Configuration Manual

MSc Industrial Internship
MSc Cyber Security

Vladyslav Bril
Student ID: x21102872

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vladyslav Bril		
Student ID:	x21102872		
Programme:	MSc Cybersecurity	**Year:**	2022
Module:	MSc Internship		
Lecturer:	Vikas Sahni		
Submission Due Date:	6th of January		
Project Title:	Automation of Remediation of Configuration Vulnerabilities Reported by the DAST Scanning Procedure		
Word Count:	2760	**Page Count:**	17

Signature: [illegible]

Date: 06.01.2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☑
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☑
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vladyslav Bril
x21102872

1 Introduction

The software developed during the academic research detailed in the main thesis document is described in this manual along with additional technical specifications and directions for installing and running the application. This document's subsequent parts include data on the development environment (hardware and software features), the software components utilized and their corresponding versions, initial installation instructions, correct development operation, and a brief description of the application's outcomes. The primary paper on the chosen topic contains details on the architecture and design phase.

2 System Configuration

The hardware specifications of the device used for the application's development and testing are briefly described in the part that follows, along with a list of all the software tools, frameworks, and solutions that were employed in the process.

2.1 Hardware Configuration

Table 1 contains the details regarding the hardware components & features of the system that was utilized to run the software developed for the corresponding topic.

Table 1: Hardware Specification & Requirements

Item Name	Device Configuration
CPU	Apple M1 ARM (8 cores)
RAM	8 GB LPDDR4
Disk Space	Apple SSD 256 GB

2.2 Software Configuration

Table 2 contains all the details regarding the software components & their details (such as version and brief description) that were used while developing a new solution to the pre-defined problem.

Table 2: Software Specification & Requirements

Item Name	Brief Description	Software Version
Operating System	macOS Ventura (22A400)	13.0.1
Java	The main programming language used while developing the application.	OpenJDK 19.0.1
Zsh	A UNIX shell that enhances the capabilities of bash and is used in a development environment.	Zsh 5.8.1 (x86_64-apple-darwin22.0)
OWASP ZAP CLI	An add-on to the main OWASP ZAP application that provides an option to run the scanner through the CLI.	OWASP ZAP 2.12.0
Selenium	A tool that automates the actions inside a browser (e.g., Chrome).	Selenium 4.7.2
Chrome Driver	A specific executable file that needs to be addressed in the main code of the application to run a Chrome instance.	ChromeDriver 108.0.5359.71
Apache Maven	A dedicated framework for automated project assembly based on files written in the POM language.	Maven 3.7.1
TestNG	A testing framework that works with Selenium and enhances the features such as verification of data gathering result or data match.	TestNG 7.7.0
JUnit	A dedicated framework for unit testing software in Java language.	JUnit 4.13.2
XAMPP	A cross-platform build that supports running a full-featured web server and includes Apache, MariaDB, a PHP script interpreter, the Perl programming language, and several more libraries.	XAMPP 8.1.5

3 Application Implementation Guideline

The set of instructions in the next part are intended to show how to install all the required components, set them up, and start them in order to reproduce the application's testing phase's outcomes. Each area below is accountable for its portion of the overall roadmap for the successful launch and confirmation of the built application's correct operation.

3.1 Pre-Requisite Download List

The following software elements must be present on a functioning device in order for it to interact with the given code before starting the actual program extraction:

- Download[1] and install any IDE that supports Java programming language (the author tested the application on two of the instances: IntelliJ IDEA CE and Visual Studio Code)
- Download[2] and install Java (make sure to install the version mentioned earlier in the Table 2)
- Download[3] and install XAMPP (or any other platform-oriented build, such as LAMP, IAMP, WAMP, etc.). Since the program mainly requires the presence of the Apache Web Server as the main server, there is no difference in the type of web stack installed (all of them are provided with the Apache Web Server)
- Download[4] and install OWASP ZAP
- Download[5] the ChromeDriver executable file and save it to the Desktop folder of the operating system (a copy of the driver can be found in the archive with the source code)

3.2 Installation and Configuration of the Application

It is possible to extract the package containing all the required code from the developed program after successfully installing every item on the aforementioned list. The purported content of the downloaded package included with this handbook is seen in Figure 1.

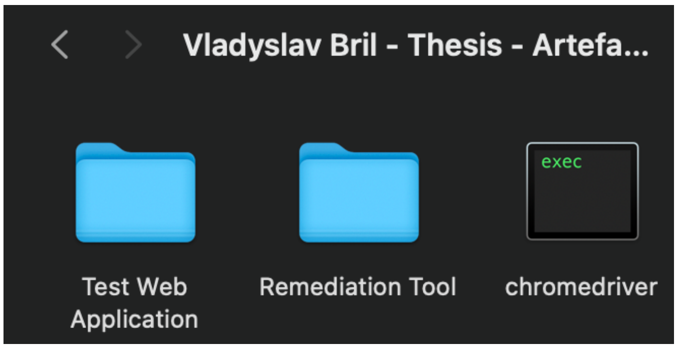
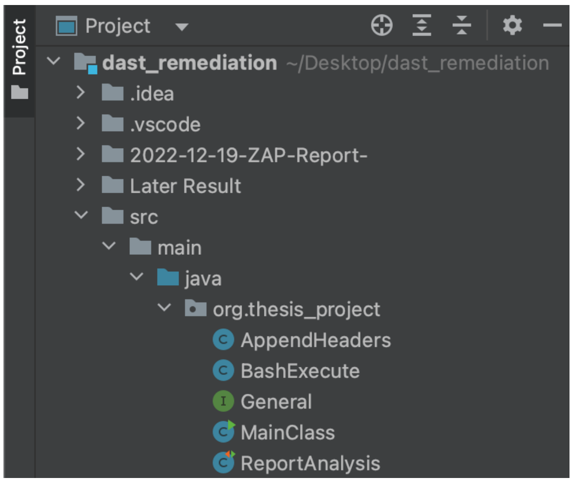


Figure 1: The original content of the archive containing the artifacts

[1] https://www.jetbrains.com/idea/
[2] https://www.java.com/en/download/
[3] https://www.apachefriends.org/index.html
[4] https://www.zaproxy.org
[5] https://chromedriver.chromium.org/downloads

The web application that will be tested is located in the *Thesis-Project* folder. The application is a website built using HTML, CSS, and JS, with the backend components written in PHP and linked to a MySQL database operated under the XAMPP stack. After installing XAMPP, this folder (*Thesis-Project*) has to be placed in the following path: */Applications/XAMPP/htdocs/*.

All of the application's necessary source code may be found in the *dast_remediation* folder. To inspect the content of the files inside this folder, address it in the previously installed IDE (such as IntelliJ IDEA) and then open it (see Figure 2).

