UNIVERSITY OF PIRAEUS

School of Information and Communication Technologies

Department of Informatics

Thesis

Thesis Title: Τίτλος Διατριβής:	**Integrating Security Scans into CI/CD Pipelines** Ενσωμάτωση ελέγχων ασφαλείας σε συστήματα συνεχής ενσωμάτωσης και εκτέλεσης με μέσα πληροφορίας
Student's name-surname:	**George Trifyllis**
Father's name:	**Paraskevas**
Student's ID No:	**Π14185**
Supervisor:	**Constantinos Patsakis, Associate Professor**

July 2024/ Ιούλιος 2024

Copyright ©

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τους δικούς μου ανθρώπους που ήταν δίπλα μου και με βοήθησαν να ξεπεράσω τα προσωπικά μου προβλήματα για να βρίσκομαι εδώ που είμαι σήμερα, χωρίς αυτούς η έκβαση θα ήταν πολύ διαφορετική.

Θα ήθελα να εκφράσω τις ευχαριστίες μου προς το προσωπικό του τμήματος της Πληροφορικής του Πανεπιστημίου Πειραιά που με οδήγησε στο μονοπάτι που ακολουθώ σήμερα και συνέβαλε σε μεγάλο βαθμό σε αυτό που είμαι.

Επίσης θα ήθελα να ευχαριστήσω τον κ. Πατσάκη που με εμπιστεύτηκε για την εκπόνηση αυτής της πτυχιακής και με ώθησε να εξερευνήσω το κομμάτι της Κυβερνοασφάλειας που όπως θα αναδείξω στην πορεία είναι το πιο κρίσιμο κομμάτι της σημερινής εποχής για τα τεχνολογικά επιτεύγματα.

Σας ευχαριστώ και πάλι για όλα , ελπίζω αυτή η πτυχιακή εργασία να φανεί χρήσιμη και να ενημερώσει όσους την διαβάσουν για τα σύγχρονα τεχνολογικά θέματα.

Περίληψη

Κατά τις τελευταίες δεκαετίες, η τεχνολογία έχει σημειώσει ραγδαία πρόοδο, κάνοντας την διαθέσιμη σε πολίτες σε μία κλίμακα που παλαιότερα ήταν αδιανόητη. Σήμερα σχεδόν κάθε ενήλικος διαθέτει τουλάχιστον μια έξυπνη συσκευή. Αυτές οι συσκευές φιλοξενούν εφαρμογές που περιέχουν ένα ευρύ φάσμα προσωπικών δεδομένων, από φωτογραφίες έως πληροφορίες για τραπεζικούς λογαριασμούς. Ισχυρά μέτρα ασφαλείας για αυτές τις εφαρμογές είναι απαραίτητα για την προστασία ευαίσθητων πληροφοριών από πιθανές απειλές. Οι εταιρείες στο παρελθόν έδειξαν δυσκολία, και ορισμένες το κάνουν ακόμα, με την ισορροπία ποιότητας και ταχύτητας κατά την προσπάθεια παράδοσης μια εφαρμογής.

Εδώ συμβάλει η Συνεχής Ενσωμάτωση και Συνεχής Παράδοση/Ανάπτυξη (CI/CD) που αποσκοπεί στην βελτιστοποίηση και την επιτάχυνση του κύκλου ζωής ανάπτυξης λογισμικού (SDLC), η δυνατότητα να δημιουργούνται αξιόπιστες, ασφαλείς και διατηρήσιμες εφαρμογές χωρίς να θυσιάζεται η ταχύτητα παράδοσης προς την αγορά. Ο σκοπός αυτής της πτυχιακής είναι να παρουσιάσει βήματα για ένα ολοκληρωμένο σύστημα μαζί με δύο CI/CD διεργασίες, μία που επικεντρώνεται στην Στατική Δοκιμή Ασφαλείας Εφαρμογών (SAST) και μια στην Δυναμική Δοκιμή Ασφαλείας Εφαρμογών (DAST). Μια μή ασφαλή εφαρμογή θα χρησιμοποιηθεί ώστε να παρουσιαστούν τα αποτελέσματα αυτών των δύο διεργασιών.

Η ασφάλεια πρέπει να είναι η πιο σημαντική πτυχή του κύκλου ζωής ανάπτυξης λογισμικού, επειδή μπορεί να καθορίσει τα διαθέσιμα εργαλεία στους προγραμματιστές και τη δομή της εφαρμογής. Γι' αυτό η ασφάλεια πρέπει να ενσωματωθεί στις διεργασίες που ακολουθεί το CI/CD.

Λέξεις Κλειδιά: Συνεχής Ενσωμάτωση και Συνεχής Παράδοση/Ανάπτυξη, Κύκλος ζωής ανάπτυξης λογισμικού, Στατική Δοκιμή Ασφαλείας Εφαρμογών, Δυναμική Δοκιμή Ασφαλείας Εφαρμογών

Abstract

Over the last few decades, technology has advanced rapidly, becoming accessible to individuals on a scale previously unimaginable. Today almost every adult possesses at least one smart device. These devices host applications containing a wide range of personal data, from photos to e-banking information. Ensuring the security of these data involves protecting the medium through which they are transferred—namely, the applications themselves. Robust security measures for these applications are essential to safeguard sensitive information from potential threats. Companies have struggled in the past , and some still do, with the balance of quality and speed when trying to deliver.

That's where Continuous Integration and Continuous Delivery/Deployment (CI/CD) aims to streamline and accelerate the software development lifecycle (SDLC), the ability to build reliable, secure and maintainable applications without sacrificing the delivery speed to market. The goal of this thesis is to provide steps for a complete environment along with two CI/CD pipelines, one that emphasizes on Static Application Security Testing (SAST) and one on Dynamic Application Security Testing (DAST). An insecure application will be used in order to show results of those pipelines.

Security should be the most crucial aspect of the development cycle of an application because it can define the available tools to the developers and the structure of the application. That's why security must be integrated into the CI/CD pipelines.

Key Words: Continuous Integration and Continuous Delivery/Deployment , Software Development Lifecycle , Static Application Security Testing , Dynamic Application Security Testing

Table of contents

List of Figures

Introduction

In today's world software security has become crucial. As organizations adopt agile development methodologies, meaning that software has less time from development to production, the need to integrate security in the software development life cycle is paramount. Integrating security scans into Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines enables early detection of vulnerabilities. By following this shift left strategy for software security we are reducing the risk of costly security breaches.

This paper will guide you and will first introduce you to the theory of why we need security to be integrated into the CI/CD pipelines by showcasing theory and then real life examples. Then with comprehensive and concise steps we will build a sandbox CI/CD environment from scratch. After that pipelines will be added which run static application security testing and dynamic application security testing against a vulnerable application.

The goal is to make everyone aware that software security is a never ending battle and we have to automate this work in order to be done efficiently. Further exploration on this domain can happen by integrating more tools that may specialize in other fields of security.

1. Introduction : Overview of CI/CD principles

Before explaining what CI/CD is we should study the past and what made us reach this point. As with all actions humans make we are trying to find the most efficient way to reach the desired results. This has made the development lifecycle to change over the years making it as efficient as possible.

1.1 Waterfall model

Initially, when projects had a small and well-defined scope, the Waterfall model was the most common approach. The Waterfall model consists of five phases: Requirements, Design, Implementation, Verification, and Maintenance. Although not using the term Waterfall model this method is defined in Royce, W. W. (2021). Managing the development of large software systems (1970).

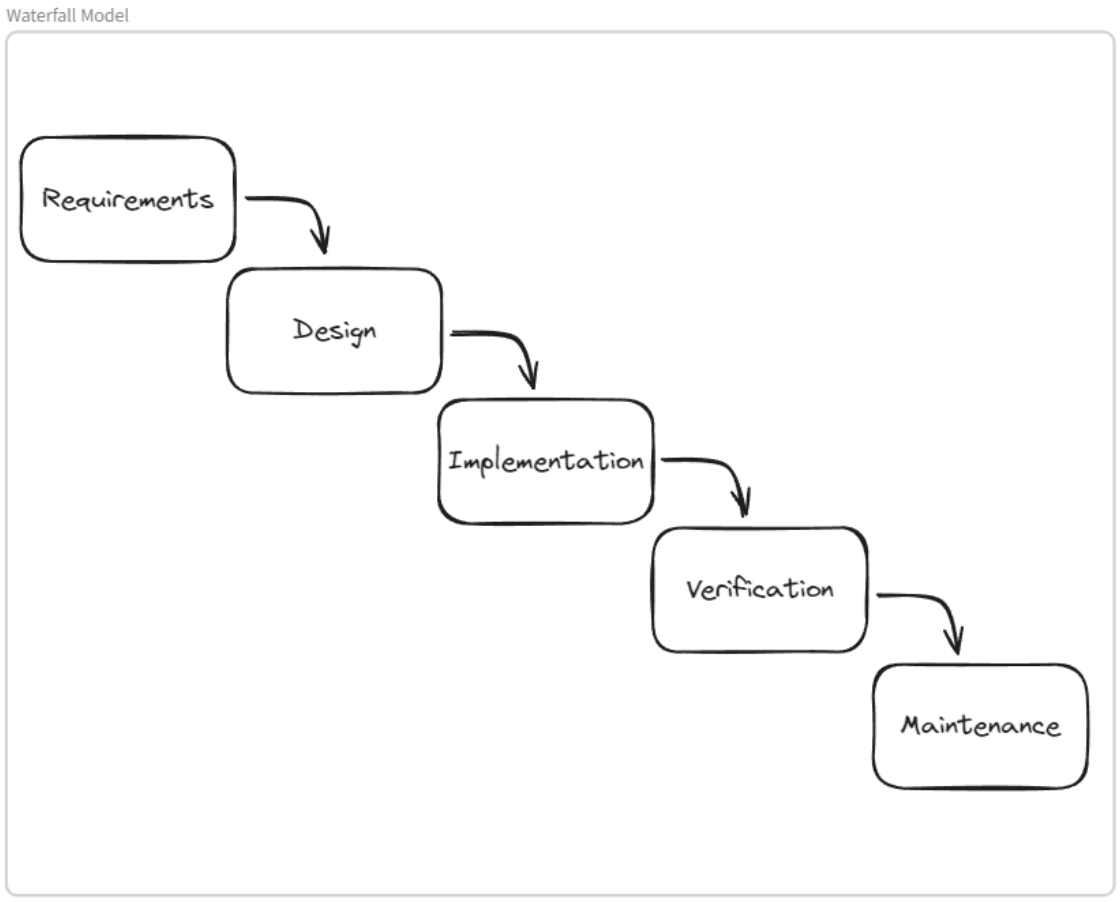


Image 1.1: Waterfall model

As you can tell this model makes the assumption that every time that we move on to the next phase the previous one is completed which even for small projects is a naive assumption to make and as Dr. Royce mentions in the previously mentioned article “the implementation described above is risky and invites failure”.

1.1.1 Security disadvantages for Waterfall model

This methodology has disadvantages regarding the security aspects of an application:

- Critical security issue: A critical security issue discovered late in the Implementation phase would require major changes and a potential violation of the initial contract.

- Minor security issue: A minor security issue arises during the Verification or the Maintenance phase then we have to return to the Implementation phase which sets us back in milestones and increases expenses every time we miss such a goal.
- Ongoing Vulnerabilities: With the downhill process that Waterfall follows it is difficult to address any new vulnerabilities discovered immediately which leads to exposing customers to a potential threat.

1.2 Agile Model

So another model must be followed if we want to be compliant with the security. That model should have the ability to have continuous development in mind , such as the Agile model.

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Manifesto for Agile Software Development

Agile methodologies are following an iterative development where a team handles in a specific timeframe, called sprint, a set of features for the application including design, implementation, testing and deployment

1.2.1 Security advantages for Agile model

This methodology has advantages regarding the security aspects of an application:

- Critical security issue: A critical security issue is discovered , this issue can be addressed in the following sprint with high priority or the feature introducing it can be pushed to the next sprint in order to not cause a delay to the rest of the deliverables.
- Minor security issue: A minor security issue arise during the sprint development team has the time to fix it during the sprint without any setbacks.
- Ongoing Vulnerabilities: The iterative nature of an Agile methodology allows the teams to respond to any new security issues that may arise and deliver fixes to the clients in a short time frame.

Agile promotes the iterative development and communication with the client so developers have to adapt to that style and that's why CI/CD has been created.

Continuous integration (CI) refers to the practice of automatically and frequently integrating code changes into a shared source code repository. Continuous delivery and/or deployment (CD) is a 2 part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.
Redhat : What is CI/CD?

If developers are the ones implementing and maintaining the application then who is responsible for creating and maintaining the CI/CD process ? The answer to that question is DevOps.

1.3 DevOps

DevOps combines development (Dev) and operations (Ops) to unite people, process, and technology in application planning, development, delivery, and operations. DevOps enables coordination and collaboration between formerly siloed roles like development, IT operations, quality engineering, and security.
Teams adopt DevOps culture, practices, and tools to increase confidence in the applications they build, respond better to customer needs, and achieve business goals faster. DevOps helps teams continually provide value to customers by producing better, more reliable products.
Microsoft : What is DevOps?

1.3.1 DevOps Goals

So DevOps are the bridge between the development team and the operations team with goals such as:

- Accelerate time to market
- Adapt to the market and competition
- Maintain system stability and reliability
- Improve the mean time to recovery

These goals can only be achieved with principles that focus on collaboration among all departments and normalization of pipeline procedures which is the CI/CD principle. Although we are mentioning theCI/CD principle we haven't mentioned security at all , are DevOps the one to integrate security? The answer is that under the aegis of DevOps there are two other teams that build security into all aspects of the process. DevSecOps and Rugged DevOps.

1.3.2 DevSecOps

DevSecOps, which stands for development, security, and operations, is a framework that integrates security into all phases of the software development lifecycle. Organizations adopt this approach to reduce the risk of releasing code with security vulnerabilities. Through collaboration, automation, and clear processes, teams share responsibility for security, rather than leaving it to the end when issues can be much more difficult and costly to address. DevSecOps is an enhancement to DevOps that builds security into all aspects of the process. The goal is to address security issues from the very start of the project. In this framework, not only does the entire team take responsibility for quality assurance and code integration but also security. In practice, this means teams discuss security implications during planning and begin testing for security issues in development environments, rather than waiting until the end. Another name for this approach is shift left security.
Microsoft: What is DevSecOps?

1.3.3 DevSecOps Goals

DevSecOps is an approach that integrates security practices within the DevOps process during the SDLC. So their goals consist of :

- Integrate security practices throughout the SDLC
- Automate the security process into the CI/CD to ensure consistent and reliable practices
- Share responsibility of security across all teams by providing tools that teams can use to maintain security and create secure and reliable software
- Protect customer data and privacy of the users

When these goals are achieved we have used security practices during the SDLC so we have shifted security left to identify vulnerabilities earlier so we can achieve an improved quality of deliverables while having cost efficiency and faster time-to-market.

So DevSecOps are the ones that integrate security practices into the DevOps process in various ways. What is the reason to even have a Rugged DevOps team? What are they offering ?

1.3.4 Rugged DevOps

I am rugged and, more importantly, my code is rugged.
I recognize that software has become a foundation of our modern world.
I recognize the awesome responsibility that comes with this foundational role.
I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.
I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security.
I recognize these things - and I choose to be rugged.
I am rugged because I refuse to be a source of vulnerability or weakness.
I am rugged because I assure my code will support its mission.
I am rugged because my code can face these challenges and persist in spite of them.
I am rugged, not because it is easy, but because it is necessary and I am up for the challenge.
The Rugged Manifesto

1.3.5 Rugged DevOps Goals

Rugged DevOps is a philosophy that extends the principles of DevOps to create software that is not only secure but also resilient and capable of withstanding attacks and adverse conditions. This can be achieved by setting goals:

- Building systems that can withstand attacks and failures as a reactive approach to security cannot scale and is doomed to fail.
- Share responsibility of security by performing activities like threat modeling, security architecture, secure coding training, and security testing so there is a lasting value across the project's team.
- Proactive measures to test and improve system's resilience in real world scenarios where users are not following "golden paths" and can be malicious.

All the previously mentioned models , goals and definitions of teams should be integrated into the CI/CD to ensure that security is a continuous, automated and integral part of the SDLC.. This integration is crucial because security is a fundamental value that everyone involved in a project should focus on, regardless of their position. By doing so, it leads to the creation of more secure, robust, and reliable software systems.

2. Security in software Development

As presented in the previous chapter 1. Overview of CI/CD principles security should be crucial during development and not left as the last part during the SDLC. This can lead to irreversible damage to a project by breaking the user's trust.Security in software development can also be referred as cybersecurity:

Cybersecurity refers to any technology, measure or practice for preventing cyberattacks or mitigating their impact.
Cybersecurity aims to protect individuals' and organizations' systems, applications, computing devices, sensitive data and financial assets against computer viruses, sophisticated and costly ransomware attacks, and more.
IBM : What is cybersecurity?

and attacks against that is referred as cyberattack:

A cyberattack is any intentional effort to steal, expose, alter, disable, or destroy data, applications, or other assets through unauthorized access to a network, computer system or digital device.
IBM : What is a cyberattack?

2.1 Parallels to physical security

Let's take examples from history and parallel them to the evolution of cybersecurity.In the beginning, humans sought shelter in caves and eventually built houses for protection from predators and the elements. This need for security evolved with the addition of physical measures like locks to deter home invasion. Similarly, the digital world necessitates cybersecurity measures like firewalls and strong passwords to prevent unauthorized access. However, these measures, like security cameras in a home, serve primarily as a deterrent and detection system. Just as a safe within a house offers an extra layer of security for valuables, data encryption provides an additional line of defense in cybersecurity. Even if a breach occurs, encrypted data remains unreadable without the decryption key, rendering it useless to attackers. As we can tell security and cybersecurity rely on layers of defenses and more parallels can be made like :

- Maintaining the house condition by checking the structure and repairing weaknesses that may arise for example a window that the lock doesn't work like patching vulnerabilities found in an application
- A plan after an attack is made , incident response ,in order to address and recover from an attack
- Even a guard dog being a form of multi-factor authentication

What everybody needs to understand , regardless of technical expertise , is that cybersecurity mirrors real world security so everybody has to be precautious about the actions that they take and that security relies on all parties. As said before security of any kind is conducted right when there is collective awareness and action. For example imagine a castle (a very well structured secure application) but allows the citizens to enter with an easy forged key (weak credentials). Intruders will either replicate the key by a brute force attack or steal the key and then roam freely inside the castle .

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. For the sake of efficiency, an attacker may use a dictionary attack (with or without mutations) or a traditional brute-force attack (with given classes of characters e.g.: alphanumeric, special, case (in)sensitive). Considering a given method, number of tries, efficiency of the system which conducts the attack, and estimated efficiency of the system which is attacked the attacker is able to calculate approximately how long it will take to submit all

chosen predetermined values.
OWASP : Brute Force Attack

2.2 Evolution of digital security in applications

As in the physical world digital security evolved with each new threat presented and in the last few decades with the technological boom the security has to keep up with all the new threats and vulnerabilities.

2.2.1 Pre-internet Era : Digital Security mirroring Physical Security

Before the era of downloading software from the internet companies were distributing software through physical mediums , floppy discs and CDs , to the customers. So the application needed to be functional with no major defects and that's why the Waterfall model was followed, there could be no incremental development as the costs to produce the physical updates and distribute them were really expensive. As there was no internet, digital security was as simple as physical security which means control the physical access to the machines , this is called securing the perimeter by creating a secure internal environment. Since all data , systems, even networks physically existed in one place companies had to restrict access to those. This was done by:

- Physical means :
  - Personnel having access control cards , limiting their access to specific rooms or buildings .
- Digital means :
  - Users having personal credentials to log into the computers
  - Access control Lists (ACL) user roles that limit access within topologies or files in the network .

This set the foundation of digital security but it is far from the current form.

2.2.2 Internet Era : Security Challenges and Agile Responses

When the internet became widely available, companies began publishing their applications to the internet thus creating the need to patch them over the internet as well. This influenced the development models to move towards agile models. However the internet introduced a new wave of security threats. Company networks were no longer isolated , even when not directly exposed to the internet , there was the need to connect geographically dispersed offices with secure connections . Along with the rise of data centers and cloud hosting security became complex. On top of these changes companies started to use Software as a service (SaaS) applications.

Software as a service (SaaS) is application software hosted on the cloud and used over an internet connection by way of a web browser, mobile app or thin client.
IBM : What is SaaS

Although SaaS applications offer the advantage that the service provider can use the CI/CD principles themselves and deliver robust and secure applications with regular updates that eliminate the need to be installed and managed by company personnel , they have to be integrated into the ecosystem of the company.

2.3 Cybersecurity strategies for complex infrastructures

2.3.1 Defense in Depth

Information security strategy integrating people, technology, and operations capabilities to establish variable barriers across multiple layers and missions of the organization.
The National Institute of Standards and Technology (USA) : defense-in-depth

Defense in depth strategy focuses on layering defenses instead of having a general secure perimeter , each layer should be secured independently from the previous ones. Furthermore there should be a Governance team , to ensure that the company is following compliance requirements , working along the Operations team in order to help the Development team to create robust applications.

2.3.2 Zero Trust

Zero trust is a framework that assumes a complex network's security is always at risk to external and internal threats. It helps organizations strategize a thorough approach to counter those threats.
IBM : What is zero trust?

Zero Trust strategy revolves around one rule "Never trust, always verify". This rule assures that all connections are authenticated even if an authentication took place in a previous layer a new independent authentication should take place for each layer.

Both of those Cybersecurity strategies ensure that we follow practices which enable authorized and secure practices, by stopping any kind of activity without permission and even when one system is compromised then linked systems will not be,

3. Importance of security in software development

Previous chapter 2. Security in software development presents strategies that companies should follow in order to have a secure ecosystem. But what happens if there is poor implementation of such strategies or even worse no implementation of them. We don't need to rely on hypothetical scenarios as everyday there are successful cyberattacks one example is the Equifax Data Breach.

3.1 Real World Example: Equifax Data Breach (2017)

One of the largest data breaches in history , if not the largest one , is that of a major credit reporting agency Equifax. Equifax exposed the personal information of at least 145.5 million people, based on data presented by the USA Government which was caused by an unpatched vulnerability in a web application framework Apache Struts. This led to social security numbers, birth dates, addresses, and more to be exposed .The cost to the company is estimated at over 4 billion dollars in total including settlements, fines, and remediation.

3.1.1 Report By the United States Government Accountability Office

The way that this data breach was done is reported in the GAO-18-559, DATA PROTECTION: Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach, following parts are from that document with highlighted parts to follow with the timeline of the actions.

Equifax has stated that, on ***March 10, 2017, unidentified individuals scanned the company’s systems*** *to determine if the systems were susceptible to a specific vulnerability that the United States Computer Emergency Readiness Team had* ***publicly identified just 2 days earlier****.* ***The vulnerability involved the Apache Struts Web Framework and would allow an attacker to execute commands on affected systems.***

According to Equifax officials, beginning on ***May 13, 2017****, in a separate incident following the initial unauthorized access, attackers gained access to the online dispute portal and used a number of techniques to disguise their activity. For example, the attackers leveraged existing encrypted communication channels connected to the online dispute portal to send queries and commands to other systems and to retrieve the PII residing on the systems.* ***The use of encryption allowed the attackers to blend in their malicious actions with regular activity on the Equifax network and, thus, secretly maintain a presence on that network as they launched further attacks without being detected by Equifax’s scanning software****. Equifax officials added that, after gaining the ability to issue system-level commands on the online dispute portal that was originally compromised, the attackers issued queries to other databases to search for sensitive data.* ***This search led to a data repository containing PII, as well as unencrypted usernames and passwords*** *that could provide the attackers access to several other Equifax databases. According to Equifax’s interim Chief Security Officer,* ***the attackers were able to leverage these credentials to expand their access beyond the 3 databases associated with the online dispute portal, to include an additional 48 unrelated databases.***

After successfully extracting PII from Equifax databases, ***the attackers removed the data in small increments****, using standard encrypted web protocols to disguise the exchanges as normal network traffic.* ***The attack lasted for about 76 days before it was discovered.***

3.1.2 The Apache Software Foundation responses

After the news that Equifax data breach was by the web application framework Apache Struts , the Apache Struts Project Management Committee (PMC) released two blog posts regarding this incident. Parts of the first blogpost Apache Struts Statement on Equifax Security Breach from PMC follow with highlighted crucial information and links to the Common Vulnerabilities and Exposures (CVE) mentioned for reader information.

*We are sorry to hear news that Equifax suffered from a security breach and information disclosure incident that was potentially carried out by exploiting a vulnerability in the Apache Struts Web Framework. At this point in time it is not clear which Struts vulnerability would have been utilized, if any. In an online article published on Quartz.com , t**he assumption was made that the breach could be related to CVE-2017-9805, which was publicly announced on 2017-09-04 along with new Struts Framework software releases to patch this and other vulnerabilities** . However, the **security breach** was already **detected in July** , which means that the attackers either used an earlier announced vulnerability on an unpatched Equifax server or exploited a vulnerability not known at this point in time –a so-called Zero-Day-Exploit. **If the breach was caused by exploiting CVE-2017-9805, it would have been a Zero-Day-Exploit by that time.***

1. *Understand which supporting frameworks and libraries are used in your software products and in which versions. **Keep track of security announcements affecting this products and versions.***
2. *Establish a process to **quickly roll out a security fix** release of your software product once supporting frameworks or libraries needs to be updated for security reasons. **Best is to think in terms of hours or a few days, not weeks or months. Most breaches we become aware of are caused by failure to update software components that are known to be vulnerable for months or even years.***
3. *Any complex software contains flaws. **Don't build your security policy on the assumption that supporting software products are flawless**, especially in terms of security vulnerabilities.*
4. ***Establish security layers.** It is good software engineering practice to have individually secured layers behind a public-facing presentation layer such as the Apache Struts framework. A breach into the presentation layer should never empower access to significant or even all back-end information resources.*
5. ***Establish monitoring for unusual access patterns to your public Web resources**. Nowadays there are a lot of open source and commercial products available to detect such patterns and give alerts. We recommend such monitoring as good operations practice for business critical Web-based services.*

Once followed, these recommendations help to prevent breaches such as unfortunately experienced by Equifax.

After 8 days that more info came into the light there was another blogpost MEDIA ALERT: The Apache Software Foundation Confirms Equifax Data Breach Due to Failure to Install Patches Provided for Apache® Struts™ Exploit which stated :

*What: **On 7 September 2017, credit reporting agency Equifax announced a data breach affecting 143 million consumers**. https://investor.equifax.com/news-and-events/news/2017/09-07-2017-213000628*
*Following this announcement, **additional claims stated that the breach was caused by** CVE-2017-9805**, an exploit in Apache Struts that was disclosed on 4 September 2017**. https://qz.com/1073221/the-hackers-who-broke-into-equifax-exploited-a-nine-year-old-security-flaw/*
*On **9 September 2017**, the Apache Struts PMC issued a statement on the Equifax data breach that included details on its response process to reported vulnerabilities and also **provided recommended security guidelines**. https://s.apache.org/8thB*
*On **13 September 2017**, Equifax issued a statement confirming that **"The vulnerability was Apache Struts CVE-2017-5638"**. https://www.equifaxsecurity2017.com/*

This vulnerability ***was patched on 7 March 2017, the same day it was announced****.*
https://cwiki.apache.org/confluence/display/WW/S2-045
In conclusion, the Equifax data compromise was due to their failure to install the security updates provided in a timely manner.

3.2 Lessons learned from a real world example

The lessons learned from these real world examples regarding infrastructure security is everything described in the previous sections.

- Equifax made the most critical mistake not patching a basic vulnerability that was discovered , patched and released to the public in a timely manner. This may be due to poorly implemented security practices or outright negligence of management.
- Silos are defensible. Once the attackers were inside the perimeter, they were able to move freely as they were not restricted to a single machine resulting in having access to more than the initially breached ones. This means that Zero Trust was not implemented in the system.

By following strictly the previous strategies and the strategies mentioned by PMC teams can reduce the risk of such cyberattacks and as learned by the Rugged DevOps.

"I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security"

4. CI/CD Ecosystem

Based on all the previous mentioned methodologies and strategies a modern ecosystem will be introduced to house our CI/CD pipelines. Of course, depending on when you are reading this, do research in order to ensure that the services used are still secure and please follow the instructions mentioned and scale based on your needs. I want to emphasize the criticality of doing research to ensure that everything is up to date as since when I started doing investigation and implementation there have been major updates to services used and even a critical patch.

The service Jenkins used in the ecosystem announced that a critical vulnerability along with 6 high vulnerabilities were found listed in the Jenkins Security Advisory 2024-01-24 and prompted in the application all the users to proceed with an update, so we can see that the advice from the PMC team is a basic rule that always must be followed with no exception.

“Understand which supporting frameworks and libraries are used in your software products and in which versions. Keep track of security announcements affecting these products and versions.”

4.1 Host of the Ecosystem

First approach was to use a Mini PC with Proxmox.

