ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2000/00(00)-0000-00
doi: 10.3778/j.issn.1673-9418.

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

基于污点分析的 Android 应用 SQL 注入漏洞静态检测技术*

潘秋红[1], 崔展齐[2,4], 王林章[1,2,3]

1. 南京大学 计算机科学与技术系, 南京 210023
2. 计算机软件新技术国家重点实验室（南京大学） 南京 210023
3. 江苏省软件新技术与产业化协同创新中心 南京 210023
4. 北京信息科技大学 计算机学院 北京 100101

Static Detection Approach of SQL Injection Vulnerabilities in Android Applications based on Taint Analysis *

PAN Qiuhong[1], CUI Zhanqi[1,4], WANG Linzhang[1,2,3]

1. Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
2. State Key Laboratory of Novel Computer Software Technology, Nanjing University, Nanjing 210023, China
3. Jiangsu Novel Software Technology and Industrialization, Nanjing 210023, China
4. Computer School, Beijing Information Science and Technology University, Beijing 100101, China
+ Corresponding author: E-mail: lzwang@nju.edu.cn

PAN Qiuhong, CUI Zhanqi, WANG Linzhang. Static detection approach of SQL injection vulnerabilities in Android applications based on taint analysis. Journal of Frontiers of Computer Science and Technology, 2017, 0(0): 1-000.

Abstract: The number of Android terminals and applications have been increasing in recent years with the rapid development of mobile Internet, which greatly changes people's life. However, mobile applications are complicated to interact, difficult to debug, and their versions update frequently. Many applications have been published without adequate testing, which makes failures caused by various vulnerabilities in Android applications occur frequently. SQL injection is a kind of common security vulnerability, which can cause user information leakage and database to be tampered maliciously. However, general static analysis tools cannot detect SQL injection vulnerabilities in An-

*The National Natural Science Foundation of China under Grant No.61472179, No.61572249, No.61632015, No.61561146394 (国家自然科学基金); the National Key Research and Development Plan No.2016YFB1000802 (国家重点研发计划); Open Program of State Key Laboratory for Novel Software Technology No.KFKT2016B12 (计算机软件新技术国家重点实验室开放课题).
Received 2017-00, Accepted 2000-00.

droid applications effectively. Aiming at this problem, this paper analyzes the code and data characteristics of SQL injection vulnerabilities, and puts forward a static detection approach based on taint analysis. It extends the open source tools FindBugs, and implements the prototype tool SQLInj. The result of experiments indicate that this approach can detect the SQL injection vulnerabilities in Android applications effectively.

Key words: SQL injection; static detection; taint analysis; legitimate check

摘 要: 随着移动互联网的迅猛发展，基于 Android 平台的移动终端以及移动应用数量逐年攀升，极大地改变了人们的生活方式。然而，移动应用具有交互复杂、难于调试、版本更新迭代频繁等特点，很多应用没有经过充分检测就投入了使用，致使 Android 应用中各种漏洞导致的故障频发。其中，SQL 注入漏洞是一类常见安全漏洞，会引发用户信息泄露、恶意篡改数据库等严重后果。但现有的通用静态分析工具大多无法有效检测 Android 应用中的 SQL 注入漏洞。针对这一问题，分析了 SQL 注入漏洞的代码特征和数据特征，提出了一种基于污点分析的静态检测方法，并在开源工具 FindBugs 的基础上，实现了原型工具 SQLInj。实验结果表明，该方法能有效检测出 Android 应用中存在的 SQL 注入漏洞。

关键词: SQL 注入; 静态检测; 污点分析; 合法性检查

文献标志码: A **中图分类号:** TP301

1 引言

SQL 注入是攻击者由外部输入向程序中原有的数据库执行语句中插入恶意 SQL 语句片段，篡改 SQL 执行命令来操作数据库的安全漏洞。无论是对于 Web 应用还是移动应用，SQL 注入都是一个严重的安全问题。攻击者可以利用 SQL 注入漏洞窃取用户敏感信息、恶意篡改数据库中的内容、提升权限等，引发严重后果。根据 OWASP（Open Web Application Security Project）发布的 Web 应用十大安全威胁中，从 2010 年至 2017 年 4 月，注入类漏洞都高居榜首[1]。

SQL 注入漏洞对软件安全产生了巨大威胁，对 SQL 注入的检测和防范是开发者在开发软件时必须考虑的一个重要因素。已经提出了很多方法用于检测 Web 应用中的 SQL 注入漏洞，主要可分为静态分析[2]、动态测试[3]两类。静态分析方法如符号执行技术，其将符号作为值赋给变量，对程序的路径进行模拟执行，精确分析程序的代码属性[2]。动态测试则要通过运行软件来得到程序的动态行为信息，包括分析软件的覆盖率、监控内存的状态、分析执行轨迹、提取程序不变式等[3]。同时，针对 Web 应用中的 SQL 注入漏洞已有很多工具得到了广泛应用，如 HP Fortify SCA[4]、Coverity[5]等。

