

LAB 3.1&3.2

王伟杰

Contents

- 1 Lab 3.1 Using splint for C static analysis**
 - 1.1 Overview
 - 1.2 下载splint
 - 1.3 提取并设置splint
 - 1.4 构造一个至少有两种不同问题的代码
- 2 Lab 3.2 Using eclipse for java static analysis**
 - 2.1 Overview
 - 2.2 Open Source Code Analyzers in Java
 - 2.3 下载PMD插件
 - 2.4 新建Java项目，编写代码
 - 2.5 使用PMD插件运行代码
 - 2.6 使用IDEA自带的代码分析

1 Lab 3.1 Using splint for C static analysis

1.1 Overview

The learning objective of this lab is for students to gain the first-hand experience on using static code analysis tools to check c program for security vulnerabilities and coding mistakes.

Splint [link](#) is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional lint checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases. More powerful checks are made possible by additional information given in source code annotations. Annotations are stylized comments that document assumptions about functions, variables, parameters and types. In addition to the checks specifically enabled by annotations, many of the traditional lint checks are improved by exploiting this additional information.

11 kinds of problems detected by Splint include:

- Dereferencing a possibly null pointer;
- Using possibly undefined storage or returning storage that is not properly defined;
- Type mismatches, with greater precision and flexibility than provided by C compilers;
- Violations of information hiding;
- Memory management errors including uses of dangling references and memory leaks;
- Dangerous aliasing;
- Modifications and global variable uses that are inconsistent with specified interfaces;
- Problematic control flow such as likely infinite loops, fall through cases or incomplete switches, and suspicious statements;
- Buffer overflow vulnerabilities;
- Dangerous macro implementations or invocations;
- Violations of customized naming conventions.

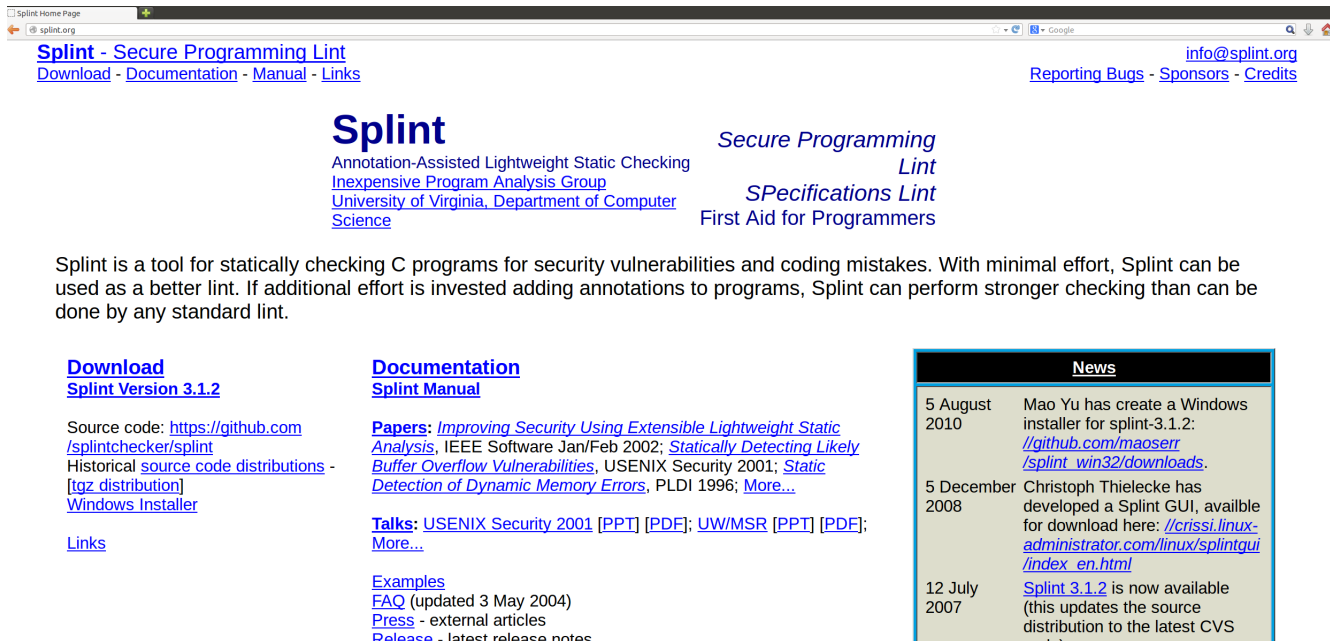
More details you can get from Splint User's Manual [link](#).

