Docker Security Vulnerabilities – Gradle Plugin
Paul Kenny | X00107570

USE CASES

Use Case:

Title: Run Docker Scan

Primary Actor: Developer

Scope: Linux OS, Docker Image

Level: Primary

1. The developer initiates the Jenkins pipeline to deploy a new component.

2. The Jenkins pipeline builds, tests and analyses the component.

3. The Jenkins pipeline also builds a new Docker image containing the developer's code changes.

4. The developer will execute the Gradle Security Vulnerabilities plugin, from the command line, to scan the Docker image for known Java platform security vulnerabilities.

5. Any security vulnerabilities found are relayed to the developer through a security vulnerabilities report in browser interface.

When a developer is finished work on a new component a Jenkins pipeline is initialised. The pipeline will build the component using the component's Gradle build files. The component is then tested and analysed. The next step in the pipeline is to build a Docker image with the changes for deployment. Once the Docker image is complete the Gradle Security Vulnerabilities plugin will be launched by the developer through the command line interface. The plugin will scan the Docker image and check against a security vulnerabilities and exposures database for known security vulnerabilities. A report will be compiled by the plugin which will automatically launch in the developer's default browser, and be accessible thereafter, to the developer in the plugin's Reports directory.

TECHNICAL ARCHITECTURE

Operating System:

Linux OS:

The project will be developed in a Linux environment using a CentOS (Red Hat) virtual machine for two specific reasons. Firstly, the Docker engine operates natively on a Linux OS. Secondly, the plugin will be executed by IBM in a Linux environment to scan the related Docker images.

CentOS 7:

CentOS is an open-source Linux based (Red Hat Enterprise Linux) operating system which will be deployed by this project as a virtual machine. All development and testing of this project will be executed on the CentOS virtual machine. (CentOS, 2017)

Software Components:

Git:

Git is an open source distributed version control system. It records changes to files over time. Specific versions can be recalled at any time and compared to previous or subsequent versions. Git will be used specifically in this project to track development changes throughout the development process.

HTML:

Hypertext Mark-up Language is the standard mark-up language for creating web pages. HTML will be used to create a web interface which will display the results of Docker image vulnerability scans.

Tape Archive Files:

A Tape Archive File (Tar/Tarball file) is a software utility to collect many files into one archive file. In this project the many file system layers that make up a Docker image will be pulled together into one tarball file using the tar software utility.

Java Archive Files:

A Java Archive file (JAR files) is a package file format used to aggregate Java class files and associated resources and meta data into one distributed application software. JAR files with known security vulnerabilities will be put into Docker images to be used in the development and testing stage of the Docker Security Vulnerabilities plugin.

Database:

In this project a database of known vulnerabilities and exposures will be built and deployed in the cloud. A MySQL database will be built on the Amazon Web Services, Relational Database Service. The database will contain tables of third party Jar files and their associated Common Vulnerabilities and Exposures (CVE) to be cross-referenced by the security vulnerabilities scan.

Oracle VM VirtualBox:

The Oracle VM VirtualBox is a cross-platform virtualisation software. In this project, it will enable the developer to deploy a CentOS (Red Hat) Linux based virtual machine which in turn enables the developer to run the Docker engine natively on their machine.

Platform Libraries and Frameworks:

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. In this project the Docker Security Vulnerabilities scanner will be built as a Gradle plugin. This will enable the developer to compile and archive (Jar) the plugin, pre-distribution. The distributed plugin will contain a compiled jar file and a separate "User" directory. From within this directory the plugin user can execute the security vulnerabilities scan on a chosen Docker image on the system.

Groovy:

Groovy is an object-oriented programming language for the Java platform. The Docker Security Vulnerabilities plugin will be written in Groovy.

Structured Query Language (SQL):

SQL is a domain-specific language used to add, update and delete data to and from relational database. In this project, it will be used to cross-reference jar files, found by the Docker image scan, with the security vulnerabilities database.

