University of Waterloo

ECE 653 Project Report

Group8

Sihem Romdhani (ID 20489875) sromdhan@uwaterloo.ca
Yanxin Wang (ID 20439686) y574wang@uwaterloo.ca
Shaocong Ren (ID 20478300) s7ren@uwaterloo.ca
Bing Lu (ID20453835) b27lu@uwaterloo.ca

Part (I) Building an Automated Bug Detection Tool

(a)

See source code comment.

(b)

Reasons of false positives:

There are several reasons leading to false positive in our program.

First is that thresholds for support and confidence are low and fixed in our program, not less than 3 and 65%. As long as the thresholds are improved higher then less bugs would be output. For example, for the Apache program if we set the requirement as support>=3 and confidence>=65 we will get as many as 205 bugs but if we set support>=10 and confidence>=80, the number of bugs will be reduced to 34. This result proves that low and fixed thresholds cause some false positive.

Secondly, intra-procedural analysis is another reason for the false positives, since in this kind of analysis there is no need to expand callees in each function. Specifically, if only one member of a pair is called be a caller directly, the other member is called by third callee of this caller, in this way this pair does not appear in this calling list directly, but both of two members are called any way. For example, A and B is a pair, A and scope2 are called by scope1 directly and B is called by scaope2, so A will be a bug in scope1 as long as it satisfy the requirement of support and confidence. However, if we expand the scope2 then AB pair will appear in scope1's calling list and it would be a regular pair.

Third reason is that there is no relation between members of pair, they are playing totally different roles using different parameters returning different values calling and called by different functions. The scenario they appearing together is only by coincidence, which leads to false positive, but this kind of false positive is difficult to find, the tester must be very familiar to the source code and know the roles of each function.

Finally, our target code is Apache so Mozilla bug database should be used in our analysis. After the previous three steps analysis all the false positives should be checked in this bug database, if someone does not appear in the database the possibility of a false positive will be higher.

Identified false positives:

bug: ap_ssi_parse_string in handle_flastmod , pair: (ap_ssi_parse_stringstrcmp), support: 5 , confidence: 71.43%

bug: apr_memcache_hash in apr_memcache_multgetp , pair: (apr_memcache_hashms_bad_conn), support: 4 , confidence: 80%

bug: ap_ssi_get_tag_and_value in handle_flastmod , pair: (ap_ssi_get_tag_and_valuestrcmp), support: 6 , confidence: 75%

bug: apr_memcache_find_server_hash in apr_memcache_multgetp , pair: (apr_memcache_find_server_hashms_bad_conn), support: 4 , confidence: 80%

bug: get_server_line in apr_memcache_multgetp , pair: (get_server_linems_bad_conn), support: 7 , confidence: 87.5%

bug: ap_set_config_vectors in ap_walk_config_sub , pair: (ap_getword_confap_set_config_vectors), support: 3 , confidence: 75%

bug: ap_ssi_get_tag_and_value in handle_elif , pair: (ap_ssi_get_tag_and_valueap_ssi_parse_string), support: 6 , confidence: 75%

bug: apr_memcache_disable_server in apr_memcache_multgetp , pair: (apr_memcache_disable_serverms_bad_conn), support: 5 , confidence: 83.33%

bug: apr_signal in sig_coredump , pair: (__errno_locationapr_signal), support: 3 , confidence: 75%

bug: apr_file_flush_locked in apr_file_gets , pair: (__errno_locationapr_file_flush_locked), support: 7 , confidence: 70%

bug: ap_ssi_get_tag_and_value in handle_if , pair: (ap_ssi_get_tag_and_valueap_ssi_parse_string), support: 6 , confidence: 75%

bug: apr_md5_init in ap_md5digest , pair: (apr_md5_final apr_md5_init), support: 3 , confidence: 75%

bug: ap_create_per_dir_config in ap_init_virtual_host , pair: (ap_create_per_dir_configap_getword_conf), support: 4 , confidence: 80%

bug: ap_ssi_get_tag_and_value in handle_fsize , pair: (ap_ssi_get_tag_and_valuestrcmp), support: 6 , confidence: 75%

(c)

Inter procedural analysis is one solution to reduce false positives. When expanding the called functions the bug may disappear form the bug analysis because the missing function may appear. Based on this we are trying to improve our bug analysis, provide a set of solutions and discuss each one.

First solution:eliminate false positives from the initial bug report by expanding the functions only one level.

