Rylan Peterson

Sunday, May 1, 2022

Prof. Lundin

CSC245

**Table Of Contents**

**Summary………………………………………………………..………….….2**

**Vulnerabilities……………………….…………..…………….…………….3-12**

(CERT.EXP01.NP-1) ………….………………………………………………….3

(CERT.ENV02.ENV-1) ………….……………………………………………….4

(CERT.FIO16.CDBV-1) ………………………………………………………….5

(CERT.MSC02.SRD-1) ………………………………………………………….6

(CERT.FIO13.PEO-1) …………..……………………………………………….7

(CERT.ERR00.LGE-2) …………….…………………………………………….8

(CERT.LCK06.INSTLOCK-2) ……….………………………………………….9

(CERT.NUM13.AIC-2) ……………………………………………...………….10

(CERT.FIO07.EXEC-3) ……………………………………………..….………11

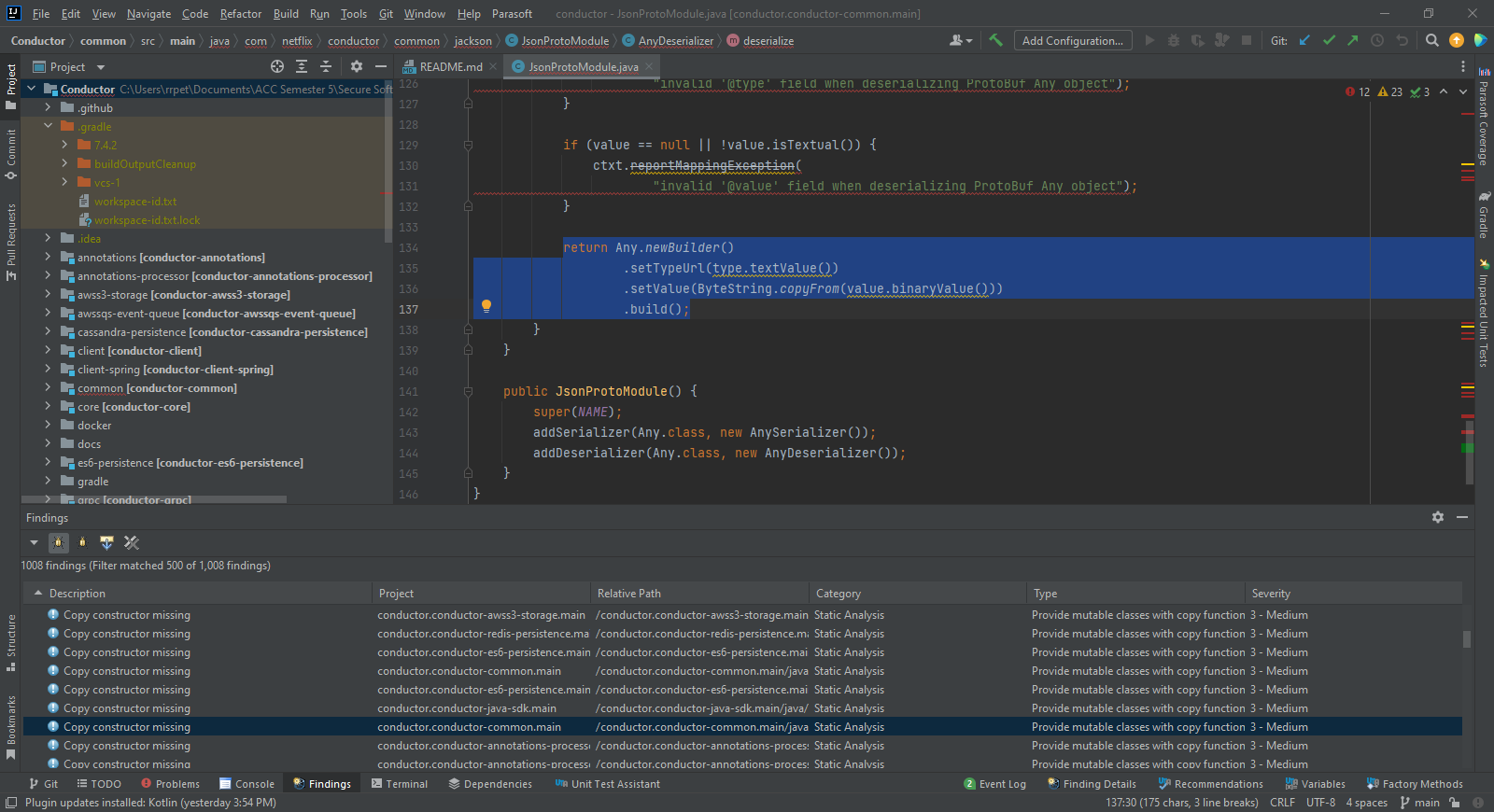
(CERT.OBJ08.INNER-3) ……………….………………..…………...….……12