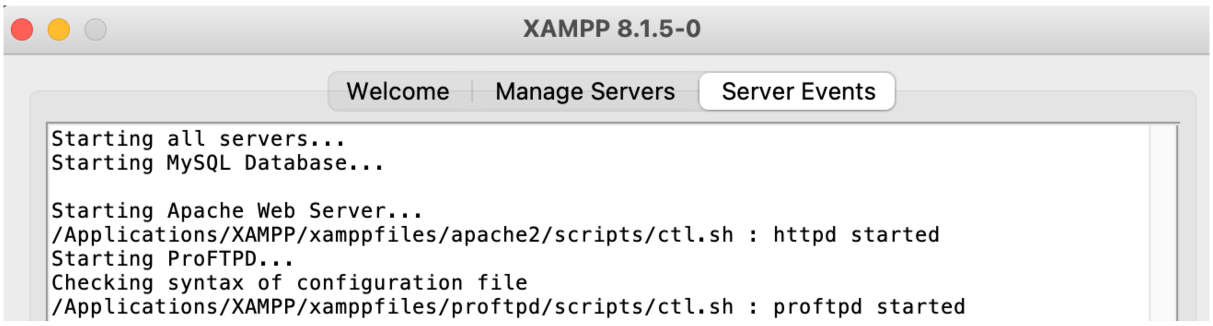


```
Project
dast_remediation ~/Desktop/dast_remediation
  .idea
  .vscode
  2022-12-19-ZAP-Report-
  Later Result
  src
    main
      java
        org.thesis_project
          AppendHeaders
          BashExecute
          General
          MainClass
          ReportAnalysis
```

Figure 2: The overview of the project in the IntelliJ IDEA IDE

To execute the Chrome instance for report scanning tasks, the last file in the archive - *chromedriver executable* - should be explicitly specified in the source code and placed in the OS's Desktop folder.

Launching all XAMPP stack services, particularly Apache Web Server, is the final step needed before starting the web application scan (see Figure 3). Make sure that the target web application can be accessed by the device user by typing in the browser the following address: *localhost/Thesis-Project/index.html*



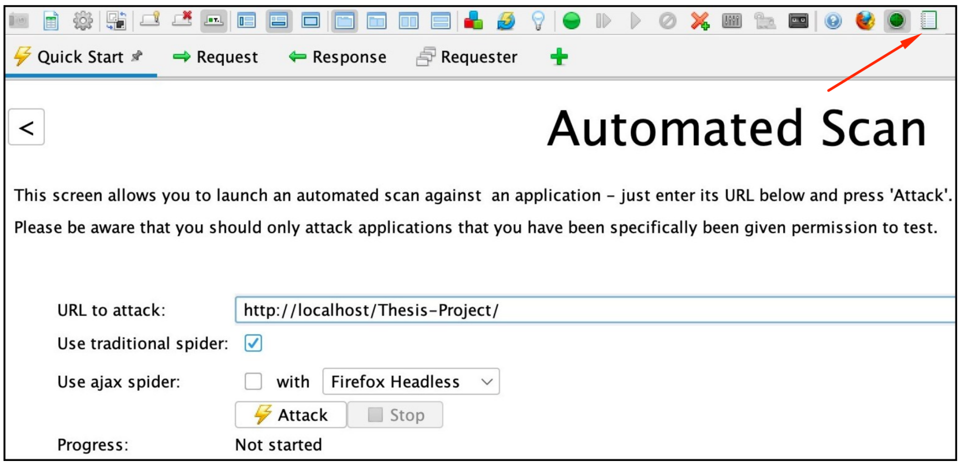
```
Starting all servers...
Starting MySQL Database...

Starting Apache Web Server...
/Applications/XAMPP/xamppfiles/apache2/scripts/ctl.sh : httpd started
Starting ProFTPD...
Checking syntax of configuration file
/Applications/XAMPP/xamppfiles/proftpd/scripts/ctl.sh : proftpd started
```

Figure 3: A successful launch of the XAMPP services (MySQL DB; ProFTPD and Apache Web Server)

3.3 Running the OWASP ZAP and Generating the Report

After the application has been installed and set up on the system, the OWASP ZAP scanning process may be started to provide a report to analyse. There are two methods to carry out this action: either through the UI (the standard method) or the CLI command (optional way). Due to access to a better reporting format from the application, the focus of the following guidance will continue to be on the UI method of executing the scan.

Open the OWASP ZAP, select the *Automated Scan* option and then paste the URL of the web application (e.g., *localhost/Thesis-Project/*). When the OWASP ZAP has finished scanning and attacking, click the *Attack* button. After that, wait until it is time to generate the report by selecting the top-right icon from the menu toolbar (see Figure 4 for the reference).



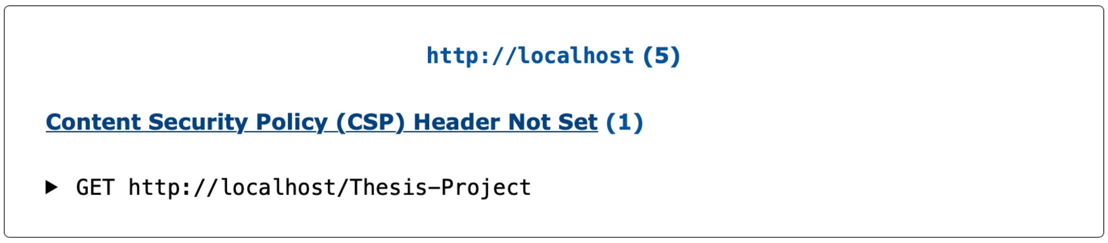
Automated Scan

This screen allows you to launch an automated scan against an application – just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/Thesis-Project/
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Progress: Not started

Figure 4: Specifying the attacking URL (highlighted textbox) and Generate Report function (red arrow)

When the scan is finished and the report is being created, open it, and view the report to examine the content of the *Warnings* section to find all the missing headers problems that are connected to the previously provided URL (ignore any occurrences of the tracking URLs, third-party ads services, etc.). Figure 5 shows an illustration of one of these warnings that highlights the absence of a *Content-Security-Policy* header in the Apache Web Server config file.