4.1.1 What is Proxmox?

Proxmox Virtual Environment (Proxmox VE) is an open-source platform designed for virtualization that combines two virtualization technologies: KVM (Kernel-based Virtual Machine) for virtual machines and LXC (Linux Containers) for lightweight container-based virtualization. Using open-source software guarantees full access to all functionality, as well as a high level of reliability and security. Features - Proxmox Virtual Environment

4.1.2 Why use Proxmox as a base host

- Security : Since Proxmox VE is an open-source virtualization platform, which means that its source code is open and can be audited by the community. The open nature of the software allows for transparency, and security-conscious users can review the code to identify and address potential security issues.
- Container and Virtual Machine Isolation : Both KVM and LXC are supported by Proxmox VE. Virtual machines provide full isolation, emulating a complete hardware environment, while containers offer lightweight and efficient virtualization at the operating system level.
- This thesis scope is to not only serve as a guide but also hold practical utility. For real-world applications, ensuring high availability is crucial. Proxmox VE facilitates this by allowing the creation of clusters and incorporating features like live migration of virtual machines and high availability. The implementation of clustering not only enhances security by introducing redundancy but also ensures the sustained operation of your virtualized environment, even in the event of a node failure.

4.1.3 Reasons why Proxmox was not used in the end

The only reason that I decided against using Proxmox besides all the benefits that it provides was that the Mini PC simply wasn't powerful enough to handle the growing number of services I wanted to host. Initially it ran everything smoothly and the services worked great. However as I added more services issues arose. After hours of investigation and trying to debug what went wrong a valuable lesson was learned , consider hardware capabilities when choosing a virtualization platform . Always have in mind scalability and even though Proxmox supports it I didn't as the Mini PC didn't have enough memory.

4.1.4 Alternative approach

Whole project was migrated to my personal computer but Proxmox requires to be installed as an Operating System (OS) and then on top virtualize everything else in a separate OS or have everything as a container. Then the decision was made to use Oracle VM VirtualBox.

4.1.5 Why VirtualBox was used as a final solution

Oracle VM VirtualBox is a free open source software that allows you to run virtual machines (VM) on your existing computer. TheseVMs act as simulated computer systems, allowing you to run different OS alongside your main one. Here's why VirtualBox stands out as a choice:

- Strong Backing and Community: VirtualBox is backed by the tech giant Oracle which ensures ongoing support. Additionally there is a large active user community that can provide troubleshooting assistance.
- System Compatibility : VirtualBox offers a wide range of supported guest OS which can be found under the Guest_OSes – Oracle VM VirtualBox wiki page.
- Sandbox creation: VirtualBox offers the option to create clonedVM which can be used as a sandbox for users to test and experiment without having to impact and in an extent compromise their personal systems.

4.2 Operating System

As with our application the used OS should be robust and supported by the developer. For me the answer was clear from the beginning with no second thoughts, Linux.

4.2.1 Why use Linux?

Linux is free and open-source meaning it is available to the general public for use and review making it a secure approach as well. This results in many versions of Linux to be available or "distributions" as they are called which include the Linux kernel along with a set of libraries and often a package management system. Those distributions can be as light and minimal as the user needs it to be but there are distributions supported by the community with ongoing updates regarding stability and security.

4.2.2 Distribution of choice

My personal choice is Ubuntu 22.04.4 LTS which at the time of writing this thesis is a robust system that has existed for almost one year. There should be no fear that it has existed for one year because it is a LTS version meaning Long-Term Support. LTS versions receive five years of free security and maintenance updates which means that for the next 4 years the system will be secure.

4.3 Service hosting

After choosing the OS of our choice we have to decide how our services will be running. We can run them natively in Linux , make them run in different VMs or run them in containers.I shall list the pros and cons of running services in the OS, in a VM, or in a Container in order to make a choice of course here you can adjust based on your needs.

4.3.1 Running Services Directly in the OS

- Pros
  - Running services directly in the OS provides the best performance as there is no virtualization thus no overhead.

  - No need to allocate and manage resources , everything is available without the need of additional layers.
  - The best of all simplicity, no extra installations needed just run the services as per manual.
- Cons
  - Since everything is running directly in the OS then there can be conflicts between applications needing a finer tuning.
  - High security risk if one service is vulnerable everything is vulnerable.
  - Scaling and maintenance is way more difficult as the entire system needs to be replicated and all services go down if the system is in need of a repair.

4.3.2 Running Services in Virtual Machines

- Pros
  - Running services in VMs provides isolation which provides security and stability.
  - Since VMs can host their own OS each service can be run in what provides best performance and stability.
  - Resources can be allocated and managed per VM providing better control.
- Cons
  - As you can tell we have a higher overhead as we need to virtualize the hardware and the OS resulting in fewer total resources for our services.
  - Since VMs run on their own OS there is always a boot time which should be taken into consideration.

4.3.3 What is a container ?

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.
Docker : What is a container

4.3.4 Running Services in a Container

- Pros
  - Containers provide a form of isolation, they share the same host OS kernel but keep the services and dependencies separate.
  - Containers have minimal overhead and are packaged with just what is absolutely necessary to guarantee stability.
  - Since containers have specified dependencies which include them we can ensure that there is a consistency across systems.
- Cons
  - Containers must use the same OS kernel as the host , limiting the variety of environments.
  - Since containers share the same OS it can pose a security risk if not managed correctly.

4.3.5 Preferred way of service hosting

In my opinion having the services run directly on the OS but in separate machines is the way to go but considering the limitation of my available hardware containers is the best solution for this project as we want it to be used as a sandbox and not as a final implementation. So we are going to need something to manage all those containers and my choice for this is Docker.

4.3.6 Docker

Docker is an open-source platform that provides tools that work based on Docker Engine.

Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process dockerd | Docker Docs.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (CLI) client docker | Docker Docs.

The CLI uses Docker Engine API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.
Docker : Docker Engine overview

Docker provides various tools that can adjust to user needs :
- Docker CLI is a tool that uses the CLI that empowers you to interact with Docker containers and manage different aspects of the container ecosystem directly from the command line.
- Docker Desktop provides the same capabilities but with a user interface.
- Docker Hub is an application with user interface that is connected to a container registry hosted by Docker where developers can share their container images for public use.

For our purposes we will use the Docker CLI that comes along the Docker Engine in order to manage our containers since it includes everything and we have no need of a UI.

4.4 Version Control

One of our services and if not the most critical one that binds everything together is the service that will provide the service control.

4.4.1 What is Version Control ?

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.
Atlassian: What is version control?

4.4.2 Why use version control ?

As organizations accelerate delivery of their software solutions through DevOps, controlling and managing different versions of application artifacts — from code to configuration and from design to deployment — becomes increasingly difficult.

Version control software facilitates coordination, sharing, and collaboration across the entire software development team. It enables teams to work in distributed and asynchronous environments, manage changes and versions of code and artifacts, and resolve merge conflicts and related anomalies.
Gitlab : Why use version control

4.4.3 Version Control Systems

As you can tell a version control system (VCS) is the backbone of our operations. This is where everyone's work is stored no matter what your role is in the team. Also at this point I should note that if you follow this guide to implement it in a real world scenario don't overlook the security of the data and follow the most basic storage rule the "3-2-1 Rule". This rule states to have 3 copies of your data , use 2 distinct types of media for storage and keep at least 1 copy off-site. Although there are three established version control systems :

- Git
- Apache Subversion
- Mercurial

I selected Git, besides being free and open-source it provides tools that can help in various scenarios while having a very active community that can help out even in the most difficult or unexpected scenarios.

Since we've chosen our VCS, it is time to select a hosting platform to store our project's repository and integrate it into our workflows.

4.5 Hosting Platform

Having selected Git as our VCS we need to select a hosting platform to store our project's repository. Since our selection is the most popular one , there are a lot of options . With the vision of having a complete and isolated sandbox system considering our hardware limitations, a platform that allows containerized deployment is the best option.

4.5.1 GitLab

GitLab is the obvious choice for our project due to its comprehensive feature set and the availability of a free community edition GitLab CE in a container available from Docker Hub.

4.5.2 Why use GitLab ?

Besides having a free edition for us to host there are a lot of other reasons :

- Intuitive User Interface (UI) : GitLab has a web interface that caters to users with varying levels of technical expertise. Having a well structured graphical interface minimizes the reliance on command-line tools, making it accessible to those less familiar with Git commands.
- Security : This is a critical feature for us since GitLab provides granular access control mechanisms.
- Stability and Scalability : GitLab is used and backed by a plethora of companies GitLab Customers. While the free tier may not offer all the functionalities available in the paid versions, the inherent stability of the platform transcends editions.

Having listed the above we should take into consideration another point if this project transitions from a sandbox environment to a full-fledged application GitLab offers paid versions with a lot more features integrated. As highlighted in the previous chapter, early adoption of features can potentially reduce long-term costs.

4.6 Automation Server

As established in the "Overview of CI/CD principles" chapter having an automations server that acts as a central hub for automated tasks is critical. It allows you to define workflows, schedule them, and execute them automatically, reducing manual work and improving efficiency. By automating these processes, the server minimizes manual intervention, thereby enhancing efficiency and reducing the potential for human error. This concept aligns perfectly with the core principles of DevOps, which states automate every automatable aspect of the SDLC to bolster efficiency and accuracy. Although GitLab offers aspects of an automation server, decoupling the automated actions from it is the best option for me so we can always change our hosting platform if needed.

4.6.1 What is Jenkins ?

Jenkins is an open-source automation server which offers a free edition to use jenkins/jenkins - Docker Image available from Docker Hub. Jenkins offers all the previously mentioned features of an automation server and to be honest we are using a fraction of them, as there is a very active community backing it with a big variety of plugins to help.

4.6.2 Why use Jenkins ?

Besides offering an open-source full fledged free edition of an automated server there are more reasons to use Jenkins:

- Intuitive UI : Jenkins has a web interface that eliminates the need of a command line interface even for the most complicated activities.
- Large Community : A vast and active community supports Jenkins offering help at every point while providing a lot of resources and tutorials.
- Plugin Availability : Jenkins offers the use of plugins released by the community so the options are limitless in what operations we can automate with Jenkins.
- Security : Being open-sourced security is critical and Jenkins acknowledges that besides the community focus on security they take their own measures , more can be learned in Jenkins Security.
- Stability and Scalability : Backed by industry leaders like IBM and Amazon AWS. Jenkins offers proven stability and scalability. Also if the need arises Jenkins can easily distribute work across multiple machines making it scalable as long as resources are available.

4.7 Repository Manager

As we are focusing on the continuous integration and not on the continuous delivery/deployment in this thesis, the use of a repository manager is not strictly mandatory. However incorporating a repository manager offers some features that are beneficial if we expand the sandbox towards the delivery part.

4.7.1 What is a Repository Manager ?

A repository manager acts as a central point to where all the build images of your projects are hosted in order for users to be able to find them and download them. Also you can set it as a proxy for package manager in order to regulate the available packages by limiting access to unsecure packages to the users.

4.7.2 Sonatype Nexus Repository

Sonatype Nexus Repository offers a free, open-source version in a container found in Docker Hub which is ideal for our sandbox because as we can see from their documentation it is not advised for large teams. Besides being free and open-source this is my choice for a repository manager as it supports various package managers so no matter what we want to run in our sandbox with minimal configuration we can execute it.

5. Ecosystem Installation

5.1 Creating the VM

As described in “Host of Ecosystem” chapter the host of the ecosystem in my case will be a VM powered by Oracle VM VirtualBox. So let's create our VM that will host our services.

5.1.1 Download VirtualBox

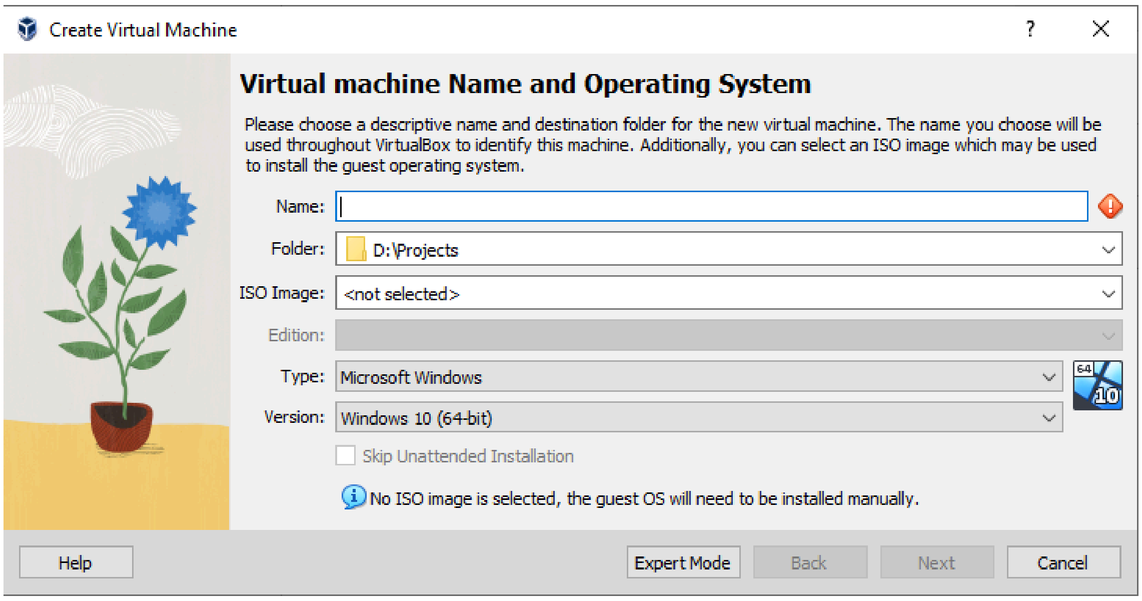
You will need to download a suitable version of VirtualBox for your system. All necessary information can be found under the official Download VirtualBox page. After downloading the executable proceed with a standard installation.

5.1.2 Download your selected version of OS

Before proceeding with the creation of the VM we will need to provide an International Organization for Standardization (ISO) image of the OS that we want to install. In my opinion Ubuntu 22.04.4 LTSis a solid choice for the reasons described under “Operating System”.

5.1.3 Configuring the VM

The first time that you open VirtualBox you will be greeted by a mostly empty window, select the option “New” to create a VM .



Virtual machine Name and Operating System

Please choose a descriptive name and destination folder for the new virtual machine. The name you choose will be used throughout VirtualBox to identify this machine. Additionally, you can select an ISO image which may be used to install the guest operating system.

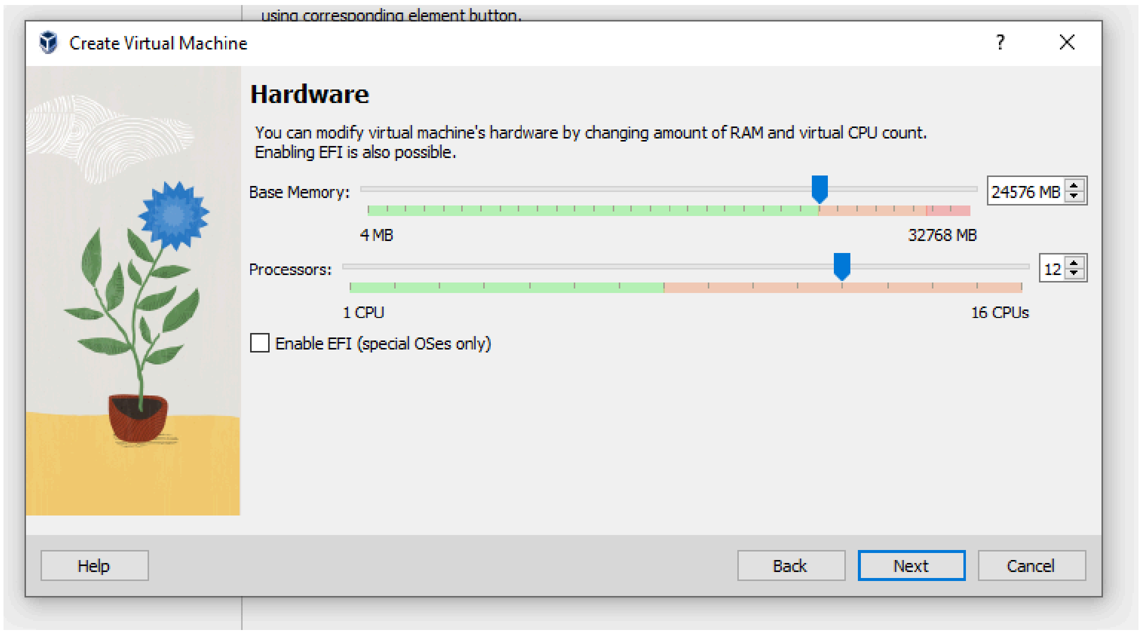
Name:
Folder: D:\Projects
ISO Image: <not selected>
Edition:
Type: Microsoft Windows
Version: Windows 10 (64-bit)
Skip Unattended Installation

No ISO image is selected, the guest OS will need to be installed manually.

Image 5.1: VirtualBox : Create Virtual Machine

You will need to provide a name, a folder to save all the necessary data for our VM and the ISO image that you downloaded before. It is highly recommended to select the checkbox Skip Unattended Installation .After that you will select the amount of RAM and virtual CPU count that will be allocated to the VM my choice was arbitrary and is probably more than what is needed but I wanted to make sure that it will not be limited by hardware :

- 24 GB of RAM
- 12 cores


using corresponding element button.

Create Virtual Machine

Hardware

You can modify virtual machine's hardware by changing amount of RAM and virtual CPU count. Enabling EFI is also possible.

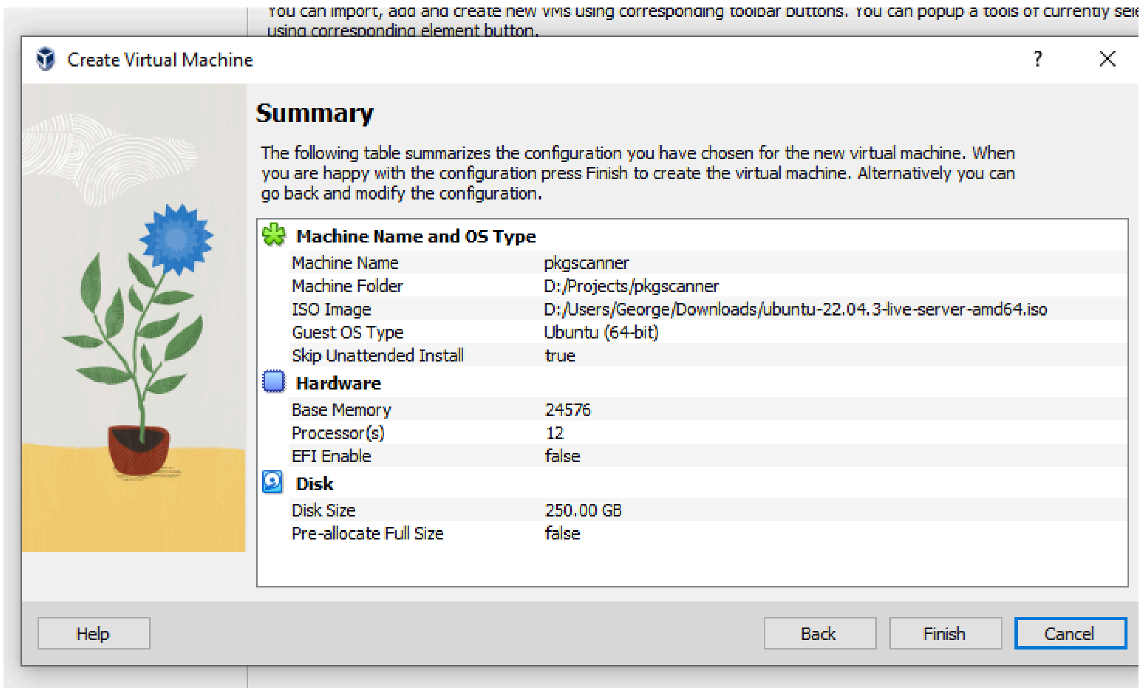
Base Memory: 24576 MB (4 MB – 32768 MB)

Processors: 12 (1 CPU – 16 CPUs)

☐ Enable EFI (special OSes only)

Image 5.2 : VirtualBox : Hardware

Then you will be prompted to select the Disk size allocated to the VM my choice was 250 GB of an HDD. If you followed along then a Summary looking something like the following will appear:


You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently sel
using corresponding element button.

Create Virtual Machine

Summary

The following table summarizes the configuration you have chosen for the new virtual machine. When you are happy with the configuration press Finish to create the virtual machine. Alternatively you can go back and modify the configuration.

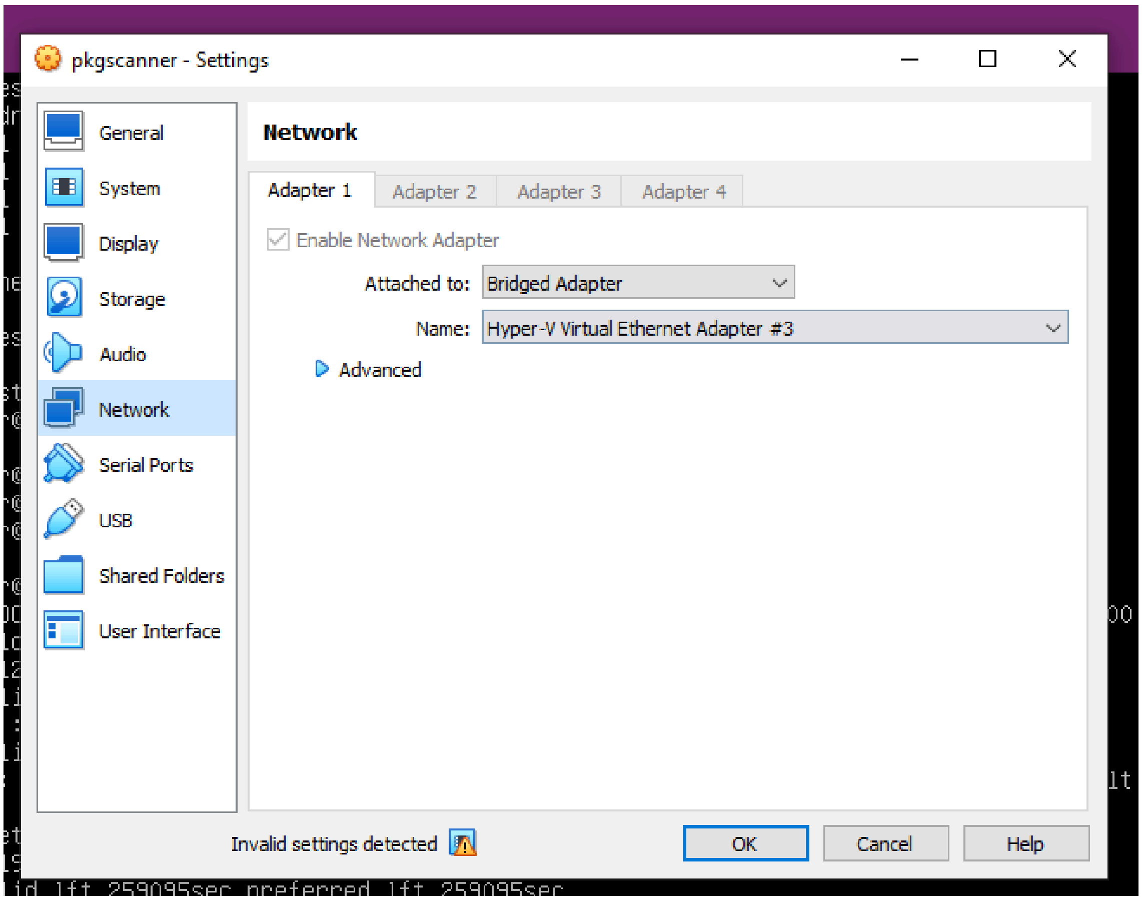
Machine Name and OS Type	
Machine Name	pkgscanner
Machine Folder	D:/Projects/pkgscanner
ISO Image	D:/Users/George/Downloads/ubuntu-22.04.3-live-server-amd64.iso
Guest OS Type	Ubuntu (64-bit)
Skip Unattended Install	true
Hardware	
Base Memory	24576
Processor(s)	12
EFI Enable	false
Disk	
Disk Size	250.00 GB
Pre-allocate Full Size	false

Image 5.3 : VirtualBox : Summary

After that you can select Finish and remember that these settings can change even after the creation of the VM in order to be adjusted to our needs. Now that we have our VM ready for the initial boot I highly recommend to go to the Settings of the VM and verify that under the Network tab :

- Checkbox Enable Network Adapter is selected
- Dropdown “Attached to” has the value “Bridged Adapter”. This setting makes the VM connect to the network as if it were a physical computer. Doing this may ease some network configurations but shares the network with your host computer making your

base machine vulnerable as well. So this setting is only recommended in the sandbox mode.



Network

Enable Network Adapter: ✓

Attached to: Bridged Adapter

Name: Hyper-V Virtual Ethernet Adapter #3

Image 5.4 : VirtualBox : VM Network Settings

5.1.4 Installing Ubuntu

Now you will need to boot the VM and proceed with the installation of Ubuntu. A standard installation is recommended with the "OpenSSH" option selected so we can access the VM from our host machine via network. Everything else is not needed at this point.

OpenSSH is the premier connectivity tool for remote login with the SSH protocol. It encrypts all traffic to eliminate eavesdropping, connection hijacking, and other attacks. In addition, OpenSSH provides a large suite of secure tunneling capabilities, several authentication methods, and sophisticated configuration options.
OpenSSH

For the sandbox mode dummy passwords and usernames were used so proceed with strong passwords.

5.1.4 Optional : Connect from your base host

There are a lot of Secure Shell (SSH) clients but I highly recommend downloading and installing Visual Studio Code as it can be used in later steps as well. Visual Studio Code is a free source code editor that supports plugins which will enable us in various ways. The IP to connect to can be found by running the following command.

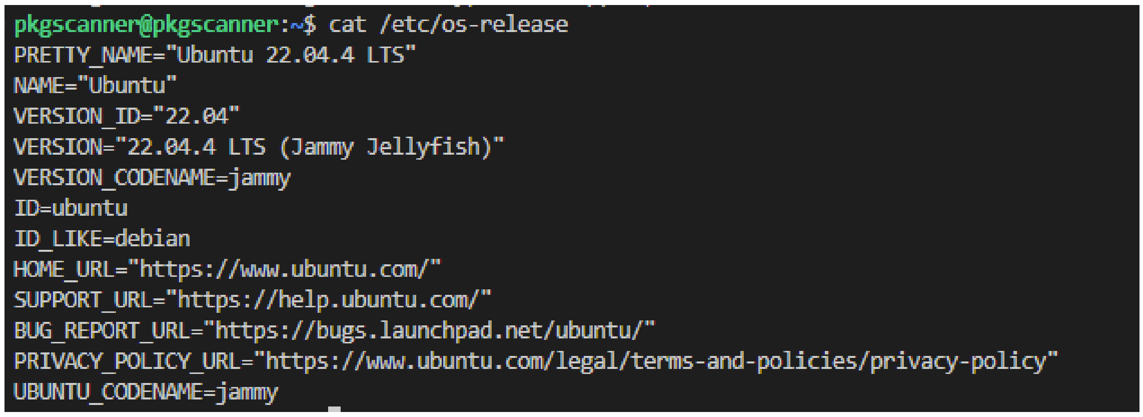
```
ip address
```

5.1.5 Verify Installation

The first operation once we log in for the first time is to verify the installation which can be done by running the following command .

```
cat /etc/os-release
```

In your command line something like the following image will appear to assure your installation.



```
pkgscanner@pkgscanner:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.4 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.4 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
```

Image 5.5 : Ubuntu cat /etc/os-release result

5.1.6 Updating and Upgrading after installation

Even though we have installed the latest LTS version we need to run a command to update and upgrade all the packages . The first command will scan for upgrades while the second one will ask for your permission to proceed with the download and installation. More information can be found in Ubuntu : Package management with APT .

```
sudo apt-get update
sudo apt-get upgrade
```

What you may have noticed is that you were asked for the password of your logged in user, this is due to the sudo part of the command which stands for superuser do. We can say that Linux in a way supports the Zero Trust strategy as even if you were logged in , once you asked to change system settings it was verified again. If you enter the correct password for a user with administrative privileges, sudo temporarily grants you elevated access to execute the specified command.

5.1.7 Setting to ensure stability for services

Our services of course use memory and need to be able to have enough of it. Even though we have provided the VM with the memory we need to also increase the limit of the mappings a process can have. This can be done by altering the `sysctl` setting `vm.max_map_count`. The recommended value is `262144` and in order to make a permanent change , you'll need to modify the system configuration file where these settings are defined. This involves editing the `/etc/sysctl.conf` file or adding a new file in the `/etc/sysctl.d/` directory. By running the following command you will open the file `/etc/sysctl.conf` in the nano command line text editor and since this alters system settings sudo should be used as well.

```
sudo nano /etc/sysctl.conf
```

Add the following line at the end of the file and save the changes made.

```
vm.max_map_count=262144
```

Then run the command to apply the changes from the configuration file.

```
sudo sysctl -p
```

After these steps we have set the VM and the OS and are ready to install Docker.

