Project Portfolio

Week 1

-Sorting which project team im in

Week 2

- Re-Learning to use ZAP

We had slightly touched on using ZAP during the Security paper which I had taken last semester, though I had forgotten simple set up protocols so had Faisal send me an old lab from the security paper which helped me do so. I installed Kali Linux using Oracle VM, James showed me how to do this as I got slightly stuck on a couple of steps. Once Kali Linux was installed, Faisal suggested that I use pen testing to practise my security vulnerability scanning skills using ZAP. I used the following link to help me get the pen testing server up and running using oracle VM. <https://www.explainhownow.com/2019/learn-penetration-testing/>.

- Installing Kali Linux

- Using a browser as a proxy to automate zap checks

-Downloading Pentester image to use pentesting examples

Week 3

Getting the pen testing server running took me longer than it should have, I ended up using the first lesson of week 3 to figure this out. In the end the solution was simple I just had to download the pen testing image from the website and use it to run the server I created on Oracle VM. Once on the server I use the ipconfig command to check for I.P and use the I.P as the link in any browser to connect to the server. Working through the pen testing examples was interesting and refreshed my memory especially with the SQL injections. I realised that ZAP was not 100% effective in scanning for vulnerabilities for the pen testing examples, this tells me that having your own knowledge or running extra vulnerability scanners is important to have higher efficiency in finding and fixing vulnerabilities. Faisal suggested that I use a browser to act as a proxy so I could automate my vulnerability scans for any webpages I visited. The process in doing this seemed relatively simple while getting help from Faisal and using tutorials to help me do so. Although after following all of the steps I failed to get feedback from ZAP using the browser proxy. This is something I will have to research further.

-Getting stuck using a browser as a proxy

-Pentesting using Zap

Week 4

I worked through most of the pen testing examples and managed to find vulnerabilities for most. I heard about another tool called Nessus which is mostly used for networking vulnerability scanning but has modules to scan web applications also. I decided to give it a go and installed this on my Kali Linux machine so I could compare vulnerability results to ZAP. Nessus was relatively straight forward to set up using a tutorial along the way. The initial scan I did took much longer than a scan on ZAP and produced much more results, which were mostly network vulnerabilities. During class Corey (2nd semester project student) guided me through basic gitlab services which students or lecturers may ask me. The documentation the various different protocols was good so I felt I could learn how to do this if need be. We concluded the session by having a quick meeting with the group to explain what we were all doing and to ask any questions if needed.

-Learnt some basic about gitlab

-Add users

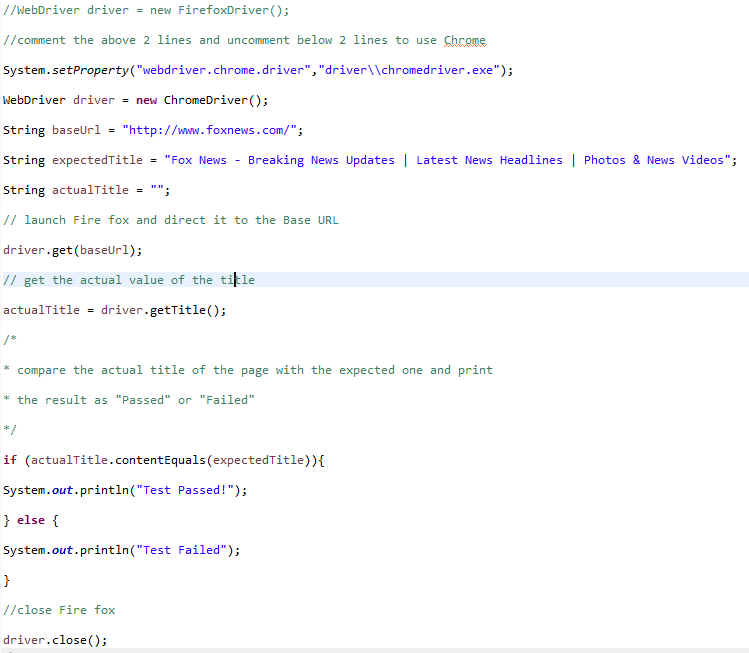
-Create new projects/repositories

-Lesson 2 - Meeting with Faisal- Sort out what direction we are all going in

Week 5

We had a meeting which we were all to give a 3minute explanation on what tasks we were all working on. I found it hard to explain the different tasks I was working on especially when it came to working with a tool called Selenium. I have been watching tutorials on how to set up Selenium but to no avail as of yet. I will continue to progress on Selenium until I set this up because Faisal told me this was an important tool and could be important in future job roles. At this point I have installed all the necessary tools for selenium and transported the .jdk files to a working project on Java. I now need the correct code to create the correct class to run a web application. I read on a tutorial that having knowledge on Java would help progress with Selenium faster, the last time I used java was a year ago during programming 3. So with the help of James I refreshed my memory on some basic java coding problems which I hope will be useful.