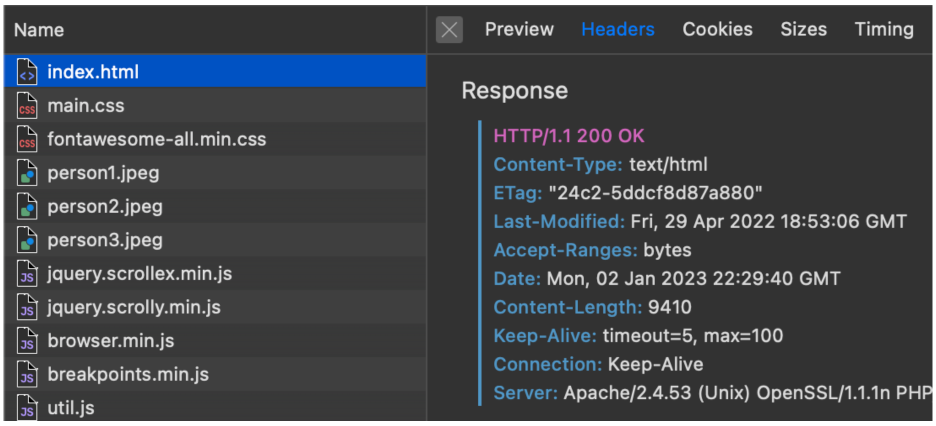
http://localhost (5)

Content Security Policy (CSP) Header Not Set (1)

▶ GET http://localhost/Thesis-Project

Figure 5: Identifying an existing misconfiguration issue related to the absence of the CSP Header in the Apache Web Server configuration

In order to verify the absence of the header manually, proceed to the *index.html* page and open the *Network* tab in the *Developer Tools* menu (see the browser specification how to open that menu). Click on the file with the *index.html* page and proceed to the *Headers* tab. Make sure that there are no headers set that were reported by the OWASP ZAP (Figure 6).



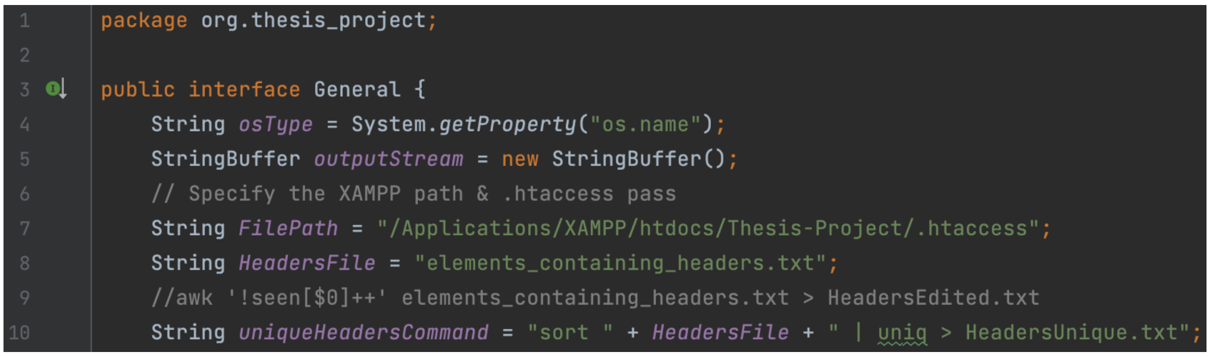
```
index.html
main.css
fontawesome-all.min.css
person1.jpeg
person2.jpeg
person3.jpeg
jquery.scrollex.min.js
jquery.scrolly.min.js
browser.min.js
breakpoints.min.js
util.js
```

```
HTTP/1.1 200 OK
Content-Type: text/html
ETag: "24c2-5ddcf8d87a880"
Last-Modified: Fri, 29 Apr 2022 18:53:06 GMT
Accept-Ranges: bytes
Date: Mon, 02 Jan 2023 22:29:40 GMT
Content-Length: 9410
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Server: Apache/2.4.53 (Unix) OpenSSL/1.1.1n PHP
```

Figure 6: Verifying the non-appearance of the list of headers found by the OWASP ZAP

3.4 Walkthrough on the Application's Source Code

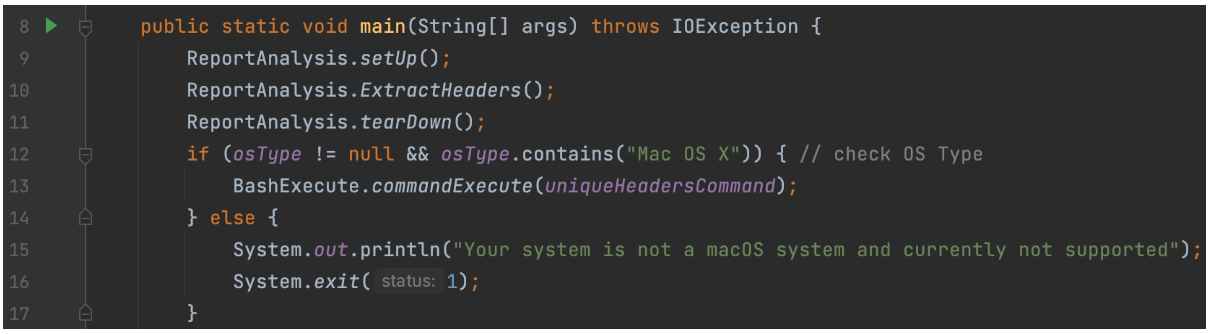
The following subsection will demonstrate how to run the developed application after opening it up in the IDE, as well as describe logical modules of the application that correspond to different actions that will take place while the application is running. First of all, proceed to the *General.java* interface file and observe the names and values of the variables (Figure 7). Make sure that you specify the path to the *.htaccess* file correctly (set by default for *macOS* general application directory). Additionally, the *HeadersFile* variable is responsible for storing the name of the file that will be created with all the instances of the found headers issues.



```java
package org.thesis_project;

public interface General {
    String osType = System.getProperty("os.name");
    StringBuffer outputStream = new StringBuffer();
    // Specify the XAMPP path & .htaccess pass
    String FilePath = "/Applications/XAMPP/htdocs/Thesis-Project/.htaccess";
    String HeadersFile = "elements_containing_headers.txt";
    //awk '!seen[$0]++' elements_containing_headers.txt > HeadersEdited.txt
    String uniqueHeadersCommand = "sort " + HeadersFile + " | uniq > HeadersUnique.txt";
```

Figure 7: General file with static variables that contain paths to other applications and names of the files to be produced by the program

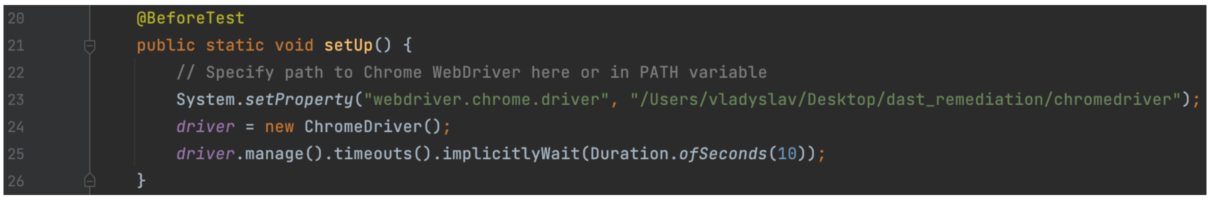
Proceed to the *MainClass.java* file to see the main executable class that operates all other subcomponents of the application. The first logical part of the application starts from line 9 and ends with line 17 (Figure 8). The purpose of this part is to work with the earlier generated report in order to extract the headers to a separate file and make sure that there is no repetition in that file for further goals (line 13 serves the aforementioned purpose).



```java
    public static void main(String[] args) throws IOException {
        ReportAnalysis.setUp();
        ReportAnalysis.ExtractHeaders();
        ReportAnalysis.tearDown();
        if (osType != null && osType.contains("Mac OS X")) { // check OS Type
            BashExecute.commandExecute(uniqueHeadersCommand);
        } else {
            System.out.println("Your system is not a macOS system and currently not supported");
            System.exit( status: 1);
        }
```

