National College of Ireland

<BSc in Computing(BSHC)>

<Cyber Security>

<2020/2021>

<Ioana Avram>

<17448556>

<x17448556@student.ncirl.ie>

<Python Vulnerability Scanner for Websites>

Technical Report

Contents

Executive Summary

The purpose of this report is to describe the project, why I decided to do this project and talk about the state of web security and web vulnerabilities. I discuss the background behind this project, requirements that the application should be able to carry out as well as mockup of what the code and architecture might look like. This report will also talk about the actual completed project, the evaluation and testing of it.

At the end of the project this will include testing, evaluations and future work on this project as well as future research and conclusions on the web security field.

1.0 Introduction

1.1. Background

As more companies and people use the internet , more websites are made, apps, servers and so on, more data is stored in the cloud. The current world pandemic has seen a large increase in attacks on such servers, and there aren't always enough people to check on these sites. One needs to keep up with updates, monitor networks and so on.

Therefore one could always use more apps to check the security of these sites to make sure they are updated and not exposed to any such vulnerabilities, as much as reasonably possible. Imperva states the most commonly found web vulnerabilities in 2019 continues to be Injection type vulnerabilities . They record a total of 28% of vulnerabilities were injection based with an increase of 21% compared to last year.(Bekerman, 2020)

MSSP Alerts comments in 2017 40% of the total web attacks were XSS(Cross SIte Scripting) and 28% of attacks were SQL Injection based.(Trustwave, 2018)

I also wanted to make an app for web security because I wanted to make something in Python after struggling to maintain consistency in learning Python on a regular basis.

The PYPL (PopularitY of Programming Language) site ranks Python as the most popular language as of December 2020 compared to a year ago with a growth of 18% in the last 5 years and a share of 30.34% according to Google trends statistics.(PYPL PopularitY of Programming Language, n.d.)

CodingInfinite also ranks Python as one of the most popular language according to statistics from Stackoverflow with a share of 41.7% over the last 5 years and it comments that Python has continued to grow while other languages such as Java declined.(Kumar, 2018)

And being in the security specialisation meant the project had to have something regarding computer security as such. Did a tiny bit of research and talked to some people about what I could make in that language and came across that I should make a web security app.

Now I'm sure there are many out there already, but I read in a security book that a good security person makes their own tools instead of relying on tools made by other people, as said in this comment "*Crackers were considered much less talented than the elite hackers as they simply made use of other tools and scripts without understanding how they worked*" from the book Hacking: the art of exploitation.(Erickson, 2008)

At the end the project will demonstrate Python skill and I'll have something to show for it when asked 'what have you made in x y or z language' .

1.2. Aims

The project aims to improve and to add to the web web security testing of apps by tackling some of the most common issues faced by web apps. And to bring together some scanning modules and functions under one app that will be able to scan for these issues.

1.3. Technology

The project was made with Python 3.7 and contains a number of scanning functions for the checking of common web vulnerabilities. It also employs the use of modules such as Beautiful soup, requests and system command libraries.

One of the scanning functions uses a basic PHP server set up separately for the sending of one test file containing an ordinary string to be used to test whether the website being tested can access external files.

The requests module is used extensively to access the target website url address, get data from it and send data back.

To test the input of forms on websites, the BeautifulSoup library is used for grabbing filling forms with test input, often small scripts designed to test the data validation and sanitisation of input fields.

1.4. Structure

Section 1 is the introduction to the project, some research and background information and the technology used in making the project.

In the System section (section 2) I will discuss the application. Requirements, mockups, what the project could look like and what functions and features it might contain. Section 2 also contains testing results, code and output screenshots from the actual application.

Section 3 and 4 are the further research and conclusions. What the further research and development could add to the project and an evaluation of the project after completion. Section 5 is references used in the project.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. SQL Injection Scanning:

SQL Injection scanning in a web app that has a database server back end.

Testing the app and researching how an SQL is both tested for and how the attack is implemented. Cross-referencing with how the app checks for the vulnerability and doing the attack manually on a test server, to make sure the vulnerability is actually there.

The app should be able to test for injection attacks, regarding the back end server hosting the web app, through both input forms and URL address modification.

The SQL injection scanning is relevant in a web scanner because it is one of the most common attacks on web apps.

SQL injection scanning should be done in 1 week or less.

2.1.1.2. XSS (Cross Site Scripting):

Testing the app and researching how the vulnerability is both tested for and how the attack is implemented. Cross-referencing with how the app checks for the vulnerability and doing the attack manually on a test server, to make sure the vulnerability is actually there.

The app should be able to test for most types of XSS such as Reflected, Stored and DOM Based.

The scanning for XSS is relevant in a web scanner because it is one of the most common attacks on web apps.Scanning for XSS should be done in 2 weeks or less.

2.1.1.3. Local File Inclusion and Remote File Inclusion:

Test for local file inclusion and remote file inclusion by checking 1. whether there are any upload pages in the web app, whether they are internal/ secret upload pages or public upload pages such as uploading profile pictures and so on and 2. What can you upload to those pages and where do the uploaded files go, some file extensions can be hidden in image files and potentially set up a backdoor.

The app should be able to test for File Inclusion attacks by checking what files are uploaded or linked to make sure they don't have any secret extensions that could potentially cause harm.

LFI and RFI are vulnerabilities of file checking in a website. Testing for LFI and RFI should be done in 2 weeks, possibly 3.

.

2.1.1.4. Providing possible solutions and adding priorities to vulnerabilities:

The app should come up with some potential mitigations to any possible vulnerabilities and how big the priority of those vulnerabilities is based on what damage or access the vulnerabilities could grant should they be exploited.

Progress could be tracked based on what vulnerabilities the app finds and what solutions are there for them.

The app only needs to record what vulnerabilities it found, their priority and possible solutions it found to those vulnerabilities.

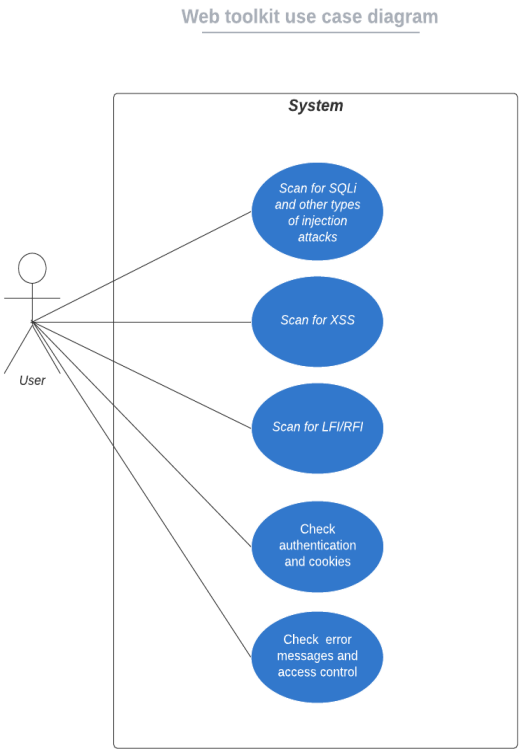
The recording of vulnerability priorities and possible solutions will help the user to improve their web app and server(s). The feature should be able to be done in 1 to 2 weeks.