To implement this solution, we have stored the list of initial bugs that we have gotten from our first intra-procedural analysis. Each bug has this format: error (the method that appear alone), the location of the bug, and the pair. Then, for each bug we get the list of thecallee related to the location, we parse this set, expand each called method only one level and test if the missing method of the pair appears or not: if it appears so this bug is false positive and eliminate it from the list of bug, else we display it.(`falsePositiveFirstLevel()`).

Although this solution may reduce the number of false positive, it has several drawbacks because it is based only on the bug report and did not analyse the program again after function expansion:

- It ignores the fact that when expanding functions the error may appear in some locations where it never appears before;
- The support value of each method may change as well as the confidence value. In this case new bugs may appear. On the other hand some other bug would not appear due to the changes of values so they may not satisfy the condition 3 for support and 65 for confidence.
- Furthermore, it should be better to expand until the deep level of each function (the deep level is when a function has no callee)

Example:

```
void scope1() {
A(); B(); C();}

void scope2() {
A(); C(); D();scope1();}

void scope3() {
A(); B(); B();}

void scope4() {
B(); scope1();}

void scope5() {
C(); D(); A();}

void scope6() {
B();scope2();}
```

void scope7() {
C();scope2();}

When we run the first bug analysis we get this result:

```
bug: A in scope3, pair: (A C), support: 3, confidence: 75.00%
bug: C in scope7, pair: (A C), support: 3, confidence: 75.00%
```

And when we run our first solution we get this result

```
bug: A in scope3 , pair: (A C), support: 3, confidence: 75%
```

The bug C in scope7 is deleted because when we go to the location scope7 and expand its callee only one level we found that scope2 calls C.
However, analysing only the first level and also keeping the same values of support and confidence may reduce the number of false positive but it ignores the fact that new bugs may appear

- Take the example of the pair (A, B)
- In the first analysis support of A is 4 , support of B is 4 is and the support of (A B) is 2 then we have confidence {(A,B), (A) }= 50% and confidence{ (A,B),(B) = 50% } → these values did not respect the condition confidence >= 65%
- If we expand function, support of A is 7 , support of B is 6 and the support of (A B) is 6 then we have confidence {(A,B), (A) }= 85,71 % and confidence{ (A,B), (B) = 100% } → these values respect the condition confidence >= 65%

  Thus we should get this bug:

```
bug: A in scope5, pair: (A B), support: 6, confidence: 85.71%
```

Due to the previous reasons, we have implemented the second solution that aims to expand all method until the deep level then repeat the analysis of support of confidence of each pair initially created.

Second solution: expanding the deep level of each function and repeat support and confidence analysis

To implement this solution we used the recursive approach to expand functions: In the Method `createCompleteProgram()`we parse our program and we call the method `createCompleteMethodSet()` that aims to expand each function and the stop condition is when a function has an empty set of callee. And in the method `analyseDeepLevel()`we clear the previous analysis and bugs and we redo a new analysis based on the new program.

Then when running this solution on our example we get this result:

```
bug: A in scope5, pair: (A B), support: 6, confidence: 85.71%
bug: B in scope3, pair: (B C), support: 5, confidence: 83.33%
bug: C in scope5, pair: (B C), support: 5, confidence: 83.33%
bug: A in scope3, pair: (A C), support: 6, confidence: 85.71%
bug: C in scope4, pair: (C D), support: 4, confidence: 66.67%
bug: C in scope1, pair: (C D), support: 4, confidence: 66.67%
```

This solution reduce false positive (`bug: C in scope7 is eliminated`)but new bugs appear.However it did not take into consideration the fact that new pairs may be created after function expansion.

Example: When we expand scope2 in Scope6 new pair (B D) is created that didn't appear before.

To improve this solution we need to rebuild pairs after function expansion then repeat the analysis.

Third solution: In this solution we aim to expand the deep level of each function, rebuild pairs, and repeat support and confidence analysis to display the bug report.

To implement this solution we create the method `analyseDeepLevelNewPair()`in which we clear the previous analysis and bugs as well as the previous set of pairs and we recalculate our pairs and redo the analysis based on the expanded functions.