With such knowledge, your goal is to achieve the followings:

- Install splint;
- Finish code samples with 2 different kinds of problems which can be detected by Splint. You can choose any 2 of 11 problems as above.
- Use splint to detect the 2 kinds of problems. Describe your observations in your report.

1.2 下载splint

进入官网



Splint
Annotation-Assisted Lightweight Static Checking
[Inexpensive Program Analysis Group](#)
[University of Virginia, Department of Computer Science](#)

Secure Programming Lint
Specifications Lint
First Aid for Programmers

Splint is a tool for statically checking C programs for security vulnerabilities and coding mistakes. With minimal effort, Splint can be used as a better lint. If additional effort is invested adding annotations to programs, Splint can perform stronger checking than can be done by any standard lint.

Download
[Splint Version 3.1.2](#)

Source code: <https://github.com/splintchecker/splint>
Historical [source code distributions](#) - [\[tgz distribution\]](#)
[Windows Installer](#)

Documentation
[Splint Manual](#)

Papers: [Improving Security Using Extensible Lightweight Static Analysis](#), IEEE Software Jan/Feb 2002; [Statically Detecting Likely Buffer Overflow Vulnerabilities](#), USENIX Security 2001; [Static Detection of Dynamic Memory Errors](#), PLDI 1996; [More...](#)

Talks: [USENIX Security 2001](#) [PPT] [PDF]; [UW/MSR](#) [PPT] [PDF]; [More...](#)

Examples
[FAQ](#) (updated 3 May 2004)
[Press](#) - external articles
[Release](#) - latest release notes

News

| | |
|-----------------|--|
| 5 August 2010 | Mao Yu has create a Windows installer for splint-3.1.2: //github.com/maoserr/splint_win32/downloads . |
| 5 December 2008 | Christoph Thielecke has developed a Splint GUI, availble for download here: //crissi.linux-administrator.com/linux/splintgui/index_en.html |
| 12 July 2007 | Splint 3.1.2 is now available (this updates the source distribution to the latest CVS code) |

1.3 提取并设置splint

```
1 tar zxvf splint-3.1.2.src.tgz # 在下载目录进行解压
2 mkdir /usr/local/splint # 创建一个新文件夹
3 cd splint-3.1.2 # 进入解压文件夹
4 ./configure --prefix=/usr/local/splint # 进行配置
5 make
6 sudo make install # 进行make安装程序
```

进行到 `make install` 这一步时出现了问题:

```
1 cscanner.o: In function `input':
2 /home/lhmd/splint/src/cscanner.c:2483: undefined reference to `yywrap'
3 cscanner.o: In function `yylex':
4 /home/lhmd/splint/src/cscanner.c:2133: undefined reference to `yywrap'
5 collect2: ld returned 1 exit status
6 make[2]: *** [splint] Error 1
```

通过查找资料,发现需要修改文件cscanner.c,在其中添加函数实现

```
1 int yywrap()
2 {
3     return 1;
4 }
```

然后重新make即可。

安装完成之后需要配置环境变量,在 `.bashrc` 中添加如下语句:

```
1 export LARCH_PATH=/usr/local/share/splint/lib/
2 export LCLIMPORTDIR=/usr/local/share/splint/imports/
3 export PATH=$PATH:/usr/local/splint/bin
```

```
lhmd@ubuntu:~/Documents$ cat ~/.bashrc
export LARCH_PATH=/usr/local/splint/share/splint/lib
export LCLIMPORTDIR=/usr/splint/share/splint/imports
export PATH=$PATH:/usr/local/splint/bin
```

