host: 192.168.1.10
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 405 Method Not Allowed
HTTP/1.1 405 Method Not Allowed
```

Figure 3-2: cURL PUT Request

Also shown in figure 3-2 was the web applications response: 405, method not allowed.

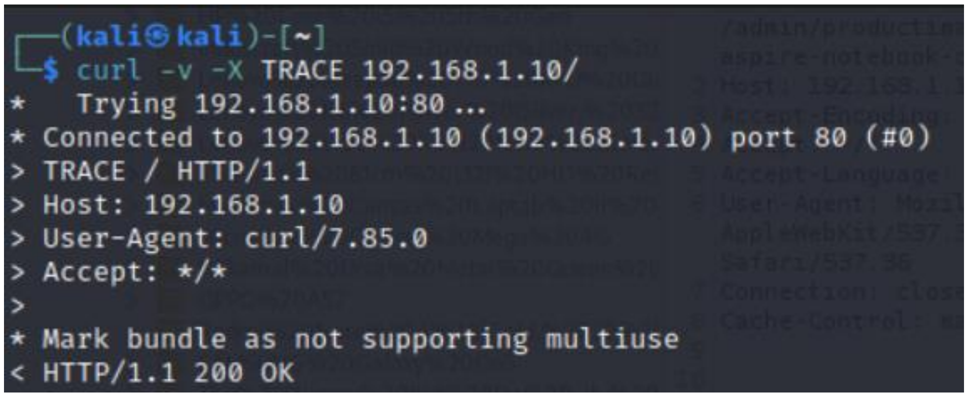
HTTP methods can also be used to bypass access control. To test this, a page was found that redirects unless a user is logged in. The page was /my-wishlist.php, which redirects to /index.php without a logged in user. cURL was used to request this page with an arbitrary HTTP method (CATS) to determine if the redirect could be bypassed. The web application correctly redirected to index.php. This is shown in figure 3-3 below.



```
┌──(kali㉿kali)-[~]
└─$ curl -v -I -X CATS 192.168.1.10/my-wishlist.php
*   Trying 192.168.1.10:80 ...
* Connected to 192.168.1.10 (192.168.1.10) port 80 (#0)
> CATS /my-wishlist.php HTTP/1.1
> Host: 192.168.1.10
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
HTTP/1.1 302 Found
```

Figure 3-3: 302 Redirect Arbitrary HTTP Method

The TRACE method needed to be tested, as if allowed this method can potentially reflect content to the client. cURL was used to test for this method, as shown below in figure 3-4.



```
┌──(kali㉿kali)-[~]
└─$ curl -v -X TRACE 192.168.1.10/
*   Trying 192.168.1.10:80 ...
* Connected to 192.168.1.10 (192.168.1.10) port 80 (#0)
> TRACE / HTTP/1.1
> Host: 192.168.1.10
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
```

Figure 3-4: 200 OK TRACE HTTP Request

As shown in the figure above, the TRACE method was accepted by the web application.

This test found that the PUT method was denied, as well as using arbitrary HTTP methods to bypass authorization. However, TRACE was found to be an accepted method.

2.5 IDENTITY MANAGEMENT TESTING

2.5.1 Test User Registration Process

This step aims to test the security of the user registration process. This was done by interacting with the website on a browser, making multiple new accounts satisfy the test conditions of the OWASP WTSG. The following information was gained:

- Anyone can register for an account
- Registration is automatic, and a valid email is required I.e. the @ symbol is required in the string
- The same identity can register multiple times
- A user cannot choose their role or permission level upon provision of the account
- A user needs an email address and contact number
- The email address and contact number is not verified

Validation during the account creation process is also non- existent; the information provided to the web application can be false and not relate to a real-life identity.

This part of testing found that user account registration could be used to create false identities, which could be a security vulnerability.

2.5.2 Testing for Account enumeration and Guessable User Account

It is sometimes possible to generate a list of valid user accounts by analysing responses from a web application. From this, each account could be brute forced for a valid password. This step aimed to test if generating a valid user list would be possible by comparing results from different user logins.

Burp Suite was used to analyse the HTTP traffic from the web application. Samples of valid, invalid, and mixed logins needed to be gathered so a comparison could be made.

First, a valid login was captured, with the results shown below in table 5-1 (full screenshots from this section, 2.5.2, can be found in Appendix A).

Address	Method	Status	Attributes
http://192.168.1.10/login.php	POST	302	SecretCookie
http://192.168.1.10/my-cart.php	GET	200	

Table 5-1: HTTP Enumeration Valid Test

A login with a non-existent user was then tested, with results shown below in table 5-2.

Address	Method	Status	Attributes
http://192.168.1.10/login.php	POST	200	
http://192.168.1.10/login.php	GET	200	

Table 5-2: HTTP Enumeration Invalid Test

Finally, a login with an existing email but an incorrect password was captured with the results shown below in table 5-3.

Address	Method	Status	Attributes
http://192.168.1.10/login.php	POST	302	SecretCookie
http://192.168.1.10/login.php	GET	200	

Table 5-3: HTTP Mixed Enumeration Test

This step found that brute forcing user account names was possible.

2.5.3 Testing for Weak or Unenforced Username Policy

This step aimed to find the possible structure of account names which would help determine if brute forcing active account names was possible.

This was done by creating or attempting to create new user accounts for Astley's store.

It was found that:

- Account names for the login page were structured x@y, where x and y were strings.
- In conjunction with results from section 2.5.2, it was found that if a user attempted to log in with a valid email, the web application would respond with a status 302; regardless of whether the user entered their password correctly. Invalid user accounts would prompt a 200 response from the web application.

This information shows that account information can be reused.

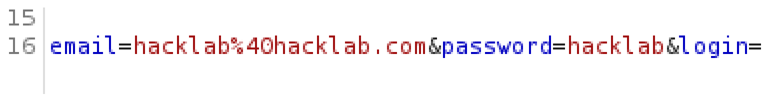
2.6 Authentication Testing

2.6.1 Testing for Credentials Transported over an Encrypted Channel

This step aimed to determine if the web application sent traffic to the client where any credentials for sensitive information were sent without some form of encryption.

Burp Suite was used to capture the exchange of information between the client and the web application, which the investigator used to analyse requests and responses for unencrypted credentials.

It was found that login information was sent to the web application in a POST request unencrypted. This is shown below in figure 4-1.



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

Figure 4-1: Unencrypted Login Credentials

As shown in figure 4-1, the login information for Mr. Steve Brown's account was sent unencrypted. Other test accounts were made and logged in to, and their credentials were sent the same way.

Further to this, the details for account creation were sent exposed in a POST request but included a full name and contact number as well.

Changing a user's password also sent the old and new password in a POST request unencrypted.

The web application was found to not use the HTTPS protocol or any HTTPS methods for the transport of sensitive information.

This test found that information exchanged with the web application was unencrypted.

2.6.2 Testing for Weak Lock Out Mechanism

Testing for any lockout mechanisms on user accounts was also done, mechanisms which would prevent the brute force guessing or cracking of user accounts.

This was tested in a web browser by attempting to incorrectly log in to a user account, then correctly logging in, with varying amounts invalid attempts. The following information was found:

- No account lockout after 3 incorrect attempts
- No account lockout after 4 incorrect attempts
- No account lockout after 5 incorrect attempts
- No CAPTCHA check to prevent brute force attacks

It was therefore found that user accounts had no lockout mechanism.

2.6.3 Testing For Bypassing Authentication Schema

Bypassing a login page removes the need for brute forcing or enumerating any user accounts for exploitation.

User logins were not found to be susceptible to any bypass, but the admin interface found at 192.168.1.10/admin/index.php was found to be vulnerable to SQL injection with SQLMap.

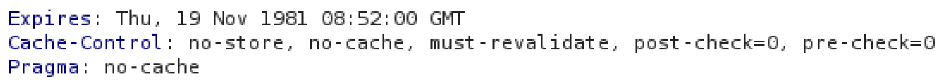
An SQL Injection is a major vulnerability.

2.6.4 Testing for Browser Cache Weakness

Web applications can store login or other sensitive information on client machines.

To test if Astley's Store did this, the browser history was used to attempt to navigate back to a user's account once the user had logged out. This was found to be not possible.

The browser cache was found to be secure as there were adequate cache control flags present in the HTTP headers, so no information was stored on the client side. This is shown below in figure 4-2.



```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
```

Figure 4-2: HTTP Header Cache Control Flags

This shows that security measures were in place to prevent the store if information on client machines.

2.6.5 Testing for Weak Password Policy

Further testing for how resilient user accounts were against brute force attacks was done by testing the password policy by creating accounts with various passwords using the web browser. The following information was found:

- All characters are permitted in a password.
- Users can change passwords as frequently as desired.
- There is no set time in place that a user must change their password after retaining that password for the length of time.
- Users can re-use old passwords
- New and old passwords can be identical
- Users can use any account details as passwords e.g. their contact number
- No minimum password length is in place.
- Common or known cracked passwords can be set.

This information shows that user passwords can be insecure.

2.6.6 Testing for Weak Security Question Answer

More testing for account resilience was done by determining any secret question systems. It was found no system was in place to check logins from a new client.

2.6.7 Testing for Weak Password Change or Reset Functionalities

The following information, using a web browser, was found for the rest password function of user accounts on the web application:

- A user must confirm an old password to set a new password.
- New passwords are sent to the web application in a POST request.
- New passwords are user-specified, and not random.
- No confirmation is asked of the user when resetting their password.

The gathered information found the password reset functionality to be insecure.

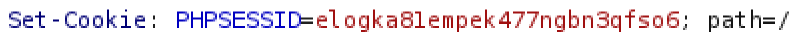
2.7 Session Management Testing

2.7.1 Testing for Session Management Schema

This step aimed to test whether session information was vulnerable, through cookies or other session variables.

Burp Suite was used to analyse cookie handling and the contents of the cookies and session variables set.

The web application was found to set a cookie, PHPSESSIONID, on first visit to the web application. This is shown below in figure 5-1.

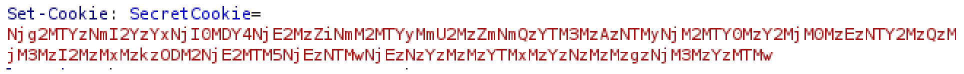