**References…………………………………….….……………..…………13-15**

Lab 5 Summary

This report is based off of the Netflix Conductor code provided by Github, sourced by professor. After using Jtest to scan the report, the scanning program was able to find 1,240 issues. I found that there are 12 vulnerabilities with a severity rating of 1, 34 with severity rating of 2, 800 with a rating of 3, Etc…

Proof of program in Intellij and a finished scan:(scan time stamp: April 30, 2022, 4:15pm)



**Vulnerability 1**

Avoid NullPointerException (CERT.EXP01.NP-1)

**Severity:1**

**Location:**

/conductor.conductor-common.main/java/com/netflix/conductor/common/jackson/JsonProtoModule.java

**CODE WITH VULNERABILITY:**

**return Any.*newBuilder*()**

**.setTypeUrl(type.textValue())**

**.setValue(ByteString.*copyFrom*(value.binaryValue()))**

**.build();**

The issue with this vulnerability, is the Type.TextValue(), and Value.binaryValue() are values that should not be Null, Each variable or expression that is dereferenced should have a non-null value

at the point of dereferencing on any flow path. This rule detects when a NullPointerException may possibly be thrown at runtime due to a null pointer dereference.

**Vulnerability 2**

Do not use the non-portable 'System.getenv()' method (CERT.ENV02.ENV-1)

**Severity:1**

**Location:**/conductor.conductor-common.main/java/com/netflix/conductor/common/utils/EnvUtils.java

**CODE WITH VULNERABILITY:**

**public static boolean isEnvironmentVariable(String test) {**

**for (SystemParameters c : SystemParameters.*values*()) {**

**if (c.name().equals(test)) {**

**return true;**

**}**

**}**

**String value =**

**Optional.ofNullable(System.getProperty(test)).orElseGet(() -> System.getenv(test));**

**return value != null;**

**}**

**//////////////////////////////////////////////////////////////////////////////**

**public static String getSystemParametersValue(String sysParam, String taskId) {**

**if ("CPEWF\_TASK\_ID".equals(sysParam)) {**

**return taskId;**

**}**

**String value = System.getenv(sysParam);**

**if (value == null) {**

**value = System.getProperty(sysParam);**

**}**

**return value;**

**}**

Above are two instances where System.getenv() is used. The reason this is a vulnerability is because the 'System.getenv()' method was deprecated in versions of the JDK prior to

version 1.5. As of JDK version 1.5, it is no longer deprecated; however, it is

still not guaranteed to work on all systems. This creates a risk within the program because it may be incompatible depending on the software that's running it. Having a well diverse code that works in different versions, or on different devices, etc is best for a program as big as this.This vulnerability is not necessarily an issue it is more of a code quality issue, as long as it's used properly it would be acceptable

**Vulnerability 3**

Canonicalize all data before validation (CERT.FIO16.CDBV-1)

**Severity:1**

**Location:**/conductor.conductor-common.main/java/com/netflix/conductor/common/constraints/TaskReferenceNameUniqueConstraint.java

**CODE WITH VULNERABILITY:**

**workflow.getTasks().stream()**

**.filter(workflowTask -> workflowTask.getInputParameters() != null)**

**.forEach(**

**workflowTask -> {**

**List<String> errors =**

**ConstraintParamUtil.*validateInputParam*(**

**workflowTask.getInputParameters(),**

**workflowTask.getName(),**

**workflow);**

With this bit of code there is a vulnerability at workflowTask.getInputParameters(), This is a vulnerability because this code validates data without first canonicalizing the data. This has been flagged as a vulnerability because when you call one of the user-specified validation methods which pass in a local variable you need to pass to one of the user-specified canonicalization methods beforehand. With CERT.FIO16.CDBV-1, A violation will also be flagged for each call to a validation method which passes in the return value of another method call without first passing this return value to a canonicalization method.

**Vulnerability 4**

Use 'java.security.SecureRandom' instead of 'java.util.Random' or 'Math.random()' (CERT.MSC02.SRD-1)

**Severity:1**

**Location:**/conductor.conductor-java-sdk.main/EXT/com/netflix/conductor/sdk/example/shipment/ShipmentWorkers.java

**CODE WITH VULNERABILITY**

**@WorkerTask("get\_order\_details")**

**public List<Order> getOrderDetails(@InputParam("orderNo") String orderNo) {**

**int lineItemCount = new Random().nextInt(10);**

**List<Order> orderDetails = new ArrayList<>();**

**for (int i = 0; i < lineItemCount; i++) {**

**Order orderDetail = new Order(orderNo, "sku\_" + i, 2, BigDecimal.valueOf(20.5));**

**orderDetail.setOrderNumber(UUID.randomUUID().toString());**

**orderDetail.setCountryCode(i % 2 == 0 ? "US" : "CA");**

**if (i % 3 == 0) {**

**orderDetail.setCountryCode("UK");**

**}**

This bit of code has a vulnerability at new Random(), this happens to be a vulnerability because enforcing this rule will help to protect against the OWASP Top 10 2017

application vulnerability "A3-Sensitive Data Exposure". Meaning that Without using java.security.SecureRandom there is a risk of others seeing sensitive data that might be shown under those random values. The 'SecureRandom' class provides a cryptographically strong pseudo-random number generator (PRNG).