I found a test example script which I could run using Selenium which is shown below.



Running the code will automatically open and run http://foxnews.com/

My problem now is trying to run ZAP to automatically scan the web applications I am running through Selenium. I have found different code examples to do so but they have been using slightly different tools than I. This will be a future task I will fix in hopes I can run a series of automated scans using scripts.

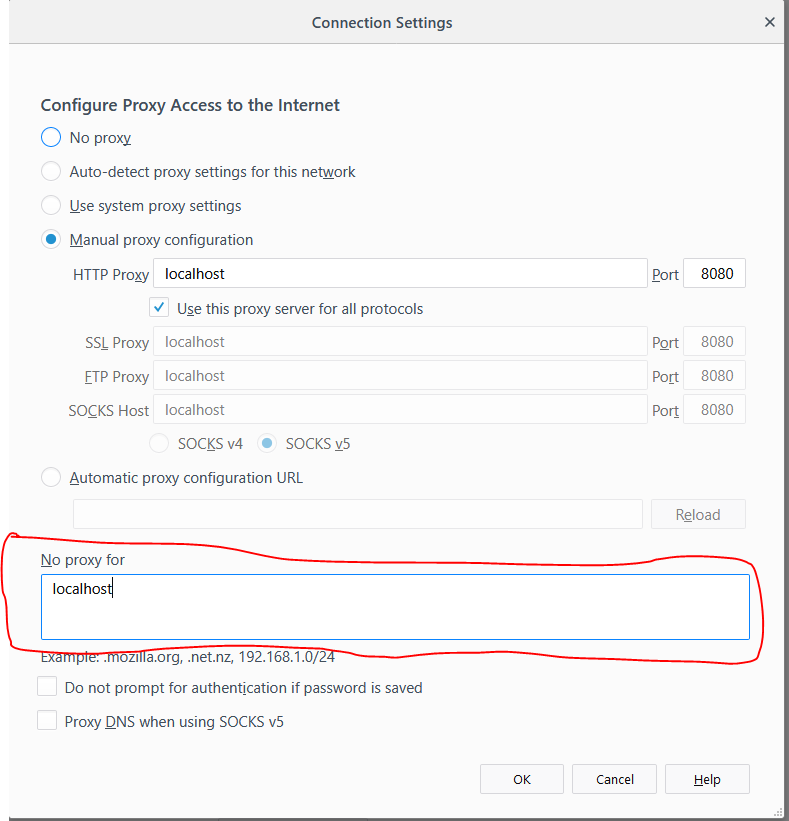
https://www.guru99.com/installing-selenium-webdriver.html

-Lesson 1 meeting with Faisal – 3min talk on what were all doing.

-Trying to learn Selenium

Week 6

On Monday, I had a talk with Faisal and he advised me that I should go back and make sure I can set the browser to work as a proxy and have ZAP run automated scans on any web applications I visit. I could not seem to fix this problem earlier on in the semester however Faisal sent me this tutorial which helped me realise where I went wrong. <https://www.youtube.com/watch?v=gk-AqjofzVU>.



After watching this I realised I had made a small error in configuring my proxy. There was a default configuration that was setting localhost as a HTTP proxy to not work. I simply had to delete text from a textbox to fix this problem. I now have ZAP running automated scans on any web application I visit through Firefox.