Figure 8: A code snippet from the *MainClass.java* file responsible for data extraction from the previously generated report

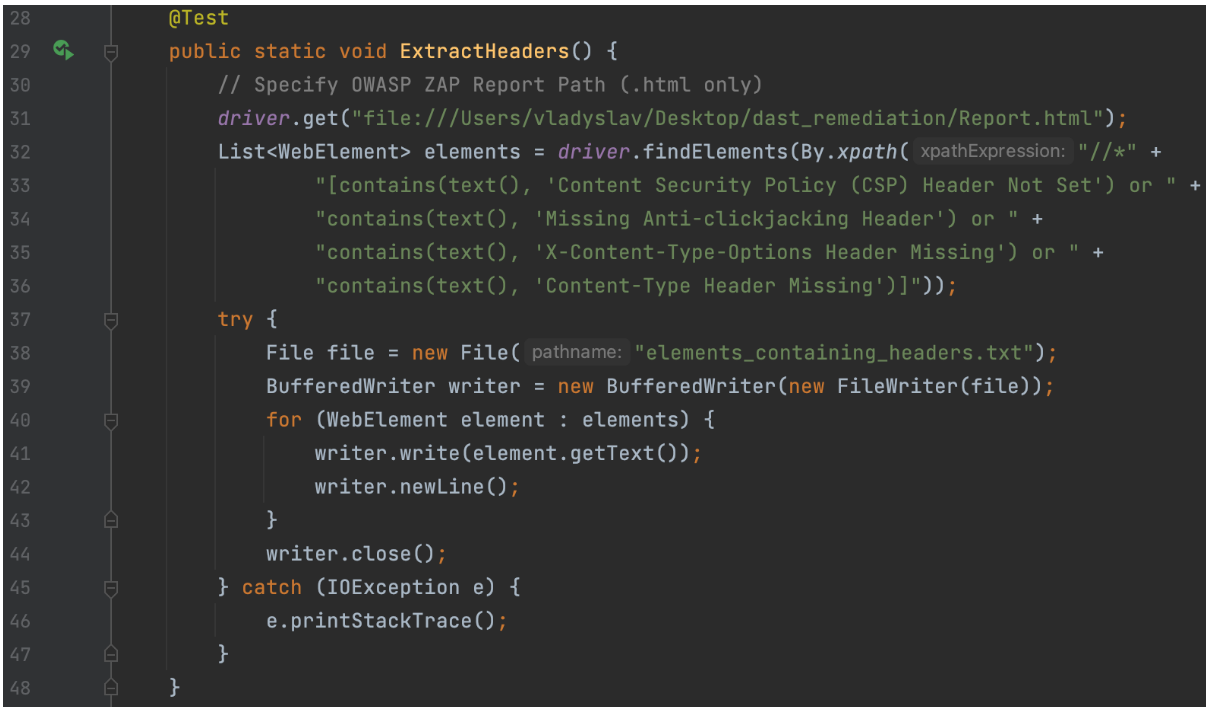
Proceed to the *ReportAnalysis.java* file to view the realisation of the report scanning and data extracting functionality of the application. This file is logically split on three different sections by using the *TestNG* annotations: the first one is labelled as *BeforeTest* and it is responsible for specifying the place of the Chrome Driver to the application (make sure to include a new path), initializes the Chrome Driver object and sets the implicit waiters for the browser (Figure 9).



```java
    @BeforeTest
    public static void setUp() {
        // Specify path to Chrome WebDriver here or in PATH variable
        System.setProperty("webdriver.chrome.driver", "/Users/vladyslav/Desktop/dast_remediation/chromedriver");
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
    }
```

Figure 9: *BeforeTest* annotation responsible for setting up the Chrome webdriver

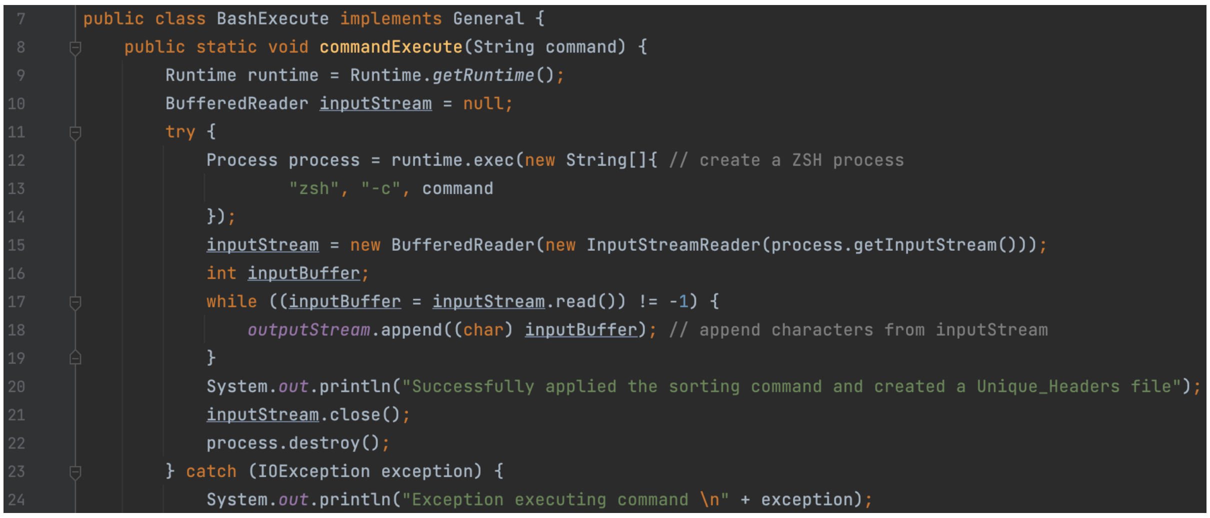
The next annotation is called *Test* and works in order to identify the matching headers from the *List* collection of *WebElements* in the generated report by OWASP ZAP (Figure 10). As for the example, a current version of the application is looking for the issues related to a poorly configured or entirely missing *Content-Security-Policy* header, *Anti-Clickjacking* header, *X-Content-Type-Options* header, or a *Content-Type* header. This method also extracts all the positively found matches into a separate file with the name specified earlier in the *General.java* interface (by default, *elements_containing_headers.txt*).



```java
@Test
public static void ExtractHeaders() {
    // Specify OWASP ZAP Report Path (.html only)
    driver.get("file:///Users/vladyslav/Desktop/dast_remediation/Report.html");
    List<WebElement> elements = driver.findElements(By.xpath( xpathExpression: "//*" +
            "[contains(text(), 'Content Security Policy (CSP) Header Not Set') or " +
            "contains(text(), 'Missing Anti-clickjacking Header') or " +
            "contains(text(), 'X-Content-Type-Options Header Missing') or " +
            "contains(text(), 'Content-Type Header Missing')]"));
    try {
        File file = new File( pathname: "elements_containing_headers.txt");
        BufferedWriter writer = new BufferedWriter(new FileWriter(file));
        for (WebElement element : elements) {
            writer.write(element.getText());
            writer.newLine();
        }
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 10: ***Test*** **annotation responsible for scanning the generated report and extracting the found misconfiguration issues - if found – to a separate file**

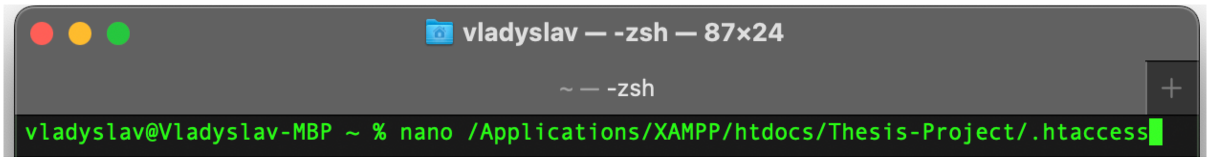
The last annotation (*AfterTest*) serves to quit the driver and remove its instance from the running application. When all three annotations are successfully completed, the operation is handled to the other part of the program which is responsible for sorting the list of extracted headers alphabetically and removes the duplicates detected. Additional overview on the process of executing the command in *Zsh* to sort the newly created file can be seen in the *BashExecute.java* file (Figure 11). Once this class is called, it creates an instance of the shell process that gets an instruction to execute as a regular shell command in a UNIX system. A user should be able to see the successful message after applying such an action, otherwise an error will appear on the screen (line 20).



