Web Application Testing

Jack Gates

Student no: 1500763

CMP319: Ethical Hacking 2

BSc Ethical Hacking Year 3

2017

[DATE]

UNIVERSITY OF ABERTAY DUNDEE

[Company address]

Abstract

This report details the vulnerabilities found in the 'AA200' application. This report will highlight where these vulnerabilities were found in the application, it will discuss how they were found, and the impact that these vulnerabilities may have on the application will also be mentioned. Lastly, this report will advise the reader on how they may mitigate these vulnerabilities using certain techniques.

There were numerous vulnerabilities found using 'The Web Application hacker's handbook'[0] methodology during this investigation that a malicious user could use to attack the website. These vulnerabilities were scattered around the application and varied in severity. All of the identified vulnerabilities were evaluated and a suitable countermeasure was found for each. Applying all of these countermeasures will secure the clients application to a sufficient standard.

Contents

Introduction

The owner of the application ‘AA200’ was concerned that there may be some bugs that could be used by a malicious person to hack into their web application. The site was tested following the methodology of the hacker’s handbook, which was chosen as it has a logical and thorough approach to testing. Throughout this investigation, many vulnerabilities were found that range pretty high in severity. These vulnerabilities were then analysed and a suitable mitigation for each issue will be mentioned for the development team to prevent or limit the damage that could be done through exploiting these vulnerabilities. These vulnerabilities can be very destructive to an application and it should be recommended that they are mitigated.

Vulnerabilities Discovered and countermeasures

Hidden content:

Discovering the hidden content of the application uncovered many hidden files, one of which was the 'robot.txt'[1] file. The file was meant to be preventing web crawling robots to access the 'Disallowed' locations, However, this file also tells hackers the places that you do not want them to look. This file held the location of a zip file containing important spreadsheet information that the hacker was easily able to find. The contents of the file can be seen in figure 1.

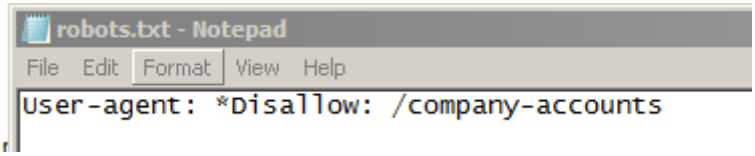

Figure 1 - robots

To prevent this, the 'robot.txt file should not be used to hide sensitive information.

The file called 'phpinfo.php' was also uncovered using a tool called 'dirbuster'[2]. The file gives away too much information about the web application that a hacker can use to their advantage. It therefore should not be accessible from the user. To prevent this from being used against you it can either be removed or secured. Sensitive files, such as 'phpinfo.php' can be secured by creating an 'htaccess'[3] file and adding your own IP address as the only one allowed. This can be done using the code in figure 2 below, the numbers represent the web developers IP address.

```
<files phpinfo.php>
        order Deny,Allow
        Deny from all
        Allow from 123.456.789
</files>
```

Figure 2 - htaccess

A directory called 'META-INF' was found and contained a file called 'sqlcm'. This file contained information that was found to be useful for hackers. This folder was uncovered during the mapping stage using the 'dirb'[4] tool, which revealed several of the directories on the application. This directory has quite a guessable name, so it could potentially be uncovered through a malicious user searching for directories. To prevent information like this from being leaked it is a good idea to not store them under directories with enticing names like META-INF, short for meta-data information. The information found will aid a hacker when they are attempting their SQL injection.

Directory browsing vulnerability

The user is able to browse to any directory on the application. This causes problems as it means the user only has to find out the names of the directories to browse to it, and this can be done through simple footprinting techniques. For example, during the mapping stage the admin directories were

```
Options -Indexes
```

Figure 3 - htaccess code

discovered, and now the malicious user can browse to these pages that they should not have access to. To prevent this, in your htaccess file you should add the following code in figure 3.

The htaccess file controls everything that the server does with it. This will prevent directory browsing as every time a user tries to browse to a directory, they are transferred to the index page in that directory instead, in the case of user/admin directories they should be directed to an authentication page. This is because '-indexes' supresses the possibility of directly listing files in a directory where a directory index is not set.

Comments:

All of the comments can be retrieved using an Nmap script[5] against the application seen in figure 4.

```
root@kali:~# nmap 192.168.1.10 -p 21,80,443,3306 --script=http-comments-displayer
```

Figure 4 - Nmap script

Comments can also be seen by just viewing the source code. Comments are targeted as they can sometimes contain useful information. A comment had been left in the source code of the 'user_products.php' page, its contents can be seen in figure &. The comment highlights the fact that there was a file that should be deleted in the final version, the attacker can assume that the reason is because this file is not supposed to be seen, and possible contains useful information. It turns out the file was not deleted, and this comment, which can be seen in figure 5, ended up putting a target on the file.

```
<!-- *** Remember that phpinfo.php should be deleted in the real version -->
    <!DOCTYPE html>
```

Figure 5 - comment

It is unwise to put information you wouldn't want a hacker to see in the comments of a page's source code. To prevent this, comments including sensitive information should be removed.

