SP-lab3.1&&3.2

3180103772 张溢弛

Lab 3.1 Using splint for C static analysis

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.

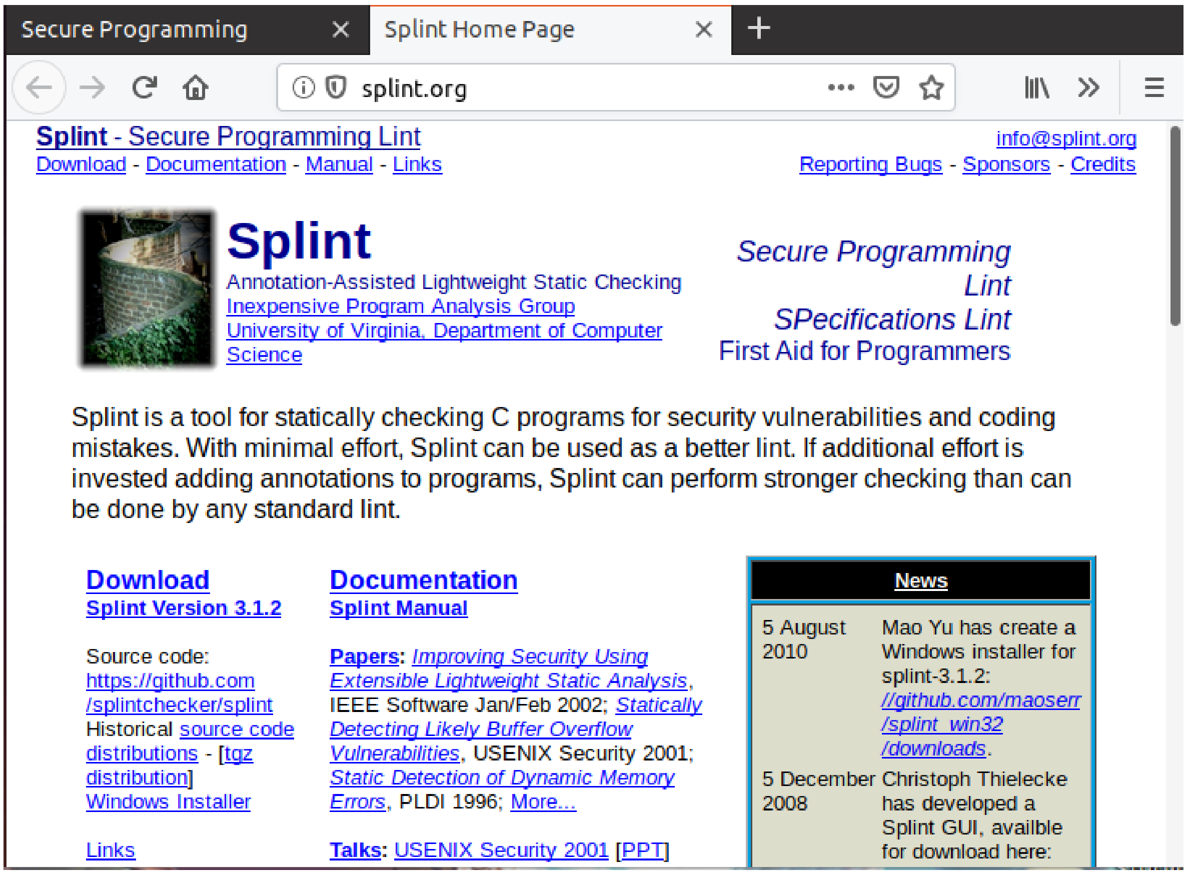
Splint link is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional lint checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no 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.

实验过程

- 先进入splint下载网站下载对应的安装包



Splint - Secure Programming Lint
Download - Documentation - Manual - Links
info@splint.org
Reporting Bugs - Sponsors - Credits

