Web Application Security Evaluation

Declan Doyle
1600219

CMP319 Ethical Hacking 2

BSc Ethical Hacking
Year 3

2018/19

Abstract

The world wide web is one of the greatest inventions of the twenty-first century, and has connected millions together, and allowed for commercial opportunities never thought possible before. Almost every webpage now runs as a web application, delivering dynamic content using JavaScript or other web technologies. Often these web applications communicate with a database using a specific language. These web applications bring new sets of security issues, which can be extremely dangerous to organisations and users.

After a security test was conducted on a web application purchased by a client, many vulnerabilities were found, including some of the most common vulnerabilities found in web applications (Information Security, 2018).

Recommendations were made in order to mitigate the vulnerabilities found, and advice was given on strong passwords, and preventing information disclosure.

Table of Contents

1. Introduction

1.1. Background

A web application is a computer program that uses web browsers and web technologies and frameworks to perform tasks over the internet, for example, social media or online shopping. The application may work on desktop clients through browsers or through a client application built with web technologies, such as Electron (GitHub, 2013). The application may also work on mobile devices through browsers or applications developed for said mobile devices.

A web application was bought from a web development company, and the owner of the site requested a security assessment of their newly purchased web application. Following the results of the security assessment, they have requested an evaluation of any vulnerabilities found, with recommendations on mitigations for said vulnerabilities.

1.2. Aim

- To highlight vulnerabilities found in the web application and recommend mitigations in an easy to understand manner for the web development team

2. Vulnerabilities and Countermeasures

2.1. Information Disclosure

2.1.1. Robots.txt

The robots.txt file is a file that is included in web applications that instructs robots what sections of a website to crawl and not to crawl. This is an important file as it can keep sections of a website private to search engines and other spidering tools. (Moz, 2018). However the file should not be used to hide secret or sensitive information completely, as it will not hide information from users, only from automated programs. In the case of the web application tested, the robots.txt file is used to attempt to hide sensitive information, and so discloses said information to an attacker. It is recommended that this information be removed from the web application entirely, as it provides no functionality to the site. If this information must stay on the web application, then it is recommended that it is stored behind an authentication wall, where a username and password must be known before the information can be viewed, rather than be open to be viewed by any user.

2.1.2. Hidden Source Code

On the *my-cart.php* page, there is a comment that reads the following:

```
<!-- *** Lampp folder is /opt/lampp. Loaded Configuration File /
opt/lampp/etc/php.ini -->
```

This error message reveals the location of the LAMP folder and configuration file. LAMP is the architectural model for web services. Leaving comments in the web application that reveal key information about the web application and its setup and configuration are very dangerous as it can reveal information that a user could use to assist an attack. Any user can view a pages source code in a browser, and so the comments left by developers are able to be seen by anyone. It is recommended that the web development team review every page's source code and the comments included, and if, like the comment in the *my-cart.php* page, give away any information about the web application, should be removed.

2.1.3. Reverse Cookies

The web application makes use of cookies, which are generated using a relatively simple obfuscation technique. The cookies contain the username and hashed password of the user logged into the web application, which means that if an attacker gets hold of the cookie, then they could decode the information and log in to the users account. Through a short trial and error period, the encoding method for the cookie was found to be taking the string, putting it through a ROT13 Caesar Cypher, and then encoding it with Base64 encoding. It is recommended that the username and password hash are not stored in the cookie of the web application.

2.1.4. Directory Browsing

The web application has directory browsing enabled, which allows a user to brows the directories of the web application that do not have a default file. This allows a malicious user to browse directories in an attempt to find information that could aid in an attack. This is especially present in the web application in the *CCGRJMJRHVLR* directory, as this contains a file with sensitive information, which a user can find by browsing that directory. It is recommended that the web developers disable directory browsing, so that an attacker cannot browse the directories of the web application

2.1.5. PHP Information

The *phpinfo.php* page that is commonly included on web servers when PHP is installed, as it shows all sorts of information about the web server, PHP version, OS version, etc (Hetzner, 2018). Whilst this information is good for the development process, when a web application is live, this information can be used by an attacker to aid in many attacks, as it gives away far too much information that is unnecessary to any user. It is recommended that this file be removed when the web application is ready to go live.

2.1.6. Hidden Guessable Folder

The web server contains a folder with a name that can easily be brute forced using various web application hacking tools. Folders that are hidden from the web application usually contain sensitive information that an attacker could use to gain information to aid in an attack. In the case of this web application, the hidden folder *cvs* contains the SQL injection countermeasures backup, which, if the attacker retrieves, can aid in an SQL injection attack. It is recommended that folders that need to contain sensitive information be put behind an authentication wall, meaning that a username and password is needed in order for the file to be accessed. This would prevent a regular user accessing the folder, preventing an attacker getting the information inside.

