

|  |
| --- |
| **Somestore Security Evaluation**  An evaluation of the vulnerabilities contained within the Somestore web application.  **By Connor Duncan**  CMP319 Ethical Hacking 3  **BSc Ethical Hacking Year 3**  2018/19 |

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

Abstract

The aim of this paper is to assess all of the vulnerabilities found within the Somestore web application. This paper is a follow on to the penetration test that was undertaken on the Somestore website (C.Duncan, 2018). This paper lists all of the vulnerabilities that were found, along with how to prevent/mitigate them. The vulnerabilities that were found include, but are not limited to, SQL injection, cross-site scripting, information disclosure and file upload vulnerabilities.

+Contents

[1 Introduction 1](#_Toc532903827)

[2 Vulnerabilities Discovered & Countermeasures 2](#_Toc532903828)

[2.1 Robots.txt 2](#_Toc532903829)

[2.2 Local File Inclusion (LFI) 2](#_Toc532903830)

[2.3 Hidden source code vulnerability 4](#_Toc532903831)

[2.4 Reversible cookie vulnerability 5](#_Toc532903832)

[2.5 Cookie Attributes Vulnerability 5](#_Toc532903833)

[2.5.1 Secure 5](#_Toc532903834)

[2.5.2 Domain and Path 6](#_Toc532903835)

[2.5.3 HTTPOnly 6](#_Toc532903836)

[2.5.4 Expires 6](#_Toc532903837)

[2.6 Directory browsing vulnerability 6](#_Toc532903838)

[2.7 User Enumeration vulnerability 8](#_Toc532903839)

[2.8 Unlimited login attempts 8](#_Toc532903840)

[2.9 HTTP Vulnerability 9](#_Toc532903841)

[2.10 File upload vulnerability 11](#_Toc532903842)

[2.11 Cross site request forgery (CSRF) Vulnerability 12](#_Toc532903843)

[2.12 Php information disclosure vulnerability and hidden folders 14](#_Toc532903844)

[2.13 SQL Injection vulnerability. 14](#_Toc532903845)

[2.13.1 Incorrectly filtered escape characters 14](#_Toc532903846)

[2.13.2 Incorrect type handling 15](#_Toc532903847)

[2.14 Brute-forceable Admin password. 17](#_Toc532903848)

[2.15 Cross-site Scripting (XSS) 17](#_Toc532903849)

[2.16 Generic Issues 18](#_Toc532903850)

[2.16.1 X-powered-by 18](#_Toc532903851)

[2.16.2 X-frame-options 18](#_Toc532903852)

[2.16.3 X-XSS-Protection 19](#_Toc532903853)

[2.16.4 GET Apache mod\_negotiation 19](#_Toc532903854)

[2.16.5 Shellshock 20](#_Toc532903855)

[2.16.6 HTTP Trace Method 20](#_Toc532903856)

[2.16.7 Phpmyadmin 20](#_Toc532903857)

[2.16.8 Database content 20](#_Toc532903858)

[2.16.9 Hidden attribute 21](#_Toc532903859)

[3 Conclusion 22](#_Toc532903860)

[3.1 Conclusions 22](#_Toc532903861)

[References 23](#_Toc532903862)

[Appendices 24](#_Toc532903863)

[Appendix A – Screenshots 24](#_Toc532903864)

# Introduction

A penetration test was carried out on the Somestore web application. The result of this test found that the Somestore web application had many security vulnerabilities contained within that varied from a minor risk to a severe risk. The aim of this report is to detail all of the vulnerabilities found within the Somestore web application and highlight the mitigation and prevention techniques that can be used on each vulnerability.

# Vulnerabilities Discovered & Countermeasures

Several vulnerabilities were discovered within the Somestore web application. This section will detail what vulnerabilities there are, what they mean and how they can be mitigated.

## Robots.txt

The robots.txt file is used to tell search engines what pages they are allowed to index on a website and which pages should not be indexed. Any user can access this file by adding “/robots.txt” and the end of the URL. When search engines are used, they will send out ‘spiders’ to search a website and return information which is then used to index the website. Within the robots.txt page, if the command “disallow” is used, then the page which is ‘disallowed’ will not be indexed. In the case of the Somestore website, the site specifies to ‘disallow’ the company-accounts page. This means that the company-accounts page will not be indexed by any search engines. The problem with using the company-accounts page in the robots.txt page is that it contains sensitive information about the company’s customers. Due to the robots.txt page being publicly available, anyone can see this page is disallowed, and navigate to it. The folders found in company-accounts page can be seen in Figure 1.

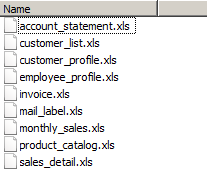


Figure - /company-accounts content

To counteract this security flaw, it is recommended that the company-accounts page is removed from the robots.txt page. Some form of authorization should also be added to this page, so that only users with the correct level of authorization are allowed to view it. For example, a login field could be created when attempting to access this page. Alternatively the page could be moved to the admin page so that only admin users can view it.

In addition to this, in order to increase search engine optimization, pages that do not need to be indexed should be added to the robots.txt page. Pages that do not need to be indexed would include /images/ pages, and /js/ pages.

## Local File Inclusion (LFI)

A Local File Inclusion vulnerability could allow the user to navigate to a location on the webserver and either display the contents of a webserver file, or execute code that is contained within the uploaded file. The addendum.php page is vulnerable to this type of attack. For example, if the user enters /../../../etc/passwd in the URL, then the contents of the passwd file will be displayed. This can be seen in Figure 2.

