**Part(2)-Using a Static Bug Detecttion Tool**

**(a) Resolving Bugs in Apache Commons**

**10065** **False positive**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

This case should be classified as false positive. As the fall through explanation indicates, the consideration for Coverity is that case 3 will falls through to case 4 as if a 'break' statement is lacking, but actually, in case 3 the ‘return’ statement has played the role as same as break, it returns expected midway value and break out of the program.

Within a switch block, each statement group terminate through ‘break’, ‘continue’ , ‘return’ or throwing an exception, as well as an annotation to illustrate how the program will continue to the next statement group, any comments can express this meaning is valid (typically use // fall through). This particular annotation does not necessary to appear in the final group of statements (usually as default).

**10066 Bug**

File: /src/java/org/apache/commons/lang/text/StrBuilder.java

This case is a bug. In order to make the class StrBuilder implements a Cloneable interface, the implementation of the interface checking is necessary. The clone() checks that the class implements Cloneable, only to ensure you don't clone non-cloneables by accident.

The Object.clone() method can validly realize the duplication function, if a clone method for object calling applied to a non-Cloneable interface, CloneNotSupportedException will be occurred.

**10067 Bug**

File: /src/java/org/apache/commons/...ption/NestableDelegate.java

It should be categorized as Bug. This class is assumed that the default character encoding and the default byte-buffer size are acceptable. The specification of these value by constructing an OutputStreamWriter on a FileOutputStream is not accepted here. Further, it is not able to set a charset using FileWriter. As a fixing, we can use OutputStreamWriter instead because it allows us to specify a charset.

**10068 Bug**

File: /src/java/org/apache/commons/lang/math/JVMRandom.java

Random.nextInt(n) is both more efficient and less biased than Math.random(), at the same time Random.nextInt(n) has less predictable output then Math.random(), with considering the efficiency. So according to the definition of bug, this bad practice can be categorized as a bug.

**10069 Bug**

File:/src/java/org/apache/commons/lang/enum/Enum.java

This is a bug. If we determine an object's type based on its name, it will has risk as reusing class. Namely, two distinct classes with same qualified class name can be loaded with different class loaders. In this case, they will be considered the same with same class loader and same name, but it returns to a fault. Two classes with the same name but different package names are distinct.

**10070 Bug**

File: /src/java/org/apache/commons/lang/enums/Enum.java

It is a bug and the reason is the same with the previous one.

**10071-10074:**

This three cases are reported that checking string equality using == or != rather than .equals(). Firstly, the difference between == and .equals() is :== tests for reference equality ,.equals() tests for value equality.

**10071 Bug**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**Line614**:This is a bug，which  may  result  in  a  fault.  Comparing  strings  with  “==”  checks   references  to  the  strings,  not  the  content  of  the  strings.  It may results in unexpected behavour. Refactoring: replace  line:  if  (str ==  "true")  { with  line:  if  (str.equals("true"))  {

**Line 617** :This is a bug as well if the String itself is null, using string.isEmpty()will results to NullPointerException. The safety way to judge whether a String is NULL can be workable as: string ==null || string.isEmpty(). When we have str==null, ut can only indicates no content is defined in str,but not ensure whether it is Null.

**10072 Bug**

File: /src/java/org/apache/commons/lang/StringUtils.java

**Line4865:** It is a bug

Because the string is an object type, so it can't simply use "= =" to judge whether two strings are equal. The “= =” compares both the memory address and the value. So even if the assigned content is the same, different address would also lead to false.

**Line 4868:** It is a bug, which involved in previous null problem. The following codes illustrates the three different assignments.

if(str ==null)

{

// str is assigned with null;

}

if (str.equals(""))

{

//str is empty string;

}

if (str.length()==0)

{

// str is empty string;

}

As the comment in line 4903 shows:

4903*\* StringUtils.getLevenshteinDistance("","")               = 0*

Thus what we need is not null but empty string, the null assignment is a bug.

**10073 Bug**

File: /src/java/org/apache/commons/...me/DurationFormatUtils.java

It is a bug and the reason is the same with the previous one.

**10074 False positive**

File:/src/java/org/apache/commons/...me/DurationFormatUtils.java

This case can be categorized as false positive.

The following methods are valid to ensure the str is not null:

1. str!=null;

2. "".equals(str);

3. str.length()!=0;

Therefore, in line 465, the != can be used.

**10075 Bug**

File: /src/java/org/apache/commons/lang/Entities.java

This case is a bug. The averaging problem between two integers in java can be solved in three ways, let us assume the two integer is a and b.

(a + b) / 2;(a + b) >> 1;(a + b) >>> 1;

The above three functions are able to figure out the average of integer a and b, but it may cause overflowing result when operating with signed integer. Since “>>” is more functional on unsigned integers. In order to avoid this issue, we can use (a & b) + ((a ^ b) >> 1) to get the average.

As we all know, every decimal number can be converted to the corresponding binary number, and can be broken down into the sum of individual bits and product with its weight. For example, for integer 10 and 6:

10 = 1010 (binary) = 1\* 23+ 0 \* 22 + 1 \*21 + 0\* 20 ; 6 = 0110 (binary) = 0 \* 23+ 1 \* 22 + 1 \*21 + 0\* 20

Then, do the sum with the two integer into such a polynomial. Considering the same and different digit in binary sequence**,** we add up number in same digit, the result equals to twice the bitwise AND; for different digit, the result is equal to the Bitwise XOR.

(10 + 6) / 2 <=> ((0010+ 1000) + (0010+ 0100)) >> 1<=> 1000

The above calculation steps for 0010 + 1100 >> 1 are calculated by (10 & 6) + ((10 ^ 6) >> 1). Thus, figuring out average for integers can be calculated by (a & b) + ((a ^ b) >> 1).

**10076-1081 Intentional**

**10076**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**10077**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**10078**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**10079**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**10080**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

**10081**

File: /src/java/org/apache/commons/lang/BooleanUtils.java

The above cases are intentional. This  is  not  a  bug  as  it  is  a  desired  practice  to  return  null  since  the  Boolean

type  can  have  values   Boolean.TRUE,  Boolean.FALSE  or  null. The returning null value will result in null pointer exception as Boolean return type has been stated. This  method   could only  be  used   if  the  return  type  is  primitive boolean  and does not involves null  as  a  valid   value.  This  is  an intentional action.

**10082 Intentional**

File: /src/java/org/apache/commons/l...xception/ExceptionUtils.java

The category of this issue is intentional. Runtime Exception is a system-wide serious exception, which has no necessity to declare in the throw statement, as well as its subclass. Before this issue, we can find the throwable class getMethod is employed, this class can cause exception. Thus this action can be intentional if the exception caused by getMethod is being considered into a kind of runtime exception.

**10083 Bug**

File: /src/java/org/apache/commons/lang/time/FastDateFormat.java

As for the deserialization issue, Findbugs complains that the array mRule is declared in a non-transient and non-serializable way with Rule interface statement. To avoid triggering this kind of problem, we need to ensure the container (type or class) that the object stored in is serializable if we wants the container to be serializable. Therefore, in this case, the array mRule should add in the pertainning information with mMaxLengthEstimate.

**10084 Bug**

File: /src/java/org/apache/commons/lang/IntHashMap.java

This bug is stating that there are fields that never read and need to be remove. By checking the code, we can find that it construct a hashMap, and use Entry as its internal interface, which is supposed to represents the key-value pair in the map. However, the relationship for referring to the next entry in the table is lacking, so it is a bug.