Project Research Document

IBM Gradle/Jenkins Plugin for Security Vulnerabilities

X00107570 | Paul Kenny

October 12, 2016

PROJECT OVERVIEW

This project is to be undertaken in association with IBM. IBM implement the continuous delivery approach to software development. This means that an engineer will write a piece of code, for a component of an application, which will be tested and deployed within the shortest timeframe allowable. The piece of code may be integrated into the production environment within 24 hours. To make this possible IBM utilise a continuous-integration pipeline strategy for their development and deployment, using Jenkins. (‘Jenkins’, 2016) Source code and unit tests are written, then are built using Gradle build scripts. (‘Gradle’, 2016) This code is then tested and analysed. When the code has passed it is deployed in a Docker container, this allows the applications to be repeatedly deployed in consistent environments. (Docker, 2016) These containers are then stored in a repository and can then be integrated into applications and deployed in the production environment as instances in the cloud. (Figure 1.) This project will focus specifically on the Java platform.

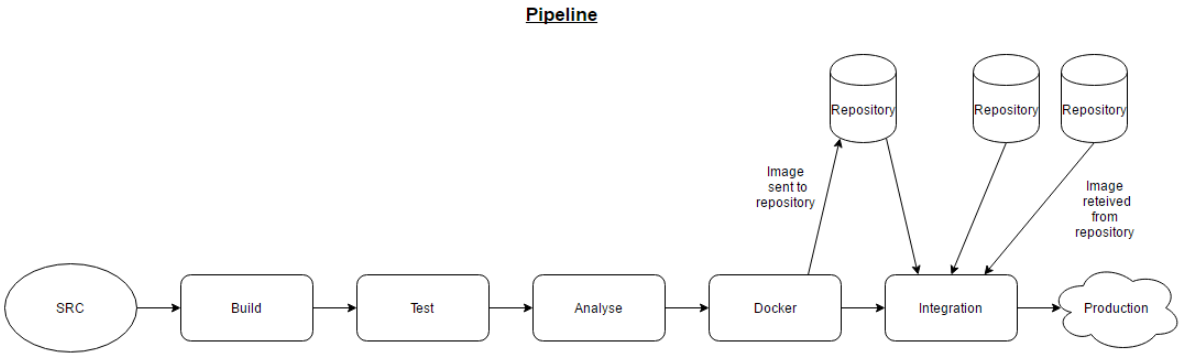

Figure 1: Continuous integration using Jenkins pipeline

During the production process, security vulnerabilities can arise at any stage. The aim of this project is to build an application which will scan the component, at various stages of the process, for known and potential security vulnerabilities. To achieve this the project will be divided into two distinct phases.

Phase 1: This phase will concentrate on the known vulnerabilities. The project will utilise various existing tools and utilities at the various stages, to scan the component for known vulnerabilities. Once vulnerabilities are identified the results are aggregated and reported back to the developer, possibly through the Jenkin's UI.

There are three main stages in this process at which security vulnerabilities may arise. Identifying the security vulnerabilities at these stages will increase the likelihood that the component will be secure at the deployment stage. Secure components mean secure applications. The three main stages identified are the Source Code, the Gradle Build and the Docker Container Image.

Source Code:

At the source code stage vulnerabilities occur through insecure coding practices. These may not be conscious practices among developers, yet the fast moving pace of continuous integration may still cause them to arise. The best way to combat these vulnerabilities is to scan the source code for vulnerable code.

