# BTN 710 Assignment 1 Report

Students: Musawar Bajwa 115864167

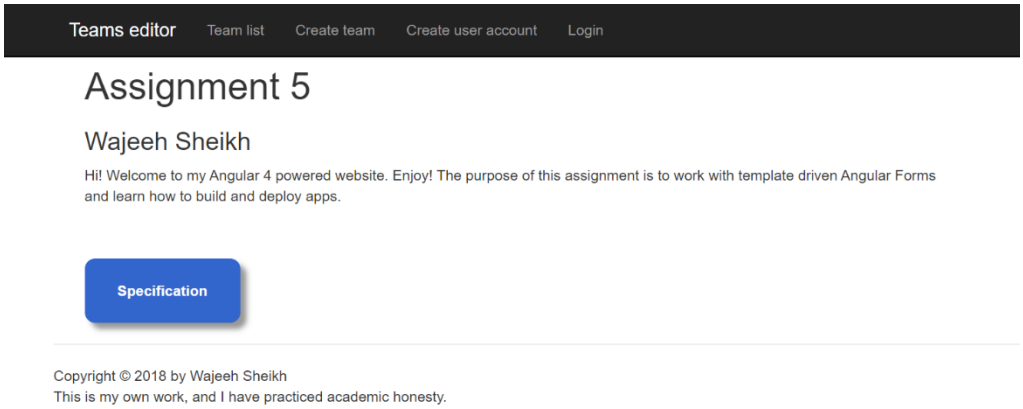
Wajeeh Sheikh 116963166

Zhiijan Zhou 043865112

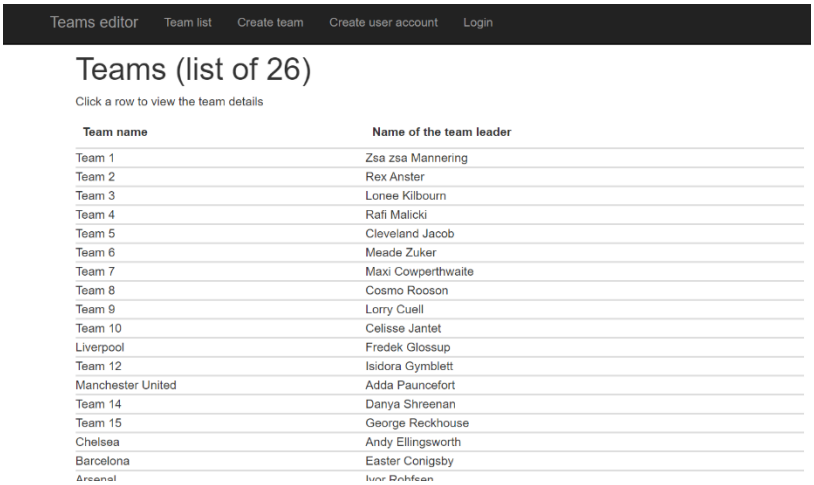
Group #: 8

# Introduction

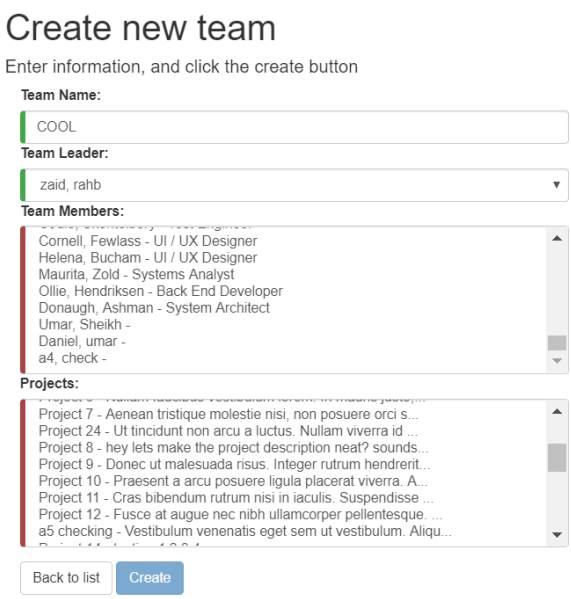
Assignment 1 for BTN 710 requires us to find a system/application to identify the security flaws in the program, by testing/attacking the system/application and analyzing the holes in the application. For our project we chose a web application, that was created in BTI 420, our 4th semester web course. It was pushed to Heroku and has a web service connected to MongoDB. We used GET, POST etc.. to add data to the database. When making this website we did not analyze the security of the website, we were more focused on the website application working as specified. We realized that there are very important security leaks in the application we created. We will be attacking this application using a program called ZAP (Zed Attack Proxy) and looking at the different vulnerabilities this application has. The link for the Heroku link of the website can be found here. Below you can find pictures of how the application looks and works.