2.1.2. Non functional requirements

2.1.2.1. Should be fast. Ideally the scanner should not take more than 1 minute per page as the scanning tests are fairly simple.

2.1.2.2. Should give as few false positives as possible. A false positive is the web scanner toolkit reporting a vulnerability on the target site or web app that is not actually there or is not actually a vulnerability. Manual checking of said vulnerabilities helps mitigate false positives.

2.1.3. Use Case Diagram



There are currently 4 scanner modules that check 3 vulnerabilities.

SQLi scanning, XSS(Cross Site Scripting), and File Inclusion (Local and Remote)

Use case descriptions of scanner module function:

Name: Vulnerability scanning

Description: The use case describes the flow of how the scanning modules check the target site

Flow of Events:

Activation: The use case starts when the user submits a website address

Basic Flow:

1. The use case starts when the user submits a website address
2. User opens the app, inputs the address they wish to test

3. User presses the enter key
4. The application proceeds to run through the scanning modules and output the result on the screen.
5. Depending on the vulnerabilities tested and found the amount of outputted information will vary
6. In the case the scanner has found vulnerabilities present, it will report what it found and where in the page it found it.
7. In the case the scanner has not found any, it will report that it has found no vulnerabilities and instruct to input a new address.
8. The application will then end

Alternate Flow:

1. User could input the target address through the GUI interface, rather than the command line app.
2. The application proceeds to run through the scanning modules and output the result on the report tab.
3. Depending on the vulnerabilities tested and found, the amount of outputted information will vary
4. In the case the scanner has found vulnerabilities present, it will report what it found and where in the page it found it.
5. In the case the scanner has not found any, it will report that it has found no vulnerabilities and instruct to input a new address.
6. The application will then end

Exceptional flow:

7. The tested website has no parameters which can be used
8. The tested website has no forms or input fields to test
9. The scanner app reports no vulnerabilities. In which case, manual scanning is required in order to confirm said lack of vulnerabilities.

Termination

The app ends if:

10. the scanning is completed successfully and the results are printed
11. if the app has found no vulnerabilities present
12. if the app cannot scan the website, due to missing parameters or forms

Special requirement: The user must have permission and/or access from the web site or app owner or maintainer to the site. For example they must either be the maintainer of the testing site, the owner or have permission from the people mentioned to perform testing.

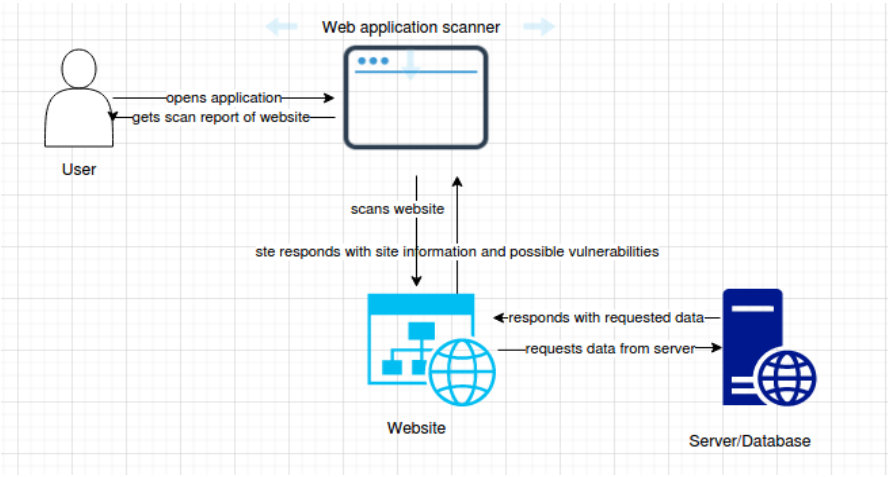
Or they must use the app on their own sites or test labs. The app may also be used in training labs or CTF sites where permitted.

Preconditions: The user must have internet access to be able to connect to the target website so that the app may perform its tests.

2.2. Design & Architecture

The app will just be by itself, it may have a database or a file where vulnerabilities found are stored or I may store them in a list. It will basically scan the web app which will request data from the server as a website would, the data will be sent back to the site and the site will respond with whatever it got and whether any vulnerabilities were detected. Once the scan is done the web scanner will come back to the user with a list of any vulnerabilities it found.

Architecture diagram:



2.3. Implementation

The application is written in Python and has a number of scanning functions. It takes as input a website address, parses that address as well as the web page itself and sends that data, mostly any forms forms in the page to the each of the scanner functions.

Some of the scanner functions such as SQLi scanning take both the address and any forms detected and then fill out the forms with test data that is used to check for the vulnerability. It submits the filled out forms back to the site/web server and waits for the response.

Once the response returns the function looks over the response and any changes in the web source code and checks that against a list of known errors that typically appear during such tests.

If the website response matches one or more of those errors the function will return the vulnerability as found and move on to the next function.

For the File inclusion scanning the functions takes the website address as the input and parses it into parts to be used for url modification.

In the case of local file inclusion the function analyses the URL address and looks for any parameters it can use to append the file or directory it wants to access. It then rebuilds the URL address and submits it back to the server.

The function will then read the response and any changes in the website code. In a successful attempt the function will find the contents of the specific file accessed inside the code of the page. If it hasn't found signs of the file accessed it will return as not found.

In the case of remote file inclusion the function employs the set up of a very simple PHP server that is set up to hold 1 file containing an ordinary string. The purpose of this file is to be accessed and opened remotely by the target website being tested.

Note: Before using the PHP server, a Python HTTP server was used but the server did not access the test file or open it, causing the function to report no vulnerability.

