Scenario Week 4

*Move-and-Tag Competition*

COMP205P – Software Engineering

Team Manticore

|  |  |
| --- | --- |
| Kazuma Hochin | zcabkho@ucl.ac.uk |
| Sam Pham | zcabsph@ucl.ac.uk |
| William Lam | zcabwhy@ucl.ac.uk |
| Zi Sim | zcabzjs@ucl.ac.uk |

*Table of Contents*

Preface

I. Video and GitHub

Index

1. Web Application’s Architecture

1.1 Introduction

1.2 Snippets! Architecture

1.3 Azure VM

2. Test Environment

3. Testing Review

3.1 Penetration Testing Overview

3.2 Web Crawling

3.3 Resource Discovery

3.4 HTTP Proxy

3.5 Password Brute Forcer

3.6 SQL Injector

4. Vulnerabilities and Solutions

4.1 SQL Injection

4.2 Weak Authentication and Session Management

4.2.1 Password Policies

4.3 Cross-Site Scripting (XSS)

4.3.1 Stored XSS

4.3.2 File Upload XSS

4.3.2 Reflected XSS

4.4 Insecure Direct Object References

4.4.1 Cookie Manipulation

4.5 Security Misconfiguration

4.6 Sensitive Data Exposure

4.6.1 Password Hashing

4.7 Missing Function Level Access Control

4.7.1 Elevation of Privilege

4.7.2 Authorisation Checks

4.8 Cross-Site Request Forgery (CSRF)

4.9 Using Components with Known Vulnerabilities

4.10 Invalidated Redirects and Forwards

4.10.1 Adding links to Snippets

4.11 Denial of Service

4.11.1 Unrestricted File Upload

4.12 Man in the Middle Attack

5. Tools Used

5.1 Website

5.2 Local Hosting

5.3 Public Hosting

5.4 Testing

5.5 GitHub

6. Points Allocation

7. Appendix

8. Reference

I – Video and GitHub

The following report is mean to be read with and associated with the following YouTube video:

[*https://youtu.be/9rU3wn4zYN4*](https://youtu.be/9rU3wn4zYN4)

For all source code of both the insecure and secure versions of the web applications can be found here:

[*https://github.com/zcabwhy/Website-Security*](https://github.com/zcabwhy/Website-Security)

*Team K*

1 – Web Application’s Architecture

* 1. – Introduction

Snippets! is a web application that allows users to post snippets and store files on the web server that is based on the Google Gruyere application. Snippets! meets almost all the requirements that was given. So it has a login and registration system and users can create snippets which can be seen by all. Each user has their own profile and can change a number of information such as profile picture, password and even the username. Some users can be administrators and have the power to give or take away permission of a user to post and turn other users to administrators. And lastly, there is a cloud based storage area.

We did have a link on the user’s homepage that the user can give out but after going through the vulnerabilities and began hashing user’s passwords, this was no longer possible. The only other specification that we missed was the ability to change your profile homepage colour.

1.2 – Snippets! Architecture

With our initial website application, we did not adhere to a design pattern because none of us were very familiar with PHP, MySQL and the software required to run a web server namely LAMP and other variants. Each page was isolated and nothing was shared except for the style CSS file.

But during discovering and fixing security issues, we noticed that there was a lot of repeated code and that the changes needed to be applied everywhere. So, in the end we implemented the MVC design pattern or Model-View-Controller pattern in our website architecture. This meant that we have avoided duplicated code and it is easier to fix problems that may arise application wide with one point of failure rather than multiple.

The MVC pattern is about how the user interacts with the software and is made up of three different parts:

* The model – is where the database is held and the associated software to modify the data. So, in the case of our web application, it is the MySQL database and the snippetModel.php which holds all the queries we are using. No other file currently has access to the database.
* The view – is where the layout for the websites are held such as HTML and CSS tags as well as taking the data from queries request by the controller and displays them such as on a table. For our development, all views are in views folder.
* The controller -  are modules that sends commands to change the model as well as the data needed. It also generates what the user should see based on the view that is required to be used. In our web application, there is one main controller which is the snippetController.php. This PHP deals with generating what the users see. All the other files are commands to be sent to the model when needed.

