**ECE 653 Project Report**

**Group 63**

Student Name: Hai Jiang Student No.: 20597659 Email: [h57jiang@uwaterloo.ca](mailto:h57jiang@uwaterloo.ca)

Student Name: Chang Liu Student No.: 20632304 Email: [c392liu@uwaterloo.ca](mailto:h57jiang@uwaterloo.ca)

**Part (1) Building an Automated Bug Detection Tool**

1. **Inferring Likely Invariants for Bug Detection**

I use the code …

1. **Finding and Explaining False Positives**

Reasons for the occurrence of false positives could be:

1. Although two methods have to be paired to realize a target function, one of them may be included in a specific method function that only gives a broader definition associated with this method. In such a case, this specific method function does not need to include the other method, since it is not for the target purpose.
2. Although two methods are paired to realize a target function in some circumstances, one of them can be paired with another method to achieve similar functions. In such a case, a false positive is generated if the new pair is regarded as a bug.

I use the code and hash map data structure in (a) to count and list the bug numbers for all the pairs. Then I figure out which 2 pairs can form 10 locations. As seen below, I list the 2 pairs I found and trace back to the source code in /opt/testing/apache\_src to check if the bug is a false positive.

1. Pair (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock) as shown below, which has totally 6 bug locations.

bug: apr\_thread\_mutex\_lock in apr\_dbd\_mutex\_lock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 95.56%

bug: apr\_thread\_mutex\_lock in apu\_dso\_mutex\_lock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 95.56%

bug: apr\_thread\_mutex\_unlock in apr\_dbd\_mutex\_unlock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 91.49%

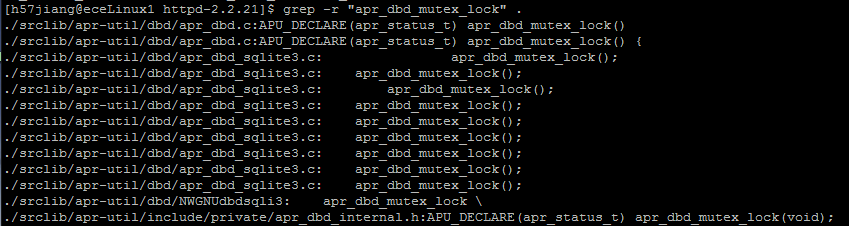
bug: apr\_thread\_mutex\_unlock in apr\_global\_mutex\_trylock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 91.49%

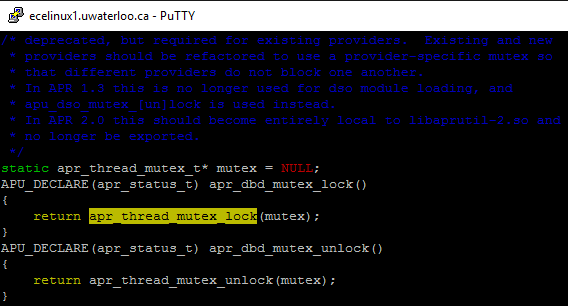
bug: apr\_thread\_mutex\_unlock in apr\_global\_mutex\_unlock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 91.49%

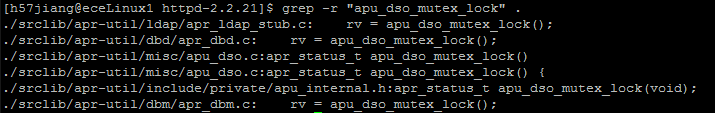
bug: apr\_thread\_mutex\_unlock in apu\_dso\_mutex\_unlock, pair: (apr\_thread\_mutex\_lock, apr\_thread\_mutex\_unlock), support: 43, confidence: 91.49%

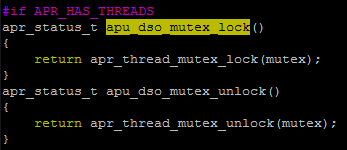
Then I use the linux instruction “grep –r” to find out which file contains the corresponding method. As a result, I check the source code of several locations and I find out some false positives.