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

Links

Documentation
Splint Manual

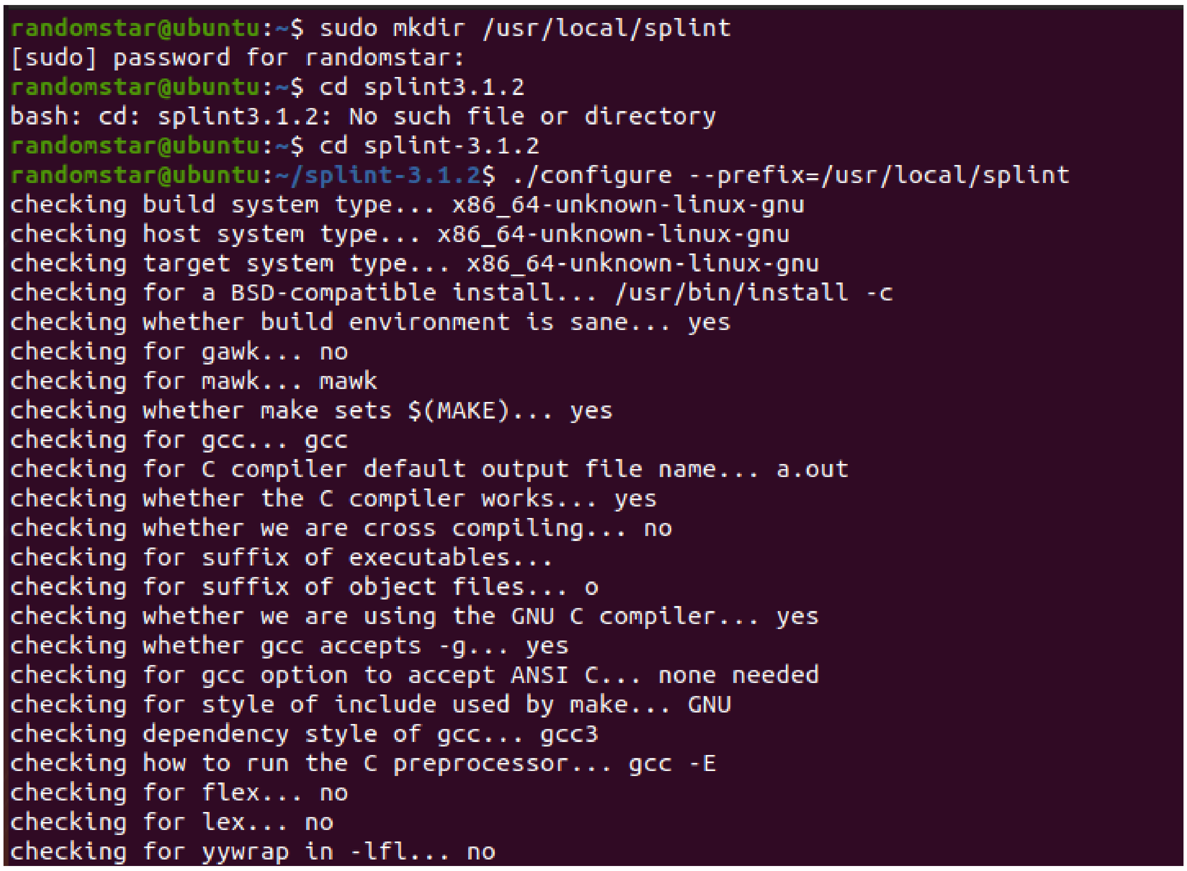
Papers: *Improving Security Using Extensible Lightweight Static Analysis*, IEEE Software Jan/Feb 2002; *Statically Detecting Likely Buffer Overflow Vulnerabilities*, USENIX Security 2001; *Static Detection of Dynamic Memory Errors*, PLDI 1996; More...

Talks: USENIX Security 2001 [PPT]

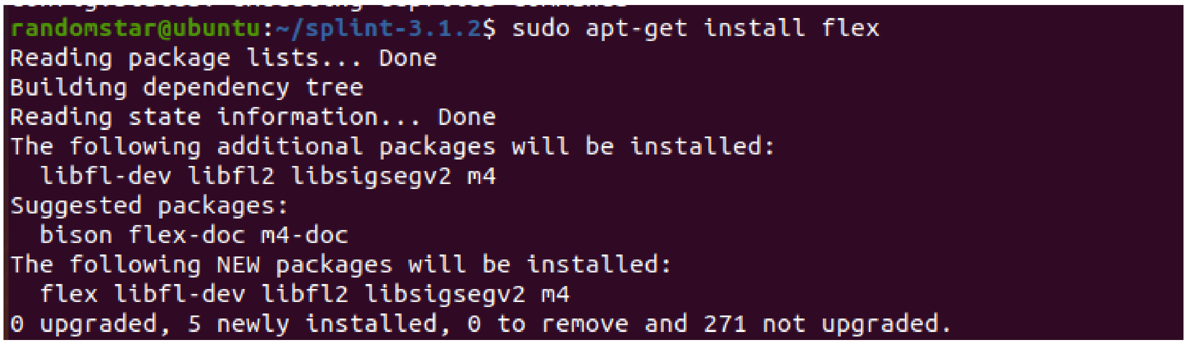
News

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:

- 按照规定的步骤配置

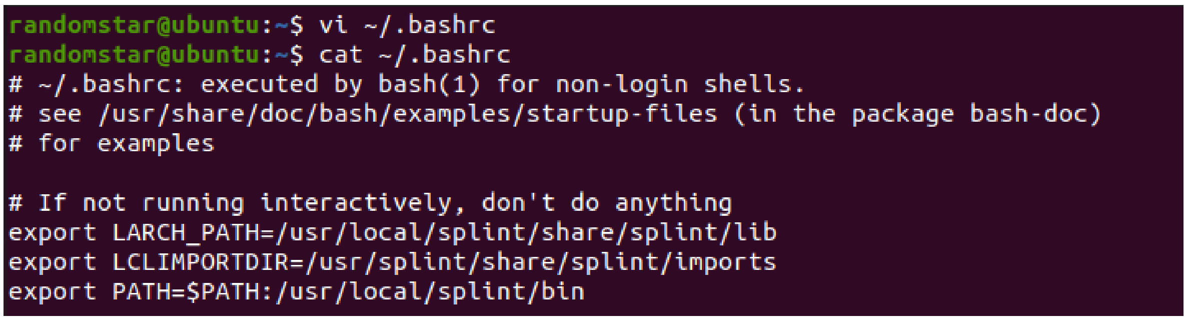


```
randomstar@ubuntu:~$ sudo mkdir /usr/local/splint
[sudo] password for randomstar:
randomstar@ubuntu:~$ cd splint3.1.2
bash: cd: splint3.1.2: No such file or directory
randomstar@ubuntu:~$ cd splint-3.1.2
randomstar@ubuntu:~/splint-3.1.2$ ./configure --prefix=/usr/local/splint
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking how to run the C preprocessor... gcc -E
checking for flex... no
checking for lex... no
checking for yywrap in -lfl... no
```



```
randomstar@ubuntu:~/splint-3.1.2$ sudo apt-get install flex
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libfl-dev libfl2 libsigsegv2 m4
Suggested packages:
  bison flex-doc m4-doc
The following NEW packages will be installed:
  flex libfl-dev libfl2 libsigsegv2 m4
0 upgraded, 5 newly installed, 0 to remove and 271 not upgraded.
```

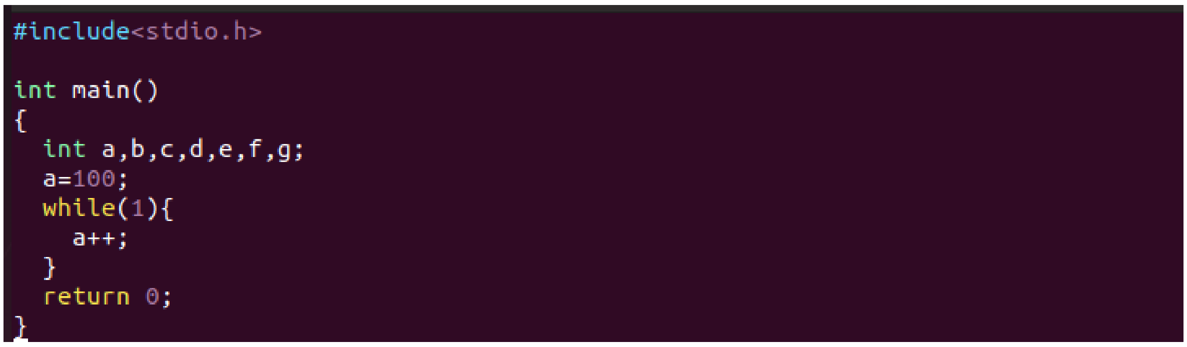
- `make`和`sudo make install`两个命令运行的时候产生了一大堆东西导致截图效果很难看，这里就省略截图了
- 配置环境变量，先修改bashrc文件中的环境变量配置



```
randomstar@ubuntu:~$ vi ~/.bashrc
randomstar@ubuntu:~$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
export LARCH_PATH=/usr/local/splint/share/splint/lib
export LCLIMPORTDIR=/usr/splint/share/splint/imports
export PATH=$PATH:/usr/local/splint/bin
```