Local file inclusion:

Local file inclusion was discovered in the 'affix.php' page. This file itself was found through spidering the application. Its parameters were tested using the fuzzing technique on 'OWASP ZAP'[6] tool to find all of the different files that it could dump out and it was discovered that it could traverse

```
<?php
$pagetype = str_replace( array( "../", "..\"" ), "", $pagetype);
?>
```

Figure 6 - poor filter

directories and access other files, such as the '/etc/passwd' file, and dump the contents on the screen. A user being able to traverse the directories and output local files has access to all of the sensitive information on these files and so this should be prevented. This attack can be mitigated through the filtering of user input. An effective remedy is to create a whitelist of all of the files that can be included in the page. The current filter in figure 6 can be worked around and is not effective at preventing Local file inclusion attacks

A more secure filter using a whitelist of accepted parameters can be seen in figure 7. This will compare the type parameter with the whitelist array of accepted types. The parameter is removed if it is not the same.

```php
<?php
$whitelist = array('faqs.php', 'terms.php');
$input = htmlspecialchars($_GET["type"]);

if (!in_array($input, $whitelist)) {
      $out['error'][] = "Invalid file";
    $pagetype = str_replace( $input, "", $pagetype);
    }

?>
```

Figure 7 - good filter

Cookies

The cookie manager plugin was used to view all of the cookies stored on a page. After analysing the cookies, it was found that a cookie known as the 'secretcookie', most likely held user information and was able to be reverse engineered as there was a pattern found using web scarab against it, the pattern can be seen in figure 8. Fifty samples of the cookie on the same user were fetched and analysed.

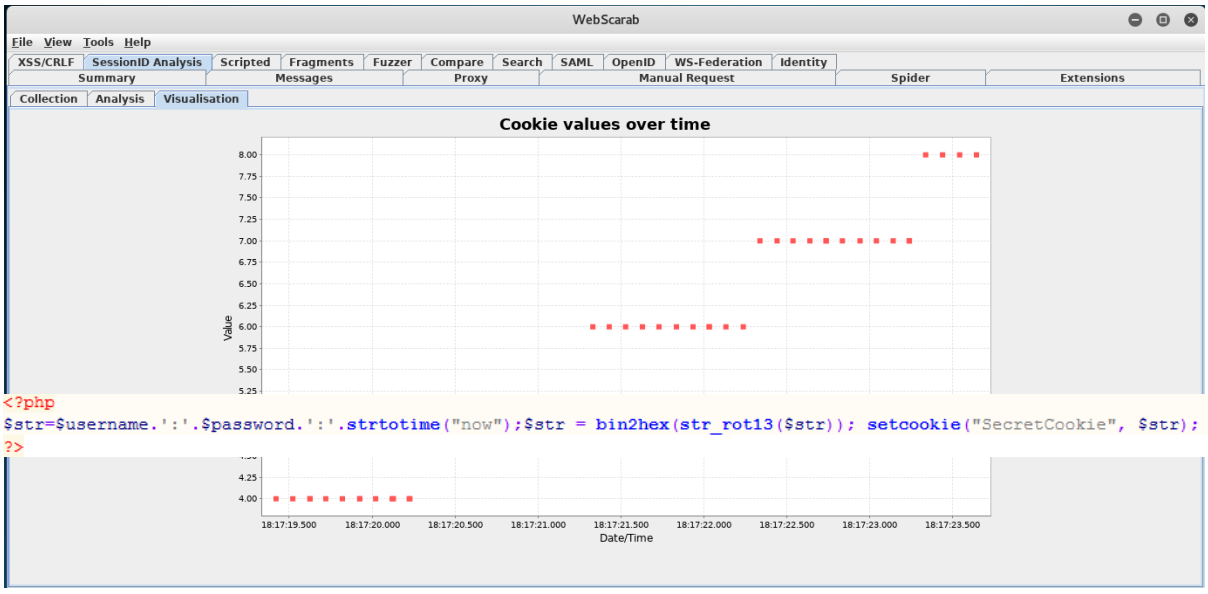

Figure 8 - cookie pattern

The cookie incremented by 1 every second on the same user. Different users yielded entirely different secret cookie values. With this information, an attacker can assume that user details are stored in the cookie along with a time stamp. Common coding methods were tested on the cookie, such as hexadecimal. It was found that by converting the cookie from hex to asci that it looked like it was split into 3 sections separated by colons. The only section that changed was the last section, meaning that it was a timestamp, and the other two sections were user details. Through testing different common decoding methods, it was found that the first section was the customers email, and the second was the customer's password. This means that if a hacker was able to steal this cookie and reverse engineer it, they will have a user's log in credentials. The current cookie encoding can be seen in figure 9.

To keep this cookie secure against reverse engineering it should be encrypted, instead of encoded.

Figure 9 - encoding method

Encryption will secure it against reverse engineering as it doesn't just use an algorithm like encoding, but it also makes use of keys that are only possessed by the client and the server, making it extremely difficult for intercepting third parties to decipher.

Another issue discovered with the cookies is that there have been no attributes set on the secret cookie (httpOnly[7]). The secret cookie was set up using this code in figure 10.

```
setcookie("SecretCookie", $str);
```

Figure 10 - set cookie

this only applies a name to a cookie and the value. Setting up other variables, such as the 'httponly' variable will benefit the security of this cookie, as it means that the cookie can't be accessed by a client side script such as JavaScript, which means XSS attacks can't steal/change the cookie value.