As shown in the picture below, I read the source code of apr\_dbd\_mutex\_lock(). Apparently, this method apr\_dbd\_mutex\_lock() is used to define mutex\_lock, by using the apr\_thread\_mutex\_lock. There is no need for apr\_dbd\_mutex\_lock() to include an unlock function. In practical use, apr\_dbd\_mutex\_lock() is called by the other functions to perform a lock operation, and apr\_dbd\_mutex\_unlock() is called to perform an unlock operation. In such case, apr\_dbd\_mutex\_lock() does not need an unlock, and similarly apr\_dbd\_mutex\_unlock() does not need a lock. As such, the associated bugs shown in the report from part (a) is false positives.









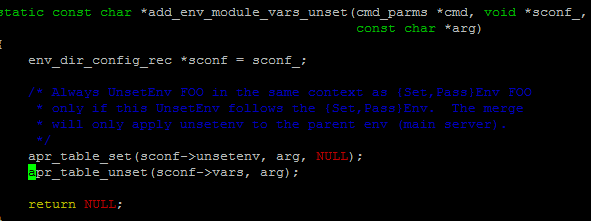
1. Pair (apr\_table\_setn, apr\_table\_unset) as shown below, which has totally 4 bug locations.

bug: apr\_table\_unset in add\_env\_module\_vars\_unset, pair: (apr\_table\_setn, apr\_table\_unset), support: 8, confidence: 66.67%

bug: apr\_table\_unset in ap\_read\_request, pair: (apr\_table\_setn, apr\_table\_unset), support: 8, confidence: 66.67%

bug: apr\_table\_unset in basic\_http\_header, pair: (apr\_table\_setn, apr\_table\_unset), support: 8, confidence: 66.67%

bug: apr\_table\_unset in strip\_headers\_request\_body, pair: (apr\_table\_setn, apr\_table\_unset), support: 8, confidence: 66.67%



Then I use the same method as mentioned above to check the pair (apr\_table\_setn, apr\_table\_unset). As shown in the picture below, I read the source code of add\_env\_module\_vars\_unset. In this function, apr\_table\_unset() follows apr\_table\_set() instead of apr\_table\_setn(). As apr\_table\_set() has already performed a set function, I believe that it is reasonable to have apr\_table\_unset() without a apr\_table\_setn() here.

1. **Inter-Procedural Analysis**

I use the code …

**Part (2) Using a Static Bug Detection Tool**

1. **Resolving Bugs in Apache Commons**
2. **CID 10065**

Missing break in switch (MISSING\_BREAK)

**Classification:**False Positive

**Explanation:**

In “case 3”( BooleanUtils.java from line 628 to line 640), if “ ch == ‘y’/’Y’ ”,it will return the corresponding Boolean vaule after checking whether it equals to “yes” or “YES”.Due to the lack of “break”,it will execution “case4"if it does not satisfy any case in “case 3”,and it will also does not any case of “case 4” since the unequal length.Finally,it will return false,which does not modify the output.

1. **CID 10066**

CN:Bad implementation of cloneable idiom(FB.CN-IDIOM)

**Classification:**False Positive

**Explanation:**

In this class, it implements Cloneable interface,but have clone() method is not called in it.It will waste the space.However,it will not cause the problem.

1. **CID 10067**

Dm: Dubious method used (FB.DM\_DEFAULT\_ENCODING)

**Classification:** Intentional

**Original code**:(NestableDelegate.java line292)

PrintWriter pw = new PrintWriter(out, false);

**Re-factory:**

PrintWriter pw = new PrintWriter((new OutputStreamWriter(out,”UTF-8”), false);

**Explanation:**

Different environments may have different defauly character encodings. you can control the character encoding the characters .should be written in such that they won't eventually end up as gibberish.

1. **CID 10068**

Dm: Dubious method used (FB.DM\_NEXTINT\_VIA\_NEXTDOUBLE)

**Classification:**Intentional

**Original code:**

110        **return** (int)(Math.random() \* n);

**Re-factory:**

110        **return** (int)( Random.nextInt(n));

**Explanation:**

Math.random() requires about twice the processing to generate an integer and is subject to synchronization. Random.nextInt(n) uses Random.next() less than twice on average- it uses it once, and if the value obtained is above the highest multiple of n below MAX\_INT it tries again, otherwise is returns the value modulo n (this prevents the values above the highest multiple of n below MAX\_INT skewing the distribution), so returning a value which is uniformly distributed in the range 0 to n-1.Therefore Random.nextInt(n) is more efficient and less biased.

