Lab 5

Managing Security in a DevOps Process

Review Question

- What is DevSecOps ?

Lecture : How to integrate Security tools in an agile process

[https://nullsweep.com/integrating-security-with-agile-development/]

Traditionally, security has worked with project teams during two phases of execution: technical requirements design and right before go-live.

After a team has gathered their business requirements and sorted out a target architecture, they may go to a security review, run through a threat modeling and data flow exercise, then take a stack of security requirements and put them into a design document. This document gets handed over to development when all the requirements are reasonably stable, and development goes off to build it for the next several months (or quarters).

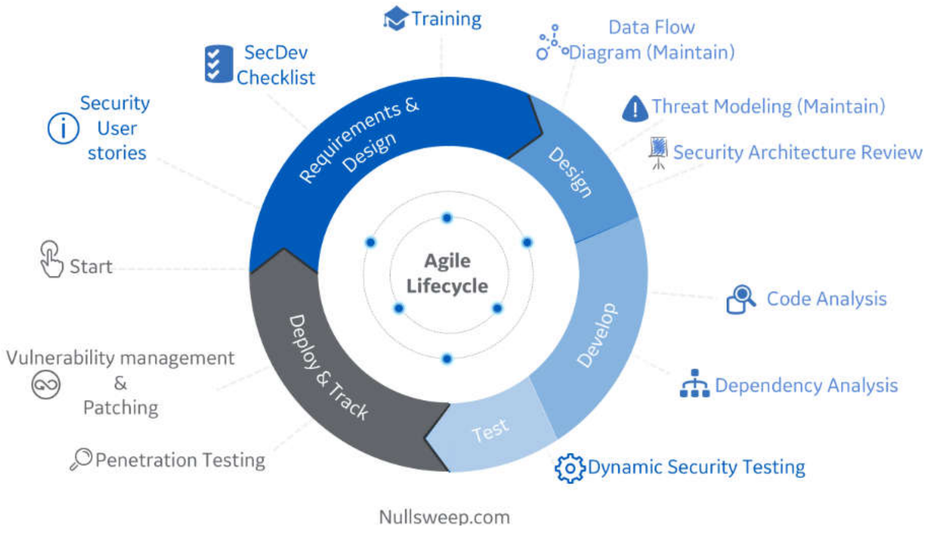
After they have a working program and users have signed off, the team comes back to security for sign off. At this point, security runs tools and penetration tests, and comes back with a stack of vulnerabilities.

Unfortunately for everyone involved, this is often one of the last things before a production go-live is scheduled, and fixing security issues certainly bothers the project schedule.

Further compounding the problem, developers who unknowingly created the security bugs may have done so months before, and so no longer have an intimate knowledge of the particular piece of code containing the vulnerability. They also may have been using an insecure programming pattern that replicated itself across the code base, resulting in a large number of similar vulnerabilities scattered around the program.

The end result is that everyone is unhappy! security is unhappy because open issues will make it into production (the schedule usually wins). Product is unhappy because they worked overtime fixing some of the issues, and have more unexpected work scheduled. Leadership is unhappy because it looks like security and development didn't work well together.

There is, of course, a better way. If we are able to automate security tooling, we can incorporate it at every stage of the agile cycle, and improve outcomes for both security and the development teams.



Security in Sprint Planning

When sprint planning, I ask teams to do a few things:

- Include security stories in the backlog, and prioritize them alongside business features. These may come about as a result of threat modeling or outputs from security audits / pentests / scanners.

- For each story selected for the current sprint, review a secure development checklist. Essentially this is a repository of guidelines on preventing common issues tied to functionality - using parameterized queries for DB interactions, filtering dynamic data before displaying on a web template, etc.
- Determine if any security specific training might be needed. This is similar to any other technical training that might be needed by the engineer assigned the story.

Security During Design

As stories are selected, teams then move on to formal or informal design as they determine if any infrastructure changes are needed, what software architectures and integrations will be used, etc. During this phase, I ask teams to consider three questions which shouldn't add significantly to the workload:

1. Does this story change where we are ingesting or outputting data? If so, update the data flow diagram.
2. Does this story change our potential risk profile, or create new threats? Take some time to update and review the threat model.
3. Is there anything in this story that we might want to review formally or informally with a security expert?

Security During Development & Test

As we move into development, automation becomes critical. Security continues to be a blocker instead of an enabler until the tools at this stage can be automated and integrated into the development cycle. To this end, I generally recommend striving for the following:

- Integrate a static analysis tool into the IDE whenever possible. SonarQube can be configured this way, along with many vendor SAST tools. This allows developers to see security bugs in near real time, minimizing re-work and maximizing learning.
- Integrate Dependency / Open source security checks into local build processes whenever possible. This will catch vulnerable dependency versions as soon as they are added to projects in one team members machine.
- Integrate both of the above into a CI/CD pipeline, and break the build on issue thresholds. This way, if someone on the team isn't running the tools locally, you can at least enforce security prior to any deployments.
- Configure dynamic test tools into the CI/CD pipeline on deploys. This one can be a little trickier to setup. To get started with this, I usually recommend configuring a ZAP baseline scan integration, or using your DAST vendor tools. Additional dynamic tests can be added over time.