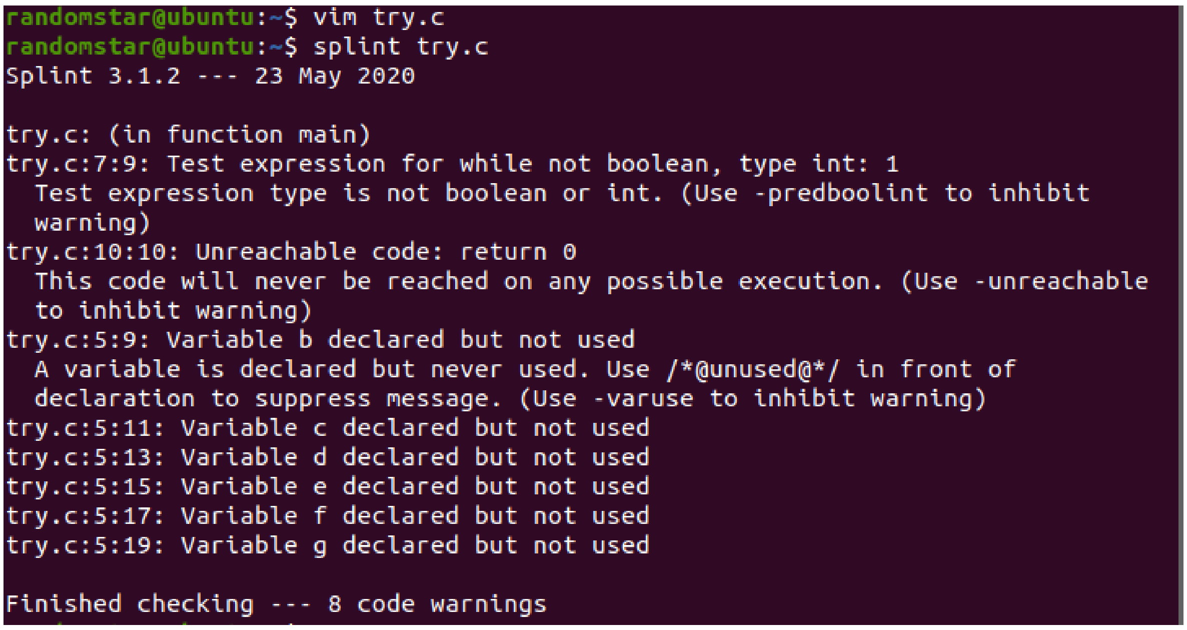
- 然后执行命令`source ~/.bashrc`
- 编写一段存在至少两种问题的C语言代码，如下图所示，存在的问题是变量未使用和死循环



```
#include<stdio.h>

int main()
{
  int a,b,c,d,e,f,g;
  a=100;
  while(1){
    a++;
  }
  return 0;
}
```

- 使用splint进行静态分析，结果如下所示



```
randomstar@ubuntu:~$ vim try.c
randomstar@ubuntu:~$ splint try.c
Splint 3.1.2 --- 23 May 2020

try.c: (in function main)
try.c:7:9: Test expression for while not boolean, type int: 1
  Test expression type is not boolean or int. (Use -predboolint to inhibit
  warning)
try.c:10:10: Unreachable code: return 0
  This code will never be reached on any possible execution. (Use -unreachable
  to inhibit warning)
try.c:5:9: Variable b 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)
try.c:5:11: Variable c declared but not used
try.c:5:13: Variable d declared but not used
try.c:5:15: Variable e declared but not used
try.c:5:17: Variable f declared but not used
try.c:5:19: Variable g declared but not used

Finished checking --- 8 code warnings
```

- splint提示到while中的表达式不是布尔表达式，并且存在死循环和变量未使用的情况，原本的代码中存在的问题都被发现了，因此本实验成功！

Lab 3.2 Using eclipse for java static analysis

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. Descibe your observations in your report.