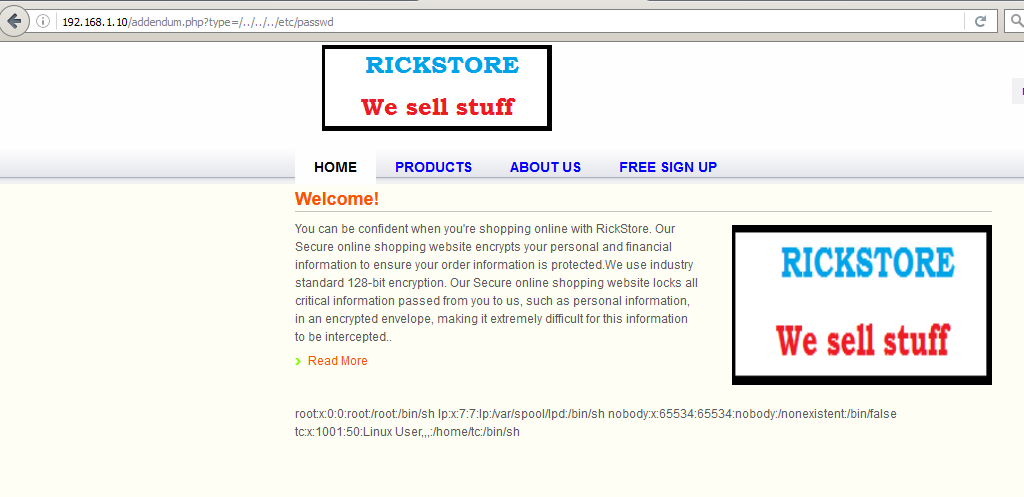


Figure 2 - LFI vulnerability

If an attacker was to find this vulnerability, then it would allow them to access confidential information and potentially run malicious code on the webserver.

The way the code works is it will call the local file “terms.php” and output it to the screen. Therefore, if terms.php is not called, then the new path will be displayed instead. The following code has been identified as the vulnerable piece of code that allows for LFI.

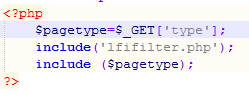


Figure 3 - LFI code

The “lfifilter.php” page is specified to be included, which presumably contains a white or black list for what can/can’t be included in the type parameter. However, upon searching there is no evident that this file exists. The contents of the terms.php page can be seen in Figure 4.

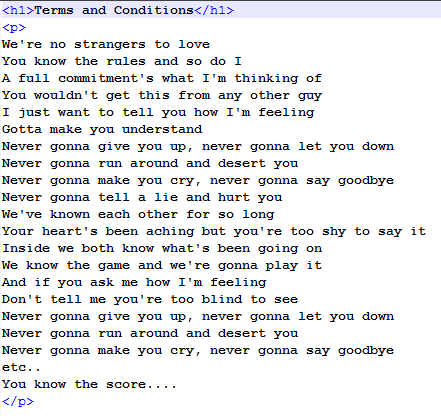


Figure 4 - terms.php

There is no other use for the addendum.php page other than to display the terms.php page. Therefore, it would be recommended to remove the code seen in Figure 3, and directly inject the code seen in Figure 4. As a result, this would remove the LFI vulnerability. Alternatively, a whitelist could be implemented that would result in the URL only accepting known good values, with any other values being rejected. An example of a term that could be included in the whitelist would be “terms.php”. If this was the only item in the whitelist, then all other inputs would be rejected.

## Hidden source code vulnerability

Often web developers insert comments into their pages that disclose information that could potentially be used to assist an attack. Comments that are unnecessary to the running or description of the code should be removed. On the /profile.php page, the comment “<!-- \*\*\* Denis Smith, d.smith@hacklab.com, phone number 01382 99999. Php expert. -->“ is found at the top of the source code. This comment should be removed as it could be used in some form of attack, such as a sophisticated social engineering attack.

## Reversible cookie vulnerability

Cookies can be used to track users that access a website, and information about users can be stored within the cookie. Storing information about users in a cookie can often be dangerous, as if an attacker discovers the encryption formula, then the cookie could potentially be used for session hijacking.

The Somestore website makes use of a cookie called “secret cookie”. Contained within this cookie is an encrypted version of the users’ username and password. This cookie should be removed as it is not used within the application at any point, and therefore serves no purpose. If the cookie is captured by an attacker, then they would be able to discover confidential details about the user if they can work out the decryption technique.

Containing such information within a cookie is not advised because if an attacker acquires the cookie, then they can use it to obtain the user’s username and password. If the password must absolutely be stored in the cookie, then it is recommended that a hashed version of the password is stored. This way if an attacker obtains the cookie then it is much harder, if not impossible, for them to obtain the plaintext password. A hashing algorithm that would be recommended is bcrypt, as it is classed as one of the most secure hashing algorithms.

## Cookie Attributes Vulnerability

Due to cookies being able to contain user sensitive information, cookies must be properly secured to avoid misuse. The way in which cookies are secured is by setting cookie attributes. There are five attributes (Paladion.net, 2018) that can be used to secure the transmission of a cookie. These are:

* Secure
* Domain
* Path
* HTTPOnly
* Expires

These will each be looked into individually.

### Secure

The secure method ensures that the cookie is only sent over an encrypted connection. Therefore, if the cookie has the secure attribute set and the browser needs to send it over HTTP rather that HTTPS, then the cookie will not be sent with the request. This is because it would be travelling over an unencrypted connection. To add in the secure attribute to the cookie, change the “setcookie” function in cookie.php to:

setcookie("SecretCookie", $str, **bool** **$secure = true**);

By adding this attribute, it would require the Somestore application to change from using HTTP to using HTTPS.