```java
7   public class BashExecute implements General {
8       public static void commandExecute(String command) {
9           Runtime runtime = Runtime.getRuntime();
10          BufferedReader inputStream = null;
11          try {
12              Process process = runtime.exec(new String[]{ // create a ZSH process
13                      "zsh", "-c", command
14              });
15              inputStream = new BufferedReader(new InputStreamReader(process.getInputStream()));
16              int inputBuffer;
17              while ((inputBuffer = inputStream.read()) != -1) {
18                  outputStream.append((char) inputBuffer); // append characters from inputStream
19              }
20              System.out.println("Successfully applied the sorting command and created a Unique_Headers file");
21              inputStream.close();
22              process.destroy();
23          } catch (IOException exception) {
24              System.out.println("Exception executing command \n" + exception);
```

Figure 11: ***BashExecute.java*** **file containing the realisation of shell commands execution in** ***Zsh***

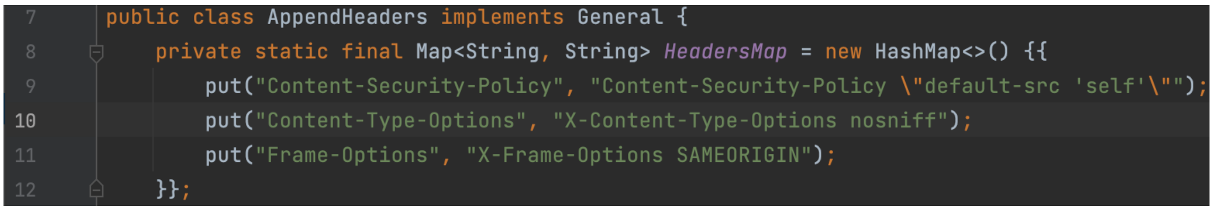
The final class that is triggered from the *MainClass.java* file is aimed to map the existing values of the headers from the generated file with the proper parameters that could be stored into the *.htaccess* Apache configuration file. In order to view the content of the *.htaccess* file (which is empty by default), proceed to any shell and type in the following command: *nano /Path/To/.htaccess/file* (located in Thesis-Project folder copied earlier; see Figure 12 for the reference).



```
vladyslav — -zsh — 87×24
~ — -zsh
vladyslav@Vladyslav-MBP ~ % nano /Applications/XAMPP/htdocs/Thesis-Project/.htaccess
```

Figure 12: A command to read the content of the *.htaccess* configuration file

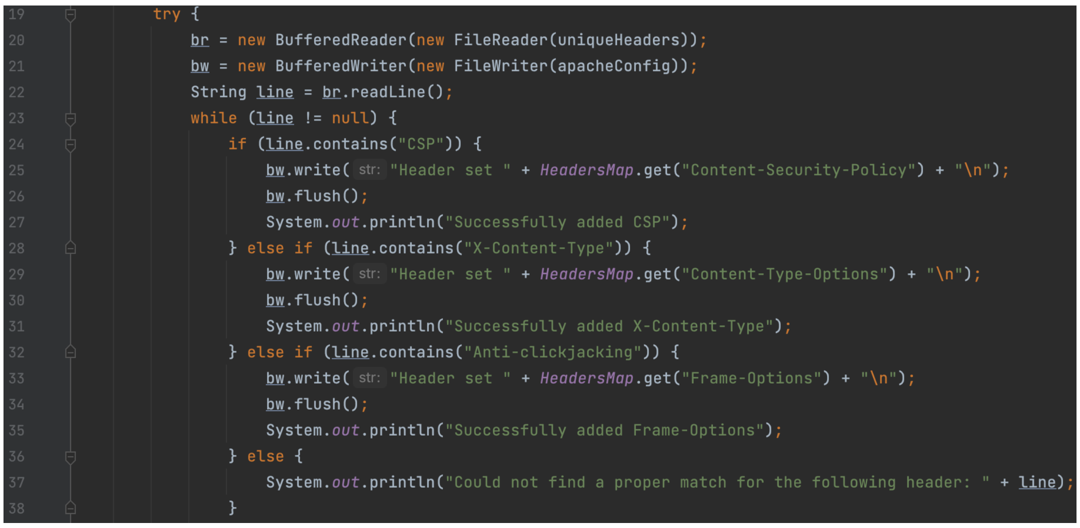
At the beginning of the class, a static *Map* called *HeadersMap* is created and filled in with a set of pre-defined values to identify all the types of headers that are currently supported by the version of the application (Figure 13).