重新 `source ~/.bashrc`

1.4 构造一个至少有两种不同问题的代码

```
lhmd@ubuntu:~/Documents$ cat loop_hole.c
#include <stdio.h>
int main() {
    int a,s,d,f,g,h,j;
    a = 1;
    var = 234;
    while(a == 1) d++;
    return 0;
}
```

这段代码有两处问题：

- var变量未定义就被使用
- d在初始化之前就被使用了
- while无限循环
- 定义了但是没有使用的变量 `s, f, g, h, j`

使用 `splint loop_hole.c` 得到如下结果：

```

lhmd@ubuntu:~/Documents$ splint loop_hole.c
Splint 3.1.2 --- 16 May 2023

loop_hole.c: (in function main)
loop_hole.c:5:2: Unrecognized identifier: var
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
loop_hole.c:6:16: Variable d used before definition
  An rvalue is used that may not be initialized to a value on some execution
  path. (Use -usedef to inhibit warning)
loop_hole.c:6:8: Suspected infinite loop. No value used in loop test (a) is
  modified by test or loop body.
  This appears to be an infinite loop. Nothing in the body of the loop or the
  loop test modifies the value of the loop test. Perhaps the specification of a
  function called in the loop body is missing a modification. (Use -inflows to
  inhibit warning)
loop_hole.c:3:8: Variable s declared but not used
  A variable is declared but never used. Use /*@unused@*/ in front of
  declaration to suppress message. (Use -varuse to inhibit warning)
loop_hole.c:3:12: Variable f declared but not used
loop_hole.c:3:14: Variable g declared but not used
loop_hole.c:3:16: Variable h declared but not used
loop_hole.c:3:18: Variable j declared but not used

Finished checking --- 8 code warnings

```

2 Lab 3.2 Using eclipse for java static analysis

2.1 Overview

The learning objective of this lab is for students to gain the first-hand experience on using static code analyzers in Eclipse to check Java program for security vulnerabilities and coding mistakes.

In this Lab, your goal is to achieve the followings:

- Install plugins in Java;
- Learn to check Java code by using static code analyzers in Eclipse. Describe your observations in your report.

2.2 Open Source Code Analyzers in Java

Here we introduce 3 kinds of open source code analyzers in Java.