### Domain and Path

The domain and path attributes are used to specify what cookies are used/allowed for different domains and paths. By adding in this attribute, it allows for a refined use of where the cookies are used. This means that cookies are not unnecessarily passed into areas where they do not need to be passed, thus meaning that they are less likely to be intercepted. To set these attributes, change the cookies.php page to:

setcookie("SecretCookie", $str, **$Domain=<domain-name>, $Path=<path-name>**);

### HTTPOnly

The HTTPOnly flag is used to stop client-side scripts from accessing the cookie. The reason that this is desired is to mitigate damage created by cross-site scripting (XSS) attacks. If the HTTPOnly flag is not set, then it presents the opportunity for the XSS attacks to steal the user’s cookies. The HTTPOnly will not stop all XSS attacks from gaining the cookie, but it will make it considerably harder to successfully execute them. To set the HTTPOnly flag, change the cookies.php page to include the following:

setcookie("SecretCookie", $str, **bool $httponly= true**);

### Expires

A cookies lifetime will be that of the browser that it is running in – unless otherwise specified. This is the purpose of the Expires attribute. The Expires attribute should not be set for cookies that contain sensitive data however, as if an attacker got ahold of a user’s cookie and were able to use it for session hijacking, then they would remain logged in until the cookie expired – not until the user exits their browser and destroys the cookie. The Expires attribute can be used to set persistent cookies that are not deleted when the user exits the browser. To set an expires cookie, change the cookies.php page to include the following:

setcookie("SecretCookie", $str, **$Expires=<date>**);

## Directory browsing vulnerability

Directory browsing vulnerabilities occur when a user is able to browse to pages that do not have a default file. From there, an attacker is able to gather information that can assist them in a future attack. For example, if the user browses to 192.168.1.10/images, then they are able to view all of the images available on the website. This can be seen in Figure 5.

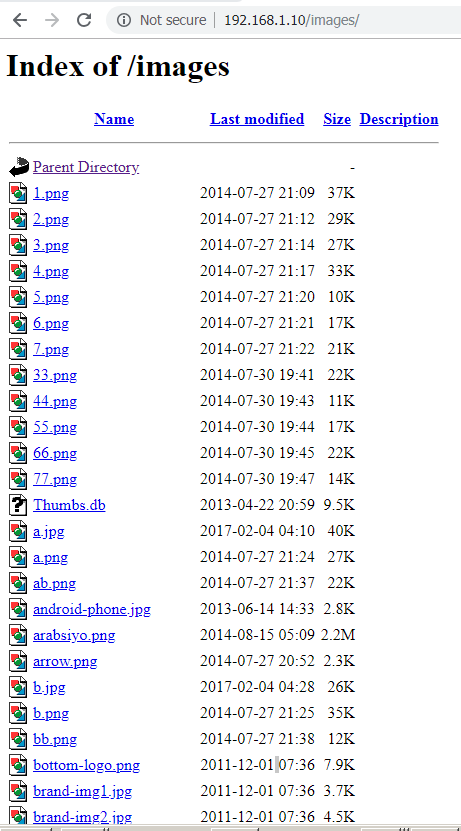


Figure - 192.168.1.10/images

To disable directory browsing, the httpd.conf file must be edited. Within this folder, edit the line that resembles “Options Includes Indexes FollowSymLinks MultiViews” by using the command **sudo pico httpd.conf**. In this line, remove the word “Indexes”, and then restart the apache server using the command **sudo /etc/init.d/apache restart.** Directory browsing should now be disabled.

## User Enumeration vulnerability

Username enumeration occurs when the application gives clues as to whether a username is valid or not. If an attacker can identify when a username is either valid or invalid, then they could set up an attack that would try several different usernames and return when a valid username is found. From here, the attacker would then be able to begin the password cracking phase to gain full access to the account.

On the /login.php page, when a username entered is invalid an alert box appears displaying “user not found”. This can be seen in Figure 6.

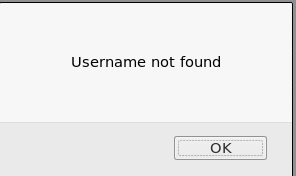


Figure - invalid username

Containing this specific message allows an attacker to identify when an invalid username has been entered, and on the contrary when a valid username has been entered. This box should be edited to display a more generic message, such as “Login attempt failed”. In order for username enumeration to be completely removed, the same message should also be displayed when an invalid password, but valid username, is entered.

## Unlimited login attempts

One of the most common ways for an attacker to try and achieve unauthorized access to an account is through a brute force attack. This involves trying a large number of passwords in an attempt to find the correct one. It would be unreasonable to assume that a legitimate user has attempted to access their account, and failed, hundreds of times. A simple way to bypass this is to add a limit to the amount of times a user can attempt to login before their account is locked. For example, if a user attempts to access their account, and fails, more than five times then the account should be locked, with the user being sent an email on how to unlock their account. However, there is a problem with this method. If an attacker continues to brute force the username, then the legitimate user will continue to be locked out of their account, as every time they reset the password, the attacker will lock them out again.

To combat this, it is recommended that a whitelist of known legitimate IPs is created that the user can login from. This way if the attacker attempts to brute force the username from an unknown IP, their attempts will instantly be blocked even if the correct password is entered. In order for a user to register a new device/IP, they must request a link to be sent to their email, before login, that will add their new IP to the whitelist once activated. This method will then prevent the attacker from being able to brute force the account, as they will not be using an IP within the whitelist, nor will they know what IPs are contained within the whitelist. The account lockout function should still be implemented, as if an attacker is able to acquire one of the IPs within the whitelist then this would still lock the account and prevent the attacker from gaining access.

In addition to this security, a CAPTCHA should be implemented. CAPTCHAs are difficult for computers to understand and successfully complete, but easy for humans. By introducing a CAPTCHA, it will prevent a brute force attack before it has even started, as the computer will be unable to bypass it.

The last step to secure the user login is to include two factor authentication (2FA). 2FA is a type of multi factor authentication. The way 2FA works is it will confirm a user’s identity by using a combination of two different factors. These factors are:

1. Something the user knows
2. Something the user has
3. Something the user is

For example, the username and password is something that the user knows. An email address or phone number is something that the user has and a fingerprint could be something the user physically has.

In the Somestore application, the user could login with their username and password from a new device, and then once logged in an email or text message could be sent to them to verify their identity. If the user acknowledges the message by, for example, entering a code that was sent to them then the user will have been successfully authenticated.

By implementing all of the above methods, the login page should be secure from username enumeration, password guessing and any other form of unauthorized access.

## HTTP Vulnerability

HTTP stands for Hypertext Transfer Protocol and it is a protocol used by the world wide web. This protocol is used to define how messages are composed and transmitted online and will dictate what action should be taken by the web servers and browsers in response to different commands.