```
Set-Cookie: PHPSESSID=elogka8lempek477ngbn3qfso6; path=/
```

Figure 5-1: Example PHPSESSID Cookie

Set-Cookie is not tagged as Secure and is set over unencrypted transport.

Another cookie is set when logging in as a user: SecretCookie; shown below in figure 5-2.



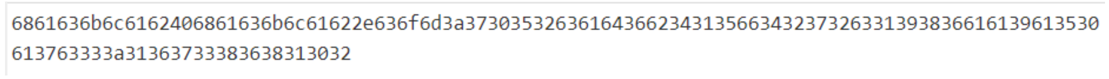
```
Set-Cookie: SecretCookie=
Njg2MTYzNmI2YzYxNjI0MDY4NjE2MzZiNmM2MTYyMmU2MzZmNmQzYTM3MzAzNTMyNjM2MTY0MzY2MjM0MzEzNTY2MzQzM
jM3MzI2MzMxMzkzODM2NjE2MTM5NjEzNTMwNjEzNzYzMzMzYTMxMzYzNzMzMzgzNjM3MzYzMTMw
```

Figure 5-2: Example SecretCookie Cookie

This SecretCookie was persistent, even when logging out of a user account. A new one was provisioned when a new user logged in.

Cyberchef was then used to test for possible decode options for the obfuscated string.

Converting the SecretCookie from Base64 gave the following result:



```
6861636b6c6162406861636b6c61622e636f6d3a37303532636164366234313566343237326331393836616139613530
613763333a31363733383638313032
```

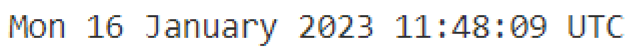
Figure 5-3: From Base64 Result

Converting this output from Hexadecimal gave the following result:

hacklab@hacklab.com:7052cad6b415f4272c1986aa9a50a7c3:1673868102

The SecretCookie was therefore predictable and not secure. The user's email is stored in this cookie.

After creating a new user account with a password and contact number consisting of 1 character each, it was found that the length if the middle and last segment of SecretCookie did not change. However, there was a small increment present in the last segment between cookie analysis. From this it was found the last segment decodes to a UNIX timestamp, as shown below in figure 5-4:



```
Mon 16 January 2023 11:48:09 UTC
```

Figure 5-4: UNIX Timestamp from Cookie

This step of testing found that session information is vulnerable through cookies.

2.7.2 Testing for Session Fixation

This step aimed to determine the authentication of users and if cookies could be forced.

Inspecting the cookie jar of the web browser in use (the cache of collected cookies) after login showed PHPSESSID and SecretCookie.

After logging in, SecretCookie was saved and then deleted. The investigator was still able to change account details associated with the user in the SecretCookie.

The investigator then logged in as a new user and changed the new SecretCookie to the previous saved SecretCookie. The investigator was not able to access account functions and was redirected to the /index.php page.

This test found that the web application was not vulnerable to session fixation.

2.7.3 Testing for Exposed Session Variables

Further testing of the session variables and cookies gave the following information:

- Session ID is transferred with a GET response, and SecretCookie with a POST response.
- Session IDs are never sent with encrypted transport.
- Cache control is applied appropriately.
- POST requests cannot be replaced with GET requests for the session ID’s.

2.7.4 Testing for Cross Site Request Forgery

After examining the web application and client response, it is likely the web application is vulnerable to CSRF. All actions on the client side are submitted to the web application in POST requests with appropriate authentication, so any action an authorized user could do could be done to a victim with the attacker using CSRF to exploit the victim's account.

2.7.5 Testing for Logout Functionality

Properly terminating a session on the web application helps to make a user's account more resilient to XSS and CSRF. To test if Astley’s Store had adequate logout functionality, a web browser was manually used to inspect the website.

The logout button was always present in an ideal location and clearly visible.

Session cookies were not deleted or modified on logout but using them to attempt to access a user account after logout was not possible.

2.7.6 Testing for Session Timeout

Having a session timeout on a web application prevents that session from being kept active to exploit a user account.

Astley’s Store did not have a session timeout as part of a security feature.

2.8 Input Validation Testing

2.8.1 Testing for Reflected XSS

This step of testing aimed to determine variables that could be reflected with XSS and evaluate the filtering of the XSS input.

This was done inputting strings manually using a web browser with payloads chosen from the XSS Filter Evasion sheet (OWASP, no date. B).

The search bar of the web application was found to be vulnerable to reflected XSS (see below, figure 6-1). The payloads used are described below.

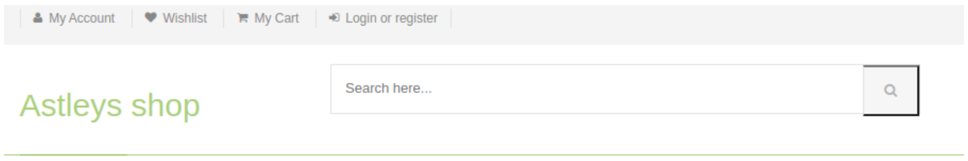
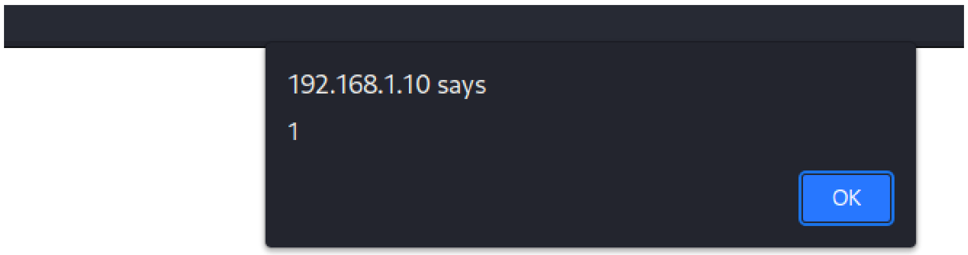


Figure 6-1: Web Application Search Bar

First, an XSS script to test JS was used:

<script>alert("1")</script>

This gave the result shown below in figure 6-2.


192.168.1.10 says

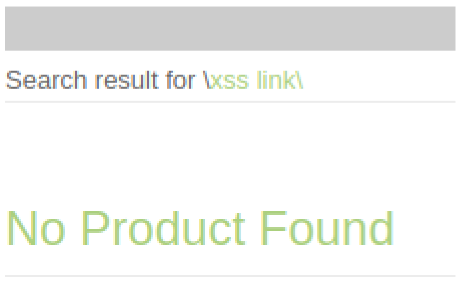
1

Figure 6-2: Alert("1") Script Result

An XSS script was then used to test the filtering of more characters:

\<a onmouseover="alert(document.cookie)"\>xss link\</a\>

This displayed text on the website, as shown in figure 6-3, that prompted an alert when the mouse was moved over the text.


Search result for \xss link\

No Product Found

Figure 6-3: XSS Link Green Text

Therefore, the web application was vulnerable to reflected XSS.

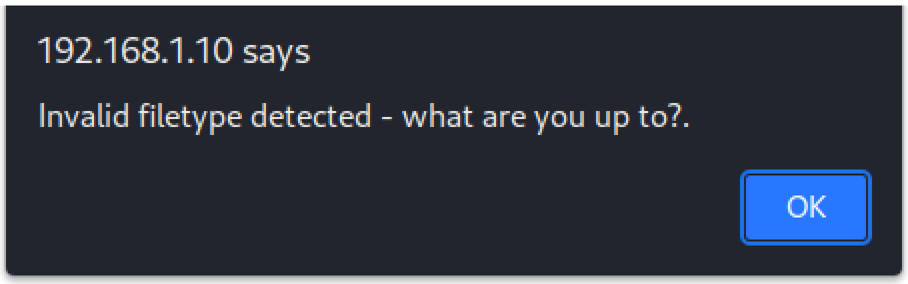
2.8.2 Testing for Stored XSS

Code can also be executed on a victim's machine using stored XSS, where the web application has been used to store malicious code that a victim will find and execute.

Burp Suite Intercept was used to modify requests to the web application for this part of the test.

The web application was not susceptible to stored XSS with user accounts.

Uploading HTML content was attempted, uploading a .jpg file to the user’s profile picture that contained HTML code. The web application detected the invalid file type successfully and displayed the error shown below in figure 6-4.


192.168.1.10 says

Invalid filetype detected - what are you up to?.

Figure 6-4: Stored XSS Test

The web application was not vulnerable to stored XSS.

2.8.3 Testing for HTTP Parameter Pollution

HTTP pollution can cause unexpected behavior in web applications. The system described by the OWASP WSTG was used to test if HTTP pollution was possible.

Burp Suite Intercept was used to capture and modify HTTP requests for this test, using the search bar as an input vector.

The first step was to send an unmodified HTTP request using the search bar. The search query of 'cat' returned search results for 'cat' in the response.

The second step was to modify the request from 'cat' to 'dog'. The web application returned search results for 'dog'.

The third step was to modify the initial request of 'cat' and concatenate the search query with the second request of 'dog'. This made the resulting request of:

product=cat&search=&product=dog&search=

The web application responded with search results for 'dog'.

This meant that the web application was not vulnerable to HTTP pollution, as the web application only responded to the last query of the response: 'dog'.

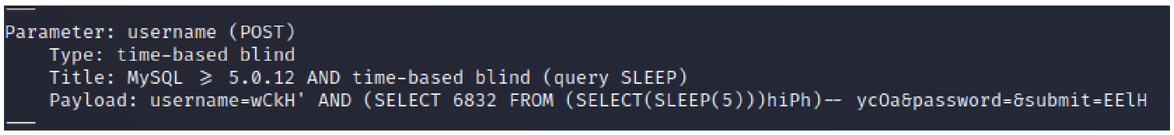
Client-side HTTP pollution was also tested for by issuing the query %26HPP_TEST to the search bar. The web application returned "search results for %26HPP_TEST", having not decoded %26 to the & symbol. This indicated the web application was not vulnerable client side to HTTP pollution.

2.8.4 Testing for SQL Injection

This test aimed to determine if it was possible to exploit a vulnerability to execute SQL queries on the attacker's behalf in the database of the web application. From previous testing in this report, it was known that the web application had a back- end MySQL database.

SQLMap was used to test input vectors; the form in /admin/index.php was found to be vulnerable, as well as the user login form in /login.php.

The vulnerability found in both is shown below in figure 6-5: a time-based blind attack (note the payload shown is for /admin/index.php).