<https://www.youtube.com/watch?v=CFzgKfnmG-Q>

<https://www.youtube.com/watch?v=0qotVMX-J5s>

Week 7

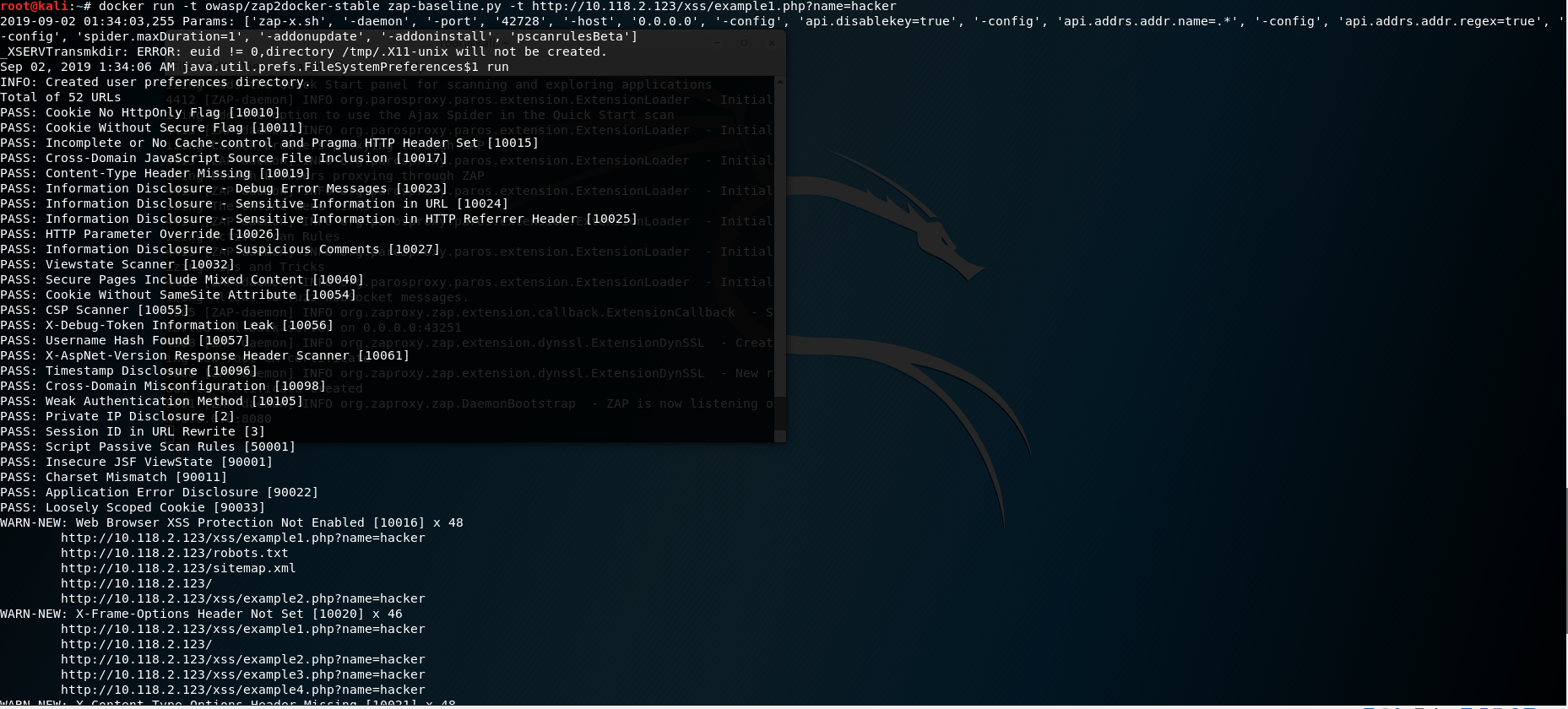
Running zap baseline scans provided from this link.