1. **CID 10069**

Eq: Problems with implementation of equals() (FB.EQ\_COMPARING\_CLASS\_NAMES)

**Classification:**Bug

**Faulty lines:**( Enum.java line 552)

**if** (other.getClass().getName().equals(**this**.getClass().getName()) == false)

**In this code,it wants to determine whether the object other is the same class as this object.However,the equal method just compare these two object if have the same name.Therefore,it can not achieve the goal of the code and is just a bug.**

**Bug fix:** (To compare the class of an object to the intended class, the getClass() method and the comparison operator "==" should be used.)

**if** (other.getClass() == **this**.getClass())== false)

**we just remove getName() method and use comparison operator "==".**

1. **CID 10070**

Eq: Problems with implementation of equals() (FB.EQ\_COMPARING\_CLASS\_NAMES)

**Classification:**Bug

**Faulty lines**:( Enum.java line 598)

**if** (other.getClass().getName().equals(**this**.getClass().getName()) == false)

**In this code,it wants to determine whether the object other is the same class as this object.However,the equal method just compare these two object if have the same name.Therefore,it can not achieve the goal of the code and is just a bug.**

**Bug fix:** (To compare the class of an object to the intended class, the getClass() method and the comparison operator "==" should be used.)

**if** (other.getClass() == **this**.getClass())== false)

**we just remove getName() method and use comparison operator "==".**

1. **CID 10071**

ES: Checking String equality using == or != (FB.ES\_COMPARING\_PARAMETER\_STRING\_WITH\_EQ)

**Classification**:Intentional

**Original code:**(BooleanUnits.java line 614)

614        **if** (str == "true")

**Re-factory :**

614        **if** (str.equals("true"))

**Explanation:**

According to the comments above the method and above this code line, the parameter of method toBoolean(String str) is interned strings, so using “==” to compare str with “true” would not cause problem.

But, to achieve scalability, if the parameter str is not a String constant or interned string, the result of this statement would not be functionally right. Using “equals” is a better way to promise the program function.

1. **CID 10072**

ES: Checking String equality using == or != (FB.ES\_COMPARING\_PARAMETER\_STRING\_WITH\_EQ)

**Classification:**Intentional

**Original code** :(StringUtils.java line 4865)

4865        **if** (str1 == str2)

**Re-factory :**

4865        **if** (str1.equals("str2"))

**Explanation:**

According to the comments above the method and above this code line, the parameter of method indexOfDifference(String str1, String str2)

is interned strings, so using “==” to compare str1 with “str2” would not cause problem.

But to achieve scalability, if the parameter str1 or str2 is not a String constant or interned string, the result of this statement would not be functionally right. Using “equals” is a better way to promise the program function.

1. **CID 10073**

ES: Checking String equality using == or != (FB.ES\_COMPARING\_PARAMETER\_STRINGS\_WITH\_EQ)

**Classification:** False Positive

**Original code:** In DurationFormatUnits.java

1. 409            **else** **if** (value == S)
2. 405            **else** **if** (value == s)
3. 401 **else** **if** (value == m)
4. 397 **else** **if** (value == H)
5. 393 **else** **if** (value == d)
6. 389 **else** **if** (value == M)
7. 385 **else** **if** (value == y)

**Explanation:**

380            Object value = token.getValue();

“value” is a constant which is defined in line 380.And for

else if statement,both the right hand and the left hand

are constant strings,so they can be compared by “==”.

**CID 10074**

ES: Checking String equality using == or != (FB.ES\_COMPARING\_PARAMETER\_STRING\_WITH\_EQ)

**Classification:**False Positive

**Explanation:**

previous.getValue() is constant in the source file, variable value just be assigned to a String value, but not instantiate a new object. So they can be compared using “==”.

**CID 10075**

IM: Questionable integer math (FB.IM\_AVERAGE\_COMPUTATION\_COULD\_OVERFLOW)

**Classification:**Intentional

**Original code :** In Entities.java(line 649)

649                int mid = (low + high) >> 1;

**Re-factory:**

649                int mid = (low + high) >>> 1;

**Explanation:**

If the result of (low + high) is zero or positive,it will not cause the problem.While the result is negative, “>>” will cause the wrong answer.

The >> operator shifts a 1 bit into the most significant bit if it was a 1, and the >>> shifts in a 0 regardless.

(low + high) >> 1 keeps the sign bit of the original, so a negative value for a gives a negative result.

(low + high)  >>> 1 works by introducing a zero sign bit, so the result cannot be negative for any (low + high).

