The background is a dark blue gradient. On the left, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram. Below these, there is a circular inset showing a detailed, grayscale image of a circuit board. In the top right corner, there is a faint, stylized pattern of white lines and squares, resembling a circuit or a data structure.

Software Vulnerabilities Static Analysis: Techniques and Tools



Our work

A Comparative Study on Software Vulnerability Static Analysis Techniques and Tools

Peng Li

Institute of Electricity and Information Engineering,
Beijing Institute of Civil Engineering and Architecture,
Beijing, China
E-mail: bicealp@sina.com

Baojiang Cui

Institute of Computer,
Beijing University of Posts and Telecommunications,
Beijing, China
E-mail: cuibj@bupt.edu.cn

Abstract—Using static analysis tools can detect software vulnerabilities, which is important for improving the security of software. Static analysis technology has developed rapidly, but the comparison and evaluation of static analysis techniques and tools are not much. This paper focuses on software vulnerability static analysis techniques and tools. First we discuss the commonly-used static analysis techniques and tools, and compare these tools in a technical perspective, and then we analyze the characteristics of these tools through the experiment, finally, combining dynamic analysis, we propose an efficient software vulnerability detection method.

Williams analyzed the effectiveness of static analysis tools and concluded that static analysis tools are effective at finding loopholes in software [2]; A. Ozment and S. E. Schechter examined the code based on OpenBSD, measured the rate of vulnerability reports, and determined whether the security was improved by using the tools [3]. But few studies have analyzed and evaluated different static analysis tools, so it is quite negative to promote and apply static analysis tools.

Generally, no tool can find all the defects in the software. For example, if we use the tool to find the defects in the code, we can find the defects in the code, but we cannot find the defects in the code.

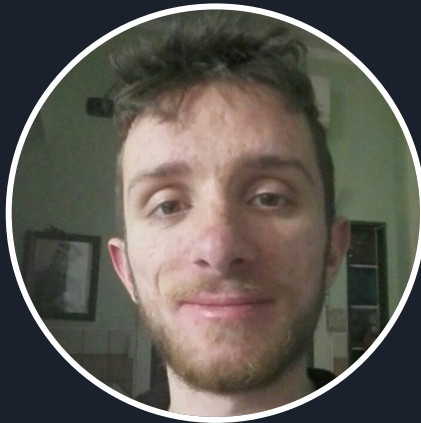


Team members



Lucio Giordano

- Pmd, SpotBugs
- Demo



Carmine D'alessandro

- Introduction
- Static analysis



Simone Faiella

- Static analysis tools
- Vulnerabilities detection



Loophole

When referring to computing, a **loophole** is an error or opening in computer [code](#) that can allow a program to be manipulated or exploited. This term generally comes up when referencing computer or network security.



Loophole

When referring to computing, a **loophole** is an error or opening in computer [code](#) that can allow a program to be manipulated or exploited. This term generally comes up when referencing computer or network security.



We don't want them



Static Analysis

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program.



Static Analysis Pros

Fast

Computationally cheap

Immediate



Static Analysis Cons

Not always effective

Limited efficiency

Limited “by design”



Lexical analysis

Lexical analysis is based on grammar structure analysis, similar to the C compiler. This analysis method is to divide program into several small fragments, and then compares these fragments with loopholes libraries to determine whether there are loopholes. Without considering the program's syntax, semantics and the interaction between subroutines, false positive rate is high.



Type inference

Type inference refers to inferring the type of variables and functions by the compiler, and judging whether its access of variables and functions is in accordance with type rules.

Programming language type system includes a mechanism for defining data types and a set of type rules.



Data flow analysis

Data flow analysis refers to collecting semantic information from the program code, and with algebraic method to determine the definition and usage of the variables at compiling time. By using the control flow graph data flow analysis determines whether a value in the program is assigned to the possible vulnerability of the variables.



Rule checking

Rule checking is to check security of the program using pre-established safety rules. There are some safety rules in program designing, such as in the 'root' privileges if the program calls 'exec' function it will bring on security implications.



Constraint analysis

Constraint analysis is divided into constraint generation and constraint solving in program analysis process. Constraint generation is to establish variable types or analyze restraint system between different states using the rules of constraint generation; constraint solving is to solve the constraint system.