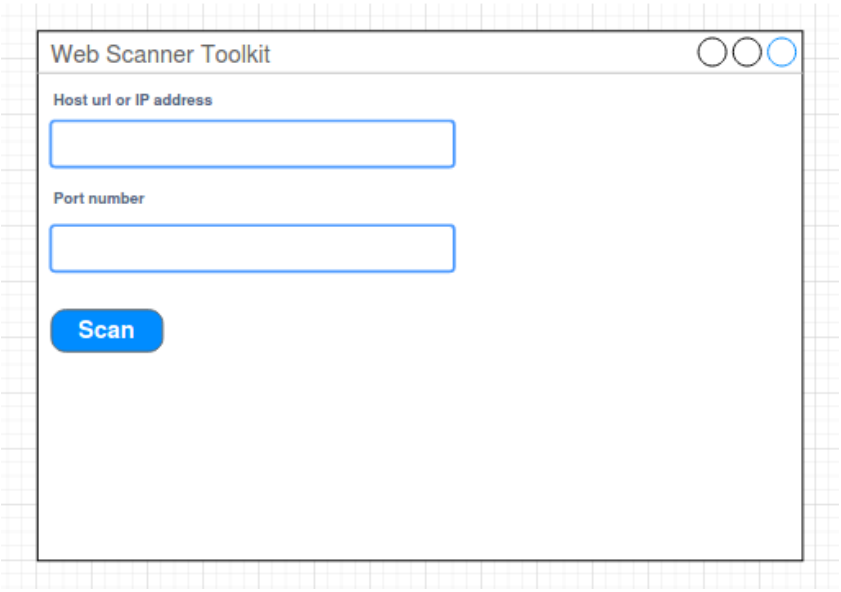
As usual the input of this function is the web address. The address is then parsed and analysed for parameters such as 'page=index.php' for example. The URL is modified to append the address of the Python server that contains the testing file. It then submits the new URL and looks over the response from the server.

In a successful attempt the page in question will have the contents of the testing file within the source code. This will have the function report the vulnerability as found.

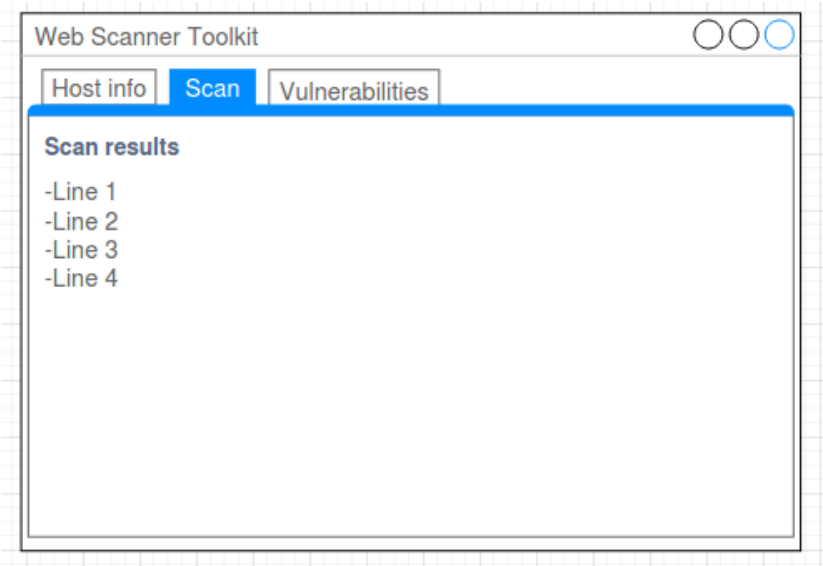
If there are no signs of the file being read, it will be reported as not found.

At the end the report will be shown in the vulnerabilities tab.

2.4. Graphical User Interface (GUI)



Draft wireframe of main screen. Here you input the website/webapp you want to test. The scan button will bring you to the scan page where you will see the scan results. Note this is a simple sample GUI because the project itself is primarily a command line application.



This is the scan page where you can see scan results, website info, any vulnerabilities found and possible mitigations.

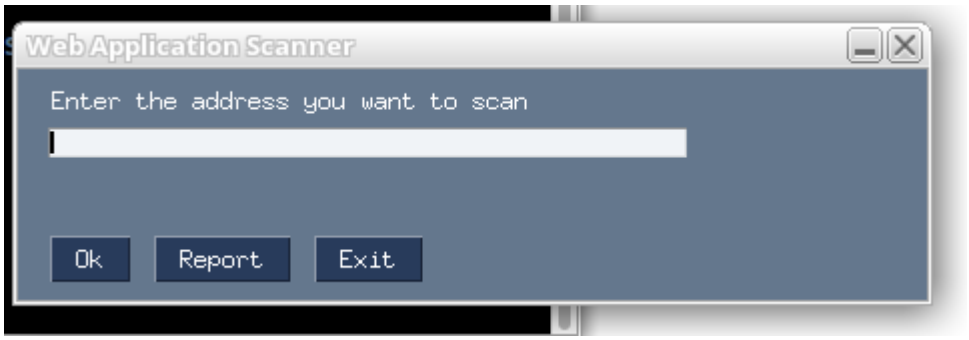
The second tab is for the scan results report where it would show what the toolkit found.

The third tab is for any vulnerabilities found in the web app with priority ratings and possible mitigations.

Implementation of a GUI for the actual app itself:

Here is the finalised version of the GUI:

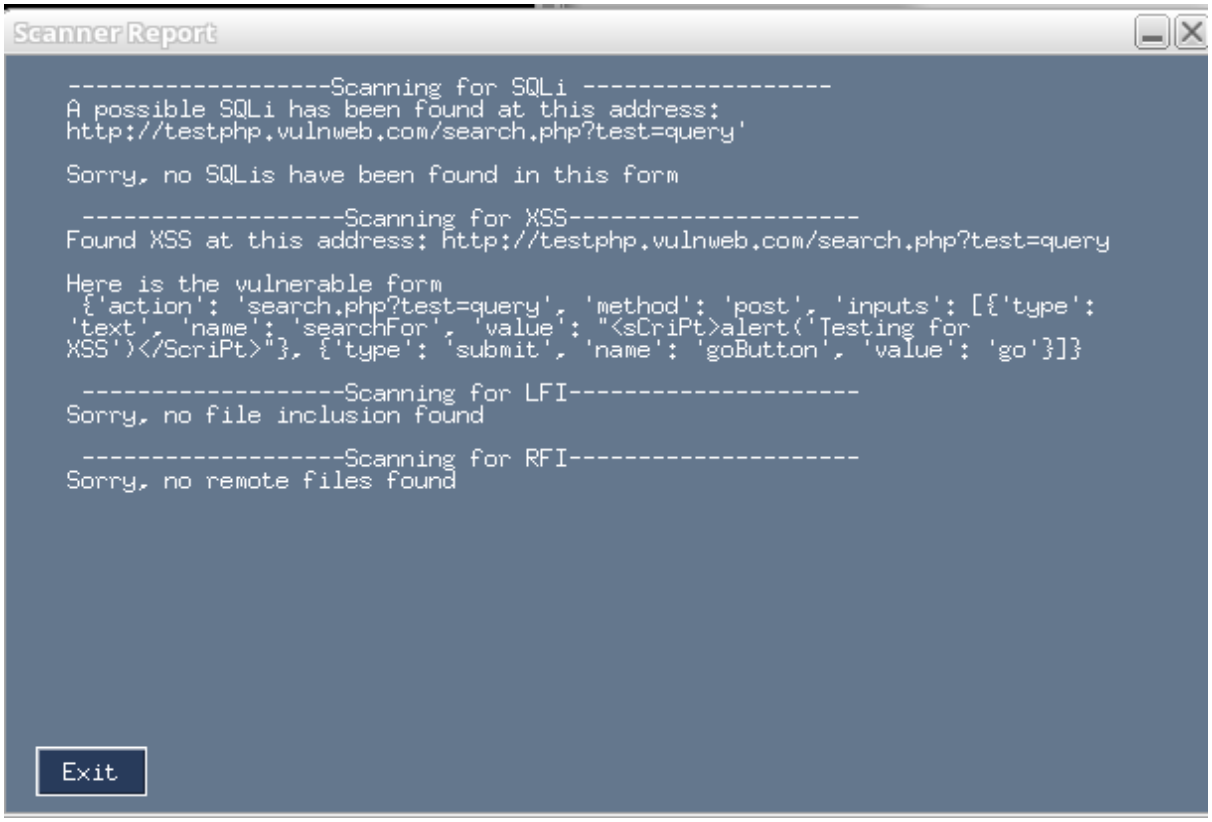
As the application only has 1 input, the web address the interface itself is very simple:



Clicking the Ok button sends the web address to the main scanner functions which will be printed out on the console.

Clicking the Report button brings up the report where the user can view the output from the functions. However do not click the report button without clicking the OK button and the Report button uses the output generated by the scanner functions. The functions need to be run in order to generate output. Note: There is a bug where if you put in an address and click Report without clicking Ok, it breaks the app. I haven't been able to fix that.

Here is the report window:



For the sake of the demonstration of the report window the address above has been used instead of the local DVWA server.

2.5. Testing

Glossary of test and system terms:

Metasploitable Virtual Machine: A virtual machine (VM) that is designed and used as a testing box for security and pentesting. It is insecure on purpose to allow for the testing of vulnerability scanners and other such security tools, as well as it is used as practice for pentesting training.

DVWA: Damn Vulnerable Web Application. Also for the use of security testing tools such as web scanners. It is as the name implies, a vulnerable web server set up. DVWA can be set up on its own or it can come as part of the Metasploitable VM.

SQLi: Short for SQL injection, a type of database attack, where input fields or addresses do not check the data submitted, leading to little or no input sanitisation. Therefore an attacker can craft a query to access back end databases and other such sensitive information.

LFI: Local File Inclusion, also known as directory traversal attack. It exposes a local file from the server hosting the website by manipulating the url to often access the root directory (or any other such directories).

It does that in the way that a regular user changes directory in a terminal, by using the "../" command at the end of an url. This means go back 1 directory, it is done repeatedly until one can access any one particular file such as the /etc/passwd file.

XSS: Cross site scripting another type of injection attack where the attacker uses typically JavaScript to bypass the input fields and display any amount of information on the web page in the form of an alert function.

RFI: Remote File Inclusion. Remote file inclusion can work in 2 ways.

1. If the target system has directory access or pulls files directly via say a file or page parameter such as ?file=index.php, an attacker can use that and substitute their own location from which to pull a file from.

If the attack is successful they can execute any files from the host system on the target system, such as having the target execute a reverse shell file from the host system.

The second way is through an upload form. This is generally only useful if the attacker can determine where the files that are submitted via the upload button go.

Test plans and information:

The scanner will check for the vulnerabilities as described and the results will be confirmed by manual scanning i.e manually performing the scan. This is to minimise false positive/false negatives.

Testing function for SQLi:

System tested: Local Metasploitable virtual machine DVWA server

Web page tested: http://server-host-address/dvwa/vulnerabilities/sqli

Expected output: On settings low and medium the SQLi function should report a vulnerability. On setting high, the function should report none as it has been secured with parameterised queries and prepared statements.

Output from the scanner application

```
------------------Scanning for SQLi ------------------
Scanning...
Sorry, no SQLis have been found at this url address
Scanning form(s)...
A possible SQLi has been found in this form:  http://192.168.59.128/dvwa/vulnerabilities/sqli/
Here is the vulnerable form:
{'action': '#', 'method': 'get', 'inputs': [{'type': 'text', 'name': 'id', 'value': ''}, {'type': 'submit', 'name': 'Submit', 'value': 'Submit'}]}

Process finished with exit code 0
```

This is the output when DVWA server is set to low security setting.

```
--------------------Scanning for SQLi ------------------
Scanning...
Sorry, no SQLis have been found at this url address
Scanning form(s)...
Sorry, no SQLis have been found in this form

Process finished with exit code 0
```

As expected the high setting returns no vulnerability.

Here is the code for the Python unit test for this function

```
returned_msg_success = "A possible SQLi has been found here: " + testing_url
returned_msg_fail = "Sorry, no SQLis have been found at this url address "
returned_form_msg_success = "A possible SQLi has been found in this form: " + testing_url
returned_form_msg_fail = "Sorry, no SQLis have been found in this form"


class SQLiTestCase(unittest.TestCase):
    def test_sqli_success(self):
        actual = pythonHalfScanner.scan_for_sqli(testing_url)

        expected = returned_msg_fail, returned_form_msg_success

        self.assertEqual(actual, expected)

    def test_sqli_form(self):
        actual = pythonHalfScanner.scan_for_sqli(testing_url)
        expected = returned_msg_fail, returned_form_msg_success
        self.assertEqual(actual, expected)
```

In order for the Python unit test to work I had to place the output of the function into 2 msg variables that would be returned.

```
Scanning...
Scanning form(s)...


Ran 1 test in 0.034s

OK

Process finished with exit code 0
```

Hence why there is no output shown on the screen during the unit test. If the output is printed instead of returned, even if it is in a variable, it won't register in the test. However printing the output in addition to returning the variables does work. This by the way is the low setting, which it should pass.

Running the test with the security setting set to high, should fail

```
Launching unittests with arguments python -m unittest testing_SQLi function.SQLiTestCase.test_sqli_success

Scanning...
Sorry, no SQLis have been found at this url address
Scanning form(s)...


Ran 1 test in 0.037s

FAILED (failures=1)
Sorry, no SQLis have been found in this form


Sorry, no SQLis have been found at this url address  Sorry, no SQLis have been found in this form != Sorry,

Expected :Sorry, no SQLis have been found at this url address  A possible SQLi has been found in this form:
Actual   :Sorry, no SQLis have been found at this url address  Sorry, no SQLis have been found in this form
```

Testing function for XSS:

System tested: Local Metasploitable virtual machine DVWA server

Web page tested: http://server-host-address/dvwa/vulnerabilities/xss_r/

Input: The web page tested above

Expected output: On settings low and medium the function should report the vulnerability as found. It should also report on what form it found the vulnerability on. On setting high, the vulnerability should not be detected as it would be patched.