Existing tools which can be utilised by this project to scan the source code:

1. **FindBugs**: FindBugs is an open-source, static analysis program designed to look for bugs in Java code. FindBugs scans the byte code to find code defects or suspicious code. There are nine separate categories of warnings, ranging from bad practice to security issues. It can be deployed through a Gradle or Jenkins plugin which will produce a report in the form of warnings to the developer. ('FindBugs', 2016)
2. **PMD**: PMD is an open-source source code analyser designed to look for bugs in Java code. Unlike FindBugs, is does not come with functionality for detecting security vulnerabilities out of the box. However, new rules can be written to PMD, in Java or XPath expressions, without compromising its inherent functionality. PMD can be deployed through a Gradle or Jenkins plugin. ('PMD', 2016)
3. **OWASP LAPSE+**: LAPSE+ is an open source static analysis tool which detects security vulnerabilities in the source code of Java applications. It is specifically aimed at detecting vulnerabilities related to the injection of untrusted data to manipulate the behaviour of an application. LAPSE+ has been developed as an Eclipse plugin. ('OWASP LAPSE project', 2016)

A combination of two or more of the above tools may be the best approach to formulating a comprehensive static analysis of the component's source code. Jenkins provides an **Analysis Collector Plugin** which can collect the different analysis results of each of the above tools and aggregate them into a single report. (Hafner, 2012)

Gradle Build:

IBM developers utilise Gradle build-scripts to build software components. When Gradle build-scripts are run, dependencies for these projects are specified within the build-scripts. These dependencies are usually third party, open source software that will allow the project to access and execute specific functionality provided by the dependencies. However, because the dependencies are from third party libraries, and open source, they may have previously unaccounted for security vulnerabilities which will in turn leave the project, and Docker image using them, vulnerable.

Existing plugins which will check the dependencies of projects for security vulnerabilities:

1. **OWASP Dependency Check:** OWASP Dependency Check is an open-source utility which identifies project dependencies with known vulnerabilities. OWASP Dependency Check will cross-reference against vulnerability databases like National Vulnerability Database (National Vulnerability Database', 2016) or Common Vulnerabilities and Exposure (Common vulnerabilities and exposures', 2016) for up to date associated security vulnerabilities. It will initially download the database but on subsequent running of the program it only requires a small XML file to keep the database current. Once vulnerabilities are identified a report will be automatically generated. Plug-ins for Jenkins and Gradle are provided. ('OWASP dependency check', 2016)
2. **SourceClear:** SourceClear is an open-source security analysis integration. SourceClear provides a set of tools which tell developers what open-source dependencies they are using, who created it, what it is doing in their applications and which components have vulnerabilities. SourceClear supports Java and can be used in continuous integration systems like Jenkins with the plug-in provided. ('SourceClear',2016)

Docker Container Image:

Once the Gradle build script has successfully built the component, a Docker container is created to host the component. The Docker container will wrap the component in a complete file system which contains the environment needed to run the software. This guarantees that the software will always run the same regardless of the system it is running on. These Docker containers are then stored in repositories so they can be deployed, when needed, in an application hosted on the cloud.

1. **CoreOS/Clair:** Clair is an open-source project for the static analysis of container vulnerabilities that offers a transparent view of the security of the container-based infrastructure. Clair is designed to perform static analysis on Docker containers. This means that containers do not need to be executed, instead the file system of the container is inspected. Clair then cross-checks if the Docker image's OS or any of its installed packages match any known security vulnerabilities. Clair can be integrated into a continuous-integration pipeline. (coreos, 2016) CoreOS/Clair will cross-reference against the Common Vulnerabilities and Exposure for up to date associated security vulnerabilities.

At this phase of the project, integrating some or all of these various tools into the current Jenkins pipeline is considered to best plan of attack. It would mean that scanning for all of the above security vulnerabilities would be integrated into the "Analysis" stage of the pipeline and an aggregated report could be created and made available to the developer before the Docker image is executed. This would mean that the developer would be aware of any flagged vulnerabilities before the Docker image is released to its repository.

Phase 2: This phase will concentrate on the potential future vulnerabilities. However, there are no current tools or utilities that can scan the component for potential, future security vulnerabilities. The project may use IBM Watson during this phase. ('IBM Watson', 2016) Watson is a question answering cognitive-computing system. It is proposed that source code or component dependencies could be used as input to Watson. Watson would then scan relevant internet forums and blogs to help identify potential vulnerabilities that are currently being discussed on these forums or in these blogs. These potential vulnerabilities could then be identified and flagged to make the developer aware of them.