User enumeration vulnerability

The 'forgotpass.php' page allows a user to submit the email they have used to register their account in order to reset their password. The new password is then sent to the users' email. This can be abused as a user is able to guess usernames and if the user exists it will display a message stating "Your New Password is Send to your Email". If it fails, the message "no user exist with this email address" Is seen. This can easily be automated using brute forcing meaning that many user names can be discovered and their password reset to a simple 3 letter password, the code for this can be seen in figure 11.

```
if(mysql_num_rows($result)){
$code=rand(100,999);
$new_pass = md5($code);
$emails=$_POST['email'];
```

Figure 11 - new password

This sets up their account perfectly for a dictionary attack, using a list for the discovered usernames, and a list of number 100-999. The hacker now has all of the enumerated customers log-in details.

To prevent this attack, the web site owner can set up security questions on the site. When a user wants to reset their password they have to type in their security questions. This will be effective because it means that the users can no longer be automatically enumerated, and it also adds another layer of security. Another way to prevent this vulnerability is to make the reset password stronger, meaning that if the attacker is able to guess a username/security questions correctly, they will not be able to run a dictionary or brute force the authentication mechanism easily.

Password Cracking

Another problem that the application has with its authentication mechanism is that there is no limit to the amount of login attempts. Giving an attacker unlimited log in attempts means that they can automate attacks against it. After a given period of time the attacker could then acquire log-in details as seen in figure 12 where the tool hydra[8] is used to crack the password using a common password dictionary. This is made even easier with the ability to reset passwords in user enumeration vulnerabilities.

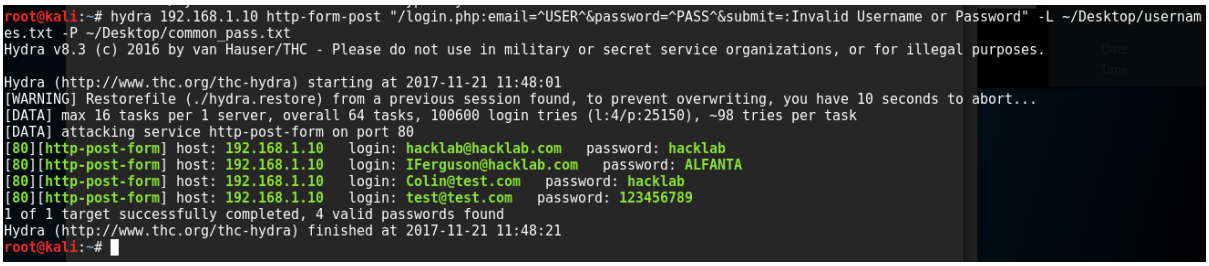

Figure 12 - dictionary attack

To prevent this, it is a good idea to implement an account lock-out mechanism. To do this, every time a user enters their username with an incorrect password, a value should be incremented and stored in the database. Every time a user logs in, it should then check that the number of failed attempts is less than the threshold of failed attempts (less than 10 is recommended) in the database. If the number exceeds the threshold, then the account is locked for a period of time after every failed attempt. The number of failed attempts stored in the database for a user will be set to 0 for every successful log in. This will secure the users against any authentication based attacks.

The admins authentication details were also tested using dictionary attacks, and were revealed to be very weak. As there is also no authentication lockout[9] mechanism, this makes it vulnerable to dictionary attacks. The authentication details are weak as they are commonly used and not atypical. For example, one of the admin's usernames is 'admin'. The passwords ('123456', 'hacklab', 'eiderdown') also have a high potential for being found in common password dictionary's. This means that dictionary attacks against the admins account are likely to yield the admins log in details. Most of the log in details are also quite short, meaning that brute forcing them will only take a matter of time.

This can be prevented through the enforcement of stronger passwords. Passwords for admins should be especially strong as they have control of other accounts and sensitive application data, so a minimum length of 14 characters is recommended. This is an adequate length as it is too long to brute force, yet is not too long to remember. It is also recommended that three different character types could be used, such as a capital, lowercase, and number character. A simple password 'Admin123456789', which abides by these principles was tested on the site 'https://howsecureismypassword.net/'[10] which revealed the time taken to brute force in figure 13.

Figure 13 - password security testing website

File upload vulnerability

From the users 'updatepassword' page, file extensions are successfully filtered out. However, this is not the case for the admins page. From the admins page, there is no attempt to filter out any file types and as a result the admin can upload any file type to the server. This is a potential hazard if the admins account is insecure, as a malicious user can gain access to their account and upload a backdoor or possibly other types of malicious files to the website. The users page checked for the extension of the file and compared it with a whitelist of accepted files, to filter out bad file types, the code can be seen below in figure 14 and 15.

```
$file_ext=strtolower(end(explode('.',$_FILES['uploadedfile']['name'])));
```

Figure 14 - check for extension 1

```
if ($fileuploadtype=="EXT"|| $fileuploadtype=="ALL"){
$extensions= array("jpeg","jpg","png");
if(in_array($file_ext,$extensions)=== false){
    echo '<script type="text/javascript">alert("extension not allowed, please choose a JPEG or PNG file.");</script>';
    echo "<script>document.location='$nextpage'</script>";
    exit();
}
}
```

Figure 15 - check for extension 2

The code in figure & separates the extensions from the file type by using the 'explode'[11] function. This function will split the file every time a period is used in the string and store the contents in an array. The 'end' function means that only the last value in the array is stored as the variable, then the file extension string is converted to lower case. In figure & the file extension string is then compared with the whitelist array of accepted file types, if it does not match any of the accepted file types, the file does not get uploaded. This makes it impossible to upload any harmful file types.