Output result from the scanner application

```
------------------Scanning for XSS---------------------
Scanning form(s)...
Found XSS at this address: http://192.168.59.128/dvwa/vulnerabilities/xss_r/
Here is the vulnerable form:
{'action': '#',
 'inputs': [{'name': 'name',
             'type': 'text',
             'value': "<sCriPt>alert('xss')</ScriPt>"},
            {'name': None, 'type': 'submit', 'value': 'Submit'}],
 'method': 'get'}

Process finished with exit code 0
```

Server setting to low:

This is the output for when the scanner detects the cross site scripting vulnerability as shown by the testing security server setting set to low. The medium setting returns the same output

```
-------------------Scanning for XSS---------------------
Scanning form(s)...
Sorry, no XSS was found

Process finished with exit code 0
```

Server setting to high

This is the output if the app does not detect the vulnerability with the testing server security setting set to high.

The code for testing this function

```
returned_form_msg_success = f"Found XSS at this address: {testing_url}"
returned_form_msg_fail = "Sorry, no SQLis have been found in this form"
class XSSTestCase(unittest.TestCase):
    def test_XSS(self):
        actual = pythonHalfScanner.scan_for_xss(testing_url)
        expected = returned_form_msg_success
        self.assertEqual(actual, expected)


if __name__ == '__main__':
    unittest.main()
```

The test should pass if the server setting is set to low

```
Here is the vulnerable form:
{'action': '#',
 'inputs': [{'name': 'name',
             'type': 'text',
             'value': "<sCriPt>alert('xss')</ScriPt>"},
            {'name': None, 'type': 'submit', 'value': 'Submit'}],
 'method': 'get'}


Ran 1 test in 0.034s

OK

Process finished with exit code 0
```

It should fail if the setting is set to high as the high setting is not susceptible to XSS

```
Scanning form(s)...


Ran 1 test in 0.035s

FAILED (failures=1)


Sorry, no XSS was found != Found XSS at this address: http://192.168.59.128/dvwa/vulnerabilities/xss_r/

Expected :Found XSS at this address: http://192.168.59.128/dvwa/vulnerabilities/xss_r/
Actual   :Sorry, no XSS was found
<Click to see difference>
```

Testing function for LFI:

System tested: Local Metasploitable virtual machine DVWA server

Web page tested: http://server-host-address/dvwa/vulnerabilities/fi/

Input: The web page tested above

Expected output: On settings low and medium the function should report the vulnerability as found if the function can open the specified local server file and read it, by modifying the URL address to expose the file.

Output from the scanner application

```
--------------------Scanning for LFI----------------------
Scripts in the path: /dvwa/vulnerabilities/fi/
The query is: page=index.php
1 parameter is present. Searching for extra parameters...
There are no more parameters. Moving on to scanning...
It worked!
Address: http://192.168.59.128/dvwa/vulnerabilities/fi/?page=/etc/passwd

Process finished with exit code 0
```

This is the output for when the scanner detects vulnerability as shown by the testing security server setting set to low. The medium setting returns the same output. Here the parameter is found and then the function changes the url to find the specific local file such as /etc/passwd.

```
--------------------Scanning for LFI----------------------
Scripts in the path: /dvwa/vulnerabilities/fi/
The query is: page=index.php
1 parameter is present. Searching for extra parameters...
There are no more parameters. Moving on to scanning...
Sorry, no file inclusion found

Process finished with exit code 0
```

Here the scanner cannot find the vulnerability despite the parameters being available. This is because the testing server security setting is set to high.

The code for Python unit test for testing this function:

```
returned_msg_success = returned_msg = "It worked! The file was found!"
returned_msg_fail = "Sorry, no file inclusion found"

class LFITestCase(unittest.TestCase):
    def test_local_file_inclusion(self):
        actual = pythonHalfScanner.scan_for_lfi(testing_url)
        expected = returned_msg_success
        self.assertEqual(expected, actual)


if __name__ == '__main__':
    unittest.main()
```

With the low setting in the DVWA server the test should pass as the low settings is susceptible to file inclusion and the access of local files.

```
Launching unittests with arguments python -m unittest testingLFI_function.MyTestCase.test_


Process finished with exit code 0
Scripts in the path: /dvwa/vulnerabilities/fi/
The query is: page=index.php
1 parameter is present. Searching for extra parameters...
There are no more parameters. Moving on to scanning...


Ran 1 test in 0.402s

OK
It worked!
```

And fail when the setting is set to high. The high security setting patches the file inclusion vulnerability so that it cannot access local server files.

```
Launching unittests with arguments python -m unittest testingLFI_function.LFITestCase.

Scripts in the path: /dvwa/vulnerabilities/fi/
The query is: page=index.php
1 parameter is present. Searching for extra parameters...
There are no more parameters. Moving on to scanning...
Sorry, no file inclusion found


Sorry, no file inclusion found != It worked! The file was found!

Expected :It worked! The file was found!
Actual   :Sorry, no file inclusion found
<Click to see difference>
```

Testing function for RFI:

System tested: Local Metasploitable virtual machine DVWA server

Web page tested: http://server-host-address/dvwa/vulnerabilities/fi/

Input: The web page tested above

Expected output: The scanner will find the vulnerability if the PHP file_include function is enabled. This will allow it to include and open remote files. The vulnerability will not be detected if the function is disabled or the setting is set to high.

Output from the scanner application

```
-------------------Scanning for RFI---------------------
Target address: http://192.168.59.128/dvwa/vulnerabilities/fi/?page=index.php
Address has a parameter: page=index.php
Sorry, no remote files found

Process finished with exit code 0
```

This is the output given by the app if there is not remote file inclusion access. It means the target site can't access the remote file. For the scanning test to work i set up a separate php server and added the server address to the script. Note that setting the php server address to 'localhost' does not work, it has to be an ip address.