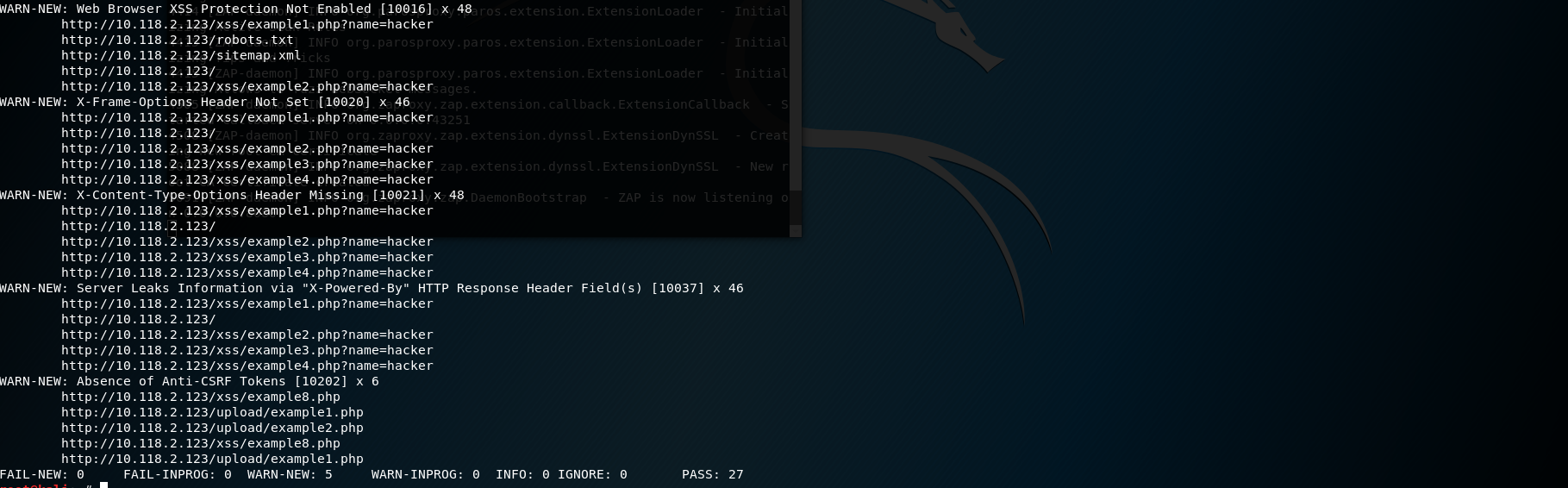
<https://blog.mozilla.org/security/2017/01/25/setting-a-baseline-for-web-security-controls/>

Faisal sent me some links which provided information on performing baseline scripts which will scan a web application. These scans were relatively easy to run, I ran them through Kali Linux against an example test web application. Links below were performed. The weekly script is updated weekly and the stable scan is like a baseline scan.

* <https://hub.docker.com/r/owasp/zap2docker-stable/>
* <https://hub.docker.com/r/owasp/zap2docker-weekly/>

You don’t even need to have ZAP installed to perform these baseline scans as the zap baseline script uses Docker and uses the 2 ZAP Docker images. The weekly baseline scan has more options available – running the script with the –h flag provides these extra options.





I think the baseline scans are an excellent way of performing quick effective scans against one or multiple sites.

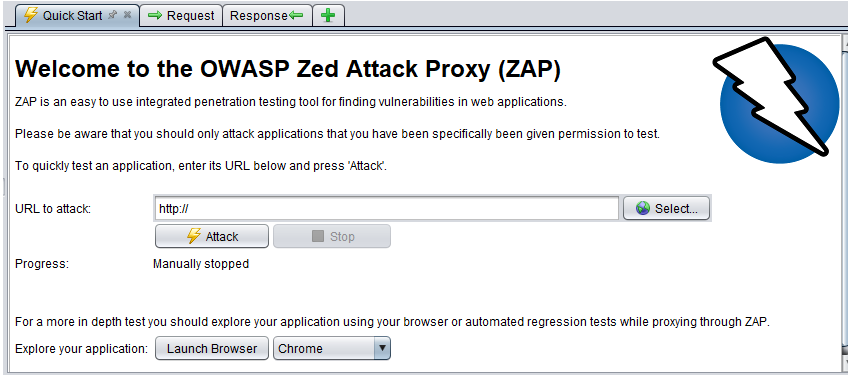
I watched through the links below where a developer for Mozilla talks through these scans and explains in depth how they work.

<https://www.youtube.com/watch?v=o_JZRgQMF4Q>

<https://www.youtube.com/watch?v=_MmDWenz-6U>

I was curious what the differences were with ZAP’s active and passive scan options. I’m aware on the differences between active and passive scanning but I wasn’t sure what type of active scans ZAP was actually doing. Through researching I found that ZAP’s active scanner was focused on finding website vulnerabilities such as SQL injection and XSS (cross-site scripting). It does this by making malicious requests which the passive scanner does not do. It is advised that we only use the active scanner on sites that we own.

Active scanning is run through the Quick Start tab. I did not initially realise this, so I have to be more careful on the sites that I am running scans on in future. I’ve already made the mistake of running a scan against the Otago polytechnic website which was not a very smart idea considering they have had an attack on their website recently.



Passive scans run on default with ZAP, so if needed you have to disable them. Zap will run the enable passive scan rules against all URLs that are either proxies through ZAP or visited by either of the spiders.