```
---
Parameter: username (POST)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: username=wCkH' AND (SELECT 6832 FROM (SELECT(SLEEP(5)))hiPh)-- ycOa&password=&submit=EElH
---
```

Figure 6-5: MySQL Database Vulnerability

Despite this known vulnerability, only some information from the database was found due to a connection error with the application on the web server. The information found pointed to the names of the tables present in the 'shopping' database, but table content could not be enumerated due to the connection error.

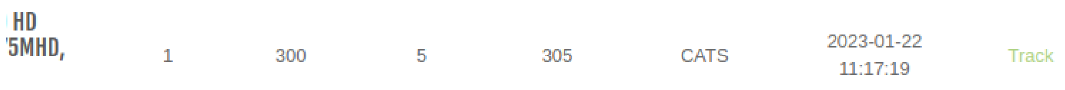
2.9 BUSINESS LOGIC TESTING

2.9.1 Test for Business Logic Data Validation

This step of testing aimed to determine if business transactions can be vulnerable to modification or injection.

Burp Suite Intercept was used to determine this.

A transaction was worked through on the web application of a user purchasing an arbitrary product. The transaction details were then tested to see if they could be disrupted.



HD 5MHD,	1	300	5	305	CATS	2023-01-22 11:17:19	Track

Figure 7-1: Business Data Validation Fail

As shown above in figure 7-1, the transaction was not validated so the payment method had been set to "CATS" by editing the POST request in Burp Suite. This invalid payment method was not identified by the web application.

2.9.2 Test Defenses Against Application Misuse

Information about the application layer defenses of the web application could be gained by evaluating results of the testing so far.

The applications defenses exhibited resilience to changing HTTP requests and control of which HTTP methods were allowed.

The application did not show any indication of defenses that would log a user out if detected attempting to find or leverage vulnerabilities.

2.10 Client-side Testing

2.10.1 Testing for Clickjacking

This test aimed to determine if the web application could be used to disguise a more malicious website's UI.

The test was conducted by creating a web page in mousepad with the code found in appendix D.

As shown below in figure 8-1, the UI of the web application could be retrieved by the test web page.

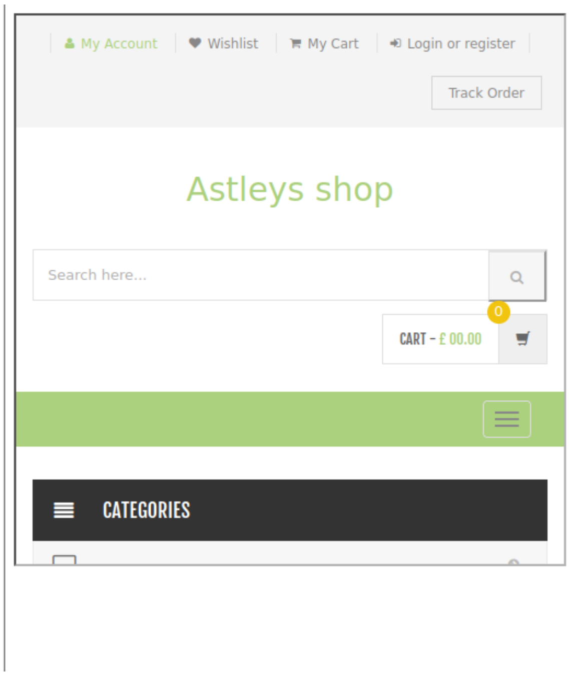


Figure 8-1: Clickjacking Test Result

This result indicated that the web application was vulnerable to be used in clickjacking attacks.

3 Discussion

3.1 Overall Discussion

The goal of this test was to evaluate the security and vulnerabilities present in the client's web application, Astley's Store, by following an established web application testing framework: the OWASP WSTG.
The web application was found to be seriously vulnerable, and would be an easy target for exploitation. Many vulnerabilities of varying severity were found, with little to no security measures beyond the apparent attempt to configure the web application correctly. This could affect the privacy of users of the web application or disrupt the business of the web application itself. The vulnerabilities found, and some security measures, are discussed below.

3.1.1 Information Gathering

Information about the web server and application was gathered, which an attacker would also be able to gather to begin searching for vulnerabilities. This means that the amount of relevant information able to be gathered can itself be a vulnerability. Testing for what information could be gathered was done over section 2.3, using a variety of tools to capture responses from the web application, to scan the web server ports and interact with the web application using HTTP methods. Specifically, the header of the HTTP responses from the web application contained what type of server was running the web application, and what version of the server.
A dump of the back-end database schema was found for later examination by a reference in the robots.txt file.
A scan of the server revealed several default scripts and files that could be used to gain more information about the system.
Cookies from the web application also revealed what framework the web server was using to host the web application: PHP.
Having this quantity of information available is a vulnerability, as this can be used by an attacker to leverage exploits more readily as known exploits can be used (Beagle Security, 2022).

3.1.2 Configuration and Deployment Management Testing

Testing ensued to determine how the web application was configured, and to test for any vulnerabilities in this configuration while the web application operated normally. Section 2.4 covered this testing with more scanning tools and analysis of HTTP responses.

Ultimately it was found that the web application was configured correctly to prevent vulnerabilities, but a critical vulnerability was found in that the web application only used HTTP and not HTTPS to communicate with clients and the TRACE method was allowed.

Using HTTP to communicate with clients allows all web traffic to be easily stolen or can be exploited to impersonate users (Cloudflare, no date). The TRACE method can be used for Cross-Site Tracing or to access headers or protected cookies (Shema, S. 2010).

3.1.3 Identity Management Testing

Testing in section 2.5 encompassed how secure the provisioning and security of user accounts was. No security features were found, pointing to several vulnerabilities. Testing was done by manually creating user accounts with a web browser and using Burp Suite to capture responses from the web application.

User accounts were found to be insecure and guessable by measuring the response from the web application, and no security features for account lockout were found. These vulnerabilities would allow the brute forcing of user accounts (OWASP, no date. C) with repetitive POST requests to the login form. With enough time and resources, all user accounts on Astley's Store could be compromised if the accounts email had been guessed or intercepted, and the account had been set with a weak or known password.

3.1.4 Authentication Testing

The tests covered in section 2.6 further analysed the security of users, but with a focus on sensitive information (e.g. passwords) to attempt to detect any vulnerabilities. This was completed with Burp Suite to capture HTTP requests and responses, a web browser to test user accounts, and SQLMap to detect vulnerabilities.

It was highlighted again that the web application makes use of the more insecure HTTP method, but also found that there was no account lockout function and passwords could be weak. An SQL injection attack was also a vulnerability that could be used to gain access to the back- end database of the web application. These factors further proved that user accounts on Astley's Store were vulnerable to attack.

3.1.5 Session Management Testing

Section 2.7 contained tests to detect any vulnerabilities with how the web application handled sessions. Burp Suite was used to capture communication between the client and the web application, and a web browser was used to test security functions of the user account.

Testing found that the session cookies were sent using unencrypted methods and that the sensitive data within the cookies could be exposed. This vulnerability would allow an attacker to gain user credentials.

There was no session timeout found for a user account, with the session expiring on closing the web browser.

User accounts could also be vulnerable to Cross-Site Request Forgery, as there was no token used to identify which session was using account functions- changing credentials or buying products. If admin accounts are vulnerable to the same attack, the whole web application could be exploited with remote execution (OWASP, no date. D).

3.1.6 Input Validation Testing

Step 2.8 of the testing attempted to find any vulnerabilities present where the user can supply input to the web application. Testing was done by manually inputting payloads into user input forms with a web browser. Burp Suite was also used to modify requests sent to the web application. SQLMap was used to assess input fields.

The web application was found to be vulnerable to reflected XSS, and no characters were found to be filtered- this vulnerability can be leveraged to perform actions on a user's behalf if the user is exposed to a malicious link (Portswigger, no date. B) provided by social engineering. In the context of this web application, these actions could include the changing of account information to gain control of a user account. The investigator did not find any fields vulnerable to stored XSS, and there was adequate security to stop malicious file uploads.

The web application responded securely and correctly to HTTP parameter pollution, in line with expected results for an Apache server using PHP: only the last occurrence of the parameter was parsed (OWASP, no date. E).

The web application was found to be vulnerable to SQL injection, with access granted to the back-end database of the application. This could be used to enumerate every user and admin account. If an admin account could be exploited, the whole web application could become compromised.

3.1.7 Business Logic Testing

Step 2.9 of testing used tests to determine if the logical flow of operation of the web application was vulnerable to disruption or exploitation. Testing was done using Burp Suite Intercept to modify HTTP requests. It was found that invalid payment methods could be submitted to the web application, a vulnerability which could be exploited to insert a victims payment details to be processed by the web server.

The application-level security measures were also evaluated, which found resistance to invalid HTTP methods. No other security measures were found.

3.1.8 Client- Side Testing

Step 2.10 focused on testing vulnerabilities in the client's browser before any information is sent to the server. A text editor was used within these tests. The web application was found to be vulnerable to clickjacking, where a user may believe they are using a web application when in fact they are interacting with a malicious, disguised web page (Kaspersky, no date.). This exploits HTML frames (iframes) which display another web page within another.

A simple web page was created with the text editor and was successful in retrieving Astley's Store within a HTML frame. This vulnerability could be used to deceive a victim that they are interacting with the web application and submit sensitive information to the malicious web application exploiting Astley's Store- details such as account logins or payment information.

3.2 COUNTERMEASURES

Possible countermeasures to some of the vulnerabilities found throughout this penetration test are highlighted below.

3.2.1 Banner Grabbing and Cookie Identifier

Information of the web server type and version was gathered in step 2.3.1 of this test, where the HTTP response headers were analysed from the web application. This information can be used to find known vulnerabilities to exploit the web application. The information shown in the response headers can be altered or removed entirely with the Apache module mod_headers (Apache, no date) in the case of Apache servers, which this test found the target server to be.

The cookies provided by the web application, as tested in step 2.3.7, also give away information to the framework the web server is using. The cookie named 'PHPSESSID' indicated that PHP was in use, and as before the response headers indicated what version. The cookie should be given a more generic name to stop easy identification of the web application framework.

3.2.2 Default Files and Robots.txt Schema Dump

The file found in section 2.3.2 and analysed later was schema.sql, a dump of the scheme for the database 'shopping' that the web application used as its back-end database. The information within this file enumerated several tables that were of interest- 'admin' and 'users' which can be used in SQL vulnerabilities.

Default files that would give away more information about the web application were found from a Nikto scan in section 2.4.1.

If these files are not necessary to the function of the web application, they should be removed to prevent information leakage.

While disallowing search engines to access schema.sql with robots.txt may seem a benefit, files and directories included in robots.txt with the disallow directive are not protected from an attacker.

3.2.3 TRACE HTTP Method

As found in step 2.4.4 of testing, the TRACE HTTP method was allowed by the server. This is a vulnerability as the method can be used in XST attacks. This method can be disabled on this Apache server by editing the relevant .htaccess and directory configuration files by adding the TRACE method to the list of disallowed methods in the <Limit> directive (Apache, no date. B, which will already include the PUT method (information found in the same section, 2.4.4, of this report).

