Abertay University

"Food Plaza" Web Application Security Evaluation
CMP 319 - Ethical Hacking 2

BSc Ethical Hacking
2018/19

Jack Clark
1601798@uad.ac.uk

1. Introduction

The Web Application "Food Plaza" has previously been subject to a vulnerability assessment, detailing the steps taken and vulnerabilities found that could be exploited by an attacker. At the request of Food Plaza, this document is being produced as a Security Evaluation to include further depth on the vulnerabilities found and present mitigations to ensure that an attacker has less chance of exploiting the application.

2. Vulnerabilities and Countermeasures

2.1.Robots.txt Disclosure

The web application makes use of a `robots.txt` file to stop search engines accessing the directories listed and potentially showing them in search results. The `robots.txt` file for the application is shown below:

```
User-agent: *
Disallow: /company-accounts
```

FIGURE 1 ROBOTS.TXT FILE

Although using a `robots.txt` file is a good idea and safe in the sense that web crawlers won't visit the directory therefore it won't be displayed in a web search, it is a severe security risk to include sensitive directories in this document as it can be viewed by anyone accessing the application. To avoid both crawlers from accessing a file while not exposing it in the `robots.txt` file the following line can be added to a page within the <head> tags of the `HTML` (Using the robots meta tag, 2018):

```
<meta name="robots" content"no index, no follow" />
```

FIGURE 2 PREVENT VIEW OF FILE USING META TAGS

The above line is useful when the item being hidden is a page, however if it is a file, for example with the finances.zip, or a directory, a `.htaccess` file can be created with the following lines:

```
Options -Indexes
RewriteEngine On

RewriteCond %{HTTP_USER_AGENT} (googlebot) [NC]
RewriteRule .* - [R=404,L]
```

FIGURE 3 .HTACCESS FILE

The above lines include code to block directory listing, which will stop a user being able to access the `/company-accounts/` directory (Limited, 2018), and after that code to prevent bots from crawling the directory (Limited, 2018). Currently only `googlebot` has been entered, for demonstration purposes, however more can be added. Both sections of code respond with a `404 Object Not Found!` page. Once the `.htaccess` file has been added to the root of the application, the directory can be removed from `robots.txt` file.

To note, the `httpd.conf` file may need to be modified to allow the use of `.htaccess` files. To do allow the use of the file, the following line will need to be modified as follows:

```
 <Directory "/Applications/XAMPP/xamppfiles/apache2/htdocs">
-    Options Indexes FollowSymLinks
+    Options FollowSymLinks
     AllowOverride All
     Order allow,deny
     Allow from all
```

FIGURE 4 UPDATED HTTPD.CONF FILE

2.2.Local File Inclusion

Local File Inclusion (LFI) is found in the `extras.php` page. LFI occurs when a page is dynamically passed to the application as a parameter. It can be seen in the URL for the page:

`http://192.168.0.10/extras.php?type=terms.php`

The vulnerability exists that an attacker can change the page that is passed and if improperly sanitised then this can result in a different file on the server being displayed. The application includes the following filter to attempt to sanitise input (included from `lfifilter.php`):

```
$pagetype = str_replace( array( "../", "..\""  ), "", $pagetype);
```

FIGURE 5 LFIFILTER.PHP EXTRACT

The filter will remove `../` and `..\` from the input and replace it with a blank space. The issue with this is that if a page is requested that doesn't make use of either of the variants, then this won't be filtered. For example, the parameter could be `/etc/passwd` which won't be sanitised and the file will be displayed,

The strongest mitigation would be to include the file within the source code for the page. This will suffice as the page isn't dynamic at all, which means using a parameter passed to the application is pointless.

The following changes to the code can be made to achieve this:

```
 <?php
   $pagetype = $_GET['type'];
    include('lfifilter.php');
-    include ($pagetype);
+    include ('terms.php');
 ?>
```

FIGURE 6 UPDATED INCLUDE STATEMENT IN EXTRAS.PHP

By modifying the `extras.php` file like the above figure, this will ensure that nothing is needed to be passed as a parameter, therefore making it impossible to use LFI as an attack vector.