Security During Deployment

In the deployment phase, we need to make sure we are doing a few things:

- Updating any out of date application components (for instance, rolling patches into a docker container before deploying the latest changes)
- Considering if any changes being deployed require changes to our alerting / monitoring / logging / prevention infrastructure.

Hands-on Lab: Creating a Secure Pipeline: Jenkins with SonarQube and DependencyCheck

In this lab, you will walk through a basic Jenkins setup for CI/CD, and integrate SonarQube and DependencyCheck for security scanning. In my next article, I'll expand this to include DAST tools.

1. Quick infrastructure setup

If you want to follow along, there is a pre-made docker-compose file for you that will create both a Jenkins and a SonarQube instance. To get these running, you will need git, docker, and docker-compose installed on your machine. Then run the following commands:

```
$ git clone https://github.com/Charlie-belmer/secure-pipeline-example.git
$ cd secure-pipeline-example/sast_pipeline_example
$ docker-compose up
```

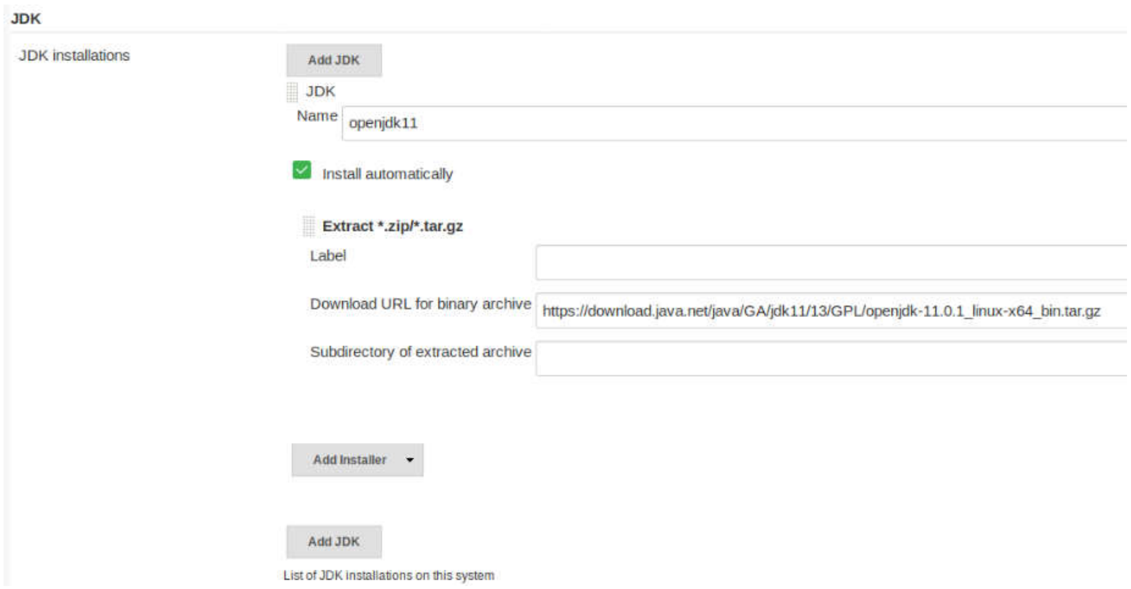
Watch the startup data for the Jenkins initial admin password - you will need this to get setup.

2. Configuring Jenkins To Build WebGoat

We're going to scan a known vulnerable webapp, WebGoat, which is an OWASP project used for learning basic web penetration testing skills and vulnerabilities. A good scanner should find a lot of things!

Anyway, let's get on with Jenkins. Navigate in your browser to http://localhost:8080 and enter the admin password shown in the terminal running docker. Go ahead and install the default plugins (for a deployed instance, I would recommend only installing plugins you will actually use) and create your first admin user.

WebGoat requires Java 11 to build, which Jenkins won't install automatically. Head over to the **main page -> Manage Jenkins -> Global Tool Configuration**. There are two sections here we will update now: JDK installation and Maven installations. We need to add a link to a Java 11 installer - I used https://download.java.net/java/GA/jdk11/13/GPL/openjdk-11.0.1_linux-x64_bin.tar.gz and we can use the default maven. Your config should look like this:


JDK

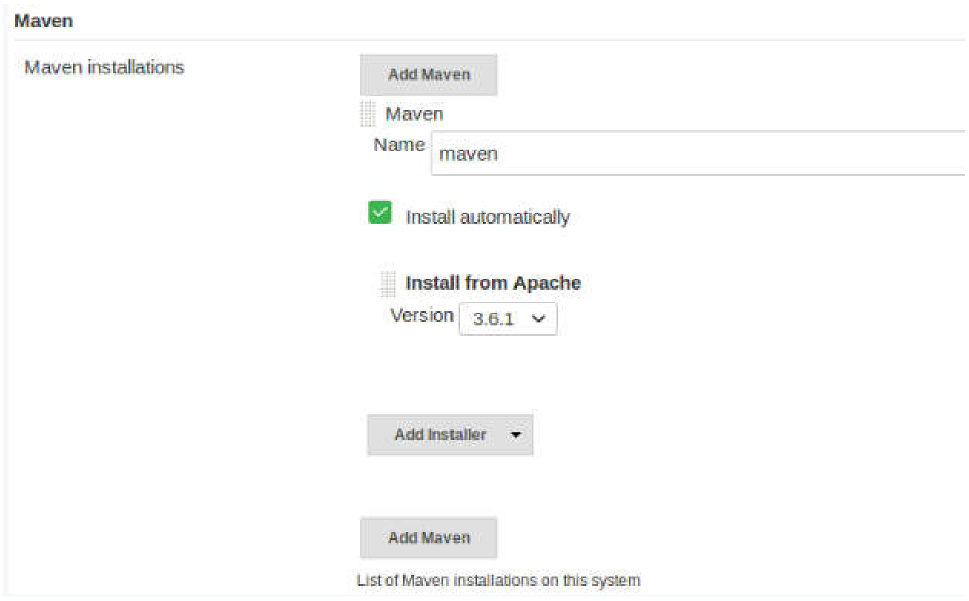
JDK installations

JDK
Name: openjdk11

✓ Install automatically

Extract *.zip/*.tar.gz
Label:
Download URL for binary archive: https://download.java.net/java/GA/jdk11/13/GPL/openjdk-11.0.1_linux-x64_bin.tar.gz
Subdirectory of extracted archive:

Jenkins JDK11



Jenkins Maven install configuration

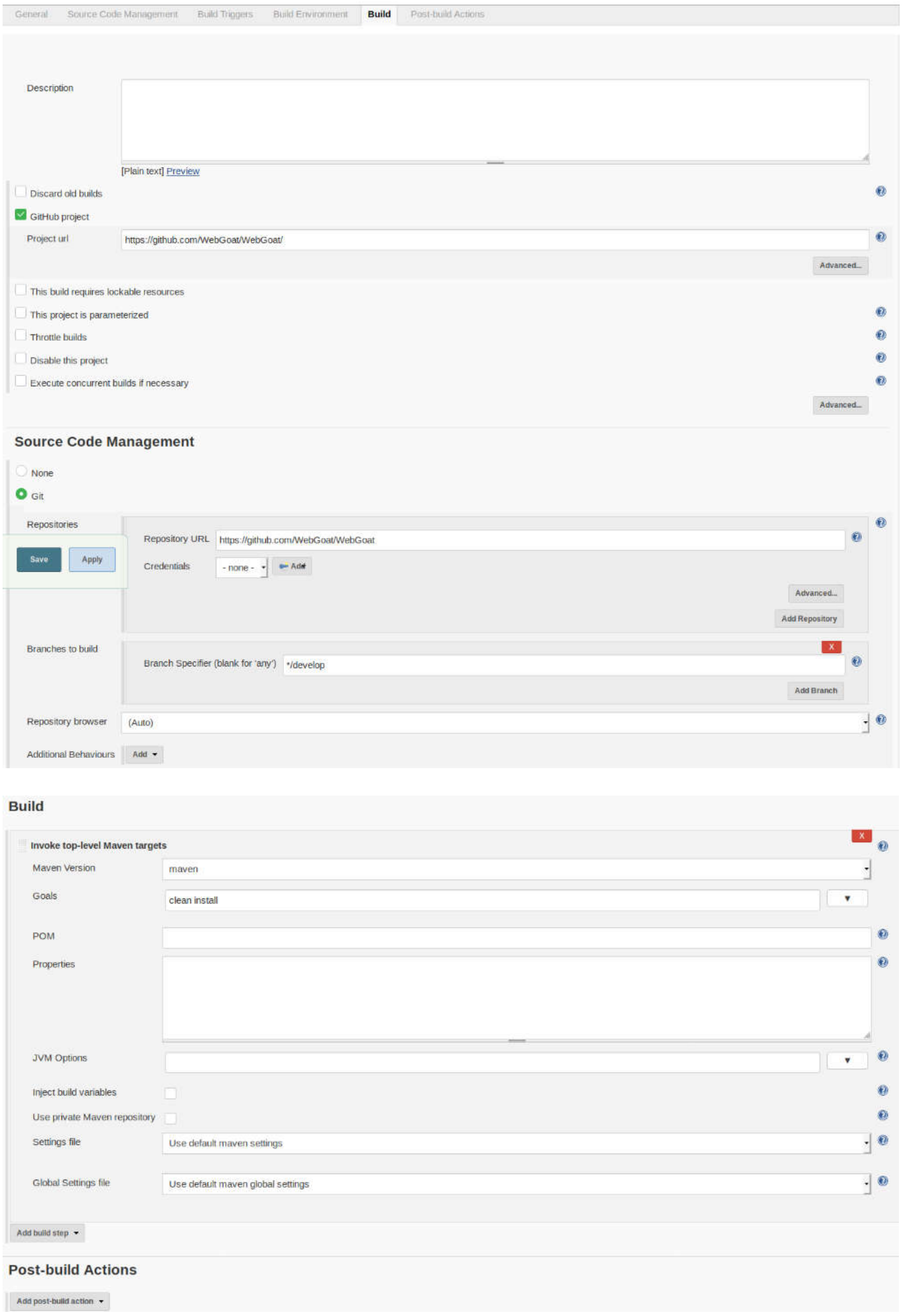
Finally, we have to set the JAVA_HOME variable. In the Jenkins -> Manage Jenkins -> Configure System menu, enable environment varaibles and set JAVA_HOME equal to /var/jenkins_home/tools/hudson.model.JDK/openjdk11/jdk-11.0.1/.