Docker:

Docker is an open-source project that automates the deployment of applications inside software containers. These containers wrap the application in a file system that contains everything needed to run the application consistently, regardless of the environment it will be running in. In this project, sample applications will be added to Docker images for deployment. The Docker images will be scanned by the Docker Security Vulnerabilities scanner to check for known security vulnerabilities.

Cobertura:

Cobertura is a Java based tool which is executed to calculate the percentage of source code access during the test phase of the project.

Apache Commons Compress:

Apache Commons Compress is a library that defines an API for working with archive file of all types. In this project, it will be utilised in the inspection of tarball and jar files (contained within the Docker image), during the scanning process, to identify and list internal jar files found.

Maven Local/Central Repository:

Maven Local and Central repositories are locations where third party project dependencies are stored. When Gradle builds the Docker Security Vulnerabilities plugin, it will access these repositories and get the project dependencies required by the project at test, compile and run time.

Connector/J:

Connector/J is the JDBC driver required by Java/Groovy projects to connect the project application to a given local or remote database. In this project, it will be utilised to connect the Docker Security Vulnerabilities plugin to the cloud based AWS RDS MySQL vulnerabilities database.

Spock:

Spock is a testing and specification framework for Java and Groovy applications. In this project, it will be used to test the plugin's source code.

Other Resources:

National Vulnerabilities Database (NVD):

The NVD is a database of known software vulnerabilities produced by the U.S. National Institute of Standards and Technology and is sponsored by the U.S. Department of Homeland Security. It is a repository of standards based vulnerability management data. The NVD includes security checklists, security related software flaws, misconfigurations, product names, and impact metrics. (NVD, 2009)

Common Vulnerabilities & Exposures List (CVE):

CVE is a list of common names of publicly known security issues. The CVE list contains unique identifiers of security vulnerabilities and exposures. These identifiers enable the user to obtain information from a variety of CVE-Compatible information sources. CVE is sponsored by US-CERT, in the office of Cybersecurity and Communications at the U.S. Department of Homeland Security. (US-CERT, 2016) US-CERT advocates the use of CVE to the U.S. Government.

In this project, the Docker Security Vulnerabilities plugin will access a database of known vulnerabilities. This database will contain information collected from the NVD and CVE list.

Development and Production Distributed Architecture:

For the development and production stages of the project a distributed architecture will be used.

Developer:

The project developer will write and test the application code locally, on a CentOS VM, on their personal workstation. During the prototyping phase of the project, a sample application (containing JAR files with known vulnerabilities) will be added to a base Docker image. This image will be scanned by the prototype application for known security vulnerabilities. During the scanning process the application will cross-check with the AWS RDS MySQL cloud based vulnerability database to check if a scanned Docker image contains any matching vulnerabilities. Once the scanning process is finished, scan findings will be reported to the developer through a web interface.

All application code will be pushed to IBM's GitHub repository. The project developer's mentor at IBM will be able to access the application source code via GitHub, to assess progress and aid in development. This will facilitate continuous collaboration between the developer and mentor. (figure 1)
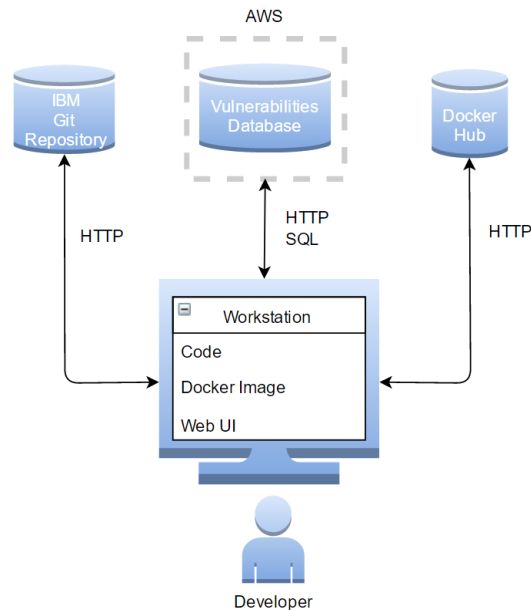

Figure 1: Project distributed architecture

User:

The Docker Security Vulnerabilities plugin user will clone the plugin directory from IBM's GitHub repository. The directory will contain two further directories. The first directory will contain the pre-compiled Docker Security Vulnerabilities, which will contain all the necessary functionality to run the vulnerabilities scan on a chosen Docker image. The second directory, the "user" directory will contain directories to store temporary directories, created and deleted during the image scan, and to store vulnerabilities reports generated by the scan. For this second "user" directory the user can execute the scan (through the command line interface) on a chosen Docker image stored on their local system.

