BUILDING AN AUTOMATED PIPELINE FOR SECURITY AND FUNCTIONAL TESTING ON WEBAPPLICATIONS

Realization

Bachelor Applied Computer Science

Thierry Eeman
R0242545

Academiejaar 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

TABLE OF CONTENTS

ABSTRACT

This thesis is written based on the internship of Thierry Eeman at Eurofins Digital Testing Hasselt and was created to graduate at Thomas More Hogeschool Geel, Belgium, as a bachelor Applied Computer Science. The subject of this internship is creating a link between automated functional testing of webapplications and cyber security testing and create an elegant solution to tackle both challenges at once using existing tools and frameworks.

First off, a scope needed to be set to fit the timeframe that would fit within the duration of the internship which was set to three months.

After determining the scope of the project, the first step was to perform a study and create an overview of all the components needed to deliver this project. After initial interviews with Pieter Meulenhoff, a cyber security expert who served as a project consultant and product owner, I was able to distill the requirements and MVP for the project. I then created the scope of the project in a document with a description of all the tasks at hand, a time table and also a number of deliverables linked to markers in the time table. This also resulted in a complete architectural overview of the components for this project.

After creating the whole architecture, the first step was to set up all the individual components and create configurations to let them work and communicate together. A Selenium Grid was needed to automate parallel testing. This Grid needed to pass all traffic via an intercepting proxy to the internet. This proxy would collect all data and make it accessible through an API, which allows any dashboarding tool to collect the data for display and processing.

Once all the systems were operational, I needed to pass actual test runs through the Grid and proxy. In order to do so, I created a Maven artifact that can be imported in other projects. The purpose of the library was to provide low intrusion in the existing codebase. Web drivers are usually created in one single line of code and my library provides a way to choose the right browser with the right settings in one line. Finally I used an existing internal test project on a web application.

1 Acknowledgements

First and foremost, I would like to express my sincere gratitude to Resillion. The company provided me with an excellent platform to undertake this internship project, and the opportunity to learn and grow within such a dynamic and progressive environment has been an invaluable experience.

To Martijn Degrève, the head of SQS, I extend my heartfelt thanks. Your incredible technical guidance has been instrumental in my understanding of writing efficient tests. Your passion and knowledge have greatly enriched my internship experience.

My appreciation extends to Tiziana Treccase, a part of the HR team who showed immense interest in our progress and consistently ensured our well-being throughout the project. Your support has been a source of comfort and motivation, making the internship experience more rewarding.

I am profoundly thankful to Pieter Meulenhoff, advisor at Eurofins Cyber Security. As both product owner and technical support, your invaluable insights on security and the resources you provided played a pivotal role in navigating the complexities of the project. Your mentorship has deepened my understanding of cyber security, for which I am immensely grateful.

I would like to express my sincere thanks to Caroline Vanderheyden, our internship supervisor, for her constant support and guidance. Your regular check-ins and preparation for the final jury presentation were instrumental in our progress and ensured a smooth journey throughout the internship.

Last but not least, I would like to acknowledge Jonas Claes, my fellow student and a hub of knowledge, whose insights and assistance in troubleshooting during various stages of the project have been invaluable. Your camaraderie and support made the process much more engaging and enjoyable.

Each of you has contributed to making my internship a rich and enlightening experience, and for that, I am deeply grateful.

2 TERMINOLOGY

CSRF	Cross-Site Request Forgery, an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated
ZAP	Zed Attack Proxy
CI	Continuous Integration
CD	Continuous Deployment
Npm	Node Package Manager
OWASP	Open Web Application Security Project
Proxy	a type of software testing that checks if a system meets specified requirements
Functional testing	a method of testing that systematically tries all possible inputs or combinations
Brute force testing	a method of testing that systematically tries all possible inputs or combinations
API	Application Programming Interface, which allows different software applications to communicate and interact with each other

3 FIGURES AND DIAGRAMS

4 INTRODUCTION

In the ever-evolving technological landscape, software systems have become an integral part of daily life, underpinning critical infrastructure, finance, healthcare, and more. Ensuring the reliability and security of these systems is not just desirable, but essential. This thesis focuses on the integration of security vulnerability testing into an existing automated functional testing project — a step that is crucial in building robust, secure, and reliable software systems.

Functional testing and security testing each play a pivotal role in the software development lifecycle. Functional testing verifies that each function of the software operates in conformance with the requirement specification. This process ensures that the software system is doing what it is supposed to do and that all components are working together harmoniously. It is a critical step in maintaining high software quality and in preventing defects that could lead to system failures, data loss, or user dissatisfaction.

On the other hand, security testing aims to uncover vulnerabilities, threats, risks in a software system that could potentially be exploited. As the impact of security breaches continues to escalate—both in terms of financial costs and reputation damage—the importance of security testing cannot be overstated. A solid security testing process helps in identifying weak spots in a system's security framework and in taking the necessary preventive measures.

Resillion, with its two branches in Hasselt, Belgium and Groningen, The Netherlands, has the unique opportunity to bridge the gap between functional and security testing. The Belgium branch's expertise in functional testing and the Holland branch's specialization in security testing can be leveraged to create a more comprehensive, holistic approach to software testing.

By integrating these two domains, Resillion can ensure not only that software functions as expected but also that it is secure from potential threats. This would minimize the risk of security breaches and improve overall software quality. The collaboration between these two branches would foster knowledge sharing and skill enhancement, enriching the expertise within the company.

Moreover, the integration would lead to efficiency in testing processes and reduce costs. By incorporating security testing within the functional testing framework, the company could save resources that would otherwise be spent conducting these tests independently. This synergy between the departments would also streamline communication and coordination, reducing the potential for misunderstandings or missed vulnerabilities.

This internship sets out to explore the practical and theoretical aspects of this integration, aiming to provide a framework that can be used as a guideline for similar projects in the future. By doing so, I hope to contribute to the creation of software systems that are not only functional but also secure, furthering the mission of producing reliable, trustworthy technology for all users.

5 PRESENTING RESILLION

Figure 1 - Logo Resillion

Resillion is a global leader in testing and quality engineering services, with a focus on cyber security, digital media content testing, and quality assurance. The company was born from the expertise of Eurofins Scientific's Digital Testing, Cyber Security, Digital Forensics, and Content divisions, bringing together over 700 experts passionate about making IoT work and delivering top-tier testing technologies. They provide end-to-end capabilities regardless of your industry or stage in the digital journey, promising to guide you through to market.

The expert team of Eurofins Digital Testing, consisting of specialists in various disciplines, works closely with clients to understand their specific needs and offer customized solutions that ensure optimal performance, safety, and efficiency of their digital systems. This is achieved by delivering a wide range of services and solutions aimed at maintaining the highest quality standards in the industry.

A crucial aspect of Eurofins Digital Testing's work is designing and conducting tests to assess the performance and safety of digital products and services. This could, for example, involve testing the speed at which a device connects to the internet, or evaluating the security measures taken to protect sensitive data from unauthorized access.

In addition, Eurofins Digital Testing offers quality assurance services that help companies identify and resolve any issues that may arise during the development and implementation of their digital products and services. This can range from analyzing the usability of an application or website, to checking the compatibility of a device with various operating systems and networks.

As part of their comprehensive service offering, Eurofins Digital Testing also assists companies in complying with national and international regulations and standards, and in developing products and services that are compatible with existing systems and devices. This allows companies to remain competitive in an ever-changing market and adapt to the growing demands of consumers and business customers.

In terms of Cyber Security, the company's Netherlands branch is at the forefront of providing comprehensive cyber security services. They strive to protect, detect, respond, and recover from evolving threats to safeguard organizations. Their team of experts use cutting-edge tools and methodologies to identify vulnerabilities, provide solutions, and improve & integrate cyber security within organizations. This covers a range of activities across the lifecycle, such as security assessments, risk management, managed services, emergency response, and security engineering.

6 PROJECT SCOPE

6.1 Description

Title

BUILDING AN AUTOMATED PIPELINE FOR SECURITY AND FUNCTIONAL TESTING ON WEBAPPLICATIONS

Project Justification

During my internship at Resillion, I will work on a project to integrate functional and security testing within the CI/CD pipeline and develop a comprehensive dashboard for presenting test results. This integration aims to enhance efficiency, optimize resources, and improve the quality and security of software applications.

In the current setting, functional and security testing are often conducted separately, which can lead to inefficiencies and potential overlook of certain vulnerabilities. Manual investigation by security testers is time-consuming and can distract from more complex tasks. Additionally, current reporting methods may not be sufficient to effectively expose and communicate vulnerabilities.

Our proposed solution involves leveraging automation to integrate functional and security testing in the CI/CD pipeline. We also aim to develop a more sophisticated dashboard and reporting application that exposes vulnerabilities detected during test runs in an intuitive and user-friendly manner.

6.2 Timetable

Timetable

My internship project at Resillion will be structured into six distinct phases, each with its own objectives and duration. This systematic approach will ensure the effective execution of tasks and the timely delivery of project milestones.

In phase 1, which will last for a week, I will meet my two mentors who will guide me through the crucial aspects of the project, namely test automation and cyber security. We will establish a schedule for weekly meetings, allowing for consistent follow-up and feedback. Together, we will define the scope of the project and determine the minimum viable product (MVP) to be delivered by the end of the internship.

Phase 2, spanning over two weeks, will be dedicated to familiarization and investigation. I will familiarize myself with the tools used internally at Resillion to understand the proper workflow. I will also delve into the field of cyber security, particularly Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), to decide how best to approach the project. Furthermore, I will explore automated test frameworks and their potential interaction with vulnerability scanning.

During the two-week phase 3, I will experiment with various existing applications or solutions to understand what works well and what doesn't. I will select the appropriate tools for the project and develop a theoretical architecture that will allow the different components to interact with each other. I will then visualize this architecture and its components in a schematic diagram.

In phase 4, lasting another two weeks, I will focus on configuration and exploration. I will set up the basic building blocks of the project and explore their capabilities. I will investigate how vulnerabilities are captured, including their format and information, and how this can be retrieved via an API. I will also learn how to address these components through code and conduct tests on a dummy website.

Phase 5 will be about upscaling and refinement over a period of three weeks. I will scale the architecture from a local setup to a server environment, rewrite code to make it expandable and easy to implement beyond the initial setup, and look for an existing Resillion project to test the scanning process. This phase will also involve further configuration of containerized applications to suit a server environment.

Finally, in phase 6, also spanning three weeks, I will implement and evaluate the setup. I will test it on an existing Resillion project, evaluate various vulnerability dashboarding tools for visualization, and streamline the entire project in a Continuous Integration/Continuous Deployment (CI/CD) pipeline. Lastly, I will create documentation on the repository to ensure the sustainability and scalability of the project.

Through this structured and phased approach, I am confident I will be able to accomplish my project goals and deliver a viable product by the end of my internship at Resillion.

6.3 Deliverables

Deliverables	

1. **Project Scope**

Will consist of:

- *Project title*
- *Project justification*
- *Project scope*
- *List of deliverables*

2. **Security research document**

Will consist of:

- *SAST & DAST research*
- OWASP and CWE numbers research

3. **Project Architecture**

Will consist of:

- Architectural schematic
- Declaration of the interaction of the separate components
- Used technologies list

4. **Local application containers and Java implementation**

Will consist of:

- Docker compose files for the different components
- Java Classes

5. **Scalability through generalization**

Will consist of:

- *Maven Artifact to use library in other projects*
- Java Classes with builder pattern for better usability

6. **Dashboarding tools research**

Will consist of:

- Dashboard tool requirements
- Comparison between existing applications

7. CI/CD pipeline integration

Will consist of:

- Deployment through Ansible
- Release pipeline on Azure Devops

8. End report

Will consist of:

- *Scope of the project*
- *Project Architecture*
- *Local application containers and Java implementation*
- *Scalability through generalization*
- *Dashboarding tools research*
- *CI/CD pipeline integration*
- *Bibliography*

7 BUILDING AN AUTOMATED PIPELINE FOR SECURITY AND FUNCTIONAL TESTING ON WEBAPPLICATIONS

7.1 Introduction

7.2 Used Technologies

7.2.1 Docker

Docker is a software platform that allows developers to package, distribute, and run applications in a containerized environment. Containers are lightweight, standalone executable packages that include everything an application needs to run, such as code, runtime, system tools, libraries, and settings.

Figure 2 - Logo Docker

Docker provides a standardized way to package and distribute applications, regardless of the underlying infrastructure. This makes it easier for developers to deploy their applications across different environments, such as on-premises, public or private cloud, or hybrid environments.

Docker also provides tools for managing containers, such as Docker Compose for orchestrating multiple containers, and Docker Swarm for clustering and scaling containerized applications.

Some benefits of using Docker include:

- **Portability**: Docker containers can be run on any platform that supports Docker, making it easy to move applications between development, testing, and production environments.
- **Consistency**: Docker containers ensure that applications run the same way across different environments, eliminating compatibility issues and improving reliability.
- **Efficiency**: Docker containers are lightweight and require fewer resources than traditional virtual machines, enabling better resource utilization and cost savings.
- **Scalability**: Docker containers can be easily scaled up or down to meet changing demand, making it easy to manage and scale applications.

Overall, Docker provides developers with a powerful set of tools for building, packaging, and deploying applications, making it easier and more efficient to develop and manage modern applications.

The key components of a Docker container are:

- **Docker image**: A Docker image is a read-only template that contains instructions for creating a Docker container. It includes the application code, runtime, system tools, libraries, and dependencies needed to run the application.

- **Container Runtime**: The Docker runtime is responsible for running the Docker container. It is the process that creates, starts, stops, and manages the lifecycle of the container.
- **Filesystem**: A Docker container has its own filesystem, which is isolated from the host operating system. The filesystem contains the application code and all its dependencies.
- **Network**: Docker containers have their own network interface, which allows them to communicate with other containers and the outside world. Each container has a unique IP address, and Docker provides various network options for connecting containers.
- **Environment Variables**: Environment variables are used to configure the behavior of a Docker container. They can be set at runtime and are used to pass configuration information to the application inside the container.
- **Docker Registry**: A Docker registry is a storage and distribution system for Docker images. It allows developers to share and distribute their Docker images with others, either publicly or privately.
- Overall, these components work together to create a self-contained, portable environment that allows applications to run consistently across different environments.

For this project, the use of more complex multi-container Docker applications is required. These will be created using a docker-compose YAML file. The basic setup of a docker-compose.yml file includes:

- **version**: The version of the Docker Compose file syntax being used. This should be the first line of the file and is required.
- **services**: A list of services to be run as containers, with each service representing a separate container. Each service has its own set of configuration options, such as the image to be used, ports to be exposed, environment variables to be set, and volumes to be mounted.
- **networks**: An optional section that defines custom networks to be used by the services.
- **volumes**: An optional section that defines named volumes or bind mounts to be used by the services.