Week 8

This week have been asked by Faisal to write and produce a document which I can share to the DevOps group next Friday. The document will show what I’ve learnt and been working on over the past 8 weeks, this will give me an opportunity to learn more and to provide my group with more knowledge on what I’m doing. During this talk Faisal asked which direction I wanted to head in the future. I’ve decided I wanted to progress further with where I got with Selenium using ZAP, with my manual testing skills and firstly to understand the automated baseline script flags and to understand the output that resulted from the script.

I realised that even while running scans on websites I did not fully understand the output of what the code actually meant. This was pointed out by Faisal when he asked whether or not I understood what the various different output commands meant, in which I did not.

Firstly, understanding the automated scan script.

docker run -t owasp/zap2docker-stable zap-baseline.py -t <https://www.example.com>

-t – this simply meant the parameter that is the target website.

docker run -u zap -p 8080:8080 -i owasp/zap2docker-stable zap.sh -daemon -host 0.0.0.0 -port 8080 -config api.addrs.addr.name=.\* -config api.addrs.addr.regex=true -config api.key=<api-key>

-u – URL of the config file

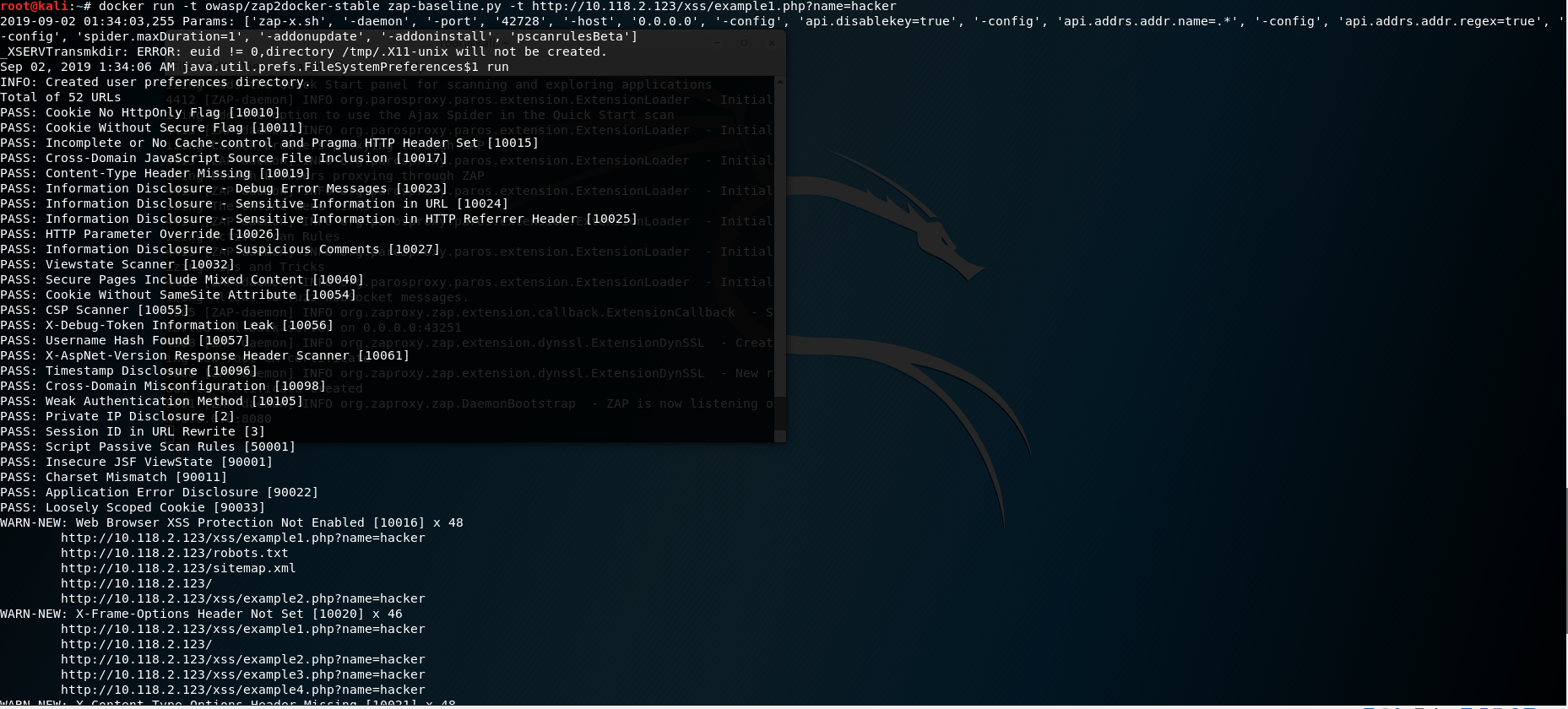
-daemon starts ZAP in daemon mode without a UI