Now let's create a pipeline for WebGoat and make sure it builds successfully. Back on the main page choose new item -> freestyle project.

The initial setup is pretty simple:

- Add Webgoat to the various github setting locations (https://github.com/WebGoat/WebGoat/)
- Set the target branch to */develop
- Create a maven build step ("Invoke top level maven targets") and give it the command "clean install"

Here is the full pipeline configuration:



Description

[Plain text] Preview

- [ ] Discard old builds
- [x] GitHub project

Project url: https://github.com/WebGoat/WebGoat/

Advanced...

- [ ] This build requires lockable resources
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary

Advanced...

Source Code Management

- ○ None
- ● Git

Repositories

Repository URL: https://github.com/WebGoat/WebGoat

Credentials: - none -

Branches to build

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

Repository browser: (Auto)

Additional Behaviours

Build

Invoke top-level Maven targets

Maven Version	maven
Goals	clean install
POM	
Properties	
JVM Options	
Inject build variables	☐
Use private Maven repository	☐
Settings file	Use default maven settings
Global Settings file	Use default maven global settings

Post-build Actions

Basic WebGoat Pipeline

Try running it and making sure that everything builds successfully.

4. Adding SonarQube and DependencyCheck

SonarQube setup & security

We already have a SonarQube instance running, we just need to link and configure Jenkins to use it. Log in to http://localhost:9000 and use the default sonarQube login of admin/admin.

- Although this is only for practice, I still want to secure our SonarQube instance, so do the following:
- Change the admin password
- Go to administration-> security and turn on "Force user authentication"
- Create a new user for Jenkins.
- Log into the new user, go to the profile -> security section, and generate a token. Copy this for later use.

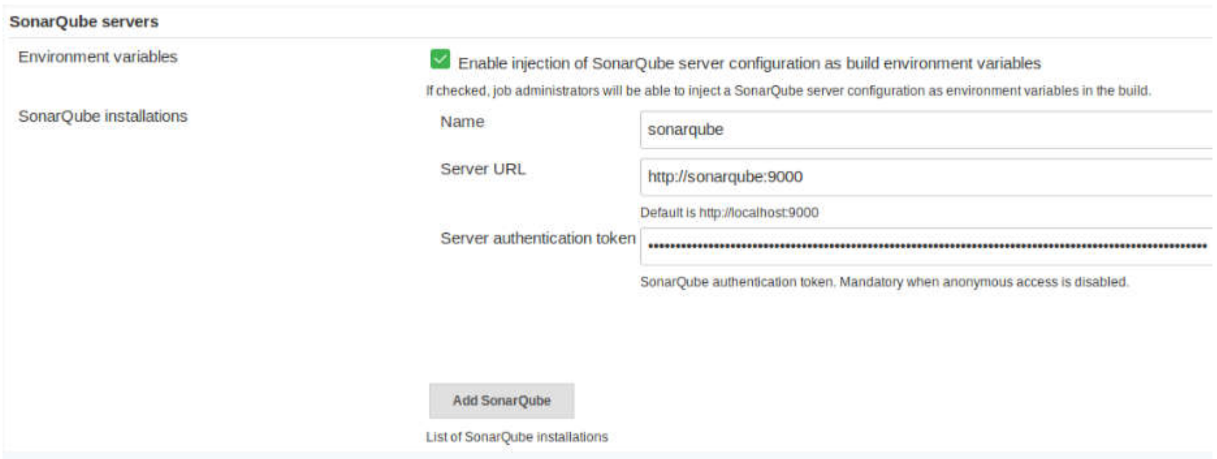
Finally, create a project named "webgoat" with your jenkins user.