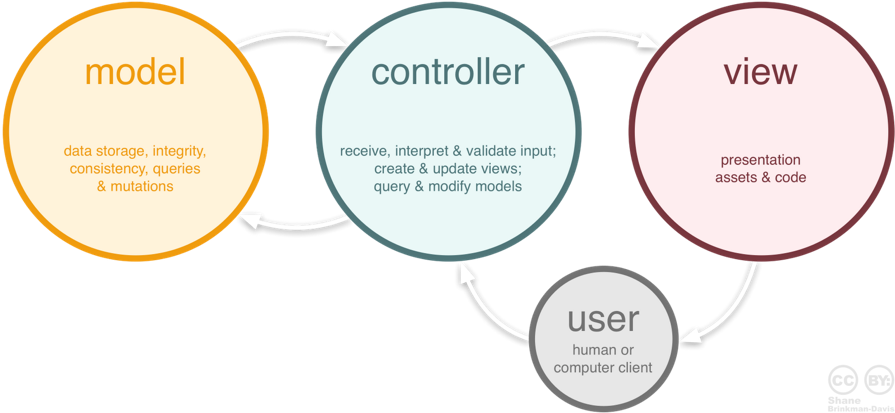


Figure 1. The flow of the MVC pattern

1.3 – Azure VM

Our web application is currently being hosted on a virtual machine provided by the Azure platform. Both the insecure and secure versions are hosted on the Azure VM for testing purposes. A network security group was used to monitor the request and the response packets sent between the client and the server.

***Please refer to the Appendix for the commands used to set up the virtual machine used on Azure.***

2 – Test Environment

For the environment, we manually tested our Azure Virtual Machine with the terminal and a browser. Initially, we set up our servers and databases without changing the default username and passwords. But after a vulnerability was found with security misconfiguration, we decided to change the database password to a password of length 20.

We conducted penetration testing using python testing tools through the terminal with the MAMP localhost. We used python libraries: Scrapy and Request to create these testing programs and these were mostly automated, such as: the web crawler and the password bruteforcer.

A git has also been initialised in the VM, which pulls the changes we made on Github onto the server.

We also used Vega to automatically test the secure website and to find vulnerabilities. Vega was used alongside MAMP i.e. localhost as Vega actually took up a lot of bandwidth.

3 – Testing Review

3.1 – Penetration Testing Overview

In order to identify security weaknesses in our web application we followed the penetration testing methodologies covered in the lynda.com video tutorials: “Learning Python Web Penetration Testing” by Chrisitan Martorella. Penetration testing is an offensive testing used to identify the weaknesses and vulnerabilities in a web application. We used Python to create our own testing program and in addition modified the existing Python penetration testing tools to automate the penetration testing activities.

We first used web crawler using Scrapy (Python library) to extract useful information from the web application such as: forms and links to identify the components and their relationships. We then used resource discovery bruteforcer using Requests (Python Library) to identify hidden resources which cannot be found by the webcrawler if they are not linked anywhere in the application or are embedded in the Javascript code.

We used MITM proxy (HTTP Proxy) to intercept messages and to read packets. We used Python to create a script to intercept and print the URL, the status code and the cookie of the request and response packets.

After we have mapped out the application structure and its components we then moved on to discovering and exploiting the vulnerabilities. We used the SQL injector (Requests) to exploit the SQL injection vulnerabilities within the application and the password brute forcer (Requests) to discover authentication issues.

3.2 – Web Crawling

Web crawling allows us to extract useful information from the web application to identify the components and how they are linked with each other to map out the application structure. We used Scrapy, a web crawling framework written in Python to perform this task. Our scrapy project called “snippets\_crawler” has a file called “snippets\_spider.py” where given the start URL it will recursively access all the links in the page and continue until it covers the whole application. The crawler will extract the information we need and will save it in a JSON file which can be used as a starting point to identify the vulnerabilities in the later phase.

Our snippets crawler will extract the following information per page: links and forms which takes user inputs and its parameters. The code below is the JSON file where the extracted data is stored.