**Vulnerability 5**

Do not pass exception messages into output in order to prevent the application from leaking sensitive information (CERT.FIO13.PEO-1)

**Severity:1**

**Location:**/conductor.conductor-annotations-processor.main/java/com/netflix/conductor/annotationsprocessor/protogen/ProtoGenTask.java

**CODE WITH VULNERABILITY**

**public void generate() {**

**ProtoGen generator = new ProtoGen(protoPackage, javaPackage, goPackage);**

**try {**

**generator.processPackage(sourceJar, sourcePackage);**

**generator.writeMapper(mapperDir, mapperPackage);**

**generator.writeProtos(protosDir);**

**} catch (IOException e) {**

**System.err.printf("protogen: failed with %s\n", e);**

**}**

**}**

This Code has a vulnerability at System.err.printf(), The problem with this line is that the Print method 'printf()' is not allowed in a "catch" block. Sensitive information might be leaked when exception messages are passed into output. Hackers trying to gain information about a server application could look at exception messages leaked from the server. The recommended way to obtain exception information is to deploy a logging system (instead of using print methods).

**Vulnerability 6**

Ensure all exceptions are either logged with a standard logger or rethrown (CERT.ERR00.LGE-2)

**Severity:2**

**Location:**/conductor.conductor-annotations-processor.main/java/com/netflix/conductor/annotationsprocessor/protogen/ProtoGenTask.java

**CODE WITH VULNERABILITY:**

**public void generate() {**

**ProtoGen generator = new ProtoGen(protoPackage, javaPackage, goPackage);**

**try {**

**generator.processPackage(sourceJar, sourcePackage);**

**generator.writeMapper(mapperDir, mapperPackage);**

**generator.writeProtos(protosDir);**

**} catch (IOException e) {**

**System.err.printf("protogen: failed with %s\n", e);**

**}**

**}**

This code has an issue at Catch() this is because using a logging mechanism to track caught exceptions allows you to obtain information about the possible security vulnerabilities and appropriately fix code defects. Exceptions that are caught but are not logged should be rethrown. This vulnerability is bad because exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.

**Vulnerability 7**

Do not use an instance lock to protect shared static data (CERT.LCK06.INSTLOCK-2)

**Severity:2**

**Location:**/conductor.conductor-java-sdk.main/EXT/com/netflix/conductor/sdk/testing/LocalServerRunner.java

**CODE WITH VULNERABILITY:**

**String command =**

**"java -Dserver.port="**

**+ localServerPort**

**+ " -DCONDUCTOR\_CONFIG\_FILE="**

**+ configFile**

**+ " -jar "**

**+ serverFile;**

***LOGGER*.info("Running command {}", command);**

This code has an issue at LOGGER.info(). This program must not use instance locks to protect static shared data because instance locks are ineffective when two or more instances of the class are created. Consequently, failure to use a static lock object leaves the shared state unprotected against concurrent access. This creates an issue because the shared static variable "LOGGER" is accessed while the instance lock "this" is held.

**Vulnerability 8**

Avoid implicit casts from integer data types to floating point data types (CERT.NUM13.AIC-2)

**Severity:2**

**Location:**/conductor.conductor-core.main/java/com/netflix/conductor/metrics/Monitors.java

**CODE WITH VULNERABILITY:**

**private static void gauge(**

**String className, String name, long measurement, String... additionalTags) {**

***getGauge*(className, name, additionalTags).set(measurement);**

**}**

This code above has an issue at: “Set(measurement)” and “long measurement”. Implicit casts from integer data types to floating point data types can be dangerous, especially when they are unexpected. This violation pops up in cases where an "int" value is implicitly cast to a "float" value and all cases where a "long" value is implicitly cast to either a "double" value or a "float" value. This can happen when integer values are passed to methods which take floating point arguments and when integer values are assigned to floating point variables. This makes the security of your code more vulnerable, because unexpected numeric results may cause unexpected behavior and create security holes. Making it a possible entry point for attacks.