An effort to move away from HTTP has begun, with popular browsers now stating when a user is on a HTTP website by displaying “not secure” in the URL bar. This can be seen in Figure 7.

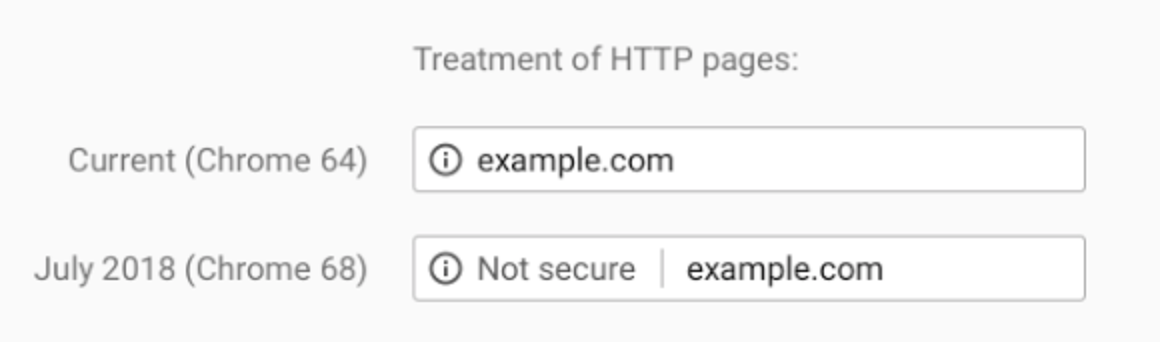


Figure 7 - HTTP URL bar

However, HTTP is starting to become outdated due to its insecurity. The reason HTTP has been deemed insecure is because it does not encrypt the traffic that it sends which makes it vulnerable to sniffing attacks. A sniffing attack is when an attacker steals or intercepts packets that are transmitted across a network using a tool that will capture network packets (called a sniffer). Due to HTTP not making use of encryption, the packets are transmitted in clear text which makes easy reading for an attacker. This could result in sensitive information about the company being disclosed or information being gained that could be used to mount an attack on the network.

To combat these types of attacks, the application should make use of HTTPS. HTTPS stands for Hypertext Transfer Protocol Secure. HTTPS is classed as secure because it makes use of SSL, which encrypts all of the data that is sent across the network. By encrypting the data, it means that if an attacker does use a sniffer on the network, then it will be unable to decipher the data submitted.

In order to implement HTTPS, the web server must have its own dedicated IP address, meaning it cannot be shared with other websites. This is to ensure that all traffic going to the web server’s IP is intended for the Somestore application, and no other website.

SSL works by using certificates. These certificates are made up of a unique series of characters and numbers that form a unique ID for each website. Using HTTPS, when a user connects to the website, the unique ID is checked, and if it matches with the website then all subsequent traffic is encrypted.

An SSL key can be self-made, however the best and easiest way to obtain an SSL certificate is to purchase one from a verified third-party, such as Symantec, VeriSign, or InstantSSL. The reason it is better to purchase one from a third-party website is because most popular browsers check the unique SSL ID with the “Certificate Authorities” (CA), who are then able to vouch for if the SSL ID is legitimate or not. When a third party is used to generate the ID, they automatically tell the CA that a new ID has been created. This then adds an extra layer of security to the website.

Once the SSL has been obtained, a CSR (Certificate Signing Request) must be made. The CSR will contain information such as the company name, domain name and location. It is used to verify and create the SSL certificate. To create the CSR, type the following command into the command line, and fill out all the corresponding sections:

**sudo openssl req -new -newkey rsa:2048 -nodes -keyout Somestore.key -out Somestore.csr**

Once this has been created, the website used to create the SSL certificate should be navigated to, and all of the text in the Somestore.csr file should be copied and pasted into the CSR field of the website.

The WebHostManager (WHM) should then be accessed by navigating to 192.168.1.10:2087. From here, login and then navigate to the “Install an SSL Certificate” section. Then copy the SSL certificate generated by the third-party website into the box provided and press submit.

The pages where user-sensitive data is contained should now be modified so that they can only be accessed through HTTPS. These are pages such as the login page, account details page etc. Therefore, if the user is navigating from these pages, then the page they are coming from must be secure. To ensure this happens, add the following code into the top of the php pages:

**If($\_SERVER[‘HTTPS’] != “on”) {**

**$url = https://. $\_SERVER[‘192.168.1.10’] . $\_SERVER[‘REQUEST\_URI’];**

**header(“Location: $url”);**

**exit;**

**}**

Where REQUEST\_URI is the subdomain that the user should be redirected to if the connection was not HTTPS.

The site should now be fully secured through HTTPS.

## File upload vulnerability

File upload vulnerabilities can result in significant damage to a website or server. If a user can upload an unrestricted file, then the user could potentially gain complete access over the website and server. A malicious user may attempt to upload php code that they can then connect to, resulting in them gaining access from the server side. The end result of an unrestricted file upload depends on where the file is stored and what the application does with the file.

There are two problems that can arise. The first problem comes from file metadata, such as the path and filename. The data contained within the uploaded files could potentially trick the application into overwriting a critical file on the server. If the file is stored in a bad location, then the file could potentially cause unintended harm to the server.

The second problem comes from the file size and/or content. A wide range of attacks can come from the content of the file and the file size. The damage cause by these attacks depends on what the attacker has specified within the file and how the server processes files.

On the account details page of the Somestore application, the user is able to upload a file as their profile picture. From looking at the source code, checks are implemented to make sure that the file is a valid type. The accepted files are “jpeg”, “jpg” and “png”. This can be seen in Figure 8.