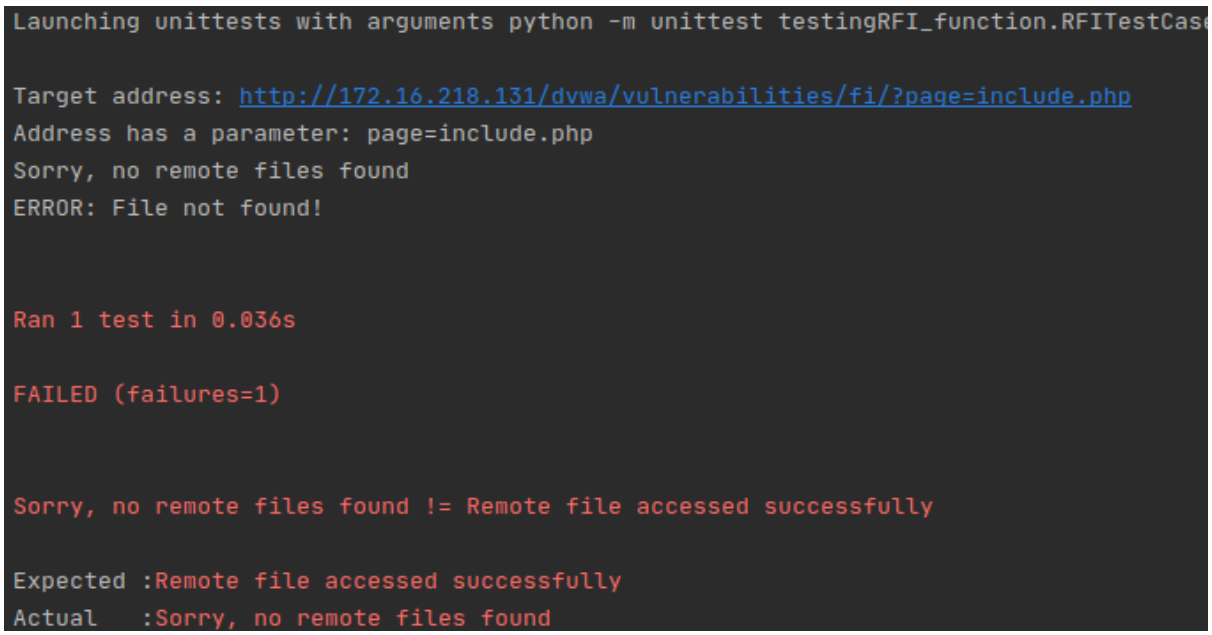
Here is the code for the unit test for this function

```
returned_msg_success = "Remote file accessed successfully"
returned_msg_fail = "Sorry, no remote files found"


class RFITestCase(unittest.TestCase):
    def test_remote_file_inclusion(self):
        actual = pythonHalfScanner.scan_for_rfi(testing_url)
        expected = returned_msg_success
        self.assertEqual(expected, actual)


if __name__ == '__main__':
    unittest.main()
```

If the server security settings are set to low and the php include options are enabled the test should pass. The IP is different because I changed the networking settings in VMWare which gave it a new address.

```
Launching unittests with arguments python -m unittest testingRFI_function.RFITestCa

Target address: http://172.16.218.131/dvwa/vulnerabilities/fi/?page=include.php
Address has a parameter: page=include.php
Remote file accessed successfully


Ran 1 test in 0.031s

OK

Process finished with exit code 0
```

This is the output of the test function if it fails, i.e does not find or report the vulnerability.



2.6. Evaluation

The project was tested as seen above using the VM test server.

The functionality of the project and functions was evaluated by changing the security settings to check what the output from the application is during different states of vulnerability. If the server was secured or patched the application should return output accordingly, not report vulnerabilities where there are none.

Conversely the server should not report absence of vulnerabilities where there could possibly be one or more. Hence manual testing of the vulnerabilities is done to confirm the output result.

3.0 Conclusions

The strengths of the project are that it is relatively fast and performed well on the tests, with very few errors. It can scan for all functions at once and output what it found in a clear way. It also has few false positives as the scans can be both performed manually and by using the application.The application is easy to use as there is only 1 input, the website address.

The limitations of the project is currently it can only do one web page at a time unless the list of websites to test is coded into the application to have the functions run multiple times and it does not have the complexity or volume as that of more popular tools or commercial based tools. It therefore can't be used in a commercial environment or in large websites with potentially hundreds of pages.

It also does not check for a wide variety of vulnerabilities, just the most common ones present as outlined in the OWASP Top 10.

4.0 Further Development or Research

With more research and development the project could have more features and requirements not implemented in this iteration of the project. Such as:

- a working web spider than can scan all the pages of a web app and store for the use of the scanner functions
- extra input parameters so that the functions can scan multiple inputs at one time.
- more complex scanning functions that can scan for a wider variety of vulnerabilities in comparison to other popular solutions and this iteration
- external community testing and suggestions
- It could also offer more effective ways and solutions for minimising the vulnerabilities found.

5.0 References

PYPL PopularitY of Programming Language index. n.d. *PYPL Popularity Of Programming Language*. [online] Available at: <https://pypl.github.io/PYPL.html> [Accessed 20 December 2020].

Kumar, A., 2019. *Top Programming Languages Of 2020 According To Stats & Surveys*. [online] Coding Infinite. Available at: <https://codinginfinite.com/top-programming-languages-2020-stats-surveys/> [Accessed 20 December 2020].

Muscat, I., 2019. *What Is Local File Inclusion (LFI)?*. [online] Acunetix. Available at: <https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/> [Accessed 20 December 2020].

Bekerman, D. and Yerushalmi, S., 2020. *The State Of Vulnerabilities In 2019 | Imperva*. [online] Imperva. Available at: <https://www.imperva.com/blog/the-state-of-vulnerabilities-in-2019/> [Accessed 20 December 2020].

Trustwave. 2018. *5 Most Common Web Application Attacks (And 3 Security Recommendations) - MSSP Alert*. [online] Available at: <https://www.msspalert.com/cybersecurity-breaches-and-attacks/5-most-common-web-application-attacks/> [Accessed 20 December 2020].

Erickson, J., 2008. *Hacking: The Art Of Exploitation*. 2nd ed. No Starch Press, p.3.

6.0 Appendices

This section should contain information that is supplementary to the main body of the report.

1.1. Project Plan/Project proposal

National College of Ireland

Project Proposal

< Toolkit for Web Security>

[Improving Web Application Security with a toolkit]

<5 November 2020>

<BSc in Computing(BSHC)>

<Cyber Security>

<2020/2021>

<Ioana Avram>

<17448556>

<x17448556@student.ncirl.ie>

Contents

1.1.1. Objectives

To make an application for testing websites and web apps for any known vulnerabilities like a vulnerability scanner toolkit. And to implement update checking and patch management for these servers and software so that they can be up to date.

1.1.2. Background

As more companies and people use the internet, more websites are made, apps, servers and so on, more data is stored in the cloud. The current world pandemic has seen a large increase in attacks on such servers, and there aren't always enough people to check on these sites. One needs to keep up with updates, monitor networks and so on.

Therefore one could always use more apps to check the security of these sites to make sure they are updated and not exposed to any such vulnerabilities, as much as reasonably possible. Imperva states the most commonly found web vulnerabilities in 2019 continues to be Injection type vulnerabilities. They record a total of 28% of vulnerabilities were injection-based with an increase of 21% compared to last year.(Bekerman, 2020)

MSSP Alerts comments in 2017 40% of the total web attacks were XSS(Cross-Site Scripting) and 28% of attacks were SQL Injection based.(Trustwave, 2018)

I also wanted to make an app for web security because I wanted to make something in Python after struggling to maintain consistency in learning Python regularly.