JSON File:

**{"form destination: ": "addsnippettodb.php", "form url": "http://51.140.53.253/newsnippet.php"},**

**{"input: ": "<input type=\"submit\" class=\"btn btn-block btn-primary\">"},**

**{"links": ["<a href=\"profile.php\">Logged in as </a>"]},**

However, the crawler cannot find hidden resources that are not linked anywhere to the application. So we used Brute force resource discovery to discover these hidden resources.

3.3 – Resource Discovery

This process attempts to locate and retrieve all the available files and links on the website. A resource brute forcer matches words from a wordlist to find links which are accessible on a website. The brute forcer creates multiple threads that replaces the given URL with the words and sends HTTP requests to the links. If the HTTP request status code returns as anything other than 404, the link is then printed out on the terminal, along with the status code.

To run the code on terminal, the tester should type “python bruteforce.py -w http://51.140.42.234/DUMMY.php -t 5 -f commons.txt”. This produces the accessible links in the terminal.

3.4 – HTTP Proxy

HTTP Proxy is an intermediate server between the client and the server that can intercept and read the packets. We used Mitmproxy which is an interactive console that can read and intercept packets and display information about the packets. The Mitmproxy can run inline scripts written in Python which can be used to print the packets’ information onto a text file. The main purpose of the HTTP Proxy is to find URL links that are embedded in Javascript as the crawler cannot find links Javascript code.

We used a python file called ‘mitm.py’ to print the HTTP request method, the request URL, the request cookie, the response status code and the response cookie for each pair of request and response packet. This information is written to a text file called ‘httplogs.txt’.

httplogs.txt:

**GET http://51.140.42.234/profile.php**

**PHPSESSID=8o2eb9ui8ug0vpoqdbvksejqu0**

**200**

**PHPSESSID=8o2eb9ui8ug0vpoqdbvksejqu0**

This information will be used in SQL injection and XSS script inject when we are finding vulnerabilities.

3.5 – Password Brute Forcer

The password brute forcer tries different passwords for a username to find which password is the correct one. In the file ‘passwordBruteForcer.py’, the code takes in the username of the account it is hacking, then it tries different passwords from the ‘password.txt’ until it finds the correct password and prints it out onto terminal.

The code also filters the passwords using the number of characters of the response when a password is used. This is because the correct password produces a different response compared to the wrong passwords so only the correct password is produced through this method. The code uses multi-threading to speed up the brute forcer.

To run the code on terminal, type ‘python3 passwordBruteForcer.py -w http://51.140.42.234/login.php -t 5 -f password.txt -p "name=me&password=FUZZ" -c 2469’ in the directory where the code is. Please note the URL of the website might change so please use the most up to date URL.

3.6 – SQL Injector

A SQL injector is used to extract certain valuable data from a website that uses a SQL database through an SQL vulnerability in the website. The SQL injector relies on the tester discovering an SQL vulnerability in a website, then proceed to use the injector on it. The injector first uses some test characters to check whether an SQL error message appears on the URL page, which would indicate that the page is indeed prone to SQL injection.

The next step is to automate the extraction of data from the SQL database. The SQL injector then inserts a specific code snippet that can extract the column names of  database, the version of the database and the current user. These data would then be quite useful in extracting further data or even wrongfully updating data in the database.

To run the code, you have to sign in to the website, and in the terminal, type “python SQLinjector.py -w http://51.140.42.234/changeSnippet.php?newSnippet=FUZZ -i injections.txt” This would then produce the column names available in the database, the current version of the database and the current user of the database.

4 – Vulnerabilities and Solution

4.1 - SQL Injection

SQL injection attack injects script into the SQL queries to access the database and read/write database.

changePassword.php :

**$sql = "UPDATE users SET password='$newPassword' WHERE name='$name'";**

The code above is used to change the current user password to the ‘$newPassword’ which matches the current user name ‘$name’.