The plugin will scan the image, identify jar files contained by the image and cross-reference theses jar files with the cloud based vulnerabilities database. Once the scan is complete, a html report will be automatically launched on the users default browser.

Vulnerabilities Database:

The project will be utilising the NVD and CVE as information resources to build a sample vulnerabilities database. The Docker Security Vulnerabilities plugin will cross-reference jar files, found in a chosen Docker image, with known security vulnerabilities associated with these jar files found in the security vulnerabilities database.

The new vulnerabilities database will be a relational database built as an AWS RDS MySQL database instance. This database will contain information on selected third party jar files and the security vulnerabilities associated with them (see database schema below). The plugin will utilise a JDBC driver to connect to the data base instance and use SQL queries to retrieve the necessary data from the database.

Docker Hub:

Docker Hub is a Docker image repository stored in the cloud. Images can be pushed to and pulled from the repository. This project will use Docker Hub to store Docker images which will be used in the prototype testing phase of the development process.

Virtual Machine (VM):

A VM is an emulated computer system. Based on physical computer architecture, it provides all the same functionality as a physical system. In this project a VM will be used to facilitate the development process natively in the Linux OS.

Git Repository:

As stated above, Git will be used as the version control system to manage this project. The Git Repository is the location where all versions of the code in development will be stored, and then subsequently cloned from by the plugin user. This project will use IBM's Git repository.

Workstation:

For this project the primary workstation will be the laptop provided to the developer from IBM. The laptop will communicate with IBM through a VPN provided by IBM.

Security Certificates, Authentication:

Both the developer and IBM mentor will share access to the GitHub repository and the AWS RDS MySQL database instance. Access to IBM's GitHub repository is secured through the developer and mentor's IBM ID. Access to the AWS RDS MySQL database instance is secured through the AWS security credentials.

Database & Schema:

The AWS RDS MySQL database:

Hostname: jar-vul.crxuc0o6w3aw.us-west-2.rds.amazonaws.com

Port: 3306

Username: Paul

Password: paulk990099

Jar Table: Table of vulnerable third party jar files.

Jar Table	
JAR_NAME	The name of the third-party jar as found in the Maven Central repository.
JAR_DESC	A description of the jar and its functionality.

Table 1: Jar Table

CVE Table: Table of Common Vulnerabilities & Exposures associated with the jar files.

| CVE Table	
CVE_ID	CVE identifiers are unique, common identifiers for publicly known information-security vulnerabilities in publicly released software packages.
JAR_NAME	The name of the third-party jar as found in the Maven Central repository.
CVE_DESC	A description of what the vulnerability is and how it can be used to exploit a system.
CVSS_SCORE	Common Vulnerability Scoring System Score is used to determine the potential severity of the vulnerability.
CVSS_FLAG	Another metric to determine the potential severity of a vulnerability. Low, Medium or High severity.
ACCESS_VECTOR	Shows how a vulnerability may be exploited e.g. locally, network etc.
AUTH	Authentication. Shows whether authentication is required to exploit a vulnerability.
IMPACT_TYPE	What type of impact on the system the vulnerability could lead to e.g. unauthorised disclosure of information.
VUL_TYPE	What type of vulnerability it is e.g. a credentials management vulnerability is related to how credentials are managed by the system.
CWE_ID	Unique identifier for different vulnerability types.
CWE_LINK	URL providing a link to more information about the type of vulnerability.
NVD_LINK	URL providing a link to the NVD to more information about the vulnerability.

