# Learning a Classifier for False Positive Error Reports Emitted by Static Code Analysis Tools


---

## 术语

- taint analysis：检查从不可信来源到安全关键汇聚点的数据流

---

## Abstract

- 当代软件系统的大规模和高复杂度→准确的static code analysis（SCA）不可行
- SCA工具的over-approximate和assumption→误报率高
- 解决部分问题的方法：探索导致误报 `程序结构` ，并根据此程序结构进行预测

1. 预处理代码：将与缺陷报告相关的代码独立出来
   a. 简单的方法：将包含warning的方法body部分取出来
   b. 计算warning的backward slice
2. 使用一种简单的机器学习（Navie Bayes）方法探索与缺陷报告相关的 `程序结构`
3. 学习分类器，使用更高级的机器学习技术（long short-term memories，LSTM）来过滤误报

- 个案研究：一个广泛使用的Java SCA工具（FindSecBugs），训练导致错误报告的程序（Java字节码表示）
- 对SCA工具产生误报的原因做了一些解释
- 人工标注的有监督学习

---

## Approach

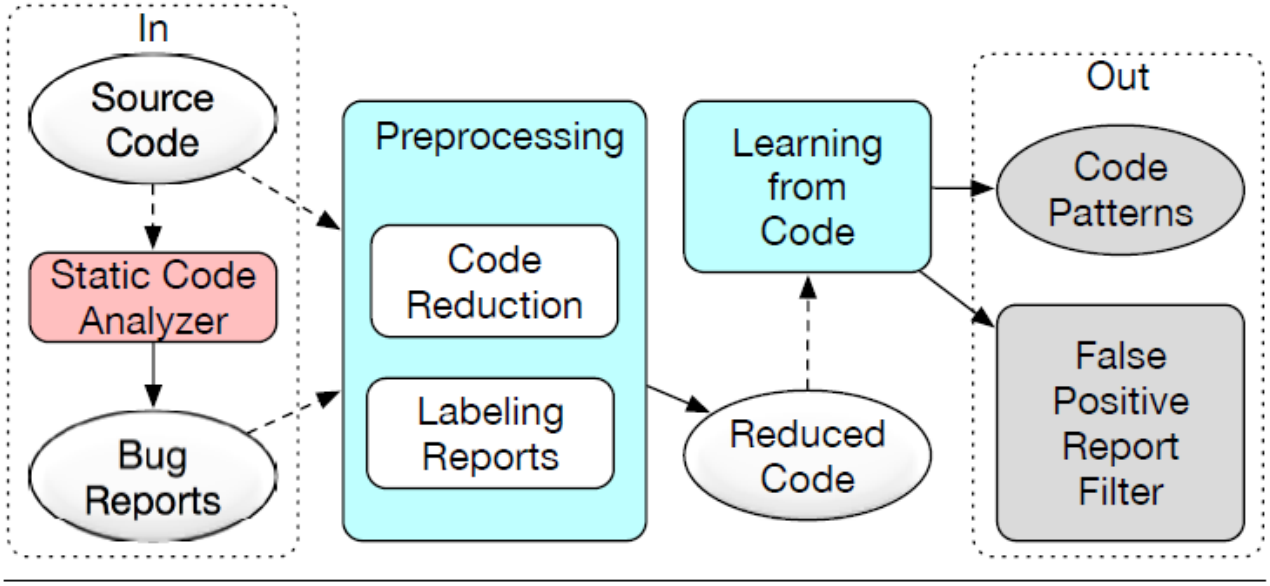


**Figure 1.** Learning approach overview.

## Code Preprocessing

- Method body
- Program slicing
  计算从warning line到程序入口的程序slice可以得到一个backward slice，覆盖所有与错误报告相关的代码位置
  WALA，计算backward slice的工业级规模框架

## Learning

- Goals
  a. discovering code pieces correlated with these classes: Navie Bayes
  b. learning a classifier: LSTM