The attacker may create an account with username: **a’ OR ‘a’ = ‘a** . If the attacker then goes to change the password the SQL query will become:

**UPDATE users SET password='newPassword' WHERE name = ‘a’ OR ‘a’ = ‘a’;**

where **‘a’ = ‘a’** will always be true and hence the password for all the other users will be changed to **newPassword.** The problem here is that the user input is used to dynamically construct SQL query. This problem can be fixed using the ‘prepare’ statement:

**$sql = $conn->prepare("UPDATE users SET password = :newdata WHERE name = :name");**

**$sql->bindParam(':newdata', $newdata);**

**$sql->bindParam(':name',$name);**

Using the ‘prepare’ statement the SQL query will be parsed and compiled separately to the parameter ‘$name’. It will then compare whether the name matches ‘$name’. So in the above case it will search for the name which matches **a’ OR ‘a’ = ‘a.** The parameter will not be combined to the SQL query and will be treated merely as a string literal preventing SQL injections.

4.2 - Weak Authentication and Session Management

With user registration and signing in being a part of the website, a potential authentication issue may arise leading to leakage of confidential information or data of the website. This would allow attackers to retrieve session ids or even gain admin access if the issue is not fixed. Some methods of authentication would be to strengthen the passwords, provide password change controls, hashing passwords before storage and through stronger session ID protection.

4.2.1 Password Policies

In both the registering section and the changing password section, there are a few password policies used to prevent weak authentication. These include limiting the length of password to 8 characters or above and making the password case sensitive. This makes it harder to use a password brute forcer to try different passwords because there are many combinations available.

The code below in ‘createUser.php’ directs the user to the register page and displays an error to the user when the password typed is less than 8 characters.

createUser.php

**if (strlen($password) < 8) {**

**header("Location: /?action=register&status=password");**

**exit();**

**}**

4.3 - Cross-site Scripting (XSS)

XSS attack injects scripts into the web application. XSS applies to any web application which accepts input from users.

4.3.1 Stored XSS

Stored XSS attack is when the malicious scripts are stored on the server. The victim could then execute the script from the server.

Suppose an attacker adds a snippet with the following:

**<a onmouseover="alert(1)" href = "#">read this!!!</a>**

The snippet will only show the message ‘read this!!!” which can be seen by any users. If the victim puts the mouse over the message the javascript code will be executed.

Solution:   
The problem here is that the attacker can add messages with characters (such as:”<”,”>”) which allows the attacker to inject html/javascript scripts. The simplest solution will therefore be to escape them using the htmlspecialchars function. Htmlspecialchars function converts special characters into a safe form so that untrusted user input can be displayed safely without it being executed as a code which prevents attackers from injecting scripts.

addsnippettodb.php:

change **$message = $\_GET["snippet"];**

to **$message = htmlspecialchars($\_GET["snippet"]);**

So message = “<p>hello</p>” will be encoded as **&lt;**p**&gt;** hello **&lt;/**p**&gt;**”

Now the attacker cannot inject scripts through forms.

4.3.2 File Upload XSS

The user can upload a HTML file containing javascript code and that code can run on the website.

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

If the code is uploaded as a HTML file, the code displays the PHP session id.

Solution:

There should be a limitation to the file extension so only jpeg, jpg, png, txt, pdf and gif files can be uploaded. Otherwise an error is displayed to the user.

$expensions= array("jpeg","jpg","png","gif","pdf","txt");

if(in\_array($file\_ext,$expensions)=== false){

$errors[]="Extension not allowed, please choose a JPEG, JPG, PNG, GIF, PDF or TXT file.";

}

4.3.3 Reflected XSS

In reflected XSS, the user can change the link name parameter when accessing the ‘userdetails’ page through a URL. This can cause the user to insert script into the URL and hack the website. An example is <http://51.140.42.234/userdetails.php?linkname=%3Cbutton%3ETry%20it%3C/button%3E>. This URL causes a button to be displayed which means HTML was inserted into the code.

The solution to fix the problem is to check with the database whether the link name exists as a username and if it does then their snippets are displayed. If not an error message is displayed to the user saying the username does not exists.

if(mysqli\_num\_rows($result\_validUser) == 0) {

       echo "<h4>User does not exist!</h4>";

     }

     else {

        //Run code.

     }

I have also fixed the problem by using the htmlspecialchars function to remove html tags so code cannot be inserted.