2.2. Command Injection

2.2.1. Local File Inclusion

Local file inclusion is an attack that allows a user to trick a web applications functionality that is supposed to be used to dynamically include local files or scripts, but is instead used to include and view files on the web server, not intended for the web application (Acunetix, 2017). On the *appendage.php* page, the URL of the webpage can be changed so that rather than the page xxx.php being visible, a hidden file can be used, such as the *passwd* file that is used in Linux to store usernames and passwords. In order for an attacker to display a core linux file like the *passwd* file, directory traversal - the act of moving forward or backwards in directories - would have to be used, which the web application attempts to filter out. If the web application detects "../" or ".." in a string then it will remove it and submit the request, however all an attacker would have to do is obfuscate the string entered so that when the filtered strings are removed, the correct syntax remains, for example, "….//". It is recommended that rather than try to filter out this syntax, the web application should deny any requests that contain this syntax. It is also recommended that there should be a whitelist of files that are allowed to be included in the web application, so that only files that the website owner want the user to access can be accessed by the user.

2.2.2. SQL Injection

The login function of the web application is vulnerable to SQL injection. Because of the way the SQL query is designed, if an attacker enters a specific syntax, then they can bypass the need for a username and password to login to the web application. There has been an attempt to detect commonly used syntax in SQL injection commands, however this is easily circumventable, as an attacker could simple change "1=1" to "10=10" or any other logic statement that is true. As this is such a dangerous vulnerability, the web developers should ensure that this vulnerability is patched immediately. It is recommended that the *mysql_real_escape_string()* function is used, as this will strip any characters involved in SQL injection out of any query (Security Planet, 2018). The developers should also sanitise inputs further by removing any characters not necessary to the functionality of the web application, such as the hash key, and the percent symbol, as any user entering these into a username field most likely has malicious intent. It is futile to attempt to disallow every SQL injection statement like the current mitigation, as there are potentially infinite different inputs that could cause harm, and they cannot all be caught.

2.2.3. Cross-Site Scripting

The product search feature of the we application is vulnerable to cross-site scripting attacks. If an attacker enters a <script> tag then the script will be executed, which could lead to the cookie being captured, or any form of JavaScipt code being injected into the web application (Dionach, 2016). The cross-site scripting is also persistent in the adding a comment on a product feature. This means that every time a page is loaded, the injected script will execute. It is recommended that the same mitigations for SQL injection be put in place, where user input is sanitised, so that if a user tries to insert the "<" or ">" characters, they are removed from the input string, as these characters are usually used maliciously. There should also be protection against HTML encoding, as an attacker could enter the string "%3C" or "%3E" which are "<" and ">" in HTML, and the web application may translate the characters, meaning that a script could be executed.

2.3. Authorisation

2.3.1. Cookie Attributes

Cookies in web applications can have several attributes set to them, altering their behaviour. In the web application, there are no attributes set on the cookies used, which increases the vulnerabilities of the cookies. Attributes like the *HTTPonly* attribute prevent client-side scripts accessing the cookie, which prevents cross-site scripting attacks stealing the cookie. The *secure* attribute can also be enabled to only allow the cookie to be transmitted using HTTPS, meaning that the cookie will be encrypted in transit, making it significantly more difficult for an attacker to steal the cookie. It is recommended that to mitigate against cookie theft, the *HTTP*only and *secure* attributes be enabled, and research be conducted into other attributes that the developers may want to enable (Paladion, 2010).

2.3.2. User Enumeration

When entering a username into the web application, an error message is displayed if the username is not in the database of users. However if the username is and the password is incorrect, then a different error message is displayed. This allows an attacker to perform username enumeration, allowing them to attempt to guess a username until a correct one is found. It is recommended that a generic error message is displayed regardless of the conditions of a failed login attempt. This means that a user will not know if the username is contained in the database.

2.3.3. Unlimited Login Attempts

There is no account lockout feature, meaning that a user has unlimited login attempts, even if they repeatedly get their details wrong. This allows an attacker to brute force the password for any account, as they can keep entering passwords with no fear of getting a username locked. It is recommended that an account lockout feature be introduced, so that if a user gets their password wrong a certain amount of times, for example, 10 times, their account would be locked. This could be for a certain amount of time, or until the user gets in contact with the shop owners.

2.3.4. No HTTPS

The web application does not use HTTPS, meaning that all traffic is unencrypted. An attacker could perform a sniffing attack, so that any information a user sent to the web server would be visible by the attacker. It is recommended that the web development team enable HTTPS by getting an SSL or TLS certificate. These are certificates that verify that the traffic is encrypted and allows HTTPS to be enabled (Let's Encrypt, 2018).