5.2 Docker Installation

As described in “Service hosting” we need to install the Docker Engine in our system in order to host our containers. We can follow the official, a non GUI installation found in Install Docker Engine on Ubuntu . Nonetheless I will offer an explanation of every step.

5.2.1 Prerequisites

Docker as all the applications have an OS requirement please review that you are compatible as the OS is left on to the final user for selection. This installation guide will be for Ubuntu 22.04.4 LTS. Also if you are not in a fresh system verify that there are no conflicting packages installed more info can be found in the Uninstall old versions of the official documentation.

5.2.2 Installation

First of all we need to add Docker's official GPG key (please review on the official documentation site that the following is the latest configuration) . GPG key stands for GNU Privacy Guard key which follows the standards of private and public key.

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

The commands in order do:

- As a superuser (`sudo`) update the list of available software packages and their versions.
- As a superuser (`sudo`) create a directory with specific permissions (`-d`).
  - `/etc/apt/keyrings` file location.
  - `-m 0755` permissions for the newly created directory.
    - 0 means the lack of special permission bits.
    - 7 as the next digit means grant all permissions (read, write, execute) for the current owner.
    - 5 in the third digit means grant read and execute permissions to the group that owns the directory.

        - 5 in the last digit means read and execute permissions to the users that do not belong in the previous categories.
- As a superuser (`sudo`) download form the internet (`curl`).
    - `-fsSL` silence the curl output even at fails and follow redirects on the provided link
    - `https://download.docker.com/linux/ubuntu/gpg` This is the URL from where the GPG key file will be downloaded.
    - `-o /etc/apt/keyrings/docker.asc` outputs the downloaded file to the destination.
- As a superuser (`sudo`) change the permissions (`chmod`) on the downloaded file to
    - `a` all users
    - `+` add permission
    - `r` read permission
- Print into the terminal the result of (`echo`) the provided
    - From the Debian package repository `deb`
    - Get system architecture `arch=$(dpkg --print-architecture)`
    - Then use the GPG key that we downloaded before `signed-by=/etc/apt/keyrings/docker.asc`
    - From the URL `https://download.docker.com/linux/ubuntu`
    - Get your running Ubuntu Version name `$(. /etc/os-release && echo "$VERSION_CODENAME")` the `stable` version
- `|` gets the data provided by the `echo`
    - `tee` reads it as a normal input and write it into the file `/etc/apt/sources.list.d/docker.list`
- Send any output to `> /dev/null`
- As a superuser (`sudo`) update the list of available software packages and their versions.
- As a superuser (`sudo`) install the following packages.

At this point Docker suggests to run the command which downloads a test image and runs it in a container.

```
sudo docker run hello-world
```

If everything is done correctly a confirmation message is printed and the container exits. This concludes the basic installation.

5.2.3 Services installation and configuration

As a measure of keeping everything organized I suggest creating the following directories to assign to our containers as means of having a permanent storage in our VM .

```
sudo mkdir -p /jenkins_home
sudo mkdir -p /nexus-data
sudo mkdir -p /sonarqube_data
sudo mkdir -p /sonarqube_extensions
sudo mkdir -p /sonarqube_logs
sudo mkdir -p /gitlab_data_config
sudo mkdir -p /gitlab_data_logs
sudo mkdir -p /gitlab_data_data
sudo mkdir -p /postgresql
sudo mkdir -p /postgresql_data
sudo mkdir -p /zap_data
sudo mkdir -p /shared
```

Docker offers the convenience of being able to create and do basic configurations for our services through one file named docker-compose . Although we could use a different docker compose for each service I believe that having a single configuration file is providing a lot more benefits as:

- Shareability : A single Docker Compose file makes sharing my application configuration easy while making sure that they can replicate my environment by running a single command.
- Readability and Maintainability : Having all service configurations in one location improves readability and maintenance as we have a clear picture of ports and volumes used.

First create the docker-compose file and open it with nano in order to edit it.

```
touch docker-compose.yaml
nano docker-compose.yaml
```

Check in Annex for "docker-compose.yaml" the configuration of the environment. I will explain quickly what it does but the explanation for each service will be included in the respective section. The provided docker compose file consists of three sections:

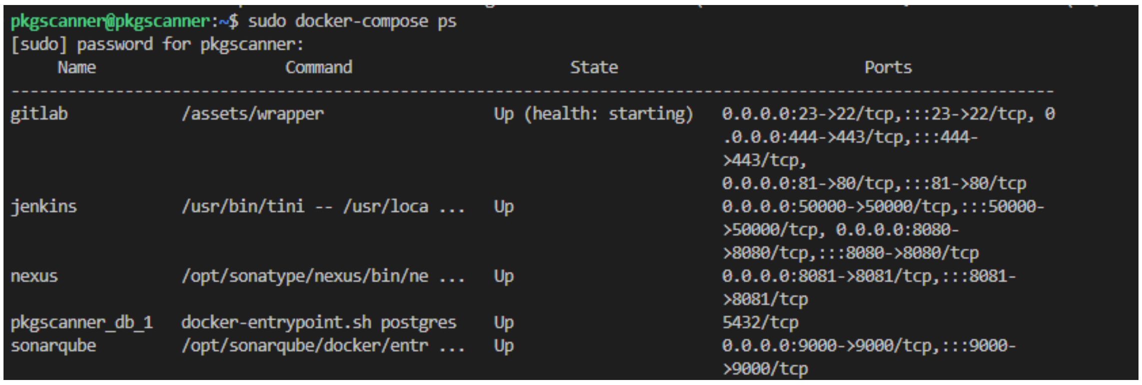
- version : Specifies the version of the compose file format we are using in order to ensure compatibility with the docker-compose-plugin version we are using.
- services : As the name implies this is where we define each service used and their configurations.
- volumes : Binding the directories created in a previous step with a name example given
  - \`postgresql_data\` is the name of the volume.
  - \`driver: local\` Local driver will be used to manage this volume.
  - \`driver_opts:\` Set driver options as:
    - \`type: none\` There is no need for special file system types. Although this is implied if not mentioned , it is nice to be able to see it.
    - \`o: bind\` Create a shortcut between the mentioned directory of the host machine and the directory of the container.
    - \`device: /postgresql_data\` Location of the host machine directory.

After saving the docker-compose file you need to run the following command in order for docker to start , download all the necessary images and boot the containers.

```
sudo docker-compose up
```

Logs will be printed from each service about the operations that they take. It is expected at the first boot to contain a lot of messages but pay attention because it will inform you about errors and general status. If everything was successful, running the following command in another command line will produce the following , showing that everything runs as expected.

```
sudo docker-compose ps
```



```
pkgscanner@pkgscanner:~$ sudo docker-compose ps
[sudo] password for pkgscanner:
    Name                  Command                      State                         Ports
-----------------------------------------------------------------------------------------------------------
gitlab          /assets/wrapper                  Up (health: starting)   0.0.0.0:23->22/tcp,:::23->22/tcp, 0
                                                                         .0.0.0:444->443/tcp,:::444-
                                                                         >443/tcp,
                                                                         0.0.0.0:81->80/tcp,:::81->80/tcp
jenkins         /usr/bin/tini -- /usr/loca ...   Up                      0.0.0.0:50000->50000/tcp,:::50000-
                                                                         >50000/tcp, 0.0.0.0:8080-
                                                                         >8080/tcp,:::8080->8080/tcp
nexus           /opt/sonatype/nexus/bin/ne ...   Up                      0.0.0.0:8081->8081/tcp,:::8081-
                                                                         >8081/tcp
pkgscanner_db_1   docker-entrypoint.sh postgres  Up                      5432/tcp
sonarqube       /opt/sonarqube/docker/entr ...   Up                      0.0.0.0:9000->9000/tcp,:::9000-
                                                                         >9000/tcp
```

Image 5.6 : Ubuntu : command sudo docker-compose ps

If we want to stop our running containers we should run \`docker-compose stop\` instead of \`docker compose down\` , this is because we want to gracefully stop all containers defined in our \`docker-compose.yaml\` which will save any configuration that we made in the container. If we want to start them again then the \`docker-compose up\` is used.

5.3 GitLab Installation

As described in “Version Control” and ”Hosting Platform” we are going to install GitLab so we can host and keep versioning on our projects and pipelines. We can follow the official GitLab installation guide as a guideline and adjust to our needs.

5.3.1 GitLab Docker Configuration

```
services:
  gitlab:
       image: gitlab/gitlab-ce:latest
       container_name: gitlab
       restart: always
       stop_grace_period: 5m
       ports:
       - "444:443"
       - "81:80"
       - "23:22"
       volumes:
       - gitlab_data_config:/etc/gitlab
       - gitlab_data_logs:/var/log/gitlab
       - gitlab_data_data:/var/opt/gitlab
       shm_size: 256m
```

Let me explain line by line what we are doing here. First of all we are going to download and launch the latest version of the official GitLab Community Edition `image: gitlab/gitlab-ce:latest` . Then we are setting it a name by defining `container_name: gitlab` so we can tell it apart from the rest of our containers. In the next two lines `restart: always` and `stop_grace_period: 5m` we are telling to the system that it should restart in case of any momentary fail and when we are executing a stop command to not forcefully shutdown the container but wait for the container to close by itself in 5 minutes and if it hasn't exited yet then forcefully exit. This is done in order to be sure that we don't lose any data or stop any critical ongoing process. The `ports` part defines which ports the GitLab container will use and are in the format `<external port>:<internal port>` which is done in order to not alter the application's base configuration but we can still maintain with the external port that there is no conflict in our system as two services requesting the same port resulting in errors. For example the GitLab user interface is hosted on the 80 port by default but we may already have a service running on 80 which is a pretty common port for a web interface , so in order to avoid any conflicts I have set the ports to one above for the external port that we will be accessing from our host computer. Then the `volumes` part assigns the locations that we created in the previous chapter “Docker” . GitLab recommends to assign at least 256 megabytes of shared memory for the container which we do from the command `shm_size: 256m`.More information about default configuration for ports and connection can be found in the official documentation Package defaults | GitLab.

5.3.2 Login after docker compose up

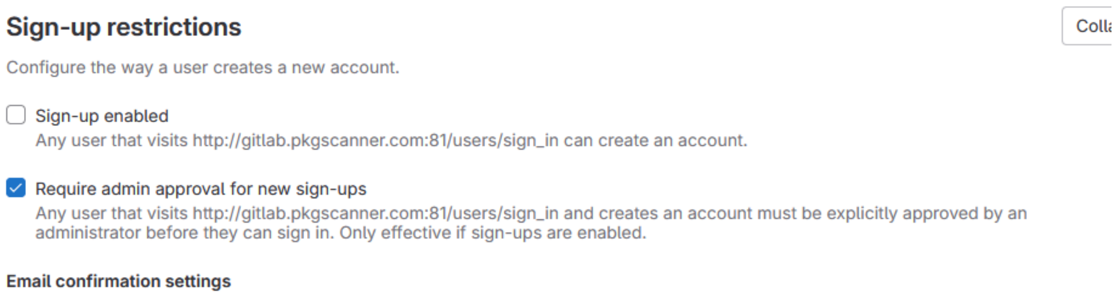
With the configuration that I have provided you will be able to open the link http://{your_VM_address}:81/users/sign_in and land on the sign in page of GitLab , but there is a problem it requires to log in when we haven't created any credentials. GitLab creates a password for the root user which can be retrieved by running.

```
sudo docker exec -it gitlab grep 'Password:'
/etc/gitlab/initial_root_password
```

After logging in please change your password and decouple your user from the root user.

5.3.3 Optional : Sign-up Restrictions

In a sandbox scenario it shouldn't matter but in a real world scenario we should always disable the sign-up option for new users which can be found under http://{your_VM_address}:81/admin/application_settings/general#js-signup-settings.



Sign-up restrictions

Configure the way a user creates a new account.

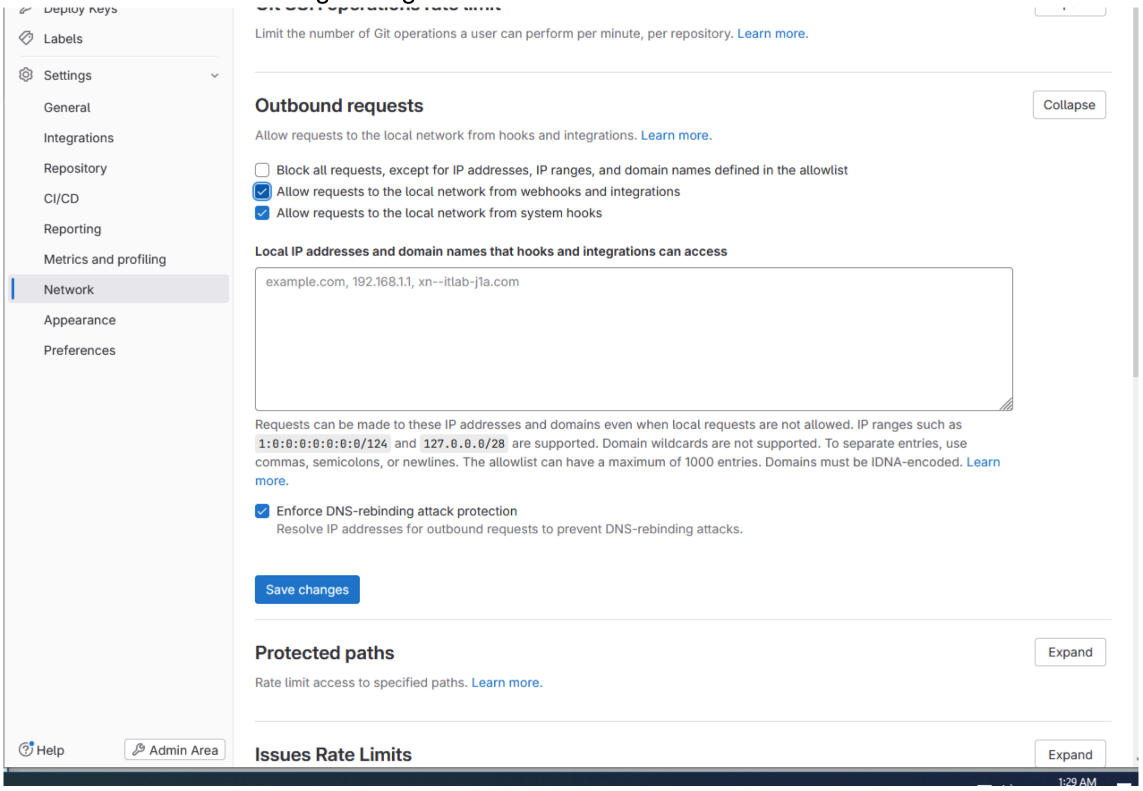
- [ ] Sign-up enabled
  Any user that visits http://gitlab.pkgscanner.com:81/users/sign_in can create an account.
- [x] Require admin approval for new sign-ups
  Any user that visits http://gitlab.pkgscanner.com:81/users/sign_in and creates an account must be explicitly approved by an administrator before they can sign in. Only effective if sign-ups are enabled.

Email confirmation settings

Image 5.7 : GitLab : Sign-up restrictions

5.3.4 Outbound requests

Since in sandbox mode we are running all the services under the same local network then we have to enable the following settings which can be found in the Admin Panel.



Limit the number of Git operations a user can perform per minute, per repository. Learn more.

Outbound requests

Allow requests to the local network from hooks and integrations. Learn more.

- [ ] Block all requests, except for IP addresses, IP ranges, and domain names defined in the allowlist
- [x] Allow requests to the local network from webhooks and integrations
- [x] Allow requests to the local network from system hooks

Local IP addresses and domain names that hooks and integrations can access

example.com, 192.168.1.1, xn--itlab-j1a.com

Requests can be made to these IP addresses and domains even when local requests are not allowed. IP ranges such as `1:0:0:0:0:0:0:0/124` and `127.0.0.0/28` are supported. Domain wildcards are not supported. To separate entries, use commas, semicolons, or newlines. The allowlist can have a maximum of 1000 entries. Domains must be IDNA-encoded. Learn more.

- [x] Enforce DNS-rebinding attack protection
  Resolve IP addresses for outbound requests to prevent DNS-rebinding attacks.

Save changes

Protected paths

Rate limit access to specified paths. Learn more.

Issues Rate Limits

Image 5.8 Outbound Requests

5.3.4 Add new projects

At this point you can add new projects in your GitLab instance. We are going to need one to store our Jenkins pipelines, I named mine 'PipelineConfigs', and one to save our test project. You can either create a new project or upload an existing one from your computer. When creating a new project GitLab also creates a 'README.md' file at the root level where it contains useful information on how to progress. It is highly advised to follow these steps in order to upload your project using Git.

5.4 Jenkins Installation

As mentioned in the "Automation Server" chapter we will use Jenkins. You can find general and installation information in the official repository GitHub - jenkinsci/docker: Docker official jenkins repo.

5.4.1 Jenkins Docker Configuration

```
  jenkins:
      image: jenkins/jenkins:lts-jdk17
      container_name: jenkins
      ports:
      - "8080:8080"
      - "50000:50000"
      restart: always
      stop_grace_period: 5m
      volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      networks:
      - jenkins_network
```

As you can see there is an overlap of variables used in order to set the Jenkins container, the only new one is the 'networks: -jenkins_network'. This line connects the container to the network named "jenkins_network" in order to provide communication with other services in the environment. Also we are mounting the Docker socket '/var/run/docker.sock' this is only required for the sandbox mode where all services run in containers under the VM further explanation will be given at a relative point.After the initial Jenkins installation when we try to access the UI from the http://{your_VM_address}:8080/login you can see that a login is expected even though we haven't set any credential yet. This is due to Jenkins auto generating an admin user along with a password assigned to that user. Jenkins informs the user where to find that password in the logs, in case you missed it the password can be found at /var/jenkins_home/secrets/initialAdminPassword.

```
nano /jenkins_home/secrets/initialAdminPassword
```

After having successfully logged in I recommend updating the password to something familiar and unique to you, also as in GitLab to create a user other than the one that was created at startup. When Jenkins is at a running state we should log in to the container and run the required commands to update and install any necessary packages that will be used from our projects.

```
sudo docker exec -it -u root jenkins /bin/bash
```

With this command we will log in to the Jenkins container with the user root at the /bin/bash shell.Once you are into the container run the following in order to update any dependencies that have updates.

```
apt-get update
```

5.4.2 Maven installation in Jenkins

Since the application that I will use to showcase the SAST and DAST pipelines is a Java Maven one then I proceed with the installation of Maven to Jenkins in order to be able to run Maven commands.

```
apt-get install maven
```

Once installation is finished guide yourself to the http://{your_VM_address}:8080/systemInfo there you can see all the System Information along with environmental variables. You will see that the HOME environmental variable matches the one that we provided in the docker-compose file. So inside your home directory for Maven and under the '.m2' folder add a file named 'settings.xml'. This file contains the information for Maven on where to search for packages and we specify two servers 'nexus-snapshots' and 'nexus-releases' while providing a username and a password in order to complete the authentication. As you can tell we are not inputting the username and password that we will use for Nexus because this file is saved in a non encrypted format instead we are passing variables that Jenkins will replace at runtime. Then we mirror the Maven Central repository by providing the url of our Nexus repository management. The way mirroring work is to first ask the provided repository and in case it fails then query the Maven Central Repository.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings
    xmlns="http://maven.apache.org/SETTINGS/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
[http://maven.apache.org/xsd/settings-1.1.0.xsd](http://maven.apache.org/xs
d/settings-1.1.0.xsd)">
    <servers>
       <server>
             <id>nexus-snapshots</id>
             <username>${env.MAVEN_NEXUS_SNAPSHOTS_USERNAME}</username>
             <password>${env.MAVEN_NEXUS_SNAPSHOTS_PASSWORD}</password>
       </server>
      <server>
            <id>nexus-releases</id>
            <username>${env.MAVEN_NEXUS_RELEASES_USERNAME}</username>
            <password>${env.MAVEN_NEXUS_RELEASES_PASSWORD}</password>
      </server>
   </servers>
   <mirrors>
      <mirror>
            <id>central</id>
            <name>central</name>
<url>http://{your_VM_address}:8081/repository/maven-group/</url>
            <mirrorOf>*</mirrorOf>
      </mirror>
   </mirrors>
</settings>
```

5.4.3 Required Plugins for Jenkins Pipelines

As previously mentioned there are a lot of plugins that we can utilize but I have narrowed it down to the following as the ones that we will use:

- GitLab Plugin
- SonarQube Scanner for Jenkins
- Config File Provider
- Remote Jenkinsfile Provider
- Multibranch Scan Webhook Trigger
- Pipeline Utility Steps

The dependencies on these plugins will either be installed as dependencies or Jenkins will ask you to proceed with the installation of those.

5.5 Sonatype Nexus Repository Installation

As mentioned in the “Repository Manager” chapter we will use Sonatype Nexus Repository. You can find general and installation information along the official image sonatype/nexus3 - Docker Image.

5.5.1 Sonatype Nexus Repository Docker Configuration

```
nexus:
     image: sonatype/nexus3
     container_name: nexus
     restart: always
     stop_grace_period: 5m
     ports:
     - "8081:8081"
     volumes:
     - nexus_data:/nexus-data
```

As you can tell by now we are following the same pattern of setting our containers in the docker-compose file. Declaring the image, the container name, handling what happens on failure and how to shutdown, along with declaration of ports used and the volume that we mount.

5.5.2 Login after docker compose up

The same pattern is followed here as well we can open the http://{your_VM_address}:8081/#browse/welcome welcome page but when we try to log in we can see that there is already a registered user named ádmin’. In order to retrieve the password of the ádmin’ user we are going to need to log in to the container from CLI.

```
sudo docker exec -it nexus /bin/bash
```

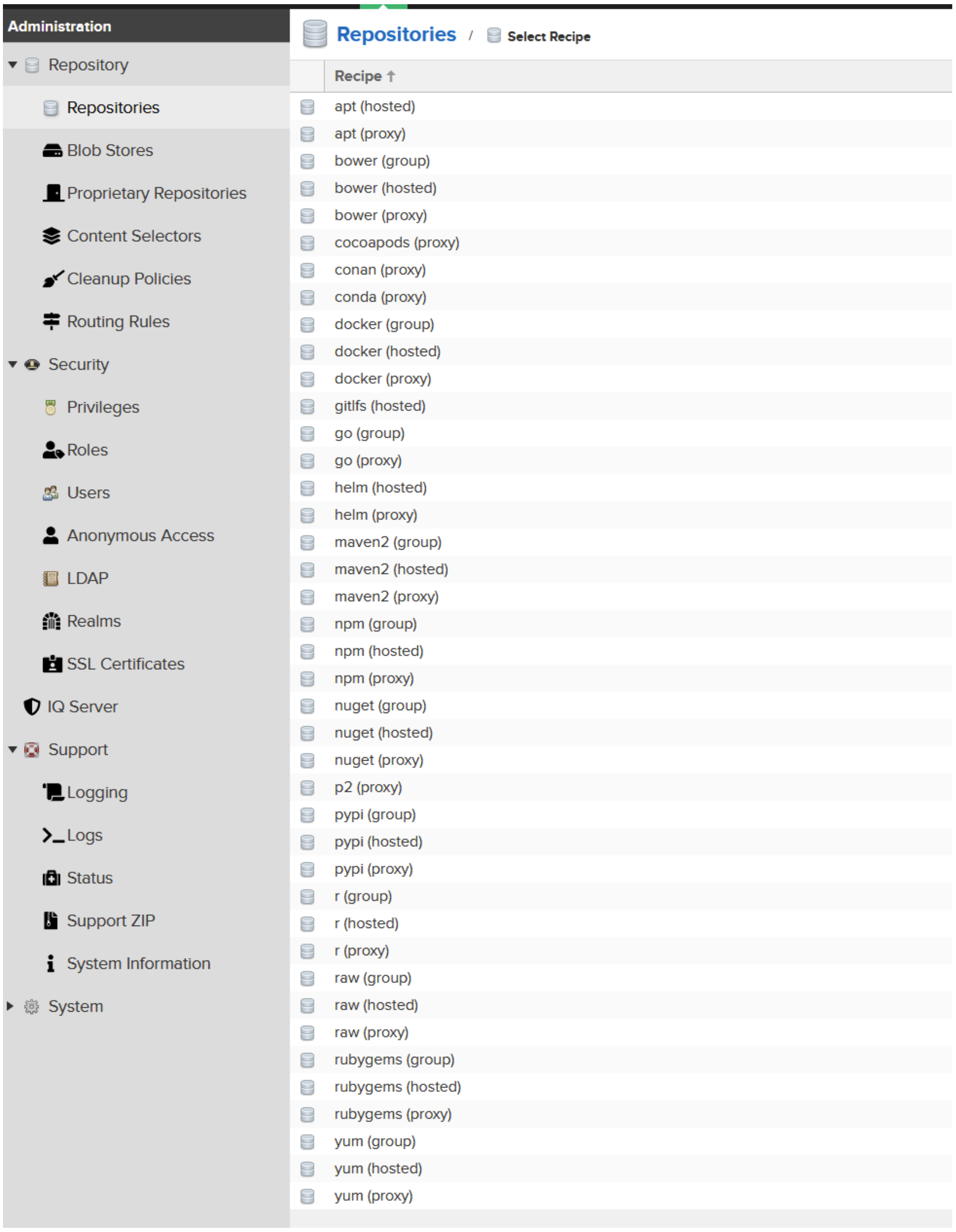
Once we are inside the container we can execute the following to display the password.