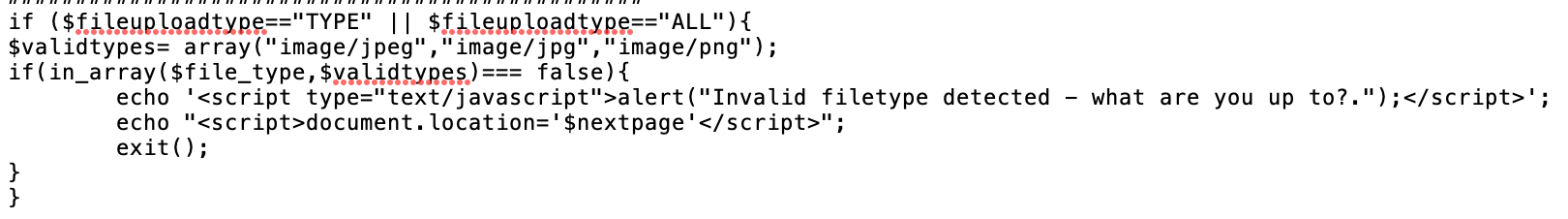


Figure 8 - changepicture source code

If any other type of file is inserted then the file is rejected and an alert message is displayed. However, these checks occur on the client side and not the server side, thus leaving them vulnerable to interception and modification after the client-side checks have been implemented, but before the file reaches the server. Therefore, if a file is called “MaliciousCode.php.jpg”, then it will pass the client side check, but can then be intercepted before reaching the server, and changed to “MaliciousCode.php”. This will then bypass the check and be upload to the server as php code.

In order to secure the file upload function, many different techniques must be used. To secure the file upload, the file upload function must:

* Disable php from running inside the file.
* Only accept the image file type
* Only accept images of type “jpeg”, “jpg”, “gif” or “png”
* Not contain the word “php” in the title
* Disallow images with two file types
* Change the image name

There is a script on GitHub that contains a secure file upload function. This script can be found at <https://stackoverflow.com/questions/38509334/full-secure-image-upload-script>

This script ensures that all of the points above are implemented and that several other possible vulnerabilities are prevented too. It is recommended that this script is copied and pasted to be used on the file upload page.

## Cross site request forgery (CSRF) Vulnerability

CSRF attacks are where a user that is authenticated on a website is forced to execute unwanted actions on the application. With CSRF attacks, the attacker has no way to see the result of the attack, and they do not result in any theft of personal data. Instead they result in changing the state of something on the application. With this type of attack, an attacker may force the user into executing actions of the attacker’s choice, such as changing their password. A CSRF attack will exploit the trust between a browser and a web server. If a unique identifier can be found that links the website to an account, then CSRF can be used to further exploit the account. This identifier could come in the form of a cookie, such as the PHPSESSID cookie on the Somestore application.

The best way to secure an application against CSRF attacks is by using CSRF tokens. Both the browser and server know what the token is, and they are a unique to each users’ session. Tokens should be large random values that are generated using cryptographically secure random number generators. For data submitted using forms, the CSRF token is submitted in a hidden form. When the form is submitted, the token should match the token for that users’ session. If the tokens do that match, then the form will be rejected.

In order to create the CSRF token, enter the following code(PHP, 2018) into the php pages that contain forms:

**session\_start();**

**if (empty($\_SESSION[‘token’])) {**

**if (function\_exists(‘mcrypt\_create\_iv’)) {**

**$\_SESSION[‘token’] = bin2hex(mcrypt\_create\_iv(32, MCRYPT\_DEV\_URANDOM));**

**} else {**

**$\_SESSION[‘token’] = bin2hex(openssl\_random\_pseudo\_bytes(32));**

**}**

**}**

**$token = $\_SESSION[‘token’];**

This code will create a randomly generated, cryptographically secure 32 byte token.

An if statement is then required to encompass the form fields. The code to do this is as follows:

**if (!empty($\_POST[‘token’])) {**

**if (hash\_equals($\_SESSION[‘token’], $\_POST[‘token’])) {**

**//enter the form data here**

**}**

**else {**

**//reject the form**

**}**

**}**

This statement will then ensure that a token is present, before submitting the form. If no token is present, meaning that CSRF attacks would be possible, then the form is not sent. This implementation should be used on all pages that process forms.

## Php information disclosure vulnerability and hidden folders

The phpinfo.php page displays information about the configuration, software versions and paths on the web application. A large amount of information contained within this page can be used to assist an attacker. For example, knowing the apache server version allows them to search for vulnerabilities for that server version. Knowing a path name allows the attacker to find out where specific files are stored, which can assist an attack. All of this information allows an attacker to create a more sophisticated and specially crafted type of attack. It is recommended that this page is either completely removed, or only available to users with certain access levels e.g. admins.

Hidden folders that are not intended to be viewed by users can also be found using tools such as dirbuster. These tools test generic folder names for their existence. In this case, the hidden folder 192.168.1.10/vb was found. Contained within this directory is a file called sqlcm.bak, which contains information in regards to the SQL injection prevention measures. This can be seen in Figure 9.

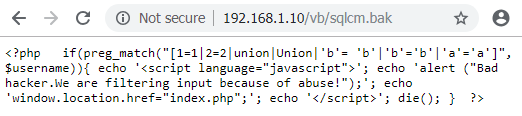


Figure - Sqlcm.bak

The same process as above should be used for this folder, and it should either be taken down, or only available to users with a certain access level. All other folders that contain information about the inner workings of the application should also be removed/restricted.

## SQL Injection vulnerability.

SQL Injection (SQLI) is a form of code injection. When an application submits a SQL statement, if the data is not correctly filtered then it could be vulnerable to SQL injection. This means that by using specially crafted payloads, the SQL statement used to submit the original data could be used to submit a completely different request than the developer intended. This can then lead to the attacker dumping or deleting the databases entire contents, modifying data within the database and/or adding data to the database. Because of this, SQL injection is viewed as one of the most severe vulnerabilities that a website can have.

SQLI can occur because of two reasons, incorrectly filtered escape characters and incorrect type handling. These will be discussed below.

### Incorrectly filtered escape characters

An escape character is a character that creates an alternative interpretation on the following characters. For example, the following SQL statement (En.wikipedia.org, 2018) can be vulnerable to SQLI as it does not correctly filter escape characters.