Patch comparison

Patch comparison includes source code patch comparison and binary code patch comparison, and is mainly used to find "known" loopholes. After the software security vulnerabilities are found, the manufacturers usually release corresponding patches, so you can compare the code with patches to determine the location and causes of vulnerability.



Symbolic execution

Symbolic execution is to represent the program's input by using symbol values rather than actual data, and produce algebraic expressions about the input symbols in the implementation process. By constraint solving method symbolic execution can detect possibility of errors.



Abstract interpretation

Abstract interpretation is a formal description of program analysis. Because it only tracks program attributes users concern, the interpretation of semantic analysis is similar to its actual semantic meaning.



Theorem proving

Theorem proving is based on semantic analysis of the program, and can solve problems of infinite state systems.

Theorem proving first converts the program into logic formulas, then proves the program is a valid theorem by using axioms and rules.



Model checking

Model checking process first constructs formal model for the program such as state machine or directed graph, then traverses and compares the model to verify whether the system meets pre-defined characteristics.

Static analysis tools

Here there are seven types of widely-used software vulnerabilities of static analysis tools from open source for analysis and comparison.

Initially a general description will be made, then two tools that has been used within the Eclipse environment will be analyzed and shown.





Static analysis tools

Tools	Technology	Program	Advantages	Disadvantages	Ability to detect
ITS4	Lexical analysis	C/C++	High efficiency, and without pre-treating the source code.	The analysis is imprecise, and false positive rate is high.	Inferior
SPLINT	Rule checking	C/C++	It can find a large scale program's faults using rules.	Due to restricted rules, it can only detect the given faults.	Powerful
UNO	Model checking	C	It can detect complex program, and test strictly.	State space explosion may happen.	Powerful
Checkstyle	Lexical analysis	Java	High efficiency, and without pre-treating the source code.	The analysis is imprecise, and false positive rate is high.	Inferior
ESC/Java	Theorem proving	Java	The strict reasoning is used to control the conduct of testing, and false positive rate is low.	It is different to process automatically, and need career man to analyze; The scalability is not high.	Powerful
FindBugs	Lexical analysis and Dataflow analysis	Java	High efficiency.	The analysis is not precise enough, and applying only to specific problem-solving.	Powerful
PMD	Lexical analysis	Java	High efficiency, and without pre-treating the source code.	The analysis is imprecise, and false positive rate is high.	Inferior

ITS4

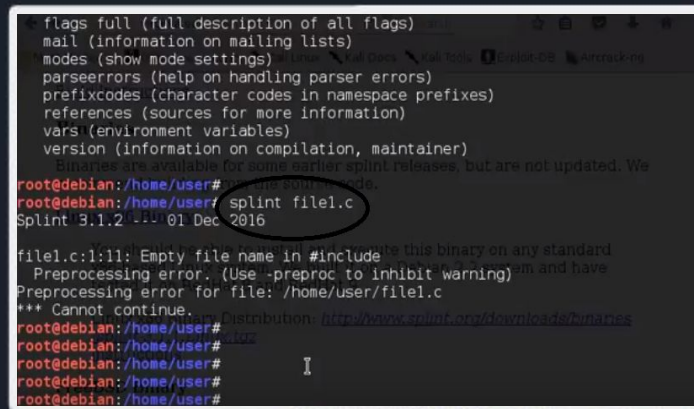
- ITS4 is a simple tool that statically scans C and C++ source code for potential security vulnerabilities.
- It scans source code looking for function calls that are potentially dangerous...
- ...and provides a problem report, including a short description of the potential problem and suggestion on how to fix the code



SPLINT

SPLINT (Secure Programming Lint) is the expansion of LCLINT tool (for detecting buffer overflows and other security threats).