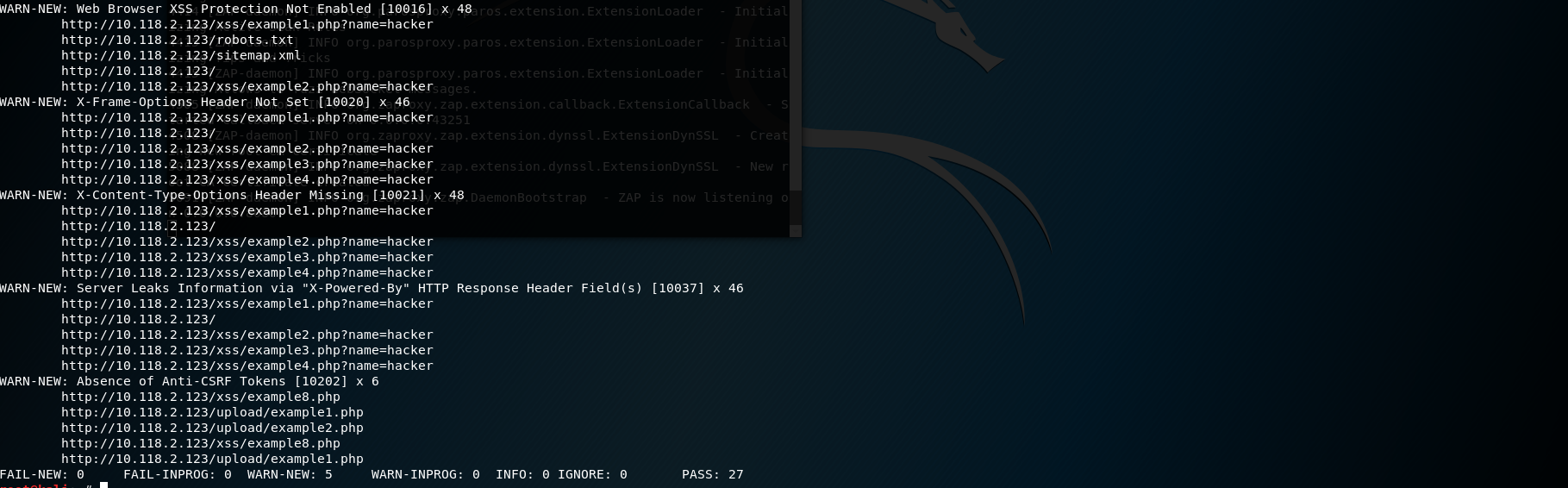
-config Overrides the specified key=value pair in the config file.

-host overrides the host used for proxying the specified in the config file.

-port overrides the port specified in the config file

Below is the baseline stable output on the example test website.





Originally I had no idea what this output actually meant. I didn’t need to understand every one of the output options but there were a couple in particular that Faisal said would be important to note.

**Cookie no secure flag**

The secure flag is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of a the cookie in clear text.

To accomplish this goal, browsers which support the secure flag will only send cookies with the secure flag when the request is going to a HTTPS page. Said in another way, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request.

By setting the secure flag, the browser will prevent the transmission of a cookie over an unencrypted channel.

**XSS protection not enabled**

XSS protection is designed to enable the XSS scripting filter built into modern web browsers. Usually enabled by default but using it will enforce it. Will enable the browser to block the response in the event that a malicious script has been inserted from user input instead of sanitizing.

**HTTP only enabled**

The cookie cannot be accessed through client side script (if the browser supports the flag ). Browser will not reveal the cookie to a third party if a user accidently accesses a link that exploits the flaw

**Absence of anti-csrf tokens**

Sets a token value that the attacker will not know, without this the attacker will not be able to use an attack, for example when the web browser sends a POST request the attacker might know the syntax of this request, the attacker could you code to publish an advertisement on the users profile. Token sets a token in the session cookie of your web browser right after you log in. All form submissions then include a hidden field containing the token.

