Making Findbugs More Powerful

Mahdi Nasrullah Al-Ameen, Md.Monjurul Hasan, Asheq Hamid

Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, USA
{mahdi.al-ameen, mdmonjurul.hasan, asheq.hamid}@mavs.uta.edu

Abstract—**To find bugs in software, a number of automated techniques have been developed over years. In recent years the research on finding bugs are being considered with utter importance as the automated detection of bugs plays a momentous role to minimize the cost of testing software. Findbugs is a widely used bug finding tool for java that supports plug-in architecture for adding new bug detectors. We have explored the already detected bug patterns and noticed that there are a number of bug patterns that are yet not detected by findbugs. Thus, our research is a momentous step to make findbugs more reliable and effective. We have written bug detectors to detect 8 different bug patterns. Our analysis and experiments have identified 4 bug patterns that are never detectable by findbugs. We have tested our bug patterns with PMD and have found that PMD cannot detect those bug patterns that our bug detectors can detect. We have run a number of popular applications to test the effectiveness of our bug detectors and our results show that our detectors can successfully detect the bug patterns they aim for and the percentage of false positive, reported by our detector is 15.45% that is much less than the percentage of false positive reported by findbugs.**

Keywords. Static Analysis; bug finding tools; findbugs.

I. Introduction

Now-a-days, automated techniques to detect bugs in software are becoming popular to ensure the quality of software at optimized cost. In recent years, researchers have deeply focused on improving the automated bug finding tools that are being used in many industries and organizations. Some of the techniques proposed in the research, require sophisticated program analysis.

Java is a popular programming language and the bugs in a Java program are worthwhile to be detected. Findbugs is the popular bug finding tool for Java; as of September 2006, the official website of findbugs had 270,000 downloads of their bug finding tool [6]. A number of popular companies and industries are currently using findbugs but not all of them give permissions to reveal their identities publicly. ITA Software, Glassfish, ObjectLab, Sleepycat Software are few of those, who have publicly stated that they use findbugs [6]. Realizing the widespread popularity of findbugs, many prominent companies, organizations and institutions are providing financial support for research on findbugs. Google, Sun Microsystem, SureLogic, National Science Foundation, University of Maryland are few of those mentionable names in this respect [6].

Findbugs uses static analysis to inspect java bytecode (compiled class file) for occurrences of bug patterns. A bug pattern is a code idiom that is often an error. The common reasons that may lead to the occurrences of bug patterns in a program include difficult language features, misunderstood API methods, misunderstood invariants when code is modified during maintenance, use of the wrong boolean operator. Findbugs does not need to have the source code as it does static analysis on bytecode. Also it does not need to execute the program that makes the tool very easy to use. Because its analysis is sometimes imprecise, FindBugs can report *false warnings*, which are warnings that do not indicate real errors. In practice, the rate of false warnings reported by FindBugs is less than 50% [6].

Findbugs supports plug-in architecture that allows anyone to add new bug detector. We have explored the bug patterns that are already detected by findbugs. The open source community on findbugs is constantly working to make findbugs more powerful. We have investigated the bug patterns that they have detected to make sure we are not being wasteful with time in detecting a bug pattern that is already detected by the findbugs community.

On going through the already detected bug patterns, we have surprisingly noticed that though findbugs can detect many complex bug patterns but there are many bug patterns, that seem simple, but yet not detected by findbugs. Exploring such weakness of this popular bug finding tool, we have decided to create bug detectors for those bug patterns, before we go for detecting more complex bug patterns. Thus, our research is a very important step to make findbugs more reliable and effective. Our analysis and experiments have identified a number of bug patterns that are never detectable by findbugs. We have revealed these weaknesses/limitations of findbugs that are inherent from its principle of using bytecode for

978-1-4244-9698-3/11/$26.00 ©2011 IEEE

detection. We believe, it is worthwhile to explore its weakness before we step further for making findbugs more powerful.

II. Detected Bug Patterns

We have detected 8 different bug patterns by using our bug detectors that are yet not detected by findbugs. In this section, we will discuss those bug patterns with corresponding examples.