**CID 10076**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

 61*\* @return the negated Boolean, or <code>null</code> if <code>null</code> input*

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10077**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

305*\* @return Boolean.TRUE, Boolean.FALSE, or <code>null</code>*

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10078**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

221*\* @return Boolean.TRUE if non-zero, Boolean.FALSE if zero,*

222*\*  <code>null</code> if <code>null</code> input*

223*\*/*

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10079**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

336*\* @return Boolean.TRUE, Boolean.FALSE, or <code>null</code>*

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10080**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

502     \* <p><code>'true'</code>, <code>'on'</code> or <code>'yes'</code>

503     \* (**case** insensitive) will **return** <code>true</code>.

504     \* <code>'false'</code>, <code>'off'</code> or <code>'no'</code>

505     \* (**case** insensitive) will **return** <code>false</code>.

506     \* Otherwise, <code>null</code> is returned.</p>

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10081**

NP: Null pointer dereference (FB.NP\_BOOLEAN\_RETURN\_NULL)

**Classification:**Intentional

**Solution:**left as-is

**Explanation:**

In general,it is not reasonable to return null,which may lead to null-pointer exceptions.While in the above comments,it is said that:

557*\* @return the Boolean value of the string,*

558*\*  <code>null</code> if no match or <code>null</code> input*

Therefore ,This method allows to return null, So, it’s an

intentional warning. According to the function of the

method, it should be left as-is.

**CID 10082**

REC: RuntimeException capture (FB.REC\_CATCH\_EXCEPTION)

**Classification:**Intentional

**Original code :**In ExceptionUnits.java (line 97)

 97        **catch** (Exception e)

**Re-factory:**

 97        **catch** (RuntimeException e)

**Explanation:**

In Java, there are two types of exceptions: checked exceptions and un-checked exceptions. A checked exception must be handled explicitly by the code, whereas, an un-checked exception does not need to be explicitly handled.

Any exception that derives from "Exception" is a checked exception.

Any exception that derives from "RuntimeException" is an un-checked exceptions.

Generally,throwing a checked exception will not cause the problem.While the method could not be normally handle an un-checked exceptions .  RuntimeExceptions do not need to be explicitly handled by the calling code.

**CID 10083**

Se: Incorrect definition of Serializable class (FB.SE\_BAD\_FIELD)

**Classification:** Intentional

**Original code:**In FastDateFormat.java (line 137)

137    **private** Rule[] mRules;

**Re-factory:**

137    **private transient** Rule[] mRules;

**Explanation:**

By default, all of object's variables get converted into a persistent state. In some cases, you may want to avoid persisting some variables because you don't have the need to persist those variables. So you can declare those variables as transient. If the variable is declared as transient, then it will not be persisted.

**CID 10084**

UrF: Unread field (FB.URF\_UNREAD\_FIELD)

**Classification:**False Positive

**Explanation:**

 85            **this**.key = key;

the key has never been read,and it seems always same with hash and is redundant .But it will not cause the problem.

1. **Analyzing Your Own Code**

1.After running Coverity on my own code for part1 (a),1 error is defected as following:



1. **CID 10090**

Dm: Dubious method used(FB.DM\_DEFAULT\_ENCODING)

Original Code: line 34 in pipair\_java.java



Fixed Code:



Explaination:  Different environments may have different defauly character encodings.This would guarantee that, given an input file that uses encoding "UTF-8", it will be parsed in the same way regardless of what machine you are executing your program in.

2.run Coverity on fixed code, 0 error is defected.



3.the reasons for only finding one error:

1)Maybe there are some errors not documented in the Coverity, and these errors covered in our code can not be defected.

2)As the Coverity is a static analysis tool,some errors ,such as memory leaks and time dependencied, can only be defected by the dynamic tool

**Appendix :**

the output from Coverity:

|  |  |
| --- | --- |
| **Error#1** | |
| **Meta Variable** |  |
| Checker | FB.DM\_DEFAULT\_ENCODING |
| File | /home/c392liu/sa\_output/pipair\_java.java |
| Function | pipair\_java.main(java.lang.String[]) |
| Ordered | true |
| **Event** | |
| **Variable** |  |
| main | true |
| tag | defect |
| description | {CovLStr{v1}{Found reliance on default encoding: {0}.}{new java.util.Scanner(InputStream)}} |
| line | 34 |