7.2.2 Selenium

Figure 3 – Logo Selenium

Selenium is an open-source automation tool that is widely used for automating web applications. It allows you to automate web browsers like Chrome, Firefox, Safari, and others, to carry out functional and regression testing of web applications.

Selenium can interact with the web page like a user does. It can perform tasks such as clicking on buttons, filling out forms, and navigating through web pages. Additionally, Selenium can handle alerts, pop-ups, and multiple windows. It can also simulate user behavior, such as scrolling, hovering over elements, and dragging and dropping.

Selenium supports a variety of programming languages, including Java, Python, C#, Ruby, and JavaScript. This makes it a popular choice among developers and testers with different programming backgrounds.

Selenium is not limited to just automating web applications. It can also be used for web scraping, web crawling, and for automating repetitive tasks on the web.

Overall, Selenium is a powerful automation tool that enables developers and testers to improve the quality of web applications while reducing the time and effort required for manual testing.

XPath is a language used to locate elements within an HTML or XML document. It is commonly used in Selenium to identify and interact with web elements on a web page. XPath expressions can be used to select web elements based on their attributes such as ID, class name, name, text, or any other attribute that uniquely identifies the element.

When using Selenium, you can use XPath expressions to find elements on a web page using the find_element_by_xpath method. For example, if you want to locate a button with the text "Click me", you could use the following XPath expression:

```
//button[text()='Click me']
```

This expression will select all button elements that have the exact text "Click me". Once you have selected the element using an XPath expression, you can interact with it by clicking it, sending text to it, or performing any other actions that Selenium supports.

XPath can also be used to locate elements relative to other elements on the page, or to locate elements within a specific section of the page. This can be useful when working with complex web pages that have a large number of elements. In such cases, XPath can help you write precise and efficient selectors to locate the elements you need to interact with.

7.2.3 Selenium Grid

Figure 4 - Logo Selenium Grid

Selenium Grid is a tool that allows you to run Selenium tests across multiple machines and browsers in parallel. It allows you to distribute tests across multiple servers and run them simultaneously, which can significantly reduce the time it takes to execute your tests.

Selenium Grid consists of a hub and multiple nodes. The hub acts as a central point that manages the distribution of test cases to different nodes. Nodes are machines that run the Selenium WebDriver instances and execute the tests in different browsers and operating systems.

To use Selenium Grid, you need to set up a hub and at least one node. Once the hub and nodes are set up, you can configure your test scripts to communicate with the hub, which will route the test requests to the available nodes based on their capabilities. For example, if you have a test script that needs to run in Chrome, the hub will find a node that has Chrome installed and route the test to that node.

Selenium Grid also allows you to specify the number of instances of a particular browser to run in parallel on a single node. For example, if you have a test suite that requires 10 instances of Firefox, you can configure the node to run 10 instances of Firefox at the same time.

Overall, Selenium Grid is a powerful tool that enables you to run your Selenium tests on multiple machines and browsers in parallel, reducing the time required to run your test suite and improving your test coverage. It is particularly useful for organizations that need to test their web applications across multiple browsers and operating systems.

Figure 5 - Logo Java

7.2.4 Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It was developed by Sun Microsystems (now owned by Oracle Corporation) and was first released in 1995. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. This makes Java a portable language, as applications written in Java can run on various types of devices and operating systems.

Key Characteristics and Features of Java:

- **Platform Independent**: Java is platform-independent because when you compile a Java program, it is compiled into bytecode. This bytecode can be interpreted on any device with a JVM. This means you can write a Java program once and run it anywhere that has a JVM, a principle often summarized as "write once, run anywhere" (WORA).
- **Object-Oriented**: Java is an object-oriented language, which means it represents concepts as "objects" that have data fields (attributes) and associated procedures known as methods. Object-oriented programming is designed to reduce complexity and improve maintainability of code.
- **Strongly Typed**: Java is a strongly typed language. This means that every variable and expression has a type that is known at compile-time, and all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.
- **Multithreaded**: Java supports multithreading, which allows multiple sequences of code (threads) to run concurrently within a single program. This is a powerful feature for building complex, high-performance applications.
- **Secure**: Security is a key consideration in Java. The Java platform allows users to download untrusted code over a network and run it in a secure environment in which it cannot do any harm: it cannot infect the host system with a virus, cannot read or write files from the hard drive, etc.
- **Robust**: Java has strong memory management, automatic garbage collection, exceptions, type-checking at compile time, and the absence of explicit pointers, which all contribute to the robustness of Java applications.
- **Large Standard Library**: Java has a rich set of standard class libraries that provide powerful and flexible data structures, algorithms, and utilities.

Java is widely used in various domains like web applications, enterprise software, embedded systems, mobile applications (notably Android apps), and more. The longevity and popularity of the language mean that there is a large community of Java developers and a wealth of online resources for learning and troubleshooting Java programming.

7.2.5 Maven

Figure 6 - Logo Maven

Maven is a powerful project management tool that is primarily used for Java projects. Developed by the Apache Software Foundation, Maven is used for project build automation, dependency management, and documentation. Its default build lifecycle comprises stages like validate, compile, test, package, install, and deploy.

Here's a brief explanation of some of the key features of Maven:

- **Build Automation**: Just like other build tools like Ant and Gradle, Maven can be used to compile source code, run tests, package the compiled code into a JAR or WAR file, install the packaged code in the local Maven repository, and deploy the packaged code in a remote repository or a server.
- **Dependency Management**: One of the biggest advantages of using Maven is its dependency management. You can declare your project's dependencies (i.e., JAR files or libraries your project needs) in the pom.xml file, and Maven automatically downloads them from the central Maven repository and places them in your local repository. If the dependencies have their own dependencies (transitive dependencies), Maven resolves them too.
- **Convention Over Configuration**: Maven follows the principle of convention over configuration. This means Maven comes with sensible defaults for your project structure. For example, it assumes that your Java source code is in a directory named "src/main/java", tests are in "src/test/java", and so on. Because of these conventions, a Maven project can be set up quickly without much configuration.
- **Lifecycle**: Maven has a lifecycle for building projects. This includes phases like validate, compile, test, package, install, and deploy. When you run a Maven command, you're actually running a lifecycle phase. For example, when you run the "mvn install" command, you're running the install phase of the lifecycle. Each phase includes a series of goals, and these goals are responsible for specific tasks.
- **Plugins**: Maven uses plugins to perform its tasks. For example, the Compiler Plugin compiles source code files, the Surefire Plugin runs tests, and the JAR Plugin packages compiled code into a JAR file. You can add and configure plugins

7.2.6 OWASP ZAP

Figure 7 - Logo Owasp Zap

The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular free, open-source web application security testing tools. It is actively maintained by hundreds of international volunteers and is a flagship project of the Open Web Application Security Project (OWASP), a worldwide not-for-profit charitable organization focused on improving the security of software.

ZAP is used for finding security vulnerabilities in web applications during the development and testing phase. It can also be used for manual security testing by developers and security professionals.

Here are some of the key features of OWASP ZAP:

- **Intercepting Proxy**: ZAP operates as a man-in-the-middle proxy, allowing you to intercept and modify the traffic passing between a client (usually your web browser) and a web application.
- **Automated Scanner**: ZAP can automatically scan applications for common security vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and others.
- **Passive Scanner**: Without actively probing or attacking the application, ZAP's passive scanner can analyze traffic to and from the application and identify potential security issues.
- **Spidering and AJAX Spidering**: ZAP can crawl an application to discover its structure and content, including AJAX-based content, thereby identifying additional areas for testing.
- **Forced Browsing:** ZAP includes a fuzzer, which can be used for brute-force testing of application parameters to identify hidden files and directories.
- **WebSockets**: ZAP supports WebSocket communication and allows users to view, intercept, and modify WebSocket messages.
- **Scripting**: ZAP supports various scripting languages, enabling customization of scanning rules, active and passive scans, and more.
- **Port Scanning**: It can also perform port scanning to identify additional attack surfaces.
- **Authentication and Session Support**: ZAP can understand and manipulate application-level authentication and session management, allowing it to perform access control testing.
- **EST API**: ZAP provides a REST API that allows you to interact with ZAP programmatically, integrating it into your continuous integration/continuous deployment (CI/CD) pipelines.

The ZAP tool is widely used by application developers, quality assurance testers, and professional penetration testers. It's also a good tool for those new to web application security and provides a wide range of resources, including a comprehensive wiki, a broad selection of guided videos, and a user group for help and support.

OWASP stands for the Open Web Application Security Project. It is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted. OWASP's mission is to make software security visible, so that individuals and organizations worldwide can make informed decisions about true software security risks.

OWASP is a non-profit organization and all of its materials are available under a free and open software license. It provides unbiased, practical, cost-effective information about computer and Internet applications, and it has become a reputable source of information about application security.

Here are some of the main activities and resources provided by OWASP:

- **OWASP Top 10**: Perhaps the most widely known OWASP project, the OWASP Top 10 is a regularly-updated report outlining the ten most critical security risks to web applications. It serves as a starting point for organizations aiming to improve their application security.
- **Software Tools and Documentation**: OWASP develops and provides free tools, standards, and guidelines that are used by organizations around the world to improve the security of their software. Examples include the ZAP (Zed Attack Proxy), a popular tool for identifying vulnerabilities in web applications, and the Application Security Verification Standard (ASVS), a framework for security requirements.

- **Community**: OWASP promotes a global community approach to tackling application security. Its community includes corporations, educational organizations, and individuals from around the world. It organizes local chapters, conferences, and seminars where security professionals can share their knowledge and experiences.
- **Education**: OWASP has a significant educational focus. It provides training, tutorials, videos, and other educational materials to help individuals and organizations understand and improve application security.
- **Research**: OWASP also engages in research into application and web security, helping to advance the industry's knowledge and capability in these areas.

In summary, OWASP is a critical resource in the world of application security, providing tools, documentation, and community resources to help improve the security of software worldwide.

7.2.7 Ansible

Ansible is an open-source software provisioning, configuration management, and application-deployment tool. It was developed and is maintained by Red Hat. Ansible is designed to help automate the often complex process of deploying and managing software, especially in a large, networked environment.

ANSIBLE

Figure 8 - Logo Ansible

Here are some key points about Ansible:

- **Provisioning**: Ansible can set up the various servers your applications need to run on.
- **Configuration Management**: Ansible can manage the configuration of your servers, ensuring they maintain the desired state. It can install or update software, start or stop services, or apply system updates across all your servers, among other tasks.
- **Application Deployment**: Ansible can deploy applications to your servers, ensuring that all dependencies are managed and the correct versions of each piece of software are used.
- **Automation**: Ansible lets you automate all these processes, which not only saves time and effort but also ensures consistency and reduces the potential for human error.
- **YAML for Playbooks**: Ansible uses a simple language (YAML, in the form of Ansible Playbooks) that allows you to describe automation jobs in a way that approaches plain English.
- **Agentless**: Unlike some other configuration management tools, Ansible is agentless, meaning you don’t need to install any additional software on the nodes that Ansible manages. It communicates over SSH for Unix-based systems or WinRM for Windows systems and pushes small programs called "Ansible modules" to the nodes.
- **Idempotency**: Ansible tasks are idempotent, meaning you can run the same tasks multiple times but the outcome will always be the same, i.e., they will bring the target system to the desired state without overwriting the existing state if it's already in the desired state.
- **Large Collection of Modules**: Ansible comes with a large set of modules (small, reusable programs) that can be used to manage various

parts of your system, such as files, databases, cloud infrastructure, network devices, etc.
- **Inventory**: Ansible uses an inventory file to track which servers it manages. This file can be in various formats, like INI or YAML.

Ansible is particularly popular for its simplicity and ease of use, as well as its ability to enable infrastructure as code (IaC). This allows system configurations to be versioned and treated as you would any other code, facilitating collaboration and maintaining a historical record of changes.

7.2.8 Defect Dojo

DefectDojo is an open-source application that was developed by the security team at Pearson and is currently maintained by the OWASP Foundation. It is a tool designed to help manage security testing efforts, specifically in the field of application security.

Figure 9 - Logo Defect Dojo

DefectDojo allows you to manage your application security testing program from a single place. It integrates with various scanning tools and provides a centralized space to manage the results. It is specifically designed to facilitate security testing throughout the software development lifecycle (SDLC).

Here are some key features of DefectDojo:

- **Import Scanning Results**: DefectDojo can import results from a wide range of security tools, including static analysis tools, dynamic analysis tools, dependency checkers, and more. This allows you to bring together results from multiple tools in one place.
- **Track and Manage Findings**: DefectDojo helps you manage and track the findings from your security tools, keeping all your application security information in one place. You can assign, track, and get the status of findings.
- **Product Metrics**: DefectDojo provides metrics and trend analysis, which can help you understand the security posture of your applications over time. This can provide valuable insights and help you make informed decisions about where to focus your security efforts.
- **Risk Management**: It allows risk-based prioritization of findings, which can be helpful in determining what issues to tackle first.
- **Integration with JIRA**: DefectDojo has built-in integration with JIRA, which means you can create and update JIRA tickets directly from within the tool.
- **Test Management**: DefectDojo can be used to manage security tests for each of your software releases, keeping a record of what tests were performed and what findings were identified.
- **Endorsements and Reviews**: Users can endorse findings, and findings can be peer reviewed with a history of the conversation and changes.

DefectDojo is a tool designed to be used by security teams, developers, and managers to facilitate the tracking and resolution of identified security issues. It is flexible and can adapt to various types of security testing methodologies.

7.2.9 Azure Devops

Azure DevOps is a suite of development collaboration tools created by Microsoft. It provides an integrated set of features that you can access through your web browser or IDE client. It was formerly known as Visual Studio Team Services (VSTS) but was rebranded as Azure DevOps in September 2018.

Figure 10 - Logo Azure Devops

The Azure DevOps suite consists of several key features:

- **Azure Boards**: This is a powerful work tracking system that uses the Scrum and Kanban methodologies. It provides a rich set of capabilities including native support for Scrum and Kanban, customizable dashboards, and integrated reporting. It can track tasks, bugs, and features to support project management and agile software development.
- **Azure Repos**: This is a version control system that provides two types of version control: Git and Team Foundation Version Control (TFVC). It's a set of version control tools that can track code changes and help manage code history.
- **Azure Pipelines**: This is a continuous integration (CI) and continuous delivery (CD) platform that can automatically build and test your code project and make it available to other users. It supports most popular languages and project types, and can deploy to a variety of targets including Azure itself, other cloud platforms, on-premises systems, or even mobile app stores.
- **Azure Test Plans**: This is a solution for tests and capturing data about defects. It provides a comprehensive toolkit for planning, tracking, and discussing tests within your team.
- **Azure Artifacts**: This is a package management resource that allows teams to share Maven, npm, and NuGet packages. It integrates with your CI/CD pipeline for easy use and access.