The PYPL (PopularitY of Programming Language) site ranks Python as the most popular language as of December 2020 compared to a year ago with a growth of 18% in the last 5 years and a share of 30.34% according to Google trends statistics.(PYPL PopularitY of Programming Language, n.d.)

CodingInfinite also ranks Python as one of the most popular languages according to statistics from Stackoverflow with a share of 41.7% over the last 5 years and it comments that Python has continued to grow while other languages such as Java declined.(Kumar, 2018)

And being in the security specialisation meant the project had to have something regarding computer security as such. Did a tiny bit of research and talked to some people about what I could make in that language and came across that I should make a web security app.

Now I'm sure there are many out there already, but I read in a security book that a good security person makes their own tools instead of relying on tools made by other people, as said in this comment "*Crackers were considered much less talented than the elite hackers as they simply made use of other tools and scripts without understanding how they worked*" from the book Hacking: the art of exploitation.(Erickson, 2008)

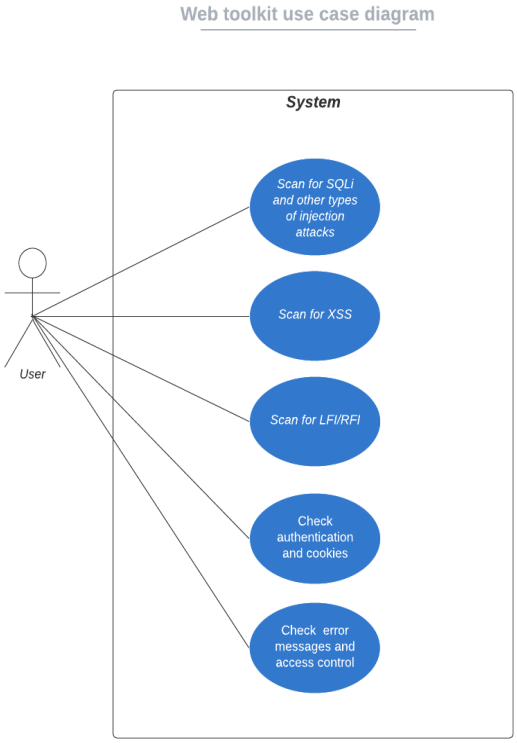
In the end, the project will demonstrate Python skill and I'll have something to show for it when asked 'what have you made in x y or z language'.

1.1.3. Technical Approach

The technical approach could be as follows:

I will do more research on what other apps have and what actually would go into one.[conduct literature review of available tools or research papers, looking for improvements, future work, finding gaps]

Based on that I will formulate some requirements and make notes, taking into account the current attack landscape, what are web apps and sites currently struggling with in terms of security.



Based on my research of web app scanners and recent web vulnerability statistics and the like, this is a diagram of possible features I could have in the web toolkit. SQL Injections and injection attacks in general, continue to be one of the most common vulnerabilities besides things like XSS (Cross-Site scripting). There are of course other vulnerabilities such as LFI/RFI (Local file inclusion and remote file inclusion) as well as unsecured cookies and broken authentication.

The app will then be coded in Python supplemented by Python tutorials and courses. I will likely use either Pycharm or Visual studio code for the programming environment.

I will then proceed to test the app using virtual machines and dummy servers that I will deploy myself to stay within devices and resources that I can control and have access/permission to.

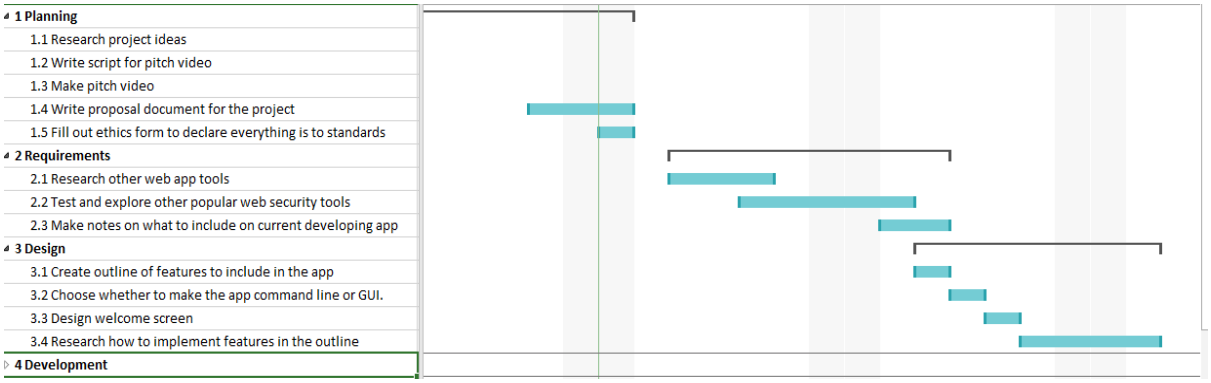
Possibly towards the end of the project term, I may employ the usage of publicly available test sites that are designed for the purpose of security testing. But that's not a guarantee.

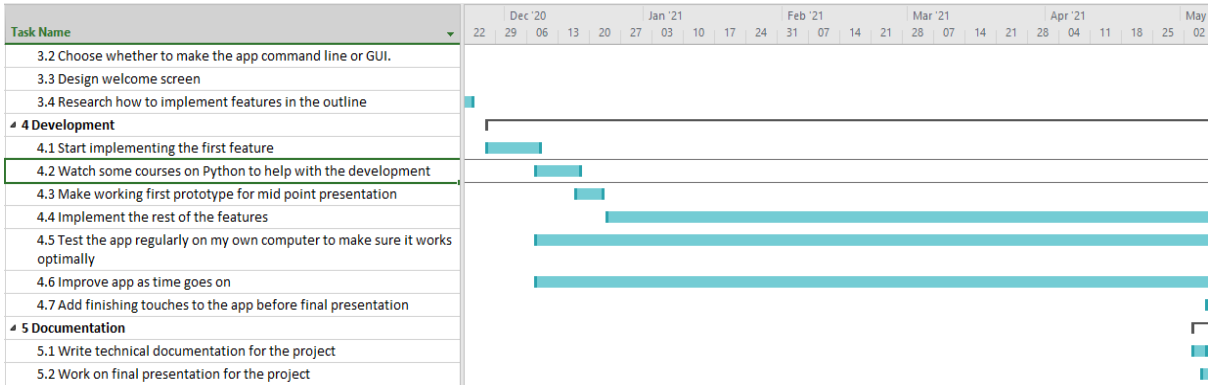
It is most likely the case that I will be using only my resources for better assurance that should anything go wrong I can more easily fix it than it being someone else's servers and having to be responsible for something I do not own.

1.1.4. Special Resources Required

Resources for learning Python, laptop, maybe other testing equipment such as virtual machines, possibly other devices I own.