Thus, we get this result:

```
bug: A in scope5, pair: (A B), support: 6, confidence: 85.71%
bug: B in scope3, pair: (B C), support: 5, confidence: 83.33%
bug: C in scope5, pair: (B C), support: 5, confidence: 83.33%
bug: A in scope3, pair: (A C), support: 6, confidence: 85.71%
bug: C in scope4, pair: (C D), support: 4, confidence: 66.67%
bug: C in scope1, pair: (C D), support: 4, confidence: 66.67%
bug: D in scope5, pair: (B D), support: 3, confidence: 75.00%
```

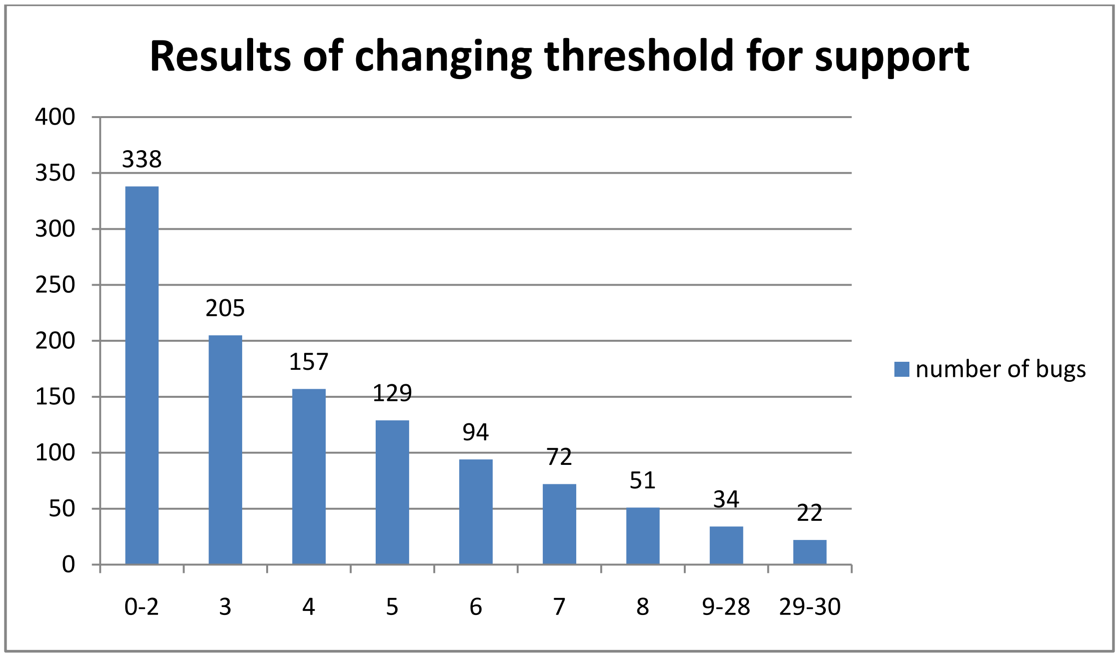
`bug: D in scope5, pair: (B D)`is a new bug because in the previous solution we didn't recreate pairs after function expansion we considered only the initial pairs.

- ➔ We assume that the third solution is the best one because all function are expanded, all possible pairs are created and the analysis is done on a complete program.

(d)

Another solution to reduce false positives is to increase the support. The reason of this is that, as described above in part(b), low support indicates the combination of functions is weak, since they only appear together for very limited times. To implement this solution, the only thing needs to be changed is the argument representing the support. If the threshold of support is increased, there will be less false positives. To test this solution, several test cases with different support thresholds were conducted on the same callgraph with the same confidence threshold. In these tests, support threshold ranges from 0 to 30 with confidence threshold equals to 65. All tests are based on callgraph generated from test3. The numbers of bugs in the reports are:

support	0-2	3	4	5	6	7	8	9-28	29-30
# of bugs	338	205	157	129	94	72	51	34	22



From the above results, we can easily find out that increasing the threshold of support will decrease the total number of bugs found in program. The remaining question is that are those eliminated bugs false positives or at least some of them are? By comparing and analyzing the bug reports, the answer to this question is yes, most of they are false positives by definition. Therefore, increasing the support is a practical solution to reduce false positives.

One solution to find more bugs, on the other hand, is to decrease the support. As tables shown above, when we decrease the threshold of support, we find more bugs in total. Even though most of them may be false positives, we always have the probability of finding a real bug. The reason for this is that we analysis in a statistical perspective. The fact that a pair of functions appears several times does not necessarily mean that they got some functional relations. Therefore, missing one of them may not be a bug. However, even a pair of functions may appear only once in the program, they probably have some functional relations. This is why we still can find more bugs when decreasing the threshold of support.