2.3.Hidden Source Code

Within the source code for the page `foodzone.php` there lies a comment that contains personal information:

```
<!-- *** Denis Smith, d.smith@hacklab.com, phone number 01382 99999. Php expert. -->
```

FIGURE 7 FOODZONE.PHP SOURCE CODE COMMENT

Leaving a comment within the source code of the page that contains sensitive or personal information can help an attacker during an attack. This also opens up an avenue for social engineering attacks in this case as the comment has contact information and a name for, what can be assumed, the developer of the page.

To mitigate this risk, it is normally best practice not to write sensitive information in a file like this to begin with. However if it must be done for any reason, it is best practice to thoroughly analyse the file before publishing to anywhere public. This could be done manually or using a tool such as `grep` to search for a pattern that appears in comments across the file. For example, using `grep` to search for `<!--` would result in the above comment being returned.

2.4.Reversible Cookie

The web application makes use of cookies when a user logs in and while they are traversing through the website. An example of which is shown below:

```
6861636o6p6162406861636o6p61622r636s6q3n686163606p61623n31353339363033363435
```

FIGURE 8 EXAMPLE SECRET COOKIE

The issue with the cookie is that it is based on the currently logged in users username and password, and that it can easily be decoded. From analysing the source code provided, the code used to create the cookie was found and is shown below:

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

FIGURE 9 COOKIE ENCODING CODE

The code above shows that the cookie contains the variables `$username` and `$password`, which refer to the users username and password. As mentioned before, this already is a security issue. Further more, there is the use of `str_rot13(bin2hex($str))`, which means that the cookies plaintext is converted to hex and then ROT13, neither of which are secure.

Due to the simplicity of the cookie encoding, a tool such as CyberChef (CyberChef, 2018) can be used to guess the encoding types, which when the correct ones are found an attacker will have a username and a password for an account.

To avoid this occurring, both strong encryption and cookie signing should be used. A recommended encryption method is using `bcrypt` with a random, cryptographically secure value as the salt.

2.5.Cookie Attributes

The application as previously mentioned makes use of ookies for authentication to some pages. It however doesn't set any attributes to protect the cookie. This means that there are no protections on the cookie, which is key given how weak the encoding of the cookie is.

Two attributes can be used to secure the cookie (PHP: Runtime Configuration - Manual, 2018):

- `HTTPOnly`: This attribute will allow a cookie to be accessed using only the HTTP protocol (this includes HTTPS). This will assist in preventing attacks such as Cross-Site Scripting which uses JavaScript.

- `Secure`: The Secure flag will ensure that a cookie is only transferred using a secure connection. For this to work however, HTTPS must be configured on the web application (see Section 2.8).

These flags can be set in the `php.ini` file of the Application with the following parameters:

```
session.cookie_secure = True

session.cookie_httponly = True
```

FIGURE 10 SETTING COOKIE ATTRIBUTES

If the `php.ini` file can't be accessed, the cookie.php file can be edited with the following:

```
-$str=$username.':'.$password.':'.strtotime("now");$str = str_rot13(bin2hex($str
)); setcookie("SecretCookie", $str, secure, httponly);
+$str=$username.':'.$password.':'.strtotime("now");$str = str_rot13(bin2hex($str
)); setcookie("SecretCookie", $str, null, null, null, true, true);
```

FIGURE 11 UPDATED COOKIE.PHP FILE

2.6.Cookie Logout Vulnerability

When a user logs out of their profile, their cookie isn't unset from the server. This means that a user can log out and an attacker can still gain access by using SQL Injection to bypass the need for a username or password and is authenticated with the left behind cookie. An attacker can also capture the cookie after the user has logged out and, given its reversibility, retrieve the users credentials from it.