```
cat /nexus-data/admin.password
```

Once we login, changing the default password and creating another user for Jenkins is highly advised. Always remember to decouple your actions from the default users. Remember that you can always go to the official documentation page Sonatype Help.

5.5.3 Optional : Creating Repositories

Sonatype Nexus Repository comes out of the box with some pre configured repositories , verifying that the repositories that are needed by your project exist under the repositories tab. If you need to create a new one then you can go to the Administration panel , select Repositories from the side menu and click the button “Create repository”. A screen like the following will appear.



Recipe ↑
apt (hosted)
apt (proxy)
bower (group)
bower (hosted)
bower (proxy)
cocoapods (proxy)
conan (proxy)
conda (proxy)
docker (group)
docker (hosted)
docker (proxy)
gitlfs (hosted)
go (group)
go (proxy)
helm (hosted)
helm (proxy)
maven2 (group)
maven2 (hosted)
maven2 (proxy)
npm (group)
npm (hosted)
npm (proxy)
nuget (group)
nuget (hosted)
nuget (proxy)
p2 (proxy)
pypi (group)
pypi (hosted)
pypi (proxy)
r (group)
r (hosted)
r (proxy)
raw (group)
raw (hosted)
raw (proxy)
rubygems (group)
rubygems (hosted)
rubygems (proxy)
yum (group)
yum (hosted)
yum (proxy)

Image 5.9 : Sonatype Nexus Repository : Available repositories for creation

5.6 SonarQube Installation

Up to this point I haven't mentioned what tools we are going to use for the SAST and DAST part. SonarQube will be the tool that we use for the SAST. SonarQube is an open-source platform that inspects code quality, that means identifying possible bugs , vulnerabilities and security issues.

5.6.1 SonarQube Docker Configuration

```
sonarqube:
    image: sonarqube:lts-community
    depends_on:
    - db
    environment:
    SONAR_JDBC_URL: jdbc:postgresql://db:5432/sonar
    SONAR_JDBC_USERNAME: sonar
    SONAR_JDBC_PASSWORD: sonar
    container_name: sonarqube
    restart: always
    stop_grace_period: 5m
    ports:
    - "9000:9000"
    volumes:
    - sonarqube_data:/opt/sonarqube/data
    - sonarqube_extensions:/opt/sonarqube/extensions
    - sonarqube_logs:/opt/sonarqube/logs
db:
    image: postgres:12
    environment:
    POSTGRES_USER: sonar
    POSTGRES_PASSWORD: sonar
    volumes:
    - postgresql:/var/lib/postgresql
    - postgresql_data:/var/lib/postgresql/data
```

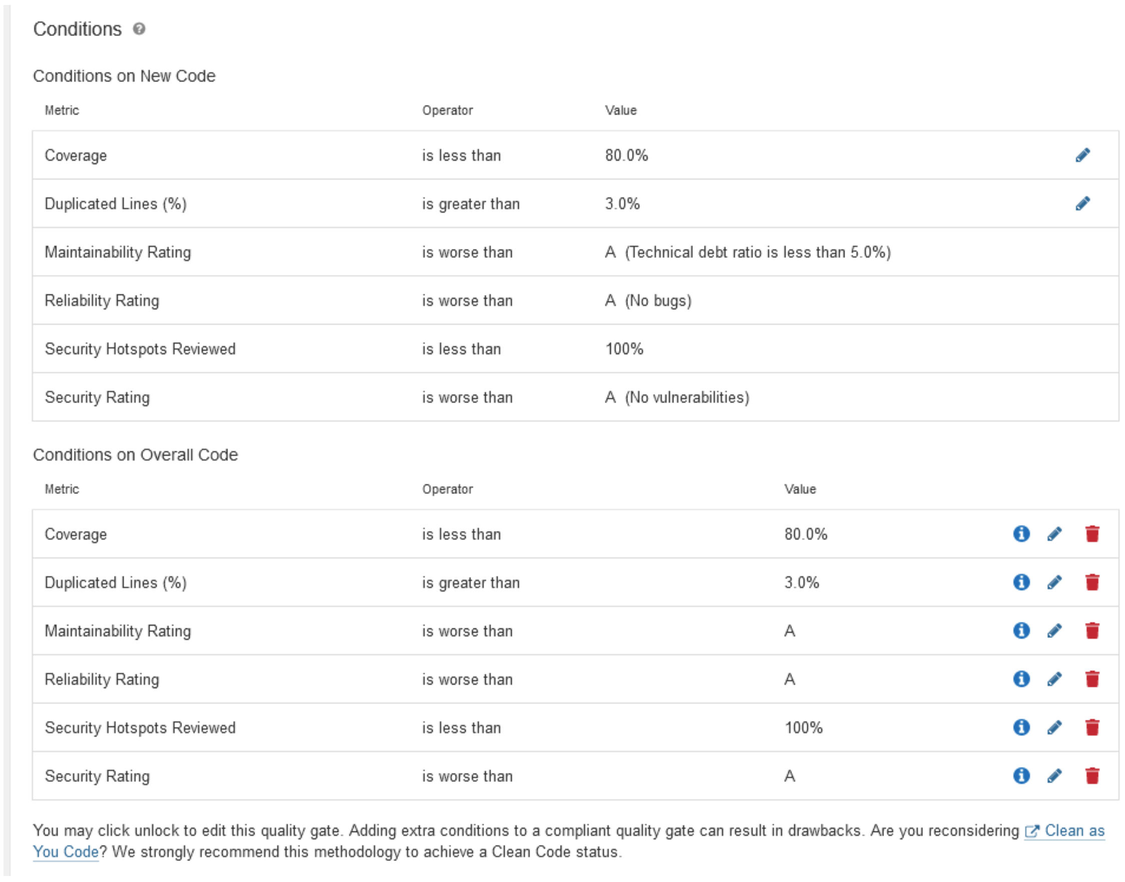
First of all we are following the official guide that can be found in Installing from Docker | SonarQube Docs and we are using the free image version as previously mentioned 'sonarqube:lts-community'. For SonarQube to run efficiently a database is required. We are creating the database under the 'db' section where we are using Postgres along with a default user for sonar to use. But first we are informing the SonarQube container to wait for the database creation with the 'depends_on' line. Then following the official documentation we are assigning SonarQube the default url of the connection and the username / password of the database user in order for SonarQube to complete the actions needed.

5.6.2 Login after docker compose up

After the container is at a ready state we can head to http://{your_VM_address}:9090 page where we can log in with the default credentials admin / admin. As it is now known you are advised to change the default password and create another user for usage.

5.6.3 Optional : Adjusting Quality Gate

The philosophy that SonarQube is based around is that it checks a repository's code and reports the issues that it spotted. This is what is called a Quality Gate based on metrics that we can adjust. We decide if we are "happy" based on the report with the state of the code and release a statement that says if we have a PASS or a FAIL. The correct way to work is when any kind of mistakes are found we fix them but as you can tell this is not the case with all the projects. If the problems are not fixed when first discovered, SonarQube will consider them known issues. This leads to a problem, executing back to back an analysis with no code change will open the Quality Gate. For this reason I have decided to set a new Quality Gate where you will set the conditions on Overall Code to match the ones of the conditions on New Code, as shown in the following image.


Conditions

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 5.0%)
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Conditions on Overall Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

You may click unlock to edit this quality gate. Adding extra conditions to a compliant quality gate can result in drawbacks. Are you reconsidering Clean as You Code? We strongly recommend this methodology to achieve a Clean Code status.

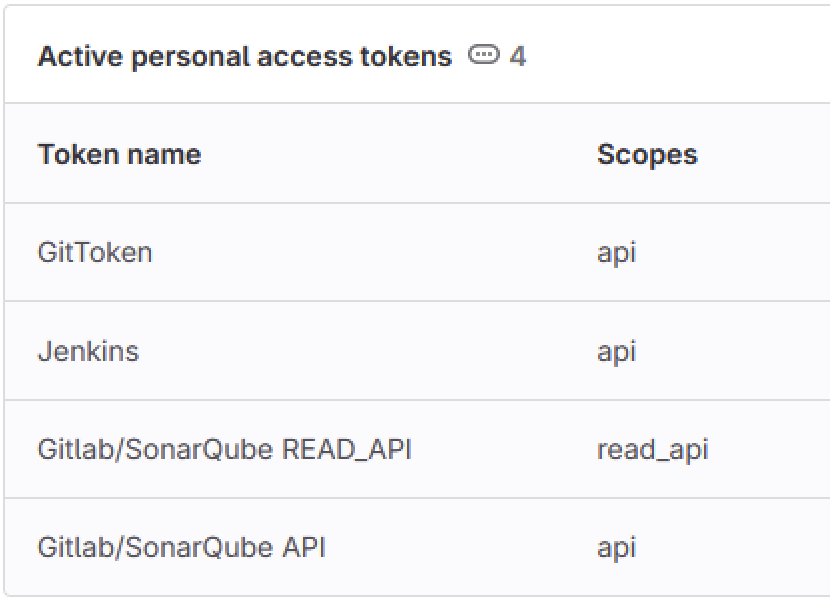
Image 5.10 : SonarQube : Adjusted Quality Gate

6. Ecosystem Interconnection

At this point we have created all the necessary services and they are running. It is time to connect everything with each other and complete the system installation.

6.1 GitLab tokens and Hook

We need to go into GitLab and create an environment user as I call him. That user will serve as the user Jenkins and SonarQube uses in order to connect to GitLab. After we have created that user we can go into the http://{your_VM_address}:81/-/user_settings/personal_access_tokens page and select on the top right "Add new token". I recommend adding at the token name the service that will use that token and the reason why it will use it if we need multiple of them. Proceed with the creation of the following access tokens. More information about Personal Access Tokens can be found in the Personal access tokens | GitLab.



Active personal access tokens 4

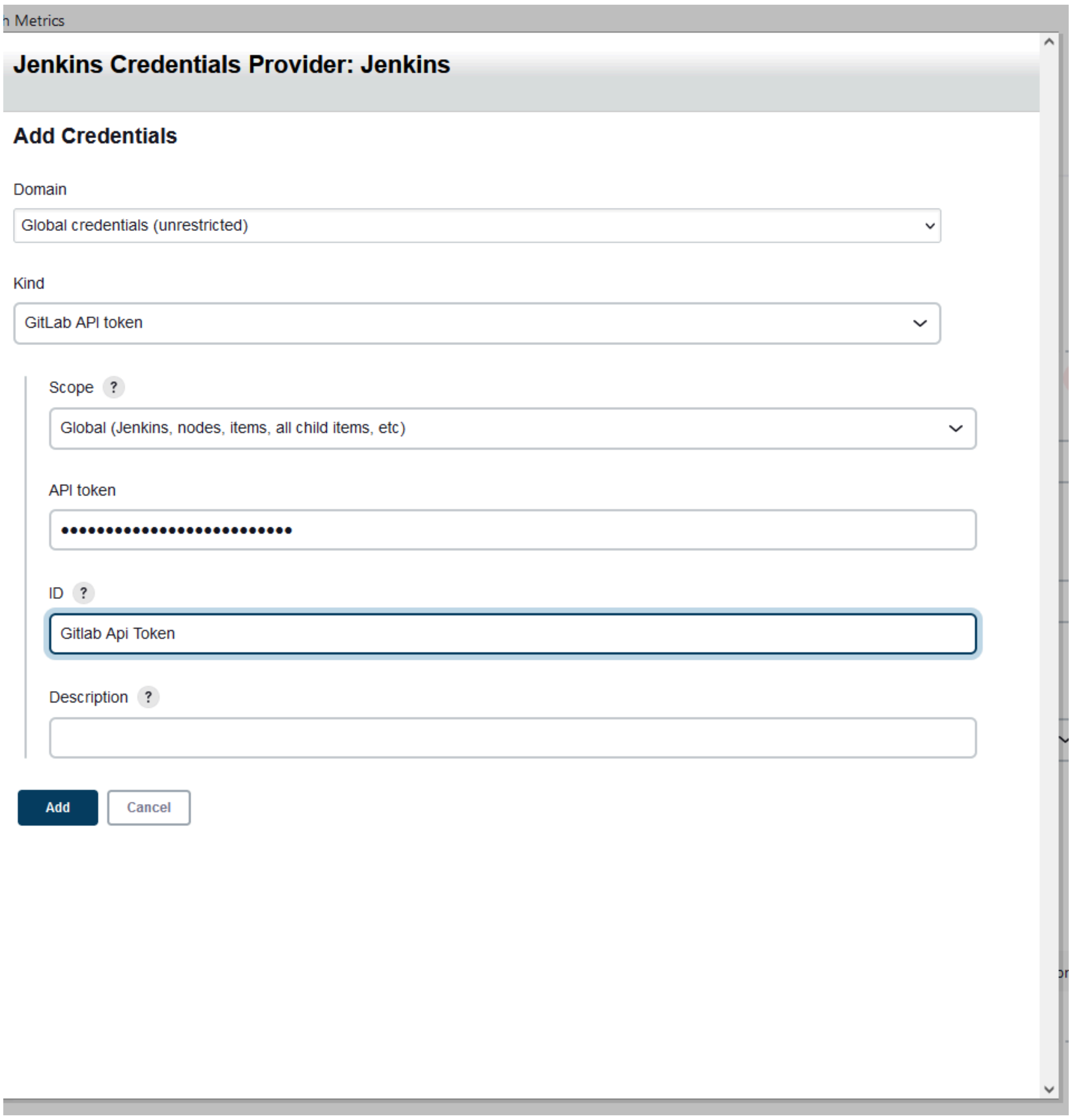
Token name	Scopes
GitToken	api
Jenkins	api
Gitlab/SonarQube READ_API	read_api
Gitlab/SonarQube API	api

Image 6.1 Gitlab : Personal Access Tokens

Then we can go under each Project that we have created and set Project Hooks. These settings can be found at the project screen under the option Webhooks. Select from the top right the option "Add new webhook" . At the URL you can set it as http://{your_VM_address}:8080/multibranch-webhook-trigger/invoke?token={unique_token}. This is the link towards your Jenkins instance where with a unique token that you can create will inform Jenkins every time that hook is triggered. We can configure when that hook is triggered by goin to the "Trigger" section and select what we want. Personally I believe that selecting "Push"events along with "Regular expression" which states the branch naming convention that we want it to trigger for , click "Save changes" when you have adjusted based on your needs.

6.2 Jenkins GitLab Connection

First of all we need to go and set into Jenkins the token that we have created from GitLab which is done under the http://{your_VM_address}:8080/manage/credentials/ address. Fill the following and replace with you token.


Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: GitLab API token

Scope: Global (Jenkins, nodes, items, all child items, etc)

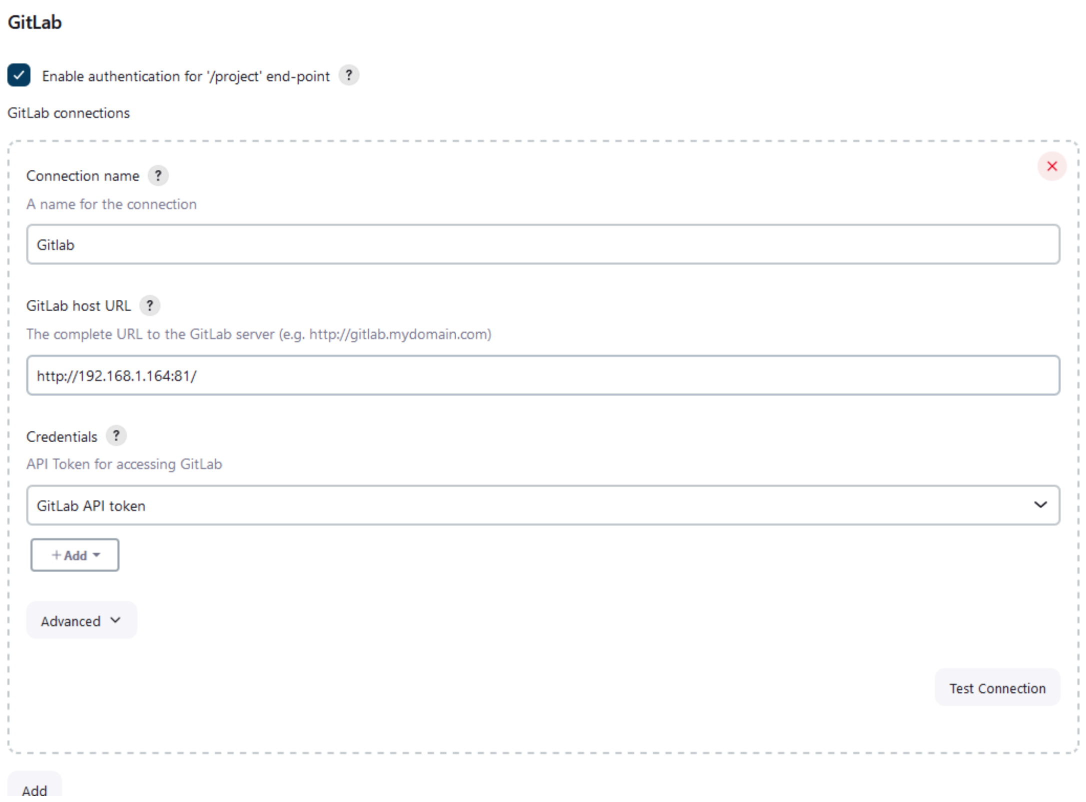
API token: ••••••••••••••••••••••••••

ID: Gitlab Api Token

Description:

Image 6.2 : Jenkins : Add Credential Screen

Then we can go and set that connection under the GitLab section in the page http://{your_VM_address}:8080/manage/configure and it should look like this.



GitLab

☑ Enable authentication for '/project' end-point

GitLab connections

Connection name
A name for the connection
Gitlab

GitLab host URL
The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)
http://192.168.1.164:81/

Credentials
API Token for accessing GitLab
GitLab API token

Image 6.3 : Jenkins : Setting GitLab connection

6.3 SonarQube GitLab connection

First of all we are going to create a token for Jenkins to use when connecting to SonarQube, this can be done under “Administration” then “Security-Users” and now for the wanted user you can select the “Tokens”option and create one. Now we can head back to the main page and select from top right the option “Create Project” and then “More” select the GitLab icon and fill the necessary data, this is where you will add the “api” token. Once you have set this a new prompt will be shown to grant access to your projects here you will add the “read_api” token. Now we can import our projects from GitLab to SonarQube. In order to set up the webhooks we can open a project and go to the following setting. More info about webhooks in SonarQube can be found under Webhooks.

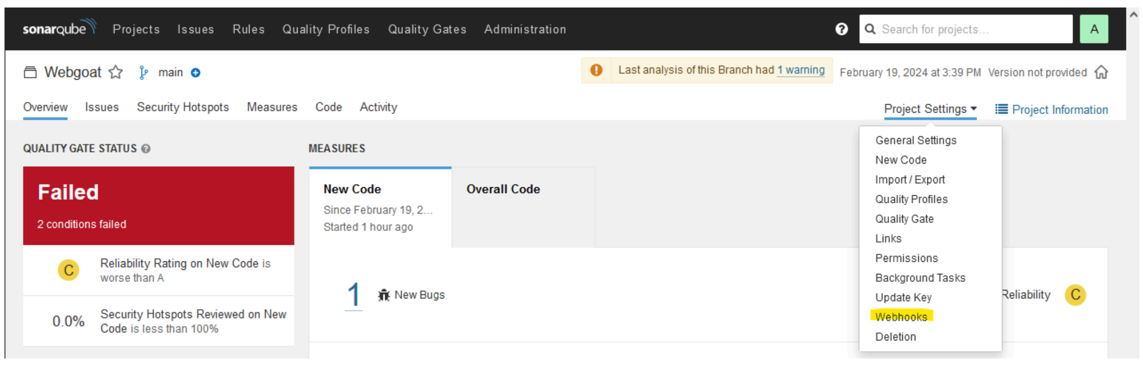
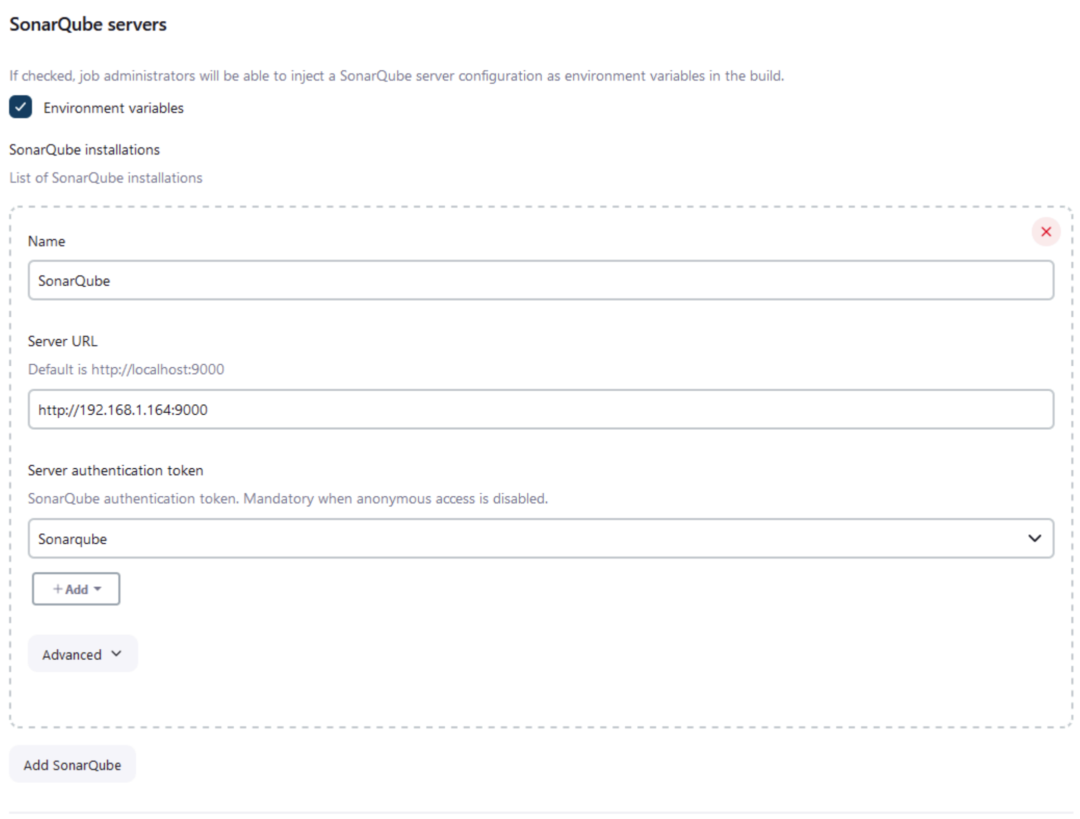


Image 6.4 : SonarQube : How to add webhooks

6.4 Jenkins SonarQube connection

As we did for GitLab , first we have to set the token that we have created and then we can go to the Jenkins page http://http://{your_VM_address}:8080/manage/configure and add the information regarding SonarQube.


SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

✓ Environment variables

SonarQube installations

List of SonarQube installations

Name: SonarQube

Server URL

Default is http://localhost:9000

http://192.168.1.164:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

Sonarqube

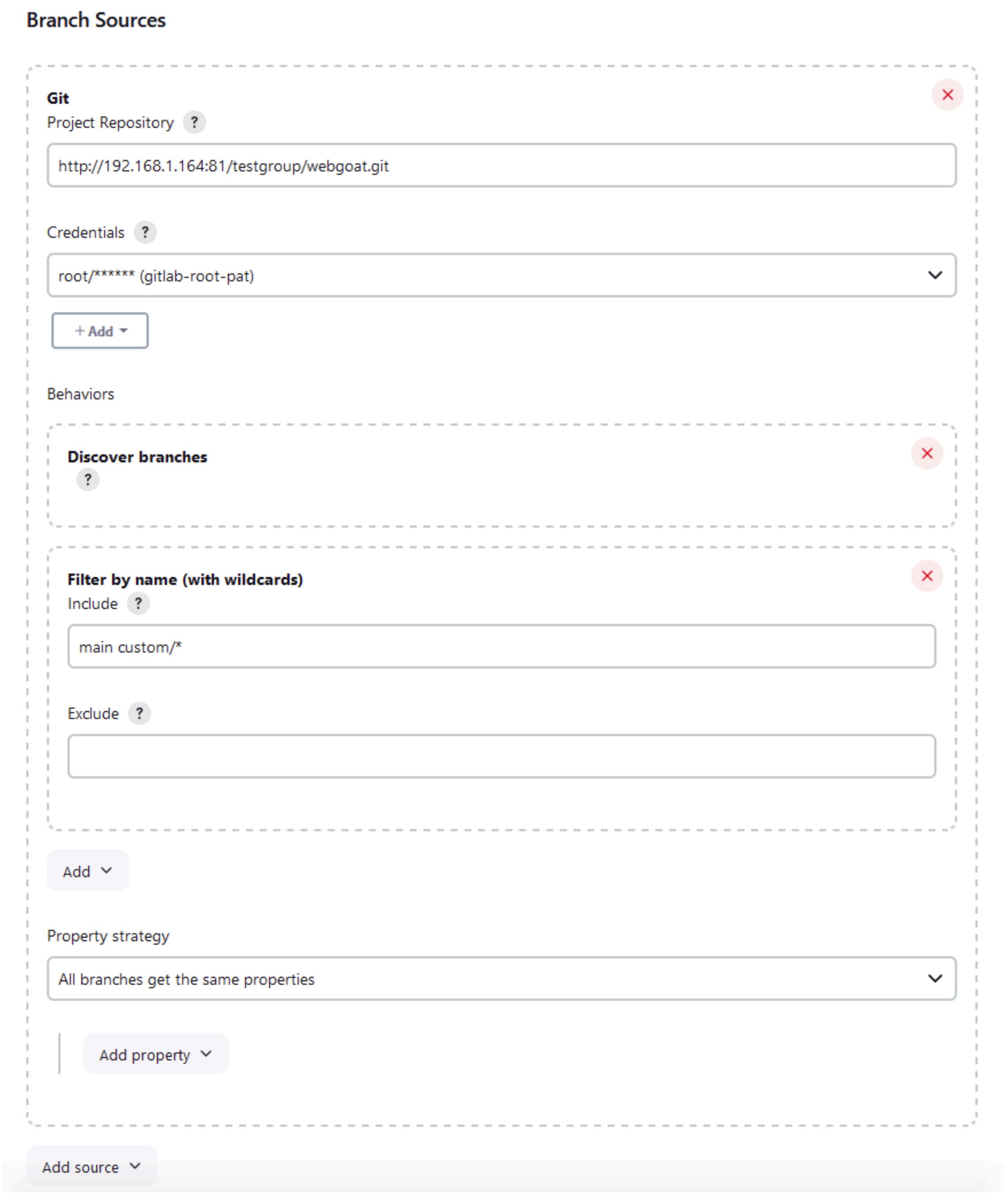
Image 6.5 : Jenkins : SonarQube addition

6.5 Adding to GitLab the PipelineConfig repository

If you haven't yet created a repository where you will save your pipelines now is the time to do so. I suggest not just copy the pipeline but to write it so you can understand what it does and what you need to change. I have split it into 2 major pipelines, one that runs the SAST with name "JenkinsFile_Advanced" along with the Maven operations and the publish to Nexus Repository Manager and one that runs the DAST with name JenkinsFile_Zap_WebGoat_Scan. Further explanations on the pipelines will be given at a later chapter.

6.5 Adding to Jenkins a MultiBranch Pipeline

If everything has been set correctly we can go to them home page of Jenkins select from the left side the “New Item” option, write our pipeline name which is usually the Project name and select MultiBranch Pipeline. First set the repository of the targeted project as follows and adjust based on your needs.


Branch Sources

Git

Project Repository: http://192.168.1.164:81/testgroup/webgoat.git

Credentials: root/****** (gitlab-root-pat)

Behaviors

Discover branches

Filter by name (with wildcards)

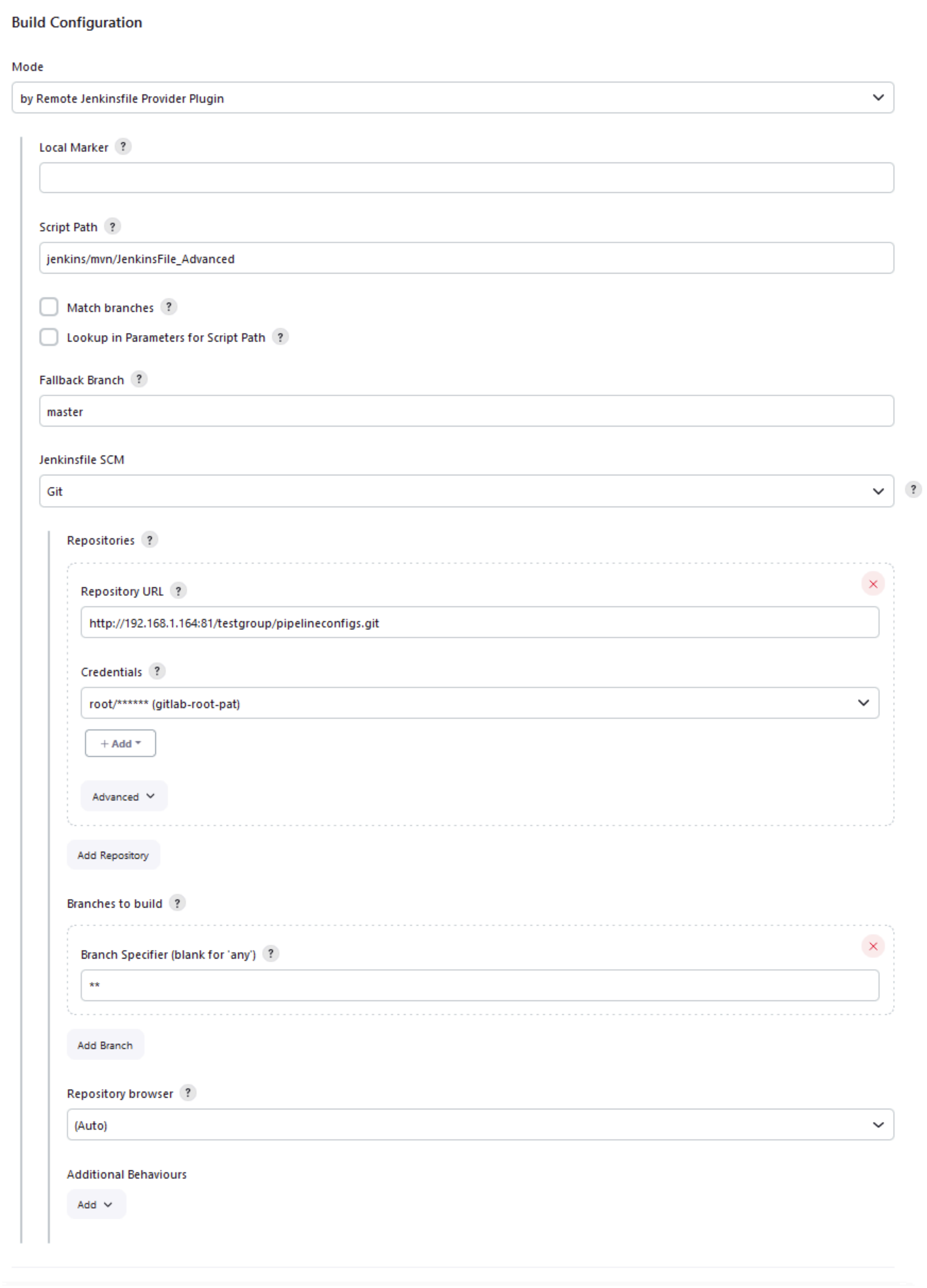
Include: main custom/*

Exclude:

Property strategy: All branches get the same properties

Image 6.6 : Jenkins : New MultiBranch Pipeline Git settings

After filling that information you will also need to specify to Jenkins where to look for the Jenkinsfile. In our case this file exists in the separate repository that we created. Fill in the required data as on the following image and adjust based on your environment.


Build Configuration

Mode: by Remote Jenkinsfile Provider Plugin

Local Marker:

Script Path: jenkins/mvn/JenkinsFile_Advanced

☐ Match branches

☐ Lookup in Parameters for Script Path

Fallback Branch: master

Jenkinsfile SCM: Git

Repositories

Repository URL: http://192.168.1.164:81/testgroup/pipelineconfigs.git

Credentials: root/****** (gitlab-root-pat)

Branches to build

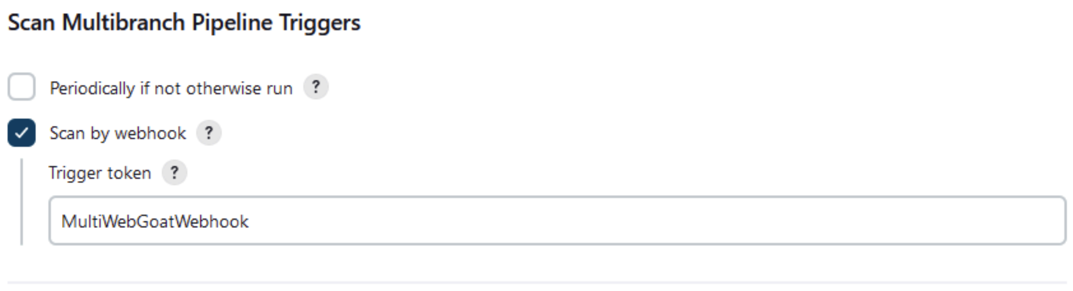
Branch Specifier (blank for 'any'): **

Repository browser: (Auto)

Additional Behaviours

Image 6.7 : Jenkins : Pipeline configuration

After we have filled that information we can also set the name of the webhook that we created before in GitLab for that project. The rest of the settings can remain empty but of course take a look at them and adjust based on your needs.



Scan Multibranch Pipeline Triggers

- [ ] Periodically if not otherwise run
- [x] Scan by webhook

Trigger token: MultiWebGoatWebhook

Image 6.8 : Jenkins : Configure pipeline Hook

7. Executing the pipelines

The time has finally come to execute our pipelines, this can be done in two ways either by starting the job by hand or triggering the webhook. I first show how this can be done by hand and then explain the process with the webhook.

7.1 Jenkinsfile Overview

Jenkinsfiles are written in groovy, which provides concise, familiar and easy to learn syntax which is based on Java. A Jenkinsfile can be split into two parts: the Environment and the Pipeline. In the environment we can declare global variables , functions and parameters while in the pipeline part we can use those declared in globals in order to adjust the outcome. The pipeline block consists of :

- Agents : this is where we declare who will run the pipeline. By default Jenkins executes the pipeline in the same instance but based on the official documentation it is highly advised to set a Master/Slave instance relationship in order to not affect Jenkins as a service.
- Stages: During the pipeline we are splitting the major parts of the work into stages. Those stages should be either isolated or dependent on a previous status. This is why those stages run in a linear fashion and not in parallel.
- Steps : Each Stage can consist of multiple steps. In order to achieve the results that we want we take it one step at the time so the pipeline can be readable as our code should be.
- Post : What actions we want to do when all the stages have finished.

You can think of a Jenkinsfile as a cooking recipe, you gather the ingredients and the tools and you make the food by following sequential steps. The result is always consistent as long as there is no random factor. More info can be found in Pipeline Syntax and Pipeline: Basic Steps.

7.2 WebGoat

WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons.

This program is a demonstration of common server-side application flaws. The exercises are intended to be used by people to learn about application security and penetration testing techniques.

WARNING 1: While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program. WebGoat's default configuration binds to localhost to minimize the exposure.

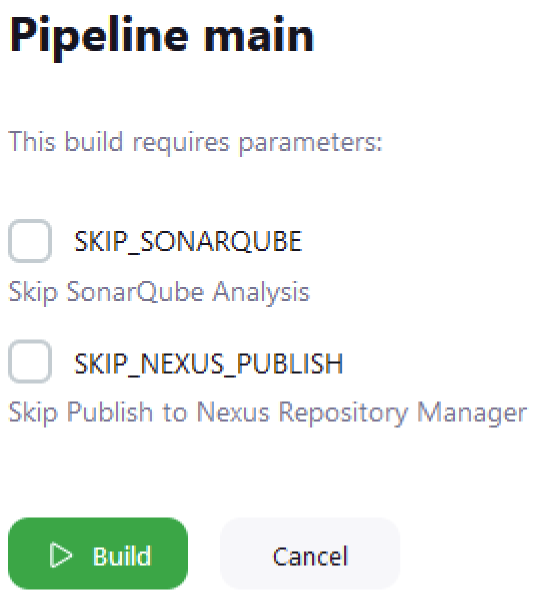
WARNING 2: This program is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get caught. If you are caught engaging in unauthorized hacking, most companies will fire you. Claiming that you were doing security research will not work as that is the first thing that all hackers claim.

GIthub : WebGoat

WebGoat is a very good example that we can run our pipelines against. It is intentionally an insecure application with vulnerabilities in order for people to practice their security skills.

7.3 SAST Pipeline

The first time that we open a pipeline we will see that it is empty and our created branches do not show up. This is due to Jenkins not being aware yet what exists, we can make Jenkins aware by clicking the button “Scan Multibranch Pipeline Now”. This opens a connection to the GitLab where it will connect with the provided information and then run a few commands to fetch any branches that match the name that we gave in the settings. Then our branches will appear, open the one that you want the pipeline to run on. Since Jenkins has not yet retrieved the Jenkinsfile you will need to run the job by pressing on the left “Build”, this job will fail on its own due to parameters that I have set in the Jenkinsfile. Once it has failed you will see that in the “Stage View” as Jenkins calls it. The “Build” button has changed to “Build with Parameters” if you click it the following will appear.


Pipeline main

This build requires parameters:

☐ SKIP_SONARQUBE

Skip SonarQube Analysis

☐ SKIP_NEXUS_PUBLISH

Skip Publish to Nexus Repository Manager

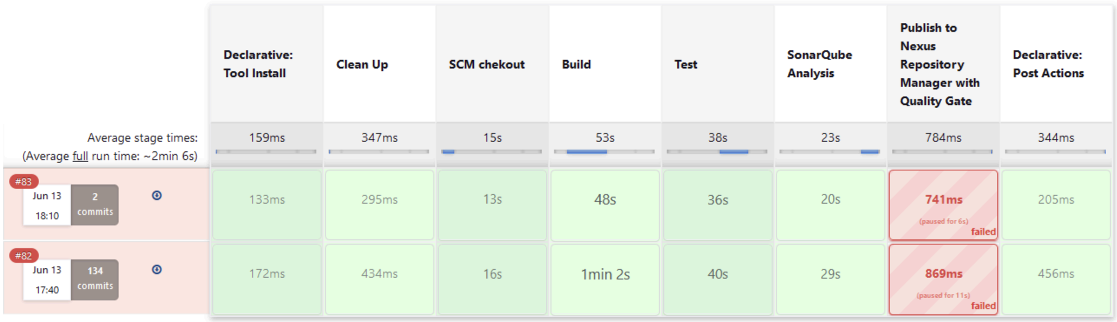
Image 7.1 : Jenkins : Build with parameters

These parameters are set into our Jenkinsfile by the following part of code.

