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*

1. Move-and-Tag Problem

1.1 Introduction

1.2 Finding the Solution

1.3 Visualisation

2. Algorithms & Complexity

3. Testing

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. Processing Input and Output Data

5. Distribution of Work

6. Repository

7. Reference

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.

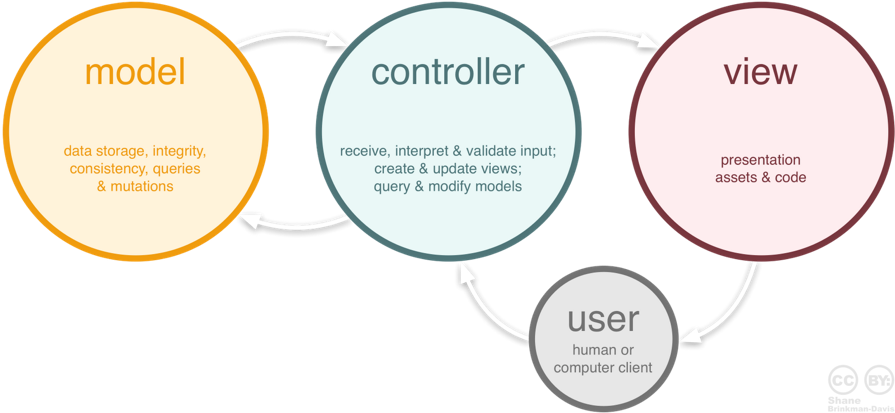


Figure 1. The flow of the MVC pattern

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

The algorithms were tested using a variety of methods. We created some simple test cases with different robot positions and different polygons. We also used the visualisation library Matplotlib to visualise the different maps and see if any of the robot paths intersect with any of the polygons. Also, we used the library to display the tree of the RRT algorithm and to make sure the code is working correctly. We also continuously submitted robot paths to the server to make sure our algorithms work even with large number of robots and polygons. Furthermore, we used the IDE debugger to debug our algorithms and to check if the variables are storing the correct values. Moreover, we manually tested the algorithms by drawing out the algorithms using small test cases through unit testing.

4 – Processing Input and Output Data

The parser (C#) was used to generate a text file for each configuration with the coordinates of robots and polygons converted into a valid format which can be used by the Python visualiser and RRT.

The python program was used to apply RRT to find a collision free path between two robots selected by the greedy passive time algorithm. The RRT program will produce a solution text file with the paths for each robot. This text file was used by the python visualiser to display the path each individual robot took.

5 – Distribution of Work

We split the design and implementation workload so that we can try out different algorithms to solve the problem. Sam and Sim both worked on trying to implement the visibility graph using Pyvisgraph and also the in-order and greedy claim and greedy passive timed algorithm. William and Kazuma both worked on implementing the rapidly exploring random tree algorithm to generate the path between the robots and also to avoid the obstacles. Sam also focused on processing the processing data, automation at a number of stages and finding what two robots should be connected. Kazuma also implemented the visualiser using Matplotlib.

6 - Repository

All our work during this scenario week can be found on in a GitHub repository linked below. It has the different algorithms that we implemented and the results of each algorithm that we implemented.

[*https://github.com/kiriphorito/MoveAndTag-Manticore*](https://github.com/kiriphorito/MoveAndTag-Manticore)

7 - 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]