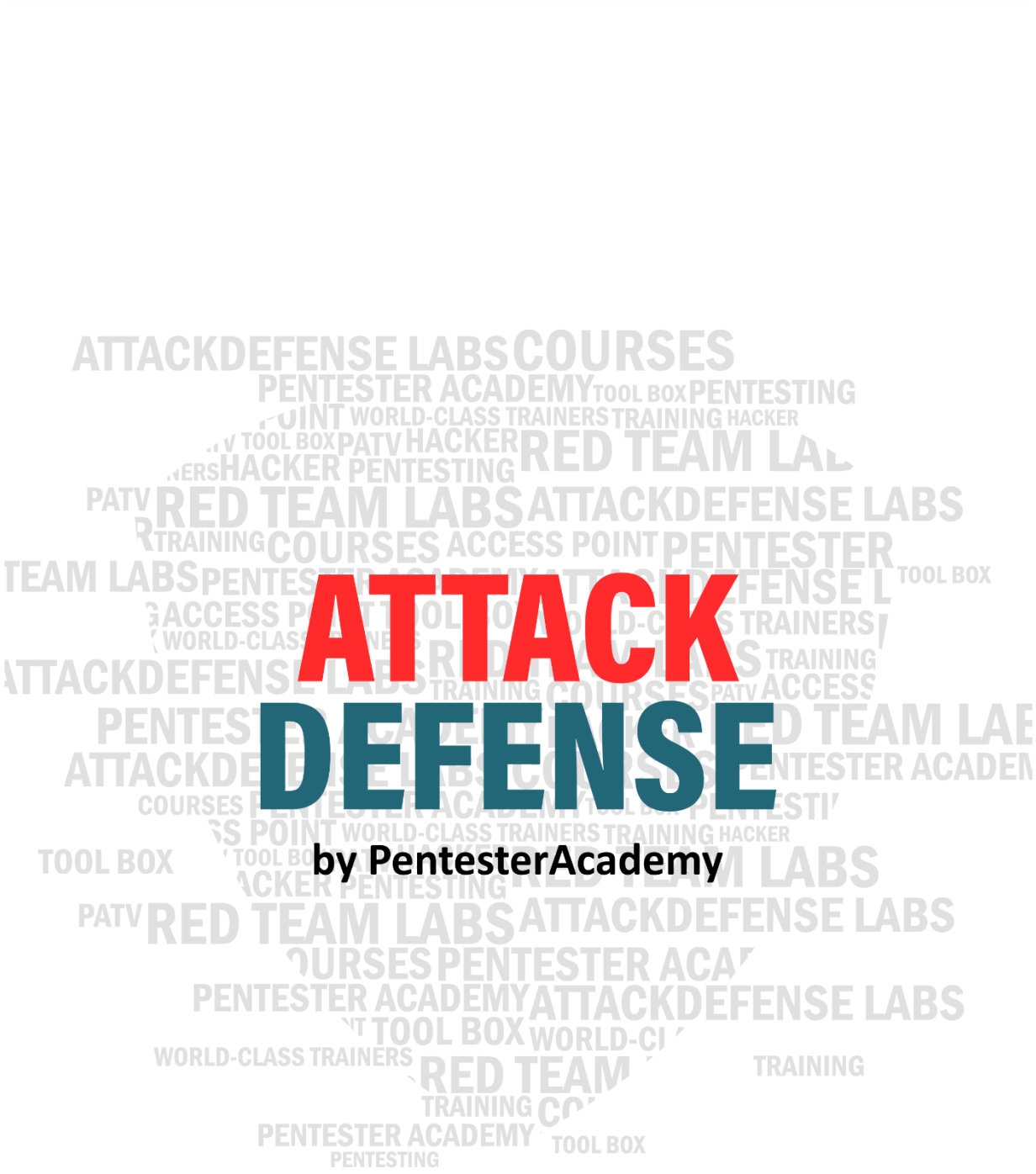
ATTACK
DEFENSE
by PentesterAcademy

Name	JUnit: Automated Tests for Java Project
URL	https://attackdefense.com/challengedetails?cid=2259
Type	DevOps Basics: Functional Testing

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Challenge Description

JUnit is a framework that is used for testing the project written in the Java programming language.

A Kali CLI machine (kali-cli) is provided to the user. The source code for three sample web applications is provided in the home directory of the root user.

Objective: Learn to write and run the tests using the JUnit framework!

Instructions:

- The source code of web applications is provided at /root/github-repos

Solution

Step 1: Check the provided web applications.