A. Zero length Array

The array, declared with length zero cannot be used to store any data. Figure 1 shows an example of this kind of bug pattern.

```
int[] zero1 = new int[0];
```

Figure 1

B. Negative length Array

When an array is declared with negative length, it carries no meaningful uses in the program. An example of this bug pattern is illustrated in figure 2.

```
int[] zero2 = new int[-5];
```

Figure 2

C. Divide by zero

Findbugs performs static analysis to detect bug patterns and hence does not get the values of the variables in an expression, that come from the user's input in run time. Considering the limitation, to detect 'Divide by zero' bug pattern, we have focused on constants that are already defined in the code segment. An example is shown in figure 3.

```
int a, b = 9,c = 3;
a = b / (b%c);
```

Figure 3

D. Integer Overflow

Referring to the fact, described in section C, we have focused on defined constants to detect 'Integer Overflow' bug pattern.

```
int a2 = 1234567809, b2 = 1234567890;
int c2 = a2 + b2;
```

Figure 4

E. Out of bound array indexing

As we find in figure 5, the value of the variable b5 becomes negative while being used for array indexing. It leads to the occurrence of 'Out of bound array indexing' bug pattern.

```
int[] a5 = new int[5];
int b5 = 0 ;
int c5 = a5[--b5];
```

Figure 5

F. Probable out of bound array indexing

When array.length library function is used in the initialization (figure 7) or in condition checking (figure 6) of a loop, the probability arises to use the value of the length of array as the array-index inside the loop. Thus we have safely used the term: 'probable' to give warning for out of bound array indexing.

```
int[] array2 = new int[5];
int b7;
for(int i = 0 ; i<=array2.length; i++){}
```

Figure 6

```
int[] array9 = new int[5];
int b9;
for(int i = array9.length ; i>=0; i--){}
```

Figure 7

G. Never executed for loop

Figure 8 and 9 illustrate the scenario in which a loop will be never get executed. It is not expected in a program and our bug detector successfully identifies the bug pattern.

```
for(int i = 2; i <= 1 ; i++ ){}
```

Figure 8

```
for(int i = 2; i = = 1 ; i++ ){}
```

Figure 9

H. Unexpected behavior of loop

The loop, illustrated in figure 10 starts from the initialized value of i = 2 and as the value is decreasing, walking through the whole range of negative values of a integer variable, it becomes positive and the loop terminates when the value of i becomes 4. Usually, such iteration in a loop is unexpected and from our experiences, we have never seen such a logic formation through a loop. Another example is focused on figure 11. In both cases, the warnings are generated for the unexpected behavior of loop.

```
for(int i = 2; i <= 3 ; i-- ){}
```

Figure 10

```
for(int i = 2; i >= 1 ; i++ ){}
```

Figure 11

III. Bug Patterns: Never Detectable by findbugs

We have figured out a number of bug patterns, that are never detectable through byte code analysis. As for example, leading zeros (figure 12) are not apparent in the bytecode and thus, 'needless leading zeros (unless it is an octal value)' bug pattern cannot be detected by findbugs.

```
int a4 = 00023;
```

Figure 12

If-else blocks should be within curly braces. From figure 13, considering the indentation, user may expect, else clause corresponds to the first if clause, but according to the language grammar, it corresponds to the second if clause. The programmer might believe this code will result in x=4 but it actually results in x=5. Findbugs can never detect this bug pattern as the curly braces in a source code cannot be figured out from the corresponding bytecode.

```
int x = 5, y = 3;
 if(x = = y)
   if (y= =3)
      x=3;
 else
     x = 4;
```

Figure 13

In the bug pattern, illustrated in figure 14, the user might expect to see *four: 4* as output. But actually it shows *four: 22*. The code will show the desired output if the addition of numerical values is done within parenthesis. In this case, the bytecode does not contain any parenthesis but only the resolved values (i.e. 22). Thus the findbugs cannot detect this bug pattern as from the byte code analysis, it is impossible to guess the expected output.

```
String four = "four: " + 2 + 2;
System.out.println(four);
```

Figure 14

From the code snippet in figure 15, the user might expect to get a6 = 7 as the output, but in reality the output is a6 = 5. User should write "a6+=5" to get the desired output. The bytecodes generated for a6 = 5 and a6 =+ 5 differ no way. So, from bytecode analysis, it is not possible to detect if the user intends to do the addition and mistakenly puts the '+' operator on the wrong side of '=' operator.

```
int a6 = 2;
a6 =+ 5;
 System.out.println(a6);
```

Figure 15

IV. Results and Analysis

A. Comparing with PMD:

PMD is another popular bug finding tool for java that works on source code to detect bug patterns. We have written bug detectors for 8 different bug patterns that the findbugs cannot detect. From experiments we have found that PMD also cannot detect these bug patterns. So, the bug patterns we have worked on are worthwhile to be detected and our research have essentially made findbugs more powerful and effective, in comparison to PMD.