Request Forgery

Cross site

The 'passwordupdate.php' which allows a user to update account information like log in details, is vulnerable to CSRF. This means that if a hacker can trick a user into visiting a web page, they could trick the user into changing this information without their knowledge. An attacker could use this to gain access to user accounts. The best way to get a user to click on links is through using the comment section function to socially engineer them, as it allows for the posting of hyperlinks.

To prevent CSRF attacks, a hidden form field can be included in the form to authenticate the request. This form field can be used to hold data an attacker would not know, such as the session id cookie of the user. If a secret such as this is embeded in the page, and the server only allows the request if this secret is included along with the request, then the form will be secure against CSRF.

On site

Local request forgery was also present in the website. It was also found that the contents of the POST request, when adding a product to the basket, could be intercepted and the contents of the parameters changed. This form or request forgery can be used to change the prices of a product before purchasing it. Figure 16 shows the request and figure 17 shows that it was used to buy products for £1 each.

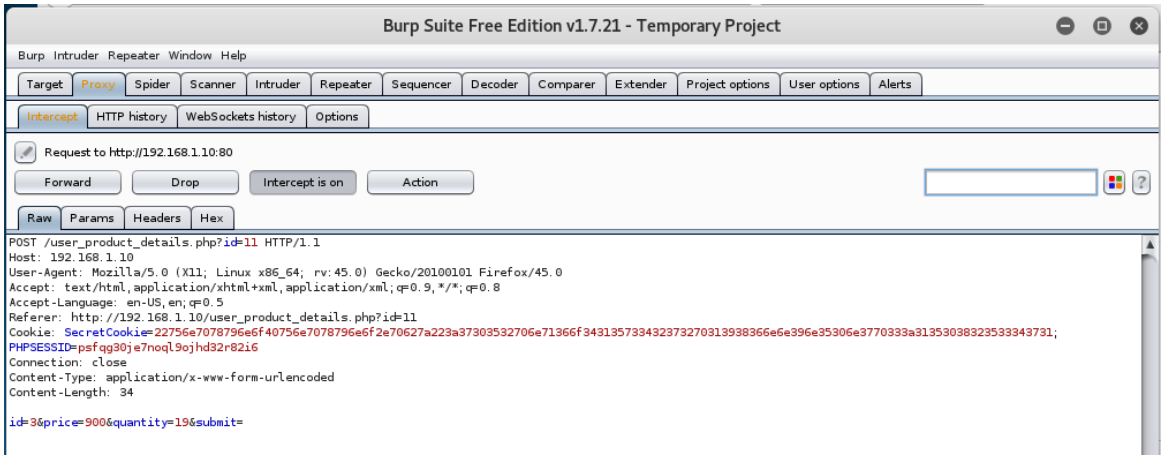

Figure 16 - intercepted request

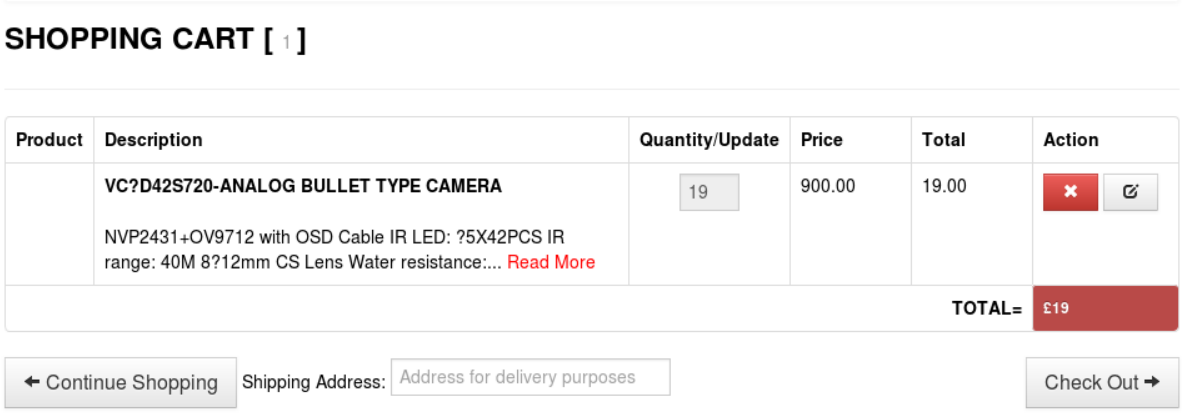

Figure 17 - shopping cart

A way to prevent this vulnerability is to not use the values from the request on the user's page when checking out, but to instead use the values that are stored in the database, as these values cn not be manipulated on the client's side.

Cross Site Scripting

Cross site scripting was discovered in the comment section when testing for input based vulnerabilities. The user registration page/update password page where a user enters their information was also found to be vulnerable against this form of input based attacks. Through the use of Cross site scripting in this application, a malicious user can store scripts that will be executed on another user's machine. An attacker can use this to perform a range of attacks, an example of one would be to steal the cookies on the page.

A simple function that will prevent xss attacks is the 'htmlspecialchars()'[12] function. This function will convert some predefined characters (ones often used for xss attacks) to HTML entities. The string that the user enters will be placed into the parameters, and any special characters they enter will not be executed as code, but will instead be turned into html.

SQL Injection Vulnerability

It was found that the index.php page was vulnerable to SQL injections. An attempt to filter out certain words was made, such as '1=1', but this was not a very good filter and was able to be worked around through adding characters like spaces to the input.

