CS 447

Instructor:Lin Tan

Adil Ali

Group G118

20365372

1.1

a. Code and make file in pi directory.

b. Two main reason for the false positive as can be seen below is:

1. The false positive does not take into account intra-procedural analysis. For example in pair (apr\_array\_make, apr\_array\_push) Bug 3, 11, 12, and 13, the analyzer do not take into account another function calls one of function from the pair. As a result, the analyzer thinks that the scope only contains one of the functions from the pair and a high support and confidence which lead to the false positive. In conclusion, overall it is not a bug but a false positive which is due to lack of intraprocedural analysis.
2. Another reason for false positive is that the program detects a pair with high confidence and support but it is not necessary for these two function to be called together as it will not cause the program to fail or behave differently from what is expected. For example, pair 2: (apr\_array\_make, apr\_hook\_debug\_show), it is not necessary to call apr\_hook\_debug\_show as this only aids with displaying debug statements and is not necessary at anytime, only when the decides to display statement to help him debug his program

Pair 1: (apr\_array\_make, apr\_array\_push)

1. Bug 1: Is a false positive , apr\_array\_push is expected to be called based on support and confidence but it is fine that they do not push element to the created array as overall its functionality works properly. Later on in the main, elements will be pushed.
2. Bug 2: Is a false positive , apr\_array\_push is expected to be called based on support and confidence but it is fine that they do not push element to the created array as overall its functionality works properly. Later on in the main, elements will be pushed.
3. Bug 3: Is a false positive as 'apr\_xml\_insert\_uri' calls apr\_array\_push.
4. Bug 4: Is a false positive , apr\_array\_push is expected to be called based on support and confidence but it is fine that they do not push element to the created array as overall its functionality works properly. Later on in the main, elements will be pushed.
5. Bug 5: Is a false positive , apr\_array\_push is expected to be called based on support and confidence but it is fine that they do not push element to the created array as overall its functionality works properly. Later on in the main, elements will be pushed. Later on in the main, elements will be pushed.
6. Bug 6: Is a false positive , apr\_array\_push is expected to be called based on support and confidence but it is fine that they do not push element to the created array as overall its functionality works properly. Later on in the main, elements will be pushed. Later on in the main, elements will be pushed.
7. Bug 7: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
8. Bug 8: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
9. Bug 9: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
10. Bug 10: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
11. Bug 11: Is a false positive as prep\_walk\_cache calls apr\_array\_make
12. Bug 12: Is a false positive as prep\_walk\_cache calls apr\_array\_make
13. Bug 13: Is a false positive as prep\_walk\_cache calls apr\_array\_make
14. Bug 14: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
15. Bug 15: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
16. Bug 16: Is not a false positive , apr\_array\_make is expected to be called based on support and confidence
17. bug: apr\_array\_make in ap\_init\_virtual\_host, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
18. bug: apr\_array\_make in ap\_make\_method\_list, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
19. bug: apr\_array\_make in apr\_xml\_parser\_create, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
20. bug: apr\_array\_make in create\_core\_dir\_config, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
21. bug: apr\_array\_make in create\_core\_server\_config, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
22. bug: apr\_array\_make in prep\_walk\_cache, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 86.96%
23. bug: apr\_array\_push in ap\_add\_file\_conf, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
24. bug: apr\_array\_push in ap\_add\_per\_dir\_conf, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
25. bug: apr\_array\_push in ap\_add\_per\_url\_conf, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
26. bug: apr\_array\_push in ap\_copy\_method\_list, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
27. bug: apr\_array\_push in ap\_directory\_walk, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
28. bug: apr\_array\_push in ap\_file\_walk, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
29. bug: apr\_array\_push in ap\_location\_walk, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
30. bug: apr\_array\_push in ap\_method\_list\_add, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
31. bug: apr\_array\_push in apr\_xml\_insert\_uri, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%
32. bug: apr\_array\_push in set\_server\_alias, pair: (apr\_array\_make, apr\_array\_push), support: 40, confidence: 80.00%

Pair 2: (apr\_array\_make, apr\_hook\_debug\_show)