*This is the home page of the web application, created by our group member Wajeeh Sheikh.*




*This is the team list page where you can view the details of each team and the names.*




*Create new team page*

## Section 1: The Exploit- Summary

For the attack we will be using a program called ZAP (Zed Attack Proxy). “ZAP is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project. ZAP is specifically designed for testing web applications” (Bennetts). The GitHub link for the project and source code can be found here. We will be using this software to attack our partners web application that was shown above. Software security testing is the process of assessing and testing a system to discover security risks and vulnerabilities of the system and its data. What ZAP does is it acts as a “man-in-the-middle proxy.” It stands between the tester’s browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. We chose ZAP as it is easy to install, and very easy to run a test on your web application, no need to install a bunch of programs and very simple GUI design to follow.


This is how the GUI looks like. 1. Menu Bar – Provides access to many of the automated and manual tools. 2. Toolbar – Includes buttons which provide easy access to most commonly used features. 3. Tree Window – Displays the Sites tree and the Scripts tree. 4. Workspace Window – Displays requests, responses, and scripts and allows you to edit them. 5. Information Window – Displays details of the automated and manual tools. 6. Footer – Displays a summary of the alerts found and the status of the main automated tools (ZAP Docs). The getting started guide can be found here along with other URLs explaining the program and how it tests for vulnerabilities:
https://github.com/zaproxy/zaproxy/releases/download/2.5.0/ZAPGettingStartedGuide-2.5.pdf

https://www.zaproxy.org/

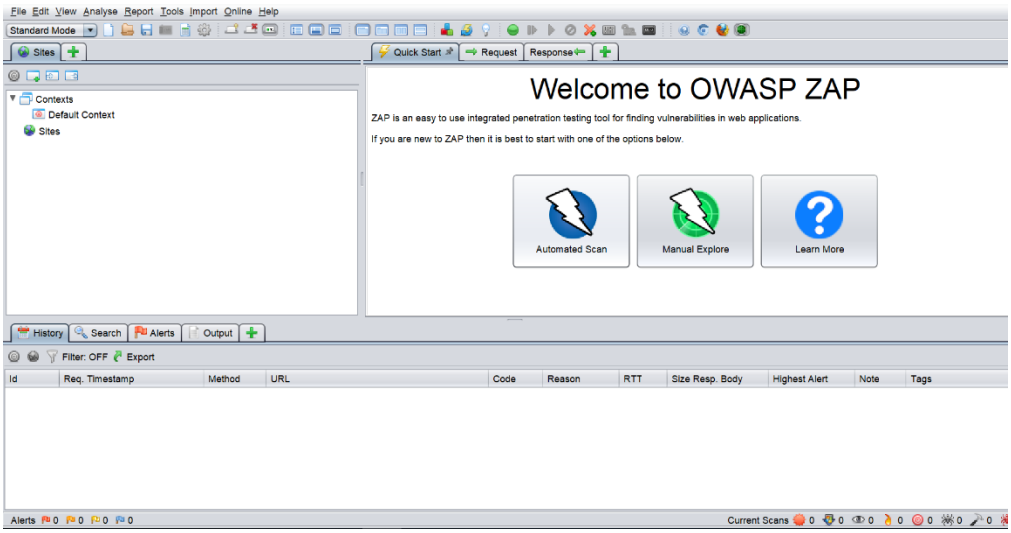
https://github.com/zaproxy/zap-core-help/wiki/HelpStartConceptsAscan

https://github.com/zaproxy/zap-core-help/wiki/HelpUiDialogsScanpolicymgr

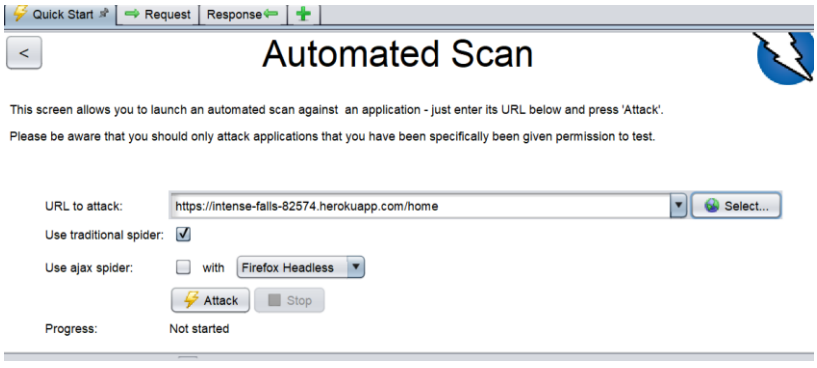
## Section 2- The Attack

Like stated before ZAP is known as a “man in the middle proxy”. It stands between the testers browser and the web application. A diagram better explaining this would be this diagram below.