Part (II) Using a Static Bug Detection Tool

(a)

REVERSE_INULL proposed error 1:
Bug.
/tomcat6/java/org/apache/catalina/ha/session/DeltaRequest.javaLine 238-241

```
        if (actions == null)
            actions = new LinkedList();
        else
            actions.clear();
```

Because *actions* is an attribute of class *DeltaRequest* and it's defined as:

```
   private LinkedList actions = new LinkedList();
```

at the beginning, what's more, there is no other assignment statement towards actions; so, the actions can't be null and the check against null is unnecessary.

Possible fix: just remain the last statement, namely *actions.clear();*, of the above code snippet.

REVERSE_INULL proposed error 2:
Bug.
/tomcat6/java/org/apache/catalina/ha/deploy/FarmWarDeployer.java Line 174-176

```
        if (engine != null) {
            configBase = new File(configBase, engine.getName());
        }
```

engine can't be null. So the checking against null is unnecessary.
The reason is that engine is defined as *Engine engine = (Engine) econtainer;*, in which *econtainer* can't be null, either. This is because the block of code shown below locates at Line 151-154 before *Engine engine = (Engine) econtainer;* if econtainer is null, the function has already returned and will never reach the above assignment statement. Please refer to FORWARD_NULL proposed error 5.

```
        if(!econtainer instanceof Engine) {
            log.error("FarmWarDeployer can only work if parent of " +
host.getName()+ " is an engine!");
            return ;
        }
```

Possible fix: discard the check against null for *engine*. Just remain `configBase = new File(configBase, engine.getName());` in first code snippet is enough.

REVERSE_INULL proposed error 3:
Bug.
/tomcat6/java/org/apache/jasper/compiler/JspDocumentParser.java Line 294 in method

"startElement(String uri, String localName, String qName, Attributes attrs)"

```
        if (attrs != null) {
```

attrs is overchecked against null.
After checking call hierarchy, methods that call this startElement() method will give *m_attributes* as parameter. *m_attributes* is defined in *com.sun.org.apache.xml.internal.serializer.serializerBase.class* as *protected AttributesImplSerializer m_attributes = new AttributesImplSerializer();* which can't be null.

Possible fix: removing *if (attrs != null)* and its corresponding curly bracket.

NULL_RETURNS proposed error 1:
False positive. In line 612, variable *session* is assigned to the return value of function *readSession*, which can't be null. This is because in the body of function *readSession* at Line 424, *session* has been checked against null. If it's null, then it will be assigned a value. So *session*, the return value of function *readSession* won't be null.

NULL_RETURNS proposed error 2:
False positive. `normContext.getContextPath()` can't be null. This is because in this method at Line 411, normContext has been checked against null. If it's null an exception will be thrown and `normContext.getContextPath()` won't reach.

NULL_RETURNS proposed error 3:
False positive. Array *files* can't be null. This is because it's assigned as *File[] files = dir.listFiles();* at Line 192. *dir* is the argument of function *scanForTlds()* which is called at Line 105 as `scanForTlds(file);`. The argument *file* can't be null, either; this is because it's assigned at Line 100 as *File file =* ***new*** *File(tkn.nextToken(), "META-INF");*

NULL_RETURNS proposed error 4:
False positive. In this statement `String strCmd = parseCmd(command);` function *parseCmd()* will return null only if *command* is null. However, *command* is defined as `StringBuffer command = new StringBuffer();` which won't be null and there is no assignment that assigns *command* to null. So, checking against null is unnecessary.

GUARDED_BY_VIOLATION proposed error 1:
False positive. In this statement *return instance;*, the value of *instance* won't be modified; so this operation is unnecessary to be guarded by lock.

GUARDED_BY_VIOLATION proposed error 2:
False positive. In this statement *if (addDeltaRequest && (deltaRequest != null))*, the value of *deltaRequest* won't be modified; so this operation is not necessary to be guarded by lock.

GUARDED_BY_VIOLATION proposed error 3:
False positive. In this statement *return members;* the value of *members* won't be modified; so this operation is not necessary to be guarded by lock.

GUARDED_BY_VIOLATION proposed error 4:
False positive. In this statement *return EMPTY_MEMBERS;* the value of *EMPTY_MEMBERS* won't be modified; so this operation is not necessary to be guarded by lock.

FORWARD_NULL proposed error 1:
False positive. *strValue* can't be null at Line 331. This's because at Line 191-197, if String *s* is null, then the variable *type* can't be assigned as *T_STR*. So, if *case T_STR* is true at Line 329, s can't be null; in this situation, even if *strValue* is null, "*return false"* will be executed at Line 330 rather than the statement at Line 331.

