UCF - CECS

Homework Assignment 4
Building Secure Java

November 5, 2023

1 Objectives

The goal of this assignment is to build a test harness to scan Java applications for well known security vulnerabilities. Several tools are used to accomplish this. This assignment is divided into two parts - the first is to install the necessary software to scan for code and policy violations in Java code. The second is to use the tools to check the code using those tools. These parts and the commands necessary to build and use these tools are outlined below.

1.1 Required OS

All the software used for this assignment will be installed on some version of a **Debian Derived Linux**[1], while the required software is configurable for Windows 10 and Mac OSX, it will be easier for all parties concerned to run on a Debian flavored **Linux**.[2] **W10's** virtualization software (*Hyper-V*) runs only on Windows 10 Pro, Enterprise, and Education versions. For detailed installation instructions for *Hyper-V* see the Hyper-V installation web page. Other virtualization hosts are available for Mac OSX, Red Hat, and other linuxes. All testing for these scripts were run using QEMU which is available for W10, Mac OSX, and other linuxes. The installation of the latest version of **Linux Mint 21 "Vanessa"** is available at the Mint 21 download page and all the *install* scripts were built and tested on **Mint 21** and **Debian 11**.

[1]Debian derived linuxes include Ubuntu, Mint, Knoppix, Xubuntu and ~120 others.
[2]This will streamline the assignment, as the subtleties of running on other Linux distros will be eliminated.

1.2 GitHub

The *GitHub* account used in a previous assignment is required for an account with *Snyk*. The *GitHub* account will also contain the input code repository to be scanned by *Snyk* for security defects. Also note that the repository must match either the *Gradle* or *Maven* build process. This assignment will use the *Maven* build suite, as explained below.

1.3 Software inventory

1.3.1 Maven

"Maven", a Yiddish word meaning *accumulator of knowledge*, began as an attempt to simplify the build processes in the Jakarta Turbine project. There were several projects, each with their own Ant build files, that were all slightly different. JARs were checked into CVS. We wanted a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects.
The result is a tool that can now be used for building and managing any Java-based project. We hope that we have created something that will make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project."[3]

Maven is an attempt to apply patterns to a project's build infrastructure in order to promote comprehension and productivity by providing a clear path in the use of best practices. Maven is essentially a project management and comprehension tool and as such provides a way to help with managing:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution

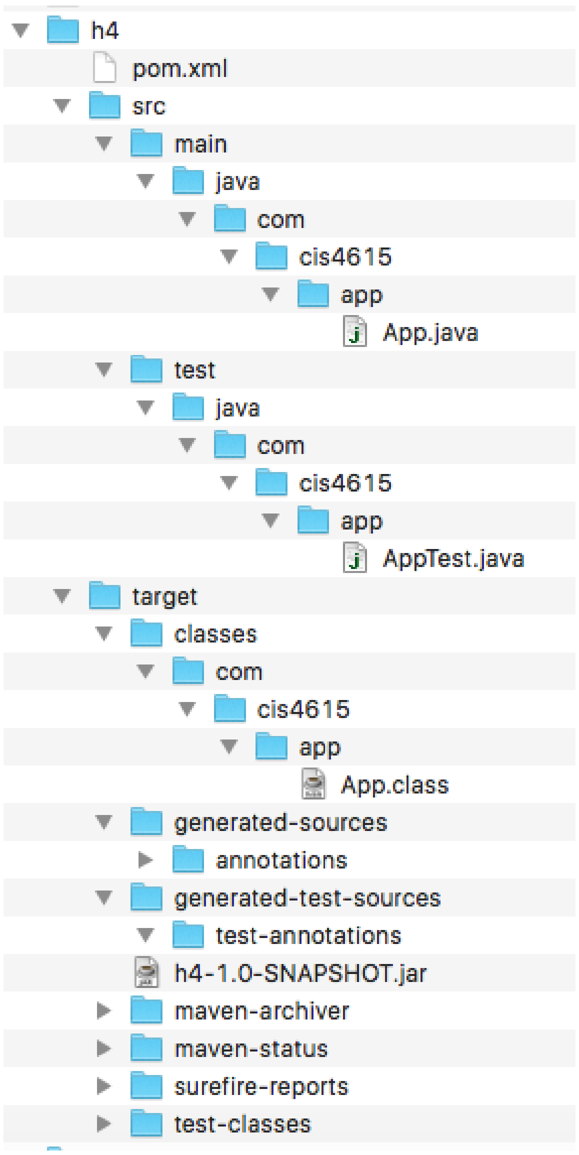
Maven will be used to *generate* the files (typically *pom.xml*) used by the *Snyk* service to validate the integrity and security of a chosen Java application and its components.

