Code Quality
Reproduce vulnerabilities with Proof-of-Vulnerability tests (Lab)

Overview

In this lab, we will try to automatically reproduce the real-world vulnerabilities (disclosed on the *National Vulnerability Database* website) in Java open-source projects. To this end, we need to spot the so-called Proof-of-Vulnerability (PoV) test cases which are *failing* if the *vulnerabilities exist* in the code base and *passing* if the *vulnerabilities are removed*. In some cases, if these PoV tests are not available, you need to write them yourself by leveraging the description on NVD and bug reports for the corresponding vulnerabilities.

Setup

We have prepared a Docker image that contains four vulnerabilities that you need to spot the PoV tests, as described in the below table.

Vulnerability	Patch	Bug Report
CVE-2013-2186	https://github.com/apache/commons-fileupload/commit/163a6061fbc077d4b6e4787d26857c2baba495d1	
CVE-2016-7051	https://github.com/FasterXML/jackson-dataformat-xml/commit/eeff2c312e9d4caa8c9f27b8f740c7529d00524a	
CVE-2018-1324	https://github.com/apache/commons-compress/commit/2a2f1dc48e22a34ddb72321a4db211da91aa933b	https://issues.apache.org/jira/browse/COMPRESS-432
CVE-2016-3720	https://github.com/FasterXML/jackson-dataformat-xml/commit/f0f19a4c924d9db9a1e2830434061c8640092cc0	https://github.com/FasterXML/jackson-dataformat-xml/issues/190

First, make sure that you have installed Docker and start it on your machine. To start and access to our Docker container, run the below command:

```
$ docker run -it bqcuongas/sselab
```

You are now inside the container, you will recognize the terminal cursor turns into `root@xyz:/SSE_LAB#` (`xyz` is the id of your container). In the current directory, there are four folders (corresponding to four vulnerabilities). In each folder, the source code of the project is checked out to the revision that the vulnerabilities exist.

Your Tasks

We will guide you to spot the PoV tests for the first vulnerability. Then, you need to do yourself to spot or write the PoV tests for the other vulnerabilities. Basic knowledge about building/running tests for Java projects (Maven) and JUnit will be required to perform this lab.

Task 1. Find/Write the PoV tests for CVE-2013-2186

We will guide you for doing this task as a tutorial. Our source code has been already isolated and checked out the vulnerable revision.

First, let take a look at the patch for this vulnerability at https://github.com/apache/commons-fileupload/commit/163a6061fbc077d4b6e4787d26857c2baba495d1:

20 src/main/java/org/apache/commons/fileupload/disk/DiskFileItem.java

```
@@ -656,6 +656,26 @@ private void readObject(ObjectInputStream in)
664 +         if (repository != null) {
665 +             if (repository.isDirectory()) {
666 +                 // Check path for nulls
667 +                 if (repository.getPath().contains("\0")) {
668 +                     throw new IOException(format(
669 +                             "The repository [%s] contains a null character",
670 +                             repository.getPath()));
671 +                 }
672 +             } else {
673 +                 throw new IOException(format(
674 +                         "The repository [%s] is not a directory",
675 +                         repository.getAbsolutePath()));
676 +             }
677 +         }
```

137 src/test/java/org/apache/commons/fileupload/DiskFileItemSerializeTest.java

```
+    @Test
+    public void testBelowThreshold() throws Exception {
+        // Create the FileItem
+        byte[] testFieldValueBytes = createContentBytes(threshold - 1);
+        testInMemoryObject(testFieldValueBytes);
+    }
+    @Test
+    public void testValidRepository() throws Exception {
+        // Create the FileItem
+        byte[] testFieldValueBytes = createContentBytes(threshold);
+        File repository = new File(System.getProperty("java.io.tmpdir"));
+        testInMemoryObject(testFieldValueBytes, repository);
+    }
+    @Test(expected=IOException.class)
+    public void testInvalidRepository() throws Exception {
+        // Create the FileItem
+        byte[] testFieldValueBytes = createContentBytes(threshold);
+        File repository = new File(System.getProperty("java.io.tmpdir") + "file");
+        FileItem item = createFileItem(testFieldValueBytes, repository);
+        deserialize(serialize(item));
+    }
+    @Test(expected=IOException.class)
+    public void testInvalidRepositoryWithNullChar() throws Exception {
+        // Create the FileItem
+        byte[] testFieldValueBytes = createContentBytes(threshold);
+        File repository = new File(System.getProperty("java.io.tmpdir") + "\0");
+        FileItem item = createFileItem(testFieldValueBytes, repository);
+        deserialize(serialize(item));
+    }
```

Developers of Apache Common FileUpload added an if-precondition to check if the null byte existed in the input repository patch. If it does, an IOException is thrown. Developers also added some new test cases in the patch. These tests are the *potential* PoV tests, especially, those ones that assert the IOException will be thrown in the execution.