3.2.4 Guessable Weak User Accounts

Over the course of sections 2.5-2.7, user accounts were found to be vulnerable to guessing and brute force attacks.

User accounts could be guessed by analysing the responses for login attempts to the web application. The application would return a status 200: OK on invalid usernames, but a status 302: Redirect on valid usernames and assign a cookie- regardless of if these usernames were paired with the correct password. This means active user accounts could be guessed by brute force. A solution to this would be to standardise login attempts to use status 200 or 302 regardless of a correct login, and only assign the SecretCookie cookie on successful login.

User accounts were also found to have close to no restrictions on the information that could be used to create them- false names, emails and contact numbers were not verified. Accounts could also be made with single letter passwords, and no policy on how many times a password could be reset or if the password could be set to a previous password. This further weakens the user accounts and the potential to brute force insecure account passwords increases.

These vulnerabilities can be patched by implementing features to directly counteract the vulnerability- a minimum password length and complexity where there was none, a link sent to an email to verify a person's identity, and restrictions on the frequency and choice of new password. These measures may require more infrastructure to be added to the web application, such as the capability to send emails, which could be costly.

3.2.5 Insecure Cookie

In step 2.7.1 of testing, the cookie assigned by the server SecretCookie was quickly decoded to be encoded with base64 and hexadecimal encoding. The information in the cookie after decoding was found to include the time of session login and the email address of the logged in user. If an attacker captured the cookie response, they may be able to begin a session as the user and have access to functions authorized users would.

This vulnerability could be mitigated by sending the client an encrypted SecretCookie if the web application needs the session to persist.

3.2.6 Reflected Cross Site Scripting

Step 2.8.1 found that the search bar of the web application was vulnerable to reflected XSS. The testing revealed no character filtering was taking place, allowing for user's sessions to be hijacked if provided with the correct XSS payload.

The solution to this vulnerability would be to implement character filtering on this input, and check such filtering exists on all other potential input vectors to prevent reflected XSS.

3.2.7 SQL Injection

In step 2.8.4 of testing the admin interface form fields and user account login fields were both found to be vulnerable to SQL injection. The enumeration of databases on the web application could potentially compromise the entire web application if an admin account is exploited.

As the test was black box and the code for database handling could not be reached, it is recommended that this vulnerability be fixed by checking how the web application handles database queries and implementing best practice methods. These include the use of prepared statements and making sure all user input is filtered and escaped (OWASP, no date. F).

3.2.8 CSRF

Tests in step 2.7.4 it was found that the web application is likely vulnerable to CSRF attacks, taking control of a victim's session.

This vulnerability could be patched with the implementation of a CSRF token- a unique token assigned to the client that the server keeps for that session and can be used to check that the client making requests is the client that was originally handed the token (Dizdar, A. 2021).

3.2.9 Clickjacking

Test 2.10.1 found the web application to be vulnerable to clickjacking, potentially deceiving a user of Astley's Store to reveal their account credentials.

Options in the HTTP headers of the web application can disallow the rendering of any web pages within an <iframe> tag, which clickjacking uses to render the disguise for the malicious web page. The X-Frame-Options header can be configured as such to completely deny all rendering or to only render from specific URLs (Hacksplaining, no date.). The Content-Securty-Policy HTTP header has the same functionality and is part of the HTML5 standard and aims to replace the X-Frame-Options header.

3.2.10 Use of HTTP

Throughout the penetration test, Astley's Store was found to exclusively use HTTP rather than HTTPS. This vulnerability should be fixed to prevent the easy theft of any communications between client and server by reevaluating the web applications framework and making use of the HTTPS protocol across the web application.

3.3 Future Work

Further work is needed to fully understand the impact of the SQL injection vulnerability. While a vulnerability was found with the capability to enumerate information within the databases of the web application, connection issues prevented any results from being identified- particularly when the schema.sql file found (see Appendix C) indicated the presence of an 'admin' and 'users' table within the 'shopping' database. Knowledge that information could be extracted from these tables, and what potential exploits could be done with the information, would allow a better assessment of the web applications vulnerabilities.

More investigation into the encoded middle segment of the SecretCookie found in section 2.7.1 is recommended, as the lack of knowledge of tools to decode this segment prevented any information on what data was being carried in the segment. While attempting to analyse multiple SecretCookies, this middle segment was of fixed length and random value. Therefore, the information within could be hashed to that fixed length and encoded, or always have a value of a fixed length: like an IP address. Decoding this segment would allow for a better vulnerability analysis of the web application, to determine what information is available to an attacker.

The precise capabilities for manipulation of the web application by a compromised admin account remains to be clarified, as access to the admin interface was not gained by brute force attack or by information leveraged from the SQL vulnerability. The damage a compromised admin account could do to the business and access to user's personal information needs to be evaluated to be included in an assessment of the vulnerability of this web application.

REFERENCES

Apache, no date. A. *Apache Module mod_headers.* [online]. Available from: https://httpd.apache.org/docs/current/mod/mod_headers.html [Accessed 23 January 2023]

Apache, no date. B. *Apache Core Features.* [online]. Available from: https://httpd.apache.org/docs/current/mod/core.html [Accessed 23 January 2023]

Beagle Security, 2022. *Fingerprinting Web Server.* [online]. Available from: https://beaglesecurity.com/blog/vulnerability/fingerprinting-web-server.html [Accessed 21 January 2023]

Cisco, no date. *What is Penetration Testing?.* [online]. Available from: https://www.cisco.com/c/en/us/products/security/what-is-pen-testing.html [Accessed 18 January 2023]

Cloudflare, no date. *Why use HTTPS?.* [online]. Available from: https://www.cloudflare.com/learning/ssl/why-use-https/ [Accessed 21 January 2023]

Digital Defense, no date. *What is Web Application Penetration Testing?.* [online]. Available from: https://www.digitaldefense.com/blog/what-is-web-application-penetration-testing/ [Accessed 18 January 2023]

Dizdar, A. 2021. *CSRF Tokens: What is a CSRF token and how does it work?.* [online]. Available from: https://brightsec.com/blog/csrf-token/ [Accessed 23 January 2023]

Gov.uk, 2022. *Cyber Security Breaches Survey: 2022.* [online document]. Available from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1088534/Business_infographic_-_Cyber_Security_Breaches_Survey_2022.pdf [Accessed 18 January 2023]

Hacksplaining, no date. *Protecting your Users Against Clickjacking.* [online]. Available from: https://www.hacksplaining.com/prevention/click-jacking [Accessed: 23 January 2023]

ISACA, 2016. *Auditing Cybersecurity.* [online]. Available from: https://www.isaca.org/resources/isaca-journal/issues/2016/volume-1/auditing-cybersecurity [Accessed 18 January 2023]

KALI, 2022A. *What is Kali Linux?.* [online]. Available from: https://www.kali.org/docs/introduction/what-is-kali-linux/ [Accessed 19 January 2023]

KALI, 2022B. *Dirb.* [online]. Available from: https://www.kali.org/tools/dirb/ [Accessed 19 January 2023]

KALI, 2022C. *Nikto.* [online]. Available from: https://www.kali.org/tools/nikto/ [Accessed 19 January 2023]

KALI, 2022D. *Sqlmap.* [online]. Available from: https://www.kali.org/tools/sqlmap/ [Accessed 19 January 2023]

Kaspersky, no date. *What is clickjacking? How to prevent clickjacking attacks.* [online]. Available from: https://www.kaspersky.co.uk/resource-center/definitions/clickjacking [Accessed 23 January 2023]

NMAP, no date. *Chapter 15. Nmap Reference Guide.* [online]. Available from: https://nmap.org/book/man.html#man-description [Accessed 19 January 2023]

OWASP, no date. A. *WSTG – v4.2.* [online]. Available from: https://owasp.org/www-project-web-security-testing-guide/v42/ [Accessed 19 January 2023]

OWASP, no date. B. *XSS Filter Evasion Cheat Sheet.* [online]. Available from: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html [Accessed 21 January 2023]

OWASP, no date. C. *Brute Force Attack.* [online]. Available from: https://owasp.org/www-community/attacks/Brute_force_attack [Accessed January 23 2023]

OWASP, no date. D. *Cross Site Request Forgery (CSRF).* [online]. Available from: https://owasp.org/www-community/attacks/csrf [Accessed January 23 2023]

OWASP, no date. E. *Testing for HTTP Parameter Pollution.* [online]. Available from: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/04-Testing_for_HTTP_Parameter_Pollution [Accessed January 23 2023]

OWASP, no date. F. *SQL Injection Prevention Cheat Sheet.* [online]. Available from: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html [Accessed January 23 2023]

PacketLabs, 2021. *3 Reasons to Review the OWASP Web Security Testing Guide.* [online]. Available from: https://www.packetlabs.net/posts/owasp-web-security-testing-guide/ [Accessed 19 January 2023]

Pedamkar, P. no date. *Black Box Testing.* [blog]. Available from: https://www.educba.com/black-box-testing/ [Accessed 19 January 2023]

Perez, Y. no date. *Mousepad.* [online] Available from: https://www.commandlinux.com/man-page/man1/mousepad.1.html [Accessed 22 January 2023]

Portswigger, no date. A. *Burp Suite Community Edition.* [online]. Available from: https://portswigger.net/burp/communitydownload [Accessed 19 January 2023]

Portswigger, no date. B. *Reflected XSS.* [online]. Available from: https://portswigger.net/web-security/cross-site-scripting/reflected [Accessed 23 January 2023]

Shema, S. 2010. *Cross-Site Tracing (XST): The misunderstood vulnerability.* [online]. Available from: https://deadliestwebattacks.com/appsec/2010/05/18/cross-site-tracing-xst-the-misunderstood-vulnerability [Accessed 23 January 2023]

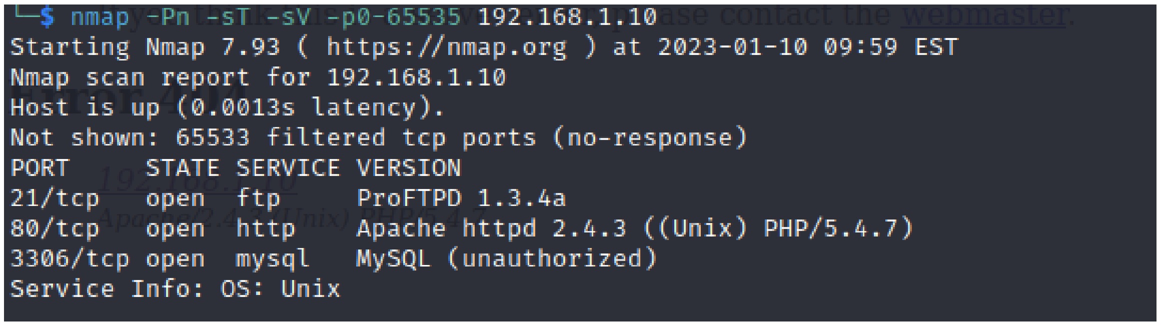
Statista, 2021. *Average cost of cyber incidents to organizations in the United Kingdom (UK) as of 2021, by industry.* [online]. Available from: https://www.statista.com/statistics/1245947/cost-of-cyber-attacks-by-industry-uk/ [Accessed 18 January 2023]

Sulcas, A. 2020. *What is cURL and What Does it Mean?.* [blog]. Available from: https://oxylabs.io/blog/what-is-curl [Accessed 19 January 2023]

Tenable, no date. *Nessus.* [online]. Available from: https://www.tenable.com/products/nessus [Accessed 18 January 2023]

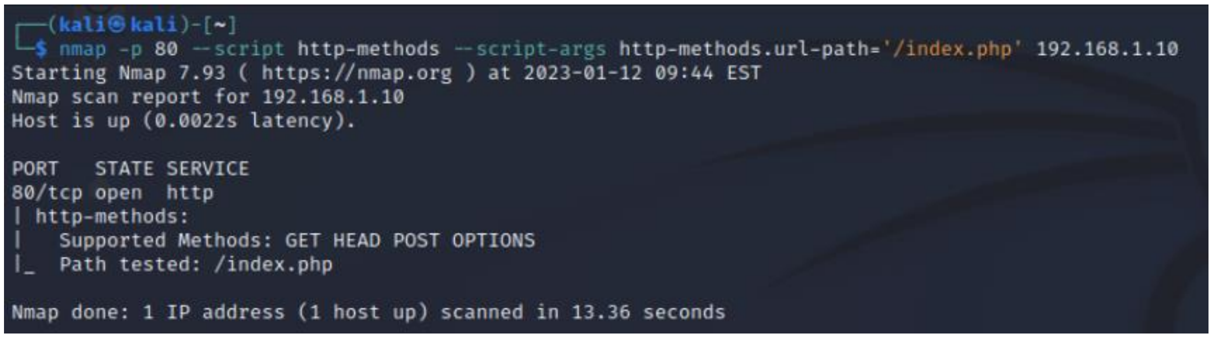
APPENDICES

APPENDIX A- BURP SUITE CAPTURES AND NMAP/NIKTO SCANS



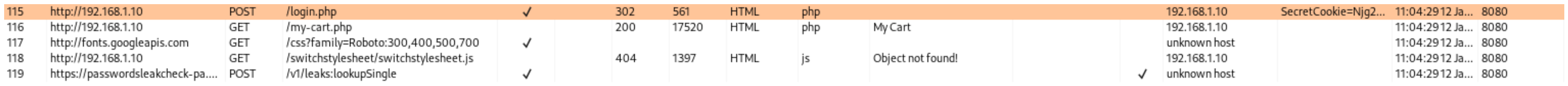
```
└─$ nmap -Pn -sT -sV -p0-65535 192.168.1.10
Starting Nmap 7.93 ( https://nmap.org ) at 2023-01-10 09:59 EST
Nmap scan report for 192.168.1.10
Host is up (0.0013s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     ProFTPD 1.3.4a
80/tcp   open  http    Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OS: Unix
```

Figure 9-1: Nmap Scan of Running Services



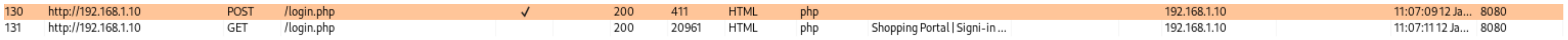
```
┌──(kali㉿kali)-[~]
└─$ nmap -p 80 --script http-methods --script-args http-methods.url-path='/index.php' 192.168.1.10
Starting Nmap 7.93 ( https://nmap.org ) at 2023-01-12 09:44 EST
Nmap scan report for 192.168.1.10
Host is up (0.0022s latency).