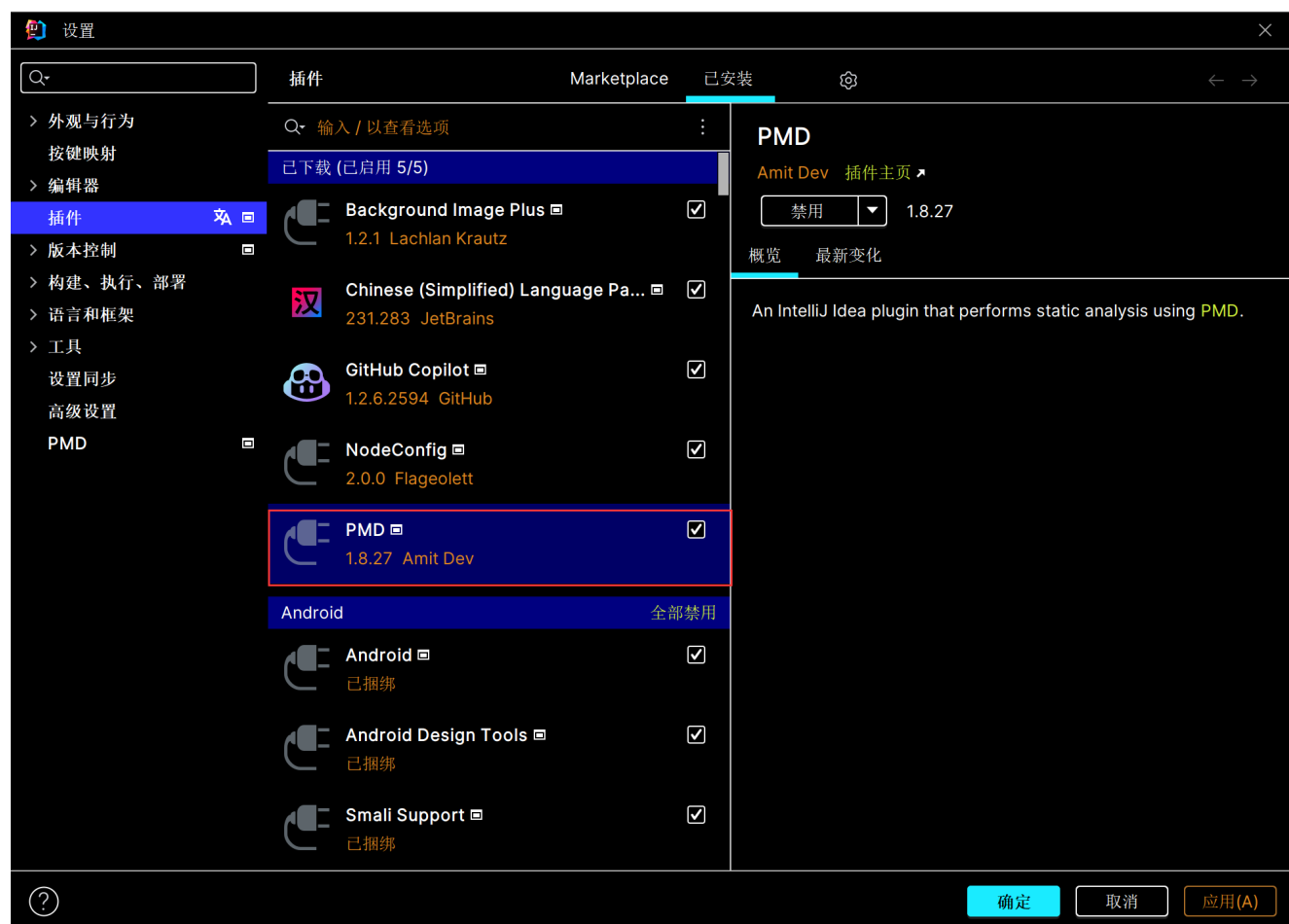
- **FindBugs**
- FindBugs looks for bugs in Java programs. It can detect a variety of common coding mistakes, including thread synchronization problems, misuse of API methods, etc. Go To FindBugs
- **PMD**
- PMD scans Java source code and looks for potential problems like:

- * Unused local variables
 - * Empty catch blocks
 - * Unused parameters
 - * Empty 'if' statements
 - * Duplicate import statements
 - * Unused private methods
 - * Classes which could be Singletons
 - * Short/long variable and method names
- **Checkstyle**
- Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard. Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration file is supplied supporting the Sun Code Conventions. As well, other sample configuration files are supplied for other well known conventions. Can be integrated into CruiseControl and Eclipse. Go To Checkstyle