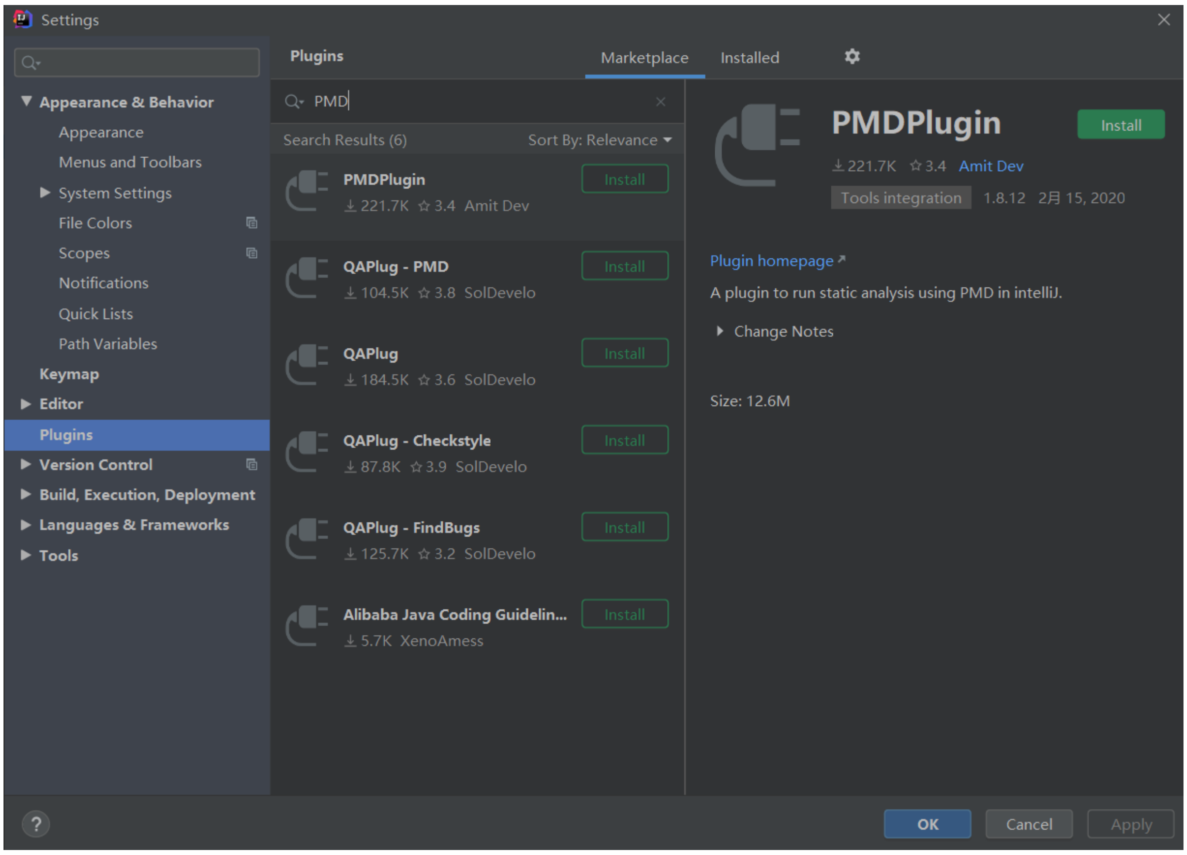
Open Source Code Analyzers in Java

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

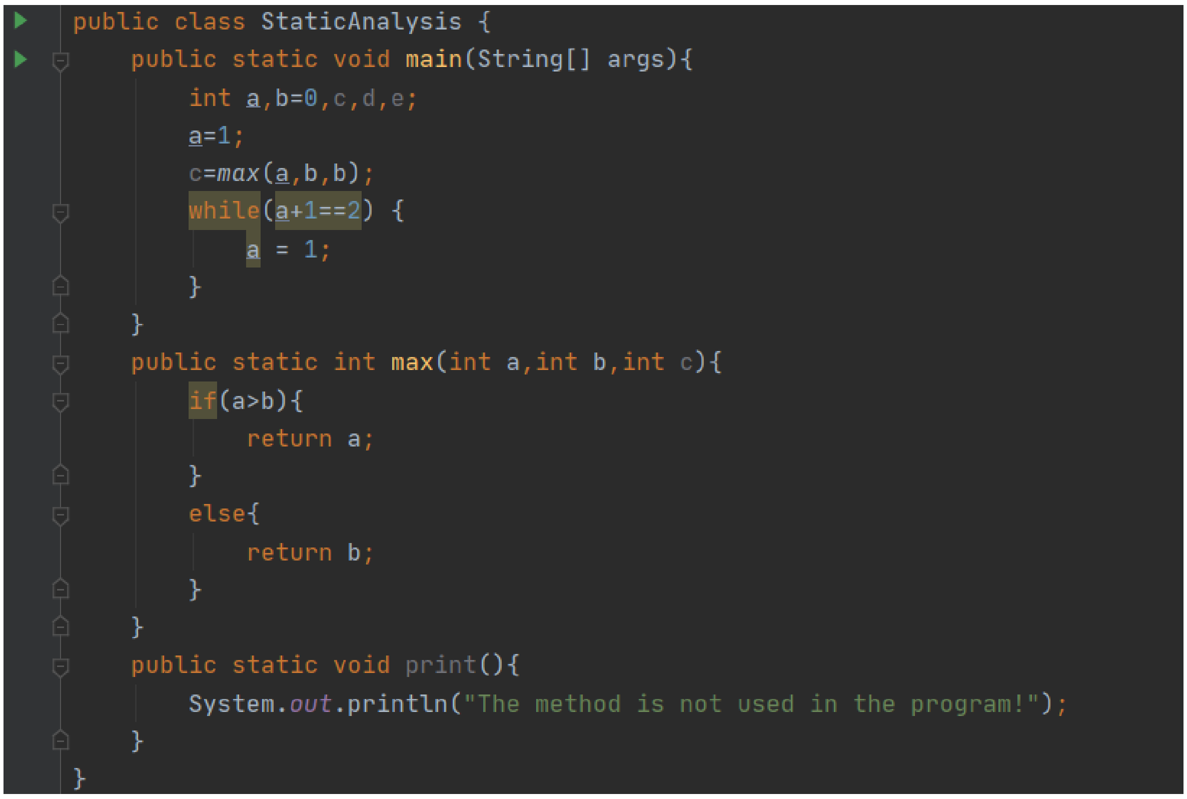
- **FindBugs**
- FindBugs looks for bugs in Java programs. It can detect a variety of common coding mistakes, including thread synchronization problems, misuse of API methods, etc. Go To FindBugs
- **PMD**
- PMD scans Java source code and looks for potential problems like:
- * Unused local variables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