```
parameters {
      booleanParam(name: 'SKIP_SONARQUBE',
      description: 'Skip SonarQube Analysis',
      defaultValue: false)
      booleanParam(name: 'SKIP_NEXUS_PUBLISH',
      description: 'Skip Publish to Nexus Repository Manager',
      defaultValue: false)
  }
```

We declare that it will be a boolean parameter so it has two states “true” or “false” and is represented in the UI by a checkbox. Then we set a name , a description for the user to understand what that parameter does and a defaultValue. Jenkins community here advises that we add another boolean parameter named “CONFIRMATION” which is the first thing that we check when pipeline starts in order to see if it was a miss-click by the user. I considered that option supplementary but not critical enough to add it depending on your pipeline feel free to add it. When we select ok and the pipeline runs we can see that a table appears like the following one. As you can tell each column is a stage of the pipeline , Jenkins informs us with color coding:

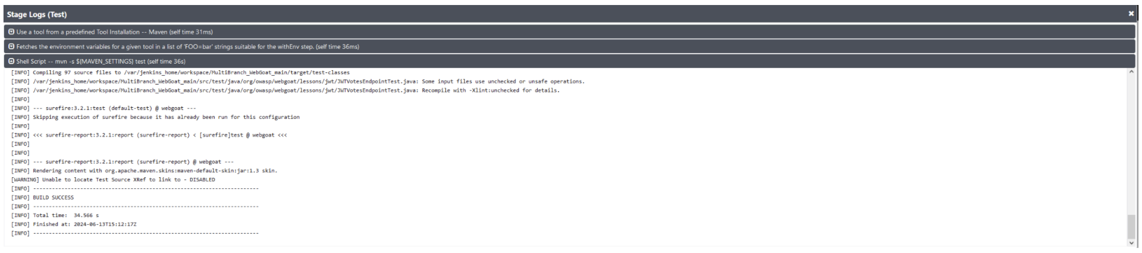
- Green color if a stage was successful , this is called SUCCESSFUL
- Yellow if the result was unexpected but not a failing one , this is called UNSTABLE
- Red if the result is a failing one , this is called FAILURE



	Declarative: Tool Install	Clean Up	SCM chekout	Build	Test	SonarQube Analysis	Publish to Nexus Repository Manager with Quality Gate	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 6s)	159ms	347ms	15s	53s	38s	23s	784ms	344ms
#83 Jun 13 18:10 2 commits	133ms	295ms	13s	48s	36s	20s	741ms (paused for 6s) failed	205ms
#82 Jun 13 17:40 134 commits	172ms	434ms	16s	1min 2s	40s	29s	869ms (paused for 11s) failed	456ms

Image 7.2 : Jenkins : SAST Pipeline

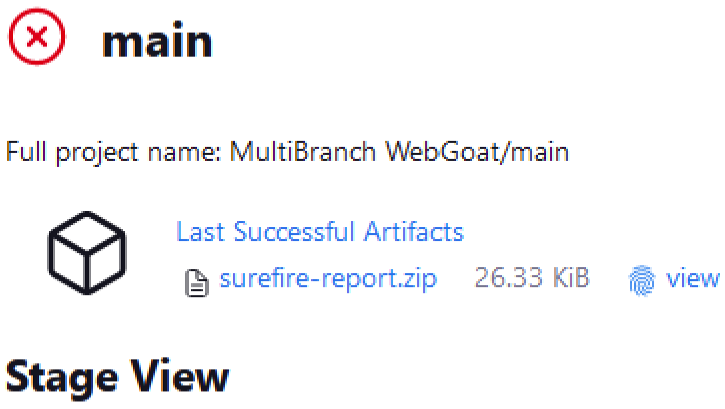
If we select a step then a window opens with logs from that step.



```
Stage Logs (Test)
⊟ Use a tool from a predefined Tool Installation -- Maven (self time 31ms)
⊟ Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. (self time 36ms)
⊟ Shell Script -- mvn -s ${MAVEN_SETTINGS} test (self time 36s)
[INFO] Compiling 97 source files to /var/jenkins_home/workspace/MultiBranch_WebGoat_main/target/test-classes
[INFO] /var/jenkins_home/workspace/MultiBranch_WebGoat_main/src/test/java/org/owasp/webgoat/lessons/jwt/JWTVotesEndpointTest.java: Some input files use unchecked or unsafe operations.
[INFO] /var/jenkins_home/workspace/MultiBranch_WebGoat_main/src/test/java/org/owasp/webgoat/lessons/jwt/JWTVotesEndpointTest.java: Recompile with -Xlint:unchecked for details.
[INFO]
[INFO] --- surefire:3.2.1:test (default-test) @ webgoat ---
[INFO] Skipping execution of surefire because it has already been run for this configuration
[INFO]
[INFO] <<< surefire-report:3.2.1:report (surefire-report) < [surefire]test @ webgoat <<<
[INFO]
[INFO]
[INFO] --- surefire-report:3.2.1:report (surefire-report) @ webgoat ---
[INFO] Rendering content with org.apache.maven.skins:maven-default-skin:jar:1.3 skin.
[WARNING] Unable to locate Test Source XRef to link to - DISABLED
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  34.566 s
[INFO] Finished at: 2024-06-13T15:12:17Z
[INFO] ------------------------------------------------------------------------
```

Image 7.3 : Jenkins : Stage logs

After it is finished we can see that the latest status exists on the top of the screen, for example here we can see that for the main branch we have a fail status (red x in the circle) and the last successful result produced a zip. Let's open the last run by clicking the number id of the build.



main

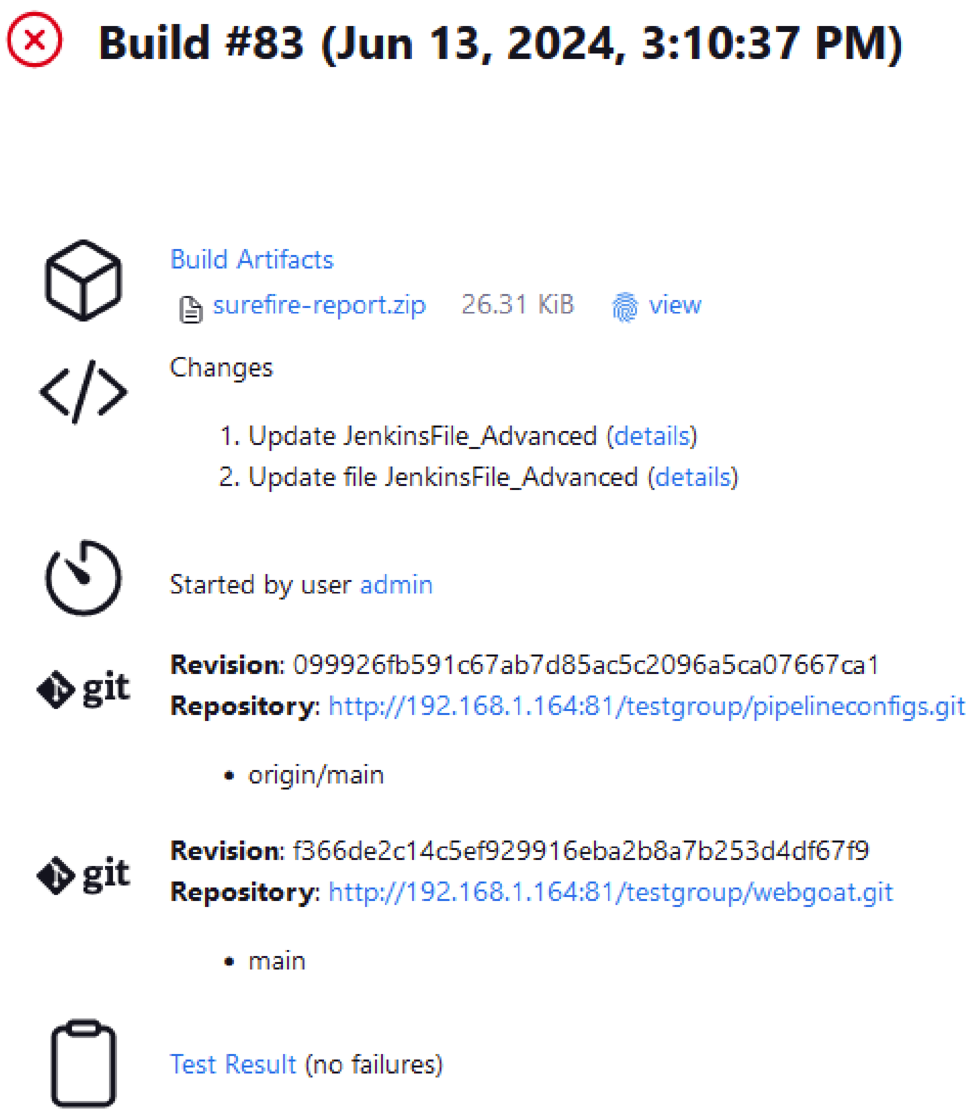
Full project name: MultiBranch WebGoat/main

Last Successful Artifacts
surefire-report.zip 26.33 KiB view

Stage View

Image 7.4 : Jenkins : Branch status

When we open the specific build number id we can see information about the build. Produced artifacts , latest commits from GitLab , who triggered that job and at which revision of the repository the build was triggered for.



Build #83 (Jun 13, 2024, 3:10:37 PM)

Build Artifacts

surefire-report.zip 26.31 KiB view

Changes

1. Update JenkinsFile_Advanced (details)
2. Update file JenkinsFile_Advanced (details)

Started by user admin

Revision: 099926fb591c67ab7d85ac5c2096a5ca07667ca1
Repository: http://192.168.1.164:81/testgroup/pipelineconfigs.git

- origin/main

Revision: f366de2c14c5ef929916eba2b8a7b253d4df67f9
Repository: http://192.168.1.164:81/testgroup/webgoat.git

- main

Test Result (no failures)

Image 7.5 : Jenkins : Build information

If we select the “Test Results” we can see that during our pipeline the test stage run had the following results.

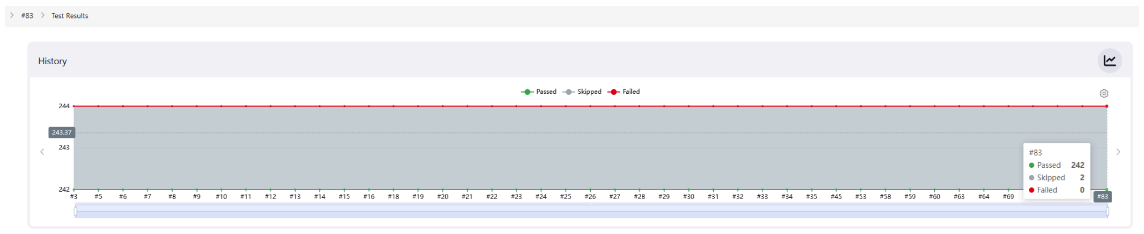
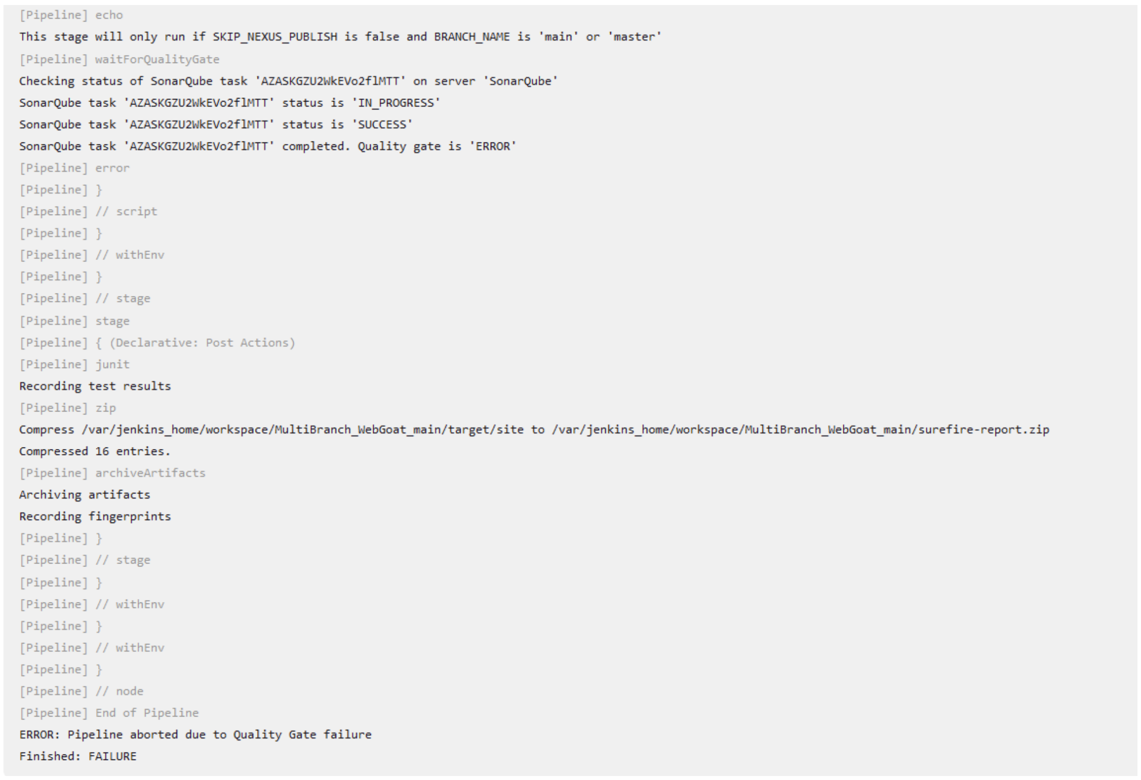


Image 7.6 Jenkins : Build Test results

If we want to see the logs further we can always select them from the left side and see what happened in detail.



```
[Pipeline] echo
This stage will only run if SKIP_NEXUS_PUBLISH is false and BRANCH_NAME is 'main' or 'master'
[Pipeline] waitForQualityGate
Checking status of SonarQube task 'AZASKGZU2WkEVo2flMTT' on server 'SonarQube'
SonarQube task 'AZASKGZU2WkEVo2flMTT' status is 'IN_PROGRESS'
SonarQube task 'AZASKGZU2WkEVo2flMTT' status is 'SUCCESS'
SonarQube task 'AZASKGZU2WkEVo2flMTT' completed. Quality gate is 'ERROR'
[Pipeline] error
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] junit
Recording test results
[Pipeline] zip
Compress /var/jenkins_home/workspace/MultiBranch_WebGoat_main/target/site to /var/jenkins_home/workspace/MultiBranch_WebGoat_main/surefire-report.zip
Compressed 16 entries.
[Pipeline] archiveArtifacts
Archiving artifacts
Recording fingerprints
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: Pipeline aborted due to Quality Gate failure
Finished: FAILURE
```

Image 7.7 : Jenkins : Build logs

Now that you are familiar with the UI, let's continue explaining the Jenkinsfile based on the order that stages are being run. The first two stages that we execute is "Clean Up" and "SCM chekout". The first one runs the native function ‘cleanWs’ which means clean workspace and the second one is to checkout the selected project that we set in the settings of the pipeline. After this stage is completed the project exists in the workspace.

```
stage("Clean Up"){
            steps {
            cleanWs()
            }
      }

      stage("SCM chekout"){
            steps {
            checkout scm
            }
      }
```

Then we are executing the "Build" and "Test" stages for which we are running a maven command to trigger a specific lifecycle phase. First that the project can compile it without running any tests and then running the tests that exist inside of the project. Although those tests are not a part of the SAST they can declare the project quality based on their status. In both steps we can see that another method is called ‘withCredentials’ this takes the settings.xml that we configured at the start and indicates to maven that it should use those settings in order to execute.

```
stage("Build"){
            steps {

                withCredentials([file(credentialsId: 'maven-settings',
variable: 'MAVEN_SETTINGS')]) {
                sh 'mvn -s ${MAVEN_SETTINGS} clean install -DskipTests'
                }
            }
        }

        stage("Test"){
            steps {

                withCredentials([file(credentialsId: 'maven-settings',
variable: 'MAVEN_SETTINGS')]) {
                sh 'mvn -s ${MAVEN_SETTINGS} test'
                }
            }
        }
```

After that we execute the "SonarQube Analysis" stage which first checks if we want to run this step by checking that parameter's SKIP_SONARQUBE value. Another addition that I made is that for this job to not run this stage if the branch name is not 'main' or 'master' this is a personal choice based on the CI/CD principles as I consider that personal branches do not need the info from Sonarqube and if someone needs it, it can be executed locally through add-ons in their IDE.Then by using environmental variables we are running a command which triggers the SonarQube analysis based on the project's Sonar key which can be retrieved in the SonarQube UI and I scan the file that is created in the build part 'target' which contains all the files and produced artifacts from the "Build" stage. As you can see I have left in this stage some lines that start with '//' these are comments and as I state there running the commented command produces only errors found under 'src' and since the project doesn't contain only Java code this is not optimal.

```
stage("SonarQube Analysis"){
            steps{
                script {
                    echo "SKIP_SONARQUBE parameter value:
${params.SKIP_SONARQUBE}"
                    echo "Running branch is : ${env.BRANCH_NAME}"
                    echo "Running job is : ${env.JOB_NAME}"
                    if (!params.SKIP_SONARQUBE && (env.BRANCH_NAME ==
'main' || env.BRANCH_NAME == 'master')) {
                        echo "This stage will only run if SKIP_SONARQUBE is
false and BRANCH_NAME is 'main' or 'master'"
                         scannerHome = tool "${SONARQUBE_NAME}";
                         withSonarQubeEnv("${SONARQUBE_NAME}") {

                            sh """
                                         ${scannerHome}/bin/sonar-scanner
\
-Dsonar.projectKey=${SONAR_PROJECT_KEY} \
                                          -Dsonar.sources=. \
                            -Dsonar.java.binaries=target/
                                        """

                            // Running the following for SonarQube produces
only errors found under src and since we have projects with not only java
it is not optimal
                            // withCredentials([file(credentialsId:
'maven-settings', variable: 'MAVEN_SETTINGS'),
                            //                  string(credentialsId:
'Sonarqube', variable: 'SONARQUBE_TOKEN')]) {
                            //     sh 'mvn -s ${MAVEN_SETTINGS} sonar:sonar
-Dsonar.token=${SONARQUBE_TOKEN} -Dsonar.projectKey=${SONAR_PROJECT_KEY}'
                            // }

                        }
                    } else {
                        echo "Skipping this stage as SKIP_SONARQUBE is true
or BRANCH_NAME is not 'main' or 'master'"
                    }
                }
            }
        }
```

Our last stage is the "Publish to Nexus Repository Manager with Quality Gate" as explained before under the Nexus Repository Manager section we have modified the default Quality Gate. First we apply the same logic as the previous stage that we don’t need to publish an image unless it is one from the main or the master that we would host in an environment. Then we check the status of the Quality Gate if it is not OK we are failing the build. If the status is OK then we proceed by taking the produced artifact under ‘target’ and we get from the pom.xml file the packaging and version of the project. Then we run the provided ‘nexusArtifactUploader’ function with the required parameters and our upload has been completed.

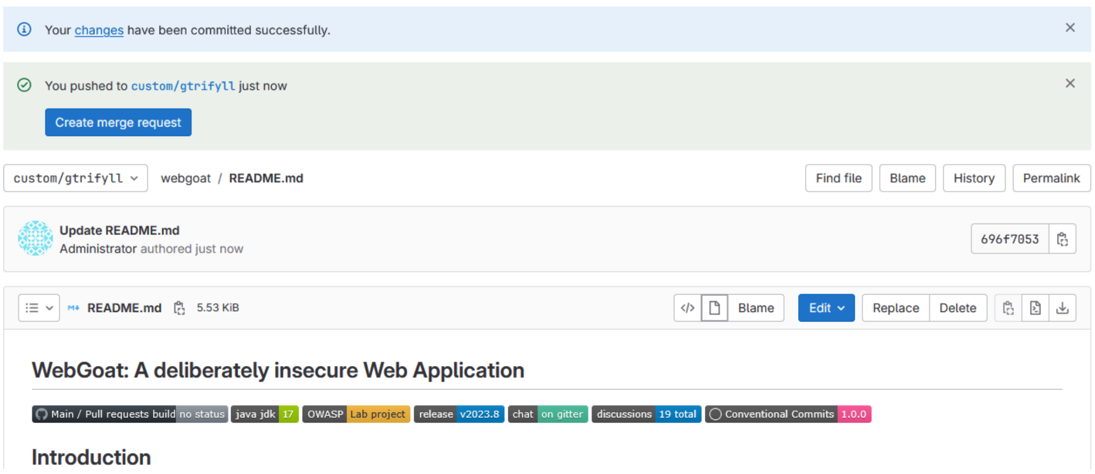
```
stage("Publish to Nexus Repository Manager with Quality Gate") {
            steps {
                script {
                    echo "SKIP_NEXUS_PUBLISH parameter value:
${params.SKIP_NEXUS_PUBLISH}"
                    echo "Running branch is : ${env.BRANCH_NAME}"
                    if (!params.SKIP_NEXUS_PUBLISH && (env.BRANCH_NAME ==
'main' || env.BRANCH_NAME == 'master')) {
                        echo "This stage will only run if
SKIP_NEXUS_PUBLISH is false and BRANCH_NAME is 'main' or 'master'"
                        // Check the status of the Quality Gate
                        // If it fails, mark the build as failed
                        if (waitForQualityGate().status != 'OK') {
                        error "Pipeline aborted due to Quality Gate
failure"
                        }
                        pom = readMavenPom file: "pom.xml";
                        filesByGlob = findFiles(glob:
"target/*.${pom.packaging}");
                        echo "${filesByGlob[0].name} ${filesByGlob[0].path}
${filesByGlob[0].directory} ${filesByGlob[0].length}
${filesByGlob[0].lastModified}"
                        artifactPath = filesByGlob[0].path;
                        artifactExists = fileExists artifactPath;
                        if(artifactExists) {
                            echo "*** File: ${artifactPath}, group:
${pom.groupId}, packaging: ${pom.packaging}, version ${pom.version}";
                            nexusArtifactUploader(
                                nexusVersion: NEXUS_VERSION,
                                protocol: NEXUS_PROTOCOL,
                                nexusUrl: NEXUS_URL,
                                groupId: pom.groupId,
                                version: pom.version,
                                repository: NEXUS_REPOSITORY,
                                credentialsId: NEXUS_CREDENTIAL_ID,
                                artifacts: [
                                    [artifactId: pom.artifactId,
                                    classifier: '',
                                    file: artifactPath,
                                    type: pom.packaging],
                                    [artifactId: pom.artifactId,
                                    classifier: '',
                                    file: "pom.xml",
                                    type: "pom"]
                                ]
                            );
                        } else {
                            error "*** File: ${artifactPath}, could not be
found";
                        }
                    } else {
                        echo "Skipping this stage as SKIP_SONARQUBE is true
or BRANCH_NAME is not 'main' or 'master'"
                    }
                }
            }
        }

    }
```

This concluded the stages and now it is time to run the post section in which we select always meaning that regardless of the build status try to execute the following commands. The first one publishes into Jenkins the Surefire test report that we executed in the “Test” stage and then we put it into a zip file named surefire-report.zip and archive it. This zip file now exists as long as there are data about that build in order to have a history that we can get back to.