由于我第一次实验就使用了IDEA，所以本次实验使用IDEA完成

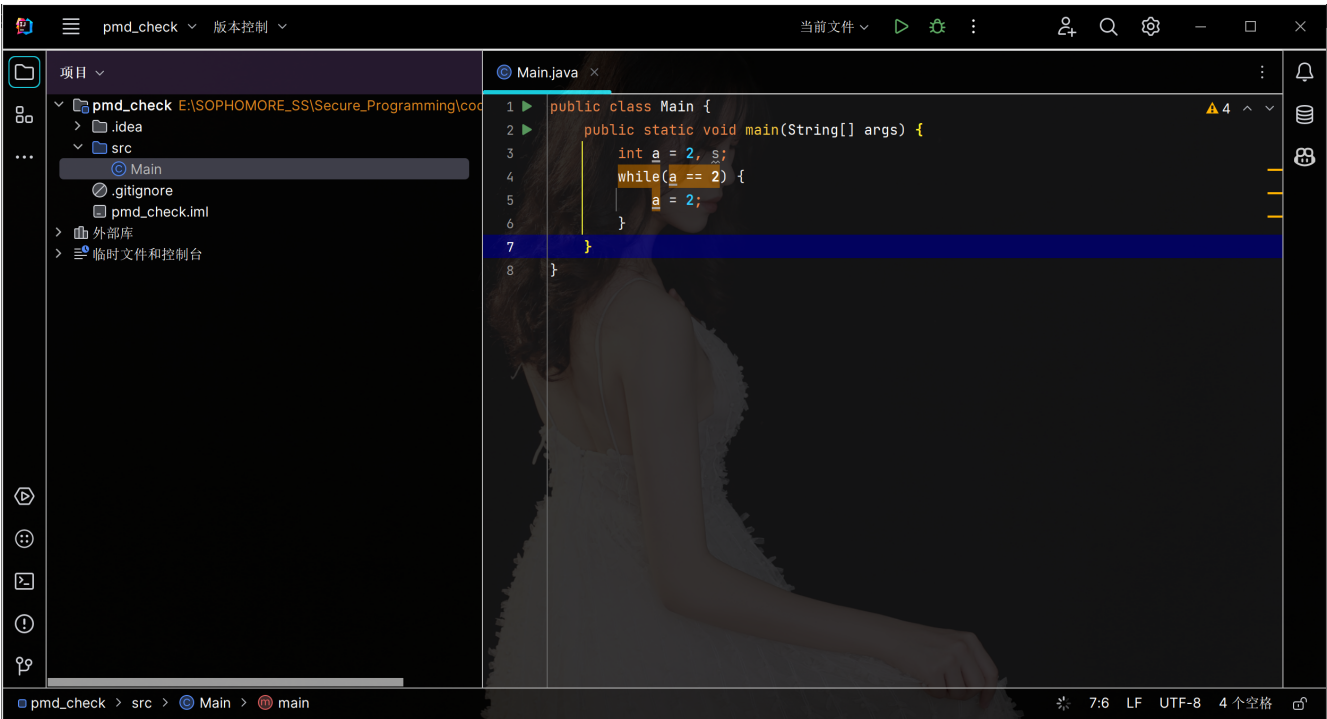
2.3 下载PMD插件

在插件商店下载PMD插件



2.4 新建Java项目，编写代码

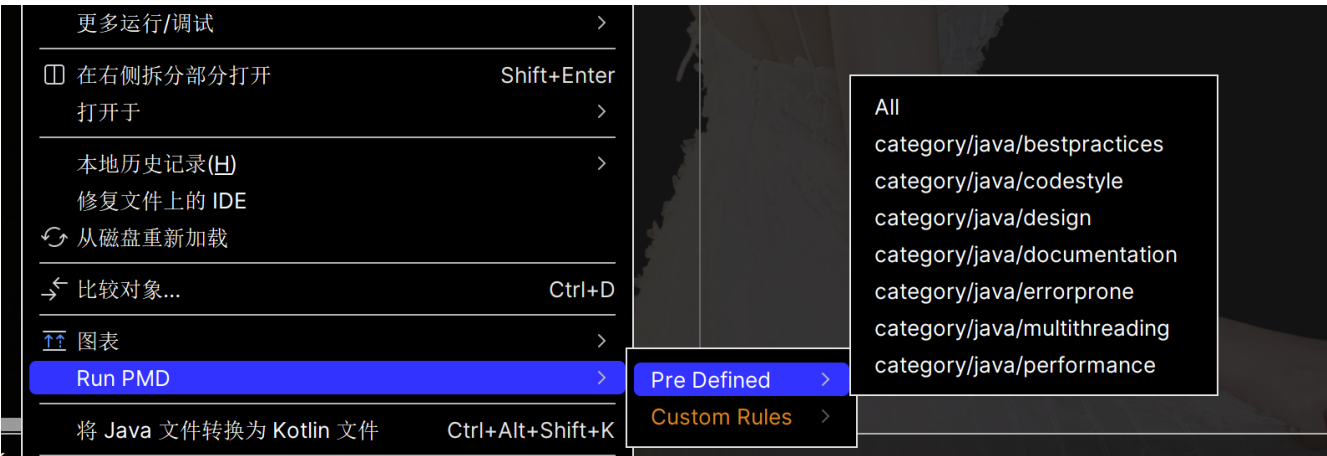
在Main.java中写入以下代码