1. Navie Bayes Inference

- P(e = 0)，误报
- P(e = 1)，正确的缺陷报告
- P(Code)，从所有代码的未知分布中获得特定代码的概率

$$P(e=0|code) = \frac{P(code|e=0)P(e=0)}{P(code)} = \frac{P(code|e=0)P(e=0)}{P(code|e=0)P(e=0) + P(code|e=1)P(e=1)}$$

- code= <I1, I2,...,In>，指令序列（字节码）

$$
\begin{aligned}
P(code|e=0) &= P(I_1, I_2, ..., I_n|e=0) \\
&= P(I_1|e=0)P(I_2, ..., I_n|I_1, e=0) \\
&= P(I_1|e=0)P(I_2|I_1, e=0)P(I_3, ...I_n|I_1, I_2, e=0) \\
&... \\
&= P(I_1|e=0)P(I_2|I_1, e=0)...P(I_n|I_1, I_2, ..., e=0)
\end{aligned}
$$

- Markvo property

$$
P(code|e=0) = P(I_1|e=0)P(I_2|e=0)...P(I_n|e=0)
$$

- 计算P(Ii | e=0)

**Algorithm 1** Computing Probabilities

```
for each code C in Dataset do
    for each instruction I in C do
        count[C.isTruePositive][I]++
        total[C.isTruePositive]++
    end for
end for
for each instruction I do
    P(I|e=1) ← count[True][I]/total[True]
    P(I|e=0) ← count[False][I]/total[False]
end for
```

1. Long Short Term Memory（LSTM）

- 使用LSTM的原因：数据是序列化的；long-term dependency与缺陷报告的是否为误报有关
- 可能在代码中形成Long-term dependency的示例：变量定义-使用对、带参数的方法调用、访问类字段
- 为了可视化模型的内部工作，倾向于有更少的cell（4个），每个cell都是LSTM
- LSTM随时间推移展开的结构