Statement = “SELECT \* FROM users WHERE name = ‘ ” + username + “ ’; ”

This statement will select all of the records for a user with a specific username. However, if a specially crafted ‘username’ is entered, then more than one users’ record could be pulled up. For example, if the user entered **‘ OR ‘1’=’1** then the statement will read as:

SELECT \* FROM users WHERE name = ‘ ‘ OR ‘1’= ‘1’;

This would then result in all users’ records being dumped from the database, rather than only one user as intended. This is just one example of how incorrectly filtering escape characters can cause damage to a database.

From here the attacker can edit the statement/add on additional statements that can cause further harm/leakage to the database.

### Incorrect type handling

This type of SQL injection will arise when the language used is not strongly typed, or the statement is not applying type constraints. A strongly typed language is a language that strictly enforces the language and syntax used, and therefore is more likely to throw an error if they syntax is not as expected. If a statement does not use type constraints, then if an unexpected type is entered then it will still be accepted. For example, strings being allowed to be entered into a field that is specified as type int. An example of this would be in the following statement:

Statement := “SELECT \* FROM userinfo WHERE id = “ + a\_variable + “;”

The variable “a\_variable” is to take in an ID number, and is therefore expecting an int. However if the user enters **1; DROP TABLE users**, then the statement will read as follows:

SELECT \* FROM userinfo WHERE id=1; DROP TABLE users;

This statement will then result in the entire ‘users’ table being deleted.

It was found that the login page uses a filter in an attempt to block SQLI attacks. It has been proven that using a filter is often ineffective, as there are ways to bypass the filters. For example, the filter found in sqlcm.php specifies not to allow attempts that say “1=1” or “2=2”. This can easily be bypassed by saying “3=3” in the SQL injection code.

To mitigate SQLI attacks, the source code should be modified. The source code should make use of prepared statements. Prepared statements make use of variable binding and are called parametrized queries. The way in which parameterized queries work is that all the SQL code is defined first, and then the parameters are passed into the code at a later stage, rather than directly into the SQL code. This allows the database to distinguish between code and data, meaning that regardless of what the user input is, it will always be interpreted as data, not code.

The login page currently uses the code seen in Figure 10 to gather a customer’s records and log them in.

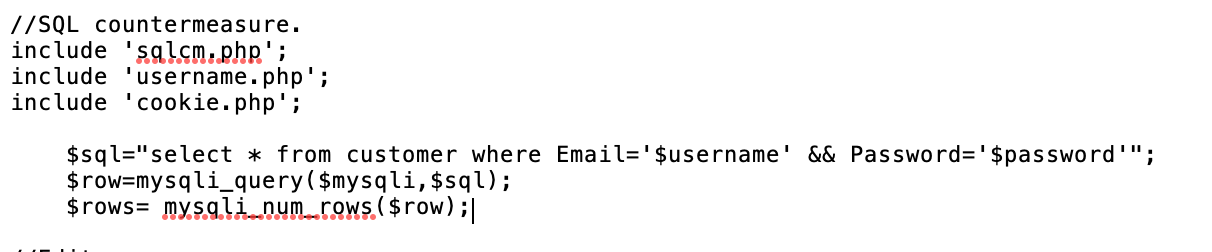


Figure - SQL login code

This does not filter the user input correctly and is vulnerable to SQL injection. The code seen in Figure 11 is code that should be used instead, as it is not vulnerable to SQL injection.

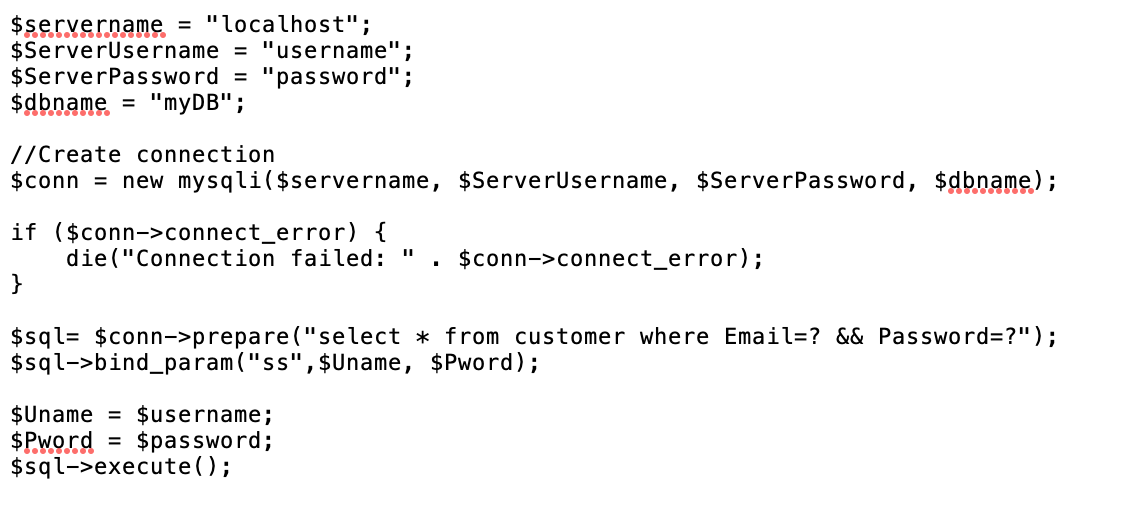


Figure - Login page prepared statements

The code above successfully filters the input. The servername, ServerUsername, ServerPassword and dbname should all be amended as required.

A connection is made to the server using the variable $conn. This connection is then tested to make sure that it is set up correctly. The prepare statement is then set up, with a “?” replacing the variables that were used previously in Figure 8. The statement is then bound together. The “ss” parameter specifies that the following two variables are integers. The variables $Uname and $Pword are then assigned the values of the username and password respectively. The sql statement is then executed and sent to the server.

These prepare statements should be used where a user is able to enter data that will be used in a statement to the SQL database.