```
post {
        always {
            // Publish Surefire test report
            junit skipPublishingChecks: true, testResults:
'target/surefire-reports/*.xml'

             // Zip the test results directory
            zip zipFile: 'surefire-report.zip', archive: false, dir:
'target/site'
            // Archive the zip file
            archiveArtifacts artifacts: 'surefire-report.zip', fingerprint:
true
        }
    }
```

7.3.1 Hooks theory

Although we have set hooks we have not used them up till now.



Your changes have been committed successfully.

You pushed to custom/gtrifyll just now

Update README.md
Administrator authored just now

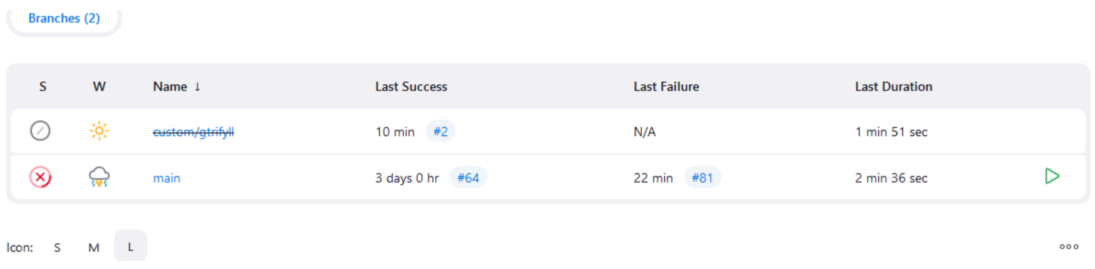
696f7053

WebGoat: A deliberately insecure Web Application

Introduction

Image 7.8 : GitLab : Create new Branch

With the way that we have configured the environment and our pipeline when we are pushing changes to a branch that matches the regex that we have specified , in this case custom/.* , the hook is triggered and lets Jenkins know to start a Build for that branch. The logic here is to run the basic stages of “Build” and “Test” in order to have a confirmation that our changes did not introduce any regression , so they can be merged into main when the time comes. As soon as that custom branch is merged into the main branch then a new Build is triggered for the main branch which now includes the new changes and runs the SAST part as well.



S	W	Name ↓	Last Success	Last Failure	Last Duration
		~~custom/gtrifyll~~	10 min #2	N/A	1 min 51 sec
		main	3 days 0 hr #64	22 min #81	2 min 36 sec

Image 7.9 : Jenkins : Hook triggered when we merge the custom branch

7.4 DAST Pipeline

7.4.1 Zed Attack Proxy

Up to this point I have only mentioned DAST but haven't explained what is the purpose of it and what tool we are going to use. Running a DAST against an application means to simulate attacks like a malicious character would do, meaning that it is a black box testing focused on the security part of the application. Running these simulations means finding these vulnerabilities and fixing them before any external user has access to the application. This helps with the protection of user data from cyber threats. DAST always should be run alongside SAST as each kind of testing can produce results that the other activity cannot, since DAST is executed at run time while SAST has full access to the code base.

For the DAST part we are going to utilize Zed Attack Proxy (ZAP).

Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of The Software Security Project (SSP). ZAP is designed specifically for testing web applications and is both flexible and extensible.

At its core, ZAP is what is known as a "man-in-the-middle proxy." It stands between the tester's browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. It can be used as a stand-alone application, and as a daemon process.

ZAP – Getting Started

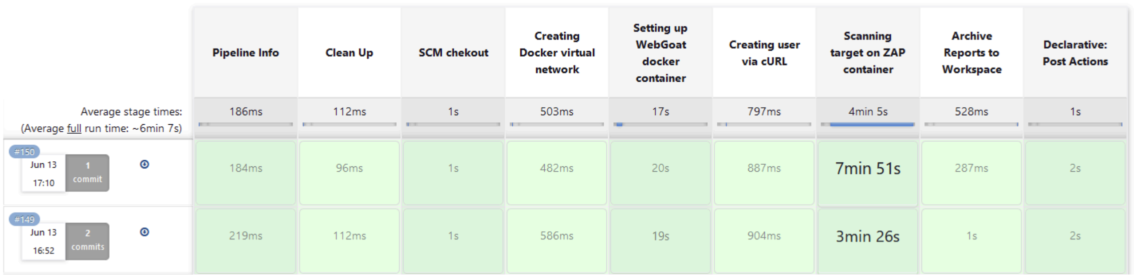
Besides that the reason it was our selected tool is because there is Docker version of it which we can host and run and because there is a growing community behind it with public forums and developers are active in it meaning easy troubleshooting for us. Also there is a plethora of addons that we can utilize or even create our own for our project needs. When ZAP is executed from a Docker container there are a few steps that are being utilized.

- Configuration Loading : In the Annex you will find a file named 'WebGoat.context' this is a file where we define addresses that we want to scan along with credentials that we can use in the application in order to be able to test for functionality beyond the log in screen.
- Web Crawling : As the first step it starts with exploring the targeted website with an automated indexing called 'Spiders' which are starting from a single point and start following links that are found in the HTML code. This is done recursively until either time is up or spiders have reached maximum depth meaning how many links the have processed after the initial one and or they cannot find any new URL to go to.
- Penetration Testing : Since ZAP now has a map of the website it starts to launch a simulated attack by sending API requests toward the targeted application and collecting any relative data.
- Report Generation : Of course all previous would go to waste if there was no report for us to extract , read and base our design and code fixes upon. Also ZAP assigns a risk

level for each entry of the report so we can effectively target any critical vulnerabilities first.

7.4.2 DAST Pipeline Stages

Following the same principles as for the SAST pipeline we create the DAST pipeline which consists of the following stages.



	Pipeline Info	Clean Up	SCM chekout	Creating Docker virtual network	Setting up WebGoat docker container	Creating user via cURL	Scanning target on ZAP container	Archive Reports to Workspace	Declarative: Post Actions
Average stage times: (Average full run time: ~6min 7s)	186ms	112ms	1s	503ms	17s	797ms	4min 5s	528ms	1s
#150 Jun 13 17:10 1 commit	184ms	96ms	1s	482ms	20s	887ms	7min 51s	287ms	2s
#149 Jun 13 16:52 2 commits	219ms	112ms	1s	586ms	19s	904ms	3min 26s	1s	2s

Image 7.10 : Jenkins : DAST pipeline

First of all we set 3 parameters ‘SCAN_TYPE' , 'NETWORK' and 'GENERATE_REPORT'.

- ‘SCAN_TYPE' is a drop down with two values that can be selected: 'Baseline' and 'Full'
- 'NETWORK' is an input with a default value of 'zap-communication-network'
- 'GENERATE_REPORT' is a boolean with default value true

```
parameters {
        choice  choices: ['Baseline', 'Full'],
                 description: 'Type of scan that is going to perform inside
the container',
                 name: 'SCAN_TYPE'

        string defaultValue: 'zap-communication-network',
                 description: "ZAP container network",
                 name: 'NETWORK'

        booleanParam defaultValue: true,
                 description: 'Parameter to know if wanna generate
report.',
                 name: 'GENERATE_REPORT'
    }
```

Then we are printing in the logs the parameters that the user selected and starting by cleaning the workspace and checking out the refenced repository in order to get the context file for ZAP.

```
stages {
        stage('Pipeline Info'){
            steps {
                script {
                    echo '<--Parameter Initialization-->'
                    echo """
                         The current parameters are:
                             Scan Type: ${params.SCAN_TYPE}
                             Scan Type: ${params.NETWORK}
                             Generate report: ${params.GENERATE_REPORT}
                         """
                }
            }
        }

        stage("Clean Up"){
            steps {
                cleanWs()
            }
        }

        stage("SCM chekout"){
            steps {
                checkout scm
            }
        }
```

This is the part that I would advise against when you are setting this environment , in a previous part I mentioned that Jenkins should be set in a Master / Slave relationship because when Jenkins runs in a Docker and you want to spin up a container for a Jenkins pipeline this will create a Docker in Docker (DinD) . It is highly not recommended due to instability, not being secure and because it is perplexing when it scales since we will have to track layers of internal and external ports. The way I found success in creating a container from my Jenkins service doing that to do it is to grant access to my host VM. This can be done by executing the following commands in the CLI.

```
sudo chmod 777 /var/run/docker.sock
sudo docker exec -it -u root jenkins /bin/bash
curl -fsSL https://get.docker.com | sh
usermod -aG docker jenkins
```

First we are granting read, write, execute access to the Docker socket file. Log in into the Jenkins container with the root user, downloading Docker and assigning to the jenkins user that runs by default in the Jenkins service the group docker. Now that I have explained that, let's proceed with the pipeline steps. We are creating a Docker virtual network in case it doesn't exist. This allows us to be able to create connections when we are setting our containers. Then we are downloading and starting the WebGoat container and run the custom waitForContainer method which is pinging the specified port on the container in order for us to get a pseudo ready status that the application is ready in order to start running the tests against it.

```
       stage('Creating Docker virtual network') {
             steps {
             script {
                   echo '<-- Docker virtual network-->'
                   def networkExists = sh(returnStatus: true, script:
'docker network inspect zap-communication-network')
                   if (networkExists != 0) {
                         echo 'Network does not exist, create it'
                         sh 'docker network create
zap-communication-network'
                    } else {
                          echo 'Network zap-communication-network already
exists.'
                    }
             }
             }
      }

      stage('Setting up WebGoat docker container') {
             steps {
             echo 'Pulling up last WebGoat container --> Start'
             sh 'docker pull webgoat/webgoat:latest'
             echo 'Pulling up last VMS container --> End'
             echo 'Starting container --> Start'
             sh """
                    docker run --detach \\
                    --network ${params.NETWORK} \\
                    -p 8383:8080 -p 9090:9090 \\
                    --name webgoat \\
                    webgoat/webgoat \\
             """
             // We hit internal ports here to verify that container is
ready along with the application
             waitForContainer("webgoat",9090)
             waitForContainer("webgoat",8080)

             }
      }
```

Then we proceed with creating a user by using the API of the application.

```
       stage('Creating user via cURL') {
            steps {
                sh """
                    curl 'http://192.168.1.164:8383/WebGoat/register.mvc'
-X POST --data-raw
'username=testuser&password=testuser&matchingPassword=testuser&agree=agree'
                """
            }
        }
```

Now it is the time to start our ZAP container and start the scan , in case anything goes wrong and produces an error I have set the catchError method in order to notify the user that there was an error produced during the scan. First I am printing into the logs the codes that the ZAP produces after it has finished. Then I get and print the paths that our data exist and will be generated by string manipulation, this is done in order to bind that data into the ZAP container that will start. Based on the SCAN_TYPE we are starting the container with a different preset that ZAP provides more info can be found in the following pages ZAP - Baseline Scan and ZAP - Full Scan. The base options that I set are:

- -t ${target} : Specify the start point of the application which is the log in screen

- -r report.html : Create a HTML report with name report
- -w report.md : Create a Markdown report with name report
- -n WebGoat.context : Use the WebGoat.context which we have loaded on the /zap/wrk directory
- -U testuser : From the context use the user testuser
- -I : Do not return failure on warning
- -j : Use Ajax spider
- -d : Show debug messages in order for the user to know what is going on
- -z "-config api.addrs.addr.name=.* -config api.addrs.addr.regex=true -config api.disablekey=true" : Based on official documentation this is not needed but as with other default values I prefer to set them in order to be visible to the end user.

The difference in options that I set for Full scan is:

- -m 10 : the number of minutes to web crawl for.
- -T 60 : setting the max time for ZAP to start and the scan to run

```
 stage('Scanning target on ZAP container') {
            steps {
                catchError(buildResult: 'UNSTABLE', stageResult:
'UNSTABLE') {
                    script {

                        echo """
                        ZAP can exit with :
                        0: Success
                        1: At least 1 FAIL
                        2: At least one WARN and no FAILs
                        3: Any other failure
                        Values 1 to 3 will mark the build as UNSTABLE
                        """

                        def jenkinsPath = "${WORKSPACE}/jenkins/zap"
                        def hostPath =
jenkinsPath.replace('/var/jenkins_home', '/jenkins_home')

                        echo "<-- ${jenkinsPath} is the path corresponding
to Jenkins -->"
                        echo "<-- ${hostPath} is the translated path
corresponding to Host machine -->"

                        echo '<-- The following will be mounted to zap/wrk
-->'
                        sh "ls -lR ${jenkinsPath}"

                        scan_type = "${params.SCAN_TYPE}"
                        echo "----> scan_type: $scan_type"
                        target = "http://192.168.1.164:8383/WebGoat/login"
                        if (scan_type == 'Baseline') {
                        sh """
                            docker run \\
                                --rm \\
                                --name zap-${BUILD_ID} \\
                                -p 8484:8080 \\
                                -p 8443:8443 \\
                                -v "${hostPath}:/zap/wrk/:rw" \\
                                --network ${params.NETWORK} \\
                                zaproxy/zap-stable \\
                                zap-baseline.py \\
                                    -t ${target} \\
                                    -r report.html \\
                                    -w report.md \\
                                    -n WebGoat.context \\
                                    -U testuser \\
                                    -I -j -d \\
                                    -z "-config api.addrs.addr.name=.*
-config api.addrs.addr.regex=true -config api.disablekey=true"
                        """

                        }
                        else if (scan_type == 'Full') {
                            sh """
                                docker run \\
                                --rm \\
                                --name zap-${BUILD_ID} \\
                                -p 8484:8080 \\
                                -p 8443:8443 \\
                                -v "${hostPath}:/zap/wrk/:rw" \\
                                --network ${params.NETWORK} \\
                                zaproxy/zap-stable \\
                                zap-full-scan.py \\
                                    -t ${target} \\
                                    -r report.html \\
                                    -w report.md \\
                                    -n WebGoat.context \\
                                    -U testuser \\
                                    -m 10 \\
                                    -T 60 \\
                                    -I -j -d \\
                                    -z "-config api.addrs.addr.name=.*
-config api.addrs.addr.regex=true -config api.disablekey=true"
                            """
                        }
                        else {
                            echo 'Something went wrong...'
                        }

                    }
                }
                }
        }
```

Once ZAP has finished based on the 'GENERATE_REPORT' value we are putting everything that ZAP has created along with the files that we mount into a zip and archive it. Then we are proceeding with stopping the webgoat container and removing it. We do not need to handle the ZAP container as we use docker run along with the --rm option which means start the container, execute, get a result and then remove the container.

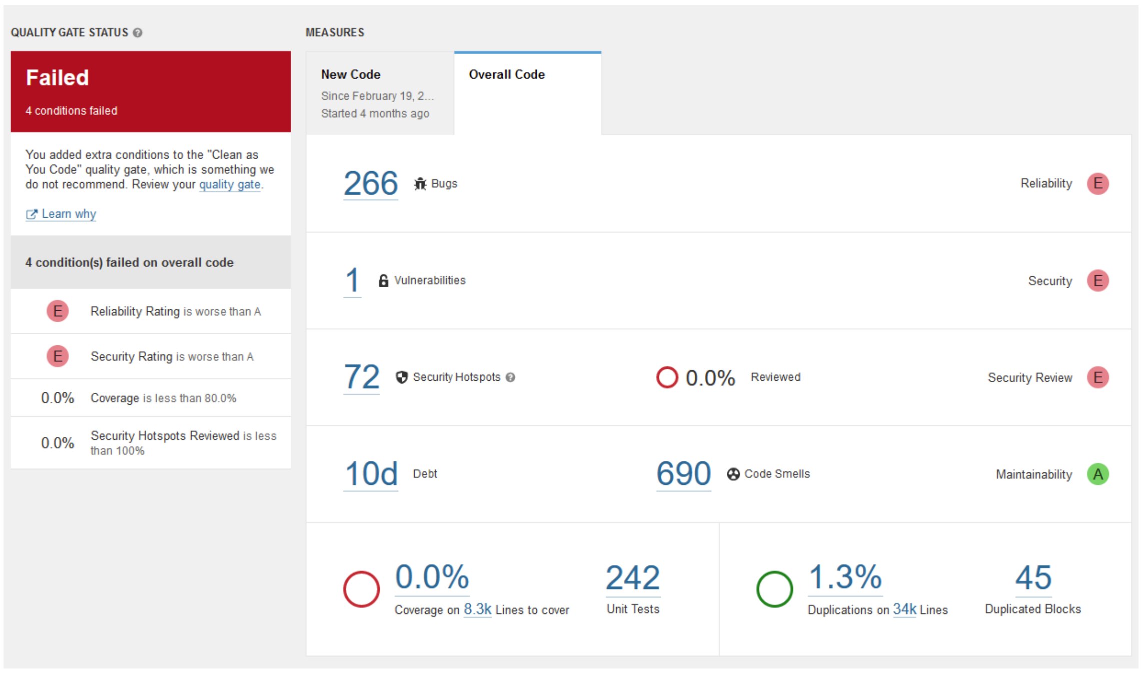
```
stage('Archive Reports to Workspace') {
            when {
                environment name : 'GENERATE_REPORT', value: 'true'
            }
            steps {
              catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {
                script {
                    zip zipFile: "ReportsAndContext-${BUILD_ID}.zip", 
archive: true, dir : "${WORKSPACE}/jenkins/zap/"
                }
              }

            }
        }
    }

    post {
        always {
            echo 'Removing containers'
            sh """
                    docker stop webgoat
                    docker rm webgoat
                """
        }
    }
```

8. Result Analysis

Now that our pipelines have been executed it is time to see the results.

8.1 SAST Result Analysis

By heading into SonarQube we can see the latest report of the project. On the top left side it informs us that it has failed the Quality Gate that we have specified . Then we can see a rating as well to inform us what the project's state is. While in the main view we can see that SonarQube has identified 266 potential Bugs, 1 Vulnerability and 72 Security hotspots. At the end it gives an estimate on how much time it would take to fix those issues along with detected Code Smells , Unit tests number and coverage and code duplication stats.



QUALITY GATE STATUS

Failed
4 conditions failed

You added extra conditions to the "Clean as You Code" quality gate, which is something we do not recommend. Review your quality gate.

Learn why

4 condition(s) failed on overall code

E	Reliability Rating is worse than A
E	Security Rating is worse than A
0.0%	Coverage is less than 80.0%
0.0%	Security Hotspots Reviewed is less than 100%

MEASURES

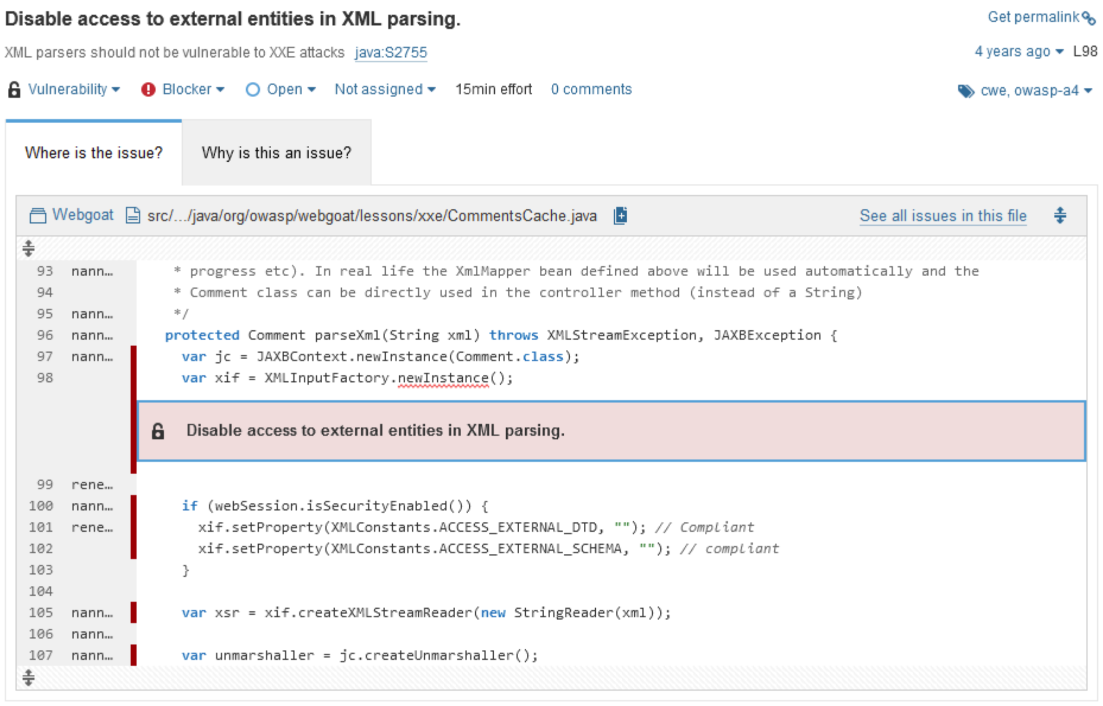
New Code — Since February 19, 2… Started 4 months ago | Overall Code

266 Bugs		Reliability E
1 Vulnerabilities		Security E
72 Security Hotspots	0.0% Reviewed	Security Review E
10d Debt	690 Code Smells	Maintainability A

0.0% Coverage on 8.3k Lines to cover	242 Unit Tests	1.3% Duplications on 34k Lines	45 Duplicated Blocks

Image 8.1 : SonarQube : Project View

For example when we select to see the Vulnerability we see the following.



Disable access to external entities in XML parsing.

XML parsers should not be vulnerable to XXE attacks java:S2755

Get permalink

4 years ago L98

Vulnerability | Blocker | Open | Not assigned | 15min effort | 0 comments | cwe, owasp-a4

Where is the issue? | Why is this an issue?

Webgoat src/.../java/org/owasp/webgoat/lessons/xxe/CommentsCache.java — See all issues in this file

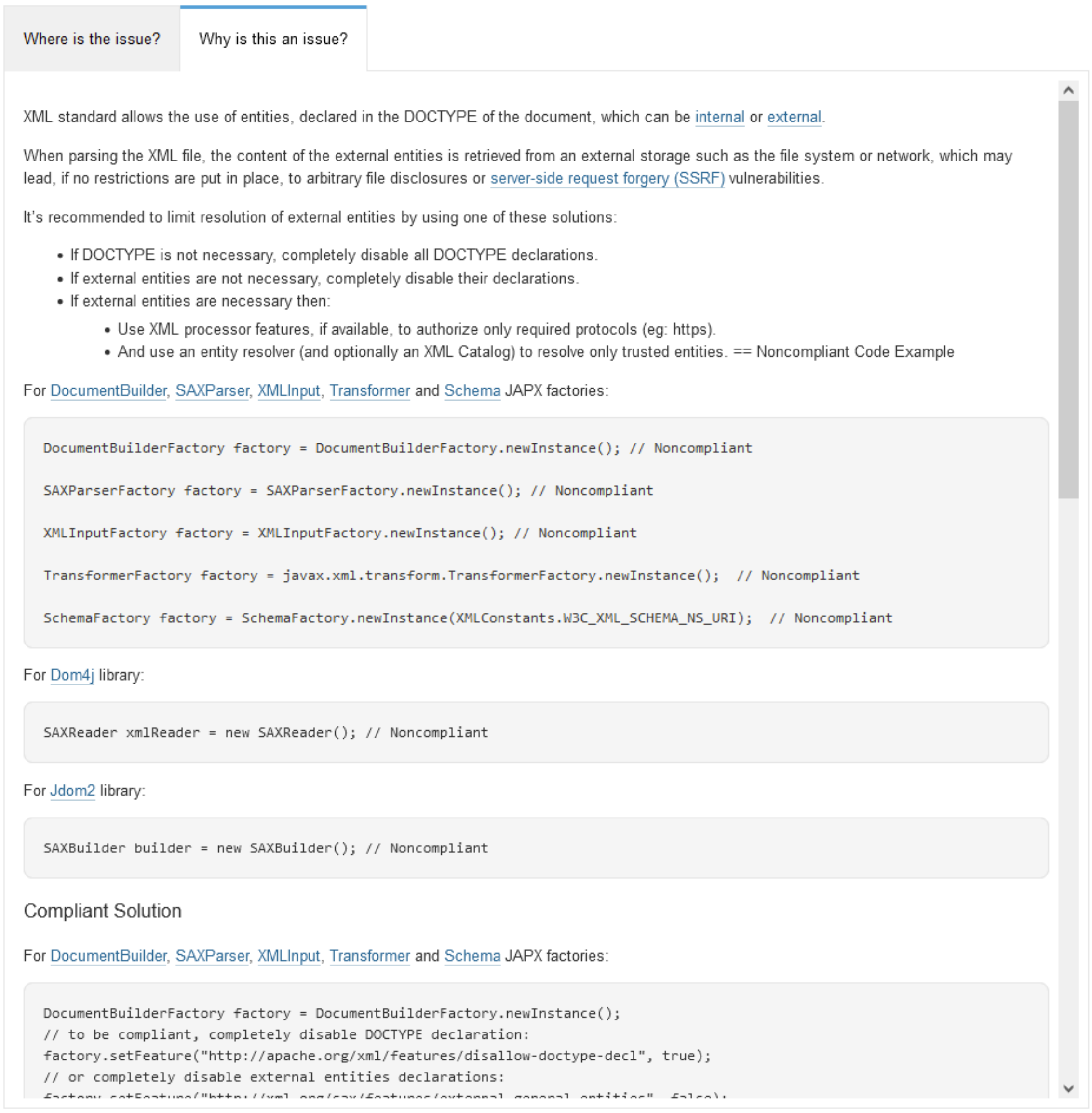
```
     * progress etc). In real life the XmlMapper bean defined above will be used automatically and the
     * Comment class can be directly used in the controller method (instead of a String)
     */
    protected Comment parseXml(String xml) throws XMLStreamException, JAXBException {
      var jc = JAXBContext.newInstance(Comment.class);
      var xif = XMLInputFactory.newInstance();
```

Disable access to external entities in XML parsing.