Command: ls -l github-repos/

```
root@attackdefense:~# ls -l github-repos/
total 12
drwxr-xr-x 5 root root 4096 Sep 16 08:31 dissector
drwxr-xr-x 6 root root 4096 Sep 16 08:31 junit5-migration-gradle
drwxr-xr-x 4 root root 4096 Sep 16 08:31 time-tracker
root@attackdefense:~#
```

We will take one example at a time and run the tool on that.

Example 1: Dissector

Step 1: Change to the dissector directory and check its contents.

Commands:
cd ~/github-repos/dissector
ls

```
root@attackdefense:~# cd github-repos/dissector/
root@attackdefense:~/github-repos/dissector#
root@attackdefense:~/github-repos/dissector# ls
dissector-agent  dissector-monitor  docs  LICENSE  pom.xml  README.md
root@attackdefense:~/github-repos/dissector#
```

Step 2: Check the contents of Pom.xml of the application. The POM (Project Object Model) file contains the configuration information for the maven to build the project.

Command: cat pom.xml

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

The JUnit has been added as a dependency in the source code.

Step 3: Listing all the test files present in the codebase.

Commands:
ls dissector-agent/src/test/java/eu/stamp_project/dissector/agent/
ls dissector-monitor/src/test/java/eu/stamp_project/dissector/monitor/

```
root@attackdefense:~/github-repos/dissector# ls dissector-agent/src/test/java/eu/stamp_project/dissector/agen
t/
ArgumentsParserTest.java  MethodListParserTest.java
root@attackdefense:~/github-repos/dissector#
root@attackdefense:~/github-repos/dissector#
root@attackdefense:~/github-repos/dissector# ls dissector-monitor/src/test/java/eu/stamp_project/dissector/mo
nitor/
ProcessEmulatorTest.java  StackTraceMonitorMojoTest.java
root@attackdefense:~/github-repos/dissector#
```

All those test classes will be scanned whose fully qualified names match the following patterns.

- **/Test*.java
- **/*Test.java
- **/*Tests.java
- **/*TestCase.java

The functions present in these test files, that are having '@Test' annotations will be executed during the 'test' phase.

For example, the file 'ProcessEmulatorTest.java' located at dissector-monitor/src/test/java/eu/stamp_project/dissector/monitor/ProcessEmulatorTest.java

Command: cat ~/github-repos/dissector-agent/dissector-monitor/src/test/java/eu/stamp_project/dissector/monitor/ProcessEmulatorTest.java

```
@Test
public void methodCalledTwiceAtSameDepth() {
    List<String> methods = new ArrayList<>(4);
    methods.add("path.to.Class.method1()");
    methods.add("path.to.ClassTest.test()");

    Set<String> tests = new HashSet<>();
    tests.add("path.to.ClassTest.test()");

    MethodSet methodSet = new MethodSet(methods, tests);
    ProcessEmulator emulator = new ProcessEmulator(methodSet);

    emulator.enter(1,1, 1);
    emulator.enter(1, 0, 2);
    emulator.exit(1, 0, 2);
    emulator.enter(1, 0, 2);
    emulator.exit(1, 0, 2);
    emulator.exit(1,1, 1);
```

This is a test case defined in the methodCalledTwiceAtSameDepth.java using the @Test annotation.

Step 4: Run the maven command to test the code of the repository.

Command: mvn test

```
root@attackdefense:~/github-repos/dissector# mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] dissector                                                          [pom]
[INFO] dissector-agent                                                    [jar]
[INFO] dissector-monitor                                          [maven-plugin]
[INFO]
[INFO] ---------------------< eu.stamp-project:dissector >----------------------
[INFO] Building dissector 2.0-SNAPSHOT                                    [1/3]
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] ------------------< eu.stamp-project:dissector-agent >------------------
[INFO] Building dissector-agent 2.0-SNAPSHOT                              [2/3]
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
```

```
[debug] [[D][0:<File.java:to.Class1:dummyMethod:1><File.java:to.Class1:another:2>]]
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.083 sec
```

```
Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for dissector 2.0-SNAPSHOT:
[INFO]
[INFO] dissector ..................................... SUCCESS [  0.006 s]
[INFO] dissector-agent ............................... SUCCESS [  7.017 s]
[INFO] dissector-monitor ............................. SUCCESS [  6.773 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14.177 s
[INFO] Finished at: 2020-09-20T07:10:52Z
[INFO] ------------------------------------------------------------------------
root@attackdefense:~/github-repos/dissector#
```

The maven has successfully completed the tests using JUnit.

Example 2:

Step 1: Change to the cloned directory and check its contents.