[3]Maven Intro

1.3.1.1 Installing Maven

Downloads for your favorite OS are at the Maven Download Page. Note that the download page will point to another page for the Debian derived linux installations. In the event that the Linux used to complete this assignment is Debian 10 or Mint 21, a simple `sudo apt install maven` command will install maven, if you are *not using* the install scripts below. Refer to the ***Install Notes*** later in the document.

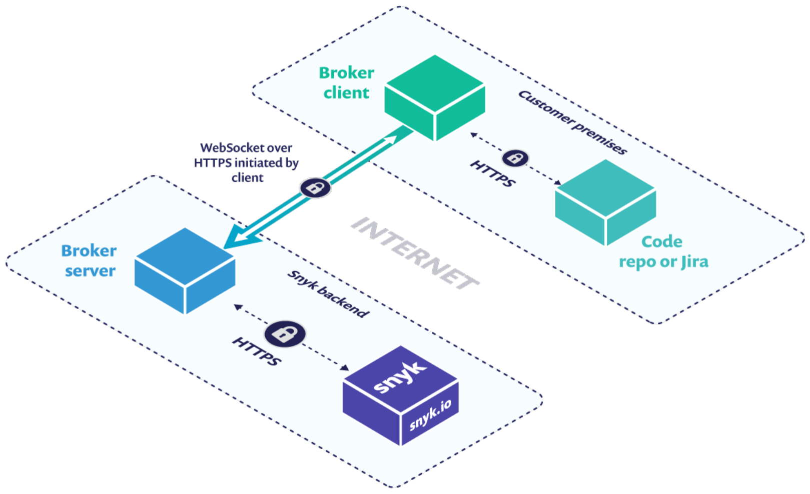
1.3.1.2 Sample project build results There are shell scripts suitable for *BASH* on Linux. They are shown below. The shell script command:`./mvnBuildProj.sh cis4615 h4` produces the following directory structure used by *Maven*. It is important to note that the configured *Maven* commands also include the *App.java* file which contains a simple `Hello, World!` application. *Maven* shell scripts included in this assignment are, subsequently, configured to use those same resources.



```
▼ h4
    pom.xml
  ▼ src
    ▼ main
      ▼ java
        ▼ com
          ▼ cis4615
            ▼ app
                App.java
    ▼ test
      ▼ java
        ▼ com
          ▼ cis4615
            ▼ app
                AppTest.java
  ▼ target
    ▼ classes
      ▼ com
        ▼ cis4615
          ▼ app
              App.class
    ▼ generated-sources
      ▶ annotations
    ▼ generated-test-sources
      ▼ test-annotations
      h4-1.0-SNAPSHOT.jar
    ▶ maven-archiver
    ▶ maven-status
    ▶ surefire-reports
    ▶ test-classes
```

1.3.2 Snyk

Snyk is a cloud based service to scan source code files for well known code security deficiencies. *Snyk* uses *Maven's* `pom.xml` config file for determining which files, languages, and resources to be scanned. It uses *NodeJS* on the local machine to run the *Snyk* client. This service architecture is shown below.



- The first step is to install *NodeJS* using the information available at the NodeJS download page.
- Then update *NodeJS* using the command:
  `npm update`
- Then install *Snyk* using the command:
  `sudo npm i -g snyk`
- Followed by the command:
  `snyk update`

1.3.2.1 Using Snyk CLI

Instructions and examples for using *Snyk* in the *CLI* are available at Snyk CLI Installtion.