Table 2: CVE Table

Vulnerabilities Database Schema:
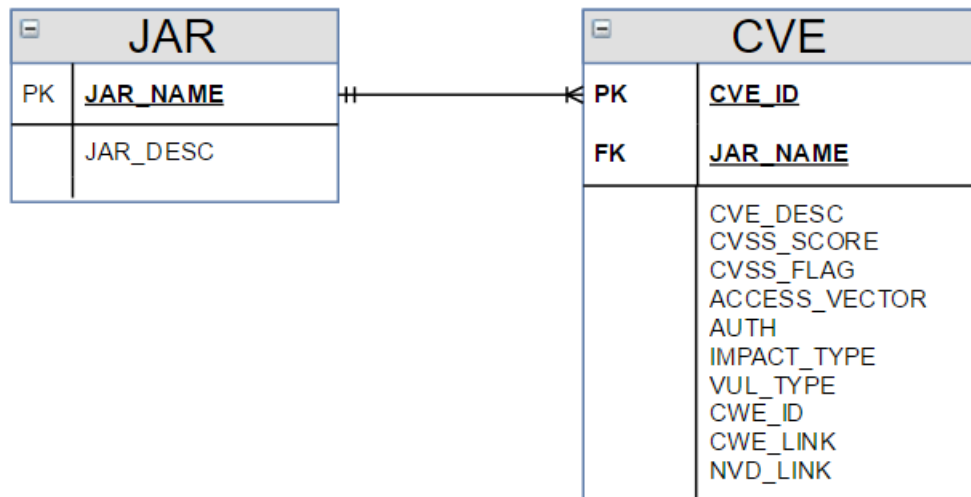
Figure 2: Security Vulnerabilities Database Schema

Overview:

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. With this project, it has facilitated the writing of the Docker image scan in a collection of Groovy classes. Creating the scan application as a plugin means that the application can be referred to from other projects, for example a Jenkins pipeline or simply in the command line interface. The plugin has a very particular structure which is expanded upon below.

Plugin File Structure:
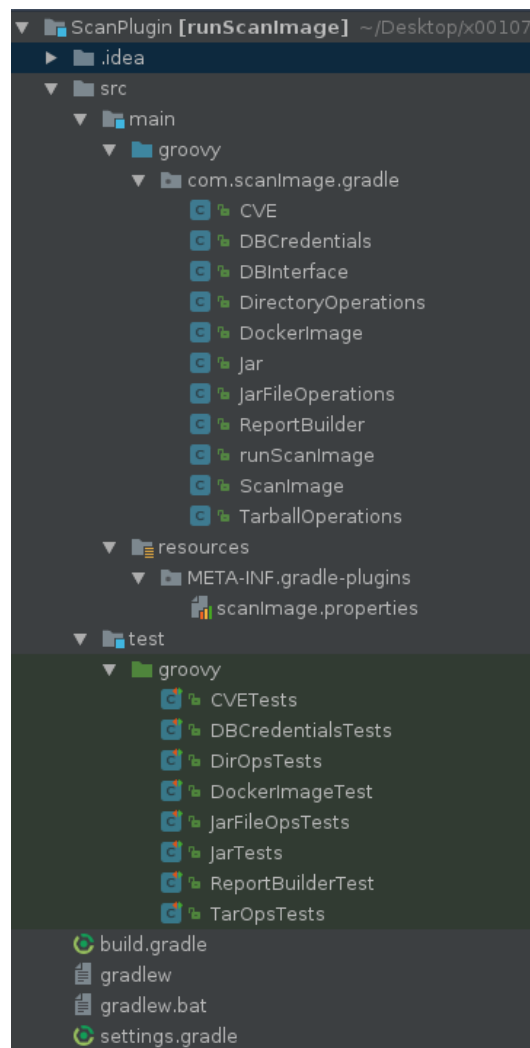
Developer Side:

Figure 3: Docker Image Security Vulnerabilities Scan File Structure

Gradle Build Script:

The src/build.gradle script defines how the application will be built. This script defines the group and version of the plugin that should be referred to by any application which later applies the plugin. It also applies the plugins that this plugin requires to build and calls all dependencies (and where to find them) that the plugin requires at compile and run time. It will also create a local Maven repository which will store the JAR file to be used by the consumer.

