IDT
INNOVATIVE
DEFENSE TECHNOLOGIES

Winter 2017 Contest

Black-box Testing Framework Documentation

2016 December 15th

Introduction

IDT has developed a testing framework that will be used by developers and testers to test executable jars (Java). This document contains all of the documentation necessary for executing and extending the framework.

We would like to welcome your development team to the 2017 winter programming contest. We are excited to work with your team to extend our testing framework to expose security vulnerabilities or weaknesses in the software under test.

Contents

1. Framework and Supporting File Archives

IDT will transfer two .zip files to your team that will contain our framework codebase and supporting files.

com.idtus.contest.winter2017.framework.zip – This is a compressed archive of the framework codebase. You will need to unzip this archive on your development machines. **It does not matter where you ultimately put this codebase on your computer because you will be importing it into Eclipse and interacting with it though an IDE.** Expanding compressed framework should reveal the following files that are part of a Maven Project for Eclipse:

com.idtus.contest.winter2017.framework\.classpath

com.idtus.contest.winter2017.framework\.project

com.idtus.contest.winter2017.framework\pom.xml

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\Main.java

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\Output.java

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\Parameter.java

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\ParameterFactory.java

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\Test.java

com.idtus.contest.winter2017.framework\src\main\java\contest\winter2017\Tester.java

com.idtus.contest.winter2017.supportingfiles.zip – This is a compressed archive of the supporting files that you will need to execute the framework. **The archive contains a directory called 'idt_contest' that we recommend putting directly into your C:\ directory (if you are using windows) or your user's directory (if you are using linux).** You will ultimately be referencing multiple files/directories as arguments for the framework, so we recommend you keep the paths short for convenience sake. This expanded structure should contain the following files:

idt_contest\jacoco\lib\jacocoagent.jar - this is the jacoco agent jar

idt_contest\jars\CommandLineEncryption.jar - this is an executable black-box jar that you will test

idt_contest\jars\LeetConverter.jar - this is an executable black-box jar that you will test

idt_contest\jars\RegexPatternMatch.jar - this is an executable black-box jar that you will test

idt_contest\jars\TesterTypeCheck.jar - this is an executable **white-box** jar that you will test

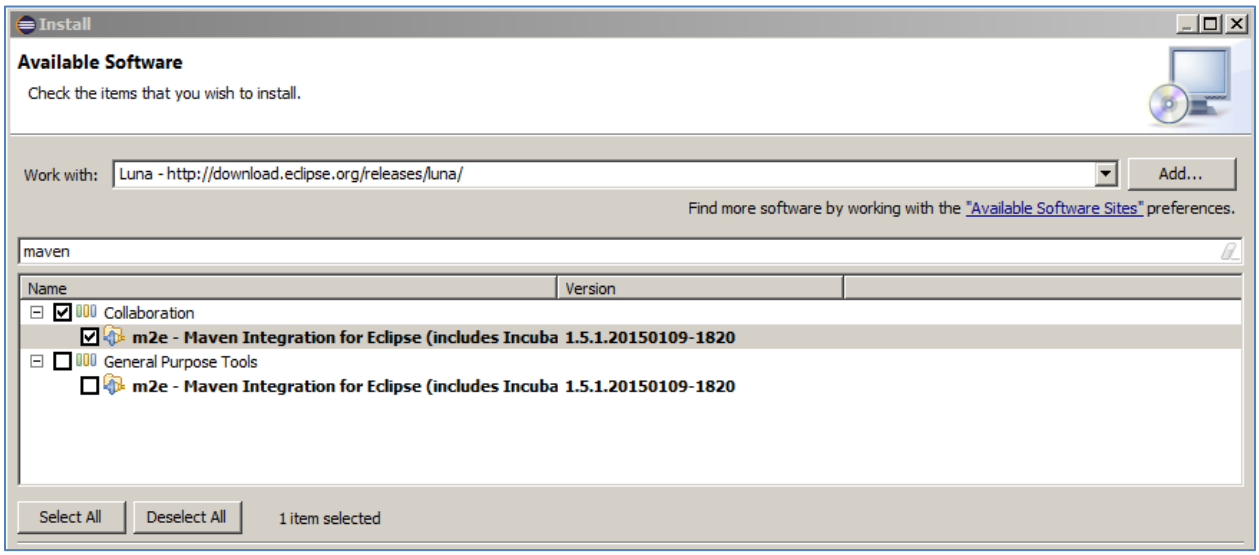
2. Framework Dependencies

We intend for you to develop and execute the framework through Eclipse. There are a minimal set of dependencies that you will need to install in order to get the framework up and running.