To verify our insight, we will run all the tests of the project on the vulnerable revision to see if there were any failing tests then re-run them on the patched

revision to see if they were passing after removing vulnerability from the code base.

To perform the tests execution, run the below command:

```
$ cd CVE-2013-2186
$ mvn test
```

And the test result:

```
testInvalidRepositoryWithNullChar(org.apache.commons.fileupload.DiskF
ileItemSerializeTest)  Time elapsed: 0.098 sec  <<< FAILURE!
testInvalidRepository(org.apache.commons.fileupload.DiskFileItemSeria
lizeTest)  Time elapsed: 0.001 sec  <<< FAILURE!
Failed tests:
   Expected exception: java.io.IOException
   Expected exception: java.io.IOException

Tests run: 70, Failures: 2, Errors: 0, Skipped: 0
```

So we found two failing tests:
`testInvalidRepositoryWithNullChar(org.apache.commons.fileupload.DiskFileItemSerializeTest)`,
and
`testInvalidRepository(org.apache.commons.fileupload.DiskFileItemSerializeTest)`

Now we need to verify if these two tests are passing after the patch is applied.
To applied the patch, run the below command:

```
$ git checkout -f 163a6061fbc077d4b6e4787d26857c2baba495d1
```

163a6061fbc077d4b6e4787d26857c2baba495d1 is the commit hash of the patch commit (see in the table)

Then we re-run the tests:

```
$ mvn test
```

And now, there are no failing tests found in the test results:

```
Results :
Tests run: 70, Failures: 0, Errors: 0, Skipped: 0
```

So we could conclude the PoV tests for our vulnerability are:

- `testInvalidRepositoryWithNullChar(org.apache.commons.fileupload.DiskFileItemSerializeTest)`
- `testInvalidRepository(org.apache.commons.fileupload.DiskFileItemSerializeTest)`

Task 2. Find/Write the PoV tests for CVE-2016-7051

Hint: This vulnerability *might be* reproduced in a similar way as the first one.

Task 3. Find/Write the PoV tests for CVE-2018-1324

Hint: Did you take a look at the bug report to see what causes the infinite loop that leads to this vulnerability?

Task 4. Find/Write the PoV tests for CVE-2016-3720

Hint: Could you reuse boilerplate code from the PoV test for CVE-2016-7051?

(Supplementary) The Dockerfile for the lab

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive
ENV LANG=C.UTF-8

# install required softwares
RUN apt update \
    && apt install -y wget curl git python3 python3-pip vim zsh unzip
bzip2 xz-utils \
    openjdk-8-jdk maven \
    openssh-client patch software-properties-common time
build-essential \
    && rm -rf /var/lib/apt/lists/*

RUN wget
https://github.com/robbyrussell/oh-my-zsh/raw/master/tools/install.sh
-O - | zsh || true

WORKDIR /SSE_LAB
ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64

# CVE-2013-2186
RUN git clone https://github.com/apache/commons-fileupload
CVE-2013-2186; cd CVE-2013-2186; git checkout
163a6061fbc077d4b6e4787d26857c2baba495d1; git checkout HEAD~1 --
src/main/java/org/apache/commons/fileupload/disk/DiskFileItem.java

# CVE-2018-1324
RUN git clone https://github.com/apache/commons-compress
CVE-2018-1324; cd CVE-2018-1324; git checkout
2a2f1dc48e22a34ddb72321a4db211da91aa933b; git checkout HEAD~1 --
src/main/java/org/apache/commons/compress/archivers/zip/X0017_StrongE
ncryptionHeader.java

# CVE-2016-7051
RUN git clone https://github.com/FasterXML/jackson-dataformat-xml
CVE-2016-7051; cd CVE-2016-7051; git checkout
eeff2c312e9d4caa8c9f27b8f740c7529d00524a; git checkout HEAD~1 --
src/main/java/com/fasterxml/jackson/dataformat/xml/XmlFactory.java
```

```
# CVE-2016-3720
RUN git clone https://github.com/FasterXML/jackson-dataformat-xml
CVE-2016-3720; cd CVE-2016-3720; git checkout
f0f19a4c924d9db9a1e2830434061c8640092cc0; git checkout HEAD~1 --
src/main/java/com/fasterxml/jackson/dataformat/xml/XmlFactory.java
```