2.3.5. Brute Forceable Admin Password

The password for the administrator section of the web application is very weak. The current password was found to be *shell* which was retrieved through a dictionary attack. If the username of the admin account is found, then the admin password can be brute forced easily, and an attack can gain access to the administrator section of the web application. It is recommended that the password for the admin section be made far more secure, as the password is so simple, that it creates almost no protection what so ever. It is also recommend that the admin username is changed, as *admin* is easily guessable, and will most likely be one of the first admin usernames an attacker will try.

2.3.6. Account Recovery

The web application has a very poor account recovery feature, where if the user forgets their password, they can enter their email and contact number, and if they are both correct, then they can reset their password. All an attacker has to do to gain access to a users account is retrieve this information, which can be easily done through social engineering attacks, something that the web application's owners do not have control over. It is recommended that the development team have a rethink on how the user can reset their password, for example, they could send an email to the email address associated with the account, with a link to reset their password. This would ensure that only the user who owned the email account would have the ability to reset the password of their account.

2.3.7.Password Policy
There is no password policy on the web application, which means the passwords that users have will tend to be less secure. Studies have shown that users will take the path of least resistance, and will use less secure passwords given the option to (Computer Weekly, 2018). A good password policy should enforce an 8 character minimum password, as well as mandatory uppercase and numeric characters (NCSC, 2016). It is recommend that the web application follow the NCSC's guidelines for passwords, as they have clear recommendations for creating strong and secure passwords.

2.4. File Injection
2.4.1. File Upload
A user of the web application can change their profile picture and upload their own. An attempt has been made by the developer to only allow specific file types, and to limit file sizes, however there are many ways to circumvent this. There is also only protection when uploading a profile picture, and there is no such protection on the administrator section of the website. It is recommended that protective measures are put in place on the administrator section, as well as updating the protection to make use of some inbuilt functions in PHP. The *getimagesize* function can be used to verify if a file is an image, which will reduce the risk of a user uploading some malicious files (Wordfence, 2018).

2.5. Client Side
2.5.1. Cross Site Request Forgery
The web application is vulnerable to cross site request forgery using the update function, as it does not verify the current user, and so if an attacker can trick a user into visiting a page that the attacker owns, the attacker could get the user to change the password to anything the attacker wants. The best defence against cross site request forgery is to implement a token based authentication system (OWASP, 2018). It is recommended that the web developers implement a system like this.

2.5.2. Client Side Authentication
When the user updated their password on the web application, they are asked to enter it twice in order to make sure they have entered the desired password. This is a handy feature to have that improves the user experience, however, in the case of this web application, the implementation is poor. The authentication is done via JavaScript, meaning that it is done in the client's browser. This is bad practice, as almost all browsers allow for the disabling of JavaScript, meaning that the web application would be forced to skip this authentication. It is recommended that the authentication be changed to server side authentication, through the use of PHP. This will force the authentication to happen, as the user will not be able to disable it.

2.6. General
2.6.1. X-Powered-By Header
The X-powered-by header is an HTTP response header that reveals what kind of server the web application is running on (StackOverflow, 2015). It is recommended that this be disabled as it reveals to any potential attacker the type of web server that the web application runs on. It can also be used to reveal false information, which could throw off an attacker.

2.6.2. Anti-Clickjacking X-Frame-Options
Clickjacking is the name given to an attack where an attacker will hide opaque or transparent layers onto of a button on a web page, tricking a user into clicking the fake layer, rather than the real button. (OWASP, 2017). This can be prevented by enabling the anti-clickjacking HTTP header, however on the web application it is not enabled, meaning that the web application is vulnerable to clickjacking attacks. It is recommended that the developers enable this header so to prevent clickjacking attacks.

2.6.3. The X-XSS-Protection header

The XXS protection header is used to alert a users browser if a cross-site scripting attack is detected, which will prevent the browser from loading the page with the attack (Mozilla, 2018). This was not enabled on the web application so it is recommended that this header is enabled in order to help prevent cross-site scripting attacks.

2.6.4. X-Content-Type-Options header

The content type options header is not set, which allows the browser to render content on a web application different to that off the MIME type. This is a standard that indicates the nature and format of a document file (Mozilla, 2018). This can be a security concern as a non standard type could be executable code. It is recommended that this header is set so that MIME standards are used.

2.6.5. GET Apache mod_negotiation

The GET Apache mod_negotiation is enabled in the web application with MultiViews enabled. This allows the web server to conduct searches in directories, allowing for web application hacking tools to brute force file names (Wisec, 2007). It is recommended that this is disabled so that file names cannot be guessed and brute forced.