A. **Java SE 1.7 or greater** - You will need to have Java installed and available as a command on the command line (added to your path). **The minimum acceptable Java version is 1.7**. The framework will be executing external Java processes, and will require the command 'java –jar' to resolve correctly from the command line.

B. **Eclipse 4.4 or greater**- You will need to have Eclipse installed because the framework was created as a Maven Project for Eclipse. The project was originally created using Eclipse 4.4. (Luna), but there do not appear to be any requirements on a specific version of Eclipse. If you have a version prior to 4.4, you should be okay. Our recommendation is to use Luna (4.4), Mars (4.5), or Neon (4.6). http://www.eclipse.org/downloads/

C. **Maven2Eclipse (m2e) –** You will need to make sure that you have Maven support in Eclipse installed to import and resolve dependencies associated with the framework project. One way to test if you already have this, is to go to File > Import, and look for Maven > Existing Maven Projects. If you do not have that option, you will need to install m2e from Help > Install New Software. Look for the site that uses the name of the version of Eclipse that you have installed (e.g. Luna, Mars, or Neon). You will find 'm2e' under the Collaboration entry. Install m2e and verify that you see the Maven options under File > Import.



Installing Maven from Help > Install New Software in Eclipse Luna

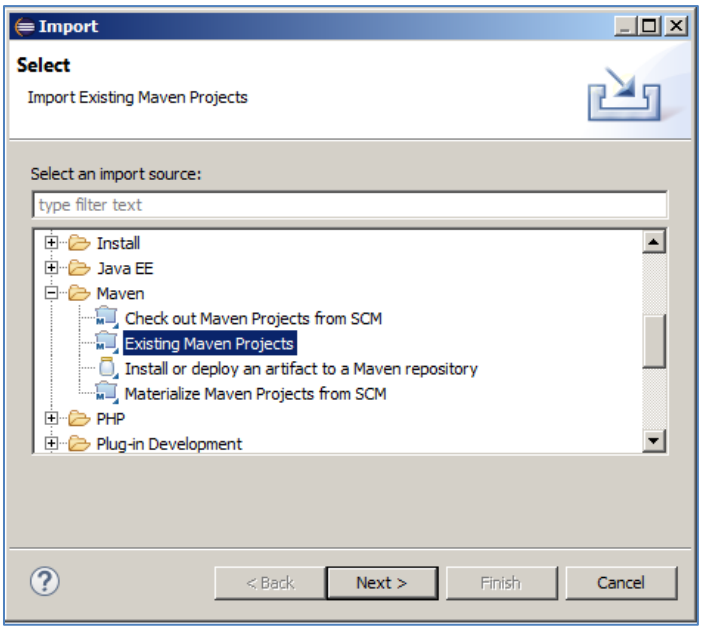
Eclipse will use Maven to resolve the library dependencies for the framework automatically using the pom.xml file that is included with the framework project:

(GROUP)	**(ARTIFACT IDs)**
org.jacoco	jacoco-maven-plugin, org.jacoco.core, org.jacoco.agent, org.jacoco.report
org.ow2.asm	asm-all
commons-cli	commons-cli

3. Importing the framework into Eclipse

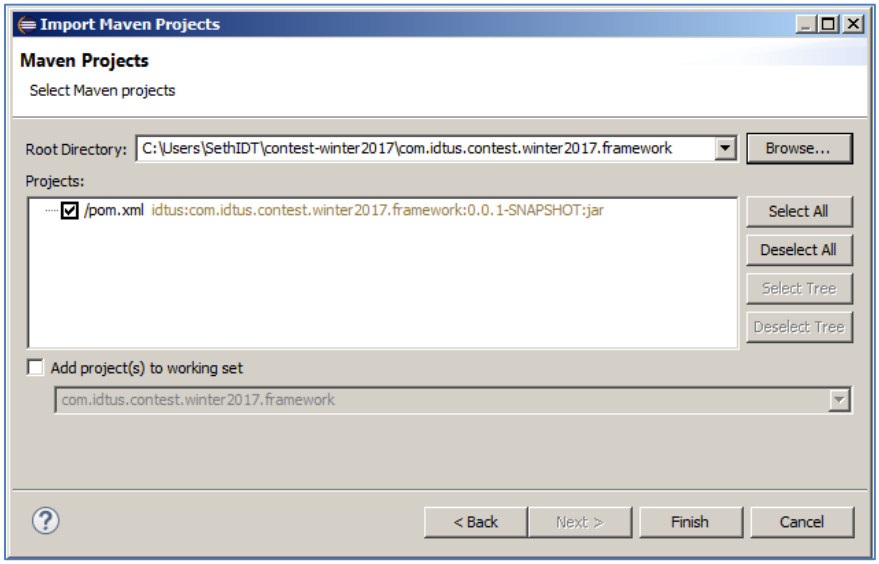
Once you have Java, Eclipse, and Maven all installed, you can import the framework project into Eclipse.