实验过程

- 实验1.1中已经安装了~~宇宙第一的JavaIDE也就是~~IDEA，这次试验我们采用IDEA而非Eclipse来进行Java的静态分析
- 首先来安装相关的插件，打开File中的settings，进入插件下载的界面，下载插件PMD

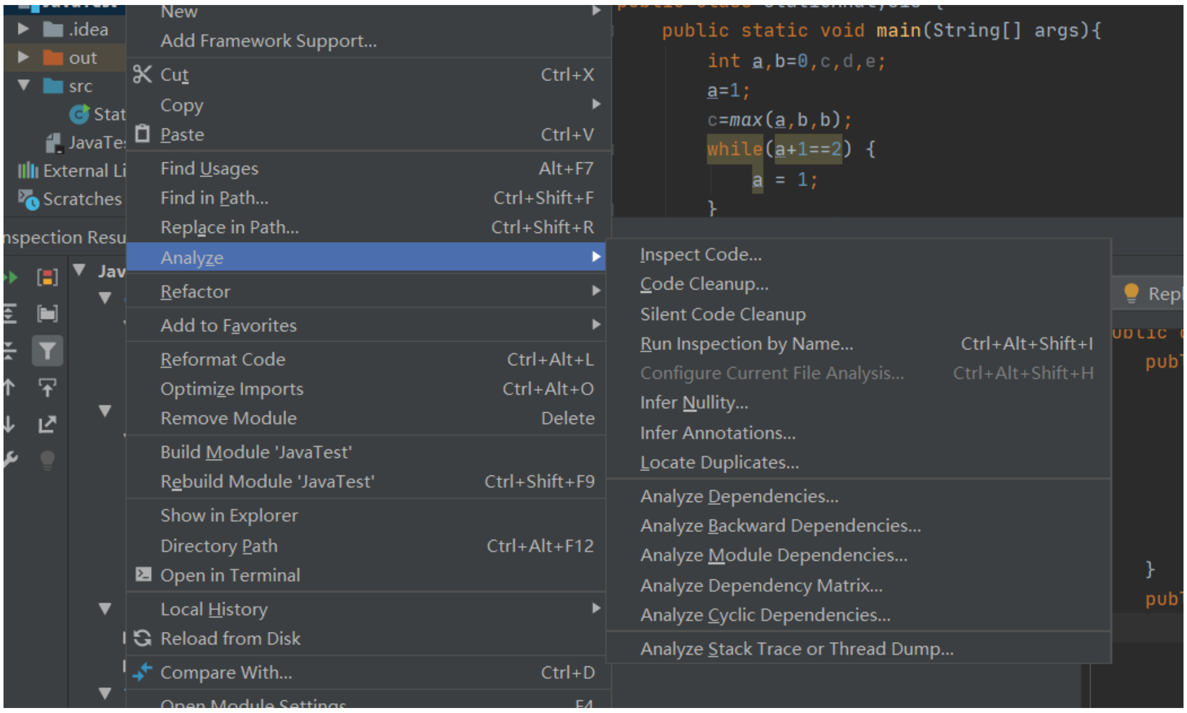

Settings
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
Installed
PMD
Search Results (6)
Sort By: Relevance
PMDPlugin 221.7K 3.4 Amit Dev Install
QAPlug - PMD 104.5K 3.8 SolDevelo Install
QAPlug 184.5K 3.6 SolDevelo Install
QAPlug - Checkstyle 87.8K 3.9 SolDevelo Install
QAPlug - FindBugs 125.7K 3.2 SolDevelo Install
Alibaba Java Coding Guidelin... 5.7K XenoAmess Install
PMDPlugin
Install
221.7K 3.4 Amit Dev
Tools integration 1.8.12 2月 15, 2020
Plugin homepage
A plugin to run static analysis using PMD in intelliJ.
Change Notes
Size: 12.6M
OK
Cancel
Apply