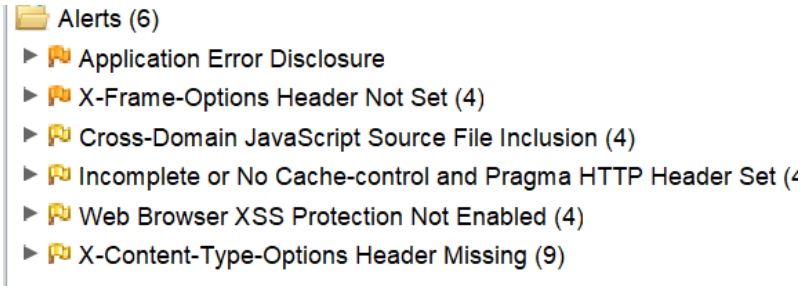

For demonstration purposes, and simplicity purposes we will be performing the automated scan found in the ZAP application. So first we will open up the application. And we will be greeted with the following welcome page with a list of options. Again for simplicity purposes and not wanting the report to be 30 pages showing the different options found in the ZAP applicaiton, we will be performing the automated scan.




Then we will enter our URL for the Heroku deployed web application. Which again can be found here.



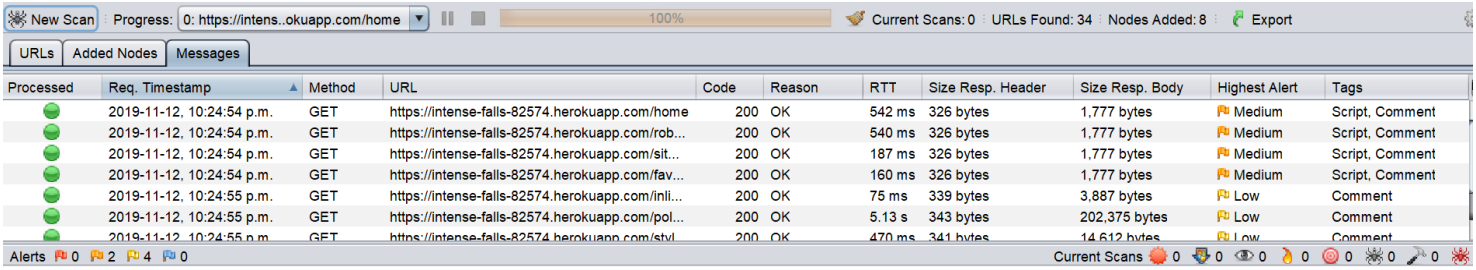
After pressing attack and letting the scan run. It comes back with the following results. The vulnerabilities will show up as alerts in the tab below shown in the image.




These are the list of vulnerabilities the application found and it shows how to improve these alerts.


The automated scan works like this, ZAP will passively scan all of the requests and responses proxied through it. So far ZAP has only carried out passive scans of your web application. Passive scanning does not change responses in any way and is considered safe. Scanning is also performed in a background thread to not slow down exploration. Passive scanning is good at finding some vulnerabilities and as a way to get a feel for the basic security state of a web application and locate where more investigation may be warranted. Active scanning, however, attempts to find other vulnerabilities by using known attacks against the selected targets. Active scanning is a real attack on those targets and can put the targets at risk, so do not use active scanning against targets you do not have permission, which in this case we were just testing the basic security state of the web application to test (ZAP Docs). We will go into further depth of Active scanning during our video

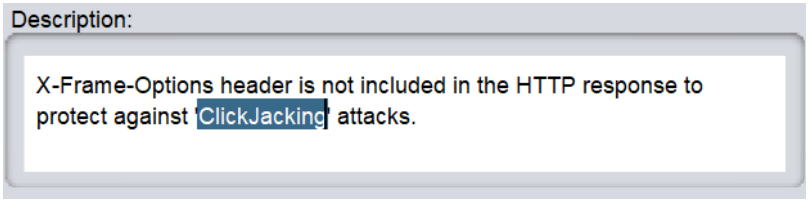
presentation of the assignment. More information on which method was used for the attack can be found under this tab of the application.




We will be looking at one of the solutions of the medium risk alerts found when scanning our web application. The alert we will be looking at is the Application Error Disclosure. This errors description is as follows this page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. A solution given to fix this certain risk is as follows. We will need to review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

## Section 3- Security Policy

Another alert we found was the we had a vulnerability to an attack called ClickJacking. A ClickJacking attacks trick web users into performing an action they did not intend, typically by rendering an invisible page element on top of the action the user thinks they are performing (Hacksplaining).




A determined hacker can harvest login credentials, spread worms on social media websites, spread malware etc.. with this type of attack. In order to protect against it, there is a security policy put in place. The Content-Security-Policy HTTP header is part of the HTML5 standard, and provides a broader range of protection than the X-Frame-Options header (which it replaces). It is designed in such a way that website authors can whitelist individual domains from which resources (like scripts, stylesheets, and fonts) can be loaded, and also domains that are permitted to embed a page. To learn more about how to implement this security policy into your code you can learn here.

Bibliography

"Protecting Your Users Against Clickjacking." *Hacksplaining*, www.hacksplaining.com/prevention/click-jacking.

https://github.com/zaproxy/zaproxy (ZAP GitHub)

https://intense-falls-82574.herokuapp.com/home (Website we attacked)

https://www.zaproxy.org/ (ZAP Docs)