To resolve this vulnerability, the cookie should be unset in the `logout.php` file. The cookie could also be initialised with an expiry date. This means that if for some reason the cookie was still set after a user logs out then the cookie is only valid for a small length of time.

2.7.Directory Browsing

Directory browsing is enabled in the server. This means that a tool such as DirBuster can be used to enumerate directories on the application. To avoid this, the same steps can be used as mentioned in Section 2.1.

2.8.Username Enumeration

The Application makes use of a "Username not found" alert when a username is entered incorrectly. The security vulnerability is that when a correct username is entered with an incorrect password the application redirects to another page reading "Login failed". This means that an attacker using a tool such as BurpSuite Intruder (Burp Suite Scanner | PortSwigger, 2018) can brute force the username and determine whether or not the username is correct based on whether or not the page is redirected or not.

To avoid this issue, it is always best practice to be vague when incorrect credentials are used. For example, if either the username or password is incorrect always make use of a page that explains to the user that the username or password is incorrect and not specify which of the two is incorrect. It is always key to also make use of the same type of alert, whether that be a popup or a redirection.

2.9.Unlimited Login Attempts

The Application also allows for unlimited login attempts with no lockout policy, therefore further allowing for an attacker to brute-force a login attempt. With the mitigations put in place from the

previous section, it still means that an attacker could still have a chance of brute forcing the password, if for example they had managed to retrieve a username.

The best practice to avoid this would be to implement a lockout policy for entering a password incorrectly too many times. For example, a policy of 10 incorrect attempts before an account is locked and needs to have the password reset via an email would be recommended.

2.10.No HTTPS

HTTP is used throughout the website with no use of HTTPS at all. This means that all traffic going between the client and server is unencrypted and in plaintext. Any credentials that are transmitted to the Server are subject to being intercepted and read if there is a Man-in-the-Middle (MITM) capturing data passing between the client and server. This can be seen below, captured using a proxy from Burp Suite:

```
POST /login-exec.php HTTP/1.1
Host: localhost
Content-Length: 57
Cache-Control: max-age=0
Origin: http://localhost
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://localhost/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Connection: close

login=hacklab%40hacklab.com&password=hacklab&Submit=Login
```

FIGURE 12 PLAINTEXT CREDENTIALS BEING TRANSMITTED

As can be seen in the above Figure, the username and password of an account can be read. If an attacker managed to gain access between the server and client then the attacker would have access to the account.

To ensure that the data transmitted is secure, the application must be setup to use HTTPS. To do so, a Certificate Authority, such as Let's Encrypt (Let's Encrypt - Free SSL/TLS Certificates, 2018) which is becoming more popular, should be used to generate a certificate that can be used for a secure connection. From there both the generated `.key` and `.crt` files can be copied to the server and the following lines added to the `httpd.conf` file:

```
Configuration for VirtualHost *:80
...
Redirect / https://localhost/
</VirtualHost>

<VirtualHost *:443>
    ServerName localhost
    SSLEngine on
    SSLCertificateFile "../etc/apache2/ssl/server.crt"
    SSLCertificateKeyFile "../etc/apache2/ssl/server.key"
</VirtualHost>
```

FIGURE 13 SSL CONFIGURATION

The above lines need to be modified so that `ServerName` is the address of the server, `Redirect` is the secure connection to the server, `SSLCertificateFile` is the location of the `.crt` file and `SSLCertificateKeyFile` is the location of the `.key` file (RedirectSSL - Httpd Wiki, 2018). Once the above lines have been added, any traffic can be redirected to port `443` instead of `80` and the website will be secure. This means that if an attacker were to intercept traffic between the client and server then it will be secured and the users credentials, for example, will be encrypted.

2.11.File Upload

Throughout the application there are multiple vectors for a File Upload attack. These include when the user changes their profile picture and any area in the admin section where a file can be uploaded.