Configure the plugins for Jenkins

We will need two new plugins for jenkins. In the Jenkins home page, go to Mange Jenkins -> Manage Plugins. On the Available tab find and select "OWASP Dependency-Check Plugin" and "SonarQube Scanner for Jenkins". Install them without restarting.

Back on the Jenkins home, go to Manage Jenkins -> Global Tool Configuration. You should see a new option for SonarQube Scanner. Add an installation here (I just chose the latest from Maven Central) and save.

Finally, head over to Jenkins -> Manage Jenkins -> Configure System and add a sonarqube instance. The URL with our docker container is http://sonarqube:9000 and the token should be the one you saved while setting up the Jenkins user in SonarQube. Here is my setup:



SonarQube servers

Environment variables: ✓ Enable injection of SonarQube server configuration as build environment variables

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

SonarQube installations

Name: sonarqube

Server URL: http://sonarqube:9000

Default is http://localhost:9000

Server authentication token: ••

SonarQube authentication token. Mandatory when anonymous access is disabled.

List of SonarQube installations

SonarQube Settings

One other thing I had to do to get SonarQube working properly. For some reason I couldn't completely determine, the SonarQube startup script was truncating the JAVA_HOME path incorrectly, causing errors during the pipeline. To solve this, log into the docker container manually and update the sonar script to the proper JAVA_HOME.

```
$ docker exec -it secure_pipeline_jenkins_1 bash
jenkins@2ea0acb5905d:/$ cd
/var/jenkins_home/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonar
qube
```

```
jenkins@2ea0acb5905d:~/tools/hudson.plugins.sonar.SonarRunnerInstallation/
sonarqube$ head bin/sonar-scanner
#!/bin/sh
#
# SonarQube Scanner Startup Script for Unix
#
# Optional ENV vars:
#   SONAR_SCANNER_OPTS - Parameters passed to the Java VM when running the
SonarQube Scanner
#   SONAR_SCANNER_DEBUG_OPTS - Extra parameters passed to the Java VM for
debugging
#   JAVA_HOME - Location of Java's installation

JAVA_HOME="/var/jenkins_home/tools/hudson.model.JDK/openjdk11-remote/jdk-
11.0.1"
```

Add SonarQube and DependencyCheck to the pipeline

Now we can add these to our pipeline and start scanning with every build.

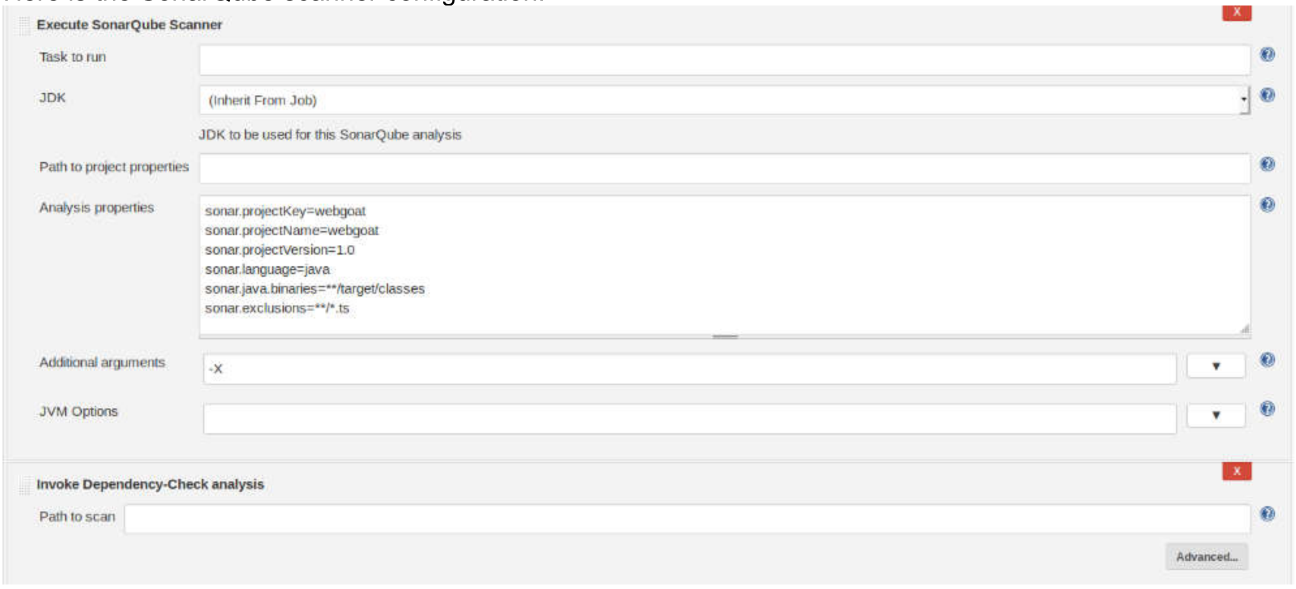
In the pipeline created earlier, add two new build steps - Invoke Dependency-Check analysis and Execute SonarQube Scanner. In the SonarQube scanner, add the configuration settings required - the project key and name should match the project you created in SonarQube.

```
sonar.projectKey=webgoat
sonar.projectName=webgoat
sonar.projectVersion=1.0
sonar.language=java
sonar.java.binaries=**/target/classes
sonar.exclusions=**/*.ts
```

I am excluding the TypeScript files above since we did not setup Node or a JS build step for our project. In a real project, we would want to ensure that they were also scannable.