**Vulnerability 9**

Do not use 'Runtime.exec()' (CERT.FIO07.EXEC-3)

**Severity:3**

**Location:**/conductor.conductor-java-sdk.main/EXT/com/netflix/conductor/sdk/testing/LocalServerRunner.java

**CODE WITH VULNERABILITY:**

**serverProcess = Runtime.getRuntime().exec(command);**

**BufferedReader error =**

**new BufferedReader(new InputStreamReader(serverProcess.getErrorStream()));**

**BufferedReader op =**

**new BufferedReader(new InputStreamReader(serverProcess.getInputStream()));**

This code has an issue at “Runtime.getRuntime() . exec()”. This issue is that the use of non-portable commands may cause unexpected behavior and have unintended security implications. This is similar to a few of the other vulnerabilities in this report, most of the time something like this might be ok, but sometimes it can cause unintended security implications making it dangerous to use and must be used with caution.

**Vulnerability 10**

Make all member classes "private" (CERT.OBJ08.INNER-3)

**Severity:3**

**Location:**/conductor.conductor-common.main/java/com/netflix/conductor/common/jackson/JsonProtoModule.java

**CODE WITH VULNERABILITY:**

**protected class AnySerializer extends JsonSerializer<Any> {**

**@Override**

**public void serialize(Any value, JsonGenerator jgen, SerializerProvider provider)**

**throws IOException {**

**jgen.writeStartObject();**

**jgen.writeStringField(*JSON\_TYPE*, value.getTypeUrl());**

**jgen.writeBinaryField(*JSON\_VALUE*, value.getValue().toByteArray());**

**jgen.writeEndObject();**

**}**

**}**

***/\*\****

***\* AnyDeserializer converts the custom JSON representation of an {@link Any} value into its***

***\* original form.***

***\****

***\* <p>{@see AnySerializer} for details on this representation.***

***\*/***

**@SuppressWarnings("InnerClassMayBeStatic")**

**protected class AnyDeserializer extends JsonDeserializer<Any> {**

This code has two issues at “AnySerializer” and “AnyDeserializer”. These are issues because member class “AnyDeserializer”, and “AnySerializer” is not "private". This can cause problems, A nested class is a class that is declared within the body of another class or interface. The use of nested classes is error-prone, because a nested class has access to the private fields of the outer class. If the nested class is declared public, the private fields of the outer class can be accessed by any other class within the package. This creates security issues because the Java language system weakens the accessibility of private members of an outer class when a nested inner class is present, which can result in an information leak. It is a protected class so under this circumstance this code may be ok and protected from information leaking.

Works Cited:

“EXP01-J. Do Not Use a Null in a Case Where an Object Is Required - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/EXP01-J.+Do+not+use+a+null+in+a+case+where+an+object+is+required. Accessed 2 May 2022.

“ENV02-J. Do Not Trust the Values of Environment Variables - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/ENV02-J.+Do+not+trust+the+values+of+environment+variables. Accessed 2 May 2022.

“FIO16-J. Canonicalize Path Names before Validating Them - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/FIO16-J.+Canonicalize+path+names+before+validating+them. Accessed 2 May 2022.

“MSC02-J. Generate Strong Random Numbers - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/MSC02-J.+Generate+strong+random+numbers. Accessed 2 May 2022.

“FIO13-J. Do Not Log Sensitive Information Outside a Trust Boundary - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/FIO13-J.+Do+not+log+sensitive+information+outside+a+trust+boundary. Accessed 2 May 2022.

“ERR00-J. Do Not Suppress or Ignore Checked Exceptions - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/ERR00-J.+Do+not+suppress+or+ignore+checked+exceptions. Accessed 2 May 2022.

“LCK06-J. Do Not Use an Instance Lock to Protect Shared Static Data - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/LCK06-J.+Do+not+use+an+instance+lock+to+protect+shared+static+data. Accessed 2 May 2022.

“NUM13-J. Avoid Loss of Precision When Converting Primitive Integers to Floating-Point - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*,wiki.sei.cmu.edu/confluence/display/java/NUM13-J.+Avoid+loss+of+precision+when+converting+primitive+integers+to+floating-point. Accessed 2 May 2022.

“IDS07-J. Sanitize Untrusted Data Passed to the Runtime.exec() Method - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/IDS07-J.+Sanitize+untrusted+data+passed+to+the+Runtime.exec%28%29+method. Accessed 2 May 2022.

“OBJ08-J. Do Not Expose Private Members of an Outer Class from within a Nested Class - SEI CERT Oracle Coding Standard for Java - Confluence.” *Wiki.sei.cmu.edu*, wiki.sei.cmu.edu/confluence/display/java/OBJ08-J.+Do+not+expose+private+members+of+an+outer+class+from+within+a+nested+class. Accessed 2 May 2022.

‌

‌

‌

‌

‌