```
      if (webSession.isSecurityEnabled()) {
        xif.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, ""); // Compliant
        xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, ""); // compliant
      }

      var xsr = xif.createXMLStreamReader(new StringReader(xml));

      var unmarshaller = jc.createUnmarshaller();
```

Image 8.2 : SonarQube : Vulnerability Where is the Issue

SonarQube shows us in the code where the vulnerability is and also has a tab with recommendations on how to fix and why is this an issue, informing and educating the development team.



XML standard allows the use of entities, declared in the DOCTYPE of the document, which can be internal or external.

When parsing the XML file, the content of the external entities is retrieved from an external storage such as the file system or network, which may lead, if no restrictions are put in place, to arbitrary file disclosures or server-side request forgery (SSRF) vulnerabilities.

It's recommended to limit resolution of external entities by using one of these solutions:

- If DOCTYPE is not necessary, completely disable all DOCTYPE declarations.
- If external entities are not necessary, completely disable their declarations.
- If external entities are necessary then:
  - Use XML processor features, if available, to authorize only required protocols (eg: https).
  - And use an entity resolver (and optionally an XML Catalog) to resolve only trusted entities. == Noncompliant Code Example

For DocumentBuilder, SAXParser, XMLInput, Transformer and Schema JAPX factories:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance(); // Noncompliant

SAXParserFactory factory = SAXParserFactory.newInstance(); // Noncompliant

XMLInputFactory factory = XMLInputFactory.newInstance(); // Noncompliant

TransformerFactory factory = javax.xml.transform.TransformerFactory.newInstance();  // Noncompliant

SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);  // Noncompliant
```

For Dom4j library:

```
SAXReader xmlReader = new SAXReader(); // Noncompliant
```

For Jdom2 library:

```
SAXBuilder builder = new SAXBuilder(); // Noncompliant
```

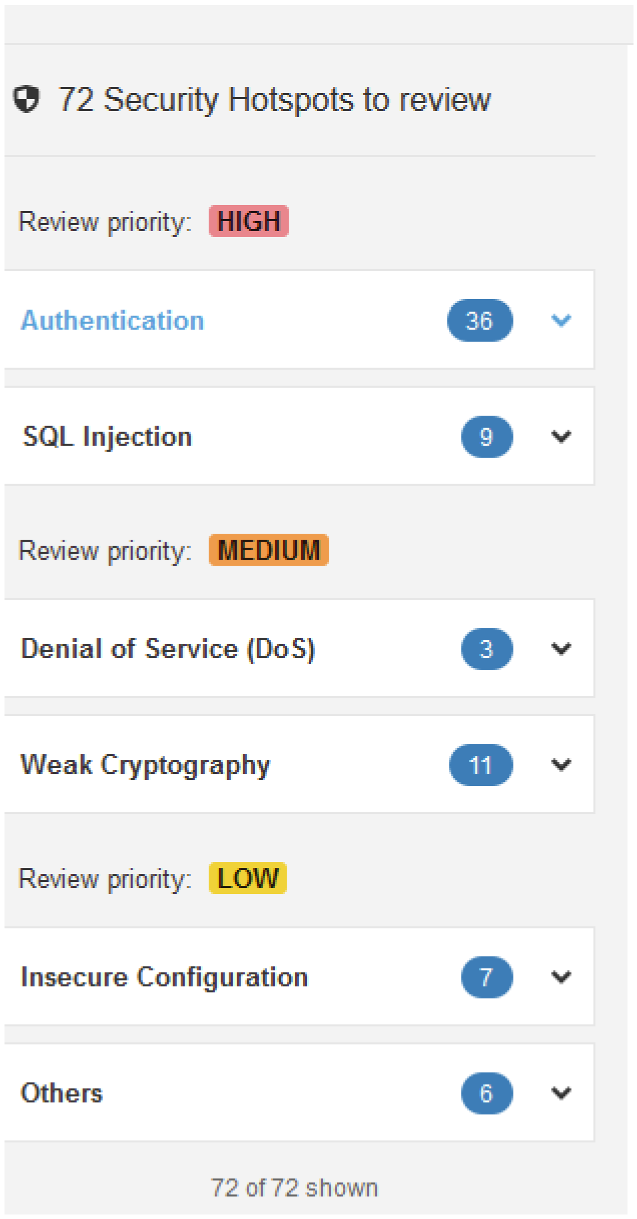
Compliant Solution

For DocumentBuilder, SAXParser, XMLInput, Transformer and Schema JAPX factories:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
// to be compliant, completely disable DOCTYPE declaration:
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
// or completely disable external entities declarations:
factory.setFeature("http://xml.org/sax/features/external-general-entities", false);
```

Image 8.3 : SonarQube : Vulnerability Why is this an Issue

Then if we select the Security Hotspots we can see that SonarQube has categorized them based on the priority from High to Low as well as grouping them based on the security vulnerability that they introduce into our application.


72 Security Hotspots to review

Review priority: HIGH

Authentication	36
SQL Injection	9

Review priority: MEDIUM

Denial of Service (DoS)	3
Weak Cryptography	11

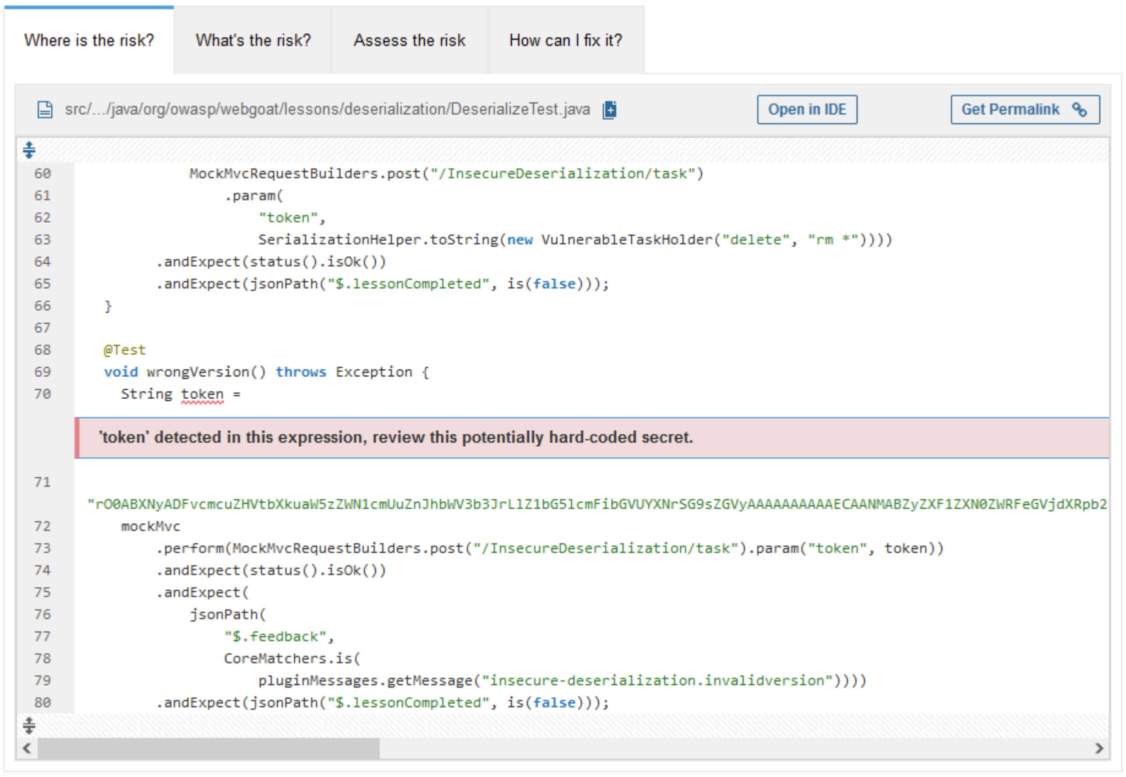
Review priority: LOW

Insecure Configuration	7
Others	6

72 of 72 shown

Image 8.4 : SonarQube : Security Hotspots

Following the same pattern, SonarQube shows us where this risk is , what is the risk , assess the risk and how we can fix it. In the following images there are a few examples from the High priority category. The first two show us that SonarQube executes secret scanning which means detecting tokens and passwords in the code. And the third show us that we have not followed the correct practices which can lead to an SQL injection.



Where is the risk? | What's the risk? | Assess the risk | How can I fix it?

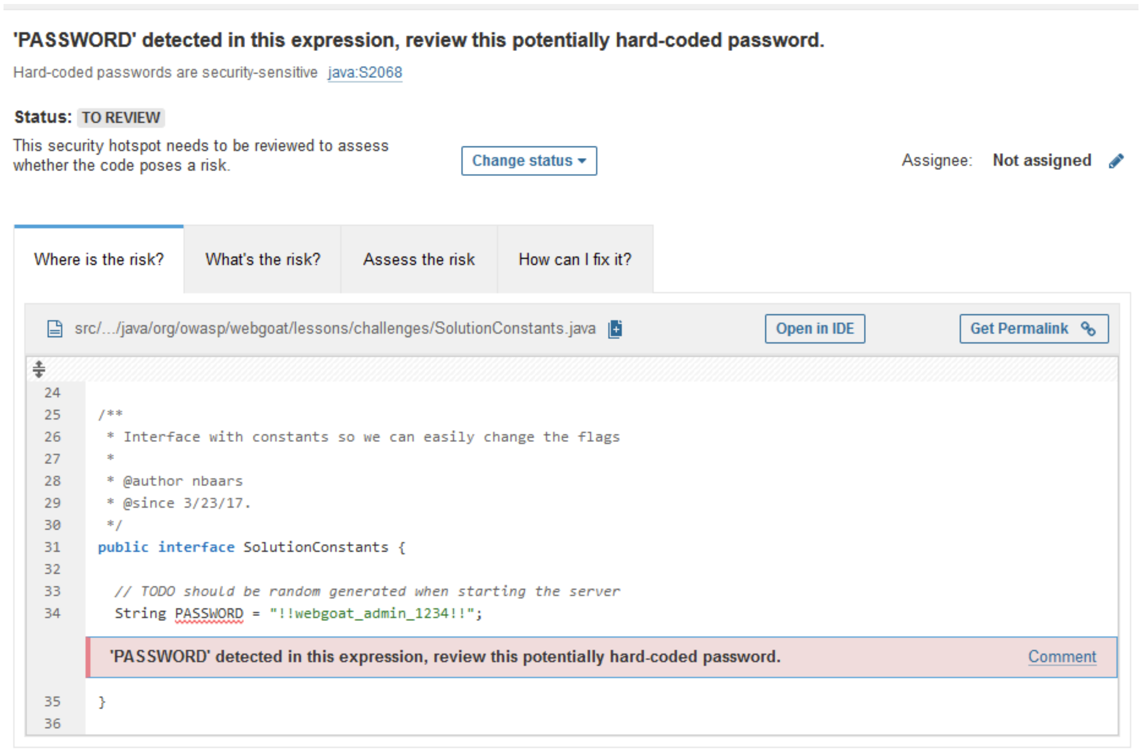
src/.../java/org/owasp/webgoat/lessons/deserialization/DeserializeTest.java

```
            MockMvcRequestBuilders.post("/InsecureDeserialization/task")
                .param(
                    "token",
                    SerializationHelper.toString(new VulnerableTaskHolder("delete", "rm *"))))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.lessonCompleted", is(false)));
  }

  @Test
  void wrongVersion() throws Exception {
    String token =
```

'token' detected in this expression, review this potentially hard-coded secret.

```
"rO0ABXNyADFvcmcuZHVtbXkuaW5zZWN1cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAAAAAAAAECAANMABZyZXF1ZXN0ZWRFeGVjdXRpb2
    mockMvc
        .perform(MockMvcRequestBuilders.post("/InsecureDeserialization/task").param("token", token))
        .andExpect(status().isOk())
        .andExpect(
            jsonPath(
                "$.feedback",
                CoreMatchers.is(
                    pluginMessages.getMessage("insecure-deserialization.invalidversion"))))
        .andExpect(jsonPath("$.lessonCompleted", is(false)));
```

Image 8.5 : SonarQube : Security Hotspot Authentication Token



'PASSWORD' detected in this expression, review this potentially hard-coded password.

Hard-coded passwords are security-sensitive java:S2068

Status: TO REVIEW

This security hotspot needs to be reviewed to assess whether the code poses a risk.

Change status ▾

Assignee: **Not assigned**

Where is the risk? | What's the risk? | Assess the risk | How can I fix it?

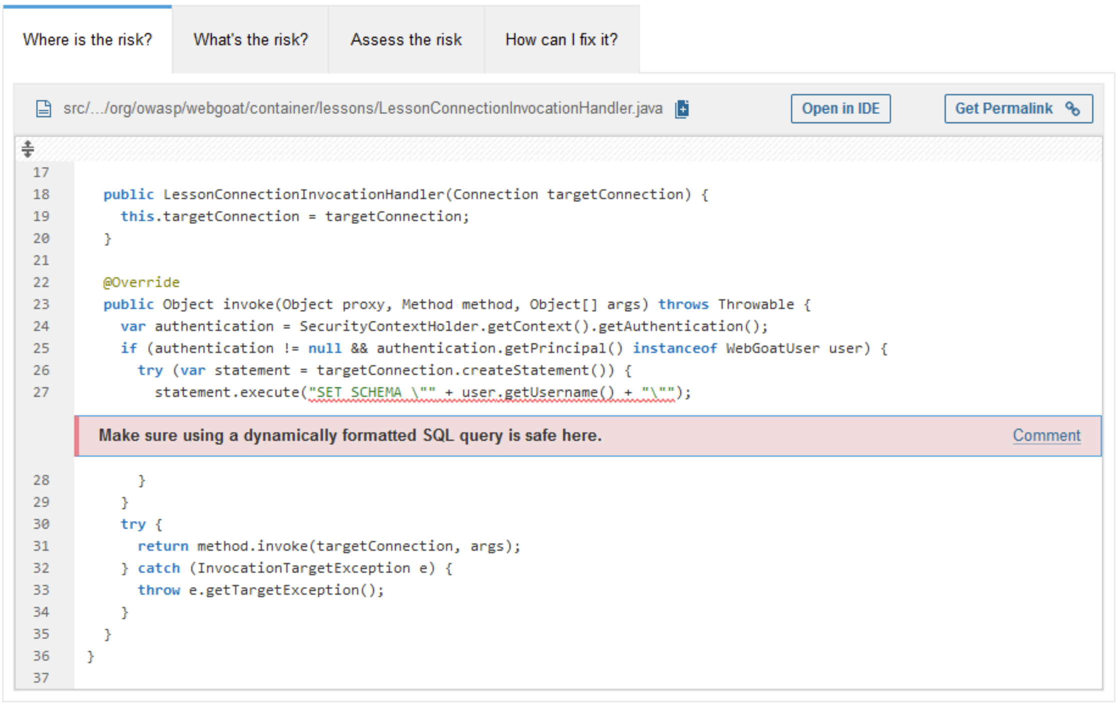
src/.../java/org/owasp/webgoat/lessons/challenges/SolutionConstants.java

```

/**
 * Interface with constants so we can easily change the flags
 *
 * @author nbaars
 * @since 3/23/17.
 */
public interface SolutionConstants {

  // TODO should be random generated when starting the server
  String PASSWORD = "!!webgoat_admin_1234!";
```

'PASSWORD' detected in this expression, review this potentially hard-coded password. Comment

```
}
```

Image 8.6 : SonarQube : Security Hotspot Authentication Password



```
  public LessonConnectionInvocationHandler(Connection targetConnection) {
    this.targetConnection = targetConnection;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    var authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && authentication.getPrincipal() instanceof WebGoatUser user) {
      try (var statement = targetConnection.createStatement()) {
        statement.execute("SET SCHEMA \"" + user.getUsername() + "\"");
```

Make sure using a dynamically formatted SQL query is safe here.

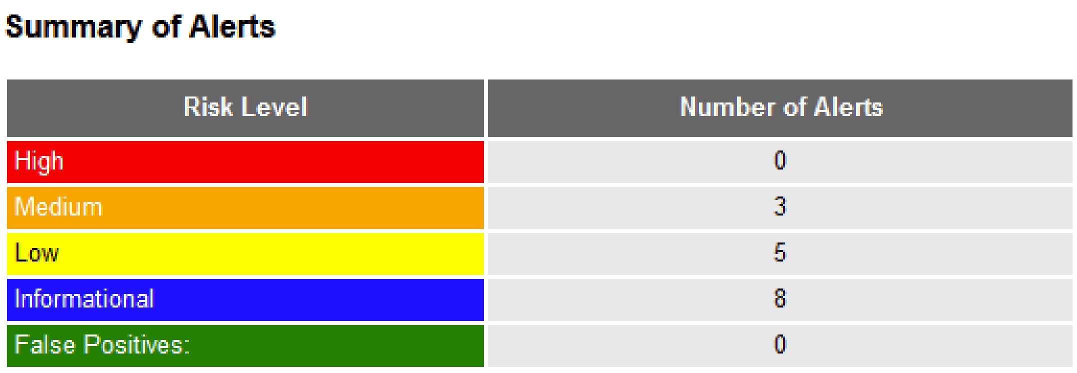
```
      }
    }
    try {
      return method.invoke(targetConnection, args);
    } catch (InvocationTargetException e) {
      throw e.getTargetException();
    }
  }
}
```

Image 8.7 : SonarQube : Security Hotspot SQL Injection

As for all issues that SonarQube finds it enables the development team to handle them which means mark them as a non issue and create comments for the finding in order to assess what is the best fix.

8.2 DAST Result Analysis

First I am going to show parts from the Baseline scan and then parts from the Full scan. As with SonarQube Zap categorizes each alert found with a risk level

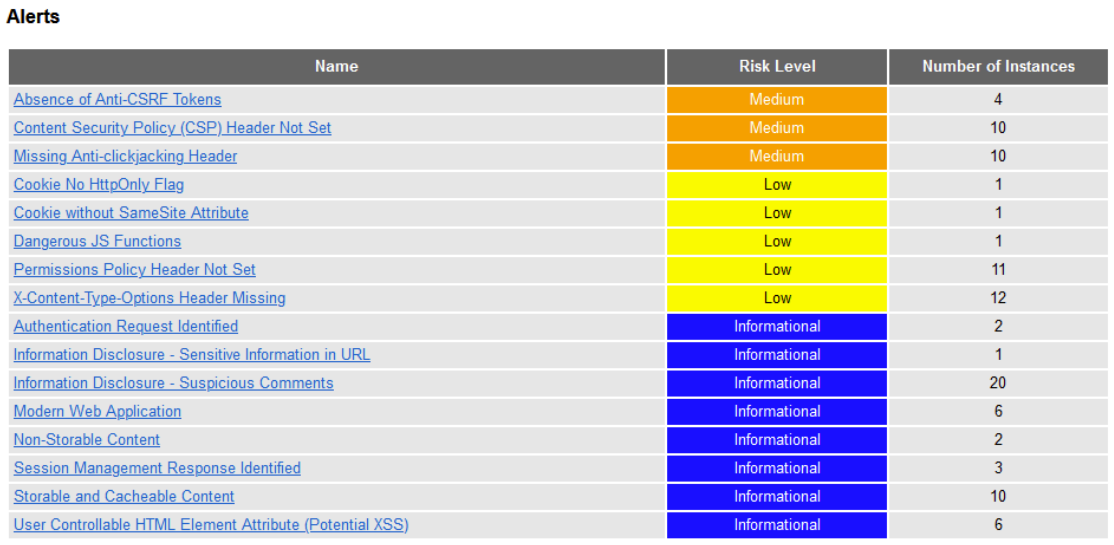


Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	5
Informational	8
False Positives:	0

Image 8.8 : ZAP : Baseline Report Summary Alert

Then shows us for each alert how many instances were found.

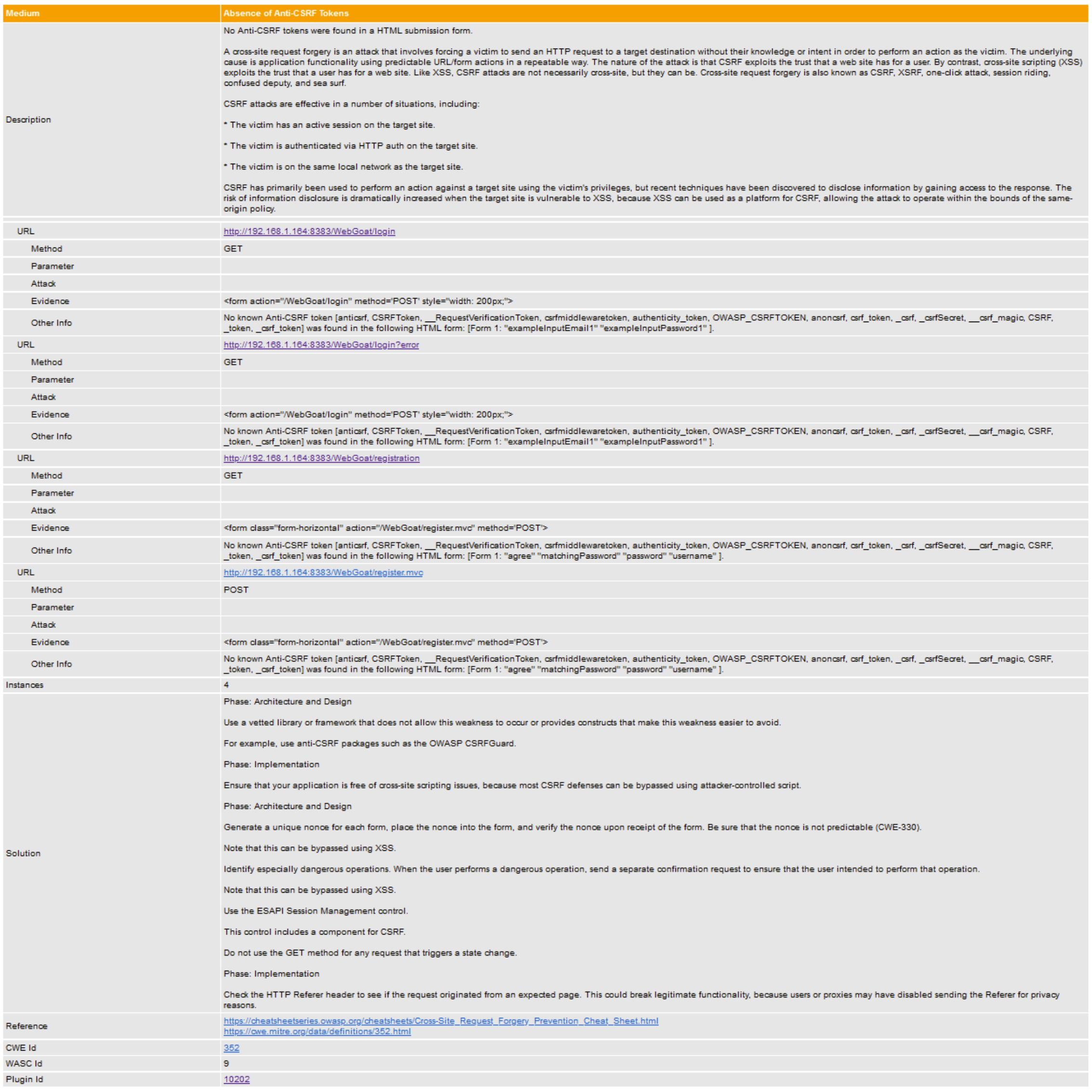


Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	4
Content Security Policy (CSP) Header Not Set	Medium	10
Missing Anti-clickjacking Header	Medium	10
Cookie No HttpOnly Flag	Low	1
Cookie without SameSite Attribute	Low	1
Dangerous JS Functions	Low	1
Permissions Policy Header Not Set	Low	11
X-Content-Type-Options Header Missing	Low	12
Authentication Request Identified	Informational	2
Information Disclosure - Sensitive Information in URL	Informational	1
Information Disclosure - Suspicious Comments	Informational	20
Modern Web Application	Informational	6
Non-Storable Content	Informational	2
Session Management Response Identified	Informational	3
Storable and Cacheable Content	Informational	10
User Controllable HTML Element Attribute (Potential XSS)	Informational	6

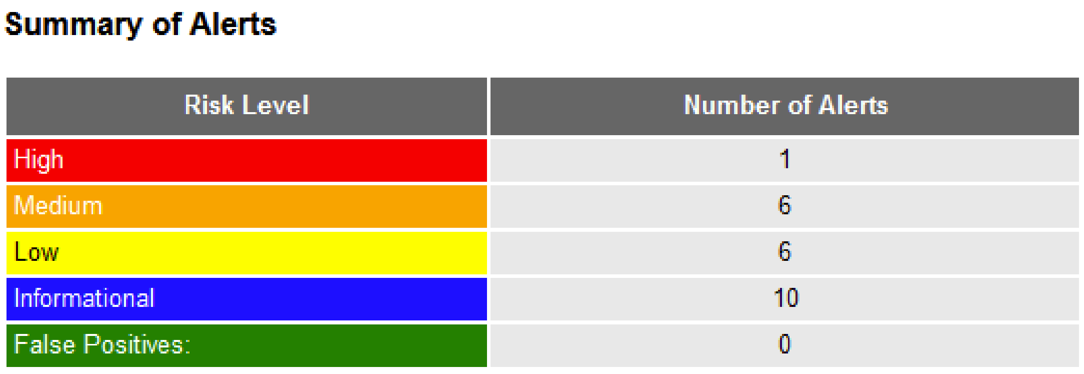
Image 8.9 : ZAP : Baseline Report Alerts

Then if we select an alert it guides us to that alert section which informs us with a description , where and how it was found as well as how to fix it with some references that we can read.



Medium	Absence of Anti-CSRF Tokens
Description	No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf. CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.
URL	http://192.168.1.164:8383/WebGoat/login
Method	GET
Parameter	
Attack	
Evidence	<form action="/WebGoat/login" method='POST' style="width: 200px;">
Other Info	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].
URL	http://192.168.1.164:8383/WebGoat/login?error
Method	GET
Parameter	
Attack	
Evidence	<form action="/WebGoat/login" method='POST' style="width: 200px;">
Other Info	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].
URL	http://192.168.1.164:8383/WebGoat/registration
Method	GET
Parameter	
Attack	
Evidence	<form class="form-horizontal" action="/WebGoat/register.mvc" method='POST'>
Other Info	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "agree" "matchingPassword" "password" "username"].
URL	http://192.168.1.164:8383/WebGoat/register.mvc
Method	POST
Parameter	
Attack	
Evidence	<form class="form-horizontal" action="/WebGoat/register.mvc" method='POST'>
Other Info	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "agree" "matchingPassword" "password" "username"].
Instances	4
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html https://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Plugin Id	10202

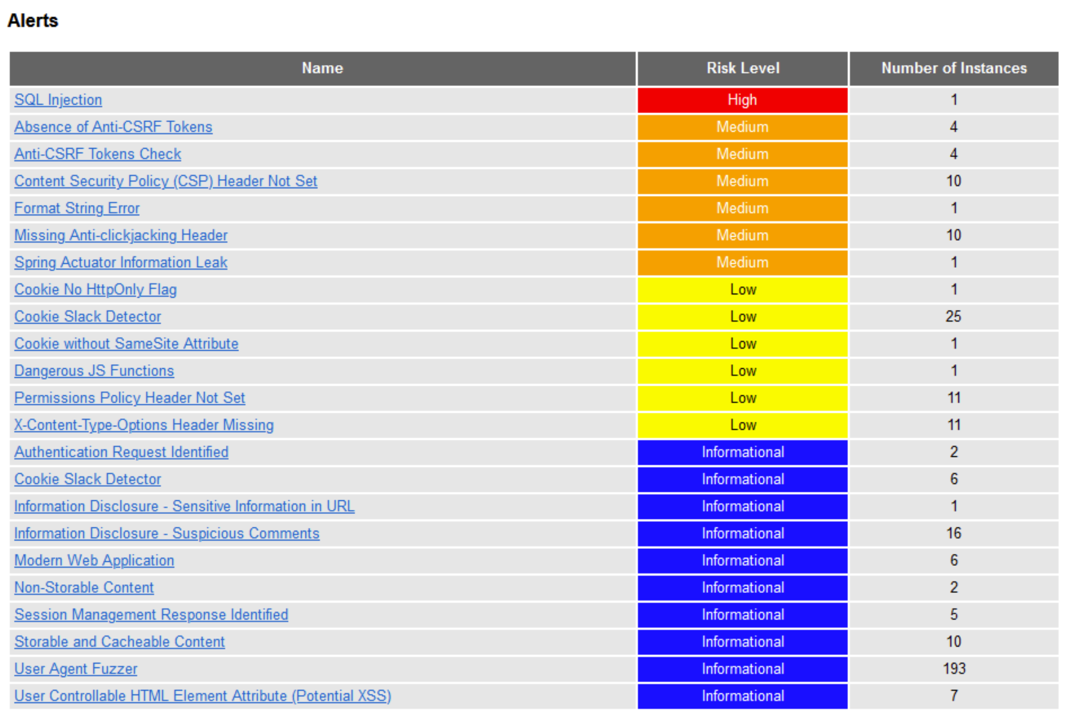
Image 8.10 : ZAP : Baseline Alert Report



Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	6
Low	6
Informational	10
False Positives:	0

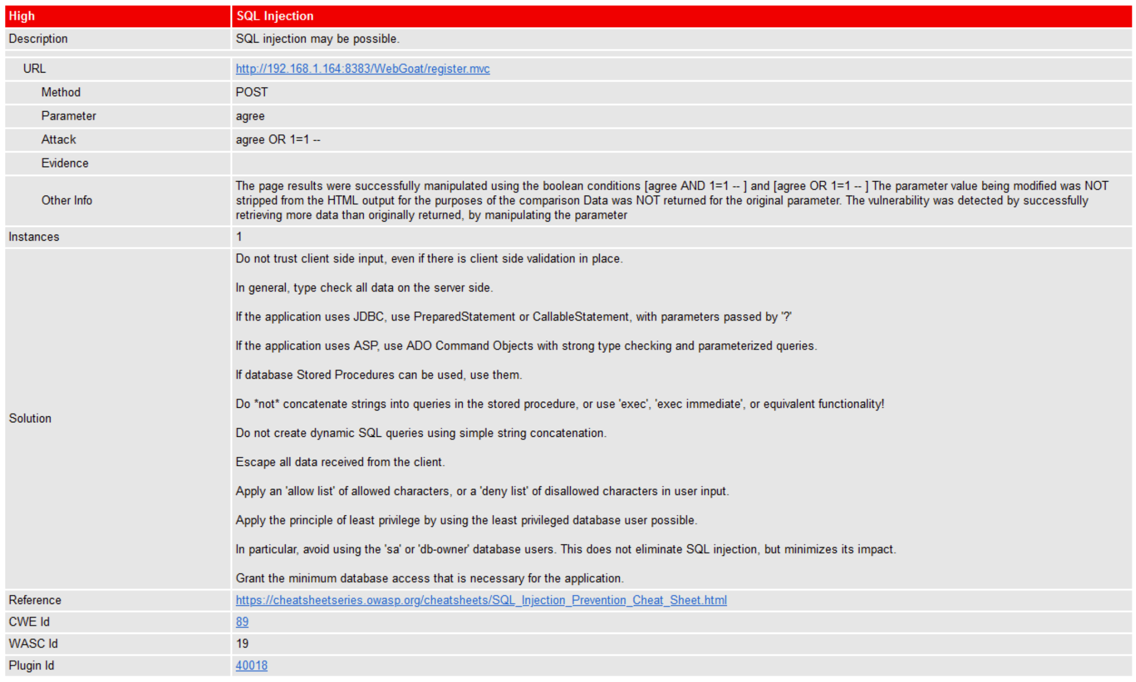
Image 8.11 : ZAP : Full scan Report Summary Alert



Alerts

Name	Risk Level	Number of Instances
SQL Injection	High	1
Absence of Anti-CSRF Tokens	Medium	4
Anti-CSRF Tokens Check	Medium	4
Content Security Policy (CSP) Header Not Set	Medium	10
Format String Error	Medium	1
Missing Anti-clickjacking Header	Medium	10
Spring Actuator Information Leak	Medium	1
Cookie No HttpOnly Flag	Low	1
Cookie Slack Detector	Low	25
Cookie without SameSite Attribute	Low	1
Dangerous JS Functions	Low	1
Permissions Policy Header Not Set	Low	11
X-Content-Type-Options Header Missing	Low	11
Authentication Request Identified	Informational	2
Cookie Slack Detector	Informational	6
Information Disclosure - Sensitive Information in URL	Informational	1
Information Disclosure - Suspicious Comments	Informational	16
Modern Web Application	Informational	6
Non-Storable Content	Informational	2
Session Management Response Identified	Informational	5
Storable and Cacheable Content	Informational	10
User Agent Fuzzer	Informational	193
User Controllable HTML Element Attribute (Potential XSS)	Informational	7

Image 8.12 : ZAP : Full scan Report Alerts



High	SQL Injection
Description	SQL injection may be possible.
URL	http://192.168.1.164:8383/WebGoat/register.mvc
Method	POST
Parameter	agree
Attack	agree OR 1=1 --
Evidence	
Other Info	The page results were successfully manipulated using the boolean conditions [agree AND 1=1 --] and [agree OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Instances	1
Solution	Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side. If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?' If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries. If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality! Do not create dynamic SQL queries using simple string concatenation. Escape all data received from the client. Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input. Apply the principle of least privilege by using the least privileged database user possible. In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact. Grant the minimum database access that is necessary for the application.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
CWE Id	89
WASC Id	19
Plugin Id	40018

Image 8.13 : ZAP : Full scan Alert Report

Conclusions

Integrating Security Scans into CI/CD Pipelines is crucial in order for us to be able to make secure applications. This should be the new standard having just a working application is not enough it has to be secure as well. Following the guidelines we have set up a complete CI/CD environment all the way from the VM level down to the Automation Server. Then we connected all the services between them enabling them to work flawlessly. Then we created two pipelines, one that does the SAST which blocks the release if the criteria are not met and a DAST that provides results from just a container and both of them provide a lot of crucial results. The targeted application is an intentionally insecure application but we have to keep in mind that this could also be the work of a new developer. I hope this serves as a guide for anyone that wants to get into DevSecOps as there are not a lot of resources that can help you with real life examples and how to set everything up. Future research could explore more SAST and DAST tools but also the CD part.

Terminology table

English Terminology	Ελληνική Ορολογία
Pipeline	Διεργασία
Waterfall model	Μοντέλο καταρράκτη
Agile	Ευέλικτο μοντέλο
Brute force attack	Επίθεση με ωμή δύναμη
Securing the perimeter	Ασφάλιση της περιμέτρου
Defense in Depth	Άμυνα σε βάθος
Zero Trust	Μηδενική εμπιστοσύνη
Zero-Day-Exploit	Εκμετάλλευση μηδενικής ημέρας
Black box testing	Λειτουργικός έλεγχος χωρίς γνώση κώδικα

Table of abbreviations-acronyms-initialisms

CI/CD	Continuous Integration and Continuous Delivery/Deployment
SDLC	software development lifecycle
SAST	Static Application Security Testing
DAST	Dynamic Application Security Testing
DevOps	Develop and Operations
DevSecOps	Develop Security and Operations
ACL	Access control Lists
SaaS	Software as a service
PMC	Apache Struts Project Management Committee
CVE	Common Vulnerabilities and Exposures
KVM	Kernel-based Virtual Machine
LXC	Linux Containers
OS	Operating System
VM	Virtual Machine
LTS	Long Term Support
API	Application Programming Interface
CLI	command-line interface
VCS	version control system
UI	User Interface
ISO	International Organization for Standardization
SSH	Secure Shell
GPG	GNU Privacy Guard key
GNU	GNU’s not Unix
ZAP	Zed Attack Proxy
DinD	Docker in Docker

Bibliography and hyperlinks

Books and Monographs:

Royce, W. W. (2021). Managing the development of large software systems (1970).

Hyperlinks:

Manifesto for Agile Software Development (http://agilemanifesto.org/)
Redhat : What is CI/CD? (https://www.redhat.com/en/topics/devops/what-is-ci-cd)
Microsoft : What is DevOps? (https://learn.microsoft.com/en-us/devops/what-is-devops)
Microsoft: What is DevSecOps? (https://www.microsoft.com/en-us/security/business/security-101/what-is-devsecops)
The Rugged Manifesto (https://ruggedsoftware.org/)
IBM : What is cybersecurity? (https://www.ibm.com/topics/cybersecurity)
IBM : What is a cyberattack? (https://www.ibm.com/topics/cyber-attack)
IBM : What is SaaS (https://www.ibm.com/topics/saas)
The National Institute of Standards and Technology (USA) : defense-in-depth (https://csrc.nist.gov/glossary/term/defense_in_depth)
IBM : What is zero trust? (https://www.ibm.com/topics/zero-trust)
GAO-18-559, DATA PROTECTION: Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach (https://www.warren.senate.gov/imo/media/doc/2018.09.06%20GAO%20Equifax%20report.pdf)
Apache Struts Statement on Equifax Security Breach (https://news.apache.org/foundation/entry/apache-struts-statement-on-equifax)
CVE-2017-9805 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9805)
MEDIA ALERT: The Apache Software Foundation Confirms Equifax Data Breach Due to Failure to Install Patches Provided for Apache® Struts™ Exploit (https://news.apache.org/foundation/entry/media-alert-the-apache-software)
CVE-2017-5638 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638)
Jenkins Security Advisory 2024-01-24 (https://www.jenkins.io/security/advisory/2024-01-24/)
Proxmox (https://www.proxmox.com/en)
Features - Proxmox Virtual Environment (https://www.proxmox.com/en/proxmox-virtual-environment/features)
Oracle VM VirtualBox (https://www.virtualbox.org/)
Oracle (https://www.oracle.com/)
Guest_OSes – Oracle VM VirtualBox (https://www.virtualbox.org/wiki/Guest_OSes)
Ubuntu 22.04.4 LTS (https://www.releases.ubuntu.com/22.04/)
Docker : What is a container (https://www.docker.com/resources/what-container)
Docker (https://www.docker.com)
dockerd | Docker Docs (https://docs.docker.com/reference/cli/dockerd)
docker | Docker Docs (http://docs.docker.com/reference/cli/docker/)
Docker Engine API (https://docs.docker.com/engine/api/)
Docker : Docker Engine overview (https://docs.docker.com/engine/)
Docker CLI (https://www.docker.com/products/cli/)
Docker Desktop (https://www.docker.com/products/docker-desktop/)
Docker Hub (https://www.docker.com/products/docker-hub/)
Atlassian: What is version control? (https://www.atlassian.com/git/tutorials/what-is-version-control)
Gitlab : Why use version control (https://about.gitlab.com/topics/version-control/)
Git (https://git-scm.com/)
Apache Subversion (https://subversion.apache.org/)
Mercurial (https://www.mercurial-scm.org/)
GitLab (https://about.gitlab.com/)
GitLab CE (https://hub.docker.com/r/gitlab/gitlab-ce)
GitLab Customers (https://about.gitlab.com/customers/)

Jenkins (https://www.jenkins.io/)
jenkins/jenkins - Docker Image (https://hub.docker.com/r/jenkins/jenkins)
Jenkins Security (https://www.jenkins.io/security/)
IBM (https://www.ibm.com/us-en)
Amazon AWS (https://aws.amazon.com/)
Sonatype Nexus Repository (https://help.sonatype.com/en/sonatype-nexus-repository.html)
Download VirtualBox (https://www.virtualbox.org/wiki/Downloads)
OpenSSH (https://www.openssh.com/)
Visual Studio Code (https://code.visualstudio.com/)
Ubuntu : Package management with APT (https://help.ubuntu.com/community/AptGet/Howto)
Install Docker Engine on Ubuntu (https://docs.docker.com/engine/install/ubuntu)
Uninstall old versions (https://docs.docker.com/engine/install/ubuntu/#uninstall-old-versions)
GitLab installation guide (https://docs.gitlab.com/ee/install/docker.html)
Package defaults | GitLab (https://docs.gitlab.com/ee/administration/package_information/defaults.html)
GitHub - jenkinsci/docker: Docker official jenkins repo (https://github.com/jenkinsci/docker)
Download and Compatibility - Sonatype Lifecycle (https://help.sonatype.com/en/download-and-compatibility.html#DownloadandCompatibility-Jenkins)
sonatype/nexus3 - Docker Image (https://hub.docker.com/r/sonatype/nexus3/)
Sonatype Help (https://help.sonatype.com/index.html?lang=en)
Installing from Docker | SonarQube Docs (https://docs.sonarsource.com/sonarqube/latest/setup-and-upgrade/install-the-server/installing-sonarqube-from-docker/)
Personal access tokens | GitLab (https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html)
Webhooks (https://docs.sonarsource.com/sonarqube/9.9/project-administration/webhooks/)
Pipeline Syntax (https://www.jenkins.io/doc/book/pipeline/syntax/)
Pipeline: Basic Steps (https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/)
GIthub : WebGoat (https://github.com/WebGoat/WebGoat)
ZAP (https://www.zaproxy.org/)
ZAP – Getting Started (https://www.zaproxy.org/getting-started/)
ZAP - Baseline Scan (https://www.zaproxy.org/docs/docker/baseline-scan/)
ZAP - Full Scan (https://www.zaproxy.org/docs/docker/full-scan).

Annex

Annex A : Docker-compose.yaml