These tools are deeply integrated and extensible, and they cover the entire end-to-end development lifecycle. They can be used together for a unified experience, or individually based on the needs of the team or organization. Azure DevOps supports both public and private cloud configurations and is used by developers and businesses of all sizes.

7.3 Orientation and Security Research

Since I walked into this project without any prior security knowledge, it was mandatory for me to gather some information on the subject. I asked a few CCS students at Thomas More for valuable resources

SAST, or Static Application Security Testing, is a type of security testing that is performed on a static code base (i.e., the code is not being executed). SAST is often referred to as "white box testing" because it requires knowledge of the code being tested. The goal of SAST is to analyze the source code, byte code, or binary code of an application for security vulnerabilities. It can identify issues such as input validation errors, code injection vulnerabilities, and more. The advantage of SAST is that it can find vulnerabilities early in the development lifecycle, when they are typically cheaper and easier to fix. Examples of SAST tools include SonarQube, Checkmarx and Veracode Static Analysis

DAST, or Dynamic Application Security Testing, is a type of security testing that is performed while the application is running in its operational state. DAST is often referred to as "black box testing" because it does not require knowledge of the underlying code. DAST tools interact with the application, mimicking the behaviors of an attacker, in order to identify security vulnerabilities. These can include issues such as cross-site scripting (XSS) vulnerabilities, SQL injection vulnerabilities, and more. Examples of DAST tools include OWASP ZAP, Nessus and Acunetix.

It's important to note that SAST and DAST are complementary techniques. SAST is typically used early in the development process, while DAST is used after the application has been deployed. Using both approaches together provides a more comprehensive assessment of an application's security.

7.4 Choosing the right vulnerability scanning tool

All of these tools are used in the domain of cybersecurity but serve different purposes. Here's a brief comparison based on their primary uses, characteristics, and licensing:

- OWASP ZAP (Zed Attack Proxy):
    - Use: Primarily used for finding vulnerabilities in web applications. It includes functionalities for both automated and manual testing.
    - Characteristics: Features include intercepting proxy, automated scanner, passive scanner, spidering, forced browsing, WebSockets support, and more.
    - Licensing: It is a free and open-source tool maintained under the OWASP foundation.
- Burp Suite:
    - Use: Primarily used for testing web application security. It's somewhat similar to OWASP ZAP but often regarded as more advanced and feature-rich, which makes it popular for professional use.
    - Characteristics: Key features include an intercepting proxy, web application crawling, advanced scanning capabilities, and various tools for manual testing.
    - Licensing: There is a free (community) version available with limited features. The professional version, with all features and capabilities, is a paid tool.
- Metasploit:
    - Use: Mainly used for executing exploit code against target machines, usually to verify vulnerability. It's a penetration testing tool that helps validate vulnerabilities that have been discovered.
    - Characteristics: It offers payload creation, exploit execution, and shellcode generation. Metasploit is also widely used for developing, testing, and executing exploit code.
    - Licensing: It has both a free version (Metasploit Framework) with basic features and a paid version (Metasploit Pro) with advanced features.
- Tenable Nessus:

    - Use: It's a vulnerability scanner that scans networks to identify vulnerabilities that could be exploited by attackers. Nessus supports more than 100,000 known vulnerabilities.
    - Characteristics: Nessus provides features like high-speed asset discovery, configuration auditing, target profiling, malware detection, sensitive data discovery, and more.
    - Licensing: It is a commercial product with a paid license, although a limited free version (Nessus Essentials) is available for personal use.

In summary, while these tools all work in the field of cybersecurity, they each have their own strengths and uses. OWASP ZAP and Burp Suite are mainly used for web application security testing, Metasploit is best for executing exploit code, and Nessus is an industry standard for network vulnerability scanning. The choice of tool depends largely on the specific needs of the task at hand.

Alternatives for OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is a popular open-source tool for penetration testing and finding vulnerabilities in web applications. If you're looking for alternatives, there are several other tools that you might consider, depending on your specific needs:

- Burp Suite: This is one of the most popular alternatives to ZAP. Burp Suite is a web application security testing toolset. Like ZAP, it includes an intercepting proxy and automated scanner, but it also has numerous other features such as intruder (for automated custom attacks), repeater (for manipulating and resending individual requests), and sequencer (for analyzing session tokens). It has a community edition with limited features and a professional edition with full features.
- Netsparker: Netsparker is a web application security scanner focused on automatically detecting SQL Injection and Cross-site Scripting (XSS) vulnerabilities, among others. It's known for its Proof-Based Scanning technology, which can verify vulnerabilities, reducing false positives.
- Acunetix: This is a fully automated ethical hacking solution that mimics a hacker to keep one step ahead of malicious intruders. Acunetix can detect a large variety of security vulnerabilities, including SQLi, XSS, XXE, SSRF, and Host Header Attacks.

7.5 Setting up the architecture for the project

7.5.1 A schematic for the entire architecture

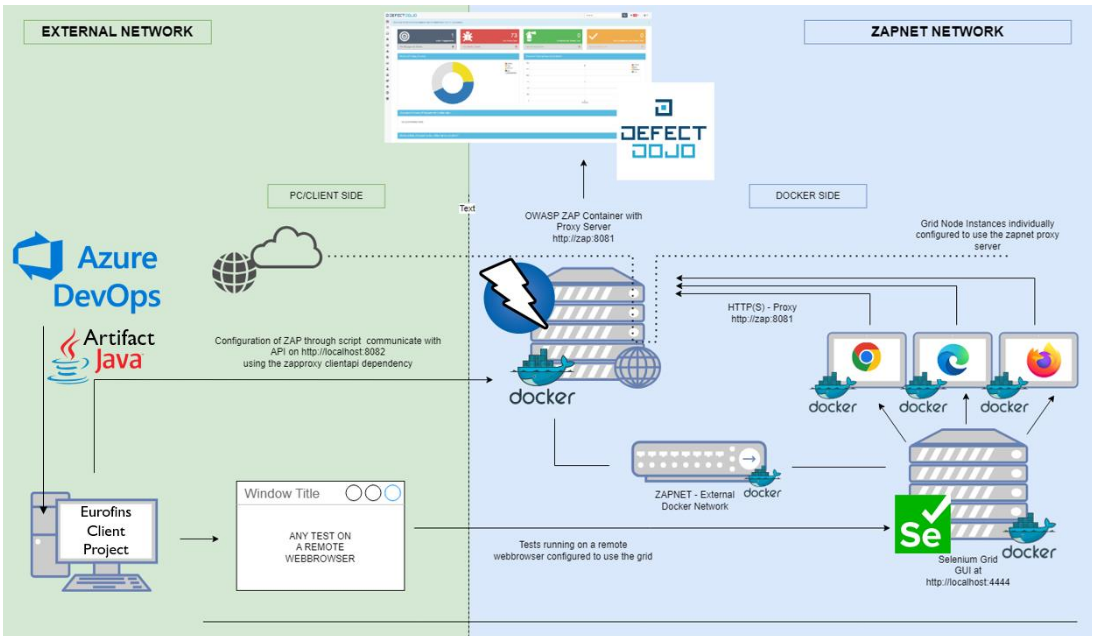

EXTERNAL NETWORK
ZAPNET NETWORK
DEFECT DOJO
PC/CLIENT SIDE
Text
DOCKER SIDE
OWASP ZAP Container with Proxy Server http://zap:8081
Grid Node Instances individually configured to use the zapnet proxy server
Azure DevOps
Artifact Java
HTTP(S) - Proxy http://zap:8081
Configuration of ZAP through script communicate with API on http://localhost:8082 using the zapproxy clientapi dependency
docker
docker
docker
docker
ZAPNET - External Docker Network
docker
Eurofins Client Project
Window Title
ANY TEST ON A REMOTE WEBBROWSER
Tests running on a remote webbrowser configured to use the grid
Se
Selenium Grid GUI at http://localhost:4444
docker

Figure 11 - Project Architecture Schematic

A full high resolution view of this schematic can also be found on my website alongside the other documents that were created for my internship project.

7.5.2 Full explanation of the schematic and the link between components

The architecture of the initiative is crafted in a manner that integrates two networks concurrently. On the one hand, we have the corporate network with designated IP addresses that facilitates internet access, the engagement with test project repositories, and the operation of all active applications. On the flip side, there is the bespoke 'zapnet' network, which has been configured in each of the Docker containers. This network, defined by software, enables seamless and secure communication among individual containers in an isolated environment.

A Resillion project repository will incorporate and employ a Maven artifact. This artifact enables the project to leverage the Selenium web drivers set up by this solution. The repository will instigate test suites via a pipeline, and these tests will be allocated to a web driver which is directed towards a Selenium Grid for execution in one of the nodes.

Each node has the capability to run browsers based on the preconfigured settings injected into the web drivers. For the OWASP ZAP to monitor the traffic and identify any vulnerabilities, the web drivers must be aligned to ZAP. Here, ZAP functions as a proxy server that channels all the traffic. This proxy does not impede the Selenium tests operating on the Grid. Rather, it simply allows all traffic to flow unimpeded and logs any alerts or findings in its session.

Upon completion of all tests, irrespective of their success or failure, the results will be documented in a ZAP scan XML file. This file can be imported into a dashboard for structured data viewing and relevance assessment. The automation of the ZAP session setup and result import should be executed within an automated deployment pipeline.

7.6 Application containers and Java implementation

7.6.1 Creating a Docker network

The first issue I need to address is to allow all the containers that I was going to configure to communicate with each other. Docker containers by default are isolated environments, which is great for general purposes. The type of complex solution I need to build will require data to persist and to be passed on to other containers to use or process. In order to achieve this, I need to create a Docker network.

A Docker network is a software-defined network that enables different Docker containers to communicate with each other. It also allows the outside world to communicate with containers and vice versa. Docker networks can span multiple machines, providing a lot of flexibility in how you can set up and structure your applications.

Here are some reasons why Docker networks are used:

- **Isolation**: Docker networks provide network isolation, so containers can have their own private networks, allowing you to better manage and secure your containers.
- **Communication**: Docker networks allow for communication between containers. By default, containers can communicate with each other if they are on the same network. This is crucial for multi-container applications where different components need to interact with each other.
- **Network Management**: Docker networks simplify the process of managing ports and network settings for your containers. Instead of manually managing ports, Docker networks can manage these settings for you, simplifying the setup process.
- **Interoperability**: Docker networks make it easier to ensure that your application works the same way on different systems. By using Docker networks, you can ensure that your application has the same network setup regardless of where it's deployed.
- **Service Discovery**: Docker provides a built-in DNS server for containers to use for automatic service discovery. When you create a container on a Docker network, it can access others by their container name.

By default, Docker provides several network drivers for different use cases:

- **Bridge**: The default network driver for a container. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
- **Host**: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly.
- **Overlay**: This network enables swarm services to communicate with each other.
- **Macvlan**: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network.

- **None**: This network disables all networking for a container.

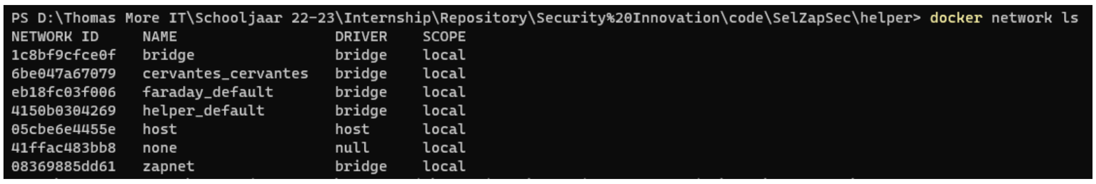
You can create a Docker bridge network using the Docker command-line interface (CLI) with the network create command and specifying the driver as bridge.

Here is the basic syntax:

docker network create -d bridge my_custom_network

In this command:

- **docker network create** is the command to create a new network.
- **-d bridge** specifies that the network driver type should be a bridge.
- **my_custom_network** is the name you want to give to your network.

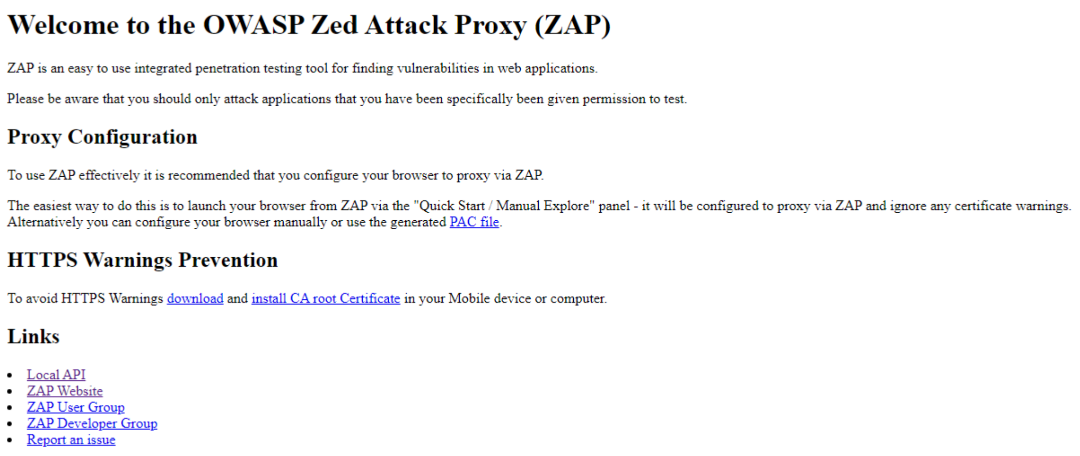


```
PS D:\Thomas More IT\Schooljaar 22-23\Internship\Repository\Security%20Innovation\code\SelZapSec\helper> docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
1c8bf9cfce0f   bridge                 bridge    local
6be047a67079   cervantes_cervantes    bridge    local
eb18fc03f006   faraday_default        bridge    local
4150b0304269   helper_default         bridge    local
05cbe6e4455e   host                   host      local
41ffac483bb8   none                   null      local
08369885dd61   zapnet                 bridge    local
```

Figure 12 - Docker Network

In this solution, the network is called 'zapnet' and it will be passed on to all the different docker compose files in order to open up communication for all the applications. This pre-existing network has to be specified in the compose file, with the additional rule 'external: true' in order for the containers to find it.

7.6.2 Configuring OWASP ZAP



Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

Proxy Configuration

To use ZAP effectively it is recommended that you configure your browser to proxy via ZAP.