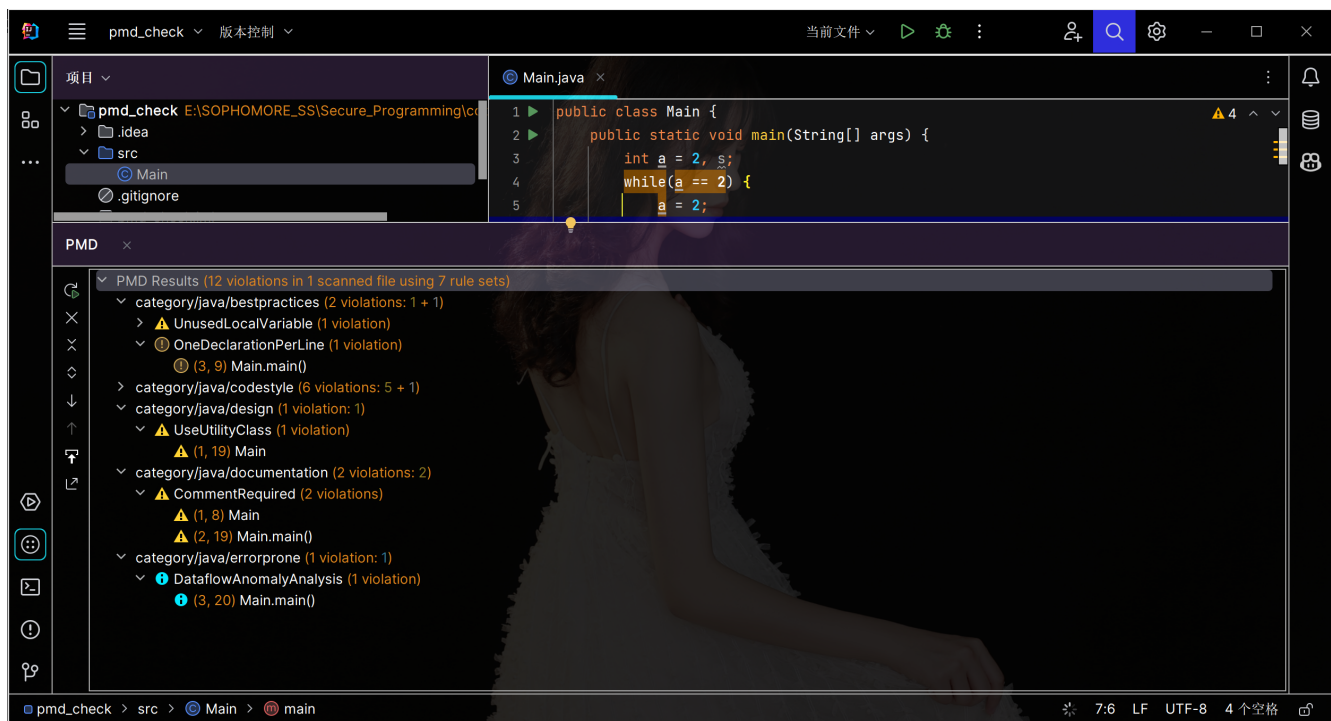


这段代码有两个问题：

- 未使用的变量
- a一直被赋予同一个值，造成while无限循环

2.5 使用PMD插件运行代码





可以发现PMD指出了s变量未被使用，但是并未指出无限循环问题。说明PMD插件还是有一定局限性。

2.6 使用IDEA自带的代码分析



发现我们的问题都被指出来了。