## Brute-forceable Admin password.

If a website contains admin level accounts, then an attacker will often seek details for these accounts, as they will often contain information about the website and its users. Admin accounts are also usually able to edit features and functionality on the website. For obvious reasons, it then becomes a problem if an attacker can gain access to an admin account. Therefore, admin passwords should be difficult to guess.

On the Somestore web application, the password for the account admin is renee. This password is not very long or complicated, and therefore leaves it vulnerable to brute force and dictionary attacks. This password should be changed to a longer and more complicated password. A good example would be “!Adm1nPa55#w03D.” This password is long, contains upper case and lowercase letters, contains special characters and contains numbers. This password would be extremely difficult to guess from any type of password cracking attack. However, due to this password being displayed in this paper, it should only be taken as an example.

It is also recommended that there is not a separate admin section for the admins to login. The admin should be able to login using the same login form as a normal user. By separating the two forms, it makes it more likely that an attacker will be able to identify when a username is that of an admin, rather than a normal user.

## Cross-site Scripting (XSS)

To prevent against XSS attacks, several rules should be implemented. The first rule is to never allow untrusted data (data crafted by the user) into areas that should not allow untrusted data. Examples of this can be seen in Figure 12, taken from the OWASP XSS prevention cheat sheet (Owasp.org, 2018)

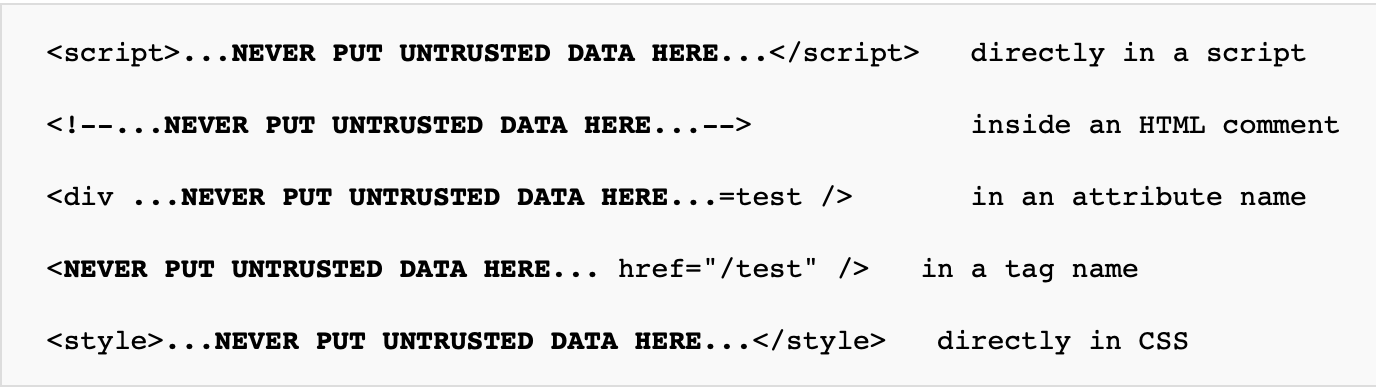
. 

Figure - untrusted data locations

The second rule is that before user data is inserted into the body of the html file, all characters should be escaped using HTML entity encoding. Figure 13 shows the characters that require escaping.

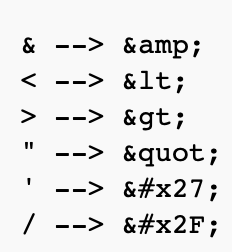


Figure - HTML escape characters

The third rule is to escape characters before they are put into attribute values, however untrusted data should never be put into complex attributes such as src, href, style or any event handlers like onmouseover. In addition to this, when putting characters into attributes, unless they are alphanumeric characters, all characters that have an ASCII value of below 256 should be escaped using the &#xHH format.

The fourth rule is that if the data will be placed into the URL, then it should be URL encoded.

By following these rules and changing the X-XSS-protection header (See section 2.16.3), the site should be secure from XSS.

## Generic Issues

There are several other flaws within the web application that will be discussed in this section.

### X-powered-by

The X-powered-by header is included in responses from the server. The purpose of this header is to display the framework that the server uses. In this case – PHP/5.4.7.

This information can prove useful to an attacker, as it can allow for a more targeted attack against the application and server. Therefore, the X-powered-by header should be removed.

To remove the header, navigate to a terminal for the web server. Once a terminal has been acquired, search for the php.ini file. By default, this file is located in /etc/php5/fpm/php.ini. Once this file has been found, run the **sudo** **pico** command to edit the contents of the file.

Search for the **expose\_php** attribute within the file, and then set this to **expose\_php = OFF**. Once this has been edited, restart the service using the command **sudo service php5-fpm restart**.

The X-powered-by header should no longer be displayed in the server response.

### X-frame-options

The X-frame-options header is not present within the server responses. This header is used to prevent click-jacking attacks, by ensuring that the websites content is not embedded within other websites.

A click-jacking attack is when an attacker creates a transparent iframe link, that when clicked on redirects the user to a website of their choice, or an identical looking website that they control. Often these links are placed over other valid links on the page, such as the “login” button, thus tricking the user into clicking on the link.

The header has three settings: deny, sameorigin and allow-from uri

The deny option denies any website from framing the content. Meaning that no website can embed the specified websites content within their website.

The sameorigin option allows content from within the same website to be framed.

The allow-from uri allows a specified website to frame the websites content. For example, the allow-from <http://www.example.com> would allow the websites content to be framed within [www.example.com](http://www.example.com).

It is recommended that the x-frame-options header is included in the server responses for all pages, with the sameorigin option being specified. To do this, the httpd.conf file must be edited on the server. On the server, use the **sudo pico httpd.conf** command to edit the file, and add in the line **header always set x-frame-options “SAMEORIGIN”**. The x-frame-options header should now be seen within the header response from the server.

