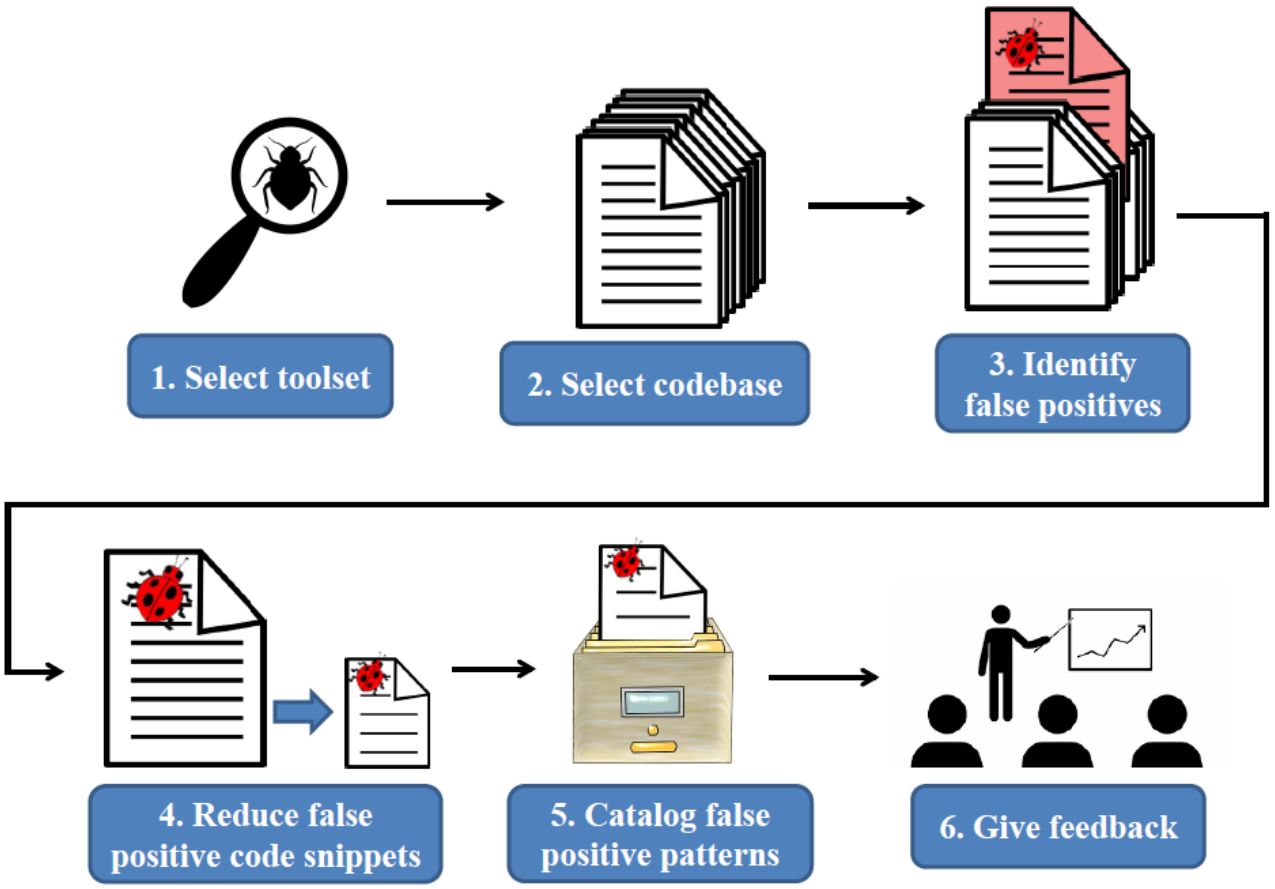
相关研究

Beller et al. [25]

- 内容：设计通用缺陷分类方案（GDC），将来自不同静态代码分析工具的警告映射到相同的分类结构
- 优点：可以区分资源警告和逻辑警告
- 不足：分类类别太广，无法根据结构对误报进行分类

Approach



1. 选择使用的静态分析工具

Static code analysis tool	Tool type
Tool A	Open source
Tool B	Open source
Tool C	Commercial

2. 选择待测代码数据集

- C/C++ Juliet test suite version 1.2 [18]：61,387测试用例，包含118个CWE漏洞；同时还按照flow variants组织
  优点：
  （1）每个test case包含：flaw（缺陷位置），potential flaw（source和sink），fix（缺陷被修复的版本），incidental（不可避免的缺陷，不同于target weakness），可用来识别true positive，false positive和false negative
  （2）是漏洞的简化版本，方便阅读和理解

3. 定义误报

- 对于误报，fgets没有越界，需要指出来
- 对于真实warning，尽管造成warning的输入产生困难，仍然定义为warning
- 因为context insensitivity导致的误报，代码在当前上下文没有问题，但是在别的上下文困难导致漏洞
- Juliet 可以用来识别误报
- 静态代码分析工具的评估框架：Static Code Analysis Tool Evaluator (SCATE) [19]：工具检测、人工验证、动态分析（Valgrind [30]）