2 Install Notes

Given that this assignment can be completed using either *Linux*, *W10* or *Mac OSX* as the ***host*** for any of the *virtualization* tools mentioned earlier, the minimal host configuration suggestions for the ***guest*** virtual machine is derived from the VirtualBox parameters below.

2.1 Using VirtualBox for Mint on Windows, Mac OSX, or linuxes

VirtualBox is an open source virtual machine application that enables running a *guest* OS as if it were on dedicated hardware. As a result there are some configuration elements that are likely to cause problems - *if not properly configured.* Some guidance is provided below.

- **Main Memory** should be configured above *1 GB* or 1024 MB. If you *host* machine has more than 4 GB, it is meaningful to have more in the *guest* machine.
- **Disk size** for the guest's virtual disk defaults to *8 GB* but 20 GB will minimize any sizing issues.
  - *VirtualBox* will ask if it acceptable to ***erase the disk*** during the OS installation process. Note that the disk it will erase is 20 GB virtual disk discussed above.
- **Video Memory** should be configured for a *minimum* of **64 MB**. Failure to do *this will cause video rendering problems* in the guest OS.
- **OS Updates** should be installed after the installation has completed. This can be accomplished from the command line by executing the following two commands in the following order: `sudo apt update` then `sudo apt upgrade`

2.2 Generic Debian/Mint install from CLI

Mint 21 was chosen for a more effective and streamlined installation and usage. *Other linux distributions are equally valid and robust, but require, typically, a bit more **sysadmin** expertise.* The following programs are required to successfully run *snyk*. *In the event you are confident using the CLI and are comfortable with the intricacies of the instructions below, **charge on**.* On the other hand several admin scripts are in the section **Install Notes** which will streamline the install processes to a matter of minutes. This is described in explicit detail in the **Install Notes**.

maven The previously discussed open source software configuration and build suite.

- *Maven* can be installed with the following command:
  `sudo apt install maven`
- Additional installation instructions for the latest version are available at Maven installation instructions.

nodeJS The Node Javascript engine that supports command line and software development using JavaScript libraries. It also includes the *Node Package Manager* more commonly referred to as ***npm***.

- Install *Node* using the following commands.
  - Install **curl** using the command:
    `sudo apt install curl`.

  - Enter the command:
    `sudo apt install nodejs`. (*This will install the latest packaged version of* ***NodeJS***.
  - Also, refer to the install script named `installCurlNJS_snyk.sh` for tested commands.
- Successful *NodeJS* and the associated *npm* installations can be verified using the following commands.
  - `node -v`
  - `npm -v`
- This will enable authorization of the *snyk* account for the *snyk* software. It can be installed by entering the the commands as follows.
  - Login to the *GitHub* account created for HW02. *Snyk will use the `GitHub` credentials to authorize the `snyk` account.*
  - Enter the command: `sudo npm i -g snyk`
  - Enter the command: `sudo chown $USER -R ~/.config`
  - Enter the command: `snyk auth`

snyk the tool suite that scans and tests software libraries based on their *Maven* configurations. Note that *snyk* can also scan software based on other project configurations tools such as *Gradle*. *Maven* will be used for this assignment. *snyk* can be installed using the following command.

- `npm install -g snyk`

See the link snyk's CLI installation for more information about running *snyk* from the command line.

2.3 Mint Install Scripts' Notes

There are three shell scripts to install *Maven*, *NodeJS*, and *Snyk* on *Mint*. These install scripts are in the ZIP file, **HW04-bash.zip** in the directory named **Mint-install**. Move these commands to your home directory (`mv Mint-install/*.sh ~`), then make them executable with the command `chmod +x ~/*.sh`). (Remember to change to the $HOME_DIRECTORY using the command `cd ~`) Before trying to run the `install` scripts, execute the command `chmod +x *.sh` The commands should be invoked in the order shown below from the user's home directory as follows:

