



**Abertay
University**

Web Application Security Evaluation

Jack Wilson

Abertay University

White Paper

BSc Ethical Hacking

2016/2017

TABLE OF CONTENTS

1. Introduction.....	3
2. Vulnerabilities Discovered and Countermeasures.....	4
2.1 Database Schema.....	4
2.2 Hidden Source Code Message.....	4
2.3 Directory Browsing.....	4
2.3.1 Edit Apache Config File.....	5
2.3.2 Disable Using .htaccess.....	5
2.4 PHPinfo.php.....	6
2.5 Temp Folder.....	6
2.6 Password Update.....	6
2.7 User Enumeration.....	7
2.8 Weak Passwords.....	8
2.9 SQL Injection.....	8
2.10 Cross-Site Scripting.....	9
2.11 Cross-Site Request Forgery.....	10
2.12 Hiding Pages.....	11
2.13 Brute-Force Attack Prevention.....	12
2.14 Unsafe Credential Transmission.....	12
3. References.....	13

1. INTRODUCTION

Previously, a penetration test was conducted on the given website knowing only the website URL and a student's login credentials. A report was written explaining the vulnerabilities found and how they were exploited.

OWASP's Web Application Protection (WAP) is a tool that analyses PHP code to detect input-based vulnerabilities (such as SQL injection and cross-site scripting) and corrects the code by inserting additional code to remove the vulnerability. (OWASP, [no date]). The aim of this report is to conduct further research and testing on the same website with full access to the source code, vulnerability reports from OWASP WAP and the vulnerabilities discovered in the previous report to suggest countermeasures for the vulnerable areas of the website.

2. VULNERABILITIES DISCOVERED AND COUNTERMEASURES

Multiple vulnerabilities were discovered in both the initial investigation, and through the source code auditing. The vulnerabilities are detailed throughout this section with possible countermeasures suggested.

2.1 DATABASE SCHEMA

The schema for the SQL database was included in the robots.txt file. This stops search engines from reading it but does not prevent users finding it. Placing the schema within a folder named *CLVUQZELLYV* would have almost certainly stopped a brute-force dictionary attack from finding the folder, however, adding that folder to the robots made it easy for an attacker to find.

An alternative method to protect the schema would be to leave it within the same folder, and include it within the robots.txt file as before, but password protect the directory the schema is in by using htaccess authentication. This would involve uploading two files (.htaccess and .htpasswd) to the same directory with the following content:

.htaccess

```
AuthType Basic
AuthName "Password Protected Area"
AuthUserFile /<full path to .htpasswdfile>/.htpasswd
Require valid-user
```

.htpasswd

```
user:password
```

In the above example, the .htpasswd file would contain login credentials for as many users as require access to the directory. It is strongly recommended to store the password encrypted in the file. (htaccess Tools, [no date]).

2.2 HIDDEN SOURCE CODE MESSAGE

A hidden message was found in the source code of the homepage (*/index.php*) on the first line that revealed the password for the MySQL root user. This information should be private and it is recommended to remove the comment.

2.3 DIRECTORY BROWSING

The web server was running Apache 2 and it was found that directory browsing was enabled. This option can allow users to find hidden folders and files that are not linked from other pages (as demonstrated in *Figure 2*, below). This can be disabled using one of two methods:

2.3.1 Edit Apache Config File

The first option to disable directory browsing is by editing the Apache configuration file found under `/etc/apache2/apache2.conf`. Remove the word *Indexes* from the options section under `<Directory /var/www>` as shown in *Figure 1*. (Stack Overflow, 2010).

```
<Directory /var/www/>
    Options FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Figure 1: Apache Configuration File

2.3.2 Disable Using .htaccess

An alternative to disable directory browsing is to edit the `.htaccess` file and add the following line:

`IndexIgnore *`

This disables listing any files and folders within the directory. This can also be changed to prevent displaying a specific file type within a directory. (.htaccess-Guide, [no date]). For example, to only hide PHP and JavaScript files, the line of text could be edited to say:

`IndexIgnore *.php *.js`

It is safest to disable directory browsing entirely by editing the Apache configuration file, however if having access to directory browsing is necessary for select directories or file types, the `.htaccess` option can be used.

Index of /1501838/images

Name	Last modified	Size	Description
 Parent Directory		-	
 Students 3K - Free r.>	2013-07-20 05:54	203	
 add.png	2013-07-20 05:54	2.0K	
 bamboo.png	2013-07-20 05:54	16K	
 bodybg.jpg	2016-07-17 06:45	2.4K	
 btmbg.png	2013-07-20 05:54	76K	
 comment.png	2013-07-20 05:54	884	
 course.jpg	2013-07-20 05:54	2.7K	
 delete.png	2013-07-20 05:54	1.4K	
 edit.png	2013-07-20 05:54	1.9K	
 fbbg.png	2013-07-20 05:54	2.1K	

Figure 2: Example of Apache Directory Browsing

2.4 PHPINFO.PHP

The file `phpinfo.php` provides a lot of details about the server such as:

- The web server type and version number
- The IP address of the host
- The operating system and version number
- The server administrator's username

Such information can be useful for debugging purposes, but should be kept internal and out of public view. One suggested fix to hide this file is to edit the `.htaccess` file. (Perishable Press, 2013). Adding the following code can allow access from specified IP addresses, and deny access from any other IP address:

<Files php-info.php>

Order Deny, Allow

Deny from all

Allow from 192.168.1.100

Allow from 72.217.23.14

</Files>

2.5 TEMP FOLDER

A folder (`/temp`) was found on the server which contained information on the SQL injection prevention techniques in a file called `sqlcm.bak`. Not only were the prevention rules poor (only containing four different rules), they were publicly available meaning if a hacker found the folder they would be able to circumvent the rules easily. Additionally, the line of code that includes the rules was commented out in the `/admin.php` source code, meaning that page was not even making use of the limited rules.

This file found on the server was also a backup file of the rules (with an extension `.bak`, as opposed to `.php`), meaning it is not required for any functionality. This should be removed from the server and backed up elsewhere, out of public view.

2.6 PASSWORD UPDATE

It was observed that the password confirmation box on `/Changepassword.php` was not verifying the password. This would simply require a few extra lines of PHP code to fix the flaw such as:

```
if($_POST[“NewPassword”] == $_POST[“ConfirmPassword”])  
{  
    //Send password update  
}  
  