Commands:
cd time-tracker/
ls

```
root@attackdefense:~/github-repos# cd time-tracker/
root@attackdefense:~/github-repos/time-tracker#
root@attackdefense:~/github-repos/time-tracker# ls
core  LICENSE  pom.xml  README.md  web
root@attackdefense:~/github-repos/time-tracker#
```

Step 2: Listing all the test files present in the codebase.

Commands:
ls core/src/test/java/training/taylor/timetracker/core/

```
root@attackdefense:~/github-repos/time-tracker#
root@attackdefense:~/github-repos/time-tracker# ls core/src/test/java/training/taylor/timetracker/core/
TrackerCoreConfigTest.java  TrackerTest.java
root@attackdefense:~/github-repos/time-tracker#
```

All those test classes will be scanned whose fully qualified names match the following patterns.

- **/Test*.java
- **/*Test.java
- **/*Tests.java
- **/*TestCase.java

The functions present in these test files, that are having '@Test' annotations will be executed during the 'test' phase.

For example, the file "TrackerTest.java" located at core/src/test/java/training/taylor/timetracker/core/TrackerTest.java

Command: cat ~/github-repos/time-tracker/core/src/test/java/training/taylor/timetracker/core/TrackerTest.java

```
@Test
public void testMe() {
    assertNotNull(tracker);
}

@Test
public void testAdd() {
    TimeEntry entry = new TimeEntry();
    entry.setDescription("Entry Test");
    entry.setRate(80.0f);
    entry.setTime(3);
    tracker.add(entry);
    assertTrue(tracker.size() > 0);
}
```

This is a test case defined in the ProcessEmulatorTest.java using the @Test annotation.

Step 3: Run the maven command to test the code of the repository.

Command: mvn test

```
root@attackdefense:~/github-repos/time-tracker# mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Time Tracker (Parent)                                              [pom]
[INFO] Time Tracker (Core)                                                [jar]
[INFO] Time Tracker (Web)                                                 [war]
[INFO]
[INFO] ----------< training.taylor.time-tracker:time-tracker-parent >----------
[INFO] Building Time Tracker (Parent) 0.5.0-SNAPSHOT                      [1/3]
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] -----------< training.taylor.time-tracker:time-tracker-core >-----------
[INFO] Building Time Tracker (Core) 0.5.0-SNAPSHOT                        [2/3]
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ time-tracker-web ---
[INFO] No tests to run.
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for Time Tracker (Parent) 0.5.0-SNAPSHOT:
[INFO]
[INFO] Time Tracker (Parent) ............................... SUCCESS [  0.207 s]
[INFO] Time Tracker (Core) ................................. SUCCESS [  8.803 s]
[INFO] Time Tracker (Web) .................................. SUCCESS [  0.082 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.382 s
[INFO] Finished at: 2020-09-20T07:19:52Z
[INFO] ------------------------------------------------------------------------
root@attackdefense:~/github-repos/time-tracker#
```

The maven has successfully completed the tests using JUnit.

Example 3: Junit5 Migration Gradle

Step 1: Change to the junit5-migration-gradle directory and check its contents.

Commands:
cd junit5-migration-gradle/
ls

```
root@attackdefense:~/github-repos# cd junit5-migration-gradle/
root@attackdefense:~/github-repos/junit5-migration-gradle#
root@attackdefense:~/github-repos/junit5-migration-gradle# ls
build  build.gradle  gradle  gradlew  gradlew.bat  README.md  src
root@attackdefense:~/github-repos/junit5-migration-gradle#
root@attackdefense:~/github-repos/junit5-migration-gradle#
```

Step 2: Check the build.gradle of the application.

Command: cat build.gradle

```
dependencies {
        testImplementation(platform("org.junit:junit-bom:5.6.2"))

        testImplementation("org.junit.jupiter:junit-jupiter") {
                because 'allows to write and run Jupiter tests'
        }

        testImplementation("junit:junit:4.13")
        testRuntimeOnly("org.junit.vintage:junit-vintage-engine") {
                because 'allows JUnit 3 and JUnit 4 tests to run'
        }

        testRuntimeOnly("org.junit.platform:junit-platform-launcher") {
                because 'allows tests to run from IDEs that bundle older version of launcher'
        }
}
```

In build.gradle, the tests are defined for JUnit to perform on the application.

Step 3: Listing all the test files present in the codebase.