A. **File** > **Import** will bring up a window with several options, select **Maven** > **Existing Maven Projects** and click on the Next button at the bottom.



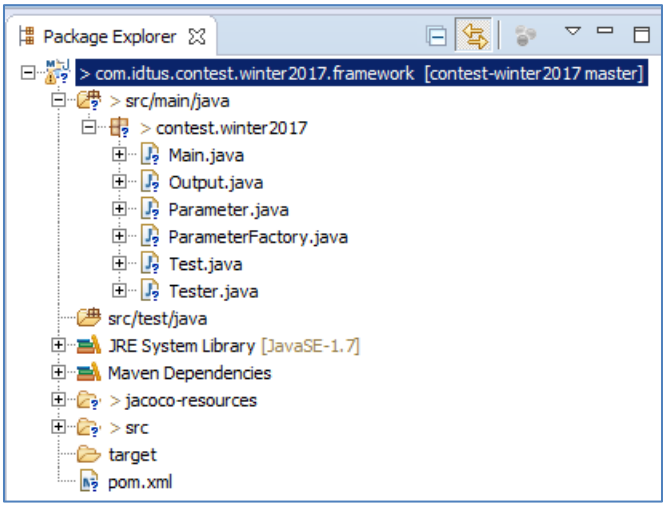
Selecting 'Existing Maven Projects' in Eclipse Import window

B. Use the Browse button on the Maven Projects page to select the path to the framework on your machine (it was unzipped from **com.idtus.contest.winter2017.framework.zip** somewhere on your system as part of step 1). The framework should be a folder called **com.idtus.contest.winter2017.framework**. When the framework is selected, click on the Finish button at the bottom.



Browse to location of the unzipped framework code on your system and click on the Finish button

C. If you have successfully imported the framework project into Eclipse, you should see framework project appear in the Package Explorer on the left hand side of Eclipse.



Successful import of framework project into Eclipse

D. If you have an error in your project due to the fact that the JRE for the project is set to something other than 1.7 or 1.8, simply switch the version of JRE on the project build path by right clicking on 'JRE System Library [JSE-1.5]' under the project contents, and selecting 'Properties'. Under the properties, you can set the proper JRE (1.7 – 1.8).

4. Executing the framework in Eclipse

The framework, at this time, requires three arguments (-jacocoAgentJarPath, -jacocoOutputPath, and -jarToTestPath) in order to execute and test a black-box executable jar. Additionally, the framework currently accepts a –h or –help switch to trigger the help menu.

```
usage: com.idtus.contest.winter2017.framework [-h] [-help]
       [-jacocoAgentJarPath <arg>] [-jacocoOutputPath <arg>]
       [-jarToTestPath <arg>]
 -h                          help
 -help                       help
 -jacocoAgentJarPath <arg>   path to the jacoco agent jar
 -jacocoOutputPath <arg>     path to directory for jacoco output
 -jarToTestPath <arg>        path to the executable jar to test
```

The entry-point for the framework is found in Main.java.

To execute the framework through Eclipse, right click on the Main.java file in the Package Explorer and select Run As > Java Application. If the three required arguments have not been set, you should see the usage print to stdout.

To set the three required arguments for the framework, you will need to open the Run Configuration for the project. This can be done by clicking on Run > Run Configurations at the top of Eclipse. Make sure that the 'Main' configuration is selected in the left side of the Run Configuration window and then select the 'Arguments' tab on the right hand side. You will need to add program arguments for the three required arguments.

jacocoAgentJarPath – We distributed the jacoco agent jar file as part of the supporting files in com.idtus.contest.winter2017.supportingfiles.zip. Inside the expanded directory, the relative path to the jacoco agent jar should be: idt_contest\jacoco\lib\jacocoagent.jar. If you put the idt_contest directory directly into the C:\ directory, your jacocoAgentJarPath (absolute) should be **C:\ idt_contest\jacoco\lib\jacocoagent.jar.**

jacocoOutputPath – This could be any directory on your system - it is not important that you use a specific one. When we were running locally, we used idt_contest\jacoco as our jacoco output directory. If you put the idt_contest directory directly into the C:\ directory, your jacocoOutputPath might be **C:\ idt_contest\jacoco.**