The easiest way to do this is to launch your browser from ZAP via the "Quick Start / Manual Explore" panel - it will be configured to proxy via ZAP and ignore any certificate warnings. Alternatively you can configure your browser manually or use the generated PAC file.

HTTPS Warnings Prevention

To avoid HTTPS Warnings download and install CA root Certificate in your Mobile device or computer.

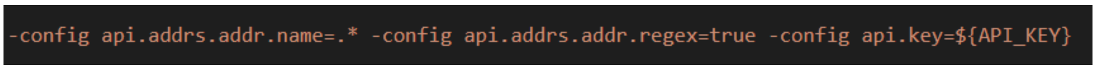
Links

- Local API
- ZAP Website
- ZAP User Group
- ZAP Developer Group
- Report an issue

Figure 13 - OWASP ZAP Headless mode GUI

Since the OWASP ZAP application sits at the heart of the architecture, it's crucial that this is operational and properly configured. Since these images already exist, a docker compose file with all the necessary configuration inside can be created. There are multiple ways to run a ZAP instance, but the ideal one for this scenario is headless mode. The reason for this is that the solution will not utilize the GUI from ZAP, it just needs receive all the findings and keep them in the database. While the ZAP application is running, it will provide an API endpoint with which configuration and extracting the findings is possible.

The command for setting up ZAP in headless mode has a few parameters, so when used in a Docker compose file, it needs to be added under the command tag in order to overwrite the default docker compose command. One of the items that is passed into this command is the API-key.



```
-config api.addrs.addr.name=.* -config api.addrs.addr.regex=true -config api.key=${API_KEY}
```

Figure 14 - OWASP ZAP API key

To secure this instance of ZAP, this API-key should not be hardcoded in the compose file, but passed on as a variable that can be retrieved from a .env file.

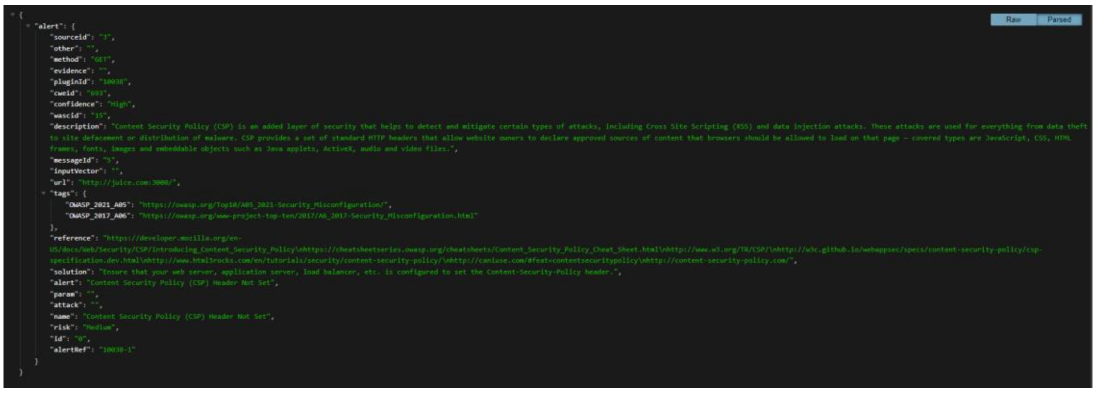
The safest way to ensure expandability in the future, is to run the headless command with xvfb.

Xvfb, or "X Virtual Frame Buffer", is a display server that performs all graphical operations in memory without showing any screen output. This is extremely useful for running applications that require a display, but don't actually need to show anything on a physical screen. In other words, it's a way to run graphical applications "headlessly".

When ZAP runs in headless mode, it doesn't need to display its graphical user interface. However, ZAP uses the Selenium browser automation framework for some of its features, and Selenium itself requires a display to operate, even if it's being run headlessly.

By using Xvfb, a virtual display can be set up in memory for Selenium to use, allowing ZAP to run fully headlessly and still take advantage of all its features. This can be particularly useful when running ZAP in an environment where there is no access to a physical display, such as a Docker container or a server without a graphical environment installed.

The Docker container seems to be unhealthy all the time. After some extra research I discovered that the health check will fail when running ZAP with a port other than the default 8080. In that case, the ZAP_PORT environment variable needs to be set in the environment file.



```
{
    "alert": {
        "sourceid": "3",
        "other": "",
        "method": "GET",
        "evidence": "",
        "pluginId": "10038",
        "cweid": "693",
        "confidence": "High",
        "wascid": "15",
        "description": "Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft
        to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML
        frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.",
        "messageId": "5",
        "inputVector": "",
        "url": "http://juice.com:3000/",
        "tags": {
            "OWASP_2021_A05": "https://owasp.org/Top10/A05_2021-Security_Misconfiguration/",
            "OWASP_2017_A06": "https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html"
        },
        "reference": "https://developer.mozilla.org/en-
        US/docs/Web/Security/CSP/Introducing_Content_Security_Policy\nhttps://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html\nhttp://www.w3.org/TR/CSP/\nhttp://w3c.github.io/webappsec/specs/content-security-policy/csp-
        specification.dev.html\nhttp://www.html5rocks.com/en/tutorials/security/content-security-policy/\nhttp://caniuse.com/#feat=contentsecuritypolicy\nhttp://content-security-policy.com/",
        "solution": "Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.",
        "alert": "Content Security Policy (CSP) Header Not Set",
        "param": "",
        "attack": "",
        "name": "Content Security Policy (CSP) Header Not Set",
        "risk": "Medium",
        "id": "0",
        "alertRef": "10038-1"
    }
}
```

Figure 15 - OWASP ZAP API json

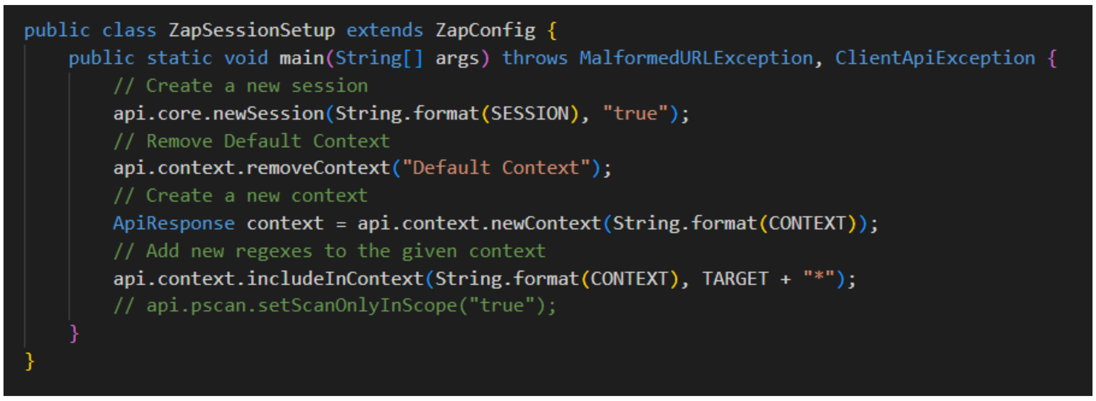
The ZAP API responds in json format, which is not ideal to work with in this case because the information is not structured well and does not allow for filtering. Further in this thesis, I will explore applications that can translate these ZAP reports and findings into structured and visually appealing dashboards.

One of the key features of ZAP is that it scans passively. This means that all the traffic that is being generated by the Selenium tests and web drivers will pass through the ZAP proxy and everything it discovers will be recorded in the application. This provides very good coverage but also a large problem: even websites and URLs that are not being tested directly can also show up in scan results. If a website utilizes Cloudflare or Stripe for instance, then those endpoints will also be visible in the ZAP API scan results. This inclusion then provides a problem for the active scanning, which will actively attack all discovered endpoints. This is unwanted behavior, as it will attack websites out of scope and those websites often have policies and countermeasures for attacks, which leads to more problems.

In order to counter this, I explored the documentation and the API and found a solution to this problem. I created two scripts that make some adaptations to the ZAP instance at runtime. The first script ensures that the proxy server is initialized. This is to make sure that all the Selenium tests will use this proxy. The second script contains session configurations for the ZAP application and sets a scope for the passive and active scan.

A session in ZAP represents all of the data related to a specific instance of ZAP usage. This includes all the requests and responses that have been sent and received, any alerts that have been generated, any scripts, breakpoints, and so on. Essentially, it's a record of everything that's happened in a particular use of ZAP. Sessions can be saved and loaded, allowing you to resume work at a later time or share your work with others. By default, ZAP uses an HyperSQL database to store the session data, but other database systems can be used if preferred.

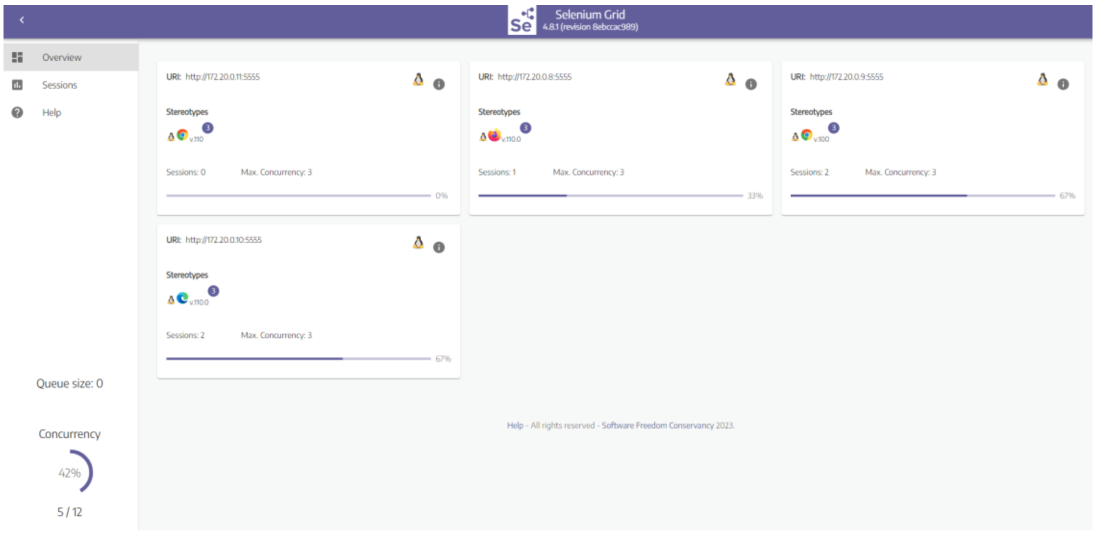
the concept of "scope" is used to limit the extent of actions that ZAP can perform. For example, you might want to limit ZAP's spidering or active scanning to only a certain subset of sites. This is useful for focusing on specific targets and ensuring that ZAP doesn't interact with systems or sites that you don't have permission to test. Scope is defined by a set of rules, such as which URLs or IP addresses are in or out of scope. These rules can be defined manually, or they can be automatically set based on certain criteria. Scoping ensures that you are only testing what you're supposed to be testing, and it helps to reduce noise in the results.



```
public class ZapSessionSetup extends ZapConfig {
    public static void main(String[] args) throws MalformedURLException, ClientApiException {
        // Create a new session
        api.core.newSession(String.format(SESSION), "true");
        // Remove Default Context
        api.context.removeContext("Default Context");
        // Create a new context
        ApiResponse context = api.context.newContext(String.format(CONTEXT));
        // Add new regexes to the given context
        api.context.includeInContext(String.format(CONTEXT), TARGET + "*");
        // api.pscan.setScanOnlyInScope("true");
    }
}
```

Figure 16 - Java Class to configure ZAP context

7.6.3 Creating a Selenium Grid



Selenium Grid
4.8.1 (revision 8ebccac989)

Overview
Sessions
Help

URI: http://172.20.0.11:5555
Stereotypes
v.110
Sessions: 0 Max. Concurrency: 3
0%

URI: http://172.20.0.8:5555
Stereotypes
v.110.0
Sessions: 1 Max. Concurrency: 3
33%

URI: http://172.20.0.9:5555
Stereotypes
v.100
Sessions: 2 Max. Concurrency: 3
67%

URI: http://172.20.0.10:5555
Stereotypes
v.110.0
Sessions: 2 Max. Concurrency: 3
67%

Queue size: 0

Concurrency
42%
5 / 12

Help - All rights reserved - Software Freedom Conservancy 2023.

Figure 17 - Selenium Grid

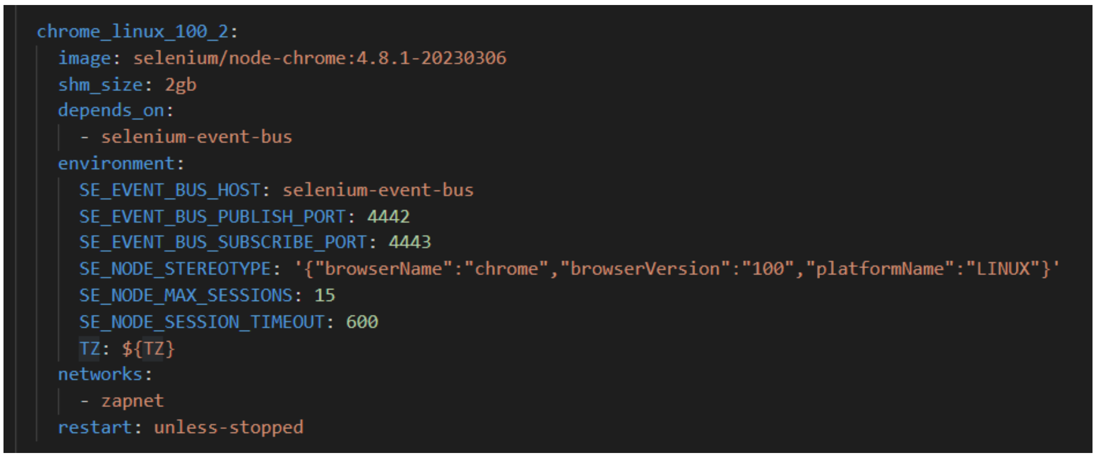
The purpose of the Selenium Grid is to allow all the tests to be ran simultaneously in parallel while taking advantage of the ZAP passive scanning abilities in order to capture vulnerabilities as the tests pass through the proxy. The project will utilize a full grid as it allows for much more configuration options. The docker compose file that starts the grid can be expanded further with additional nodes.

Each node represents a type of browser and version of that browser. The included browsers are Chrome, Firefox and Edge. There are a few issues that need to be addressed when configuring a grid like this.

There are several extra environment options that can be passed into each of the nodes that require some attention:

- **Stereotype**: In order to identify the nodes, the name, version and platform of the browser need to be specified
- **Max Sessions**: This is a tricky one. In theory this can be an infinite number. However, the grid will only deploy as many as there are CPU cores on the machine it uses.