Through the inspection of the code, it was found that a 'clean' function was in use which sanitised the user input before it was passed into the function which communicated with the database. This function was only used on the password input form and not on the username input form, leaving it open to SQL injection attacks. The 'clean' function itself will effectively prevent SQL injection attacks as the function first uses the 'trim' function on the data. This removes data which could be used to aid in SQL injections such as spaces. Figure 18 shows the code that was used for index.php.

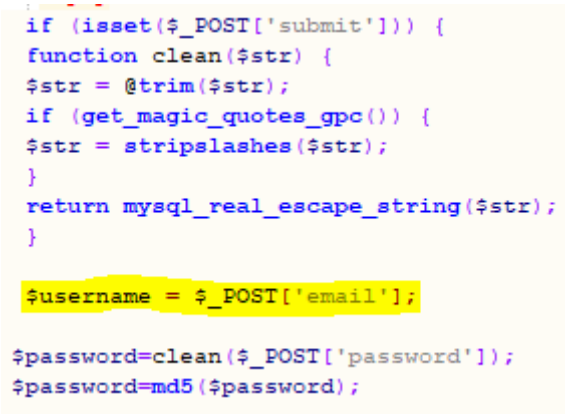

Figure 18 - cleaning input

Another effective way to prevent SQL injection attacks is to make use of prepared statements[13] and bound parameters. This is a useful technique as it separate SQL queries from user entered data and the bind parameter function only binds data to placeholders, leaving the SQL query unchanged.

HTTP

There is no use of https on the application. This means that sensitive user information that is entered can be stolen by attackers that are packet sniffing. To prevent this, https should be used to handle the POST requests that handle user input, such as passwords or usernames. It can also be used to handle requests like GET, that way data such as the secret cookie cannot be grabbed or tampered with either. To change from Http to HTTPS an SSL certificate[14] is needed, and should be installed on the websites hosting account.

Miscellaneous vulnerabilities

A tool called 'Nikto'[15] was used to scan the web application for common vulnerabilities and several issues were found. The contents of this Nikto scan can be seen in Appendix A.

Shellshock

One of the found issues in the website is the 'shellshock'[16] vulnerability, which is a vulnerability found in outdated version of apache. New ways to exploit software is being found all the time, and when these exploits are found, the software developers find fixes to these software vulnerabilities and release them in the form of a patch. These patches are only installed when the software is updated, so outdated software is usually quite vulnerable to attacks. A software flaw was found in the web servers apache environment which allowed the user to gain access. This vulnerability, known as 'shellshock', happens as there is a fault in how the Bash shell handles external environmental variables. This specific vulnerability was pretty severe and allowed the Kali Linux machine access to the secure shell.

Phpmyadmin

Another issue found was that the Phpmyadmin management package was visible to anyone who could access the web application. This means that a malicious user could attempt to guess log-in credentials or brute force this. This can be prevented by changing the path of this page through the admins directories, meaning low level users can't view it without admin access.

Another possible issue with the web application is that the server makes use of 'mulitviews'. This essentially makes file names easier to enumerate as when a file is looked for, the closest match is used instead. Deactivating this will make the web application harder for a malicious user to map.

Trace

The TRACE method is considered a vulnerability as it can be used with XSS to find sensitive information. if a cookie has been tagged as HttpOnly, then it can't be touched using javascript, meaning that XSS attacks are less likely to yield sensitive information. The TRACE method being active circumvents this by bypassing the protection HttpOnly provides. For this reason, the TRACE method should be disabled.

Headers

The Nikto scan also revealed that the 'X-XSS-protection'[17] header was not defined. This header is used to enable the XSS filter built into most modern browsers. The fact that it is not defined means that when a user does encounter XSS on the application, there is no protection against it. To enable this header for the prevention of XSS, it should be set to the value x-xss-protection:1, mode=block. This will enable XSS protection and block malicious scripts.

The header 'X-Content-Type-Options'[18] was also not defined. This header can be used to prevent the browser from sniffing (MIME types) content. To prevent this, the header should be set to 'nosniff'. This will block requests which are of type 'style' and the MIME type is not text/css. It will also block requests which are of type 'script' and the MIME type is not javascript.

The x-powered-by header[19] can be used to leak information about the server-side technology. From this header an attacker can enumerate the version of PHP being run (5.4.7). To prevent information being leaked from this header it can be used to falsify information to throw off hackers.

References

Stuttard, D. and Pinto, M. (2013). *The web application hacker's handbook*. Hoboken, N.J.: Wiley.

Robotstxt.org. (2017). *The Web Robots Pages*. [online] Available at: http://www.robotstxt.org/ [Accessed 1 Dec. 2017].

Nmap.org. (2017). *Chapter 9. Nmap Scripting Engine | Nmap Network Scanning*. [online] Available at: https://nmap.org/book/nse.html [Accessed 1 Dec. 2017].

Owasp.org. (2017). *Category:OWASP DirBuster Project - OWASP*. [online] Available at: https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project [Accessed 1 Dec. 2017].

Limited, i. (2017). *What is .htaccess? - Apache .htaccess Guide, Tutorials & Examples*. [online] Htaccess-guide.com. Available at: http://www.htaccess-guide.com/ [Accessed 1 Dec. 2017].