- 先编写一段存在多种问题的Java代码(包含变量未使用，方法未调用，死循环等多种问题)

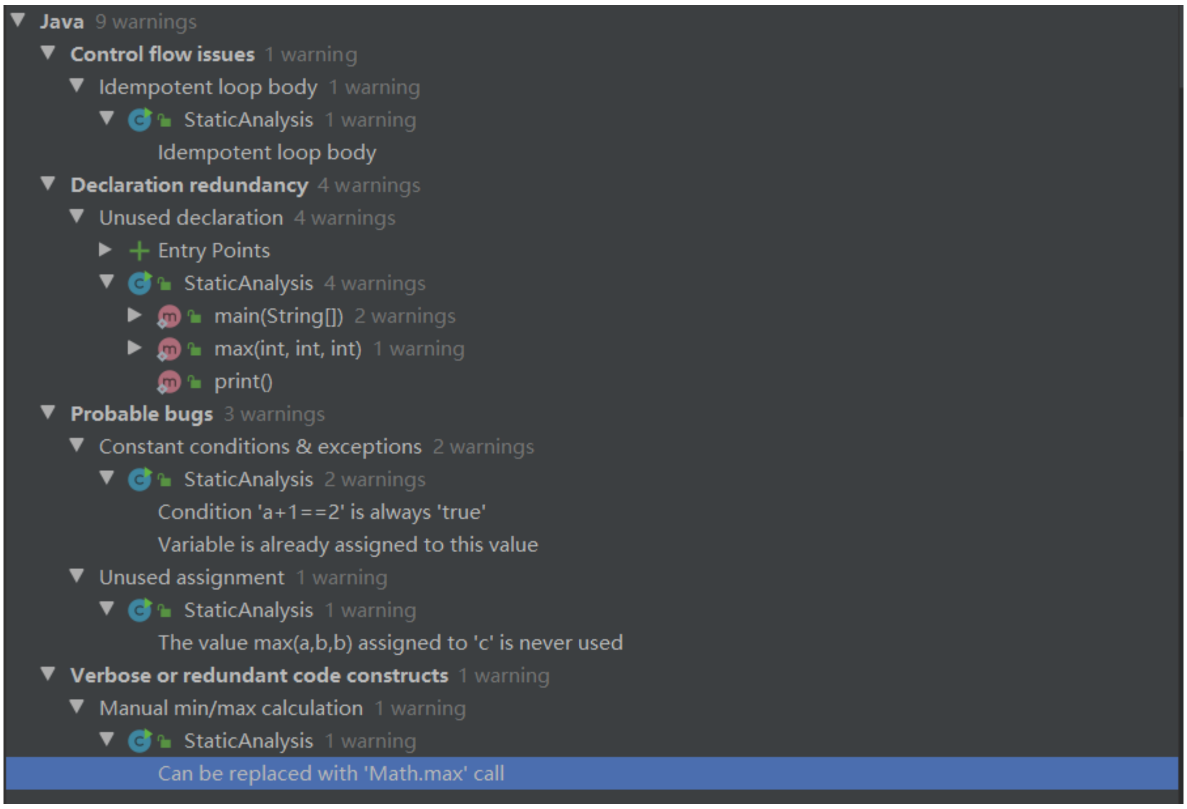


```java
public class StaticAnalysis {
    public static void main(String[] args){
        int a,b=0,c,d,e;
        a=1;
        c=max(a,b,b);
        while(a+1==2) {
            a = 1;
        }
    }
    public static int max(int a,int b,int c){
        if(a>b){
            return a;
        }
        else{
            return b;
        }
    }
    public static void print(){
        System.out.println("The method is not used in the program!");
    }
}
```

- 进行Java代码的静态分析，选择Inspect code，结果如下(这一部分是IDEA自带的检查功能)
    - 可以看到自带的检查功能也发现了所有可能存在的问题，包括变量和方法未使用，无限循环和方法参数未使用等问题