- **Session Timeout**: This variable is normally not that important. It indicates when a session should be terminated by force when it takes too long to complete a certain test. In case of this project, some tests will take several minutes to complete (because of complexity or deliberate timeout). Therefore a higher timeout is needed for the tests to pass.
- **TZ** (Time zone): The default time zone is UTC, which is not ideal when running tests in other countries. Some tests utilize time/date and are prone to fail if the time zone is incorrectly set.



```
chrome_linux_100_2:
  image: selenium/node-chrome:4.8.1-20230306
  shm_size: 2gb
  depends_on:
    - selenium-event-bus
  environment:
    SE_EVENT_BUS_HOST: selenium-event-bus
    SE_EVENT_BUS_PUBLISH_PORT: 4442
    SE_EVENT_BUS_SUBSCRIBE_PORT: 4443
    SE_NODE_STEREOTYPE: '{"browserName":"chrome","browserVersion":"100","platformName":"LINUX"}'
    SE_NODE_MAX_SESSIONS: 15
    SE_NODE_SESSION_TIMEOUT: 600
    TZ: ${TZ}
  networks:
    - zapnet
  restart: unless-stopped
```

Figure 18 - Selenium Grid node Docker compose

Since Docker Desktop runs on Linux under the hood, it's impossible (or very hard and not worth the effort) to create grid nodes on any other platform than Linux. When nodes are created with other platforms, they will deploy on the grid, but none of them will be functional and all web drivers sent to them will fail.

7.6.4 Writing Java Classes to instantiate web drivers

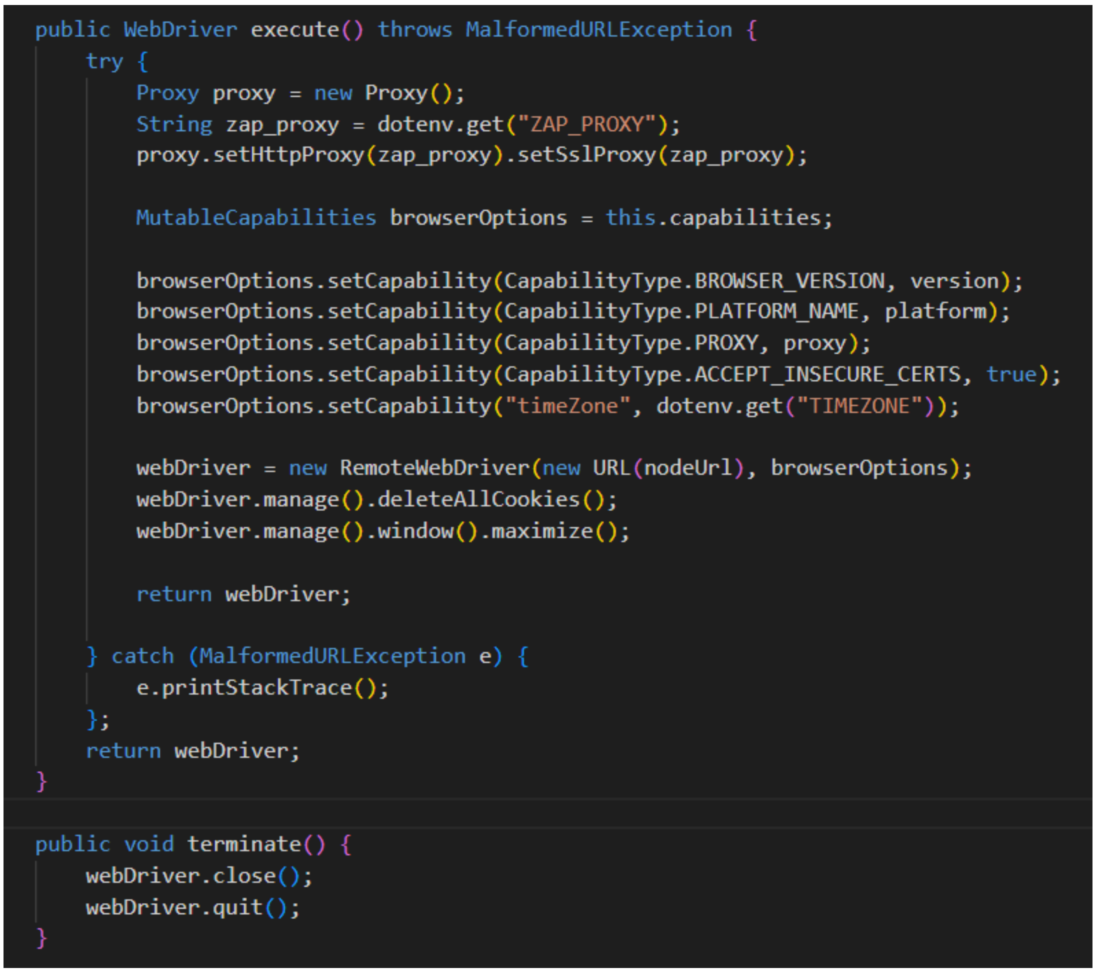
In order to create expandable and efficient code, it's necessary to look at what all the different web browsers have in common when it comes to configuration. That way a base web driver class can hold all this common information and any specific additional setup can be performed in a browser specific class. This took some time to figure out, but I was able to distill the common features for a basic allround webdriver.

When digging in the existing project repositories from Resillion, I noticed that they used the WebDriver class to create web drivers for Selenium tests. These open locally on your system, which is not solution I needed because the tests need to access web pages that are proxied by the instance of ZAP over the Selenium Grid. The solution to this was using the RemoteWebDriver class to instantiate these browsers, because it allows for test runs on remote machines, in this case the remote server.

In Selenium WebDriver, WebDriver is an interface, and RemoteWebDriver is a class that implements the WebDriver interface.

WebDriver is an interface in Selenium that contains the declaration of various methods like get(), findElement(), close(), etc. These methods provide the functionality for browser automation, such as navigating to a URL, finding elements, interacting with elements, and closing the browser.

RemoteWebDriver is a class that implements the WebDriver interface. This class is designed to handle remote connections, allowing tests to be run on a separate machine (the Selenium Server) rather than locally. RemoteWebDriver is the parent class for the browser-specific driver classes like ChromeDriver, FirefoxDriver, etc. It can be used to run tests on different browsers that are located on remote machines.



```java
public WebDriver execute() throws MalformedURLException {
    try {
        Proxy proxy = new Proxy();
        String zap_proxy = dotenv.get("ZAP_PROXY");
        proxy.setHttpProxy(zap_proxy).setSslProxy(zap_proxy);

        MutableCapabilities browserOptions = this.capabilities;

        browserOptions.setCapability(CapabilityType.BROWSER_VERSION, version);
        browserOptions.setCapability(CapabilityType.PLATFORM_NAME, platform);
        browserOptions.setCapability(CapabilityType.PROXY, proxy);
        browserOptions.setCapability(CapabilityType.ACCEPT_INSECURE_CERTS, true);
        browserOptions.setCapability("timeZone", dotenv.get("TIMEZONE"));

        webDriver = new RemoteWebDriver(new URL(nodeUrl), browserOptions);
        webDriver.manage().deleteAllCookies();
        webDriver.manage().window().maximize();

        return webDriver;

    } catch (MalformedURLException e) {
        e.printStackTrace();
    };
    return webDriver;
}

public void terminate() {
    webDriver.close();
    webDriver.quit();
}
```

Figure 19 - Java Class general RemoteWebDriver

Every web driver needs the customized proxy settings to point it to ZAP. Once this proxy is initiated, it can be added to the web driver options. Figuring out these was very tricky, as I needed a type of option that was applicable to all the different web browsers. In a more simplified approach, web drivers have their own options. A Chrome driver for instance, has ChromeOptions(). This cannot be generalized and opens the door for more complex code an code repetition.

So after digging into the official documentation of Selenium and Java web drivers, I found that Capabilities are what I was looking for.

Capabilities in a Selenium Java project are a set of key-value pairs that allow you to customize the behavior of the WebDriver during automated testing. They provide a way to specify properties of the browsers you want to use, such as browser name, version, and platform, among other things.

The capabilities are defined by using the DesiredCapabilities class in Selenium WebDriver. This class provides a series of static methods to create capabilities for different browsers (like Firefox, Chrome, Internet Explorer, etc.), and to set various browser properties.

The most important capabilities that I wanted to specify were:

- **browserName**: This capability sets the name of the browser to be used for the tests. This could be "firefox", "chrome", "safari", etc.
- **version**: This capability sets the version of the browser to be used.
- **platform**: This capability sets the operating system platform. This could be "WINDOWS", "MAC", "LINUX", etc.
- **proxy**: This capability sets whether the driver should use a customized proxy server.

However, this wasn’t working because it also didn’t generalize well enough. It worked fine for Firefox and Edge, but Chrome was not accepting the proxy settings I defined for it.

After more research and tryouts, I discovered that MutableCapabilities were giving me the results I needed.

MutableCapabilities is a base class introduced in later versions of Selenium to create more flexible capabilities. The MutableCapabilities class is an implementation of the Capabilities interface, and is the parent class of all browser-specific options classes such as ChromeOptions, FirefoxOptions, etc.

In the context of Selenium WebDriver, MutableCapabilities was designed to make it easier for users to set and manipulate browser-specific capabilities without having to directly interact with the DesiredCapabilities class. This is particularly useful as the trend is moving away from using DesiredCapabilities directly due to its static nature.

MutableCapabilities provides a flexible way to set browser-specific capabilities. It makes it easier to customize the behavior of the WebDriver and to adapt it to a wide range of testing scenarios.

7.7 Working on a live project

7.7.1 OWASP Juice Shop

Initially I conducted all tests for ZAP and the Selenium Grid on the OWASP Juice Shop, an open-source project developed by OWASP.

The OWASP Juice Shop is specifically a vulnerable web application designed to be used as a learning tool for security enthusiasts, developers, and students. The purpose of the Juice Shop is to educate about web application security in a fun and interactive way.

The application itself is a "juice shop" where users can browse products, leave reviews, and make purchases, much like a real online retail store. However, it is intentionally designed with many security flaws. These flaws range from simple ones that might be found in beginner-level tutorials, to complex, nuanced issues that even experienced professionals might struggle with.

Users are encouraged to identify and exploit these security issues, in a process known as penetration testing or "ethical hacking". This can help them understand how such vulnerabilities arise, how they can be exploited, and most importantly, how they can be prevented or fixed. The Juice Shop includes a scoreboard where users can check if they have found all the available vulnerabilities, providing a gamified learning experience.

It's worth noting that the Juice Shop is fully self-contained and can be run locally, so there's no risk of legal issues that might arise from attempting to hack a real web application without permission.

After the promising results of my Java classes and the Selenium Grid, I asked Martijn if I could start implementing this on a real project from Resillion. This would allow me to scale up the solution and encounter new problems to tackle, aiming for more generalization so that the solution could be expanded to other projects in the future.

7.7.2 Greenvalley Project

I searched and asked for a project that was also written in Java to easily translate and transform my existing solution to match the needs of this project and by extension all other Java projects. This project turned out to be Greenvalley. After getting access to the repository and the source code, I was able to start contemplating about a solution to elegantly introduce my selenium grid and web drivers into this project.

7.8 Scalability through generalization

7.8.1 Maven Artifact

In Azure DevOps, an artifact is a collection of files or packages produced by a build or a release during the process of software development. These artifacts serve as the dependencies of a deployment process or as a published outcome of your pipeline.

Artifacts can include:

- **Compiled code**: This is the code that has been compiled into a runnable state. This could be a .jar file for Java applications, a .dll file for .NET applications, or an .exe file for Windows applications.
- **Packages**: This refers to software packages that are ready to be deployed or distributed. This could include NuGet, npm, or Maven packages.
- **Test results**: The results of your automated tests can also be considered artifacts. These can include reports, logs, or any other files that were generated during testing.
- **Documentation**: If your build process generates documentation (like Javadoc or Sphinx), these files can also be considered artifacts.

In Azure DevOps, you can publish artifacts during a build using the Publish Build Artifacts task, and then use these artifacts in subsequent stages of your pipeline. In a release pipeline, you can consume these artifacts to deploy your application.

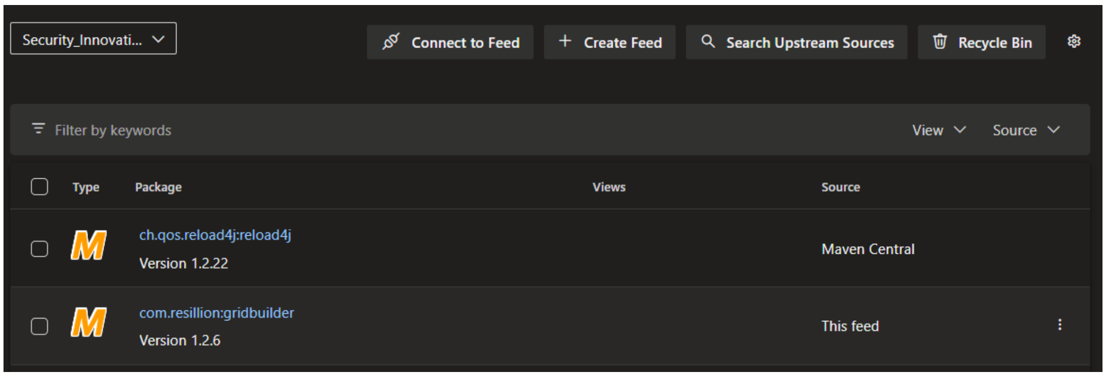
Also, Azure Artifacts is a feature of Azure DevOps that allows you to host and share packages, such as Maven, npm, NuGet, and Python packages, with your team. You can publish these packages to Azure Artifacts for sharing with your team or across your organization.

Creating a Maven artifact in an Azure DevOps repository allows you to use Azure DevOps as a private repository for your Maven packages, which can be beneficial in a few ways:

- **Version control**: You can keep track of different versions of your Maven packages and ensure that every team member is working with the correct version.
- **Security**: Hosting your Maven packages on a private repository like Azure DevOps means that you have more control over who can access and use your packages.
- **Ease of use**: Azure DevOps integrates well with many other tools, making it a convenient choice if you're already using Azure for other parts of your project.

Here are the general steps to create a Maven artifact in an Azure DevOps repository:

- Create a feed.
- Connect to the feed.
- Update your Maven settings: Copy the settings.xml provided by Azure DevOps and paste it into your Maven .m2 directory.
- Update your project's POM file: In your Maven project's pom.xml file, you need to add a <distributionManagement> section that points to your Azure DevOps feed.
- Deploy your Maven package.