Commands:
ls src/test/java/com/example/project/

```
root@attackdefense:~/github-repos/junit5-migration-gradle#
root@attackdefense:~/github-repos/junit5-migration-gradle# ls src/test/java/com/example/project/
FirstTest.java  JUnit4Test.java  OtherTests.java  SecondTest.java
root@attackdefense:~/github-repos/junit5-migration-gradle#
```

All those test classes will be scanned whose fully qualified names match the following patterns.

- **/Test*.java
- **/*Test.java
- **/*Tests.java
- **/*TestCase.java

The functions present in these test files, that are having '@Test' annotations will be executed during the 'test' phase.

For example, the file "FirstTest.java" is located at src/test/java/com/example/project/FirstTest.java

Command: cat ~/github-repos/junit5-migration-gradle/src/test/java/com/example/project/FirstTest.java

```
@Tag("fast")
class FirstTest {

        @Test
        @DisplayName("My 1st JUnit 5 test! 😎")
        void myFirstTest(TestInfo testInfo) {
                Calculator calculator = new Calculator();
                assertEquals(2, calculator.add(1, 1), "1 + 1 should equal 2");
                assertEquals("My 1st JUnit 5 test! 😎", testInfo.getDisplayName(), () -> "TestInfo is injected
 correctly");
        }

}
root@attackdefense:~/github-repos/junit5-migration-gradle#
```

This is a test case defined in the FirstTest.java using the @Test annotation.

Step 4: Start the scan on the repository using Gradle.

Command: ./gradlew clean test

```
root@attackdefense:~/github-repos/junit5-migration-gradle# ./gradlew clean test
Starting a Gradle Daemon, 2 incompatible and 1 stopped Daemons could not be reused, use --status for details

> Task :test

JUnit4Test > test PASSED

SecondTest > mySecondTest() SKIPPED

FirstTest > My 1st JUnit 5 test! 😎PASSED

OtherTests > testThisThing() PASSED

OtherTests > testThisOtherThing() PASSED

BUILD SUCCESSFUL in 22s
5 actionable tasks: 5 executed
root@attackdefense:~/github-repos/junit5-migration-gradle#
```

The gradle clean test command started running tests while building the application and successfully completed the build.

Example 4: Create a custom test case for the functionality of the 'calculator' class in the junit5-migration-gradle project

Step 1: Navigate to the project directory where test cases are saved.

Commands:
cd ~/github-repos/junit5-migration-gradle/src/test/java/com/example/project/
ls

```
root@attackdefense:~#
root@attackdefense:~# cd ~/github-repos/junit5-migration-gradle/src/test/java/com/example/project/
root@attackdefense:~/github-repos/junit5-migration-gradle/src/test/java/com/example/project# ls
FirstTest.java  JUnit4Test.java  OtherTests.java  SecondTest.java
root@attackdefense:~/github-repos/junit5-migration-gradle/src/test/java/com/example/project#
```

Step 2: Create the test case and save the file as "SampleTest.java"

```
package com.example.project;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;


class SampleTest {

   @Test
   void additionTest() {
         Calculator calculator = new Calculator();
         assertEquals(14, calculator.add(7, 7));
   }

}
```

Explanation:

- The function will be executed by JUnit because it contains '@Test' annotation from the imported library
- An object of calculator class is created
- Imported assertEquals library to use the function to check whether the result of an addition will be equal to 14 or not.

Command: cat SampleTest.java

```
root@attackdefense:~/github-repos/junit5-migration-gradle/src/test/java/com/example/project# cat SampleTest.java
package com.example.project;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;


class SampleTest {

    @Test
    void additionTest() {
         Calculator calculator = new Calculator();
         assertEquals(14, calculator.add(7, 7));
    }

}

root@attackdefense:~/github-repos/junit5-migration-gradle/src/test/java/com/example/project#
```

Step 3: Navigate to the project directory and run the test cases.

Commands:
cd ~/github-repos/junit5-migration-gradle/
./gradlew clean test

```
root@attackdefense:~/github-repos/junit5-migration-gradle# ./gradlew clean test
Starting a Gradle Daemon, 2 incompatible and 1 stopped Daemons could not be reused, use --status for details

> Task :test

JUnit4Test > test PASSED

SecondTest > mySecondTest() SKIPPED

SampleTest > additionTest() PASSED

FirstTest > My 1st JUnit 5 test! 😎PASSED

OtherTests > testThisThing() PASSED

OtherTests > testThisOtherThing() PASSED

BUILD SUCCESSFUL in 21s
5 actionable tasks: 5 executed
root@attackdefense:~/github-repos/junit5-migration-gradle#
```

The “additionTest” which was defined in SampleTest.java test case file has successfully passed.

Learnings

Perform Functional Testing using the JUnit tool.