SE465 Final Project

Vladimir Kovac
SE465 - 002
20418335
v2kovac@uwaterloo.ca

Nathan Woltman
SE465 - 002
20414750
nhwoltma@uwaterloo.ca

Aayush Rajasekaran
SE465 - 002
20429324
arajasek@uwaterloo.ca

1b)

False positives are inevitable because we can never be completely certain that two functions need to always be called together. You can't logically expand all similar functionality, writing code isn't perfect, so you can't perfectly analyze it.

i) (apr_thread_mutex_lock, apr_thread_mutex_unlock)
E.g. bug: apr_thread_mutex_lock in apr_dbd_mutex_lock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 95.56%

Apr_dbd_mutex_lock is just a call to apr_thread_mutex_lock if you're running multiple threads. It has a matching function apr_dbd_mutex_unlock, so you would not expect the lock version of the function to also perform an unlock. Therefore this is a false positive.

ii) (apr_array_make, apr_array_push)
E.g. bug: apr_array_push in ap_method_list_add, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%

Ap_method_list_add takes an external request as an argument, which already has a allocated list in its allowed_methods member. Since the list is already allocated, it doesn't require apr_array_make, therefore this is a false positive.

1c)
The only difference between the inter-procedural analysis algorithm and the default algorithm is that when the functions in a scope are being recorded, if a function already exists as a scope, that scope's functions are recorded in the new scope instead of the scope function itself. To run the program using inter-procedural analysis, the "--ipa" flag must be passed to pipair (before the support and confidence values if you're passing those as well).

An example of this reducing the false positive confidence is for the pair (apr_array_make, apr_array_push) which has the highest number of bugs. In our improved version, the confidence of the diagnostic tool went down consistently across all bug instances by at least ~3% which is a tangible improvement.

Before inter-procedural analysis.
bug: apr_array_push in ap_add_file_conf, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%

After inter-procedural analysis.
bug: apr_array_push in ap_add_file_conf, pair: (apr_array_make, apr_array_push), support: 44, confidence: 77.19%

2a)

1. Bug
Line 640-641 (in between)
This is because if the string is "tru" (case-insensitive) it will fall through to `case 4` where `str[3]` could be accessed and that will be an error because we know the string is only 3 characters long (since it matched `case 3`).
Fix this by adding a “break;” statement between lines 640 and 641.

2. Bug
line 70, StrBuilder.java
“public class StrBuilder implements Cloneable {“
Fix by implementing the clone function, or remove implementation. In subsequent versions of apache commons Cloneable was removed.

3. False Positive
The tool is warning that an OutputStream is required, but a PrintStream is given, however the PrintStream “out” field is inherited from OutputStream so this is acceptable code. This is just a limitation of the tool.

4. Intentional
Although it might be true that java.util.Random.nextInt(n) is more efficient, if you read the documentation for apache commons, they specifically choose Math.Random based on how it generates its random numbers. Depending on whether this is justified or not we can leave the code, or replace it with util.Random.nextInt(n).

5. False Positive
Although normally checking the class name itself isn’t a good test for class equality, in this case for enum objects it’s acceptable.

6. False Positive
This is the same problem as #5.

7. Intentional
This check for the string true is intentionally done with “==” instead of equals for performance reasons as mentioned in the comments, later checks are made for every character, so actual string comparison is also done. But this is usually bad practice, so it’s a good sanity check.

8. Intentional
This is the same problem as #7. It’s not clear if they did this initial check for performance reasons, although a more thorough check of each character is done later so the function will work regardless. Again this is usually bad practice, so it’s good to have this sanity check.

9. Intentional
The input and the comparator are prepared in such a way that the “==” check will always work. Although normally you shouldn’t check string equality like this, the code shouldn’t be changed because it’s part of their intentional design.

10. Intentional
Again the objects are set up in a way that the “==” operator will capture equality in this context. The code doesn’t need to be changed, but the warning is good for sanity checks since this is an anomalous case.

11. False Positive
Coverity says that calculating (low + high) >> 1 could overflow, however, since low and high are both ints, the maximum they could add up to is $2 * 2^{31} - 1$, and then “>> 1” is the same as dividing by 2, so the maximum result would be $2^{31} - 1$, which is the maximum value that can be stored in an int. It is true that the intermediate maximum value could be $2 * 2^{31} - 1$, which is larger than an int can store, but the compiler will figure out how to store this intermediate value so that the calculation will still work (so it will probably be treated as an unsigned int).

12. Intentional
It’s expected that the program returns null if null is given, however this isn’t typically good practice so this warning is a good sanity check for the programmer, and shouldn’t necessarily be considered a false positive. This code could be refactored in a way which wouldn’t require to return null, but this was a deliberate design decision.

13. Intentional
Same as #12

14. Intentional
Same as #12

15. Intentional
Same as #12

16. Intentional
Same as #12

17. Intentional
Same as #12

18. Intentional
It’s possible that they want to catch all exceptions in this case. This isn’t a false positive since generally it’s bad to catch all exceptions so broadly, so this would be a good opportunity to review the intended functionality.

19. False Positive
The rules field is serializable despite what FindBugs says, it is an inner class of FastDateFormat which implements Serializable.

20. Bug
Line 85 IntHashMap.java
“this.key = key;”

The key field is never read so it's unnecessary to have a key field, it can be corrected by removing it. It's not a large problem, but it is technically a waste.

2b)

1.

The first error detected by Coverity is that the struct doesn't initialize any of the variables, nor do they get initialized by any of the functions it calls. This is a false positive in this case, since this is a generally accepted practice to leave a struct member unitialized and then manually set it. What determines whether this is a false or positive case has to do with implementation details that Coverity can't fully interpret. We could refactor the code to ensure that these members have default values and are only modified with set functions, or constructors, however this would just be programming for the bug detector rather than actually improving the quality of the code, especially if you're just plugging in random default values just to assuage Coverity.

2.

Another error detected is an exception that isn't handled by our code. The exception we're not handling is in the c++ pair code, where you if you try to make a pair of the same object, you get an error saying that you shouldn't make a pair of the same thing. In our case, the implementation details prevent this from happening, but this probably should be fixed, and we can classify this as an actual positive error.