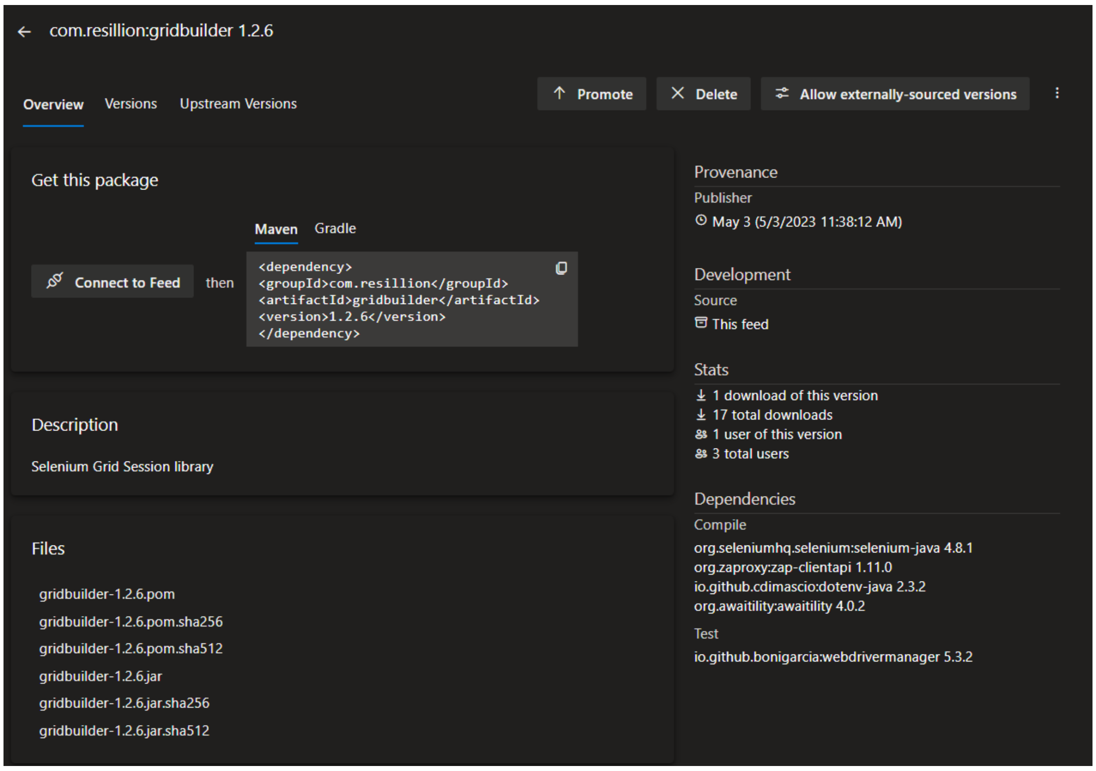
Type	Package	Views	Source
M	ch.qos.reload4j:reload4j Version 1.2.22		Maven Central
M	com.resillion:gridbuilder Version 1.2.6		This feed

Figure 20 - Maven Artifact (Azure Devops)

By creating an actual Maven package, I also had to learn about and follow the rules on semantic versioning, a versioning scheme for software that aims to convey meaning about the underlying changes in a release. It follows a version format of X.Y.Z (Major.Minor.Patch).

1. **Major (X)**: This number is incremented when there are incompatible changes in the API, meaning the software has changes that may break or are not backward-compatible with older versions. This often means that users will need to make changes in their existing setup to use this version. This was not necessary for my library.
2. **Minor (Y)**: This number is incremented when new features are added in a backwards-compatible manner. The software remains compatible with older versions, and users can use the new version without making any changes in their existing setup (though they might not be able to use the new features without some changes). I used this when adding the builder design pattern to the library.

3. **Patch (Z)**: This number is incremented when backwards-compatible bug fixes are introduced. These are usually small changes that fix errors or bugs in the software without adding any new features. These were constantly updated, whenever I fixed errors or made small changes to the code without introducing new features.


com.resillion:gridbuilder 1.2.6

Get this package

```
<dependency>
<groupId>com.resillion</groupId>
<artifactId>gridbuilder</artifactId>
<version>1.2.6</version>
</dependency>
```

Description

Selenium Grid Session library

Files

gridbuilder-1.2.6.pom
gridbuilder-1.2.6.pom.sha256
gridbuilder-1.2.6.pom.sha512
gridbuilder-1.2.6.jar
gridbuilder-1.2.6.jar.sha256
gridbuilder-1.2.6.jar.sha512

Provenance
Publisher
May 3 (5/3/2023 11:38:12 AM)

Development
Source
This feed

Stats
1 download of this version
17 total downloads
1 user of this version
3 total users

Dependencies
Compile
org.seleniumhq.selenium:selenium-java 4.8.1
org.zaproxy:zap-clientapi 1.11.0
io.github.cdimascio:dotenv-java 2.3.2
org.awaitility:awaitility 4.0.2
Test
io.github.bonigarcia:webdrivermanager 5.3.2

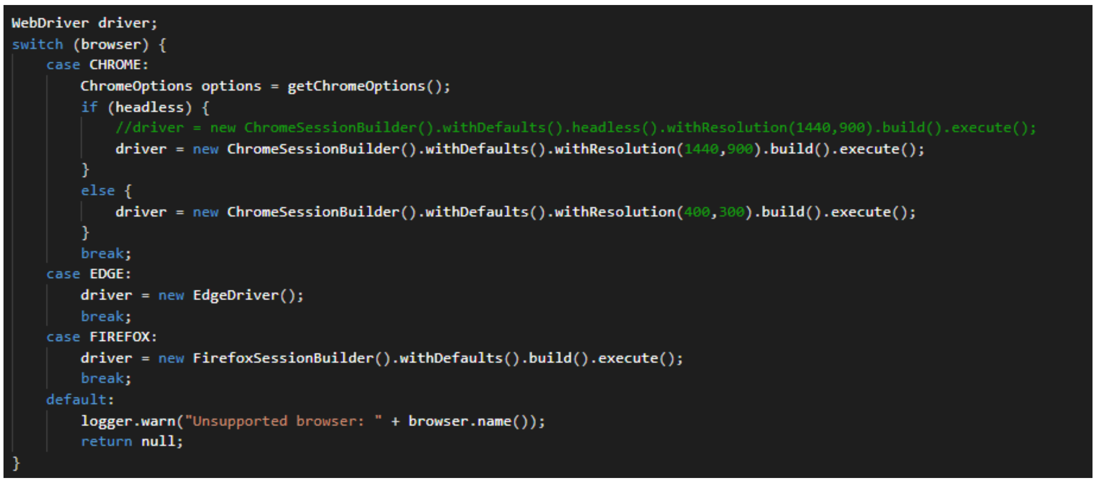
Figure 21 - Gridbuilder Artifact overview

7.8.2 Java builder pattern

I found myself struggling with the web drivers I created because, when implemented in an actual project with different cases, it became clear that flexibility was necessary to adapt to any given case instantly, without having to rewrite code or introducing complexity. Some features of a web driver needed to be added or removed as needed and weren't always necessary to begin with. It was at this point that I asked for some help from my mentor, who gave me valuable insights in the Builder design pattern.

The Builder pattern is a design pattern that provides a flexible solution to the problem of creating complex objects step by step. It falls under the category of "creational" design patterns. This pattern separates the construction of an object from its representation, allowing the same construction process to create different representations.

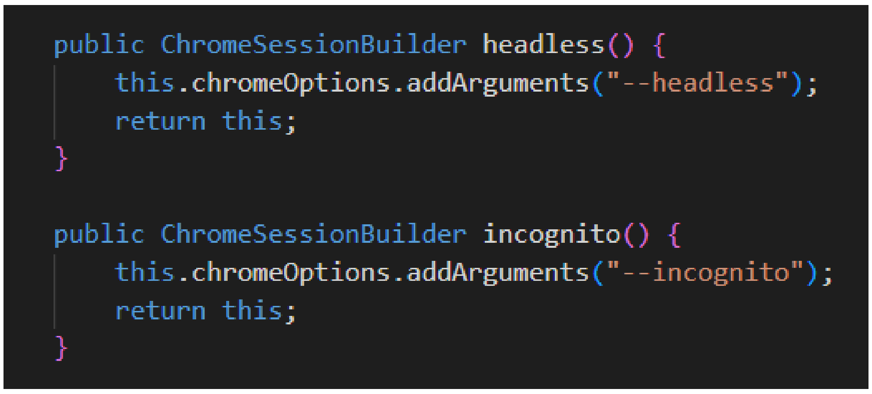
When constructing an object involves many steps, particularly if those steps don't have to be performed in a specific order or some steps are optional, the Builder pattern can be a good choice.



```java
WebDriver driver;
switch (browser) {
    case CHROME:
        ChromeOptions options = getChromeOptions();
        if (headless) {
            //driver = new ChromeSessionBuilder().withDefaults().headless().withResolution(1440,900).build().execute();
            driver = new ChromeSessionBuilder().withDefaults().withResolution(1440,900).build().execute();
        }
        else {
            driver = new ChromeSessionBuilder().withDefaults().withResolution(400,300).build().execute();
        }
        break;
    case EDGE:
        driver = new EdgeDriver();
        break;
    case FIREFOX:
        driver = new FirefoxSessionBuilder().withDefaults().build().execute();
        break;
    default:
        logger.warn("Unsupported browser: " + browser.name());
        return null;
}
```

Figure 22 – Builder design pattern examples

By adapting my code to this designer pattern, I could easily specify different scenarios. Sometimes the default settings were sufficient, in other cases the browser or version could be changed instantly. If a browser needed to be run in headless or incognito mode, the builder pattern allowed for complete flexibility in creating a RemoteWebDriver as it was needed to perform a given test.

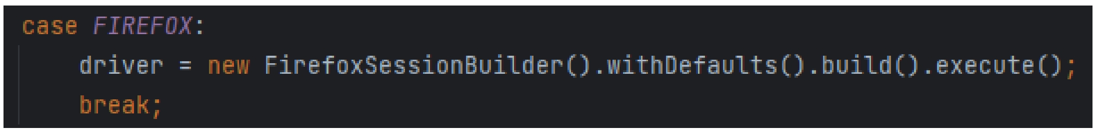


```java
public ChromeSessionBuilder headless() {
    this.chromeOptions.addArguments("--headless");
    return this;
}

public ChromeSessionBuilder incognito() {
    this.chromeOptions.addArguments("--incognito");
    return this;
}
```

Figure 23 - Builder optional methods

Another benefit to this approach is that it becomes fairly simple to adapt any existing project to use the Selenium Grid instead of a local web driver. Since a local web driver can be instantiated in one line of code, I aimed to create a RemoteWebDriver in no more than that. The builder pattern gave me that possibility, since every option is now a method that can be chained until a build point, after which the driver is executed in the grid. This together with the Maven artifact (package) created a very elegant solution that is both non-intrusive and easy to implement.



```
case FIREFOX:
    driver = new FirefoxSessionBuilder().withDefaults().build().execute();
    break;
```

Figure 24 - Session builder in Greenvalley code

7.9 Dashboarding tools research

7.9.1 Dashboard tool requirements

First of all, I needed to figure out what good requirements are for a dashboarding tool for vulnerabilities in an enterprise environment. To gather this knowledge, I held a presentation for the Dutch security department from Pieter Meulenhoff to explain the goal of the project and what it would accomplish. This meeting also served as an intake for me to gather knowledge on important requirements for such a dashboarding tool. This gave me the following key features and information:

- **Centralized Reporting**: The tool should be able to collate data from various sources and present it in a centralized, easily accessible dashboard. This should include data from automated scans, manual testing, and other security assessment activities.
- **Real-Time Updates**: In the context of security, time is of the essence. The tool should provide real-time or near-real-time updates about identified vulnerabilities, their status, and any remediation activities.
- **Risk Prioritization**: Not all vulnerabilities carry the same level of risk. The tool should be able to prioritize identified vulnerabilities based on their potential impact and the likelihood of their exploitation. This helps teams focus their remediation efforts on the most critical issues first.
- **Risk Mitigation**: The ability to mark certain vulnerabilities as false positives or exclude them from upcoming scan rounds is essential. Even in automated reporting, the results still need some manual feedback. Being able to deduplicate similar findings and create exclusions, significantly lowers the time needed to audit scan results.
- **Integration Capabilities**: The tool should easily integrate with other systems used in the software development lifecycle, such as bug tracking systems, continuous integration/continuous deployment (CI/CD) platforms, and other security tools. This enhances workflow and ensures that the tool fits well within the existing infrastructure.
- **Customizability**: Every organization has different needs and processes. The tool should offer customization options that allow it to be tailored to the specific requirements of the organization.
- **User-Friendly Interface**: The tool should be easy to use, with a clear, intuitive interface. This reduces the learning curve and allows teams to start benefiting from the tool as quickly as possible.
- **Scalability**: As the organization grows, the tool should be able to scale and handle an increasing load without performance issues.

With these features in mind, I conducted a search and tested several applications to see if they met the requirements for this integration. After initial research and consulting Pieter, I came up with three contenders for the job: Faraday, Cervantes and Defect Dojo.

7.9.2 Application Comparison

Faraday is an Integrated Penetration-Test Environment (IPE) that is designed to enable collaborative work among pentesters from the discovery phase to the delivery of the final report. It offers features like multi-user support, real-time collaboration, a centralized knowledge database, and more. Faraday also has a feature that supports data import and export from and to various formats, which can be used to import and export data from other tools like Nessus, Burp Suite, and QualysGuard.

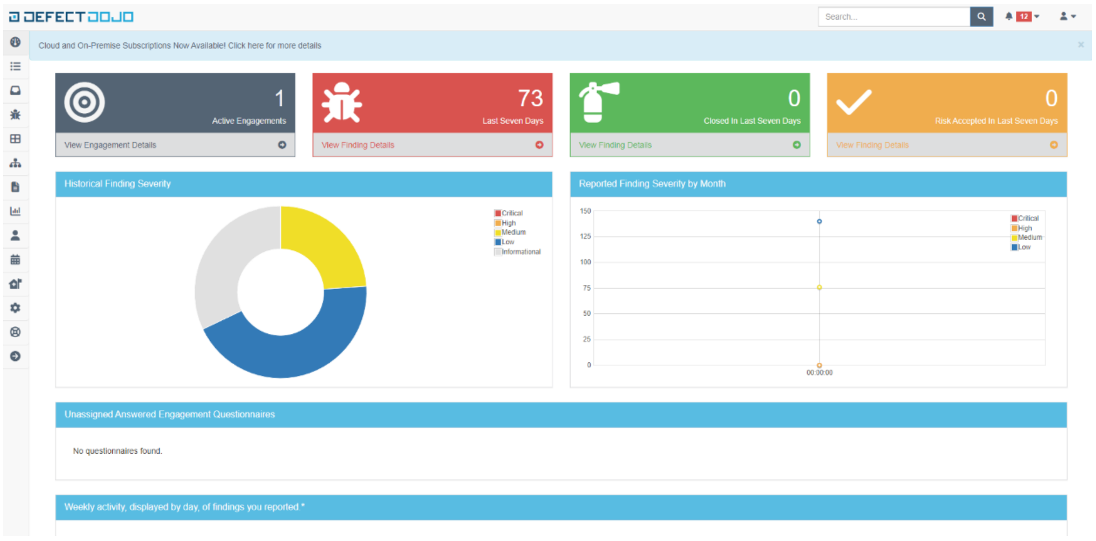
Cervantes, on the other hand, is a vulnerability management tool that enables teams to manage and keep track of vulnerabilities in a centralized manner. It allows teams to import scan data from various scanning tools and manage them in one place. It also has an integration feature that enables you to integrate with DefectDojo with just one click.