```java
public class AppendHeaders implements General {
    private static final Map<String, String> HeadersMap = new HashMap<>() {{
        put("Content-Security-Policy", "Content-Security-Policy \"default-src 'self'\"");
        put("Content-Type-Options", "X-Content-Type-Options nosniff");
        put("Frame-Options", "X-Frame-Options SAMEORIGIN");
    }};
```

Figure 13: A static *Map* collection containing the values to be put into the configuration file

The rest of the code in the class is aimed to read the content of the file that contains all the unique headers indemnified on the previous steps and match them with the existing ones from the *HeadersMap* HashMap collection. If such an occurrence takes place, the .htaccess is getting populated with the corresponding row of data, matching the header value and a proper formatting that is supported by the Apache Web Server. If there is no match found in the *HeadersMap*, the user will see an error in the console (Figure 14).

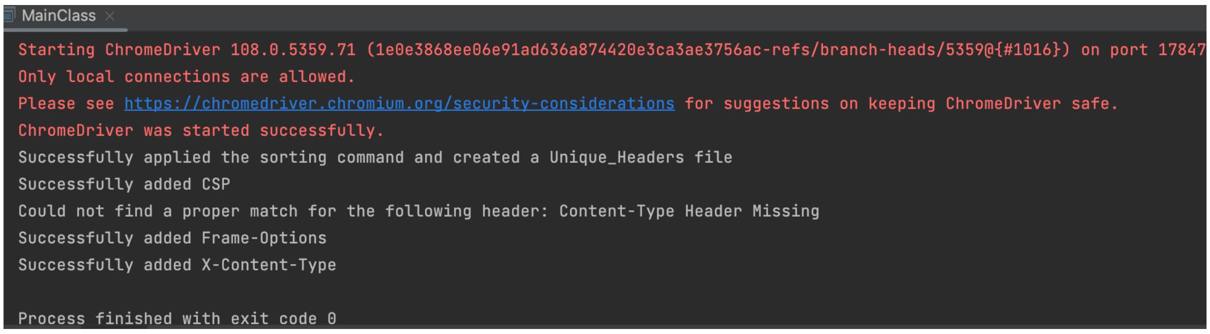


```java
try {
    br = new BufferedReader(new FileReader(uniqueHeaders));
    bw = new BufferedWriter(new FileWriter(apacheConfig));
    String line = br.readLine();
    while (line != null) {
        if (line.contains("CSP")) {
            bw.write( str: "Header set " + HeadersMap.get("Content-Security-Policy") + "\n");
            bw.flush();
            System.out.println("Successfully added CSP");
        } else if (line.contains("X-Content-Type")) {
            bw.write( str: "Header set " + HeadersMap.get("Content-Type-Options") + "\n");
            bw.flush();
            System.out.println("Successfully added X-Content-Type");
        } else if (line.contains("Anti-clickjacking")) {
            bw.write( str: "Header set " + HeadersMap.get("Frame-Options") + "\n");
            bw.flush();
            System.out.println("Successfully added Frame-Options");
        } else {
            System.out.println("Could not find a proper match for the following header: " + line);
        }
```

Figure 14: A code snippet responsible for finding the matching pairs of values from the file and the previously made file with unique headers

3.5 Running the Application and Verifying the Output

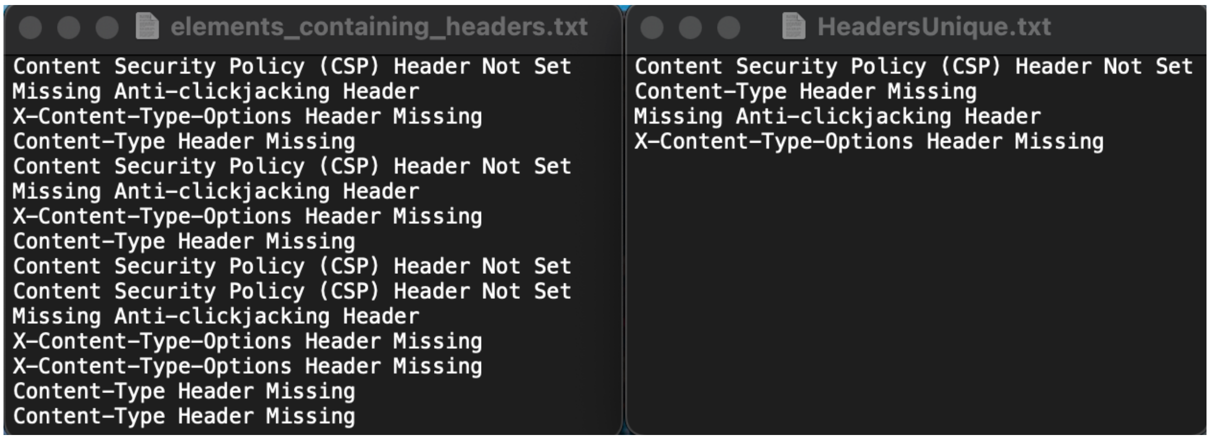
In order to successfully launch the application, proceed to the *MainClass.java* file and run the *main()* method. Make sure that the built-in IDE console is visible, since all the text prompts and additional output will be showed there. If any error occurs while running the application, the corresponding text message will be displayed in the console window, specifying the possible root of the problem (e.g., wrongly set path for web driver, error in executing a certain command, etc.). Figure 15 shows an example of a successful application execution that is supported by the text messages from the output window.



```
Starting ChromeDriver 108.0.5359.71 (1e0e3868ee06e91ad636a874420e3ca3ae3756ac-refs/branch-heads/5359@{#1016}) on port 17847
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
Successfully applied the sorting command and created a Unique_Headers file
Successfully added CSP
Could not find a proper match for the following header: Content-Type Header Missing
Successfully added Frame-Options
Successfully added X-Content-Type

Process finished with exit code 0
```

Figure 15: Result of executing the *main()* method in the *MainClass.java* file

For additional verification of the obtained results, proceed to the project's folder containing the source code. Two newly generated files could be seen in the directory, one of which contains all the occurrences of the missing headers from the report, while the other one has sorted them out and contains only unique options sorted alphabetically (Figure 16).

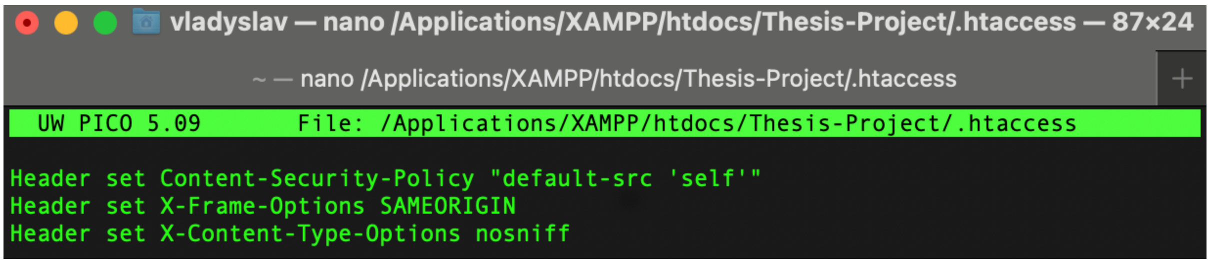