```yaml
version: "3.8"

services:
  gitlab:
       image: gitlab/gitlab-ce:latest
       container_name: gitlab
       restart: always
       stop_grace_period: 5m
       ports:
       - "444:443"
       - "81:80"
       - "23:22"
       volumes:
       - gitlab_data_config:/etc/gitlab
       - gitlab_data_logs:/var/log/gitlab
       - gitlab_data_data:/var/opt/gitlab
       shm_size: 256m

  jenkins:
       image: jenkins/jenkins:lts-jdk17
       container_name: jenkins
       ports:
       - "8080:8080"
       - "50000:50000"
       restart: always
       stop_grace_period: 5m
       volumes:
       - jenkins_home:/var/jenkins_home
       - /var/run/docker.sock:/var/run/docker.sock
       networks:
       - jenkins_network

  nexus:
       image: sonatype/nexus3
       container_name: nexus
       restart: always
       stop_grace_period: 5m
       ports:
       - "8081:8081"
       volumes:
       - nexus_data:/nexus-data
  sonarqube:
       image: sonarqube:lts-community
       depends_on:
       - db
       environment:
       SONAR_JDBC_URL: jdbc:postgresql://db:5432/sonar
       SONAR_JDBC_USERNAME: sonar
       SONAR_JDBC_PASSWORD: sonar
       container_name: sonarqube
       restart: always
       stop_grace_period: 5m
       ports:
       - "9000:9000"
       volumes:
       - sonarqube_data:/opt/sonarqube/data
       - sonarqube_extensions:/opt/sonarqube/extensions
       - sonarqube_logs:/opt/sonarqube/logs
  db:
       image: postgres:12
       environment:
       POSTGRES_USER: sonar
       POSTGRES_PASSWORD: sonar
       volumes:
       - postgresql:/var/lib/postgresql
       - postgresql_data:/var/lib/postgresql/data

  # zap:
  #   image: zaproxy/zap-stable
  #   container_name: zap
  #   restart: always
  #   stop_grace_period: 1m
  #   ports:
  #    - "8282:8080"
  #    - "8443:8443"
  #   volumes:
  #    - zap_data:/zap/wrk/
  #    - shared_data:/shared
  #   command: >
  #    sh -c "zap-webswing.sh"
  #   networks:
  #    - zap-communication-network

volumes:
  jenkins_home:
       driver: local
       driver_opts:
       type: none
       o: bind
       device: /jenkins_home
```

```
nexus_data:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /nexus-data

sonarqube_data:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /sonarqube_data

sonarqube_extensions:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /sonarqube_extensions

sonarqube_logs:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /sonarqube_logs

gitlab_data_config:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /gitlab_data_config

gitlab_data_logs:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /gitlab_data_logs

gitlab_data_data:
     driver: local
     driver_opts:
     type: none
     o: bind
     device: /gitlab_data_data
```

```
  postgresql:
      driver: local
      driver_opts:
      type: none
      o: bind
      device: /postgresql

  postgresql_data:
      driver: local
      driver_opts:
      type: none
      o: bind
      device: /postgresql_data

  zap_data:
      driver: local
      driver_opts:
      type: none
      o: bind
      device: /zap_data

  # shared_data:
  #   driver: local
  #   driver_opts:
  #    type: none
  #    o: bind
  #    device: /shared

networks:
  jenkins_network:
      driver: bridge
  # zap-communication-network:
  #   driver: bridge
```

Annex B : JenkinsFile_Advanced

```
def pipelineConfig=[

    branches: [
        "master": 'master',
        "main": 'main',
        "custom/.*": 'custom'
    ],

    sonarProjectKeys: [
        "TestMultiBranchPipelineSpringBoot/master":
'testgroup_testprojectspringboot_AY2VC-wEJgYs34GJxpb-',
        "MultiBranch WebGoat/main":
```

```
'testgroup_webgoat_AY3CEp3yPjze8Ea254MD',
    ],

]

def getBranchName( def pipelineConfig) {
    branchName = null
    pipelineConfig.branches.any {
        branchPattern, branch ->
            if(env.BRANCH_NAME ==~ /${branchPattern}/){
                branchName = branch
            }
    }
    return branchName
}

def getSonarProjectKey(def pipelineConfig) {
    def sonarProjectKey = null
    pipelineConfig.sonarProjectKeys.any { projectName, projectKey ->
        if (env.JOB_NAME == projectName) {
            sonarProjectKey = projectKey
        }
    }
    return sonarProjectKey
}

def parameterValidation(params){
    currentBuild.result = 'ABORTED'
    error('BAD PARAM: ' + params)
}

pipeline {

      agent any

      tools {
        maven "Maven"
    }

    options {
        skipDefaultCheckout(true)
    }

      environment {

        NEXUS_VERSION = "nexus3"
        NEXUS_PROTOCOL = "http"
        NEXUS_URL = "192.168.1.164:8081"
        NEXUS_REPOSITORY = "maven-snapshots"
        NEXUS_CREDENTIAL_ID = "nexus-jenkins-user"
        BRANCH_NAME = "${getBranchName(pipelineConfig)}"

        SONAR_PROJECT_KEY = "${getSonarProjectKey(pipelineConfig)}"
        SONARQUBE_NAME = "SonarQube"
        SONARQUBE_AUTH_TOKEN_NAME = "SonarQube"

    }

    parameters {
        booleanParam(name: 'SKIP_SONARQUBE',
        description: 'Skip SonarQube Analysis',
        defaultValue: false)
        booleanParam(name: 'SKIP_NEXUS_PUBLISH',
        description: 'Skip Publish to Nexus Repository Manager',
        defaultValue: false)
    }


    stages {

        stage("Clean Up"){
            steps {
                cleanWs()
            }
        }

        stage("SCM chekout"){
            steps {
                checkout scm
            }
        }

        stage("Build"){
            steps {

                withCredentials([file(credentialsId: 'maven-settings',
variable: 'MAVEN_SETTINGS')]) {
                sh 'mvn -s ${MAVEN_SETTINGS} clean install -DskipTests'
                }
            }
        }

        stage("Test"){
            steps {

                withCredentials([file(credentialsId: 'maven-settings',
variable: 'MAVEN_SETTINGS')]) {
                sh 'mvn -s ${MAVEN_SETTINGS} test'
                }
            }
        }

        stage("SonarQube Analysis"){
            steps{
                script {
                    echo "SKIP_SONARQUBE parameter value:
${params.SKIP_SONARQUBE}"
                    echo "Running branch is : ${env.BRANCH_NAME}"
                    echo "Running job is : ${env.JOB_NAME}"
                    if (!params.SKIP_SONARQUBE && (env.BRANCH_NAME ==
'main' || env.BRANCH_NAME == 'master')) {
                        echo "This stage will only run if SKIP_SONARQUBE is
false and BRANCH_NAME is 'main' or 'master'"
                         scannerHome = tool "${SONARQUBE_NAME}";
                         withSonarQubeEnv("${SONARQUBE_NAME}") {

                            sh """
                                          ${scannerHome}/bin/sonar-scanner
\

-Dsonar.projectKey=${SONAR_PROJECT_KEY} \
                                          -Dsonar.sources=. \
                               -Dsonar.java.binaries=target/
                                       """

                            // Running the following for SonarQube produces
only errors found under src and since we have projects with not only java
it is not optimal
                            // withCredentials([file(credentialsId:
'maven-settings', variable: 'MAVEN_SETTINGS'),
                            //                   string(credentialsId:
'Sonarqube', variable: 'SONARQUBE_TOKEN')]) {
                            //       sh 'mvn -s ${MAVEN_SETTINGS} sonar:sonar
-Dsonar.token=${SONARQUBE_TOKEN} -Dsonar.projectKey=${SONAR_PROJECT_KEY}'
                            // }

                        }
                    } else {
                        echo "Skipping this stage as SKIP_SONARQUBE is true
or BRANCH_NAME is not 'main' or 'master'"
                    }
                }
            }
        }

        stage("Publish to Nexus Repository Manager with Quality Gate") {
            steps {
                script {
                    echo "SKIP_NEXUS_PUBLISH parameter value:
${params.SKIP_NEXUS_PUBLISH}"
                    echo "Running branch is : ${env.BRANCH_NAME}"
                    if (!params.SKIP_NEXUS_PUBLISH && (env.BRANCH_NAME ==
'main' || env.BRANCH_NAME == 'master')) {
                        echo "This stage will only run if
SKIP_NEXUS_PUBLISH is false and BRANCH_NAME is 'main' or 'master'"
                        // Check the status of the Quality Gate
                        // If it fails, mark the build as failed
                        if (waitForQualityGate().status != 'OK') {
                        error "Pipeline aborted due to Quality Gate
failure"
                        }
                        pom = readMavenPom file: "pom.xml";
                        filesByGlob = findFiles(glob:
"target/*.${pom.packaging}");
                        echo "${filesByGlob[0].name} ${filesByGlob[0].path}
${filesByGlob[0].directory} ${filesByGlob[0].length}
${filesByGlob[0].lastModified}"
                        artifactPath = filesByGlob[0].path;
                        artifactExists = fileExists artifactPath;
                        if(artifactExists) {
                            echo "*** File: ${artifactPath}, group:
${pom.groupId}, packaging: ${pom.packaging}, version ${pom.version}";
                            nexusArtifactUploader(
                                nexusVersion: NEXUS_VERSION,
                                protocol: NEXUS_PROTOCOL,
                                nexusUrl: NEXUS_URL,
                                groupId: pom.groupId,
                                version: pom.version,
                                repository: NEXUS_REPOSITORY,
                                credentialsId: NEXUS_CREDENTIAL_ID,
                                artifacts: [
                                    [artifactId: pom.artifactId,
                                    classifier: '',
                                    file: artifactPath,
                                    type: pom.packaging],
                                    [artifactId: pom.artifactId,
                                    classifier: '',
                                    file: "pom.xml",
                                    type: "pom"]
                                ]
                            );
                        } else {
                            error "*** File: ${artifactPath}, could not be
found";
                        }
                    } else {
                        echo "Skipping this stage as SKIP_SONARQUBE is true
or BRANCH_NAME is not 'main' or 'master'"
                    }
                }
            }
        }

    }

    post {
        always {
            // Publish Surefire test report
            junit skipPublishingChecks: true, testResults:
'target/surefire-reports/*.xml'

             // Zip the test results directory
            zip zipFile: 'surefire-report.zip', archive: false, dir:
'target/site'
            // Archive the zip file
            archiveArtifacts artifacts: 'surefire-report.zip', fingerprint:
true
        }
    }
}
```

Annex C : JenkinsFile_Zap_WebGoat_Scan

```
def waitForContainer(containerName, port) {
    timeout(time: 120, unit: 'SECONDS') {
        sh """
        #!/bin/bash
        until docker exec $containerName sh -c 'curl -s
http://localhost:$port > /dev/null'
        do
          echo 'Waiting for container $containerName on port $port...'
          sleep 1
        done
        echo 'Container $containerName on port $port is ready'
        """
    }
}

pipeline {
    agent any

    options {
        skipDefaultCheckout(true)
    }
    parameters {
        choice  choices: ['Baseline', 'Full'],
                 description: 'Type of scan that is going to perform inside 
the container',
                 name: 'SCAN_TYPE'

        string defaultValue: 'zap-communication-network',
                 description: "ZAP container network",
                 name: 'NETWORK'

        booleanParam defaultValue: true,
                 description: 'Parameter to know if wanna generate 
report.',
                 name: 'GENERATE_REPORT'
    }

    stages {
        stage('Pipeline Info'){
            steps {
                script {
                    echo '<--Parameter Initialization-->'
                    echo """
                         The current parameters are:
                             Scan Type: ${params.SCAN_TYPE}
                             Scan Type: ${params.NETWORK}
                             Generate report: ${params.GENERATE_REPORT}
                         """
                }
            }
        }

        stage("Clean Up"){
            steps {
                cleanWs()
            }
        }

        stage("SCM chekout"){
            steps {
                checkout scm
            }
        }

        stage('Creating Docker virtual network') {
            steps {
                script {
                    echo '<-- Docker virtual network-->'
                    def networkExists = sh(returnStatus: true, script:
```

```
'docker network inspect zap-communication-network')
                    if (networkExists != 0) {
                        echo 'Network does not exist, create it'
                        sh 'docker network create
zap-communication-network'
                    } else {
                        echo 'Network zap-communication-network already
exists.'
                    }
                }
            }
        }

        stage('Setting up WebGoat docker container') {
            steps {
                echo 'Pulling up last WebGoat container --> Start'
                sh 'docker pull webgoat/webgoat:latest'
                echo 'Pulling up last VMS container --> End'
                echo 'Starting container --> Start'
                sh """
                    docker run --detach \\
                    --network ${params.NETWORK} \\
                    -p 8383:8080 -p 9090:9090 \\
                    --name webgoat \\
                    webgoat/webgoat \\
                """
                // We hit internal ports here to verify that container is
ready along with the application
                waitForContainer("webgoat",9090)
                waitForContainer("webgoat",8080)

            }
        }

        stage('Creating user via cURL') {
            steps {
                sh """
                    curl 'http://192.168.1.164:8383/WebGoat/register.mvc'
-X POST --data-raw
'username=testuser&password=testuser&matchingPassword=testuser&agree=agree'
                """
            }
        }

        stage('Scanning target on ZAP container') {
            steps {
                catchError(buildResult: 'UNSTABLE', stageResult:
'UNSTABLE') {
                    script {
                        echo """
                        ZAP can exit with :
                        0: Success
                        1: At least 1 FAIL
                        2: At least one WARN and no FAILs
                        3: Any other failure
                        Values 1 to 3 will mark the build as UNSTABLE
                        """

                        def jenkinsPath = "${WORKSPACE}/jenkins/zap"
                        def hostPath =
jenkinsPath.replace('/var/jenkins_home', '/jenkins_home')

                        echo "<-- ${jenkinsPath} is the path corresponding
to Jenkins -->"
                        echo "<-- ${hostPath} is the translated path
corresponding to Host machine -->"

                        echo '<-- The following will be mounted to zap/wrk
-->'
                        sh "ls -lR ${jenkinsPath}"

                        scan_type = "${params.SCAN_TYPE}"
                        echo "----> scan_type: $scan_type"
                        target = "http://192.168.1.164:8383/WebGoat/login"
                        if (scan_type == 'Baseline') {
                        sh """
                            docker run \\
                                --rm \\
                                --name zap-${BUILD_ID} \\
                                -p 8484:8080 \\
                                -p 8443:8443 \\
                                -v "${hostPath}:/zap/wrk/:rw" \\
                                --network ${params.NETWORK} \\
                                zaproxy/zap-stable \\
                                zap-baseline.py \\
                                    -t ${target} \\
                                    -r report.html \\
                                    -w report.md \\
                                    -n WebGoat.context \\
                                    -U testuser \\
                                    -I -j -d \\
                                    -z "-config api.addrs.addr.name=.*
-config api.addrs.addr.regex=true -config api.disablekey=true"
                        """

                        }
                        else if (scan_type == 'Full') {
                            sh """
                                docker run \\
                                --rm \\
                                --name zap-${BUILD_ID} \\
                                -p 8484:8080 \\
                                -p 8443:8443 \\
                                -v "${hostPath}:/zap/wrk/:rw" \\
                                --network ${params.NETWORK} \\
                                zaproxy/zap-stable \\
                                zap-full-scan.py \\
                                    -t ${target} \\
                                    -r report.html \\
                                    -w report.md \\
                                    -n WebGoat.context \\
                                    -U testuser \\
                                    -m 10 \\
                                    -T 60 \\
                                    -I -j -d \\
                                    -z "-config api.addrs.addr.name=.*
-config api.addrs.addr.regex=true -config api.disablekey=true"
                            """
                        }
                        else {
                            echo 'Something went wrong...'
                        }

                    }
                }
                }
        }


        stage('Archive Reports to Workspace') {
            when {
                environment name : 'GENERATE_REPORT', value: 'true'
            }
            steps {
              catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {
                script {
                    zip zipFile: "ReportsAndContext-${BUILD_ID}.zip",
archive: true, dir : "${WORKSPACE}/jenkins/zap/"
                }
              }

            }
        }
    }

    post {
        always {
            echo 'Removing containers'
            sh """
                    docker stop webgoat
                    docker rm webgoat
                """
        }
    }
}
```

Annex D : WebGoat.context

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
    <context>
        <name>WebGoat</name>
        <desc/>
        <inscope>true</inscope>
        <incregexes>http://192.168.1.164:8383.*</incregexes>
        <incregexes>http://192.168.1.164:8383/WebGoat/login.*</incregexes>
        <incregexes>http://192.168.1.164:8383/WebGoat/start.*</incregexes>

<incregexes>http://192.168.1.164:8383/WebGoat/start.mvc?username=testuser#l
esson/.*</incregexes>
        <excregexes>http://192.168.1.164:8383/WebGoat/logout</excregexes>
        <tech>
            <include>Db</include>
            <include>Db.CouchDB</include>
            <include>Db.Firebird</include>
            <include>Db.HypersonicSQL</include>
            <include>Db.IBM DB2</include>
            <include>Db.MariaDB</include>
            <include>Db.Microsoft Access</include>
            <include>Db.Microsoft SQL Server</include>
            <include>Db.MongoDB</include>
            <include>Db.MySQL</include>
            <include>Db.Oracle</include>
            <include>Db.PostgreSQL</include>
            <include>Db.SAP MaxDB</include>
            <include>Db.SQLite</include>
            <include>Db.Sybase</include>
            <include>Language</include>
            <include>Language.ASP</include>
            <include>Language.C</include>
            <include>Language.JSP/Servlet</include>
            <include>Language.Java</include>
            <include>Language.Java.Spring</include>
            <include>Language.JavaScript</include>
            <include>Language.PHP</include>
            <include>Language.Python</include>
            <include>Language.Ruby</include>
            <include>Language.XML</include>
            <include>OS</include>
            <include>OS.Linux</include>
            <include>OS.MacOS</include>
            <include>OS.Windows</include>
            <include>SCM</include>
            <include>SCM.Git</include>
            <include>SCM.SVN</include>
            <include>WS</include>
            <include>WS.Apache</include>
            <include>WS.IIS</include>
            <include>WS.Tomcat</include>
        </tech>
        <urlparser>
            <class>org.zaproxy.zap.model.StandardParameterParser</class>
            <config>{"kvps":"&amp;","kvs":"=","struct":[]}</config>
        </urlparser>
        <postparser>
            <class>org.zaproxy.zap.model.StandardParameterParser</class>
            <config>{"kvps":"&amp;","kvs":"=","struct":[]}</config>
        </postparser>
        <authentication>
            <type>2</type>
            <strategy>EACH_RESP</strategy>
            <pollurl/>
            <polldata/>
            <pollheaders/>
            <pollfreq>60</pollfreq>
            <pollunits>REQUESTS</pollunits>
            <loggedin>&lt;button type="button" data-toggle="dropdown"
class="btn btn-default dropdown-toggle" id="user-menu"&gt;</loggedin>
            <loggedout>&lt;button class="btn btn-primary btn-block"
type="submit"&gt;Sign in&lt;/button&gt;</loggedout>
            <form>

<loginurl>http://192.168.1.164:8383/WebGoat/login</loginurl>

<loginbody>username={%username%}&amp;password={%password%}</loginbody>

<loginpageurl>http://192.168.1.164:8383/WebGoat/login</loginpageurl>
            </form>
        </authentication>
        <users>

<user>10354;true;dGVzdHVzZXI=;2;dGVzdHVzZXI=~dGVzdHVzZXI=~</user>
        </users>
        <forceduser>10354</forceduser>
        <session>
            <type>0</type>
        </session>
        <authorization>
            <type>0</type>
            <basic>
                <header/>
                <body/>
                <logic>AND</logic>
                <code>-1</code>
            </basic>
        </authorization>
    </context>
</configuration>
```