In the DependencyCheck advanced section, check to generate HTML reports as well for easier viewing.

Here is the SonarQube scanner configuration:



Execute SonarQube Scanner

Task to run	
JDK	(Inherit From Job)
	JDK to be used for this SonarQube analysis
Path to project properties	

Analysis properties	sonar.projectKey=webgoat
	sonar.projectName=webgoat
	sonar.projectVersion=1.0
	sonar.language=java
	sonar.java.binaries=**/target/classes
	sonar.exclusions=**/*.ts

Additional arguments	-X
JVM Options	

Invoke Dependency-Check analysis

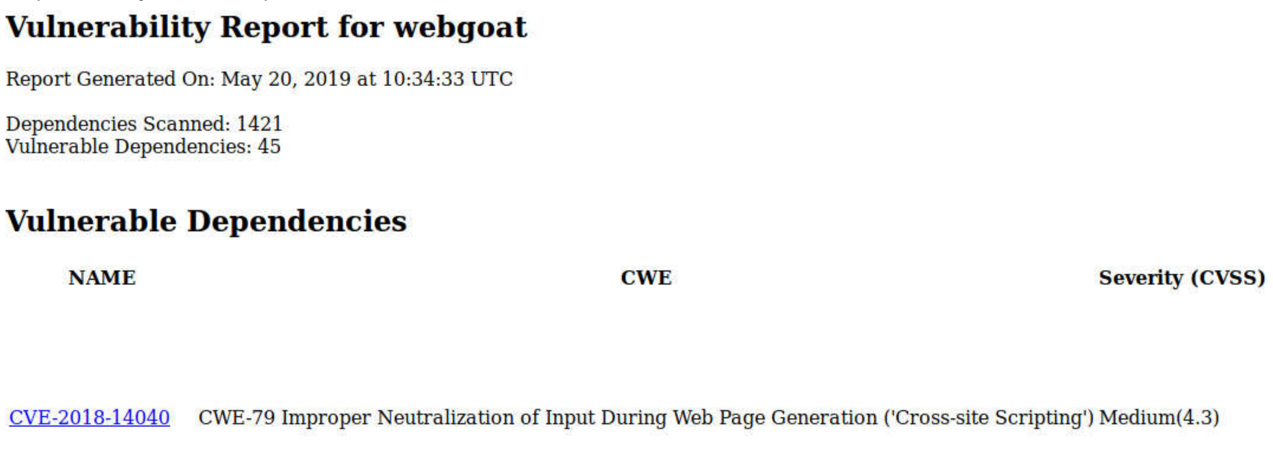
Path to scan	

Full Secure Pipeline

Kick off a build and make sure it runs correctly. Afterwards, you should be able to see results.

Viewing Reports

If all runs successfully, logging into SonarQube will show you security scan details (with plenty of findings!) and the pipeline can show you the dependencyCheck results in the workspace -> dependency-check-report.html file.