The commencement of Phase 2 will depend entirely on the completion of Phase 1. It is felt that completing Phase 1 would be more beneficial to IBM in the time available.

USERS

The main users of the security vulnerabilities application will be IBM's application developers and testers.

PLATFORMS, TECHNOLOGIES AND LIBRARIES

In addition to the above mentioned tools and utilities, this project may use any or all of the following platforms, technologies and libraries:

Gradle: Gradle is a build tool and dependency manager for programming projects. Build scripts are written, in a domain specific language based on Groovy, which tell Gradle how to build an application. Gradle creates a project object for each "build.gradle" script in the application. These objects allow the developer to access Gradle features.

The Gradle features fall into three categories: Properties, Tasks and Dependencies.

Properties consists of two property types, System Properties (configure the JVM that supports the execution of the build) and Project Properties (parameterise variables in the project).

Tasks are discrete pieces of work which tell Gradle how to build the project. Some tasks may have dependencies on other tasks, so the order in which they are executed is also controlled by the build script.

The final piece of the Gradle puzzle is the dependency. Dependencies are pieces of software, stored in repositories, that a project relies on to execute properly. Dependencies for the project

and the repositories in which they are stored are specified in the build scripts to be managed by Gradle. This project's focus will be on these dependencies and their security vulnerabilities.

Apache Groovy: Groovy is an object-oriented programming language for the Java platform. Gradle build-scripts, plugins etc. are written in Groovy. This is the main language the project will be written in.

External Vulnerability Databases: This project will use external vulnerability databases to identify the current vulnerabilities of third party software which the targeted build is dependent on.

GitHub: Web based Git repository hosting service. It offers distributed version control and source code management. All code written for the project will be stored on GitHub.

Jenkins: Jenkins is an open source automation server, written in Java, that provides continuous integration and delivery of software projects.

RISKS

The main risk involved in this project is creating a robust dependency security vulnerability identification process. Because the dependencies of the target build are predominantly third party open-source software located in third-party libraries and repositories, the opportunity for unidentified security issues to arise is great. The system will only be as reliable as the vulnerability databases that are accessed to identify the vulnerabilities. If a new security issue associated with a dependency arises and is not identified by one of these resources, then the target build could be left exposed to attack. Therefore, it is critical that as many resources as possible are identified and used for the security vulnerability checks. This may have a considerable overhead which will only be quantified more clearly once the project gets underway.

REFERENCES & BIBLIOGRAPHY

'Jenkins automation server'. Available at: https://jenkins.io/ (Accessed: 26 October 2016).

'Gradle build tool I modern open source build automation'. Available at: https://gradle.org/ (Accessed: 26 October 2016).

'Docker'. Available at: https://www.docker.com/ (Accessed: 26 October 2016).

'FindBugs - find bugs in java programs'. Available at: http://findbugs.sourceforge.net/ (Accessed: 26 October 2016).

'PMD'. Available at: https://pmd.github.io/ (Accessed: 26 October 2016).

'OWASP LAPSE project'. Available at: https://www.owasp.org/index.php/OWASP_LAPSE_Project (Accessed: 26 October 2016).

Hafner (2012) ‘Analysis collector Plugin’. Available at: https://wiki.jenkins-ci.org/display/JENKINS/Analysis+Collector+Plugin (Accessed: 26 October 2016).

‘National Vulnerability Database’. Available at: https://web.nvd.nist.gov/view/vuln/search (Accessed: 12 October 2016).

‘Common vulnerabilities and exposures’ (CVE). Available at: https://cve.mitre.org/ (Accessed: 12 October 2016).

OWASP dependency check (no date) Available at: https://www.owasp.org/index.php/OWASP_Dependency_Check (Accessed: 26 October 2016).

‘SourceClear’. Available at: https://www.sourceclear.com/registry/explore (Accessed: 12 October 2016).

coreos (2016) ‘Coreos/clair’. Available at: https://github.com/coreos/clair (Accessed: 26 October 2016).

‘IBM Watson’. Available at: http://www.ibm.com/watson/ (Accessed: 26 October 2016).