1.1.5. Project Plan





1.1.6. Technical Details

Currently, the application will be implemented in Python, with the possibility of using additional third-party libraries to enhance the features of the application. Will probably use either Pycharm or Visual studio code for the environments and virtual machines for testing.

1.1.7. Evaluation

Testing will be done using virtual machines and servers deployed on my end most likely because it gives a greater degree of assurance in the case something goes wrong. Because

they are on my end it makes it easier to fix what went wrong rather than having to go through additional hassle because I do not own whatever server went wrong.

The purpose of these servers is to be used to test web server vulnerability.

1.1.8. References

PYPL PopularitY of Programming Language index. n.d. *PYPL Popularity Of Programming Language*. [online] Available at: <https://pypl.github.io/PYPL.html> [Accessed 20 December 2020].

Kumar, A., 2019. *Top Programming Languages Of 2020 According To Stats & Surveys*. [online] Coding Infinite. Available at: <https://codinginfinite.com/top-programming-languages-2020-stats-surveys/> [Accessed 20 December 2020].

Muscat, I., 2019. *What Is Local File Inclusion (LFI)?*. [online] Acunetix. Available at: <https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/> [Accessed 20 December 2020].

Bekerman, D. and Yerushalmi, S., 2020. *The State Of Vulnerabilities In 2019 | Imperva*. [online] Imperva. Available at: <https://www.imperva.com/blog/the-state-of-vulnerabilities-in-2019/> [Accessed 20 December 2020].

Trustwave. 2018. *5 Most Common Web Application Attacks (And 3 Security Recommendations) - MSSP Alert*. [online] Available at: <https://www.msspalert.com/cybersecurity-breaches-and-attacks/5-most-common-web-application-attacks/> [Accessed 20 December 2020].

Erickson, J., 2008. *Hacking: The Art Of Exploitation*. 2nd ed. No Starch Press, p.3.

Reflective Journals:

October:

Reflective Journal for October

Ioana Avram
x17448556
Honours Bachelor in Computing --Security branch

What? --What happened

- Researched ideas for project, had no ideas
- Talked to people about what i can possibly do
- Had classes
- Did more research
- Decided to do the project in Python, only remained to decide what project
- Made a list on how I can use Python to make something security based.
- Decided as the first idea that I will try to make a Web security toolkit, for websites and web-apps and the like.
- Made the video pitch but had to write about it because video quality and my voice aren't very clear. I tend to talk very quietly.
- I wrote about it to my assigned supervisor and they said they will come back with feedback.
- Currently awaiting project feedback while working on other assignments.

So, what? --More context

- Still don't quite know what to do for the project yet, just something in Python.
- But I do have some ideas I can bounce around until I find something that is accepted.
- Also have a lot of other assignments plus career classes and things.

Now, what? --What to do next

- More research on the project.
- Waiting for feedback and results on the project topic.
- Will continue to work on all other assignments while the project topic gets reviewed.

November Reflective journal:

Reflection Journal for November
Ioana Avram
x17448556

- Mainly worked on other assignments
- Had 1 or 2 meetings with supervisor but had to nothing to report to them
- Had classes and a presentation
- Nothing much still in terms of November progress
- But it's coming towards the end of assignments so I might have time for research and requirements, maybe even getting started on the code for the project
- Will have more progress in the December report

Haven't done a January or December report as I was busy with coursework and submitting both the mid term project and the other such assignments and TABAs

February Report:

What happened?

- Attended class
- Did a lot of tryhackme stuff, part of the pentesting module
- Knit
- Made tarot cards
- Did a bit of research for project
- Currently trying to learn perl so i can study the Nikto code to see if i can gather anything useful for my project from it.
- Made a small project plan for the remaining weeks on coding my project
- Got results from last semester modules and feedback on my mid project presentation
- Attended the weekly interview skills workshops i signed up for

How did it affect me or the people around?

Not much to be quite honest. But i do have more idea on how to go about the project, if only a little bit. I still haven't learned much Python besides the basics i already knew so i have to continue watching the Python courses i have. I also need to work on the big project document, to add more to the requirements and testing sections.

What will i do in the near future?

- Start on the first part of the project
- Study more perl and python
- Take more notes on the tryhackme boxes
- Work on all other projects

March Report:

What i did

1. Had class
2. Worked on assignments
3. Had meetings with supervisor
4. Did research on the project, mostly looking up tutorials on how to code certain features
5. Asked feedback on the project and the small bit of research i did
6. Installed and set up my testing environment which consisted of a testing virtual machine called Metasploitable
7. Tested and implemented the first and second feature separately

April Report:

What I did:

1. Continued to work on the project and research code and other features regarding the programming language Python
2. Mostly was busy with CAs in other classes so that took most of my time
3. Was out of commission for a week and a half because I had gotten sick while working on a assignment for another module
4. Also worked a lot of the the terminal assignments (the TABAs) so that cut my project time even more
5. Attended the last few classes before end of term with some supervisor meetings but mostly just progress check ins
6. Combined the first and second feature into one and started working on the last two features

1.3. Other materials used

Any other reference material used in the project for example evaluation surveys etc.

I used these tutorials and sample sites to help in building the project:

thepythoncode.com. 2020. *How to Build a XSS Vulnerability Scanner in Python - Python Code*. [online] Available at: <https://www.thepythoncode.com/article/make-a-xss-vulnerability-scanner-in-python> [Accessed 4 March 2021].

thepythoncode.com. 2021. *How to Build a SQL Injection Scanner in Python - Python Code*. [online] Available at: <https://www.thepythoncode.com/article/sql-injection-vulnerability-detector-in-python> [Accessed 4 March 2021].

GitHub. 2017. *sUbc0ol/LFI-scanner*. [online] Available at: <https://github.com/sUbc0ol/LFI-scanner/blob/master/lfiscan.py> [Accessed 5 May 2021].

Offensive Security.com. 2021. [online] Available at: <https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/> [Accessed 5 May 2021].

Imperva. 2020. *Remote File Inclusion (RFI) - Detecting the Undetectable | Imperva*. [online] Available at: <https://www.imperva.com/blog/remote-file-inclusion-rfi-detecting-the-undetectable/> [Accessed 8 May 2021].

Python, R., 2020. *PySimpleGUI: The Simple Way to Create a GUI With Python – Real Python*. [online] Realpython.com. Available at: <https://realpython.com/pysimplegui-python/> [Accessed 20 December 2020].

Opensource.com. 2019. *5 open source Python GUI frameworks*. [online] Available at: <https://opensource.com/resources/python/gui-frameworks> [Accessed 22 December 2020].

I used the Metasploitable VM from this link for the testing of the project

Rapid7. n.d. *Download Metasploitable - Intentionally Vulnerable Machine | Rapid7*. [online] Available at: <https://information.rapid7.com/download-metasploitable-2017.html> [Accessed 19 March 2021].