Vulnerability Report for webgoat

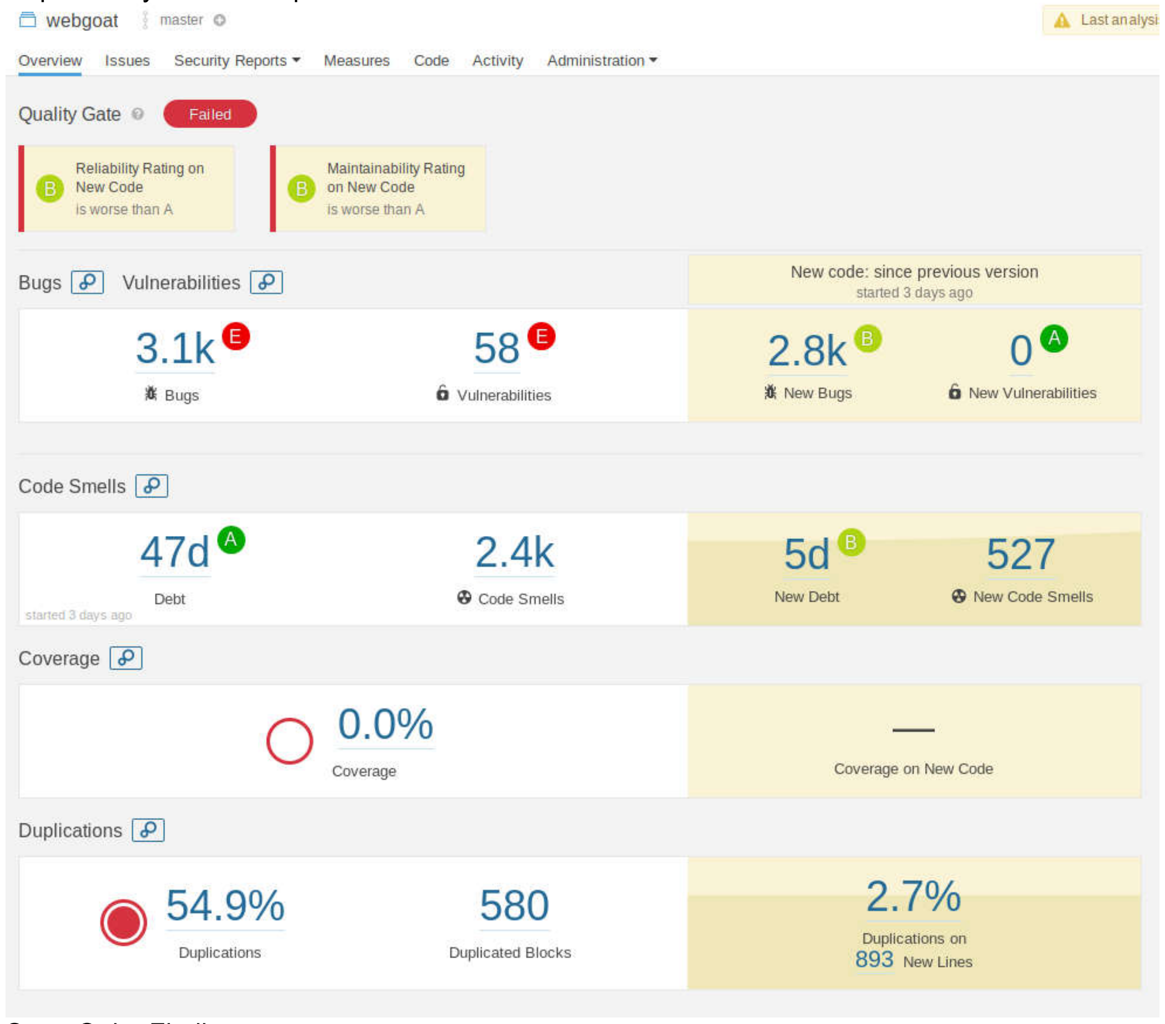
Report Generated On: May 20, 2019 at 10:34:33 UTC

Dependencies Scanned: 1421
Vulnerable Dependencies: 45

Vulnerable Dependencies

NAME	CWE	Severity (CVSS)
CVE-2018-14040	CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	Medium(4.3)

DependencyCheck Sample Data



webgoat master | Last analysi

Overview | Issues | Security Reports | Measures | Code | Activity | Administration

Quality Gate: Failed

- B Reliability Rating on New Code is worse than A
- B Maintainability Rating on New Code is worse than A