PORT   STATE SERVICE
80/tcp open  http
| http-methods:
|   Supported Methods: GET HEAD POST OPTIONS
|_  Path tested: /index.php

Nmap done: 1 IP address (1 host up) scanned in 13.36 seconds
```

Figure 9-2: Nmap Scan of HTTP Methods



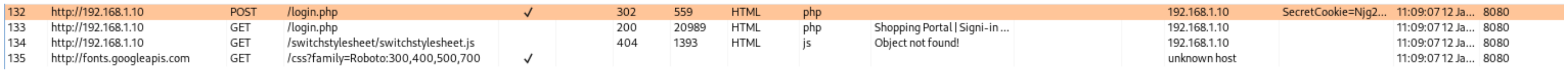
115	http://192.168.1.10	POST	/login.php	✓		302	561	HTML	php				192.168.1.10	SecretCookie=Njg2...	11:04:29 12 Ja...	8080
116	http://192.168.1.10	GET	/my-cart.php			200	17520	HTML	php	My Cart			192.168.1.10		11:04:29 12 Ja...	8080
117	http://fonts.googleapis.com	GET	/css?family=Roboto:300,400,500,700	✓									unknown host		11:04:29 12 Ja...	8080
118	http://192.168.1.10	GET	/switchstylesheet/switchstylesheet.js			404	1397	HTML	js	Object not found!			192.168.1.10		11:04:29 12 Ja...	8080
119	https://passwordsleakcheck-pa....	POST	/v1/leaks:lookupSingle	✓								✓	unknown host		11:04:29 12 Ja...	8080

Figure 9-3: Account Enumeration HTTP Method 1



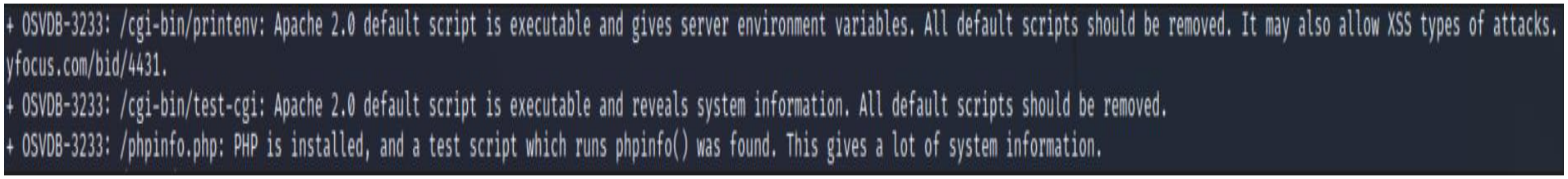
130	http://192.168.1.10	POST	/login.php	✓		200	411	HTML	php				192.168.1.10		11:07:09 12 Ja...	8080
131	http://192.168.1.10	GET	/login.php			200	20961	HTML	php	Shopping Portal | Signi-in ...			192.168.1.10		11:07:11 12 Ja...	8080

Figure 9-4:Account Enumeration HTTP Method 2



132	http://192.168.1.10	POST	/login.php	✓		302	559	HTML	php				192.168.1.10	SecretCookie=Njg2...	11:09:07 12 Ja...	8080
133	http://192.168.1.10	GET	/login.php			200	20989	HTML	php	Shopping Portal | Signi-in ...			192.168.1.10		11:09:07 12 Ja...	8080
134	http://192.168.1.10	GET	/switchstylesheet/switchstylesheet.js			404	1393	HTML	js	Object not found!			192.168.1.10		11:09:07 12 Ja...	8080
135	http://fonts.googleapis.com	GET	/css?family=Roboto:300,400,500,700	✓									unknown host		11:09:07 12 Ja...	8080

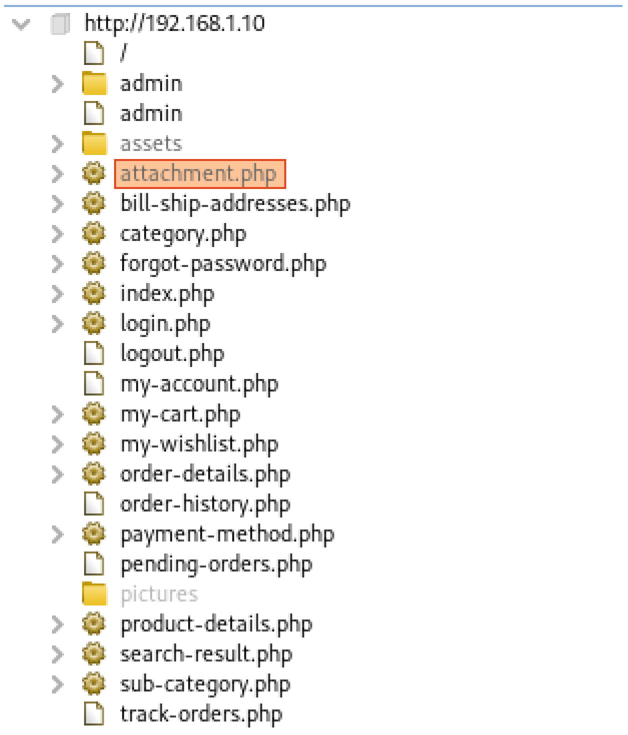
Figure 9-5: Account Enumeration HTTP Method 3