It employs several lightweight static analyses. SPLINT need to use notes to perform cross-program analysis. SPLINT set up models for control flow and loop structure by using heuristic technology.



```
flags full (full description of all flags)
mail (information on mailing lists)
modes (show mode settings)
parseerrors (help on handling parser errors)
prefixcodes (character codes in namespace prefixes)
references (sources for more information)
vars (environment variables)
version (information on compilation, maintainer)
Binaries are available for some earlier splint releases, but are not updated. We
root@debian:/home/user# splint file1.c
Splint 3.1.2 - 01 Dec 2016

file1.c:1:11: Empty file name in #include
Preprocessing error. (Use -preproc to inhibit warning)
Preprocessing error for file: /home/user/file1.c
*** Cannot continue.
Distribution: http://www.splint.org/downloads/binaries
root@debian:/home/user#
```

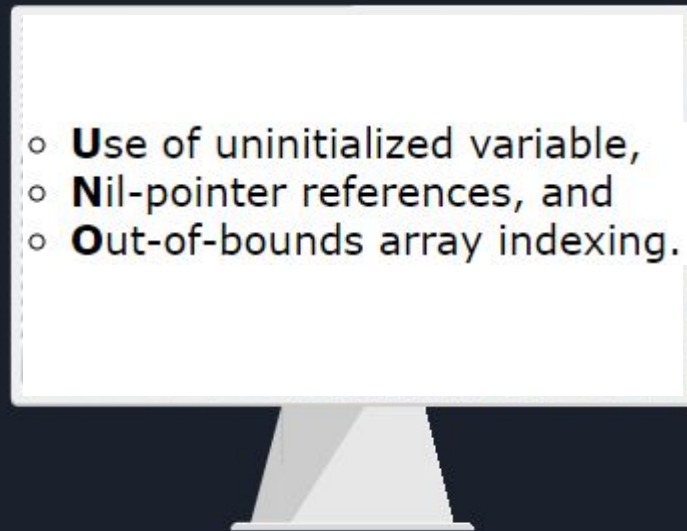


UNO

UNO uses model checking to find loopholes in the code.

UNO is named for the first character of three software defects: the use of Uninitialized variables, dereferencing Nil-pointers, and Out-of-bound array indexing.

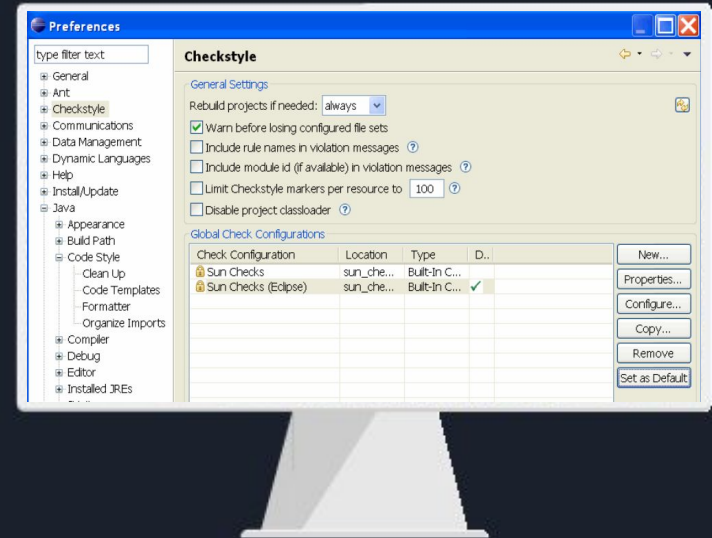
- **U**se of uninitialized variable,
- **N**il-pointer references, and
- **O**ut-of-bounds array indexing.



Checkstyle

Checkstyle is the most useful tool to help programmers write standard Java coding.

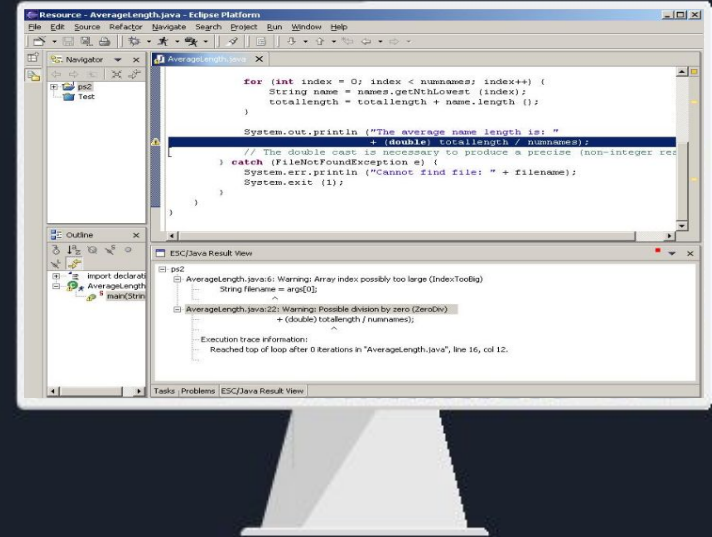
Programmers can integrate Checkstyle in development environment and use it to automatically check whether the Java codes are standard. Checkstyle is configurable and can almost support all the coding standards.



