Derek McPeak

Douglas Lundin

Parasoft Jtest results and discussion

CSC245

Table of Contents

Page 1 – Title Page

Page 2- Table of Contents

Page 3 – Introduction

Page 4 – Avoid Using Null Pointer Exception

Page 5 - Do not use '==' or '!=' to compare objects

Page 6 - Do not catch 'NullPointerException'

Page 7 - Ensure method and constructor return values are used

Page 8 – Make your classes non-cloneable

Page 9 - Use 'java.security.SecureRandom' instead of 'java.util.Random' or 'Math.random()'

Page 10 - Validate all dangerous data

Page 11- **Do not call the 'printStackTrace()' method of "Throwable" objects**

Appendix A - Conductor in Intellij

Appendix B – Parasoft Jtest results

Introduction

In this report () we will discuss a number of security vulnerabilities found inside the Netflix Conductor. The Netflix conductor is an open source microservice orchestrator where process like getting titles ready for streaming , content ingestion, encoding and deployment. Security in an open-source software(OSS) is difficult for it means it is publicly available, security needs to be of topmost importance because it can be utilized by someone who may not know how to properly verify its security and be a source of attack. In this report, we will be discussing the findings of the Java Static Analysis done by Parasoft Jtest. (Java Static Analysis, N.D.) “Parasoft Jtest verifies Java code quality and checks compliance with security standards (OWASP, CWE, CERT, PCI, etc.) by applying a wide range of static analysis checkers (1000+) and using the most comprehensive set of static code analyzers to go way beyond open source.” We used the security standards of CERT, or specially SEI CERT. Software Engineering Institute(SEI) and their CERT division is (The CERT Division, N.D.) “The CERT Division is a leader in cybersecurity. We partner with government, industry, law enforcement, and academia to improve the security and resilience of computer systems and networks”. They have a list of various guidelines, requirements, and recommendations on how to properly secure different programming languages, we will be referring to Java. The last topic to cover before diving into the modification is the Severity. We will be tacking seven of their recommendations.

Avoid NullPointerException [(CERT.EXP01.NP-1)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.EXP01.NP.html%22)) (Severity 1)

The idea of “null” in Java is, an object has no value. For example, a string with a length of 0 would be considered null. If this a null object is encountered, then java will throw an error called “NullPointerExpection” and this can create an unexpected result. What we want to do instead is proactively check if the string/input is null, this will allow us to decide what we want to do with that null value, giving us control over the software and dictate what we want to do if we encounter a null value. Another example, if a variable is left as null and isn’t checked for, this can allow a bad actor to set that value to what they want, this can be for their own testing, for an attack or to cause something to fail. That would be easily fixed by setting the access modifier of the variable to one that cannot be changed or having some validation process to validate or sanitize the data.

**The code below highlighted in blue**, “taskToScheudule.getInputParameters()” has the possibility of being null, and there is no exception handling for this value being null. If we look through the code, we don’t see any exception handling for this, the solution would be to add an exception handler for this case to prevent the program behaving unexpectedly.

File Name: **/conductor.conductor-core.main/EXT/com/netflix/conductor/core/execution/DeciderService.java**

public List<TaskModel> getTasksToBeScheduled(WorkflowModel workflow, WorkflowTask taskToSchedule, int retryCount, String retriedTaskId) {  
 Map<String, Object> input =  
 parametersUtils.getTaskInput(  
 taskToSchedule.getInputParameters(), workflow, null, null);  
  
 String type = taskToSchedule.getType();  
 TaskType taskType = TaskType.*of*(type);  
  
 // get tasks already scheduled (in progress/terminal) for this workflow instance  
 List<String> tasksInWorkflow =  
 workflow.getTasks().stream()  
 .filter(  
 runningTask ->  
 runningTask.getStatus().equals(TaskModel.Status.*IN\_PROGRESS*)  
 || runningTask.getStatus().isTerminal())  
 .map(TaskModel::getReferenceTaskName)  
 .collect(Collectors.*toList*());  
  
 String taskId = IDGenerator.*generate*();  
 TaskMapperContext taskMapperContext =  
 TaskMapperContext.*newBuilder*()  
 .withWorkflowModel(workflow)  
 .withTaskDefinition(taskToSchedule.getTaskDefinition())  
 .withWorkflowTask(taskToSchedule)  
 .withTaskInput(input)  
 .withRetryCount(retryCount)  
 .withRetryTaskId(retriedTaskId)

Do not use '==' or '!=' to compare objects (CERT.EXP02.UEIC-2) (Severity 2)

The ‘==’ or ‘!=’ are comparators meaning “equals” or “not equals” for the use of comparing objects or variables. This can be for a multitude of reasons from making sure one thing is not exactly the same as something else prevent the allocation of extra resources, and this comparator compares the reference location of objects in memory to see if they point to the same location. For example, there could be two separate variables with the same value as in, String name = John; and String Thename = new String (“John”); Both of these variables have the same name/value but reference two distinct parts in memory. If we wanted to compare them using the ‘==’ it would say they do not match, or “false” since they do not point to the same place in memory, while if we use the .equals() comparator, this compares the actual value of each object, it will come back as “true” since the actual values do match.