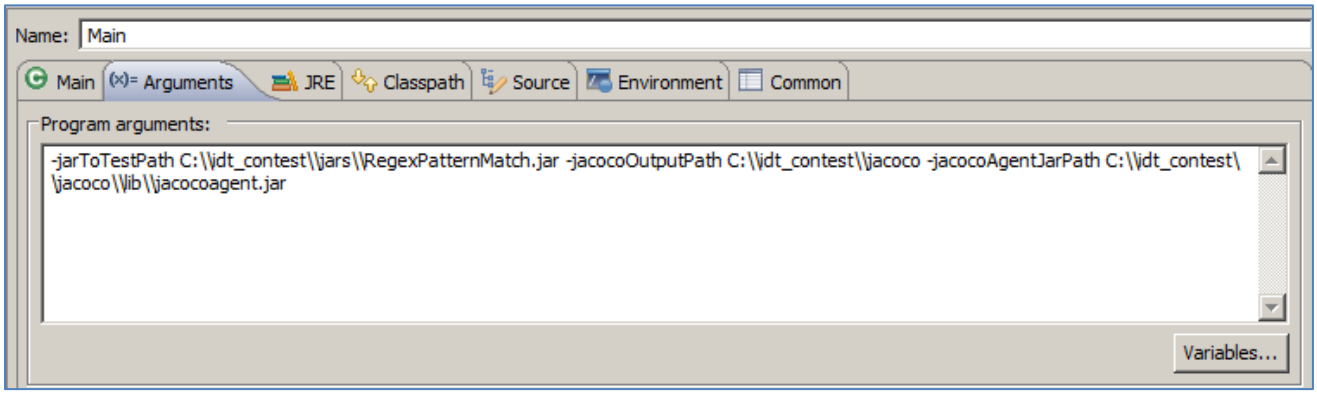
jarToTestPath – This is the path to the black-box executable jar that you intend to test. We distributed all of the sample black-box executable jars as part of the supporting files in com.idtus.contest.winter2017.supportingfiles.zip. Inside the expanded directory, the relative path to the jars to test should be: idt_contest\jars. If you put the idt_contest directory directly into the C:\ directory and intended to test RegexPatternMatch.jar, your jarToTestPath might be **C:\idt_contest\jars\RegexPatternMatch.jar**.

All arguments are entered with a hyphen preceding the argument name and a space separating the argument name and value:

-jarToTestPath C:\\idt_contest\\jars\\RegexPatternMatch.jar

-jacocoOutputPath C:\\idt_contest\\jacoco

-jacocoAgentJarPath C:\\idt_contest\\jacoco\\lib\\jacocoagent.jar



Example of program arguments filled out completely to execute framework against RegexPatternMatch.jar

After updating Arguments > Program Arguments and executing the framework again, the Console in Eclipse should show you results from the basic testing that we have currently implemented.

5. Extending the Framework

We are looking to hire your team to extend our testing framework to expose security vulnerabilities or weaknesses in the software under test. In that respect, you are welcome to change any code in the framework (or rewrite it entirely). Our intention by providing you a framework was to allow you to focus on the challenge of exploratory testing for security vulnerabilities.

In the main method of the framework, you will notice that the primary class for the framework is Tester.java.

```
// the Tester class contains all of the logic for the testing framework
Tester tester = new Tester();
if (tester.init(jarToTestPath, jacocoOutputDirPath, jacocoAgentJarPath)) {
       tester.executeBasicTests();          // this is the basic testing that we have implemented
       tester.executeSecurityTests();       // this is the security vulnerability testing
}
```

We have provided some example code inside 'executeSecurityTests()' to demonstrate some of the features of the framework that you have at your disposal. Your ultimate solution is not required to contain a 'executeSecurityTests()' method. Our intention was to share our vision for the framework, without implementing the security vulnerability testing portion.

Please keep in mind that the final software requirements document for the contest does include requirements for the inputs and the outputs of your solution (so that we can automate testing all of the solutions easily with defined inputs and outputs). Whatever happens between input and output is entirely up to your team.

We have provided one white-box jar that contains all of the source code inside so that you can copy the style to produce new black-box executable jars (in particular, the format of the xxxxxTestBounds.java class that provides metadata to the testing framework):

idt_contest\jars\TesterTypeCheck.jar

The purpose of the TesterTypeCheck is to demonstrate (and test) all of the parameter types supported by the framework. To get a better sense of the metadata provided by the xxxxTestBounds class black-box executable jar, we recommend that you put a break point in the constructor for ParameterFactory and observe the input that is passed into the ParameterFactory constructor during initialization.