else  
{  
    //Throw error  
}
```

During cross-site scripting testing, the script `<script>alert(1)</script>` was placed into the new password field on `/Changepassword.php`. The XSS attack was unsuccessful, but it was noted that the end of the new password was cut off, as there was a maximum password length of 20 characters. A warning should be in place to tell users of a maximum password limit, instead of accepting whatever the user inputs and cutting the end of the password off.

2.7 USER ENUMERATION

It was found that user enumeration was possible on the student login page (`/viewresult.php`) because of the displayed error message. If a login was attempted where the user account did not exist in the database, a JavaScript alert box would appear as shown in *Figure 3*. The error message should be changed to something more generic (such as the error message displayed on `/admin.php`), which prevents an attacker from finding out if an account exists or not, and just says that either the username or password is incorrect, and does not specify which.

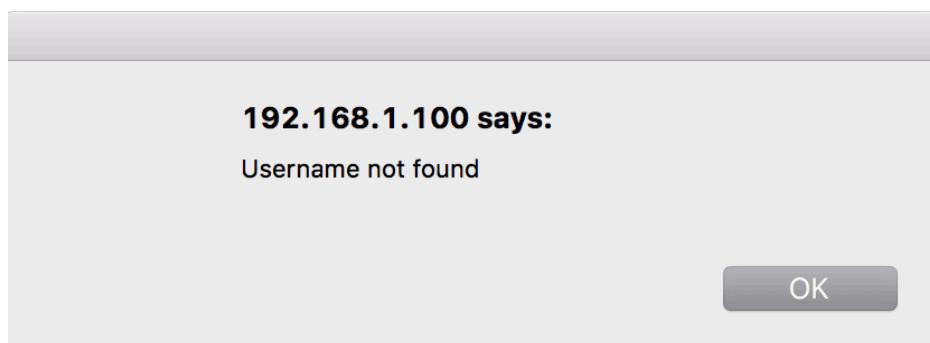


Figure 3: Error Message That Allows for User Enumeration

2.8 WEAK PASSWORDS

Per the vulnerability logging provided, the default administrator password was “friend”. The code for addadmin.php indicated that passwords were being hashed using MD5. The MD5 hash of *friend* is *3af00c6cad11f7ab5db4467b66ce503e*. The hash dumped using SQL injection for the same administrator account was *f1c70784cef9f1986cf78c56f32e6cd* which not only did not match, was not a valid MD5 hash (being 31 characters long). Login was attempted using the user account *123* and password *friend*, which failed. This indicated that the password for the administrator account was broken.

Had the hash not been broken, the password was very simplistic, being susceptible to a brute-force password attack. This is poor practice for any account, never mind an administrator account.

It is recommended to enforce a stricter password policy for all users, perhaps ensuring passwords include a mix of upper and lower case letters, numbers, special characters and enforcing a minimum length. Additionally, every password (not just lecturers and administrators) should be stored using a more modern and secure hashing function such as bcrypt, scrypt or Argon2, as opposed to the dated, and insecure MD5. (Paragon Initiative, 2016).

2.9 SQL INJECTION

SQL injection vulnerabilities were found on multiple pages throughout the website. Firstly, a vulnerability was discovered during the initial testing phase on the student login page (*/viewresult.php*) that allowed for any data from the *studentinfo* database to be read using a custom SQL union select query. Authentication bypass was also possible on the same page by entering a legitimate student number in the *Roll No* field, and entering ‘*OR 1=1#*’ in the password field. This would allow for a student’s records to be viewed without authentication. There was an attempt to prevent SQL injection by adding the following countermeasure:

```
<?php
if (preg_match ('[1=1|2=2|union]', $username))
{
    echo '<script language="javascript">';
    echo 'alert ("Bad hacker. We are filtering input because of abuse!");';
    echo '</script>'; $result = "";
```

```

}
?>
```

The above script prevented a command similar to ‘*OR I=I #*’ from being executed in the *Roll No* textbox, but there was no authentication whatsoever in the password field, so commands could be injected there. The authentication would not stop an attacker entering ‘*OR 3=3 #*’, ‘*OR A=A #*’ or any other similar command where the condition was true in the *Roll No* textbox. It was also possible to automate the SQL injection attack using a program such as SQLmap, however the countermeasures are the same regardless of attack method.

There are multiple methods to prevent SQL injection. The simplest method could be to disallow special characters such as apostrophe’s and semi-colons, and validate the input server-side to avoid tampering with a proxy. This however, can present further problems in certain cases. For example, if a user’s surname was O’Sullivan they would have to misspell their name (as *O Sullivan* or *OSullivan*) to register for the website, and possibly cause complaints from users, or users submitting support tickets saying they cannot register for the website.

Another, better, alternative is to continue to use client-side and also implement server-side validation. Validating client-side ensures legitimate users are told instantly when, for example, their email address is invalid and the server-side validation using prepared statements and parameterised queries prevents an attacker from bypassing the input validation using a proxy.

Prepared statements send the SQL queries and user input separately to the server, parse the user input and then executes the query. Using this method, the database server treats the user input as a string, rather than a query. So if a user sends ‘*OR I=I #*’, from a user login page, the server simply tries to find a user called ‘*or I=I #*’, rather than bypassing the authentication mechanism. (StackOverflow, 2008).

2.10 CROSS-SITE SCRIPTING

XSS can be complex to test for, with several vectors being vulnerable to attacks. This includes executing scripts through input boxes and through URL manipulation. The simplest way to prevent XSS attacks is to encode everything the user is sending. For example, if a malicious user was to send the command:

```
<script>alert(1)</script>
```

This would be encoded to:

```
&lt;script&gt;prompt(1)&lt;/script&gt;
```

Which would prevent the malicious JavaScript from executing. There are several open-source libraries that can be implemented to assist in preventing XSS attacks.

PHP AntiXSS

This library automatically detects the encoding of data that should be filtered. The library can also compare the data against a whitelist (approved input, such as a regular expression the web design team create) and against a blacklist (denied input, such as the string `<script>`). (Google Code, [no date]).

Kses

This code analyses input being sent to the server and removes unwanted HTML elements and attributes specified by the user. It has multiple functions such as:

- Only allowing HTML elements and attributes approved by the website designer.
- Handling element and attribute names that are case sensitive.
- It handles malformed HTML such as never-ending quotes and tags.
- It will remove `<` and `>` characters.
- It checks for minimum and maximum lengths for user input to prevent buffer overflow attacks and denial of service attacks.
- The code allows for whitelisting URL protocols such as HTTP and HTTPS, but can disallow other protocols such as FTP, to prevent a user from visiting <FTP://example.com>.

There are several other functions within this suite, but the ones listed above are the key functions. (GitHub, 2013).

There are several other libraries that have similar functions including *HTML Purifier*, *xssprotect* and *XSS HTML Filter* which all have very similar functionality, but use different languages. For example, *xssprotect* is written in Java. Some will be easier to integrate into the environment than others. (InfoSec Institute, 2013). In this case, *PHP AntiXSS* would be easier to implement since a large majority of the website is written in PHP already.

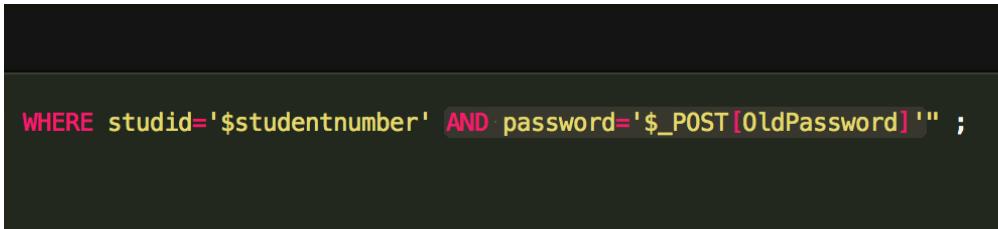
2.11 CROSS-SITE REQUEST FORGERY

Cross-site request forgery was available using the authentication bypass mentioned in *Section 2.8*, by entering a legitimate user account into `viewresult.php` and ‘`OR 1=1 #`’ into the password field. Navigating to `/Changepassword.php` as that user allowed for their password to be changed without prior knowledge of their old password. Analysing the source code (*Figure 3*) for this page shows why:

```
1  <?php
2      $sqlupd="UPDATE studentdetails SET password='$_POST[NewPassword]' WHERE studid='$studentnumber' ";
3  ?>
```

Figure 4: Password Update Source Code

As demonstrated, the code is not doing any verification that the old password is correct (or that it was even entered) it is only updating the password field for the specified user account. The code to verify the old password was provided with the source code, however it was not implemented into the production version of the website. To verify the old password, the query should be amended as shown in *Figure 4*.