DefectDojo is an open-source application vulnerability management tool. It has several core data classes including Product Type, Product, Engagement, Test, Finding, and Endpoint. Each class has a different function in the vulnerability management process. For example, a 'Product' is any project, program, or product currently being tested, while a 'Finding' represents a discovered flaw during testing.

DefectDojo also has a tagging system that facilitates the organization within each level of the data model, with tags being used for grouping objects, filtering, and tag inheritance. The tool also offers different ways to manage tags, such as creating or editing new objects, import and reimport, and bulk edit menu.

The tool also provides example workflows, like for a security engineer or a QE manager, which involve registering a new product, creating a new engagement, and adding tests and findings. Findings can be assigned a severity, and a report can be generated to send to the development team. Alerts are also given for bugs that persist longer than they are supposed to based on their severity.

7.9.3 Defect Dojo


Cloud and On-Premise Subscriptions Now Available! Click here for more details

No questionnaires found.

DEFECTDOJO
Search...
12
1
Active Engagements
View Engagement Details
73
Last Seven Days
View Finding Details
0
Closed In Last Seven Days
View Finding Details
0
Risk Accepted In Last Seven Days
View Finding Details
Historical Finding Severity
Critical
High
Medium
Low
Informational
Reported Finding Severity by Month
150
125
100
75
50
25
0
00:00:00
Critical
High
Medium
Low
Unassigned Answered Engagement Questionnaires
Weekly activity, displayed by day, of findings you reported *

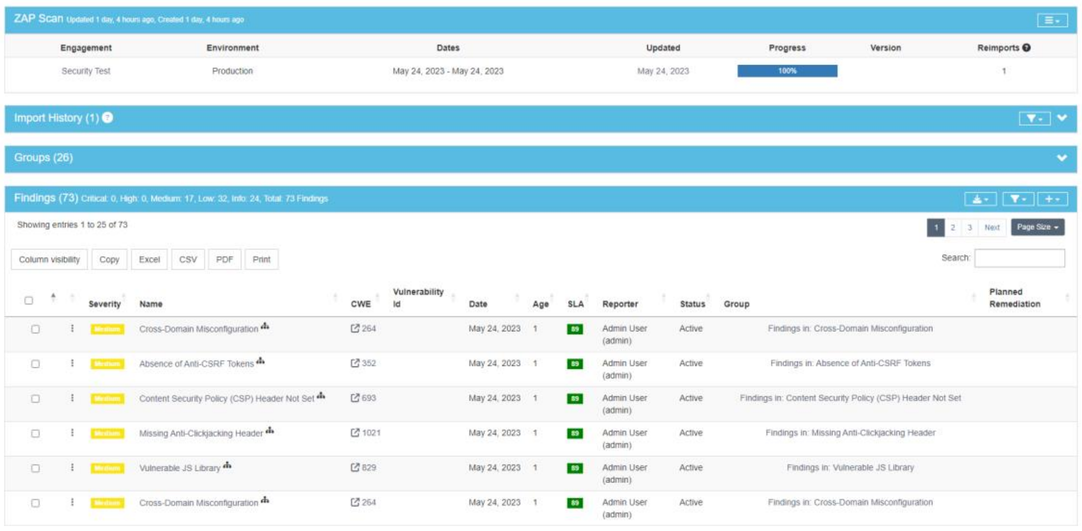
Figure 25 - Defect Dojo main view

After testing with several other applications, Defect Dojo was chosen because it delivered on the most important requirements and has an API that can be used to automate the import of zap scans.

DefectDojo has a powerful tagging system that helps in organizing and filtering objects within each level of the data model. Tags can be added and removed when creating or editing objects, during import or reimport of tests, or via the bulk edit menu for findings. The tagging system also allows for effective filtering of findings based on different criteria, and it supports tag inheritance, where tags applied to a given product are automatically applied to all objects under that product in the data model.

I started importing the Greenvalley test scans after their daily test run. This import was achieved through the API. Exporting the ZAP scan xml file had to be done manually to extract it from the Docker container. But once the scan was copied to the server, it could be imported in Defect Dojo through the API.

Each scan could be assigned to a customer and project, allowing the scans to be grouped together. This grouping then facilitated the mitigation of found vulnerabilities and also ensured deduplication of similar findings.


ZAP Scan Updated 1 day, 4 hours ago, Created 1 day, 4 hours ago

Engagement	Environment	Dates	Updated	Progress	Version	Reimports
Security Test	Production	May 24, 2023 - May 24, 2023	May 24, 2023	100%		1

Import History (1)

Groups (26)

Findings (73) Critical: 0, High: 0, Medium: 17, Low: 32, Info: 24, Total: 73 Findings

Showing entries 1 to 25 of 73

Severity	Name	CWE	Vulnerability Id	Date	Age	SLA	Reporter	Status	Group	Planned Remediation
Medium	Cross-Domain Misconfiguration	264		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Cross-Domain Misconfiguration	
Medium	Absence of Anti-CSRF Tokens	352		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Absence of Anti-CSRF Tokens	
Medium	Content Security Policy (CSP) Header Not Set	693		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Content Security Policy (CSP) Header Not Set	
Medium	Missing Anti-Clickjacking Header	1021		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Missing Anti-Clickjacking Header	
Medium	Vulnerable JS Library	829		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Vulnerable JS Library	
Medium	Cross-Domain Misconfiguration	264		May 24, 2023	1	89	Admin User (admin)	Active	Findings in: Cross-Domain Misconfiguration	

Figure 26 - Defect Dojo vulnerability overview

7.10 CI/CD pipeline integration

7.10.1 Docker restart policy

Since all containers need to be up and running all the time, all Docker containers need to have a restart policy set. The restart policy in a Docker Compose file dictates what should happen if a service's container crashes. There are several options, so I need to research these options first in the official documentation.

- "no": This is the default policy. It means that if a container crashes, Docker will not attempt to restart it.
- "always": If you set the restart policy to always, Docker will always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted.
- "on-failure": Docker will only restart the container if the container exits with a non-zero exit status (i.e., there was an error). You can optionally specify a maximum number of times Docker should try to restart the container.
- "unless-stopped": Docker will always restart the container unless it is manually stopped.

Since I want control over the container in case I need it to shut down, the ‘unless-stopped’ policy is the one most suitable for the job. So this policy will be added to all the Docker compose files to keep everything up and running unless specifically requested to exit.

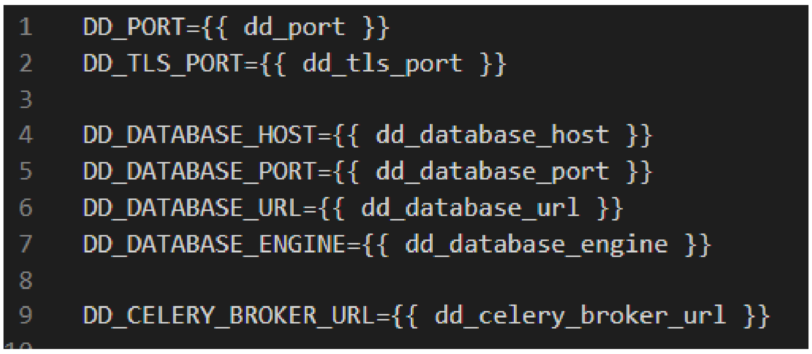
7.10.2 Ansible Deployment

For deployment automation on the server, Ansible is a very good solution as it provisions and deploys all the software on the server using a very simply principle: a playbook. In essence, this is a scenario that unfolds step by step until all the actions have been successfully completed.

To create this Ansible deployment a few key features are needed:

- vars.yml
- inventory.yml
- ansible.cfg

First off all, a vars.yml file needs to be created that holds all the sensitive information that will be passed on to the various components. The information in this file will be templated. In Ansible, templates are a way to manage entire configurations or configuration files. Ansible uses the Jinja2 templating engine to process templates. A template in Ansible is a file that defines an infrastructure configuration with variables that will be replaced by actual values when the template is used during a playbook execution.



```
DD_PORT={{ dd_port }}
DD_TLS_PORT={{ dd_tls_port }}

DD_DATABASE_HOST={{ dd_database_host }}
DD_DATABASE_PORT={{ dd_database_port }}
DD_DATABASE_URL={{ dd_database_url }}
DD_DATABASE_ENGINE={{ dd_database_engine }}

DD_CELERY_BROKER_URL={{ dd_celery_broker_url }}
```

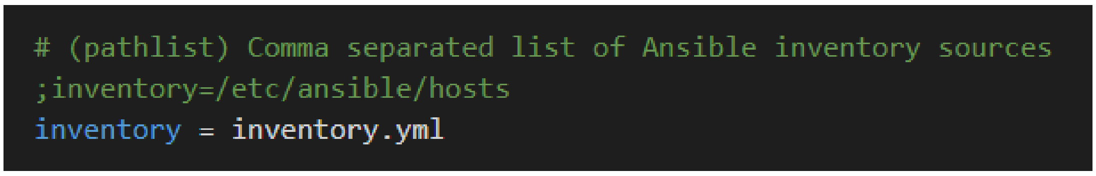
Figure 27 - Ansible vars.yml

The use of templates in Ansible allows for greater flexibility and reusability in your configurations. You can define a configuration once, as a template, and then reuse that template across multiple tasks or roles, with different variable values each time.

An inventory.yml file will also be necessary. In this file all the hosts on which the deployment has to be fulfilled, can be specified bv name. Users and IP addresses can be added here. It’s very important to initialize an ansible.cfg configuration file. This can be done with the following command:

`ansible-config init --disabled > ansible.cfg`

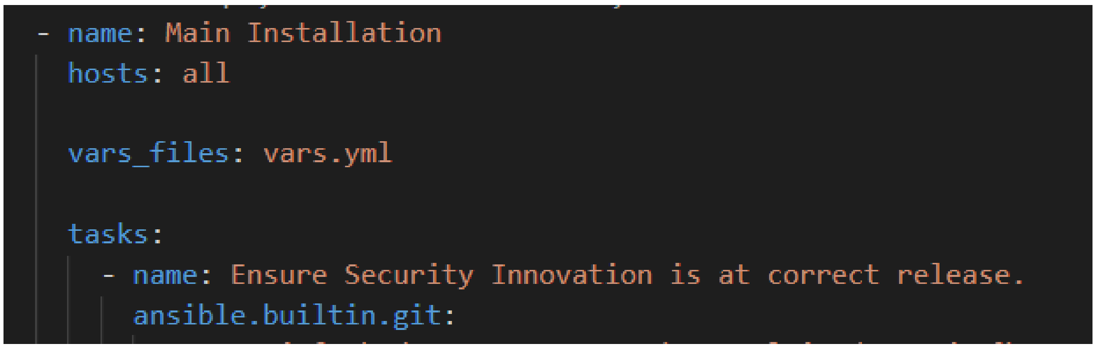
The inventory line must be commented out and the filename specified in order for deployment to succeed. Without this, the main.yml playbook cannot discover where to find its hosts.



```
# (pathlist) Comma separated list of Ansible inventory sources
;inventory=/etc/ansible/hosts
inventory = inventory.yml
```

Figure 28 - Ansible config file

Utilizing the vars.yml file which holds all the sensitive information, Ansible will run a playbook defined in the main.yml file. As the word suggests, this playbook will execute all the steps in sequence on all the predefined host machines.



```
- name: Main Installation
  hosts: all

  vars_files: vars.yml

  tasks:
    - name: Ensure Security Innovation is at correct release.
      ansible.builtin.git:
```

Figure 29 - Ansible Playbook

After setting all the hosts and the environment variables, the playbook will run a series of tasks in sequence. Figuring out this step by step approach is quite daunting at first, but becomes pretty clear after an initial setup. The scenario for this project consists of three steps:

1. Git clone the latest version of the repository in the specified working directory
2. Ensure that all the environment files for the different applications are copied to the right destination in the cloned repository. This is where the templates are used.
3. Start all the necessary containers and applications. Some containers require specific extra configuration or variables to be passed along.

7.10.3 Release pipeline on Azure Devops

The Ansible playbook allowed for easy deployment of all my containers and applications, but it still needed to be triggered manually. In order to have a full CI/CD compliant solution, I needed to create a pipeline that would trigger the Ansible playbook whenever changes were made to the project repository. This was achieved through the Azure Devops release pipeline.

A release pipeline in Azure DevOps is a way to automate the deployment of your application in different environments. It's part of Azure DevOps' built-in continuous delivery (CD) platform, and it's designed to work with the build pipelines that provide continuous integration (CI) capabilities.

A typical release pipeline has the following attributes:

- Artifacts: The release pipeline takes as input the artifacts produced by one or more build pipelines. These can be application binaries, scripts, configuration files, etc.
- Stages: The pipeline is divided into stages, each corresponding to an environment where the application needs to be deployed (for example, Dev, Test, Staging, and Production). Each stage can have its own set of tasks and configurations.
- Tasks: Within each stage, you define a series of tasks. These tasks are the actual steps required to deploy your application, such as copying files, running scripts, provisioning infrastructure, etc.
- Triggers: You can set up triggers to start the pipeline automatically when new artifacts are available (continuous deployment), or you can start it manually.

- Approvals: For critical stages (like deploying to Production), you can set up approvals. The pipeline will pause at these points and wait for a designated person or persons to manually approve before proceeding.

For my application, the most important configuration was situated in the Stages. A pipeline can be composed of multiple stages, and each stage is a logical boundary in the pipeline. A stage can represent a part of your CI/CD process. Commonly, stages are synonymous with the environment where you're deploying your application, such as development (Dev), testing (QA), staging (Staging), and production (Prod).

Each stage in a pipeline consists of one or more jobs. A job represents an execution boundary of a set of steps. All of the steps run together on the same agent. For example, you might have one job that builds your application, another that runs tests, and a third that deploys your application. You can run jobs sequentially or in parallel.