- `./installJava.sh`
- `./installMaven.sh`
- `./installCurlNJS_snyk.sh`

- Make sure to login to your *GitHub* account from HW 2, then enter the command: `snyk auth |tee -a filename.log`
- Once `snyk` has been authorized, then remember to run the command `snyk test|tee -a filename.log` where filename.log is appropriate to the project being scanned. In this assignment it could be: *helloWorld.log*, *BadJavaHW2.log*, or *java-goof.log*. (This will cause *snyk* to run the scan through the cloud on your local machine, *as long as the command is run in the directory that contains the project's **pom.xml** file.*)

It should be noted that the user's home directory will be the default directory when the `terminal` application opens a terminal window for the *CLI*. Also to be noted that these commands should be run in that order and will output the results to the console **and** the *install.log* plain text file. *The log files will be part of your assignment submission.*

There are two choices, either install the software using the commands shown above or use the *Install Scripts* shown in the preceding paragraph using the shell scripts itemized above.

3 Testing

Given the tool chain used for this assignment, testing can be done upon completion of the installation processes described later in the ***Installation Notes***. To wit, installing *Maven*, then *NodeJS*, then *Snyk* will be successful the end of each installation.

3.1 Simple end to end test

1. The command: `sysPrompt$./mvnBuildProj.sh cis4615 h4 |tee helloWorld.log` will configure a *Maven* directory and build the necessary code for a simple *"Hello, World!"* Java program.
2. It will also run the java program and display it's output.

Refer to the log files in the appendix for other validation data.

3.2 Complex end to end test

- ~~Simple code modification:~~
  1. ~~Edit the `App.java` file to reflect the code violation from HW 2's NUM03-J. Use integer types that can fully represent the possible range of unsigned data. Make sure to have a meaningful output message sent to `STDOUT` so that the log file reflects the particular source code weakness.~~
  2. ~~Use the command `sysPrompt$./mvnCompileTest.sh cis4615 h4 |tee badJavaHW2.log` to compile, build, and execute the `App.java` program. The use of the~~

~~`tee filename.log` will output everything from `STDOUT` to the file named in the command, in this example, `filename.log`.~~

3. ~~Use the command `snyk test|tee -a filename.log` to validate the project defined in the `pom.xml` file. (Note the **-a** option in the `tee` command - this will append the snyk output to the log file.)~~

- ~~Use *GitHub* to clone the Snyk Java-goof project on your local machine.~~
  1. ~~Make sure the current directory contains the `pom.xml` file. Use the command `snyk test|tee -a java-goof.log` to validate the project defined in the `pom.xml` file.~~

3.3 Complex end to end test (UPDATED)

The objectives is to clone the following git project **java-reachability-playground**, configure and run **maven**, then confirm the security or vulnerability of the code based on using the **snyk** cloud based services to validate and identify any known weaknesses or vulnerabilities. This can be accomplished using the commands outlined below. *These steps assume you've successfully installed **git**, **maven**, and **snyk**, and are logged into your **git** account. Also note that you should be running the following commands from a terminal on the virtual machine installed previously.*

- `mkdir ~/cis4615-work`
- `cd ~/cis4615-work`
- `git init`
- `git clone https://github.com/snyk/java-reachability-playground`
  `Clone the` *`java-reachability-playground`* `repository for snyk testing.`
- `cd .git/java*` *Should now be in the java-reachability-playground directory.*
- `mvn install|tee mvn-JRPlayground-install.log` *Should now configure and log the maven install for this project.* There is substantial information output to the terminal for this and the following commands. These outputs are logged.
- `mvn compile|tee mvn-JRPlayground-compile.log` *Should now configure and log the maven compile for this project.*
- `mvn exec:java -Dexec.mainClass=Unzipper|tee mvn-JRPlayground-exec.log` *Should now execute the program, process the inputs and outputs, and validate/detect any failures for this project.*

  - —-extensive maven output to include following detection—–
    [ERROR] Failed to execute goal org.codehaus.mojo:exec-maven-plugin:3.0.0:java (default-cli) on project java-reachable-goof: An exception occured while executing the Java class.
    Malicious file /tmp/evil.txt was created -> [Help 1]

- `snyk test -reachable |tee snyk-test.log`
  - A maven advisory for java-reachability-playground will spin for a short while. *Short being a relative term. It took almost 5 minutes on my VMs.*
  - the output will be as follows:
    Feature flag 'reachableVulns' is not currently enabled for your org, to enable please contact snyk support
  - This is *snyk's* polite way of saying this account is not configured for or paying for Java language support.