Week 9

This week I decided I was going to re-visit where I had got to with Selenium, as I thought it was an important aspect of automated web testing. I had previously got stuck on starting zap through selenium, although I could find example code online to do so I never fully understood it so it was hard to implement. This time round I want to be able to understand what this example code is doing in order to implement it properly. I spent time watching a few different videos linked below to help my understanding:

<https://www.youtube.com/watch?v=jFBNCM61DbA>

<https://vimeo.com/222238217>

https://www.youtube.com/watch?v=aVFZFi\_6B9g

Week 10

<https://www.guru99.com/first-webdriver-script.html>

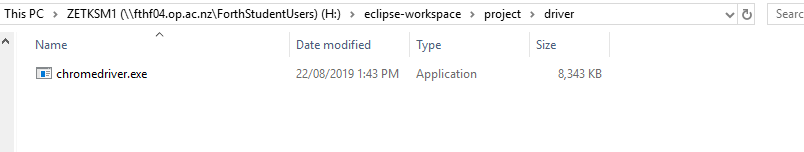
While revisiting Selenium I ran into a problem while trying to create a new class and run a different script.



I would continuously get an error than it could not find or load my main class although this class was made under a package that had worked previously for me with no obvious errors in the code. I’ve tried to fix this for a couple of hours and have decided that I will create a whole new package and re-import all of the selenium/web driver packages that are needed to run it. Doing this the first time was quite a meticulous process with quite a few small details that were easy to miss however I found a good guide on how to do this efficiently provided below.

<https://www.guru99.com/installing-selenium-webdriver.html>

After running through all the steps again I came across a different error. When running my script the chrome driver could not be found, which a simple fix was by just copying the chrome driver and creating a new driver folder in my new project workspace and pasting it inside.



My new scripts are now working and I am able to run the Facebook main page through selenium. I want to next do this in headless browser mode, as this will make it more efficient and use less resources to run. Originally I preferred the method of running it in a browser that I could see as this would let me troubleshoot any problems I could be having with my script. But this is not the most efficient method.

I am slightly more confident with selenium now as the scripts I’m trying to run are working again. I am now sure I’ll be able to start making more progress as I was stuck for a while trying to fix errors.

**Headless browser testing**

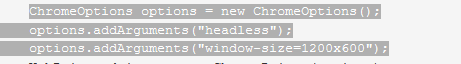
Today I am going to try to run my selenium scans through headless browsing. The reason for this is:

* Improves speed and performance, since this type of testing does not actually open a browser, the system saves the processing power that would otherwise be used in a real browser test. This leads to faster tests being executed.
* Allows testing browser less setups, there may be setups where installing a browser is not possible, such as server. In these cases, headless browsers help run automation tests easily.
* Helps you multitask, you can use your browser or your machine to do anything else while the tests run in the background. This saves hours of time that is otherwise spent staring at the screen.

I followed a tutorial I found online to try and implement their code to run my scripts in headless browser mode. The following link is where I found the code:

<https://www.built.io/blog/run-selenium-tests-in-headless-browser>

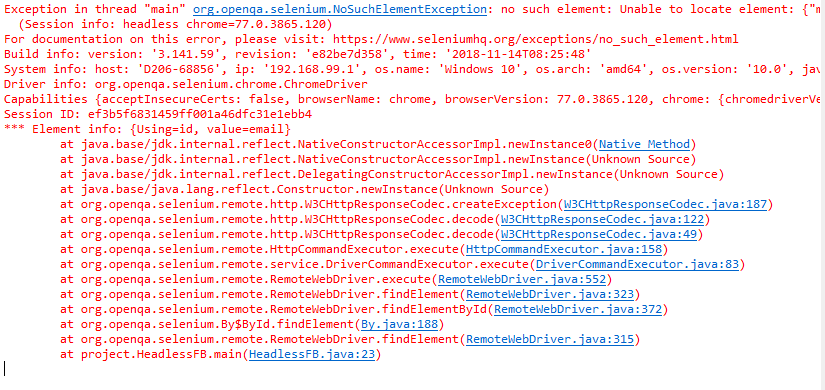
I added the following lines of code to my scripts to successfully run them in headless browser mode.



And then imported the following package.