Within each job, there are tasks. Tasks are the smallest unit of work in Azure DevOps pipelines. Each task performs a specific action, such as invoking a script, running a command, or deploying to a target environment.

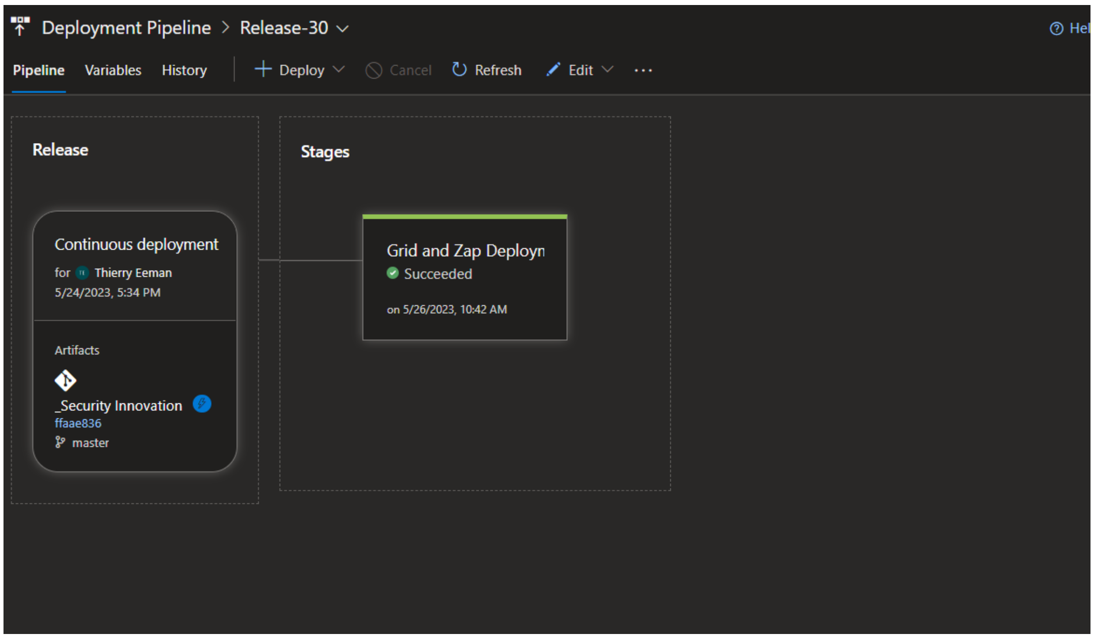
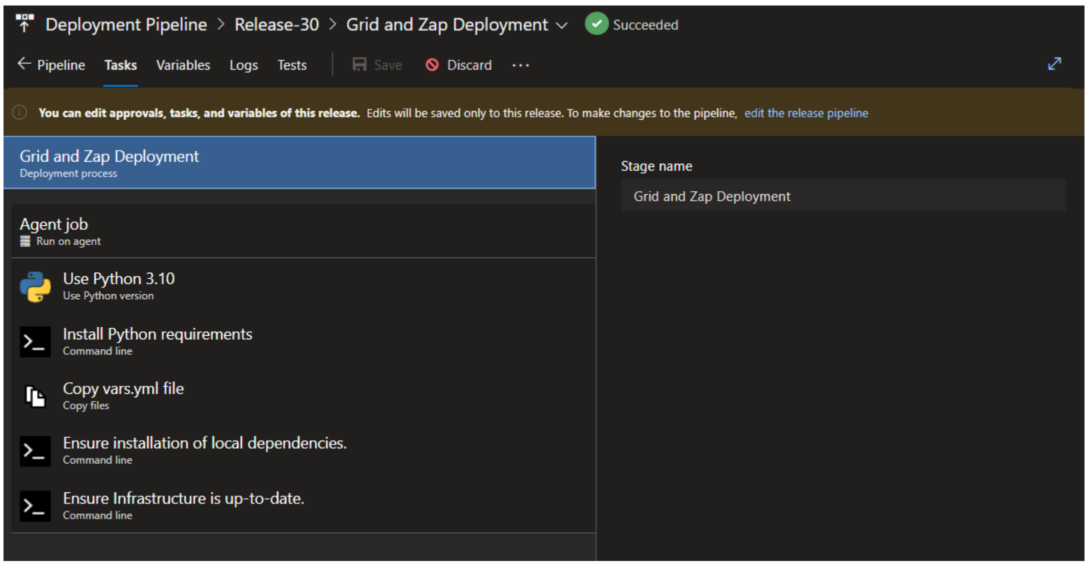


Figure 30 - Release setup and stages

I spent some time figuring out which jobs where necessary to complete the entire deployment.

I needed to:

- Install and use Python 3.10
- Install the requirements from the requirements.txt file
- copy the vars.yml file from the server to the repository
- run the playbooks (the local one and the remote one)



Deployment Pipeline > Release-30 > Grid and Zap Deployment ✓ Succeeded

← Pipeline | Tasks | Variables | Logs | Tests | Save | Discard | ···

You can edit approvals, tasks, and variables of this release. Edits will be saved only to this release. To make changes to the pipeline, edit the release pipeline

Grid and Zap Deployment
Deployment process

Agent job
Run on agent

Use Python 3.10
Use Python version

Install Python requirements
Command line

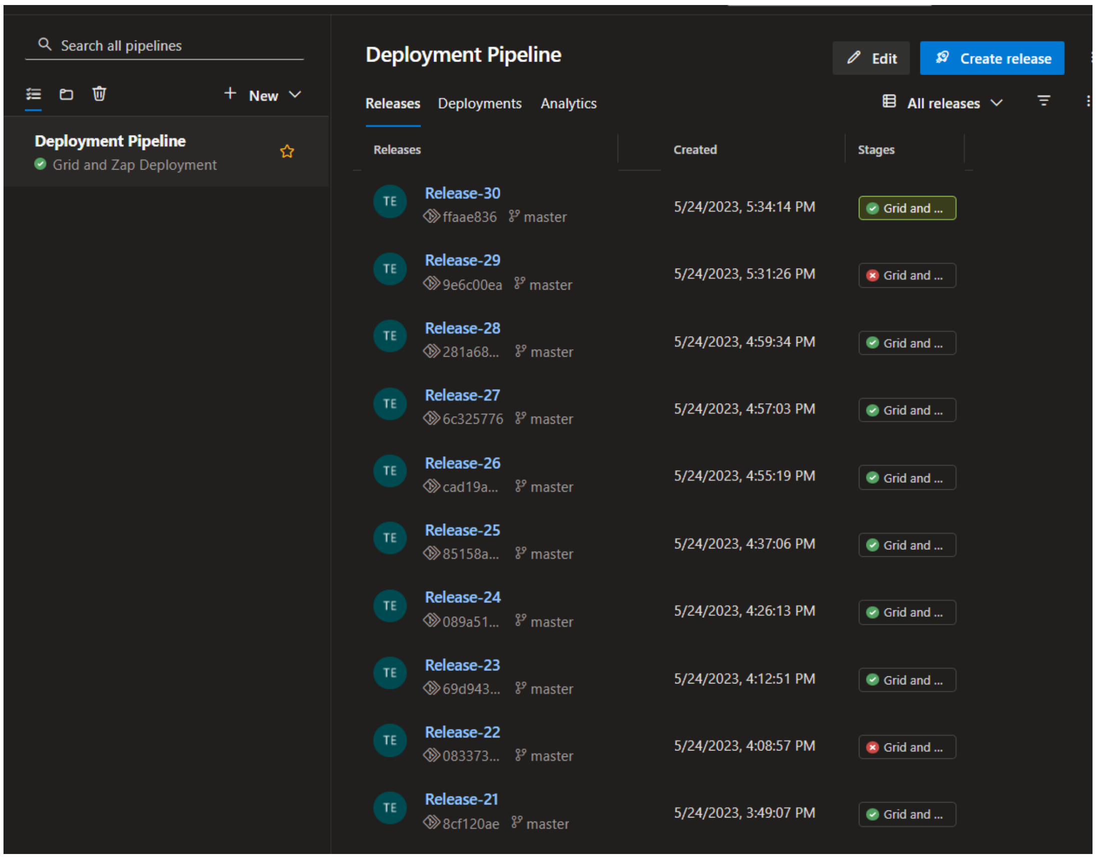
Copy vars.yml file
Copy files

Ensure installation of local dependencies.
Command line

Ensure Infrastructure is up-to-date.
Command line

Stage name
Grid and Zap Deployment

Figure 31 - Release jobs



Search all pipelines

New

Deployment Pipeline
✓ Grid and Zap Deployment

Deployment Pipeline

Edit | Create release

Releases | Deployments | Analytics | All releases

Releases	Created	Stages
TE Release-30 ffaae836 master	5/24/2023, 5:34:14 PM	✓ Grid and ...
TE Release-29 9e6c00ea master	5/24/2023, 5:31:26 PM	✗ Grid and ...
TE Release-28 281a68... master	5/24/2023, 4:59:34 PM	✓ Grid and ...
TE Release-27 6c325776 master	5/24/2023, 4:57:03 PM	✓ Grid and ...
TE Release-26 cad19a... master	5/24/2023, 4:55:19 PM	✓ Grid and ...
TE Release-25 85158a... master	5/24/2023, 4:37:06 PM	✓ Grid and ...
TE Release-24 089a51... master	5/24/2023, 4:26:13 PM	✓ Grid and ...
TE Release-23 69d943... master	5/24/2023, 4:12:51 PM	✓ Grid and ...
TE Release-22 083373... master	5/24/2023, 4:08:57 PM	✗ Grid and ...
TE Release-21 8cf120ae master	5/24/2023, 3:49:07 PM	✓ Grid and ...

Figure 32 - Successful releases upon repository update

7.11 Future project goals

7.11.1 Expandibility to other projects/languages

For the scope of this project, the whole solution with RemoteWebDrivers was based on a Maven Java project setup. Not all projects will have this fundament and in order to expand the testing of all projects in the company portfolio, additional languages and frameworks will have to be implemented. The overall architecture can remain, since the grid will accept any remote browser, regardless of the language it was written in.

7.11.2 Using Burp Suite instead of OWASP ZAP

The security department in the Netherlands uses Burp Suite instead of OWASP ZAP as a man-in-the-middle proxy. Burp Suite is not free, it has a licensed model instead. For this reason it was not implemented in this internship project. However, since it has many powerful features, it’s worth looking at for future expansions on this project.

Burp Suite is a comprehensive platform developed by PortSwigger for web application security testing. It consists of several tools that work together to assess the security of web applications. Burp Suite's key components include the Intercepting Proxy, Spider, Scanner, Intruder, and Repeater.

Advantages of Burp Suite:

- **User-Friendly Interface**: Burp Suite offers a well-designed and intuitive user interface, making it relatively easy to navigate and use.
- **Extensibility**: It provides extensive extensibility through its support for plugins and extensions, allowing users to customize and enhance its functionality.
- **Advanced Automation**: Burp Suite offers powerful automation capabilities, enabling users to automate repetitive tasks and streamline the testing process.
- **Active Scanner**: Burp Suite's active scanner performs automated vulnerability scanning and provides detailed results and potential security issues.
- **Mature and Widely Used**: Burp Suite has been around for a long time and has a large user base. It has a strong community support, regular updates, and is widely recognized as a reliable tool.

7.11.3 Report portal and Surefire fix for parallel testing

When running tests in parallel on the Selenium Grid, the logging on Reportportal becomes a nested cluster of information. The test results are no longer structured in a comprehensible manner, which renders the test run invalid and obsolete. Solving this issue depends on updates from the packages/dependencies in the main project.

7.11.4 CI/CD streamlining (automated importing in Defect Dojo)

For now all the CI/CD automation was performed with one client and one project for that client in mind. This allowed for very easy and basic import of the scan results, since they always pointed to the same direction. However, when running multiple test suits at once, these tests and especially their findings should be kept separate and directed to the right place in the application. This is to ensure that all data that passes through, is correctly tagged and reported.

7.11.5 Linking vulnerabilities to specific tests

For now, the remote web drivers do not have an id or something to recognize them with. This means that it's easy to retrieve any endpoints that failed, but it's hard or nearly impossible to see what event triggered the failure of the test. Going through all the documentation, discoveries were made. This lack of transparency can be countered by adding custom headers to the remote web drivers to identify them in the Selenium Grid.

7.11.6 DefectDojo on company level

DefectDojo has proven that is very elaborate and it seems to have all the necessary tools necessary to perform the task of providing an efficient and valuable dashboard solution for vulnerabilities. In order to fully reap the benefits of this tool, there will be need for a broader approach so that testing and reporting on vulnerabilities becomes embedded in the overall workflow of the company. This will require more testing of this application and working out a norm and standard of how it will be used within Resillion.

8 BIBLIOGRAPHY

Ansible playbooks — Ansible Documentation. (n.d.). Docs.ansible.com. Retrieved June 14, 2023, from https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#playbook-syntax

Builder. (2014). Refactoring.guru. https://refactoring.guru/design-patterns/builder

DefectDojo. (2023, June 14). GitHub. https://github.com/DefectDojo/django-DefectDojo

Docker Desktop overview. (2021, December 23). Docker Documentation. https://docs.docker.com/desktop/

Docker images for the Selenium Grid Server. (2023, June 14). GitHub. https://github.com/SeleniumHQ/docker-selenium

Finding web elements. (n.d.). Selenium. Retrieved June 14, 2023, from https://www.selenium.dev/documentation/webdriver/elements/finders/

Grid. (n.d.). Selenium. https://www.selenium.dev/documentation/grid/

OWASP ZAP – Getting Started. (n.d.). Www.zaproxy.org. https://www.zaproxy.org/getting-started/

OWASP ZAP – ZAP Docker Documentation. (n.d.). Www.zaproxy.org. Retrieved June 14, 2023, from https://www.zaproxy.org/docs/docker/

ramiMSFT. (2022, November 28). *Get started with Maven packages - Azure Artifacts*. Learn.microsoft.com. https://learn.microsoft.com/en-us/azure/devops/artifacts/get-started-maven?view=azure-devops

RemoteWebDriver. (n.d.). Www.javadoc.io. Retrieved June 14, 2023, from https://www.javadoc.io/static/org.seleniumhq.selenium/selenium-remote-driver/4.0.0-alpha-3/org/openqa/selenium/remote/RemoteWebDriver.html

SAST vs. DAST: difference and how to combine the two | Snyk. (n.d.). Snyk.io. https://snyk.io/learn/application-security/sast-vs-dast/

vijayma. (2023, January 11). *Use SSH key authentication - Azure Repos*. Learn.microsoft.com. https://learn.microsoft.com/en-us/azure/devops/repos/git/use-ssh-keys-to-authenticate?view=azure-devops