2.6.6. Shellshock

The web server is vulnerable to a vulnerability named *Shellshock.* This is a bash bug that allows for remote code execution (Symantec, 2014). This vulnerability is easily mitigated by updating the apache server, so it is recommend that the web developers update the apache server. It is also good practice to keep software up to date in general, as they will receive the latest security updates. The web developers should frequently check for updates and update all software that is outdated.

2.6.7. TRACE HTTP TRACE method

The TRACE HTTP TRACE method allows the client to see what is being received by the server and use that data for testing and diagnostics. This can be used to steal legitimate user credentials (OWASP, 2014). It is recommended that this is disabled so that an attacker cannot see the information being sent to the server.

2.6.8. PHPmyAdmin

The PHPmyAdmin login page is visible to the public and so any user on the web application can navigate to it and attempt to login, or even brute force the login page. If an attacker gains access to this then they have access to the entire SQL database as well as the management of the server software. It is recommended that this page should not be made available to all users, and instead require the administrator account to be logged into before this page is viewable.

3. Discussion

There were many vulnerabilities found within the web application, several of which are some of the most crucial and commonly found vulnerabilities found in web applications. An attacker could easily compromise the web application and so the vulnerabilities should be patched immediately. If the web development team follow the advice and recommendations given then they will greatly reduce the risk of an attack being successful. The company should investigate getting some training for the web development team in developing secure web applications, as to prevent the web application becoming less secure overtime, with the introduction of new features. This will future proof the web application and allow the company to focus more on expansion and less on risk mitigations.

4. References

Information Security. 2018. Top 10 Vulnerabilities in Web Applications. [ONLINE] Available at: https://www.greycampus.com/blog/information-security/owasp-top-vulnerabilities-in-web-applications. [Accessed 15 December 2018].

GitHub. 2013. Electron. [ONLINE] Available at: https://electronjs.org. [Accessed 15 December 2018].

Moz. 2018. Robots.txt file. [ONLINE] Available at: https://moz.com/learn/seo/robotstxt. [Accessed 15 December 2018].

Acunetix. 2017. What is Local File Inclusion?. [ONLINE] Available at: https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/. [Accessed 15 December 2018].

Paladion. 2010. Cookie Attributes and their Importance. [ONLINE] Available at: https://www.paladion.net/blogs/cookie-attributes-and-their-importance. [Accessed 16 December 2018].

Let's Encrypt. 2018. Free SSL/TLS Certificates. [ONLINE] Available at: https://letsencrypt.org. [Accessed 16 December 2018].

Wordfence. 2018. How to Prevent File Upload Vulnerabilities. [ONLINE] Available at: https://www.wordfence.com/learn/how-to-prevent-file-upload-vulnerabilities/. [Accessed 16 December 2018].

Hetzner. 2018. What is PHP info?. [ONLINE] Available at: https://hetzner.co.za/help-centre/website/what-is-phpinfo-and-how-can-i-run-it/. [Accessed 16 December 2018].

Security Planet. 2018. How to Prevent SQL Injection Attacks. [ONLINE] Available at: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html. [Accessed 16 December 2018].

Computer Weekly. 2018. Password practices still poor despite increased threats. [ONLINE] Available at: https://www.computerweekly.com/news/252440316/Password-practices-still-poor-despite-increased-threats. [Accessed 16 December 2018].

NCSC. 2016. Password Guidance. [ONLINE] Available at: https://www.ncsc.gov.uk/guidance/password-guidance-simplifying-your-approach. [Accessed 16 December 2018].

Dionach. 2016. The Real Impact of Cross-Site Scripting. [ONLINE] Available at: https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting. [Accessed 17 December 2018].

StackOverflow. 2015. What does "x-powered by" mean?. [ONLINE] Available at: https://stackoverflow.com/questions/33580671/what-does-x-powered-by-mean. [Accessed 17 December 2018].

OWASP. 2017. Clickjacking. [ONLINE] Available at: https://www.owasp.org/index.php/Clickjacking. [Accessed 17 December 2018].

Mozilla. 2018. X-XSS-Protection. [ONLINE] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection. [Accessed 17 December 2018].

Mozilla. 2018. MIME Types. [ONLINE] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. [Accessed 17 December 2018].

Wisec. 2007. Security Thoughts. [ONLINE] Available at: http://www.wisec.it/sectou.php?id=4698ebdc59d15. [Accessed 17 December 2018].

Symantec. 2014. Shellshock. [ONLINE] Available at: https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability. [Accessed 17 December 2018].

OWASP. 2014. Cross Site Tracing. [ONLINE] Available at: https://www.owasp.org/index.php/Cross_Site_Tracing. [Accessed 17 December 2018].