```
+ 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.
yfocus.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.
```

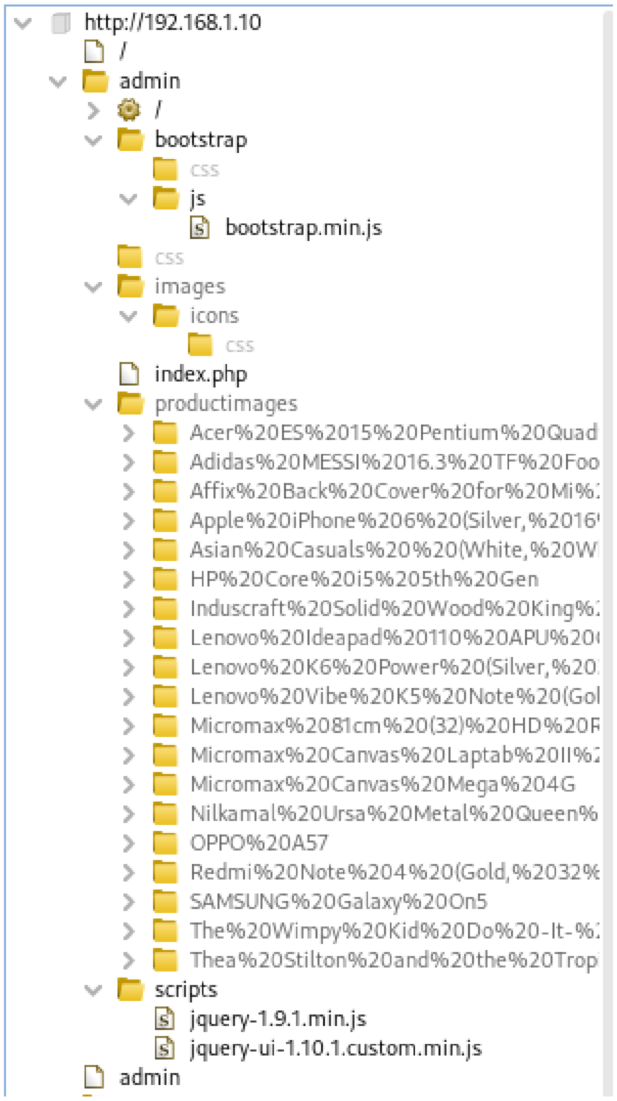
Figure 9-6: Nikto Scan of Known Directories

APPENDIX B- WEB APPLICATION STRUCTURE



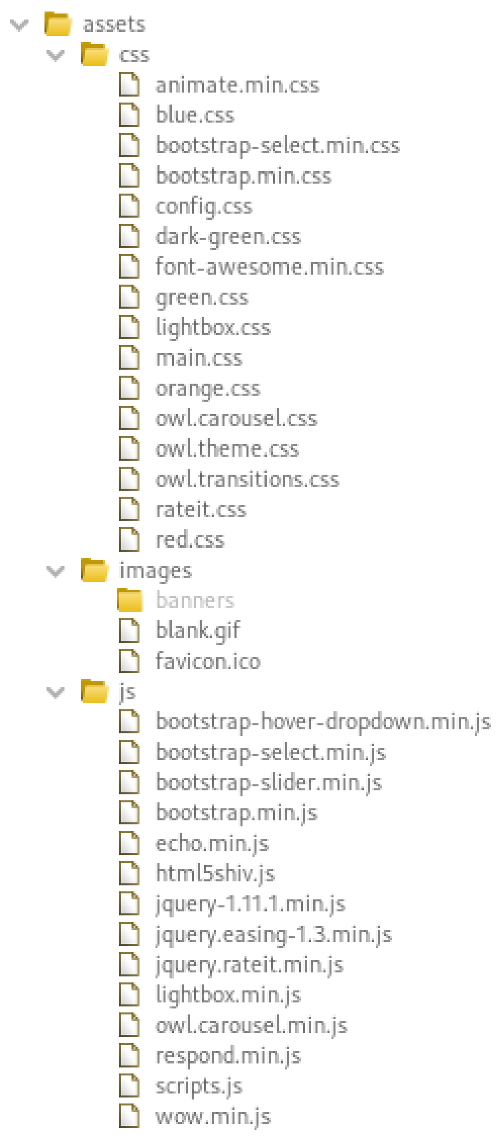
```
http://192.168.1.10
    /
    admin
    admin
    assets
    attachment.php
    bill-ship-addresses.php
    category.php
    forgot-password.php
    index.php
    login.php
    logout.php
    my-account.php
    my-cart.php
    my-wishlist.php
    order-details.php
    order-history.php
    payment-method.php
    pending-orders.php
    pictures
    product-details.php
    search-result.php
    sub-category.php
    track-orders.php
```

Figure 10-1: Web Application Structure 1



```
http://192.168.1.10
    /
    admin
        /
        bootstrap
            css
            js
                bootstrap.min.js
        css
        images
            icons
                css
        index.php
        productimages
            Acer%20ES%2015%20Pentium%20Quad
            Adidas%20MESSI%2016.3%20TF%20Foo
            Affix%20Back%20Cover%20for%20Mi%2
            Apple%20iPhone%206%20(Silver,%2016
            Asian%20Casuals%20%20(White,%20W
            HP%20Core%20i5%205th%20Gen
            Induscraft%20Solid%20Wood%20King%2
            Lenovo%20Ideapad%20110%20APU%20
            Lenovo%20K6%20Power%20(Silver,%20
            Lenovo%20Vibe%20K5%20Note%20(Gol
            Micromax%2081cm%20(32)%20HD%20R
            Micromax%20Canvas%20Laptab%20II%2
            Micromax%20Canvas%20Mega%204G
            Nilkamal%20Ursa%20Metal%20Queen%
            OPPO%20A57
            Redmi%20Note%204%20(Gold,%2032%
            SAMSUNG%20Galaxy%20On5
            The%20Wimpy%20Kid%20Do%20-It-%2
            Thea%20Stilton%20and%20the%20Trop
        scripts
            jquery-1.9.1.min.js
            jquery-ui-1.10.1.custom.min.js
    admin
```

Figure 10-2: Web Application Structure 2



```
assets
  css
    animate.min.css
    blue.css
    bootstrap-select.min.css
    bootstrap.min.css
    config.css
    dark-green.css
    font-awesome.min.css
    green.css
    lightbox.css
    main.css
    orange.css
    owl.carousel.css
    owl.theme.css
    owl.transitions.css
    rateit.css
    red.css
  images
    banners
    blank.gif
    favicon.ico
  js
    bootstrap-hover-dropdown.min.js
    bootstrap-select.min.js
    bootstrap-slider.min.js
    bootstrap.min.js
    echo.min.js
    html5shiv.js
    jquery-1.11.1.min.js
    jquery.easing-1.3.min.js
    jquery.rateit.min.js
    lightbox.min.js
    owl.carousel.min.js
    respond.min.js
    scripts.js
    wow.min.js
```

Figure 10-3: Web Application Structure 3

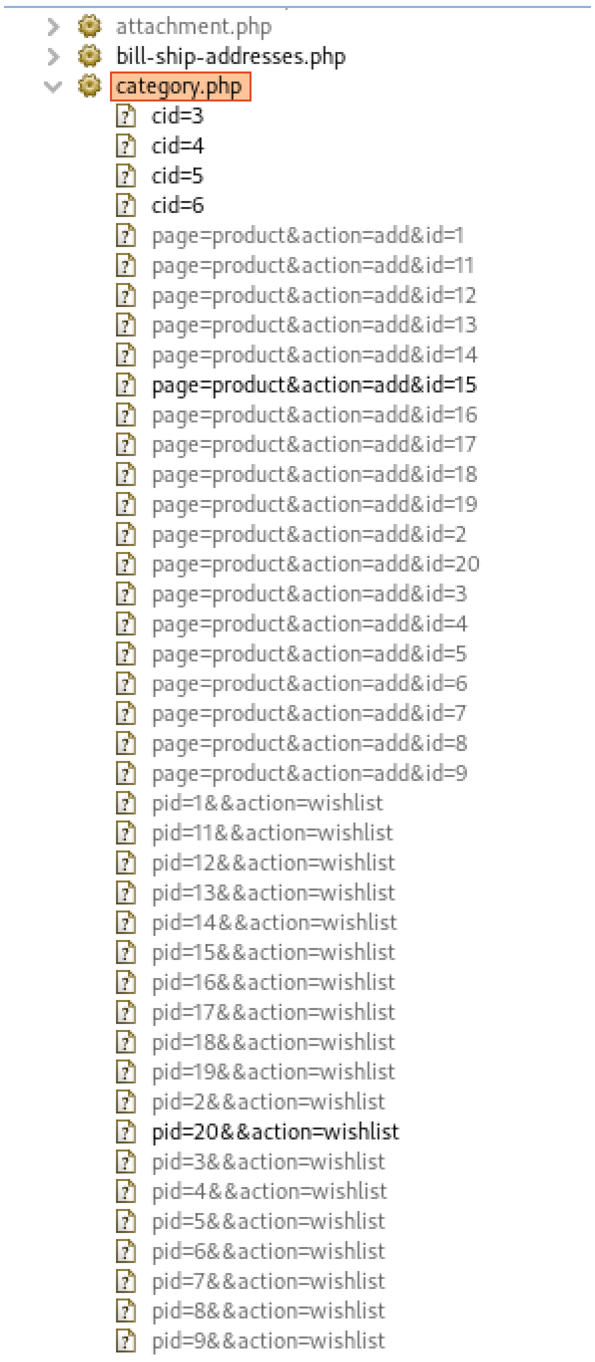
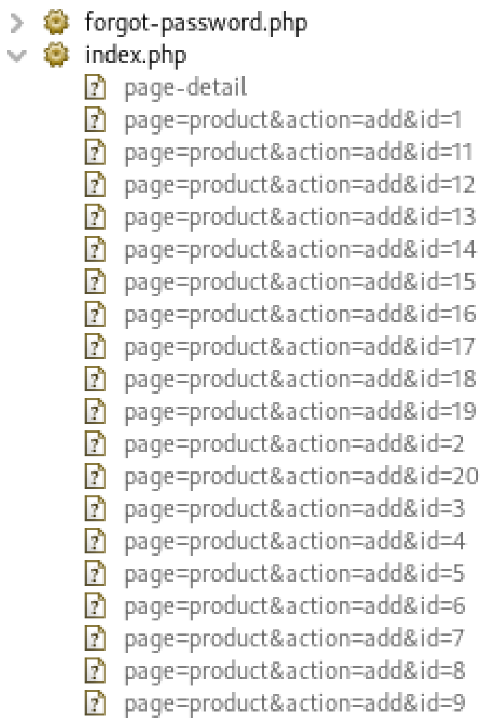
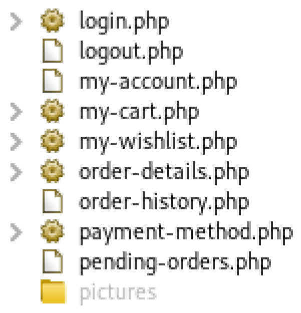

attachment.php
bill-ship-addresses.php
category.php
cid=3
cid=4
cid=5
cid=6
page=product&action=add&id=1
page=product&action=add&id=11
page=product&action=add&id=12
page=product&action=add&id=13
page=product&action=add&id=14
page=product&action=add&id=15
page=product&action=add&id=16
page=product&action=add&id=17
page=product&action=add&id=18
page=product&action=add&id=19
page=product&action=add&id=2
page=product&action=add&id=20
page=product&action=add&id=3
page=product&action=add&id=4
page=product&action=add&id=5
page=product&action=add&id=6
page=product&action=add&id=7
page=product&action=add&id=8
page=product&action=add&id=9
pid=1&&action=wishlist
pid=11&&action=wishlist
pid=12&&action=wishlist
pid=13&&action=wishlist
pid=14&&action=wishlist
pid=15&&action=wishlist
pid=16&&action=wishlist
pid=17&&action=wishlist
pid=18&&action=wishlist
pid=19&&action=wishlist
pid=2&&action=wishlist
pid=20&&action=wishlist
pid=3&&action=wishlist
pid=4&&action=wishlist
pid=5&&action=wishlist
pid=6&&action=wishlist
pid=7&&action=wishlist
pid=8&&action=wishlist
pid=9&&action=wishlist

Figure 10-4: Web Application Structure 4



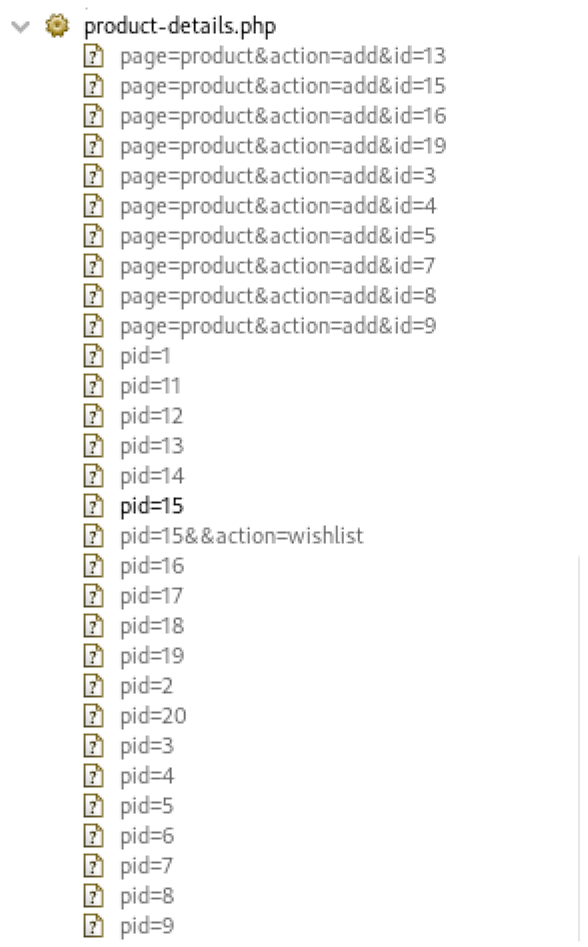
forgot-password.php
index.php
page-detail
page=product&action=add&id=1
page=product&action=add&id=11
page=product&action=add&id=12
page=product&action=add&id=13
page=product&action=add&id=14
page=product&action=add&id=15
page=product&action=add&id=16
page=product&action=add&id=17
page=product&action=add&id=18
page=product&action=add&id=19
page=product&action=add&id=2
page=product&action=add&id=20
page=product&action=add&id=3
page=product&action=add&id=4
page=product&action=add&id=5
page=product&action=add&id=6
page=product&action=add&id=7
page=product&action=add&id=8
page=product&action=add&id=9

Figure 10-5: Web Application Structure 5



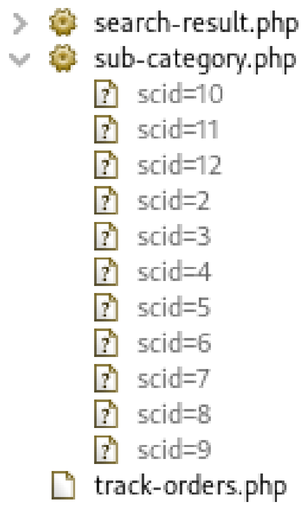
login.php
logout.php
my-account.php
my-cart.php
my-wishlist.php
order-details.php
order-history.php
payment-method.php
pending-orders.php
pictures

Figure 10-6: Web Application Structure 6