The reason I knew this had been run successfully is because the output message was the same between the headless and the non-headless script. On further inspection I realised that the script was only running some of the time, there were other instances where I was getting an error output message shown below:



This was confusing as there were other times where the code ran fine. I will investigate further to see where the error is coming from. I attempted to run my “launchbrowser” script test in headless and this confirmed that the code was working successfully. I got an output message of “Test Passed”, while the script also ran in headless mode. This was good news as I felt like I am making progress in terms of getting my head around selenium.

I ran a small test to see if I could notice any performance improvements while running the script in headless mode as opposed to non-headless mode. The time for my script to run in headless mode was 9.12seconds while the time for the script to run in non-headless mode took 10.97seconds. This confirmed that performance had increased between the 2 scripts, I would like to try this again but once my script is larger and contains more code.

**Integrating Selenium with ZAP Re-Visit**

I tried to work on a different task once again relating to integrating ZAP with Selenium. I tried to follow a tutorial that I found online.

<https://premsvmm.blogspot.com/2018/07/how-to-integrate-oswap-zap-with-selenium.html>

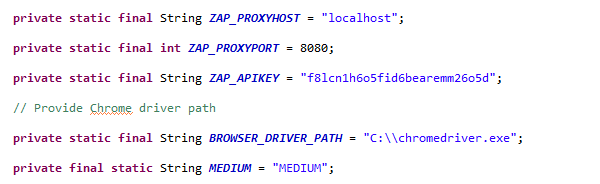
There was not a whole lot of documentation I could find, and when I did I found it extremely hard to understand. This tutorial had included example code on what I should be doing, however there was no explanation on what it was doing and how it was working. This was frustrating because every other tutorial I look at I cannot seem to find any example code to use, this seems to be the only tutorial I could find. After spending the afternoon trying to get this working I was unsuccessful. I will keep trying until I figure this out.

I found this tutorial below which seems like I will have a lot more success with as there is documentation included in this.

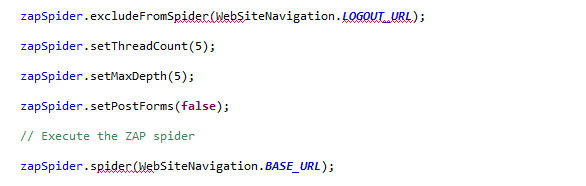
<https://dzone.com/articles/automate-zap-security-tests-with-selenium-webdrive-1>

Originally I feel like I got stuck on the idea of java script to open and run ZAP automatically, as I saw this on a previous tutorial I was looking at. However, after speaking with Faisal about this I realised I much simpler approach is just to launch ZAP manually and have the selenium script run after doing this. I think was holding me back for a long time as the method for doing my previous approach was too technically difficult for me. However, now I am back on track and have successfully created 3 new java classes which will use ZAP as a proxy when Selenium runs scripts to run tests on a web application. The first steps in doing so were easy, I basically just had to download the latest chrome driver and download a couple of jar files and place them into a folder inside my Selenium folder. I had to create my project as a maven project, which I originally had no idea what this was and why this was different from creating just a standard java project. After doing a bit of research I found that when creating a maven project a pom.xml file is created which contains information for the maven to build the project dependencies, build directory, source directory, plugins etc. Maven reads this pom.xml file then executes this. Maven projects can be run outside of Eclipse by running the batch file.

I created 3 different classes. To have ZAP run as a proxy I had to replace various fields in the code to hopefully have my ZAP work as the proxy as the code runs and for the browser path to be correct.



Testing the code that I found online has proven quite difficult as I am getting errors with it when trying to run as shown in the screenshots below. Troubleshooting the problems are hard because it isn’t my code. I originally thought it might be a problem with not importing the correct packages but everything looks fine. My next troubleshooting step is to ensure I am creating the project properly as I seem to be getting red crosses next to my java packages.



After running through the steps again on how to set up the project and packages I’m sure I’ve done this all correctly, so that is not the issue. I commented out some of the code that wasn’t working in the ZapSecurityTest class to see if the other 2 classes would run. The classes run successfully but it was hard to troubleshoot any potential issues because since the ZapSecurityTest class is not working I could not get ZAP to run as the proxy on the tests which is my main objective.

I fixed any of the red lines I was getting on my ZapSecurityTest class and was hopeful that I could then have ZAP run as the proxy for my scripts. However, it would still not work. I sent through the tutorial I was following to Faisal in hopes that he would be able to get the scripts running successfully and show me where I may have gone wrong with this.