Tools.kali.org. (2017). *DIRB | Penetration Testing Tools*. [online] Available at: https://tools.kali.org/web-applications/dirb [Accessed 1 Dec. 2017].

Owasp.org. (2017). *OWASP Zed Attack Proxy Project - OWASP*. [online] Available at: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project [Accessed 3 Dec. 2017].

Addons.mozilla.org. (2017). *Cookies Manager+ – Add-ons for Firefox*. [online] Available at: https://addons.mozilla.org/en-GB/firefox/addon/cookies-manager-plus/ [Accessed 3 Dec. 2017].

Portswigger.net. (2017). *Burp Suite Scanner | PortSwigger*. [online] Available at: https://portswigger.net/burp [Accessed 3 Dec. 2017].

Tools.kali.org. (2017). *THC-Hydra | Penetration Testing Tools*. [online] Available at: https://tools.kali.org/password-attacks/hydra [Accessed 3 Dec. 2017].

Owasp.org. (2017). *Testing for Weak lock out mechanism (OTG-AUTHN-003) - OWASP*. [online] Available at: https://www.owasp.org/index.php/Testing_for_Weak_lock_out_mechanism_(OTG-AUTHN-003) [Accessed 3 Dec. 2017].

Collider, S. (2017). *How Secure Is My Password?*. [online] Howsecureismypassword.net. Available at: https://howsecureismypassword.net/ [Accessed 7 Dec. 2017].

Php.net. (2017). *PHP: explode - Manual*. [online] Available at: http://php.net/manual/en/function.explode.php [Accessed 7 Dec. 2017].

Php.net. (2017). *PHP: htmlspecialchars - Manual*. [online] Available at: http://php.net/manual/en/function.htmlspecialchars.php [Accessed 7 Dec. 2017].

W3schools.com. (2017). *PHP Prepared Statements*. [online] Available at: https://www.w3schools.com/PhP/php_mysql_prepared_statements.asp [Accessed 7 Dec. 2017].

Trends, S., Https, W. and Mansfield, M. (2017). *What You Need to Know About Changing From Http to Https*. [online] Small Business Trends. Available at: https://smallbiztrends.com/2015/04/changing-from-http-to-https.html [Accessed 12 Dec. 2017].

Sectools.org. (2017). *Nikto – SecTools Top Network Security Tools*. [online] Available at: http://sectools.org/tool/nikto/ [Accessed 12 Dec. 2017].

Symantec Security Response. (2017). *ShellShock: All you need to know about the Bash Bug vulnerability*. [online] Available at: https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability [Accessed 12 Dec. 2017].

Owasp.org. (2017). *HttpOnly - OWASP*. [online] Available at: https://www.owasp.org/index.php/HttpOnly [Accessed 12 Dec. 2017].

KeyCDN Blog. (2017). *X-XSS-Protection - Preventing Cross-Site Scripting Attacks*. [online] Available at: https://www.keycdn.com/blog/x-xss-protection/ [Accessed 12 Dec. 2017].

Mozilla Developer Network. (2017). *X-Content-Type-Options*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options [Accessed 12 Dec. 2017].

Scott Helme. (2017). *Hardening your HTTP response headers*. [online] Available at: https://scotthelme.co.uk/hardening-your-http-response-headers/ [Accessed 12 Dec. 2017].

Appendix A – Nikto Scan

- Nikto v2.1.6

+ Target IP: 192.168.1.10

+ Target Hostname: 192.168.1.10

+ Target Port: 80

+ Start Time: 2017-11-09 10:20:09 (GMT-5)

+ Server: Apache/2.4.3 (Unix) OpenSSL/1.0.1c PHP/5.4.7

+ Retrieved x-powered-by header: PHP/5.4.7

+ The anti-clickjacking X-Frame-Options header is not present.

+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS

+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type

+ Server leaks inodes via ETags, header found with file /robots.txt, fields: 0x2a 0x55b97ca7e08c0

+ OSVDB-3268: /company-accounts/: Directory indexing found.

+ Entry '/company-accounts/' in robots.txt returned a non-forbidden or redirect HTTP code (200)

+ "robots.txt" contains 1 entry which should be manually viewed.

+ PHP/5.4.7 appears to be outdated (current is at least 5.6.9). PHP 5.5.25 and 5.4.41 are also current.

+ OpenSSL/1.0.1c appears to be outdated (current is at least 1.0.1j). OpenSSL 1.0.0o and 0.9.8zc are also current.

+ Apache/2.4.3 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and 2.2.29 are also current.

+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives for 'index' were found: HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var

+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.

+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST

+ OSVDB-112004: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).

+ OSVDB-112004: /cgi-bin/printenv: Site appears vulnerable to the 'shellshock' vulnerability (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6278).

+ /phpinfo.php?VARIABLE=<script>alert('Vulnerable')</script>: Output from the phpinfo() function was found.

+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.

+ OSVDB-12184: /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.

+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.

+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.

+ OSVDB-3092: /admin/: This might be interesting...

+ OSVDB-3268: /img/: Directory indexing found.

+ OSVDB-3092: /img/: This might be interesting...

+ OSVDB-3093: /admin/index.php: This might be interesting... has been seen in web logs from an unknown scanner.

+ OSVDB-3268: /database/: Directory indexing found.

+ OSVDB-3093: /database/: Databases? Really??

+ OSVDB-3233: /cgi-bin/printenv: Apache 2.0 default script is executable and gives server environment variables. All default scripts should be removed. It may also allow XSS types of attacks. http://www.securityfocus.com/bid/4431.