For a file upload attack to be successful, the file must first be generated, typically a PHP shell file, and then uploaded to a writable directory. Typically in the case of a PHP shell file, the code is executed when the file is visited or a page with the file in it is loaded. For example, in the admin section a PHP file can be uploaded as one of the item images. This means that whenever a user accesses this page then the code is executed and an attacker has a backdoor into the server.

To avoid this, the developer has already included a filetype filter on the change picture form for a logged in user. This should also be implemented on the admin section as well for any file upload forms.

The filter `Changepicture.php` uses a variable, `$fileuploadtype`, from the file `fileuploadtype.php` which makes the filter only check the extension of the file. The value of the variable should be changed to `"ALL"` instead of `"EXT"` which means that the file will go through every stage of the filter. To add to the filter, the filename could be sanitised to remove control character such as /, <, =, etc and also change the filename to a random value after the file has been uploaded.

The above mitigations allow for prevention of attacks through the filename and also by changing the name of the file. It means that if a malicious file does get through the filter then it will lessen the chance of an attacker finding the uploaded file.

2.12.Cross Site Request Forgery

The application is vulnerable to Cross Site Request Forgery (CSRF) which allows an attacker to inject a malicious link into the page a user is visiting. The vulnerability lies on the Change Password page of the application, which means that an attacker could inject their malicious URL and if clicked it would then allow them to change the password to whatever the attacker likes.

To mitigate CSRF, it is recommended to use a secure token that is sent alongside any forms that are submitted. To do so, a hidden input can be used with a long, cryptographically secure random value attached. By submitting this to the server, it can be stored and then for any other request sent afterwards it can be validated by the server before execution. This allows for a forged request without the token to be rejected so that no harm is done (Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP, 2018).

By using a cryptographcically secure random value it means that there is little chance that an attacker could guess it. To secure this further an expiry time can be set on the token meaning that a new one is generated frequently.

2.13.PHP Information Disclosure

The application allows for a user to view the `phpinfo.php` file by browsing to `192.168.1.10/phpinfo.php`. This file includes a list of all loaded modules, configurations and software versions of everything running on the server. This gives an attacker a basis for launching an attack as they can use the information disclosed in the file to scan for vulnerabilities.

To protect the `phpinfo.php` file, the `php.ini` file can be edited to add `phpinfo` to the `disabled_functions` parameter as shown below ((Enabling and disabling phpinfo() for security reasons, 2018):

```
 ; It receives a comma-delimited list of function names. This directive is
 ; *NOT* affected by whether Safe Mode is turned On or Off.
 ; http://php.net/disable-functions
-disable_functions=
+disable_functions= phpinfo
```

FIGURE 14 DISABLE PHPINFO.PHP

By adding `phpinfo` as shown above, it will redirect any user that is trying to access the file to a page that reads that the function has been disabled. Disabling the function is convenient if the file may be needed at a later date, however if not then it is best to delete the `phpinfo.php` file as this avoids an error message that will let an attacker know that the file still exists and could potentially be retrieved somehow.

2.14.SQL Injection

The application is vulnerable to SQL Injection attacks when executing the `login-exec.php` file. To attempt to protect against this, the application employs an SQL Injection filter in the file `sqlcm.php` as shown below:

```
preg_match("[1=1|2=2|Select|select|2=2|3=3|2 =2]", $username)
```

FIGURE 15 SQL INJECTION FILTER

The above filter will catch the most basic of attacks, for example `' OR 1=1--` however this can be easily bypassed using values like `4=4 and'a'='a'`. The recommended action would be to sanitise the input to remove characters that may be used for SQL Injection, for example "'" and "=".

To further the security it is recommended to use Prepared Statements (PHP: Prepared Statements - Manual, 2018). This will allow for the query and the data to be sent separately meaning that the full query isn't generated dynamically which is where the problem lies with standard SQL queries.