ESC/Java

ESC / Java (Extended Static Checker for Java) is a static detection tool based on theorem proving, and can find run-time error in Java code.

Programmers can build ESC / Java into the program verification environment, or install ESC Java plug-in in the Eclipse.



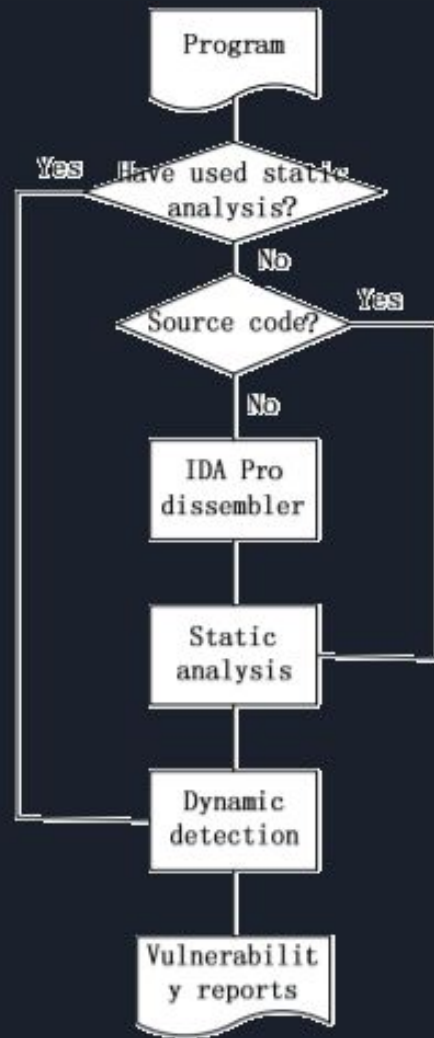
BUG CATEGORY

Bug Category	Example	ESC/Java	FindBugs	PMD
General	Null dereference	✓	✓	✓
Concurrency	Possible deadlock	✓	✓	✓
Exceptions	Possible unexpected exception	✓		
Array	Length may be less than zero	✓		
Mathematics	Division by zero	✓		
Conditional, loop	Unreachable code due to constant guard		✓	✓
String	Checking equality using == or !=		✓	✓
Object overriding	Equal objects must have equal hashcodes		✓	✓
I/O stream	Stream not closed on all paths		✓	
Unused or duplicate statement	Unused local variable		✓	✓
Design	Should be a static inner class		✓	
Unnecessary statement	Unnecessary return statement			✓

SOFTWARE VULNERABILITY DETECTION METHODS

There are two ways to improve the detection efficiency of software vulnerabilities:

- One method is to employ a variety of static test technologies and tools.
- Another method is to combine static analysis with dynamic detection.