+ OSVDB-3233: /cgi-bin/test-cgi: Apache 2.0 default script is executable and reveals system information. All default scripts should be removed.

+ /phpinfo.php: Output from the phpinfo() function was found.

+ OSVDB-3233: /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a lot of system information.

+ OSVDB-3268: /icons/: Directory indexing found.

+ /phpinfo.php?GLOBALS[test]=<script>alert(document.cookie);</script>: Output from the phpinfo() function was found.

+
/phpinfo.php?cx[]=3ZXWUDxpwcnUIOPGTs6zl4oM8hirUM5o2aD3d4nBKBks7ByOoOj2a8yBxJdz4ndBk57FoIHnfsY0R2HUvAcsYc8z3jUrbfzX5xvQB6hLmtj8eviBXTIqTyHyPJAaCp7VKSDTTX1J2J565g6fm8ggL7pX9YSEEuoRAb0CkrqOGFjQOccH7pUP5U76t5XbHl7CHcRL9W7s3Dwnx73XoMYUSjKT6KzTpkpxhWUwfPES2BgiCgFt4BsHpqjWw4zgX1o9gT55EKqPyllAuqesUxQS0paZvlvw8SB8g52rcQNGaIexA6UFJTJQqQ

RYS1A7MaHbUCB4mBkzUwj6Lt2kCwWrrXcc0xG0MPU1laXSdDmQLpk6KhxwmWkT9c0f1rS9BLSjnSVg
xTvqxoxa9wKDNd5ddmZYHnWHnVeKzeO354AC8k8mpaphrUPTX6na7hfsoPw20onSXjvPn6psjw6oVO
6YxdGvSyO0fTP47luh95xFfL9azMPIPSJmE47UfpH3nhTX4GNdfoAZtiLLP72qmNsGBRblP7sBiqOiia4vicI
hSrGMHWPabJDjN9461v5e49bfwQCpE9Q5M2n6sSDkoS3Pv4Zd7lU8Fu0UXZngfqGGduP2xDUBv4MN
hSoLh10nu0TpXD3oL5ANkPwULNuNw9jVqaj2WYzWET1pYTVL6k3GuFOBnyQ27wDudpjyfwQSEAwdO
PIbRpR2yZZAR468TpyjaeEePvVeH5wnad5c7mNxYMyOt5PhKeYH0U97IccgmOzTKJjmbsjYUjggy1H3bO
weOFfmx0BiVeRRjcsoHqA6gvqmHlSIWB0rFbCPIDvN9VGHs3IIEqodIQTQypdHtMf8GV9SdTwdEuPVr0t
GJMBcGkkCwzoyGmQVqCnJ9VO2BT6vxWdmHnwrg1bc6RzWC88cEAjaScqzhBNywRnZ23xe6IYUaL9b
kKffAiMH0S0UKanSIj1SxnR0d2bLLG8QIsh6tC2YZWEDrEYHayjbgCG3PHvb1vmQE4yfAqXSwC6OTuffjr2
RsZ547nMgtMvySH49BSzOsK3CmaYCN2IIMaz8UwcWp70yksUInZ3rLkRaBeVmUMUAw0rJnNbJRZxhe
7Zu6U5TMTiftnFuVu1u5yATBCAkr0AgwPFPWzNovkI1TGu4jZj3tmcBggyMOtywcrJJTWPapkkFjQgiIrtU
cTXk3UIxckUeNdbNfCgU7kP6NOydMkndTq8AjrZ6gxSHZ7wGdbuAwa6tBbyQRdlizT2VlcLueLeHeflNdL
WFZ3qU69gVr8zqAltMWgEihzzTp5GrjnTJRnAZCax9BGfv3nGLGWJGcKFNNQBabwZz2Twij3q10KDcdU
XOiIdNlkrPNAvFqvVzVB0WU2e258TK7e3RUfBjOJPJ2mQMn6OGr3G5S20LHwVdglaik1vdWrkfmjj82RG
Aooig3ZJYWwTGwqASc1CL0UC8XIELPHLsNOa1MtPQv6Rx8G9AbXxNj5znCPKxgiQ0MFT6N7wa6VWR
WlaT4yM7oZuM5YyprKi5j4j2dArzx8K3B5gQPsxfC2MJg9sxBOospeGfvj3SPbdg8qDEC9GbXmIqW3tjlF9
hn68EzAnlRWMipFfaQtuG9TO84CO4YduHtQhEHFy5aT0c4UuMYaBuYr52ENhRuKX9pEcHJ5BpjFkndo
EbA4LXWz42d65QNgFLyjtgZ07hlABoB1VA3eKN9mKaOctPqeXrPbHhyO9LNBreIikOvM29w300KlJeyRa
agSY79HYyLctkagXOvGNI8GlBFT6nuP7ToqSXzw6sFlz45ZkaOdtl3nQwrAYduCIB9YsGwG7ZXHEVP0YO0
mLybIqhsW1b4fl1IaCd3cNYuwHbBi3UeTcBCSzZHsEJzCkCQ2HoAO4bY8llwhpVGv2s3I3q6QkQfb516lia
eqe0W6xTxhvIfoUiIFh6S7ifGpwOyKcwVoYbVJT5HYL3T4CTfBE01L5CBo2TweSwNbLs3t4LnqXYiDOzap