```
Content Security Policy (CSP) Header Not Set
Missing Anti-clickjacking Header
X-Content-Type-Options Header Missing
Content-Type Header Missing
Content Security Policy (CSP) Header Not Set
Missing Anti-clickjacking Header
X-Content-Type-Options Header Missing
Content-Type Header Missing
Content Security Policy (CSP) Header Not Set
Content Security Policy (CSP) Header Not Set
Missing Anti-clickjacking Header
X-Content-Type-Options Header Missing
X-Content-Type-Options Header Missing
Content-Type Header Missing
Content-Type Header Missing
```

```
Content Security Policy (CSP) Header Not Set
Content-Type Header Missing
Missing Anti-clickjacking Header
X-Content-Type-Options Header Missing
```

Figure 16: Two newly generated text files containing the headers from the produced report

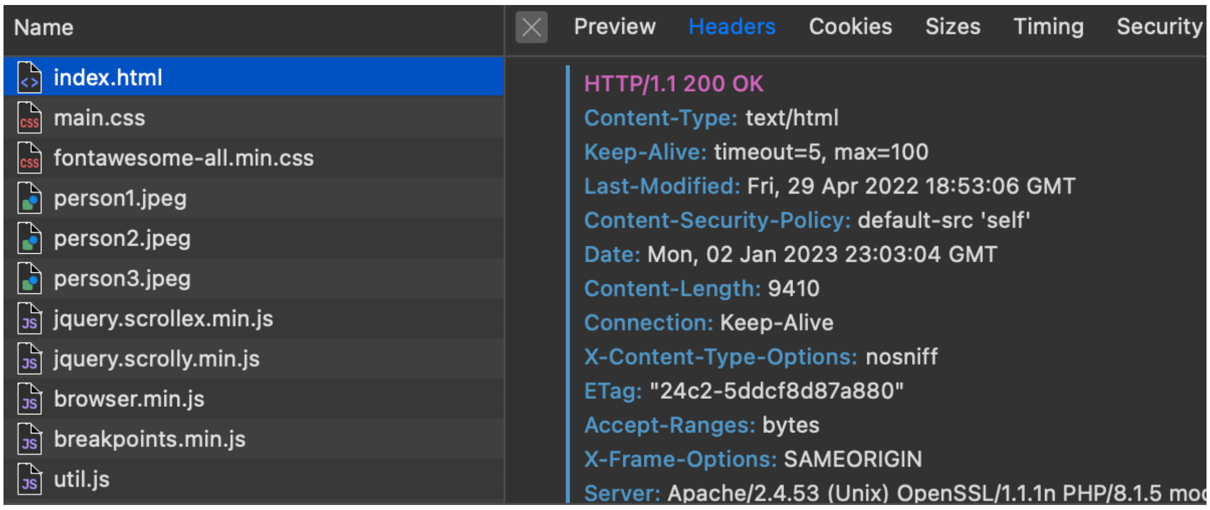
Moreover, after executing the command showed in Figure 12 once again, an *.htaccess* configuration file should be now populated with three configured server-oriented headers with a proper formatting style (see Figure 17 for the reference).



```
vladyslav — nano /Applications/XAMPP/htdocs/Thesis-Project/.htaccess — 87×24
~ — nano /Applications/XAMPP/htdocs/Thesis-Project/.htaccess
  UW PICO 5.09        File: /Applications/XAMPP/htdocs/Thesis-Project/.htaccess