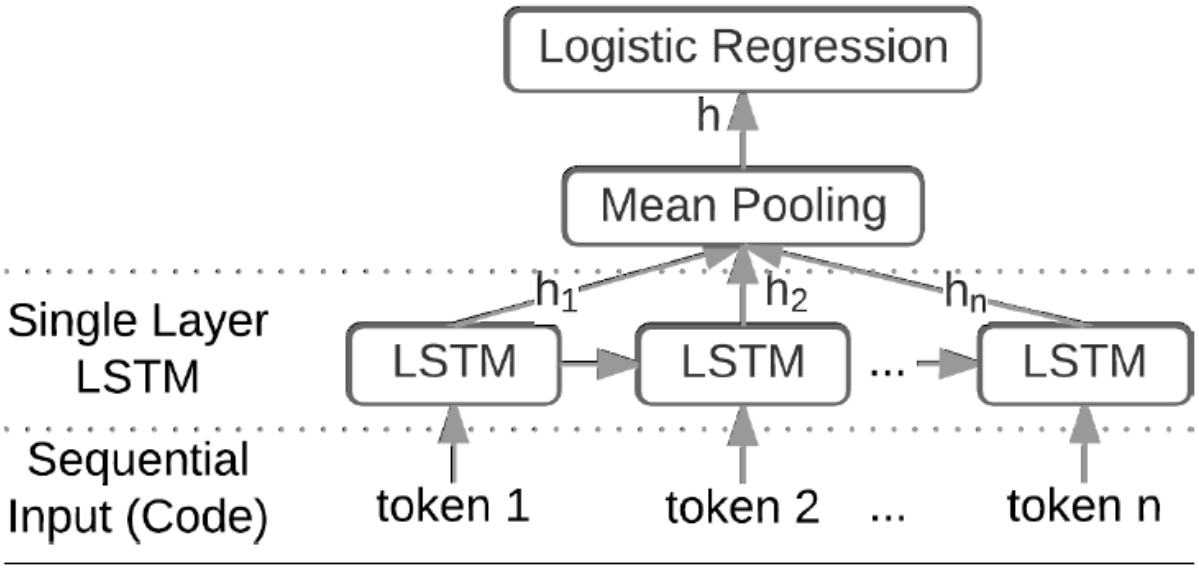


**Figure 2.** The LSTM model unrolled over time.

# Case Study

## 实验目标工具和warning类型

- SQL注入缺陷类型
- FindSecBugs，FindBugs的安全检测插件
  a. taint analysis
  b. SQL注入的安全关键汇聚点-Connection.execute(String)
  c. 传递给该方法的string如果来自一个不可信的源（HTTP Cookie；用户输入），则认为是受污染的
- 目前模型是为某SCA工具扫描出来的特定缺陷训练的，不能同时支持多种缺陷类型

## Data：没有标记好的数据，使用评价SCA 工具性能的数据集-OWASP benchmark

- 误报示例

```java
public class BenchmarkTest16536 extends HttpServlet {
  public void doPost(HttpServletRequest r){
    String param = "";
    Enumeration<String> headers =
         r.getHeaders("foo");
    if (headers.hasMoreElements()) {
      // just grab first element
      param = headers.nextElement();
    }
    String bar = doSomething(param);
    String sql = "{call verifyUserPassword('foo','"
                + bar + "')}";
    Connection con = DatabaseHelper.getConnection();
    CallableStatement stmt = con.prepareCall(sql);
    stmt.execute();
  } // end doPost

  private static String doSomething(String param){
    String bar = "safe!";
    HashMap<String,Object> map = new HashMap();
    map.put("keyA", "a_Value");
    map.put("keyB", param.toString());
    map.put("keyC", "another_Value");
    bar = (String) map.get("keyB");
    bar = (String) map.get("keyA");
    return bar;
  }
}
```

**Figure 3.** An example Owasp test-case that FindSecBugs generates a false positive error report for (simplified for presentation).

## 预处理

- 更关注数据的字节码：程序特定的token和syntactic component更少；更适合机器学习算法（simplified，itemized）
- method body和backward slice（WALA）
- backward slice，理论上来说，backward slice应该包括覆盖与错误报告相关的所有代码的位置，但是太耗时了

  a. 忽略exception objects, base pointers, and heap components

  b. 将entry point设置的尽可能靠近warning method（warning line是起点，entry point是终点）

  c. 去除与本case study无关的utilities classes，Java classes，第三方库

- WALA的输出会附加如"new", "branch", "return"，表示purpose
- 移除program-specific tokens和literal expressions→是分类器具有更好的共享性

---

# Restful and Analysis

1. Navie Bayes

- False Positive Dependency

$$\left[\frac{P(I|e=0)}{P(I|e=0)+P(I|e=1)}-0.5\right]$$

a. 接近1，误报

b. 接近-1，正确的缺陷

c. 接近0，P(I | e=0) 约等于P(I | e=1)，与SQL注入缺陷无关

- 实验结果

| classifier | dataset | training time (m) | recall | precision % | accuracy |
|---|---|---|---|---|---|
| Naive Bayes | method body | 0.02 | 60 | 64 | 63 |
| | backward slice | 0.03 | 66 | 75 | 72 |
| LSTM | method body | 17 | 81.3 | 97.3 | 89.6 |
| | backward slice | 18 | 97 | 78.2 | 85 |

**Table 1.** Results.

- 可能造成误报，或错过正确缺陷的指令

| instructions | False Positive Dependence | |
|---|---|---|
| | method body | backward slice |
| invoke esapi.Encoder | −0.09 | −0.36 |
| invoke java.util.ArrayList | 0.04 | 0.18 |
| invoke java.util.HashMap | 0.18 | 0.25 |

**Table 2.** Important instructions

- 缺陷：多条指令说明是正确的缺陷，但是一条或几条指令说明其是安全的（误报），Navie Bayes会识别错误