To use prepared statements, the query needs to be restructured:

```
$sql = "SELECT * FROM members WHERE login='$username'";
```

FIGURE 16 OLD SQL STATEMENT

```
$stmt = $mysqli->prepare("SELECT * FROM members WHERE login=?");
$stmt->bind_param("s", $username);
$stmt->execute();
```

FIGURE 17 PREPARED STATEMENT EXAMPLE

Once the queries have been restructured and the filter improved, it will make it more difficult for an attacker to use SQL Injection to attack the application.

2.15.Guessable Directory

The application contains multiple directories that may be sensitive. Two of which are `/company-accounts` and `/databases`. The `/databases` directory contains a backup of the SQL Injection filter, which means that if an attacker were to gain access to the file, then they know exactly what to avoid using to gain access. The `/company-accounts` directory has previously been discussed in Section 2.1.

To avoid an attacker gaining access to any directory that they should not, the same steps as mentioned in Section 2.1 should be followed. To summaries, a `.htaccess` file can be used to stop any directories without a default file (`index.php`) being visited. If a user does attempt to visit then they will be greeted with a `404 Not Found` page.

2.16.Admin Password Attack

The admin login portal can be easily brute-forced due to the simple username and password that it uses, `admin` and `megan`. The issue with the portal being easy to brute force means that after a short time an attacker can gain privileged access to the application as well as members data.

The login can be brute-forced using a tool such as BurpSuite Intruder. Due to the page being an admin portal, then it is easy to assume that it uses a stereotypical admin username, such as `admin`. The password can be brute forced easily due to the short length and low complexity of it with a dictionary attack.

To protect against a brute force attack, a password lockout policy should be implemented, as discussed in Section 2.8, and the username and password changed. It would be recommended to at the minimum change the password for the account. As mentioned previously, a good suggestion for a strong password is a long, four word story that can be easily remembered. The username should ideally be changed as well, to something that isn't similar to admin or would be classed as stereotypical for an admin account. The name of the admin could be used, or something random that again is easily memorable.

2.17.Cross-Site Scripting

An attacker can use Cross-Site Scripting (XSS) to attack members of the application. From a logged in user account, a rating can be left using the Rate Us section of the website. This section will add a rating that any user, logged in or not, can access by browsing to `member-ratings.php`.

The issue with the rating system is that the input isn't validated. This means that a Reflected XSS attack can be used by entering the following into the comment input field:

```
<script>alert(document.cookie)</script>
```

FIGURE 18 XSS ATTACK SCRIPT

The above script when entered will present the user with a JavaScript alert box showing their cookie. This is a severe issue as it shows that the cookie can be accessed by JavaScript, which means an attacker can use other methods to exfiltrate a logged in users cookie. It is also reflected XSS, which is where the command is stored on the sever. This is incredibly dangerous as anyone visiting the page will be affected and there is nothing that a standard user can do stop prevent it.

To assist in mitigating this particular exploit, it was mentioned earlier that the HTTPOnly flag can be set on cookies, meaning that a cookie can only be accessed using the HTTP protocol.

To mitigate the threat of XSS, the X-XSS-Protection header can be set from the server. By implementing the header it allows browsers to filter some types of XSS attacks. To enable the header, the following must be added to the `http.conf` file (X-XSS-Protection, 2018):

```
<IfModule mod_headers.c>
Header set X-XSS-Protection "1; mode=block"
</IfModule>
```

FIGURE 19 SETTING X-XSS-PROTECTION HEADER

After adding the above lines, the header will now be set and most of the XSS attacks will be blocked. This can be seen when accessing the `member-ratings.php` page again with no alert. However, as mentioned the header will only block some types of XSS attacks. A recommendation to try and stop more would be to implement a filter that will sanitise the input from the client. This can include blacklisting known-bad phrases such as `<script>`, `alert()` and control characters such as <, > and `'`. It also means that if a member were to use a browser that doesn't support the X-XSS-Protect header, for example Firefox, then there will still be protection.

With both of the above mitigation used, it makes it incredibly difficult for an attacker to launch a successful XSS attack.