4. 提取产生误报的代码（reducing the source code）

- 缩减的时候会产生几种可能：
  （1）还能扫描到相同的漏洞，漏洞还是误报；
  （2）漏洞变成正报
  （2）无法扫描到相同的漏洞
- 缩减步骤：

- 去除不必要的代码
- 函数重命名为有意义的名字
- 用库函数替代用户定义函数，去除外部依赖（头文件）

5. 对误报进行分类识别误报模式

- 描述误报分类的属性：
  - Name：误报名称
  - False positive warning：不同工具描述的融合
  - Description：对导致误报的代码结构的一种非正式的解释
  - Tools：发现这个误报模式的工具列表
  - Frequency：误报在codebase出现的频率
  - Code snippet：缩减后的误报代码片段
  - Change：使工具不产生告警的小代码变化

6. 向静态分析工具的作者提供反馈，验证我们的工作

Result and discussion

- 误报层次结构

- 静态分析工具开发者反馈
  开发者承认了3个误报模式，一个被认为是正报，一个模式无法复现（使用平台不同）
- False positive frequencies

False Positive Pattern	Tool A	Tool B	Tool C
1. Alloc using external size	0	0	240
2. Array input conditional	0	0	19
3. Array resize	0	0	12
4. Buffer store	0	0	428
5. Buffer underflow usage	3,685	0	0
6. Clear list	0	0	61
7. File close virtual method	0	0	2
8. List overrun	0	0	63
9. Null array access	0	55	0
10. Operation through alias	68	1,029	0
11. Predictable conditional	3,322	384	0
12. Return local param	1,456	0	0
13. Sscanf uninit var	0	0	2
14. Union simultaneous access	0	250	0

计算方法：

（1）对于每个误报模式，找出静态代码分析工具在Juliet上标记的所有警告，这些警告语模式指示的警告类型相同

（2）对上一步的结果进行随机采样并检查warning，确定误报的比例

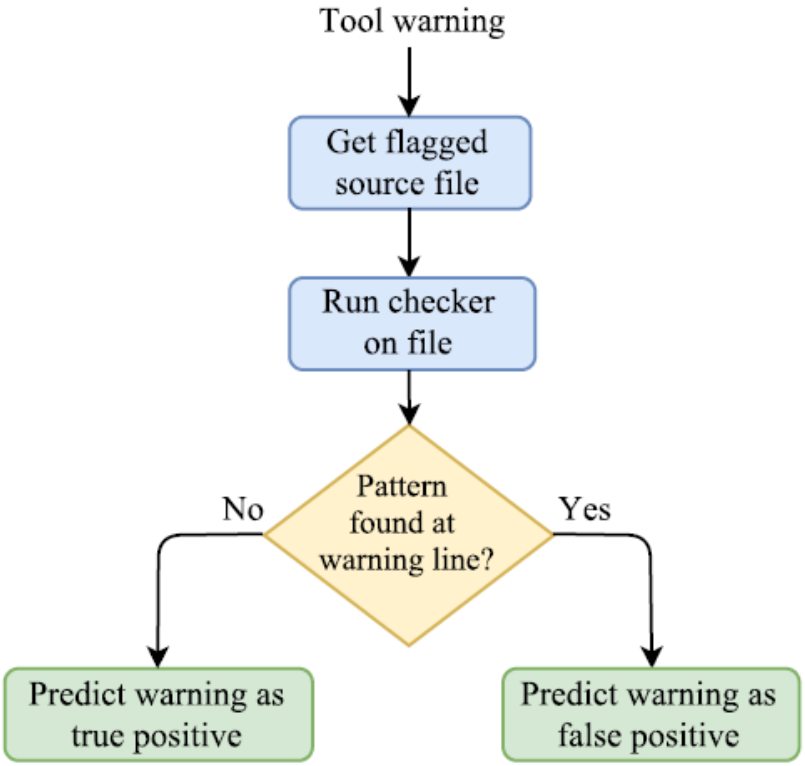
（3）使用采样结果来计算置信区间为95%的从集合中随机选择一个为该误报模式实例的概率

（4）对剩余的未采样警告应用置信区间的下界来近似误报模式的实例总数

误报过滤器

Filter Approach

- 工具：Clang’s LibASTMatchers framework [37]
- 提供了类似于C/C++的丰富接口 [38]
- 对于每一种漏洞模式，设计了一种检测器



- 步骤：

  （1）识别产生警告的文件

  （2）在源码文件上运行checker，若匹配到该误报模式，且位置与产生警告位置相应则为误报，若不相同则不是误报

  （3）应该使用多个checker进行检测，任何一个checker匹配到，即判定其为误报

过滤误报的前提是能检测出尽全部（可能多的）漏洞