xgMEtw7lK2DoMehZVvQqQoV2NC79J9mLcmk5fhkch0Y3kKBE6zB8g776LUwxRb4iirqEmDh4Y1AAGW
VtuuJuoS4c0ElFDnSex5e7UjwZsF0KdyLsprSTbL1huFyR5oehLa49ykGiGac8LQhmAXSw1ivGk8nuAB116
dA9IlRip8u3yBL7QMAVfXtweZCgnlCAJQtLby5NZDWdZNRhvZtCpIFnbFHticYKFFSGBPoKRLHIWs6FaP4
7DI68fOULjzdq4hl2gf4K3AHO8lhWe0nQwVM9Clo5ivhuCNiZNVAFyuy5SLVDJzMCebiFKodcpG52spd5
elEdKSCmR7IZ2dfU5VXqOQm5UHCVcqcKUKNzusdVrK4cHgUGutGC5uIbvtGTqpmuMbjUyIhETmvuh5
Q6aUp1PRRsbYAIgTCewFG7s1Sn8RkDn6pIOzdp0P8bt8nbzX7r0UcfKRe3vK0mtgBx6f3exFcIfU223xhnL
7ehNSKjXNU03c9C0FczjY6S4LjB4Gn5DqACORtPhvIfCEsxCZddR4IkR5qznkGtpqFKwyqSOFb2QOol5xJd
HeIGfbqku6jcJjzS4AwT6JEyILUognHthq0sd3vw1KtKY2Yww3od7qJnRQtIikoMOTtJrUOcHvt4pc5aBbyH
oXYJGhof1J1Pu7rVWoi3i07xfUkR7POBGCxXNvNwlz4UEOhUltbkckXA1i4TrYLmAYSSjVMlfx2nq9Quu9e
RYoMqVa3qSWAEKlvHqjXOXkcMb6TcKQMJAD1EoaY3WR49SWFvGzgqJRBVv0whBtB4rXrOE6DDCxFT
rncsIjftFO6t59tgPhopkVIQvPpItR35dHbBEzrucWFv1Msvatfk1oiDag6oIuyAmKRdUxm4LXGGYSsntkEw
Lb3gfZcOhY7TxjmXIkzGTZygVyRthpvbBhprMUBWCn8HplDs1KjSDziLEbHs04DUl77FwCz9I8R7gZbH4a
eurorxN2LPt5mJzmzy6HgZFCFgXS4TunEffsURLNJs6DBRbIHRBLyIBEOD12I6Vs2g60d84r9YlpVsWisGYz
g95Z9R89SnbKU9XUypCbvWX094J1XmICMbUFkcFkxF4SSP4Mp1tB7Qlp2jEy2BFZaAkfQqzrV7xBj0twT
NlS9VLYqTQcxdQJqEF2IttMnlG44r8Ez5MklRMgl9hv8aohwTw6Zm6gs1ZJIKa5br6U44juLQ4JKcNDnJ7n
J4GbTfpw13DBJfO3Ra9LjlTWqFWnIfmoYF0gDuIZPbzSrRSAnTY4tj1459BBNN7lb7cmThGx6Gw4bLzKZ7
JVOAbCtbHG3Q4wxEjtfhzxBsc7Le8vttOHPCgj3E3LlvQfSLApGijHz8PwzTPXBYjRG5PgatPi6sI9mBrJLFdg
12uK723OPh8Hwahh6Sxh7wDRNcjlEqdq4Yek4EKfDViYLS8DoVJphtJ9NU2qsXDyYC8m48OFdg0632HP
YI903FdquOeUbwBM5G4S0WVPJiYj1tW0NNJWGUJJ2h7o54OdvAAt7OeZWdkaIFOi6dXdd351MxmBK
4cCN7mVASAnkY4e0IGN6O5zcSvZflpasyEIetNZXnzxznV3uPkpv9GJwIa8ECN5icQUcffdXm4eLfCfuUSsl
M9MQhktgnYpyH9F5RiXdOE3GgWVtVqaGXr6dkIqpq2IT0njn0HlqWwHTGTewbDHAKQnkzvfgp4EJ1U
vcprxiMdR1NFcnsUgXNTgXewOaHppw6y45BkR2UULJnOLs5CMLbo6W28WTujcbVLwabRfBKuvTFh2n
LOvX2xhpKhPpUOEjmw3ATxCnBdRsurMvu8hRsdBBabjwcUkLAK2rfmrRx6UEU98626HmPjoRFf61WLV
P7h7K4ZnxZX7FUNumGzXjFdxeeGRMsrOUejR6rR3Q17rIJeJ3vGAXXpke0cfl3AdIjyKkE9KpyPDMdTu9M
VSxEa6igpNoBCAWtI2KwEyd3nAwHIS31FbR2oRi4wSIWTPA9POyGjaejK7UZ69EgMSZ9tBIcxfaBoEy2K
QIoYeLFaz1ZYV8JSi3Ttz5LIV2glz6SFKZ8Bnoe4RGeDhEVpc8q6qTcJBmFnlvHGV<script>alert(foo)</scri
pt>: Output from the phpinfo() function was found.

+ OSVDB-3233: /icons/README: Apache default file found.

+ /login.php: Admin login page/section found.

+ 9308 requests: 0 error(s) and 36 item(s) reported on remote host

+ End Time: 2017-11-09 10:20:40 (GMT-5) (31 seconds)

+ 1 host(s) tested