2.18.Shellshock

The backend server for the application is vulnerable to Shellshock, an exploits that leverages a flaw in how older versions of Bash handle a malicious environment variable. The attack requires a vulnerable directory to attack on the server and in this case it is detected as `/cgi-bin/printenv`. The exploit allows for arbitrary commands to be executed on the server (Hunt, 2018).

The only solution to mitigating the Shellshock vulnerability is to update the version of Bash that the server runs. Every version through 4.3 of Bash is vulnerable to the exploit. When the version is updated however the server will be completely protected from the exploit.

2.19.Password storage

When analysing the `register-exec.php` file, it is revealed in the following SQL statement how the passwords for users are stored (highlighted in red):

```
"INSERT INTO members(firstname, lastname, login, passwd, question_id, answer)
VALUES('$fname','$lname','$login','".md5($_POST['password']).''','$question_id
','".md5($_POST['answer'])."')";
```

FIGURE 20 REGISTER-EXEC.PHP SQL STATEMENT

As can be seen in the above figure, the password is hashed using `md5` before being stored in the database. The issue with `md5` is that it is well-known to be insecure and easily reversible.

It is recommended to always store passwords in a format that isn't reversible, for example a password could be hashed using `bcrypt` or `PBKDF2`. The reason that a hashing algorithm such as `bcrypt` is recommended is down to it being a relatively slow hashing algorithm, which helps to slow down brute force attempts.To further on this, it is always recommended to add a salt, a strong cryptographically random value, to the data being hashed (Password Storage Cheat Sheet - OWASP, 2018).

2.20.Service Versions

When an nmap scan was used (see Appendix A), the version of the services that are running on the server were shown. When looking at these, it can be seen that they are all majorly out of date. The issue with out of date software is that whenever an exploit is found it is typically patched in the next update, so the current versions are very vulnerable to a lot of attacks.

The current version of Apache installed (version 2.4.3) currently has 23 CVE's assigned to it, with the first being found in 2013 (Apache Http Server version 2.4.3 : Security vulnerabilities, 2018). The version of OpenSSL (1.0.1c) has 71 CVE's listed with 3 with a score of 10.0, meaning that the vulnerabilities are very effective and dangerous (Openssl Openssl version 1.0.1c : Security vulnerabilities, 2018). Alongside this PHP 5.4.7 has 43 CVE's. This shows how vulnerable the server is to exploitation (PHP PHP version 5.4.7 : Security vulnerabilities, 2018).

The best way to mitigate the risk of exploitation is to update the software. This means that all of the current vulnerabilities typically wont apply.

3. Discussion

The Food Plaza application was previously found to have severe security vulnerabilities throughout. The mitigations that are listed throughout this document should be implemented as a matter of urgency, as if not then the application could be breached by an attacker very quickly and very easily. Implementing the mitigations will also protect the users of the application from having their personal information leaked or stolen.

4. References

Apache Http Server version 2.4.3 : Security vulnerabilities (2018). Available at: https://www.cvedetails.com/vulnerability-list.php?vendor_id=45&product_id=66&version_id=142325&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=3&trc=23&sha=8d43ece6f6ec7b215766fc0be2e707acaeb4f6de (Accessed: 17 December 2018).

Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP (2018). Available at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Primary_Defense_Technique (Accessed: 15 December 2018).

Burp Suite Scanner | PortSwigger (2018). Available at: https://portswigger.net/burp (Accessed: 14 December 2018).

CyberChef (2018). Available at: https://gchq.github.io/CyberChef/ (Accessed: 14 December 2018).

Enabling and disabling phpinfo() for security reasons (2018). Available at: https://www.drupal.org/node/243993 (Accessed: 15 December 2018).

Hunt, T. (2018) *Everything you need to know about the Shellshock Bash bug*, *Troy Hunt*. Available at: https://www.troyhunt.com/everything-you-need-to-know-about2/ (Accessed: 17 December 2018).

Let's Encrypt - Free SSL/TLS Certificates (2018). Available at: https://letsencrypt.org (Accessed: 15 December 2018).

