Group 102

SE 465 - Project

Jonathan Hoi Fung Lai, 20426773, SE465-001, jhflai@uwaterloo.ca

Zhao Xuan Xu, 20428545, SE465-002, zxxu@uwaterloo.ca

Mingxin Wu, 20429597, SE465-002, mxwu@uwaterloo.ca

Part I

a) See main.cpp
b) There are false positives, because first of all, it may be a coincidence that two functions are usually called together in the same function, but may not be necessary. Secondly, there may be false positives because some functions in a pair are being called implicitly within other functions, so that our program thinks the function is not being called when it is.

   Two pairs of false positive pairs are:
   (apr_array_make, apr_hook_debug_show) and
   (apr_array_make, apr_array_push)

   One instance of the false positive in the first pair is in the function ap_hook_default_port. This occurs because our program cannot understand external nodes, one of which may have called apr_array_make.

   One instance of the false positive in the second pair is in the function ap_init_virtual_host, the false positive occurred because the pair are only often placed together because apr_array_push requires apr_array_make, but not vice versa.
c) For this part, we broke down how the call graph is parsed from the input. We did an initial pass of the input text to parse out the function callers and callees. Then, if the option to expand is passed as an argument, we go through a second pass. This second pass would build a new call graph by expanding on each callee for a root function. Then, pairings of the child callees of each root node is build from this extended graph and their support and confidence are calculated just like part A.

Part II

a)

Id #10065: **Bug**
The problem is with line 641 of the BooleanUtils.java file.There should be a break statement following the case for value 3 so if string is of length 3 and also has ‘t’ as the first character, there would not be an index out of range exception on line 647.

Id #10066: **Intentional**
It is best practice to avoid the clone() method in general since it is full of issues.

Id #10067: **Bug**
The bug is on line 290 of nestableDelegate.java. The function “public void printStackTrace(PrintStream out)” should allow users to input their own encoding as a parameter to print the stack trace rather than relying on the default.

Id #10068: **Bug**
The bug is on line 110 of JVMRandom.java. Rather than using “(int)(Math.random() * n)” they should use Random.nextInt(n) for better efficiency

Id #10068: **False Positive**
The class name comparison being falsely considered a bug is only there to check that the classes don’t have different names, a deeper comparison is done afterwards when it is checked that the class has the same name.

Id #10070: **False Positive**
This is the exact same problem as #10068 above: The class name comparison being falsely considered a bug is only there to check that the classes don’t have different names, a deeper comparison is done afterwards when it is checked that the class has the same name.

Id #10071: **Bug**
The bug is on line 614 of BooleanUtils.java. The string comparison should be done with “string.equals(Object other)” function rather than “==”.

Id #10072: **Bug**
The bug is on line 4865 of StringUtils.java. The string comparison should be done with “string.equals(Object other)” function rather than “==”.

Id #10073: **Bug**
The bugs are on lines 385, 389, 393, 397, 401, 405, 409 of DurationFormatUtils.java. String comparisons should be done with “string.equals(Object other)” function rather than “==”;

Id #10074: **Bug**
The bug is on line 485 of DurationFormatUtils.java. The string comparison should be done with “string.equals(Object other)” function rather than “==”.

Id #10075: **Bug**
The bug is on line 649 of Entities.java. Adding low and high may result in an overflow. To fix this replace “(low + high) >> 1” with “(low >> 1) + (high >> 1) + (low & high & 1)”

Id #10076: **Intentional**
This negate function correctly returns an explicit null, even though it should return a Boolean object, because it would not make sense to return either true or false if the input is a null. Thus the code should be left as is.

Id #10077: **Intentional**
This toBooleanObject function correctly returns an explicit null, even though it should return a Boolean object, because a null value is explicitly defined in the parameters, and when the value is equal to this null value, the function should return null. Thus the code should be left as is.

Id #10078: **Intentional**
This is the same problem as #10076: This toBooleanObject function correctly returns an explicit null, even though it should return a Boolean object, because it would not make sense to return either true or false if the input is a null. Thus the code should be left as is.

Id #10079: **Intentional**
This toBooleanObject function correctly returns explicit nulls, even though it should return a Boolean object, because a null value is explicitly defined in the parameters, and when the value is equal to this null value, the function should return null. In addition if the null value is null itself we should also return null. Thus the code should be left as is.

Id #10080: **Intentional**
This toBooleanObject function correctly returns an explicit null, even though it should return a Boolean object, because it would not make sense to return either true or false if

the input does not match any of the strings that could mean true or false. Thus the code should be left as is.

Id #10081: **Intentional**
This toBooleanObject function correctly returns explicit nulls, even though it should return a Boolean object, because a null value is explicitly defined in the parameters, and when the value is equal to this null value, the function should return null. In addition if the null value is null itself we should also return null. Thus the code should be left as is.

Id #10082: **False Positive**
Catching type “Exception” should also catch “RuntimeException” and thus there is no bug here.

Id #10083: **Bug**
The bug is on line 137 of FastDateFormat.java. All fields need to be serializable but mRules is not. It should be made serializable or the implementation of the class needs to change.

Id #10084: **False Positive**
Even though the “key” field may never be read here, someone else may need it in the future, or subclasses may need it.

b)
Coverity did not find any bugs for our program. This is because we followed common programming practices.