**The code below highlighted in blue** shows that the two variables “ restTemplate1” and “restTemplate” are being compared against each other using the ‘!=’ causing their memory references to be compared while we want to using ‘.equals’ this will ensure we are always comparing the actual values are being compared.

File Name: **/conductor.conductor-contribs.test/java/com/netflix/conductor/contribs/tasks/http/DefaultRestTemplateProviderTest.java**

final RestTemplate restTemplate =  
 defaultRestTemplateProvider.getRestTemplate(new HttpTask.Input());  
final StringBuilder result = new StringBuilder();  
Thread t1 =  
 new Thread(  
 () -> {  
 RestTemplate restTemplate1 =  
 defaultRestTemplateProvider.getRestTemplate(  
 new HttpTask.Input());  
 if (restTemplate1 != restTemplate) {  
 result.append("different");  
 }  
 });

Do not catch 'NullPointerException' [(CERT.ERR08.NCNPE-3)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.ERR08.NCNPE.html%22)) (Severity 3)

The ‘NullPointerException’ is a form of ‘Exception’ in java. Exceptions are almost self-explanatory, but we will define them an event that disrupts the normal flow of a program. An example of this would be an input field on a website where you expect them to enter their name but instead, they enter their age. We are expecting a ‘String’ and they give us an ‘Integer’ without an exception handler our program would fail. To prevent this, we use exceptions that the case where something unexpected happens, we can manage it without it disrupting the flow. In the case of the NullPointer, we do not want to ‘catch’ this for this is inefficient and it can be difficult to determine where that *null* occurrence came from.

**The code below highlighted in blue** below shows us checking trying to catch a NullPointerException, to fix this we are remove it and in lieu fix the underlying issue, being making sure any values are not set as null by giving them a default value that is not null.

File Name: **/conductor.conductor-core.main/EXT/com/netflix/conductor/core/utils/Utils.java**

public static void checkNotNull(Object object, String errorMessage) {  
 try {  
 Preconditions.*checkNotNull*(object, errorMessage);  
 } catch (NullPointerException exception) {  
 throw new ApplicationException(ApplicationException.Code.*INVALID\_INPUT*, errorMessage);  
 }  
}

Ensure method and constructor return values are used [(CERT.EXP00.NASSIG-3)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.EXP00.NASSIG.html%22)) (Severity 3)

A method is block of reusable code that has some sort of function, as an example you can have a method called ‘getSquareRoot’ where a user passes in any number, and it will do the arithmetic to get the square root of any number. One approach, the incorrect one, would be accepting the users input then having all the code right after, taking up a bunch of space in the main method reducing readability. Imagine an input then a bunch of code, then another input then a bunch of code etc. The correct way would be to have this method created and when you want to get the square root, you can take in the input, and have a single line of code after, calling the method, giving it the users input.

A return is at the end of the that method you would ‘return’ the users value. In this case, the square root. When a return is not used, we do not know if, what we want to do is actually being done. **The code below highlighted in blue**, you will see the variable “queue” does not have any return. This could allow a bad actor to update the value, we can fix this by adding in a return statement and this will allow us to know what the value is when the method runs.

File Name: **/conductor.conductor-core.main/EXT/com/netflix/conductor/core/events/DefaultEventQueueManager.java**

private void listen(ObservableQueue queue) {  
 queue.observe().subscribe((Message msg) -> defaultEventProcessor.handle(queue, msg));  
 }  
}

Make your classes noncloneable [(CERT.OBJ07.MCNC-5)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.OBJ07.MCNC.html%22))  (Severity 5)

Cloning is as it sounds, it clones something. In java’s case, it close an entire class. A class in java is a group of methods around a general theme. For example, you can have an “animal” class and all its methods will be based around creating an animal, giving fur color, genus, eye color, etc. (SEI CERT OBJ07-J.) "An attacker can misuse the clone feature to manufacture multiple instances of a singleton class, create thread-safety issues by subclassing and cloning the subclass, bypass security checks within the constructor, and violate the invariants of critical data." We can fix this by creating a method that prevents it from being cloned.