```
WHERE studid='$studentnumber' AND password='$_POST[OldPassword]' ;
```

Figure 5: Updated Change Password Query

2.12 HIDING PAGES

Several pages were discovered throughout the website that allowed unauthenticated access to insert entries to the *studentinfo* database. The pages included:

/courseinsert.php

/subjectxx.php

/lecture.php

/studentins.php

/addadmin.php

/attendance.php

/exam.php

Adding a subject, as an example, would require (authenticated) access to */subject.php*. Clicking the add subject button would spawn an email window, which had an iframe of */subjectxx.php*, a page with the input boxes for adding a subject to the database. However, */subjectxx.php*, and the other pages listed above were not authenticated, so if an attacker was to find these pages using Dirbuster (or a similar program) they would have unauthenticated access to insert and update the database.

Some additional pages were also discovered which allowed unauthenticated access to the administrator dashboard, and to read and reply to messages from the *Contact Us* page. These pages were:

/dashboard.php

/contactview.php

/inbox.php

It is therefore recommended to require an administrator login to view the various pages listed above.

2.13 BRUTE-FORCE ATTACK PREVENTION

It was noted that during the testing there were no measures in place to prevent a brute-force password attack. There are a few options to mitigate a brute-force attack. The first option is to implement a temporary account lockout feature, where access to the account is completely disabled for a set amount of time. This would stop an attack; however, it means a legitimate user may not be able to access their account.

The second option is to install an Intrusion Detection System (IDS) within the network. An IDS will capture and monitor all traffic moving through the network to detect potentially malicious packets (such as viruses, denial of service and brute-force attacks).

An IDS can work in one of two ways. Firstly, through Signature-Based IDS, where the system detects packets that are known to be malicious through analysing the data. For example, an attacker may be sending a known malicious payload to a web server, the IDS would recognise the payload and alert a network administrator to the attack.

The second type of IDS is called Anomaly-Based IDS. This system takes a baseline measure of activity over a network and monitors for any abnormalities (such as a brute-force attack, or a user being logged in at a strange time of day). This system may take longer to develop, as custom rules must be implemented to match the threat model, but it can be more accurate than signature-based IDS with less false positives.

After an IDS detects suspicious activity, it can simply alert a network administrator, or drop the packets automatically from the given IP address. This method not only prevents the attacker from continuing an attack, but it still allows a legitimate user to access their account.

2.14 UNSAFE CREDENTIAL TRANSMISSION

There is currently no support for HTTPS on the website. If an attacker was capturing traffic they would be able to steal information such as usernames and passwords. HTTPS sends all data to and from the website over an encrypted connection, making it harder for an attacker to steal credentials using this method.

3. REFERENCES

OWASP. OWASP WAP-Web Application Protection. [no date]. Accessed 30 November 2016 at https://www.owasp.org/index.php/OWASP_WAP-Web_Application_Protection

Stack Overflow. Prevent SQL Injection in PHP. 2008. Accessed 2 December 2016 at <https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php>

Stack Overflow. Disable Directory Browsing. 2010. Accessed 2 December 2016 at <https://stackoverflow.com/questions/2530372/how-do-i-disable-directory-browsing>

.htaccess-Guide. Disable Directory Listings. [no date]. Accessed 2 December 2016 at <http://www.htaccess-guide.com/disable-directory-listings/>

Perishable Press. Is It Secret? Is It Safe? 22 August 2013. Accessed 2 December 2016 at <https://perishablepress.com/htaccess-secure-phpinfo-php/>

Htaccess Tools. Password Protection with htaccess. [no date]. Accessed 2 December 2016 t <http://www.htaccesstools.com/articles/password-protection/>

Paragon Initiative. How to Safely Store Your Users' Passwords in 2016. Accessed 7 December 2016 at <https://paragonie.com/blog/2016/02/how-safely-store-password-in-2016>

Google Code. php-antixss. [no date]. Accessed 7 December 2016 at <https://code.google.com/archive/p/php-antixss/>

GitHub. kses. 30 July 2013. Accessed 7 December 2016 at <https://github.com/RichardVasquez/kses/>

Infosec Institute. How to Prevent Cross-Site Scripting Attacks. 10 October 2013. Accessed 7 December 2016 at <http://resources.infosecinstitute.com/how-to-prevent-cross-site-scripting-attacks/>