```
product-details.php
    page=product&action=add&id=13
    page=product&action=add&id=15
    page=product&action=add&id=16
    page=product&action=add&id=19
    page=product&action=add&id=3
    page=product&action=add&id=4
    page=product&action=add&id=5
    page=product&action=add&id=7
    page=product&action=add&id=8
    page=product&action=add&id=9
    pid=1
    pid=11
    pid=12
    pid=13
    pid=14
    pid=15
    pid=15&&action=wishlist
    pid=16
    pid=17
    pid=18
    pid=19
    pid=2
    pid=20
    pid=3
    pid=4
    pid=5
    pid=6
    pid=7
    pid=8
    pid=9
```

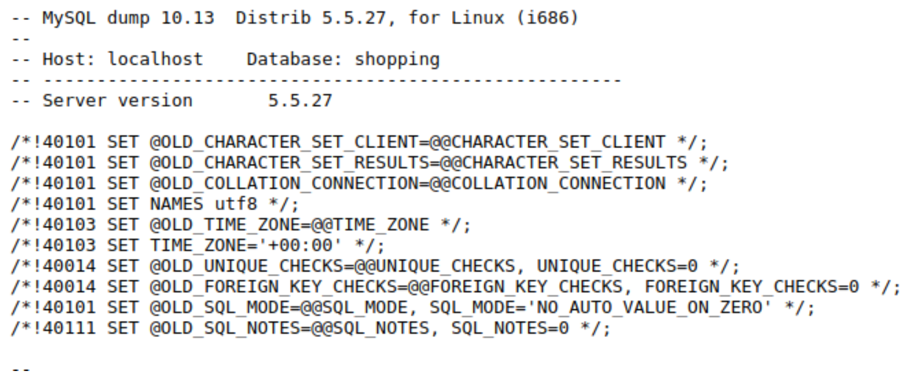
Figure 10-7: Web Application Structure 7



```
search-result.php
sub-category.php
    scid=10
    scid=11
    scid=12
    scid=2
    scid=3
    scid=4
    scid=5
    scid=6
    scid=7
    scid=8
    scid=9
track-orders.php
```

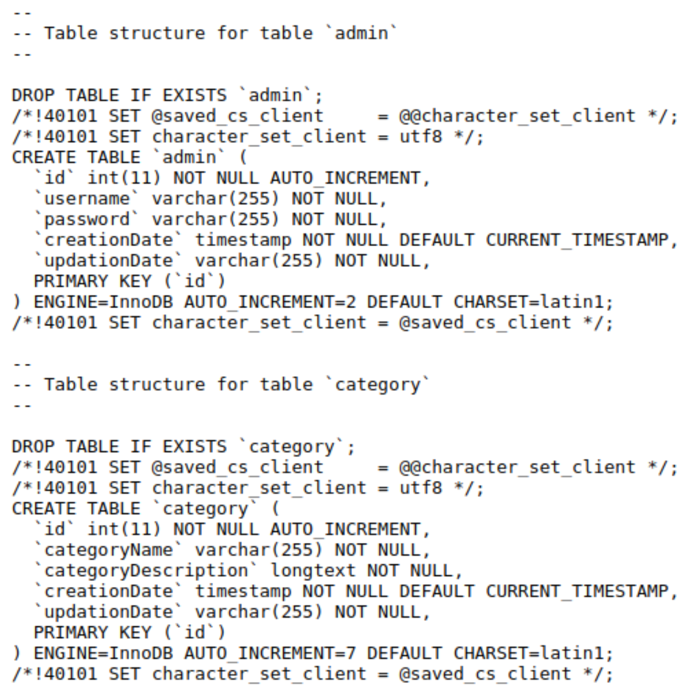
Figure 10-8: Web Application Structure 8

APPENDIX C- SCHEMA.SQL



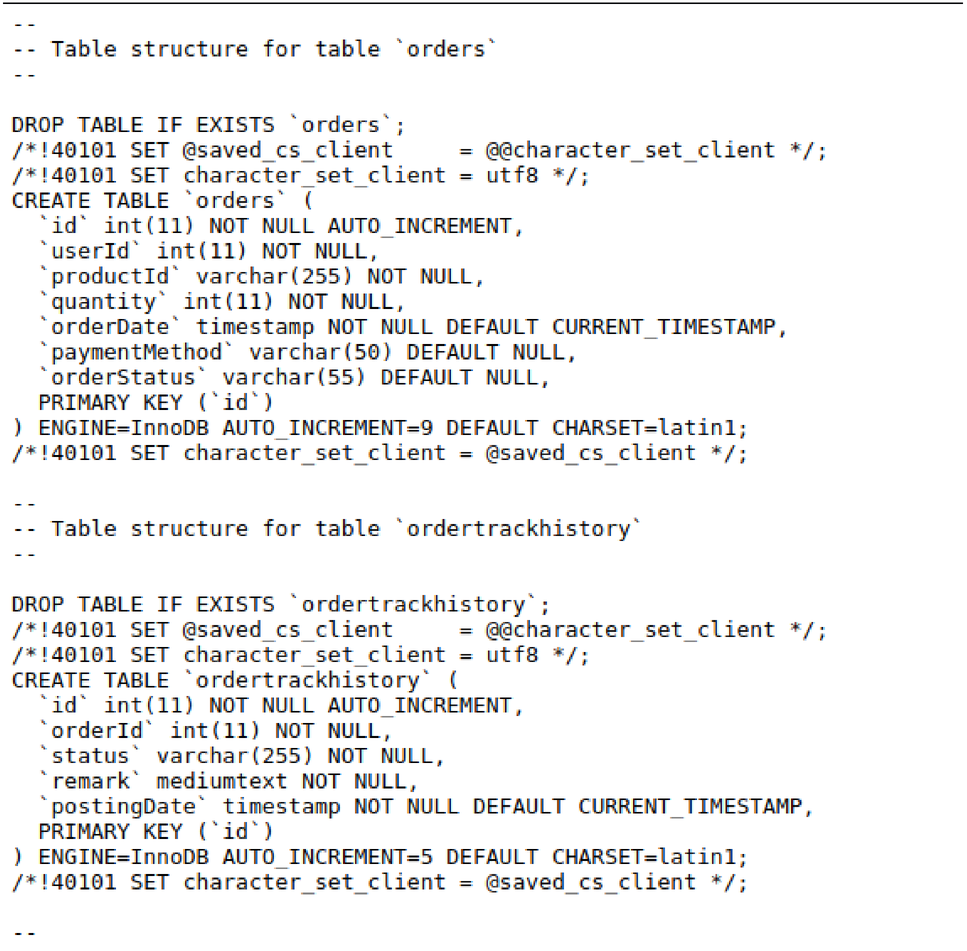
```
-- MySQL dump 10.13  Distrib 5.5.27, for Linux (i686)
--
-- Host: localhost    Database: shopping
-- ------------------------------------------------------
-- Server version	5.5.27

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
```

Figure 11-1: schema 1



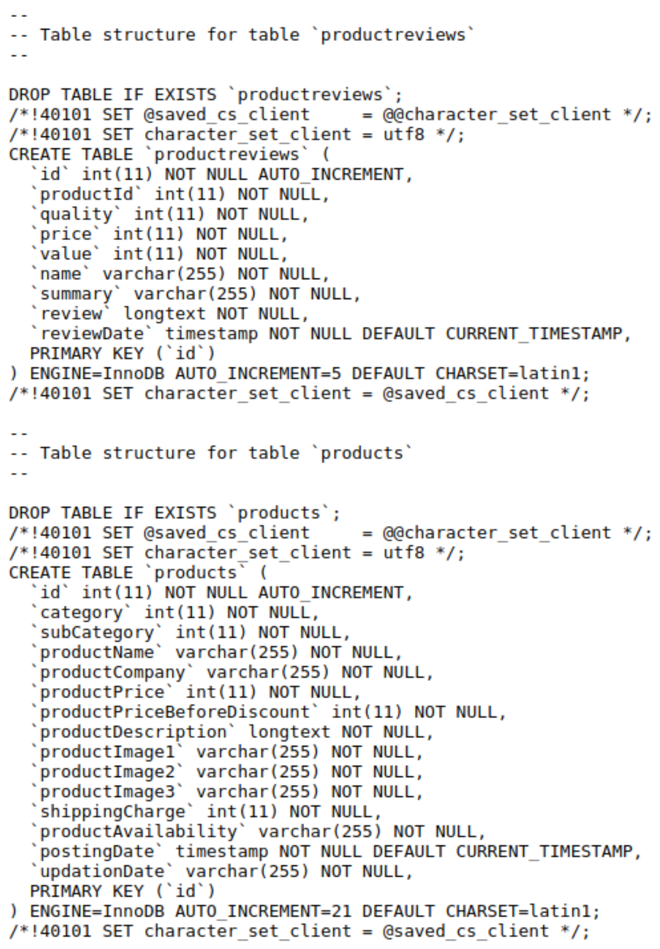
```
--
-- Table structure for table `admin`
--