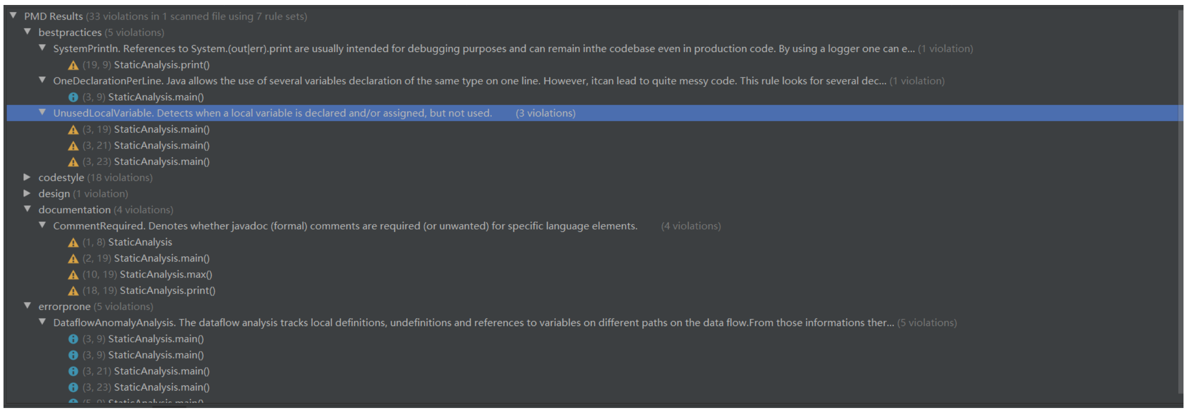


```
    public static void main(String[] args){
        int a,b=0,c,d,e;
        a=1;
        c=max(a,b,b);
        while(a+1==2) {
            a = 1;
        }
```



```
▼ Java 9 warnings
  ▼ Control flow issues 1 warning
    ▼ Idempotent loop body 1 warning
      ▼ StaticAnalysis 1 warning
          Idempotent loop body
  ▼ Declaration redundancy 4 warnings
    ▼ Unused declaration 4 warnings
      ▶ Entry Points
      ▼ StaticAnalysis 4 warnings
        ▶ main(String[]) 2 warnings
        ▶ max(int, int, int) 1 warning
          print()
  ▼ Probable bugs 3 warnings
    ▼ Constant conditions & exceptions 2 warnings
      ▼ StaticAnalysis 2 warnings
          Condition 'a+1==2' is always 'true'
          Variable is already assigned to this value
    ▼ Unused assignment 1 warning
      ▼ StaticAnalysis 1 warning
          The value max(a,b,b) assigned to 'c' is never used
  ▼ Verbose or redundant code constructs 1 warning
    ▼ Manual min/max calculation 1 warning
      ▼ StaticAnalysis 1 warning
          Can be replaced with 'Math.max' call
```

- 也可以进行`run PMD`的操作，会发现这段代码的如下问题



```
▼ PMD Results (33 violations in 1 scanned file using 7 rule sets)
  ▼ bestpractices (5 violations)
    ▼ SystemPrintln. References to System.(out|err).print are usually intended for debugging purposes and can remain inthe codebase even in production code. By using a logger one can e... (1 violation)
        (19, 9) StaticAnalysis.print()
    ▼ OneDeclarationPerLine. Java allows the use of several variables declaration of the same type on one line. However, itcan lead to quite messy code. This rule looks for several dec... (1 violation)
        (3, 9) StaticAnalysis.main()
    ▼ UnusedLocalVariable. Detects when a local variable is declared and/or assigned, but not used.      (3 violations)
        (3, 19) StaticAnalysis.main()
        (3, 21) StaticAnalysis.main()
        (3, 23) StaticAnalysis.main()
  ▶ codestyle (18 violations)
  ▶ design (1 violation)
  ▼ documentation (4 violations)
    ▼ CommentRequired. Denotes whether javadoc (formal) comments are required (or unwanted) for specific language elements.      (4 violations)
        (1, 8) StaticAnalysis
        (2, 19) StaticAnalysis.main()
        (10, 19) StaticAnalysis.max()
        (18, 19) StaticAnalysis.print()
  ▼ errorprone (5 violations)
    ▼ DataflowAnomalyAnalysis. The dataflow analysis tracks local definitions, undefinitions and references to variables on different paths on the data flow.From those informations ther... (5 violations)
        (3, 9) StaticAnalysis.main()
        (3, 9) StaticAnalysis.main()
        (3, 21) StaticAnalysis.main()
        (3, 23) StaticAnalysis.main()
```

- PMD插件也帮助我们发现了上述所有的问题，在IDEA中点击对应的问题就可以访问对应的代码片段，可以说这个功能做的非常厉害
- 至此实验3.2成功完成！