- `snyk monitor -reachable|tee snyk-monitor.log`
  - A progress advisory will also spin here, also for about 4 minutes.
  - the output will be as follows:
    Server returned unexpected error for the monitor request. Status code: 405

Submit the log files from each step identified above as part of your assignment.

4 Submission instructions

You must submit this assignment in **Webcourses** as file uploads. Note that all submissions will be via **Webcourses**.
Be sure to include the following log files:

1. install.log
   - The *install shell scripts* append their output to the `install.log` file located in the directory where the scripts were run.
2. helloWorld.log
   - `sysPrompt$./mvnBuildProj.sh cis4615 h4 |tee helloWorld.log`

The logfile check list is as follows:

1. mvn-JRPlayground-install.log
2. mvn-JRPlayground-compile.log
3. mvn-JRPlayground-exec.log

4. snyk-test.log
5. snyk-monitor.log

ZIP the log files above into your **Webcourses** submission.

5 Grading

Scoring will be based on the following rubric:

Table 5.1: Grading Rubric

Deduction	Description
-100	***helloWorld.log*** output for *baseline* & **default installation logs** not submitted.
- 75	*Maven* not installed or configured.
- 75	Cannot compile programs using *Maven*.
- 75	*Node* not installed or configured.
- 75	*Snyk* not installed or configured.
- 25	Missing log files - each file mvn-JRPlayground-install.log mvn-JRPlayground-compile.log mvn-JRPlayground-exec.log snyk-test.log snyk-monitor.log
- 15	Missing installation log files (each)
Start with 100 points and deduct per the schedule above	

6 Resources

1. The script `./mvnBuildProj.sh cis4615 h4`.

```
#! /bin/bash
# command usage example
#
# ./mvnBuildProj.sh cis4615 h4
#
# $1 is the "company" name in the example above it is cis4615
# $2 is the program name in the example above it is h4
#    MAVEN WILL CREATE ALL THE FILES IN THE DIRECTORY NAMED IN $2
#    THIS INCLUDES A SIMPLE JAVA "Hello, World!" PROGRAM!
#
# Derived from the "Maven in 5 minutes tutorial" available at:
# https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
#
# This shell script will create a "Hello, World!" app in Java
# using maven's configure & build tools.
```

```
#
# IT IS HIGHLY RECOMMENDED TO USE THIS SCRIPT IN AN EMPTY DIRECTORY
# CREATED JUST FOR THIS EXAMPLE AND ITS TESTING.
#
# NOTE:
#   There will be an extensive directory set built for the app
#   & testApp, roughly equivalent to all the directories created in Eclipse,
#   NetBeans and IntelliJ
#
mvn archetype:generate\
    -DgroupId=com.$1.app\
    -DartifactId=$2\
    -DarchetypeArtifactId=maven-archetype-quickstart\
    -DarchetypeVersion=1.4\
    -DinteractiveMode=false

cd $2

# Compile and build the Java app in $2
mvn package

java -cp target/$2-1.0-SNAPSHOT.jar com.$1.app.App
#
# See mvnExecute.sh for compiling and running the code
```

2. The shell script `./mvnCompileTest.sh cis4615 h4`.

```
#! /bin/bash
# command usage example
#
# ./mvnCompileTest.sh cis4615 h4
#
# $1 is the "company" name in the example above it is cis4615
# $2 is the program name in the example above it is h4
#
# Run this script in the app direcory created in
# the application’s root directory which in the
# was identified as "h4".
#
#
cd $2

mvn clean compile

cd target/classes

java com/$1/app/App

cd ../../..


# Derived from
# https://www.dineshonjava.com/maven-example-hello-world/
```