No code is currently needed to be removed to prevent this, but code is needed to be added. The below code demonstrates the needed method to be added to the effected classes to prevent them from being cloned. As of right now, hundreds of instances where this code needs to be applied, so I will not list every instance but a record of these are held in the report incase these changes are scheduled to be made, however since this software is open source, it may not be a justifiable change as nothing is needed to be hidden.

public final Object clone() throws CloneNotSupportedException { // FIXED

throw new CloneNotSupportedException();

}

Use 'java.security.SecureRandom' instead of 'java.util.Random' or 'Math.random()' [(CERT.MSC02.SRD-1)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.MSC02.SRD.html%22))  (Severity 1)

Java has a many different ways of obtaining a “random” number. Random numbers are generally used as keys for security to authenticate something access to something. The first two that might come to mind are util.Random and Math.random. The problem with both of these are, they are very easy to predict, and not difficult to discover how they work. Instead, we want to use Security.SecureRandom which uses a superior form of cryptography to encrypt the number or key.

Currently, any instance where java.util.Random is the random number generator that is used and will flag, like before I will not list out the instances, but give one instance of the occurrence. **The code below highlighted in blue** below is the source code of the error.

|  |  |  |  |
| --- | --- | --- | --- |
| File Name:  **/conductor.conductor-contribs.test/java/com/netflix/conductor/contribs/queue/amqp/AMQPObservableQueueTest.java** | | | |
|  |  |  |  |

import java.util.Random;

final Random random = new Random();

To resolve this error, we would want to replace the above code with the below code.

java.security.SecureRandom r = new java.security.SecureRandom();

Validate all dangerous data [(CERT.IDS11.VPPD-2)](javascript:openWin(%22file:/C:/Users/azdmc/Downloads/parasoft_jtest_2021.2.1_win32_x86_64/jtest/rules/doc/CERT.IDS11.VPPD.html%22))

We want to validate all inputs, return values of remote methods, variables, and many other things. For example, if we want to validate an input, we can do this by using a method that checks the input for any unexpected data. If we discover that data, we can then choose what to do from there whether that be deleting only the dangerous part of the input, deleting all of it, logging it, etc.

**The code below highlighted in blue** we see the variable “map” being used before validation. We would want to validate “map” before it’s called for use. A way we can do this by creating a list of expected inputs in this method, and check maps against it before it is used.

File Name: **/conductor.conductor-core.main/EXT/com/netflix/conductor/core/config/ConductorProperties.java**

public Map<String, Object> getAll() {  
 Map<String, Object> map = new HashMap<>();  
 Properties props = System.*getProperties*();  
 props.forEach((key, value) -> map.put(key.toString(), value));  
 return map;  
}

**Do not call the 'printStackTrace()' method of "Throwable" objects (CERT.ERR01.ACPST)**

**PrintStackTrace happens when an “exception is thrown” or an error occurs. This will print out the list of method calls that were being made when the exception was thrown. This can be helpful to identify where the problem is. This however is a double-edged sword because it will let a bad actor know what methods are in the program which can give them an avenue for attack. The only place this is acceptable, is if this is being sent to a log file.**

**The code below highlighted in blue, we see that an exception “ex” is being thrown and not logged. We would want to have it logged instead of being printed to the console.**

|  |  |  |  |
| --- | --- | --- | --- |
| File name: **/conductor.conductor-postgres-persistence.main/java/com/netflix/conductor/postgres/dao/PostgresQueueDAO.java** | | | |
|  |  |  |  |

public void removeWorkflowWithExpiry(  
 String workflowId, boolean archiveWorkflow, int ttlSeconds) {  
 try {  
 WorkflowModel workflow = getWorkflowModelFromDataStore(workflowId, true);  
  
 removeWorkflowIndex(workflow, archiveWorkflow);  
 // remove workflow from DAO with TTL  
 try {  
 executionDAO.removeWorkflowWithExpiry(workflowId, ttlSeconds);  
 } catch (Exception ex) {  
 Monitors.*recordDaoError*("executionDao", "removeWorkflow");  
 throw ex;  
 }

Appendix A

Text

Description automatically generated

Appendix B

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

A picture containing table

Description automatically generated

References

Java Static Analysis. Parasoft. No date. <https://www.parasoft.com/products/parasoft-jtest/java-static>-analysis/#:~:text=Parasoft%20Jtest%20verifies%20Java%20code,go%20way%20beyond%20open%20source.

The CERT Division. Carnegie Mellon University. No Date. <https://www.sei.cmu.edu/about/divisions/cert/>