DROP TABLE IF EXISTS `admin`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `admin` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `creationDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updationDate` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `category`
--

DROP TABLE IF EXISTS `category`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `category` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `categoryName` varchar(255) NOT NULL,
  `categoryDescription` longtext NOT NULL,
  `creationDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updationDate` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Figure 11-2: schema 2



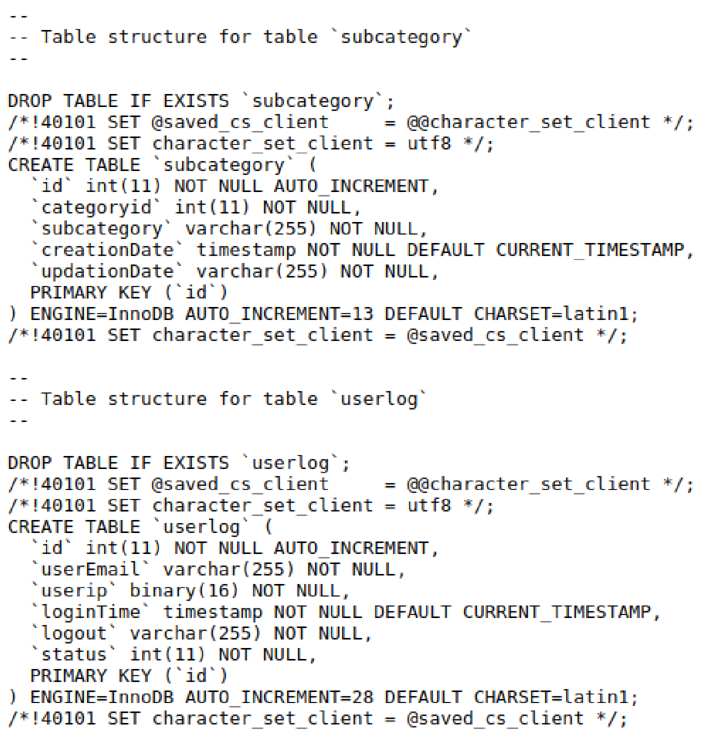
```
--
-- Table structure for table `orders`
--

DROP TABLE IF EXISTS `orders`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `orders` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `userId` int(11) NOT NULL,
  `productId` varchar(255) NOT NULL,
  `quantity` int(11) NOT NULL,
  `orderDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `paymentMethod` varchar(50) DEFAULT NULL,
  `orderStatus` varchar(55) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `ordertrackhistory`
--

DROP TABLE IF EXISTS `ordertrackhistory`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `ordertrackhistory` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `orderId` int(11) NOT NULL,
  `status` varchar(255) NOT NULL,
  `remark` mediumtext NOT NULL,
  `postingDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
```

Figure 11-3: schema 3



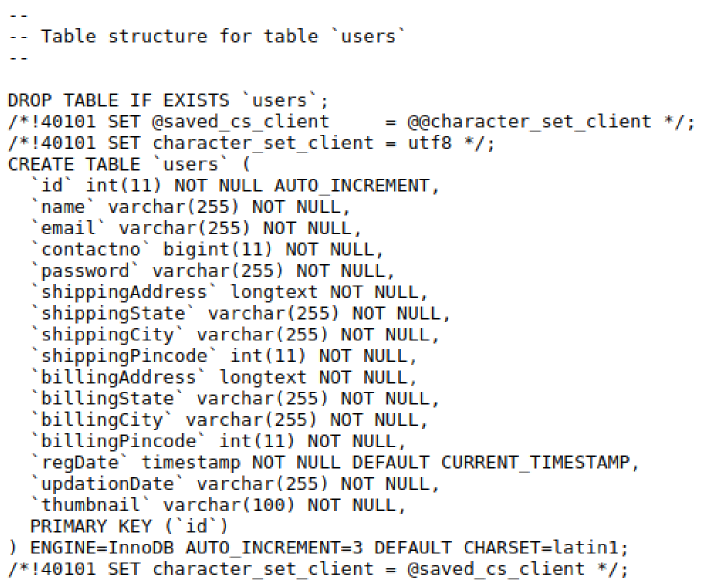
```
--
-- Table structure for table `productreviews`
--

DROP TABLE IF EXISTS `productreviews`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `productreviews` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `productId` int(11) NOT NULL,
  `quality` int(11) NOT NULL,
  `price` int(11) NOT NULL,
  `value` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `summary` varchar(255) NOT NULL,
  `review` longtext NOT NULL,
  `reviewDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `products`
--

DROP TABLE IF EXISTS `products`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `products` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `category` int(11) NOT NULL,
  `subCategory` int(11) NOT NULL,
  `productName` varchar(255) NOT NULL,
  `productCompany` varchar(255) NOT NULL,
  `productPrice` int(11) NOT NULL,
  `productPriceBeforeDiscount` int(11) NOT NULL,
  `productDescription` longtext NOT NULL,
  `productImage1` varchar(255) NOT NULL,
  `productImage2` varchar(255) NOT NULL,
  `productImage3` varchar(255) NOT NULL,
  `shippingCharge` int(11) NOT NULL,
  `productAvailability` varchar(255) NOT NULL,
  `postingDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updationDate` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Figure 11-4: schema 4



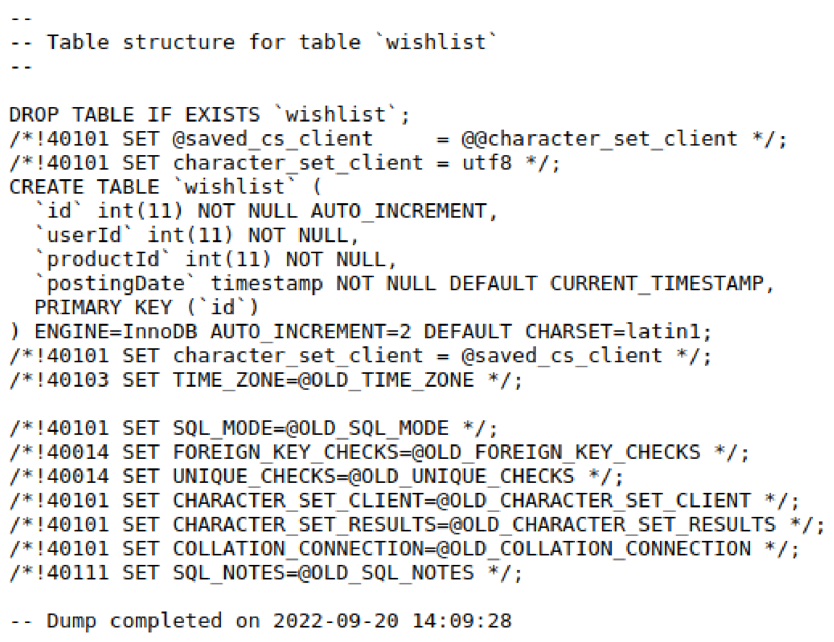
```
--
-- Table structure for table `subcategory`
--

DROP TABLE IF EXISTS `subcategory`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `subcategory` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `categoryid` int(11) NOT NULL,
  `subcategory` varchar(255) NOT NULL,
  `creationDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updationDate` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `userlog`
--

DROP TABLE IF EXISTS `userlog`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `userlog` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `userEmail` varchar(255) NOT NULL,
  `userip` binary(16) NOT NULL,
  `loginTime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `logout` varchar(255) NOT NULL,
  `status` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=28 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Figure 11-5: schema 5



```
--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `contactno` bigint(11) NOT NULL,
  `password` varchar(255) NOT NULL,
  `shippingAddress` longtext NOT NULL,
  `shippingState` varchar(255) NOT NULL,
  `shippingCity` varchar(255) NOT NULL,
  `shippingPincode` int(11) NOT NULL,
  `billingAddress` longtext NOT NULL,
  `billingState` varchar(255) NOT NULL,
  `billingCity` varchar(255) NOT NULL,
  `billingPincode` int(11) NOT NULL,
  `regDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updationDate` varchar(255) NOT NULL,
  `thumbnail` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

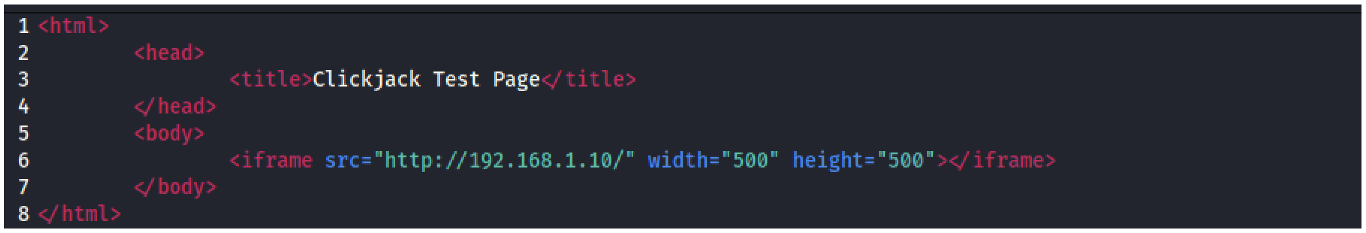
Figure 11-6: schema 6



```
--
-- Table structure for table `wishlist`
--

DROP TABLE IF EXISTS `wishlist`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `wishlist` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `userId` int(11) NOT NULL,
  `productId` int(11) NOT NULL,
  `postingDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2022-09-20 14:09:28
```

Figure 11-7: schema 7

APPENDIX D- CLICKJACKING TEST WEBPAGE CODE



```
<html>
        <head>
                <title>Clickjack Test Page</title>
        </head>
        <body>
                <iframe src="http://192.168.1.10/" width="500" height="500"></iframe>
        </body>
</html>
```

Figure 12-1: Clickjacking Test Code