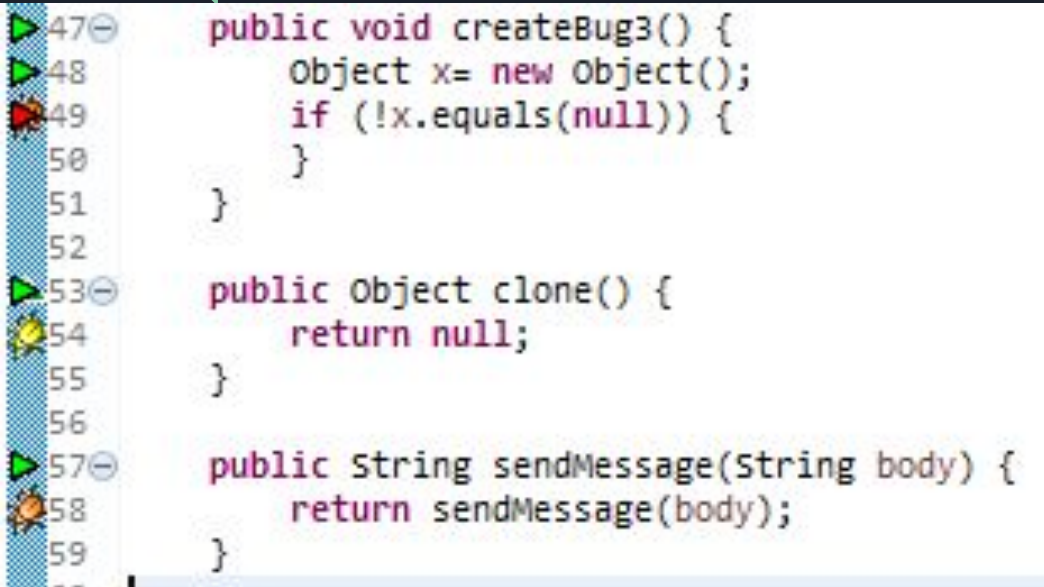
PMD-SpotBugs

Pmd and SpotBugs are static code analyzer.

FindBugs	Lexical analysis and Dataflow analysis	Java	High efficiency.	The analysis is not precise enough, and applying only to specific problem-solving.	Powerful
PMD	Lexical analysis	Java	High efficiency, and without pre-treating the source code.	The analysis is imprecise, and false positive rate is high.	Inferior

- Spiritual successor of FindBugs
- SpotBugs works on **byte code**. It means analyzing the byte code created by the compiler.
- Vulnerabilities are exposed too late. The code needs to be compiled.
- Checks for more than 400 bug patterns
- Pmd works on **source code**. This means analyzing the original un-compiled code.
- Scans code fragments and non-compiled code, allowing near real-time feedback on the code
- Comes with a set of rules to find 'problems'

Bug discovery



The screenshot shows a Java code editor with three methods. On the left margin, there are two types of markers: a red bug icon (SpotBugs) and a yellow arrow icon (Pmd). The bug icon is next to line 49, and the arrow icon is next to line 54. The code is as follows:

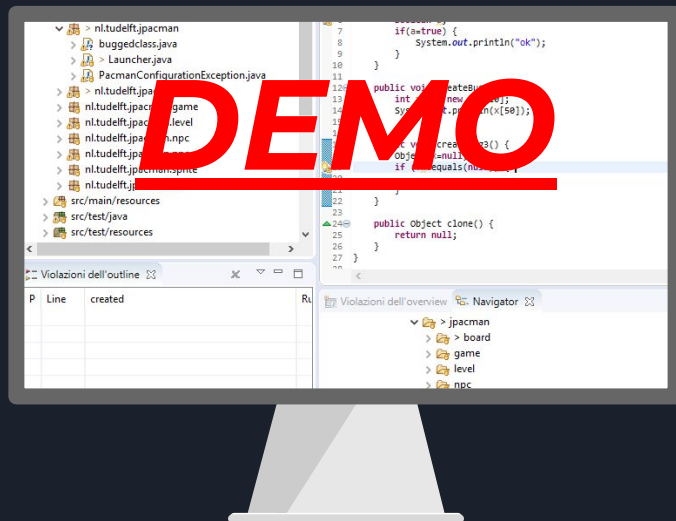
```
47 public void createBug3() {  
48     Object x= new Object();  
49     if (!x.equals(null)) {  
50     }  
51 }  
52  
53 public Object clone() {  
54     return null;  
55 }  
56  
57 public String sendMessage(String body) {  
58     return sendMessage(body);  
59 }
```

This code is horrible but eclipse cannot find bugs or even give warnings.

Through these two tools, we were able to find:

1. the wrong usage of equals, used to compare an element with null
2. clone returning null
3. an infinite recursive loop


- The bug is 'SpotBugs' output
- The arrow is 'Pmd' output





Conclusion



1. Running a static code analyzer is a great idea for java projects. It stops buggy code before it reaches a critical environment
 2. Many tools are open source
 3. Using more tools can help detecting more problems more precisely.
- 

01010100 01101000 01100001 01101110 01101011 00100000 01111001 01101111 01110101 00100001

Thank you!

01010100 01101000 01100001 01101110 01101011 00100000 01111001 01101111 01110101 00100001