<?php echo htmlspecialchars($\_GET['linkname']); ?>

4.4 - Insecure Direct Object References

This refers to when an authorised user gains unwanted access to files or web pages, skipping typical procedures while doing so. For example, a user may gain access to the profile page without creating a username. This might potentially harm the website, storing unwanted data or creating null values in database fields. To prevent that, each page checks if a user has correct authentication before presenting the page. Else, they get redirected to the login page.

4.4.1 Cookie Manipulation

When you go to profile.php without logging in and you change your username to the an existing one in the database, you have access to the person’s account as the session id name is set the username you typed in.

To fix the issue, we can verify the user’s session id at the top of every web page by checking the authorized variable in the session id and making sure it is true. This is because we will set the authorized variable to true if the user is logged in. If the user is not logged in then the authorized variable is false and so the user is redirected to the main page when it access a web page other than the login page.

login.php:

**$\_SESSION[‘authorized’] = true;**

Other webpages

**if (!isset($\_SESSION['authorized']) && $\_SESSION['authorized'] !== TRUE) {**

**header('Location: login.php'); exit();**

**}**

**else {**

**//Code runs.**

**}**

4.5 - Security Misconfiguration

This primarily happens when systems use outdated software, or stick to default settings while using the components. In our website, the database had default username and password settings, which makes it vulnerable to attackers who can retrieve data via file upload. The attacker could easily guess the password and username, allowing unauthorised access to the database and damaging the website.

With this in mind, we generated a strong password and used it for our database. The password has about 20 characters, which makes it difficult for attackers to brute force it. This prevents unwanted connection to our database which may affect the website.

4.6 - Sensitive Data Exposure

This occurs when passwords are stored in plain text in the database of the website. If an attacker successfully gains access to the database via a security vulnerability in the website, he would then gain valuable information about the users and passwords of the website. This can be prevented by hashing the passwords.

4.6.1 Password Hashing

This vulnerability occurs when someone gains access to the database. The attacker would then be able to have a record of all the passwords for every user. This can be solved by hashing the passwords. As md5 and SHA256 hashes are relatively common, the passwords can be easily decoded if they were encoded with these hashes. So, this issue is solved by using the password\_hash() and password\_verify(), which are PHP-specific functions used to hash the passwords. One advantage is that there is no existing function that can reverse the password\_hash() process, which means you cannot recover the passwords once they are hashed. This is useful as it implies that the attacker could not just search for the PHP library to decode the available passwords.

**$password = htmlspecialchars($\_POST["password"]);**

**$hashedpassword = password\_hash($password, PASSWORD\_DEFAULT);**

4.7 - Missing Function Level Access Control

This vulnerability happens when an anonymous user gains similar access as a typical registered user, or when a user gains admin access to the website, granting him unauthorised administrator status to update the database of the website.

4.7.1 Elevation of Privilege

This happens when an attacker gains unauthorised permissions to the website. In our example, only the admin is allowed to distribute author and admin roles to other users. But this can be skipped if attackers type in the following code as the URL:

http://localhost/website-security/insecure-website/makeadmin.php?optradio=1

To fix this issue, the form that distributes admin/author roles should use the “POST” instead of “GET” method as the information gets hidden from the URL.

But this does not solve the problem entirely, as there’s another way of getting unauthorised access. This is done by entering a snippet of SQL code as your username in the login page:

**a’ OR 1 = 1; UPDATE users SET admin = 1 WHERE ‘name’ = ‘John**

This elevation of privilege is done through some SQL injection to tamper with the database, allowing the user ‘John’ to be admin. This code can be slightly edited to allow every user to have the admin role.

The solution to this problem is stated above under “SQL injection”. Essentially, the user input should be processed before the SQL query is ran.

4.7.2 Authorisation Checks

The authorisation checks are implemented in every page. Essentially, this checks whether a user is given access to visit the page. Else, they would be redirected to the main signin page. More details of the solution can be seen under “Cookie Manipulation” in 4.4.

4.8 - Cross-Site Request Forgery (CSRF)

Assuming the victim has an authenticated account on a web application. CSRF attack forces to perform actions on the victim’s account. The attacker creates URL links and tricks the victim to click on the URL link on external webiste which accesses the target web application, performing certain actions without the victim knowing on their account.