Header set Content-Security-Policy "default-src 'self'"
Header set X-Frame-Options SAMEORIGIN
Header set X-Content-Type-Options nosniff
```

Figure 17: A newly edited *.htaccess* file with the missing headers reported by the OWASP ZAP

Furthermore, after relaunching the Apache Web Server using the XAMPP 'Restart Now' button in the UI and proceeding to the index.html page, the web server will return the specified headers with the appropriate values. To view that, repeat the steps specified above Figure 6 (Network tab). The result of successfully changing the configuration settings of the Apache Web Server is presented on the Figure 18.



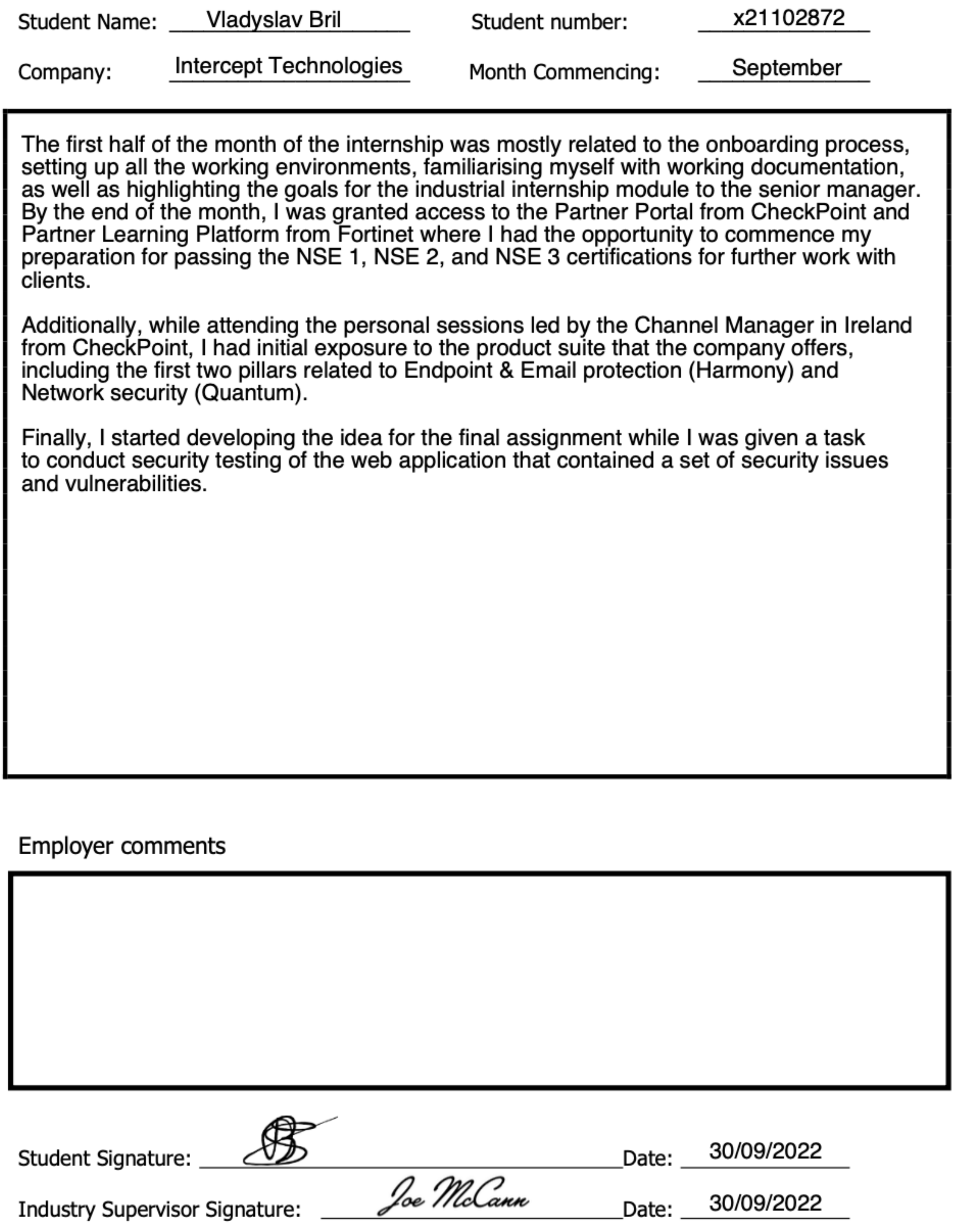
```
index.html
main.css
fontawesome-all.min.css
person1.jpeg
person2.jpeg
person3.jpeg
jquery.scrollex.min.js
jquery.scrolly.min.js
browser.min.js
breakpoints.min.js
util.js
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Keep-Alive: timeout=5, max=100
Last-Modified: Fri, 29 Apr 2022 18:53:06 GMT
Content-Security-Policy: default-src 'self'
Date: Mon, 02 Jan 2023 23:03:04 GMT
Content-Length: 9410
Connection: Keep-Alive
X-Content-Type-Options: nosniff
ETag: "24c2-5ddcf8d87a880"
Accept-Ranges: bytes
X-Frame-Options: SAMEORIGIN
Server: Apache/2.4.53 (Unix) OpenSSL/1.1.1n PHP/8.1.5 mo
```

Figure 18: A successful work of the automated script for fixing the headers related issue in the Apache Web Server configuration

Finally, if the OWASP ZAP scan is retriggered on the current state of the application, the newly generated report won't show the instances of missing headers being detected while investigating the application provided, which makes a point that an automated script is capable of fixing the reported issues after the DAST scanning procedure by inserting the changes into the configuration of the software that match the pre-defined pattern in the source code.

4 Monthly Activity Reports



Student Name:	Vladyslav Bril	Student number:	x21102872
Company:	Intercept Technologies	Month Commencing:	September

The first half of the month of the internship was mostly related to the onboarding process, setting up all the working environments, familiarising myself with working documentation, as well as highlighting the goals for the industrial internship module to the senior manager. By the end of the month, I was granted access to the Partner Portal from CheckPoint and Partner Learning Platform from Fortinet where I had the opportunity to commence my preparation for passing the NSE 1, NSE 2, and NSE 3 certifications for further work with clients.

Additionally, while attending the personal sessions led by the Channel Manager in Ireland from CheckPoint, I had initial exposure to the product suite that the company offers, including the first two pillars related to Endpoint & Email protection (Harmony) and Network security (Quantum).

Finally, I started developing the idea for the final assignment while I was given a task to conduct security testing of the web application that contained a set of security issues and vulnerabilities.

Employer comments

Student Signature: [illegible] Date: 30/09/2022

Industry Supervisor Signature: Joe McCann Date: 30/09/2022

Student Name:	Vladyslav Bril	Student number:	x21102872
Company:	Intercept Technologies	Month Commencing:	October

The second month of the industrial internship continued with the preparation for passing both the NSE 2 and NSE 3 certifications which were eventually acquired by the end of the month after successfully completing the online exams on the Partner Learning Platform from Fortinet.

Furthermore, I continued attending the personal meetings organized by CheckPoint to familiarize myself with the rest of the products that the company's clients highly utilize. I had both theoretical and practical exposure to such technologies as SASE, ZTA/ZTNA, different types of Firewall appliances, SD-WAN, SOAR, E-Mail SEG, and more.

I also began working with one of the clients from Intercept Technologies that required both Endpoint and Email protection for their working devices, as well as the MSP/MSSP. The rest of the month was spent installing, configuring, and monitoring the work of the CheckPoint solutions in terms of asset security (Harmony Endpoint Protection).

As a result, the majority of the devices that the company was utilizing had security soft installed and configured which guaranteed a secure workflow on those machines.

Finally, after starting to develop the main concept for the module assignment and conducting dynamic security testing (DAST) on the web application using the OWASP ZAP scanner, I realized that the idea of automatic vulnerability remediation should take place in modern software release cycles.

Employer comments

Student Signature: [illegible] Date: 31/10/2022

Industry Supervisor Signature: Joe McCann Date: 31/10/2022

Student Name:	Vladyslav Bril	Student number:	x21102872
Company:	Intercept Technologies	Month Commencing:	November

The third month of the industrial internship kicked off with different workshops and fast-paced training from both Fortinet and CheckPoint, such as reducing complexity with Fabric Management Centre, constructing a secure SD-WAN Architecture, improving application access and security with Fortinet ZTNA, creating a comprehensive Security Fabric etc.

In terms of working with the company's clients, I continued providing assistance with the Endpoint protection software, as well as hardware and/or software-related queries, namely, DHCP/DNS-related issues that affected the network performance of the device, OS misconfiguration, software components updates, etc.

In addition, I was also issued with the task to track the POV process for the E-Mail security that took place on the base of the company I was assigned for. During the two-week POV, I was able to analyze the generated report regarding the attacks that were aimed at the company through email (phishing, malware, spam, etc.). As a result, over 120+ emails were identified as potentially dangerous emails, while some contained malware inside.

Finally, after previously conducting the DAST testing, I started working on the literature review to identify if there are any existing solutions that would match my topic which was related to the automatic remediation of DAST findings. Moreover, another search was aimed at pinpointing any existing DevSecOps model that would be able to fit such a realization.

Employer comments

Student Signature: [illegible] Date: 29/11/2022

Industry Supervisor Signature: Joe McCann Date: 29/11/2022

Student Name:	Vladyslav Bril	Student number:	x21102872
Company:	Intercept Technologies	Month Commencing:	December

During the last month of the industrial internship, I continued working with the endpoint protection software, providing all the assistance with the installation & configuration of the utilized software. Moreover, I had the opportunity to generate and analyze the report related to the work done by the e-mail security agent that was previously set up for the client company.

Speaking of the thesis project, I continued programming the implementation part of the work and defining additional criteria for evaluating the software component produced. As a result, by the end of the month, a fully working application was delivered as a part of the master's thesis. The last part of the work was related to conducting a set of testing activities to verify that the application is working as expected and does not generate any uncommon errors or additional vulnerabilities.

As for the last two weeks working in the company, I had a couple of overview sessions related to different platforms that deliver security features through a single cloud component (e.g., CrowdStrike).

Employer comments

Student Signature: [illegible] Date: 23/12/2022

Industry Supervisor Signature: Joe McCann Date: 23/12/2022