### X-XSS-Protection

The X-XSS-Protection header is used to prevent some types of cross-site scripting (XSS) attacks by using an XSS filter. Instead of sanitizing the response from the XSS, this header will instruct the browser to block the response from the malicious code. The syntax **x-xss-protection:1; mode=block** should be used. This tells the browser that if an XSS attack is detected, to prevent any rendering of the page.

To enable the x-xss-protection header, the httpd.conf file again needs to be edited. Using the same command as previous, **sudo** **pico httpd.conf**, add the following line in to the file:

**Header always set x-xss-protection “1; mode=block”**

The x-xss-protection header should of now be present in the server header responses.

### GET Apache mod\_negotiation

GET Apache mod\_negotiation is enabled with MultiViews, allowing an attacker to brute force filenames. By enabling MultiViews, it means that if an attacker searches for only a partial filename, then the application will look to find the closest match.

For example, if 192.168.1.10/index is entered in the URL, with the accept option in the header set to **\*/\***, then the website will find the page /index.php. This means that MultiViews is enabled on the / directory. Therefore if a user sends a specially crafted packet to 192.168.1.10/index, with the accept header set to **application/example; q=1.0**, then the application will reply with all of the possible permutations of /index, thus revealing a lot of hidden directories. This process could be repeated to search for other hidden pages.

In order to stop this, MultiViews must be disabled. To disable MultiViews, navigate to the **httpd.conf** folder, and run the command **sudo** **pico httpd.conf**. Then search for the section starting <Directory “Library/WebServer/Documents”>, change the line starting **Options** and remove the word “MultiViews”.

The webserver should then be restarted with the command **sudo** **/etc/init.d/apache restart.**

### Shellshock

The /cgi-bin/printenv folder is viewable to all users. A vulnerability has been previously found within the bash version of /cgi-bin/. This vulnerability is called shellshock and if exploited successfully then it can give an attacker access to the webserver.

In order to secure the web server against shellshock, the bash version needs to be upgraded. To upgrade the bash version, the command **sudo apt-get update && sudo apt-get install** should be run on the server. This will upgrade the bash service to the latest version and patch the shellshock exploit.

### HTTP Trace Method

The HTTP Trace method is designed to be used for diagnostic purposes. If it is enabled, then the server will respond to any requests that’s use the trace method by echoing in its response the request that was received. However, the trace method is vulnerable to Cross-site tracing – a form of cross site scripting. Therefore, it should be disabled to avoid any users exploiting this method.

To disable the trace method, locate the **httpd.conf** folder on the server. Run the command **sudo pico httpd.conf** to edit the file and at the bottom of the file inserted a line saying **TraceEnable off**. Then restart the apache server using the command **sudo service httpd restart**. The server will no longer make use of the trace method.

### Phpmyadmin

The phpMyAdmin page, used to handle the administration of mySQL, is viewable by navigating to 192.168.1.10/phpMyAdmin. This page should not be displayed publicly and if an attacker can guess the username/password then the attacker can cause serious harm to the database.

It is recommended that this page is not directly viewable to the public and is either removed or placed behind an area that requires special access e.g. admin

### Database content

Upon reviewing the database content, it was found that the database stores information in plain text. This is bad practice as if an attacker gains unauthorized access to the database then they can view all of its contents. For example, if the database was currently compromised, then the attacker would be able to gain all of the usernames and passwords that are held on the Somestore application.

To combat this, it is recommended that only a hashed version of the sensitive data is stored. A hash function will take an input and return a fixed-size alphanumeric string which is irreversible. For example, if the word “password” was to be ‘hashed’, using the sha256 hashing algorithm, then it would translate into “5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8”.

When a query is then made to the database, for example for a users’ password during the login phase, then the password being searched for would need to be hashed using the same hash function as used previously, and compared to the password hashes stored in the database. By doing this, it means that if the database is compromised, then the amount of data revealed is restricted and the attacker would not have access to all user’s accounts.

### Hidden attribute

It was identified that the hidden attribute is used to pass through sensitive data. This attribute should not be used as it is available for editing by the user. For example, the change password function uses the hidden attribute to pass through the users’ email address. This email address can then be changed, and if another account is known, then the email for the known account can replace the current email in the hidden field. This would then change the password for the other users account. See the previous paper, Somestore Web Application Investigation by Connor Duncan, for further details on this matter. In order to fix this, an if statement should be added to the code, specifying that if the original email does not match the email contained in the current form, then do not allow the request to continue. This will then stop an attacker from being able to change other users’ passwords.

# Conclusion

## Conclusions

In conclusion, the Somestore web application has several severe vulnerabilities that need patching. However, all of these vulnerabilities can be patched without too much difficulty. Many of the vulnerabilities found within the application are very common vulnerabilities that are often the most commonly exploited, such as SQL injection, Cross-site scripting and information disclosure. With the guidance provided in this document, these vulnerabilities should be severely mitigated.

# References

Owasp.org. (2018). *Cross-Site Request Forgery (CSRF) - OWASP*. [online] Available at: https://www.owasp.org/index.php/Cross-Site\_Request\_Forgery\_(CSRF) [Accessed 16 Dec. 2018].

PHP, H. (2018). *How to properly add CSRF token using PHP*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/6287903/how-to-properly-add-csrf-token-using-php [Accessed 16 Dec. 2018].

En.wikipedia.org. (2018). *SQL injection*. [online] Available at: https://en.wikipedia.org/wiki/SQL\_injection [Accessed 17 Dec. 2018].

Owasp.org. (2018). *XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP*. [online] Available at: https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet [Accessed 17 Dec. 2018].

Connor Duncan - Somestore Web Application Investigation, An investigation into the vulnerabilities in the Somestore application.

Paladion.net. (2018). *Cookie Attributes and their Importance*. [online] Available at: https://www.paladion.net/blogs/cookie-attributes-and-their-importance [Accessed 17 Dec. 2018].

Owasp.org. (2018). *Blocking Brute Force Attacks - OWASP*. [online] Available at: https://www.owasp.org/index.php/Blocking\_Brute\_Force\_Attacks [Accessed 18 Dec. 2018].

# Appendices

## Appendix A – Screenshots