Forces Victim to create snippets on their account:

<http://51.140.47.4/addsnippettodb.php?snippet=ALERTCOMMENT>

Since the request is sent from the victim’s browser the browser cannot distinguish whether the snippet form was created and sent by the same user. To protect the user from CSRF attacks we will be using CSRF tokens. The CSRF token will be generated at every state changing action and will be added as a hidden field in the form. The request will be validated by comparing the form token with the session token preventing the CSRF attack. We will also ensure that POST methods are used instead of GET methods when using forms.

newsnippet.php

**<form method="post" action="addsnippettodb.php">**

**<input type="hidden" name = "csrf\_form\_token" value="<?php echo $csrf\_token; ?>" />**

The form now has a hidden value which is called ‘csrf\_form\_token’ which is sent throught the form to the ‘addsnippettodb.php’. This form token value will be compared with the session token value and validates whether the form was sent by the same user.

**$csrf\_token = md5(uniqid(rand(), TRUE));**

**$\_SESSION['csrf\_token'] = $csrf\_token;**

**$\_SESSION['csrf\_token\_time'] = time();**

The csrf token is generated every time the user visits the page using a simple random number generator function which should be sufficient enough for the demonstration purpose. This token is added to the session csrf token every time with the time the token is generated. Now when the user submits the snippets this csrf\_token value is sent through the form using POST method which will be compared with the $\_SESSION[‘csrf\_token”] to validate the session, preventing CSRF attacks.  By adding the token time, the server also ensures that the token was created and the form sent within the time limit (5 minutes) so that only recent requests will be accepted.

4.9 - Using Components with Known Vulnerabilities

We did not use any libraries or frameworks in our project so there are no existing vulnerabilities for this category in our website. Also, all of the tools we used to build the website have known vulnerabilities so we tailored our testing tools to expose those vulnerabilities.

4.10 - Invalidated Redirects and Forwards

This occurs when users are given a URL inputs or links that would redirect them to other web pages. An attacker may attempt to produce a link that leads to a phishing site, stealing the information and credentials of the user.

4.10.1 Adding links to Snippets

In our website, a user has the ability to add snippets to be seen by other users. This would allow attackers to add clickable links that would direct unsuspecting users to malicious websites. To fix it, **htmlspecialchars()** was introduced to encode tags, preventing the inclusion of tags as snippets. More details can be seen under “Stored XSS” in 4.3.

4.11 - Denial of Service

Denial of Service(DOS) is an event which prevents regular users from accessing the website and its resources. Typically, it occurs when attackers successfully overload the servers, draining the resources of the server. This would result in a server crash, temporarily shutting down the website. In our website, an attacker can execute the attack by uploading a massive file, which would overload the file system of the website.

4.11.1 Unrestricted File Upload

The user can upload any number of files with no restrictions on the size of the files. This can cause memory overload because there is a limit to the memory in the server and can cause Denial of Service as the server might crash and stop working due to no memory.

To fix this problem, I have added code to the ‘fileupload.php’ file where the files uploaded are limited to less than or equal to 2MB. If the file uploaded is greater than 2MB then an error is displayed to the user.

**if($file\_size > 2097152){**

**$errors[]='File size must be less than or equal to 2 MB';**

**}**

4.12 - Man in the Middle Attack

Man in the middle attack is the intercepting of packets sent between the client and the server and the modification the packets’ content. Mitmproxy can be used to intercept packets sent. Also Python scripts can be created and used in Mitmproxy to modify the packets and to hack the website. One example of the man in the middle attack is to inject SQL code into the URL and hack into the website’s SQL database.

A solution to fix the man in the middle attack is to use HTTPS for the application protocol instead of HTTP because HTTPS encrypts the packets sent between the client and the server and requires a certificate for authorization. We did not implement this solution because it requires us to buy a SSL certificate to use the HTTPS protocol.

5 – Tools Used

Over the course of the project we used a variety of tools to help us build the website and the testing scripts.

5.1 Website

For the website we used HTML, CSS, Bootstrap, Javascript, PHP and MySQL.

We used HTML, Bootstrap and CSS to create the secure and insecure websites’ front end user interface and to style the websites. We used PHP for server side programming and we embedded PHP inside HTML to retrieve information from the database and the server. We also used PHP to hash the password and also for file uploading.