We have identified 4 different bug patterns that findbugs can never detect. In this case, PMD can detect 'needless leading zeros' and 'if statements without curly braces' bug patterns. It is not surprising, as PMD analyzes the source code, it is possible to identify a if statement without curly braces and also the unnecessary leading zeros at the beginning of a literal (unless it is an octal value).

B. Effectiveness of our Bug Detectors:

We have tested our bug detectors for different applications and the result is outlined in table 1. From analysis we find that our bug detectors can successfully detect the bug patterns they aim for. and the average percentage of false warning is 15.45%. The percentage of false warning reported by findbugs is 50% [6]. So, our bug detectors make a significant contribution to reduce the percentage of false warning of findbugs.

Applications	Detected bugs	Percentage of False Positive
jboss-osgi-installer-1.0.0.Beta10	37	21.62%
android-sdk_r10-linux_x86.tgz	873	13.75 %
Spring-security-2.0.4	42	2.38%
hibernate-distribution-3.6.3.final	84	26.19 %
jEdit-4.3.2	30	13.33 %

Table 1: Effectiveness of our Bug Detector

V. Related Work

In recent years, researches on findbugs have got a boost because of its increasing popularity and effectiveness. In [3], a number of bug patterns have been identified that are found in several widely used applications and libraries. The obvious and embarrassing nature of bugs, detected in [3] greatly convinces the researchers for wider adoption of automatic bug finding tools. In [4], five bug finding tools, including findbugs are applied to a variety of Java programs. Experimental results in [4] show that the tools often find non-overlapping bugs. The authors have proposed a meta-tool that combines the output of the tools together.

In [2], authors focus on using findbugs in production software development environments including Sun’s JDK, Eclipse and portions of Google’s Java code base. Google conducted a company wide FindBugs “fixit" in 2009, where hundreds of engineers worked on thousands of FindBugs warnings. They fixed and fled reports against many of those bug patterns. In [1], authors have discussed the learning from this exercise and have analyzed the resulting dataset. In [5] authors have discussed on bug finding tools (findbugs, PMD, QJ Pro) for Java with reviews and tests. They have found that the bug patterns, detected by bug finding tools are the subset of bug patterns, found through review. They have also figured out that dynamic tests detect bug pattern that are completely different than the bug patterns, detected by bug finding tools.

VI. Future Work and Conclusion

We are working on findbugs with motivation of making the tool more powerful. In our research, we have successfully detected 8 different bug patterns that the findbugs cannot detect. We have created corresponding plug-ins to be accommodated with the findbugs that will essentially make findbugs more effective and reliable. Our identified bug patterns that findbugs can never detect are worthwhile in the research on findbugs. In our future work, we will focus on more complicated bug patterns.

REFERENCES

[1] Nathaniel Ayewah, William Pugh, “The Google FindBugs Fixit”, published in the Proceedings of ISSTA’10, 19th international symposium on Software testing and analysis ,2010.

[2] Nathaniel Ayewah, WilliamPugh, J. David Morgenthaler, John Penix, “Using FindBugs On Production Software”, published in the Proceedings of OOPSLA '07, Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, 2007.

[3] David Hovemeyer and William Pugh, “Finding Bugs is Easy”, published in the Proceedings of ACM SIGPLAN, Vol. 39, Issue 12, 2004.

[4] Nick Rutar, Christian B. Almazan, Jeffrey S. Foster, “A Comparison of Bug Finding Tools for Java”, published in the Proceedings of the ISSRE '04, 15th International Symposium on Software Reliability Engineering, 2004.

[5] Stefan Wagner, Jan Jurjens, Claudia Koller, and Peter Trischberger, “Comparing Bug Finding Tools with Reviews and Tests”, http://www.springerlink.com/content/l9y49kxlpu3re752/

[6] http://findbugs.sourceforge.net/

[7] http://fb-contrib.sourceforge.net/