Limited, i. (2018) *Blocking offline browsers and 'bad bots' - Apache .htaccess Guide, Tutorials & Examples*, *Htaccess-guide.com*. Available at: http://www.htaccess-guide.com/blocking-offline-browsers-and-bad-bots/ (Accessed: 13 December 2018).

Limited, i. (2018) *Disable directory listings - Apache .htaccess Guide, Tutorials & Examples*, *Htaccess-guide.com*. Available at: http://www.htaccess-guide.com/disable-directory-listings/ (Accessed: 13 December 2018).

Openssl Openssl version 1.0.1c : Security vulnerabilities (2018). Available at: https://www.cvedetails.com/vulnerability-list.php?vendor_id=217&product_id=383&version_id=141776&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=3&trc=71&sha=2faf0943ab4e6c8f6fdd7b06e6efe1e773b2f909 (Accessed: 17 December 2018).

Password Storage Cheat Sheet - OWASP (2018). Available at: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Do_not_limit_the_character_set_and_set_long_max_lengths_for_credentials (Accessed: 17 December 2018).

PHP PHP version 5.4.7 : Security vulnerabilities (2018). Available at: https://www.cvedetails.com/vulnerability-list.php?vendor_id=74&product_id=128&version_id=142897&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=3&trc=43&sha=183054c6e189e4c737e67d23dd9adca68a514f27 (Accessed: 17 December 2018).

PHP: Prepared Statements - Manual (2018). Available at: http://php.net/manual/en/mysqli.quickstart.prepared-statements.php (Accessed: 16 December 2018).

PHP: Runtime Configuration - Manual (2018). Available at: http://php.net/manual/en/session.configuration.php (Accessed: 14 December 2018).

RedirectSSL - Httpd Wiki (2018). Available at: https://wiki.apache.org/httpd/RedirectSSL (Accessed: 15 December 2018).

Using the robots meta tag (2018). Available at: https://webmasters.googleblog.com/2007/03/using-robots-meta-tag.html (Accessed: 13 December 2018).

X-XSS-Protection (2018). Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection (Accessed: 16 December 2018).

5. Appendices

A. Nmap Scan

```
Starting Nmap 7.70 ( https://nmap.org ) at 2018-12-18 14:20 GMT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is
disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.1.10
Host is up (0.00035s latency).
Not shown: 996 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)
| http-robots.txt: 1 disallowed entry
|_/company-accounts
|_http-server-header: Apache/2.4.3 (Unix) OpenSSL/1.0.1c PHP/5.4.7
|_http-title: Food Plaza:Home
443/tcp  open  ssl/http Apache httpd 2.4.3 ((Unix) OpenSSL/1.0.1c PHP/5.4.7)
|_http-server-header: Apache/2.4.3 (Unix) OpenSSL/1.0.1c PHP/5.4.7
|_http-title: Access forbidden!
| ssl-cert: Subject: commonName=localhost/organizationName=Apache Friends/
stateOrProvinceName=Berlin/countryName=DE
| Not valid before: 2004-10-01T09:10:30
|_Not valid after:  2010-09-30T09:10:30
|_ssl-date: 2018-10-15T10:40:54+00:00; -64d03h39m36s from scanner time.
3306/tcp open  mysql    MySQL (unauthorized)
MAC Address: 00:0C:29:59:83:8B (VMware)
Device type: general purpose
Running: Linux 2.6.X|3.X
OS CPE: cpe:/o:linux:linux_kernel:2.6 cpe:/o:linux:linux_kernel:3
OS details: Linux 2.6.32 - 3.5
Network Distance: 1 hop
Service Info: OS: Unix

Host script results:
|_clock-skew: mean: -64d03h39m36s, deviation: 0s, median: -64d03h39m36s

TRACEROUTE
HOP RTT     ADDRESS
1   0.35 ms 192.168.1.10

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.69 seconds
```