Bugs | Vulnerabilities — New code: since previous version, started 3 days ago

3.1k E Bugs	58 E Vulnerabilities	2.8k B New Bugs	0 A New Vulnerabilities

Code Smells

47d A Debt	2.4k Code Smells	5d B New Debt	527 New Code Smells

started 3 days ago

Coverage

0.0% Coverage	— Coverage on New Code

Duplications

54.9% Duplications	580 Duplicated Blocks	2.7% Duplications on 893 New Lines

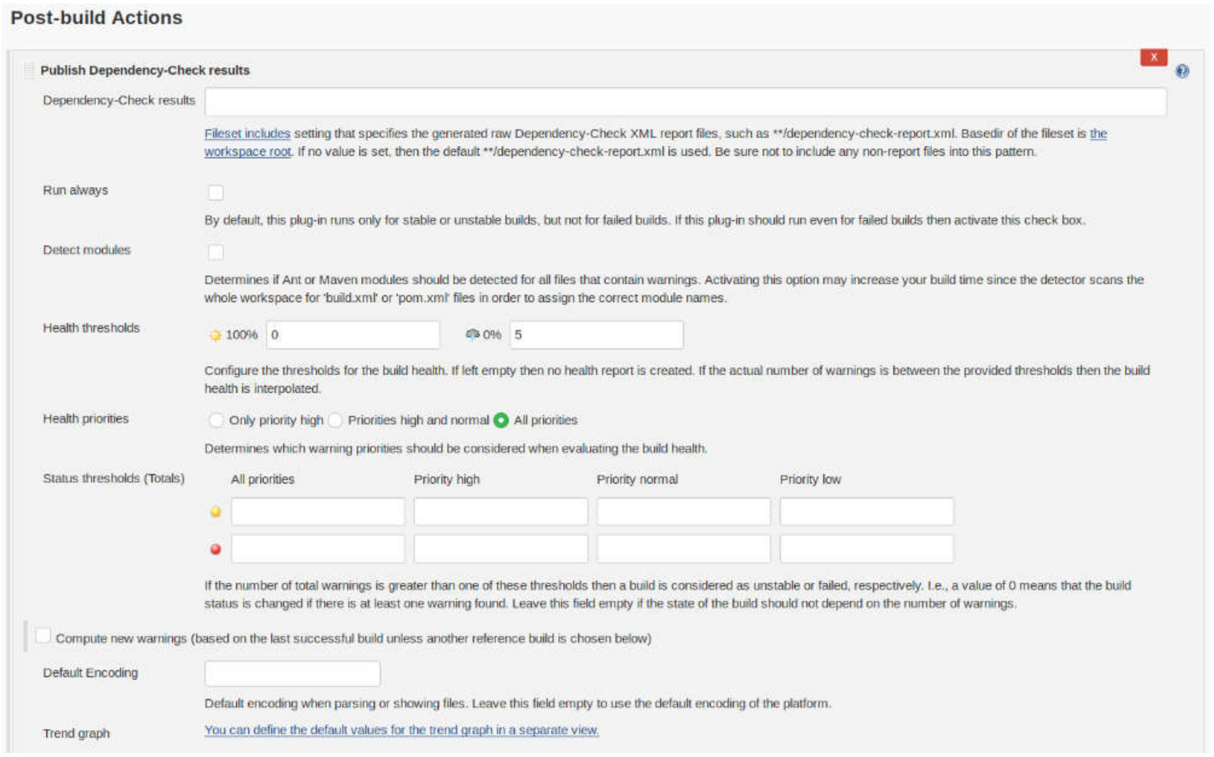
SonarQube Findings

You can and should at this point consider additional SonarQube plugins (or other SAST tools) that are specifically for your languages and frameworks.

Breaking the Build

We want to know when something isn't working right at the build phase. SonarQube gives us this for free with the plugin (you should see a nice red ERROR tag under the SonarQube Quality gate) but DependencyCheck requires one more configuration.

Add a post-build check for "Publish Dependency Check Results" and expand the advanced tabs. Just add some threshold data and the build will fail or be marked unstable according to the rules set.



Post-build Actions

Publish Dependency-Check results

Dependency-Check results

Fileset includes setting that specifies the generated raw Dependency-Check XML report files, such as **/dependency-check-report.xml. Basedir of the fileset is the workspace root. If no value is set, then the default **/dependency-check-report.xml is used. Be sure not to include any non-report files into this pattern.

Run always ☐

By default, this plug-in runs only for stable or unstable builds, but not for failed builds. If this plug-in should run even for failed builds then activate this check box.

Detect modules ☐

Determines if Ant or Maven modules should be detected for all files that contain warnings. Activating this option may increase your build time since the detector scans the whole workspace for 'build.xml' or 'pom.xml' files in order to assign the correct module names.

Health thresholds: 100% 0 — 0% 5

Configure the thresholds for the build health. If left empty then no health report is created. If the actual number of warnings is between the provided thresholds then the build health is interpolated.

Health priorities: Only priority high / Priorities high and normal / [✓ All priorities]

Determines which warning priorities should be considered when evaluating the build health.

Status thresholds (Totals)

	All priorities	Priority high	Priority normal	Priority low
●				
●				

If the number of total warnings is greater than one of these thresholds then a build is considered as unstable or failed, respectively. I.e., a value of 0 means that the build status is changed if there is at least one warning found. Leave this field empty if the state of the build should not depend on the number of warnings.

☐ Compute new warnings (based on the last successful build unless another reference build is chosen below)

Default Encoding

Default encoding when parsing or showing files. Leave this field empty to use the default encoding of the platform.

Trend graph: You can define the default values for the trend graph in a separate view.

Getting a CI/CD pipeline running with some basic security checks can be done within a few minutes. This will help keep your published artifacts in better shape and ensure the team has an opportunity to learn about security issues as soon as they emerge.

Answer to the review Question

What isDevSecOps ?

If you want a simple DevSecOps definition, it is short for development, security and operations. Its mantra is to make everyone accountable for security with the objective of implementing security decisions and actions at the same scale and speed as development and operations decisions and actions.

Every organization with a DevOps framework should be looking to shift towards a DevSecOps mindset and bringing individuals of all abilities and across all technology disciplines to a higher level of proficiency in security. From testing for potential security exploits to building business-driven security services, a DevSecOps framework that uses DevSecOps tools ensures security is built into applications rather than being bolted on haphazardly afterwards.

By ensuring that security is present during every stage of the software delivery lifecycle, we experience continuous integration where the cost of compliance is reduced, and software is delivered and released faster.