Gradle Settings Script:

The src/settings.gradle script defines the name of the project.

Plugin Properties Script:

The src/main/resources/META-INF/gradle-plugins/scanImage.properties script defines the entry point, or implementation class, of the project (in this instance the runScanImage.groovy class). This class is explained in more detail below.

The Implementation Class:

The src/main/groovy/com/scanImage/gradle/runScanImage.groovy class is the plugin implementation class. This is the plugin entry point. In this class the central task of the plugin, i.e. to kickoff the Docker image scan, is defined. This class will take in the image name parameter as an argument from the command line interface and pass it into the src/main/groovy/com/scanImage/gradle/scanDockerImage.groovy class which will execute the Docker Image scan.

On completion of the project the developer builds the project and uploads archives, containing project dependencies. The resulting JAR file is then ready to be utilised by a consumer.
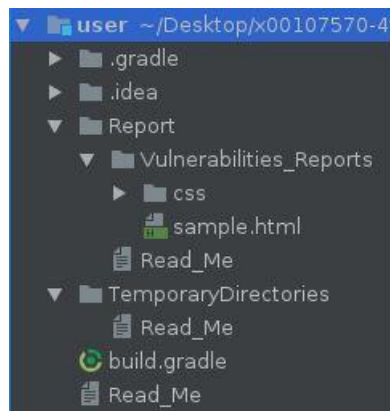
Consumer Side:

Figure 4: Docker Image Security Vulnerabilities Scan User Directory Structure

Once the Docker Image Security Vulnerabilities Plugin is complete, compiled and archived on the developer side it is then ready to be utilised by a consumer. IBM will store the plugin in a repository to make it available to its developers. On download of the plugin, the plugin will contain two separate directories. One will be the ScanPlugin directory (Fig 3), which will contain the compiled plugin in the form of a JAR file, the second will be the consumer, or "user" directory. The contents and usage are explained below.

Gradle Build Script:

> The consumer side build.gradle script will perform two tasks. Firstly, it will access the local Maven repository which stores the Docker Image Security Vulnerabilities plugin in its JAR archived form. Secondly, it will apply the plugin to the current user build.

The consumer will navigate to the user directory and execute the Docker image scan by passing the name of a Docker image into statement as an argument:

> > gradle scan -PimageNameArg="<docker image name>"

The plugin will then commence its scan of the Docker image.

Temporary Directories:

> During the scanning process, the image scan must create (and then eventually remove) temporary directories necessary to the scanning process. It will create these directories in the user/TemporaryDirectories directory.

Security Vulnerabilities Report:

> Once the scan is complete, a HTML report will be generated for the user and opened automatically in their default browser. This report will be made available to the user in the Report/Vulnerabilities_Reports directory.

13

Risks:

The main long-term risk involved in this project is creating a robust security vulnerability identification process. The system will only be as reliable as the vulnerability databases that are accessed to identify the vulnerabilities. If a new security issue arises and is not identified by the primary resource, then the target Docker image could be left exposed to attack.

During the development phase of this project the database used will be the AWS RDS MySQL database instance containing tables of sample information collected from the NVD and CVE list. This is because the NVD and CVE list do not provide any way to access their resources programmatically, e.g. an API etc. It is felt that this is not a long-term solution for IBM.

The NVD, however, provides an RSS XML bulk feed of stored known security vulnerabilities and to this effect, it is suggested that IBM consider a future project to create a comprehensive vulnerabilities database from this NVD RSS feed. This falls outside the scope of this project.

Prototype:

The first prototype will consist of a simple Gradle plugin which will be able to scan a Docker image and identify simple files.

References:

US-CERT (2016) United States computer emergency readiness team. Available at: https://www.us-cert.gov/ (Accessed: 15 November 2016).

NVD (2009) National Vulnerability Database. Available at: https://www.nist.gov/programs-projects/national-vulnerability-database-nvd. (Accessed 18 April 2017)