For the database, we used MySQL to store the snippets the user has created and stored and the different accounts created in the server. We also used MySQL to store information about admins and authorized authors and the user’s private snippet.

5.2 Local Hosting

For hosting the website, we used MAMP/XAMPP.

MAMP and XAMPP are both solutions stacks that contains MySQL, Apache and PHP used for web development. We used MAMP for Mac OS and XAMPP for Windows. MAMP and XAMPP allows us to create and test the website locally and because the software used are similar to the ones used in the server to host the website, it made deploying very easily. It also allows us to push and pull from GitHub easily because the root directory of MAMP and XAMPP’s website can be changed.

5.3 Public Hosting

For public hosting, we used Azure, Ubuntu v16.04.1, Apache, PHP v7.0.13 and MySQL v5.7.16.

Microsoft Azure is a cloud computing service that allows users to create, deploy and run web applications on data centers across the world. We were given Azure accounts to host our website on a server. The servers that we used have Ubuntu v16.04.1, Apache, PHP v7.0.13 and MySQL v5.7.16. Ubuntu is the Linux operating system used by the server and Apache is used to turn the computer into a web server. It also has PHP and MySQL to run the server side code of the website and also allows databases to be created and manipulated by the website code.

5.4 Testing

For testing we used Python version 3 and Mitmproxy.

We created our penetration testing tools for our website using python because it is a simple programming language and also has a range of libraries to support penetration testing.

For scripting the web crawler, we used the Python library Scrapy which allows us to recursively go to different web pages on the website and to find places for SQL injections and hacking into accounts.

We also created the code for resource discovery bruteforcer, the password brute forcer and the SQL injection detection and exploitation using Python. We used the Python library Requests for all of these scripts.

For the HTTP Proxy we used Mitmproxy and Python to read and intercept packets sent from the client to the server. We created a Python script to read the packets and get the URL, the status code and the cookie for both request and response packets. We also used Mitmproxy’s API to code the script.

5.5 GitHub

We created two directories to store our source code. The first one, named “insecure-website” is setup to store the code for the website with vulnerabilities. This is the initial version of the website without any fixes. The second directory, named “azure-testing” is where we made changes and fixes on the vulnerabilities, and testing them in a test environment. This version contains the secure version of the website, after we implemented the fixes we made. Besides that, it also contains the source code of the automated testing tools that we created for this application to test for vulnerabilities.

*The link to our Github page is stated in the first page of the report.*

6 - Points Allocation

Kazuma Hochin - 25

Sam Pham - 25

William Lam - 25

Zi Sim - 25

7 – Appendix

7.1 - Commands to set up Azure VM:

*sudo apt-get update*

*sudo apt-get install apache2 mysql-server php php-mysqlsudo apt-get install libapache2-mod-php*

*git clone  /react-text* [*https://github.com/zcabwhy/Website-Security.git*](https://github.com/zcabwhy/Website-Security.git)

*sudo cp -r Website-Security/secure-website/\* /var/www/html/*

*sudo rm /var/www/html/index.html*

*sudo chown www-data /var/www/html*

*sudo apt-get update*

*sudo apt-get upgrade*

8 - Reference

* “Web Application Exploits and Defenses.” [Online]. Available: http://googlegruyere.appspot.com/. [Accessed: 9-Jan-2017].
* C. Martorella, “Learning Python Web Penetration Testing,” Lynda.com - from LinkedIn. [Online]. Available: https://www.lynda.com/Python-tutorials/Learning-Python-Web-PenetrationTesting/521198-2.html. [Accessed: 10-Jan-2017].
* OWASP, “OWASP Top 10 2013-Top 10” [Online]. Available: https://www.owasp.org/index.php/Top\_10\_2013-Top\_10. [Accessed: 9-Jan-2017]
* PHP, “Safe password hashing” [Online]. Available: http://php.net/manual/en/faq.passwords.php. [Accessed: 14-Jan-2017]
* Veracode, “OWASP Top 10 Vulnerabilities” [Online]. Available: <https://www.veracode.com/directory/owasp-top-10>. [Accessed: 12-Jan-2017]