1. Bug 1: Is a false positive , apr\_array\_make is expected to be called based on support and confidence. However, apr\_hook\_debug\_show does not affect the overall functionality as it is to aid with debugging.
2. Bug 2: Is a false positive , apr\_array\_make is expected to be called based on support and confidence. However, apr\_hook\_debug\_show does not affect the overall functionality as it is to aid with debugging.
3. Bug 3: Is a false positive , apr\_array\_make is expected to be called based on support and confidence. However, apr\_hook\_debug\_show does not affect the overall functionality as it is to aid with debugging.
4. Bug 4: Is a false positive , apr\_array\_make is expected to be called based on support and confidence. However, apr\_hook\_debug\_show does not affect the overall functionality as it is to aid with debugging.
5. bug: apr\_hook\_debug\_show in ap\_hook\_default\_port, pair: (apr\_array\_make, apr\_hook\_debug\_show), support: 28, confidence: 87.50%
6. bug: apr\_hook\_debug\_show in ap\_hook\_http\_scheme, pair: (apr\_array\_make, apr\_hook\_debug\_show), support: 28, confidence: 87.50%
7. bug: apr\_hook\_debug\_show in ap\_hook\_log\_transaction, pair: (apr\_array\_make, apr\_hook\_debug\_show), support: 28, confidence: 87.50%
8. bug: apr\_hook\_debug\_show in ap\_hook\_post\_read\_request, pair: (apr\_array\_make, apr\_hook\_debug\_show), support: 28, confidence: 87.50%

c) Inter-Procedural Analysis

The algorithm is very simple:

1. Read in call graph and create a mapping of graph node and its call functions.
2. Iterate mapping and for each call graph:
   1. If level of expansion required is 0, we do not expand any functions i.e. our original code.
   2. If level of expansion required is 1, we recurse the graph node and if a call function is an existing graph node then we replace the call function with its existing graph node functions and insert into the call graph node set. Note the set in java, always ensure only unique functions are retain and none are repeated.
   3. At level n, for n > 1, repeat step b, n times then continue to step 3.
3. Iterate through call graph expansion, and generate a list of mappings of call functions and their respective graph node. This new list of mappings is very helpful and its property is as follows:

* Support = each mapping size tells us directly the support of a given call functions.
* Support (function1, function2) = the size of intersection of two mappings sets is the support of the two functions.
* Confidence (function1, function2) = the intersection of the two mappings divided one of the two mappings tells us its confidence.

1. Iterate the mapping from part 3 twice, and if function1!= function 2 , support >= T\_Support, confidence >= T\_Confidence then
   1. Iterate through the set difference of function1 and function2 and print there is a bug as required.

Our algorithm increases precision by analyzing the call graph in many context and levels. For example, false positives from question 1.b as seen below would not have been detected as bugs if we execute level one expansion:

* Bug 3: Is a false positive as 'apr\_xml\_insert\_uri' calls apr\_array\_push.
* Bug 11: Is a false positive as prep\_walk\_cache calls apr\_array\_make
* Bug 12: Is a false positive as prep\_walk\_cache calls apr\_array\_make
* Bug 13: Is a false positive as prep\_walk\_cache calls apr\_array\_make

1.2

a) Coverity Apache Commons:

|  |  |
| --- | --- |
| 10039 and 10028 Dereference null value return | Classification: Bug  Severity: Major  Action: Fix required  Fault: swapPosition(nextGreater(deletedNode, index), deletedNode, index);  Comments: nextGreater checks whether deleteNode is NULL or not so this warning is not an issue as both functions performs check. But the main issue arises when nextGreater set deleteNode to NULL and swapPosition calls NULL.getParent(index).  Fix:  Node x = nextGreater(deletedNode, index)  if(x != NULL){  swapPosition(x, deletedNode, index);  } |
| 10042 and 10041  Volatile not atomically updated | Classification: Bug  Severity: Major  Action: Fix required  Fault: modcount++  Comments: multiple threads can modify count. An intervening thread2 can overwrite any new value written by a thread prior, thread1.  Fix:  Insert at beginning  private final Object modlock = new Object();  then synchronize modcount.  synchronized (modlock) {  modcount++;  } |
| 10040 and 10030 Thread Deadlock | Classification: Bug  Severity: Major  Action: Fix required  Fault:  synchronized (map) {              return get(map).isEmpty();}  Comment: if thread1 holds lock map and thread2 holds lock, and thread 2 require lock map and thread 1 requires lock, then system will be deadlock  Fix:  synchronized (map) {  newMap = getMap(map);  }  Return newMap.isEmpty(); |
| 10038  Thread Deadlock | Classification: Bug  Severity: Major  Action: Fix required  Fault:  **synchronized** (list) {    **return** get(expected).indexOf(o);}  Comment: if thread1 holds lock map and thread2 holds lock, and thread 2 require lock map and thread 1 requires lock, then system will be deadlock.  Fix:  synchronized (list) {  newlist = get(expected);  }  Return newlist.indexOf(o); |
| 10037 and 10031  Derefernce null value return | Classification: bug  Severity: Major  Action: Fix required  Fault: AVLNode movedNode = getLeftSubTree().getRightSubTree();  Comments: If getRightSubTree returns null then getLeftSubtree will have a null method call which will cause it to crash. |
| 10036 and 10029  Check of thread-shared field evades lock acquisition | Classification: Intentional  Severity: Minor  Action: Ignore  FastTreeMap.**this**.remove(lastReturned.getKey()); is not overwritten as it is set to a fixed value of NULL and it cannot be overwritten by two threads or concurrently as the developer throws an exception if there is a concurrent modification at line 661. This is bad practice as the developer should be consistent and use lock as done elsewhere. |
| 10035 and 10028  Unguarded write | Classification: Bug  Severity: Major  Action: Fix Required  Fault: last--;  Comments: Developer forgot to add guard to protect write to global variable last. Multiple threads can modify last. An intervening thread2 can overwrite any new value written by a thread prior, thread1.  Fix:  Insert at beginning  private final Object lastlock = new Object();  synchronized (lastlock) {  last--;  } |
| 10034 and 10032  Check of thread-shared field evades lock acquisition | Classification: Bug  Severity: Major  Action: Fix Required  Fault:  bucket++;  Comments: Multiple threads can modify bucket. An intervening thread2 can overwrite any new value written by a thread prior, thread1. This can cause a miscount and misexecution of the while loop.  Fix:  **synchronized** (m\_locks[bucket]) {  **while** (bucket < m\_buckets.length) {  .............}  } |
| 10033  Arguments in wrong order | Classification: False Positive  Severity: Minor  Action: Ignore  Comments: Developer intended this to create an inversion map based on his comments |
| 10026  Unguarded Read | Classification: False Positive  Severity: Minor  Action: Ignore  Comments: Developer always acquires the list lock before getting an item from it so no need for a lock to be acquired here as it will lead to a deadlock. |
| 10025  Dereference null value return | Classification: False Positive  Severity: Minor  Action: Ignore  Comments: findNext check f parameters are Null earlier so considering that parameters are not null, I do not believe it can dereference a null value assuming transformer.transform(root) does not produce NULL |
| 10024, 10023 and 10022  Missing call to superclass | Classification: Intentional  Severity: Minor  Actions: Ignore  Comments: It is not a bug but it adds to inconsistency and is bag practice as it add inconsistency to code.  Fix:    Set set = **super**.keySet();  **return** UnmodifiableSet.decorate(set); |

1.2 b)

As seen out of the three found, 1 is a bug and 1 is intentional. The 1 bug where I did not close the stream was actually a common practice of mine and I only learnt or realize that I should always close the stream. The reason why I have never realize this bug is because the try and catch is normally a standard use in Java when reading a file and I never considered an exception occurring because there is never one that occur when I read in file.

The 2nd bug was intentional and this is not a bug but is consider bad practice by some as no checks are perform on the array when line is split by ’. I did not perform the check because I knew the string always follow that format due to if condition. However after researching, I notice that it is considered bad practice by some to not perform some check on the split string so that accessing the index would not be out of bounds.

Using coverity on my code I found the following:

|  |  |
| --- | --- |
| 10211 [GC: Suspicious calls to generic collection methods](http://ecelinux5.uwaterloo.ca:8080/docs/en/cov_checker_ref.html#static_checker_FB) | Line:  callee = strLine.split("'")[1];  This is not a bug. It is intentional as it is bad practice. I should:   Callee1[] = strLine.split("'");  If(Callee1.size() != 0) {  callee = callee1[1];  } |
| 10210 [Dm: Dubious method used](http://ecelinux5.uwaterloo.ca:8080/docs/en/cov_checker_ref.html#static_checker_FB) | This is clearly a false positive as the assumption of the encoding is always true as we generate our call graph. |
| 10209 [Resource leak on an exceptional path](http://ecelinux5.uwaterloo.ca:8080/docs/en/cov_checker_ref.html#static_checker_RESOURCE_LEAK) | This is a bug. If an exception occurs while reading, the stream will never occur.  Fix:  } catch (Exception e) {  e.printStackTrace();  } finally {  br.close();  } |