FORWARD_NULL proposed error 2:
False positive. *preparedRemoveSql* and *dbConnection* can't be null, this is because they have been checked against null before dereferenced.

FORWARD_NULL proposed error 3:
False positive. Variable *message* won't be null. This is because method *public void messageReceived(ClusterMessage message)* is called by *public void messageReceived(Serializable message, Member sender)* which is called by *public void messageReceived(ChannelMessage msg)* in */tomcat6/java/org/apache/catalina/tribes/group/GroupChannel.java* where *msg* has been checked against null.

FORWARD_NULL proposed error 4:
False positive. "*this.wEnv*" in function "*org.apache.jk.common.JniHandler.initNative(java.lang.String)*" won't be null. This is because *wEnv* has been assigned as *this.wEnv=we;* in */tomcat6/java/org/apache/jk/core/JkHandler.java.* "*we*" is a parameter which is assigned as *this* when it's called in */tomcat6/java/org/apache/jk/core/WorkEnv.java*.

FORWARD_NULL proposed error 5:
Bug.
/tomcat6/java/org/apache/catalina/ha/deploy/FarmWarDeployer.java   Line 158-161

```
        try {
            oname = new ObjectName(engine.getName() +
":type=Deployer,host="
                    + hostname);
        }
```

engine may be null. So we need to check it.

Possible fix: modify Line 151 -154 as below

```
        if(!(econtainer instanceof Engine)) {
            log.error("FarmWarDeployer can only work if parent of " +
host.getName()+ " is an engine!");
            return ;
        }
```

RESOURCE_LEAK proposed error 1:
Bug.
/tomcat6/java/org/apache/jasper/compiler/JDTCompiler.java   Line 228-236

```
                private boolean isPackage(String result) {
                    if (result.equals(targetClassName)) {
                        return false;
                    }
```

```
                  String resourceName = result.replace('.', '/') +
".class";
                  InputStream is =
classLoader.getResourceAsStream(resourceName);
                  return is == null;
              }
```

Resource "is" isn't freed before method return.

One possible fix: replacing *return is == null;* by the two lines below.

```
                  boolean isStatus = (is == null);
                 is.close();
                  return isStatus;
```

RESOURCE_LEAK proposed error 2:
False positive. "*props*" is an attribute of object JkMain. The purpose of method LoadPropertiesFile() is to acquire resource; so, the resource shouldn't be closed.

RESOURCE_LEAK proposed error 3:
False positive. The statement at Line 400 *sSocket = ssc.socket();* acquires socket connect and then *eSocket* will be closed in function destroy at Line 518. So, it's unnecessary to close *ssc* in here.

CALL_SUPER proposed error 1:
False positive. In the implementation of */tomcat6/java/org/apache/catalina/tribes/group/ChannelCoordinator.sendMessage(Member[] destination, ChannelMessage msg, InterceptorPayload payload)*, it totally overrides the corresponding method in super class; so, there is no need to call the super class.

(b)

Based on the analysis summary report after run Coverity on code from part (I), in total 5 defects are found, 3 of them are found by CHECKER NULL_RETURNS and the other 2 found by RESOURCE_LEAK. Following are two defects found in the report.

1. NULL_RETURNS Error #2(on object "currentCaller")

In method *ProgramCall.anlyseLLVMoutPut(java.lang.String)* every single lines are read, in the scenario of reading a line with caller after reading a line with callee, calling a method on null object "currentCaller" might occur.

```
CS<0x6b342e0> calls function 'tolower'
Call graph node for function: 'ap_find_list_item'<<0x4ab8630>>    #uses=6
```

In the example above, when program find a caller 'ap_find_list_item' after a callee 'tolower', the object "currentCaller" might be null, at this time reference this object will be a bug.We can initialize the object"currentCaller" before the method call or change the conditional statements leading up to the method call.

2. RESOURCE_LEAKError #2 (on resource “bufferedReader” and “read”)

In method *ProgramCall.anlyseLLVMoutPut(java.lang.String)* every single lines are read, after two assignments “read = new InputStreamReader(new FileInputStream(file), encoding)” and “bufferedReader = new BufferedReader(read);”, resource "bufferedReader" is not closed or saved in function "java.io.BufferedReader.readLine()". Then when read the source “bufferedReader” several times, it may throw java.io.IOException or lead to the variable “bufferedReader” go out of scope leaks the resource it refers to. The same problem will occur on resource “read”.