近年来，随着无线通信技术和移动互联网迅猛发展，移动终端普及率快速提高。据 CNNIC 第 39 次《中国互联网络发展状况统计报告》[6]，截至 2016 年 12 月，我国手机网民规模已达到 6.95 亿，台式电脑、笔记本的上网比例则持续下降，网民上网设备进一步向移动端集中。其中，据 CNNIC《2015 年中国手机网民网络安全状况报告》[7]，使用 Android 操作系统的手机占 67.4%。根据 Veracode 2016 年发布的软件安全状态报告，在 Android 应用存在的安全漏洞中，SQL 注入漏洞位列第八[8]。然而，Coverity、FindBugs[9]等面向 Web 应用的通用软件质量及安全漏洞检测工具未关注移动应用的 SQL 注入特性，导致无法有效识别 Android 应用中的 SQL 注入漏洞。此外，用于检测 SQL 注入漏洞的方法未使用污点分析等技术，针对 Android 应用进行静态污点分析的工具，如 FlowDroid[10]等，不支持直接检测 SQL 注入漏洞，即使通过人工修改配置文件增加对 SQL 注入漏洞的描述，也只是分析 SQL

方法中的 SQL 参数是否被污染，没有考虑应用中是否对外部输入进行过合法性检查，导致误报率较高。

针对上述问题，本文提出了一种基于污点分析的 Android 应用 SQL 注入漏洞静态检测方法。首先，根据 Android 应用的字节码文件进行程序静态分析，并在其上定位 SQL 注入漏洞的 SQL 方法和 SQL 参数；然后，通过静态污点分析方法，检测 SQL 参数是否来自外部输入；最后，基于 SQL 注入的输入验证机制，通过识别应用中是否存在对污染的 SQL 参数进行过合法性检查，来判断是否存在 SQL 注入漏洞。

本文的贡献包括以下两点：

1. 针对 Android 应用，提出一种基于污点分析的 SQL 注入漏洞静态检测方法；

2. 基于所提出的方法实现了原型工具 SQLInj，并设计了一组实验来验证所提出方法的有效性。

本文的组织结构如下：第 2 节介绍 SQL 注入漏洞模型；第 3 节详细介绍基于静态分析的 Android 应用 SQL 方法及参数定位；第 4 节介绍方法的原型工具的实现和实验评估；第 5 节介绍与本文相关的研究工作；第 6 节总结了本文的工作并对下一步的研究计划进行展望。

2 SQL 注入漏洞模型

SQL 注入漏洞是发生在使用数据库对数据进行管理的应用程序中的一种安全漏洞，其本质是攻击者通过表单等方式进行外部输入，在应用程序中预先定义好的数据库查询语句中插入恶意的 SQL 语句，篡改其含义来欺骗数据库服务器执行非授权的查询[11]，通过这些操作获得数据库信息，非法读取、修改、添加、删除数据，私自添加账号，注入木马等其它病毒。

攻击者一般通过构造有特殊含义的 SQL 语句进行 SQL 注入攻击，攻击方式大致可以分为以下几类[12][13]：重言式与注释符攻击、非法/逻辑错误查询攻击、联合查询攻击、推断攻击、基于存储过程或函数攻击、复合查询攻击、编码替换攻击等。

对于每种 SQL 注入攻击方式，其注入的字符串中通常会包含不同的敏感字符，这些字符是 SQL 语言中定义的拥有特殊含义的字符，攻击者就是通过这些字符篡改原 SQL 指令。我们将各类 SQL 注入攻击方式可能采用的敏感字符进行了整理，如表 1 所示。

Table 1 SQL injection sensitive characters
表 1 SQL 注入敏感字符

攻击方式	敏感字符
重言式	Or, =
注释符	--, #
联合查询	union
推断攻击	and, wait
基于存储过程或函数攻击	xp_cmdshell
替换编码攻击	exec, char
非法/逻辑错误查询攻击	select, from
复合查询攻击	select, drop, like, insert, delete, update
敏感标点	“*”, “‘”, “;”, “or”, “-”, “--”, “+”, “//”, “/”, “%”, “#”, “(”, “) "

为了能够检测出 Android 应用中的 SQL 注入漏洞，首先需要对 SQL 注入漏洞进行建模和分析，然后在此基础上对程序进行静态分析，并在代码中定位 SQL 注入攻击的 SQL 方法和 SQL 参数，然后通过污点分析及判断合法性检查判定程序中是否存在 SQL 注入漏洞。

假设将由若干语句（Statement）组成的 Android 应用程序（Program）表示为 $P=\{s_1, s_2, \cdots, s_n\}$，则可将 SQL 注入漏洞模型定义如下：

- SQL 方法为语句集合 $S_{SQL} \subseteq P$ 中的元素，其中，对任意的语句 $s_i \in S_{SQL}$，s_i 为 SQL 操作语句，SQL 操作语句包括 rawQuery、execSQL、query 等与数据库访问相关的 API 调用；
- 对于 SQL 方法 s_i，变量集合 $Par_i=\{par_{i,1}, par_{i,2}, \ldots, par_{i,m}\}$ 为 s_i 所使用的参数，其中，$par_{i,j} \in Par_i$ 即为 s_i 的 SQL 参数；
- 对于任意的 SQL 参数 $par_{i,j} \in Par_i$，若其与 P 的外部输入相关，且在使用前未进行过合法性检

查，则认为 SQL 方法 s_i 存在 SQL 注入漏洞。

以图 1 所示的代码片段为例，其功能为当 USER 表中存在用户输入的用户名且其密码也正确时允许其登录。首先，分析程序发现第 5 行为 SQL 方法 android.database.sqlite.SQLiteDatabase.rawQuery；然后，分析该 SQL 方法 rawQuery 中的 SQL 参数为 *sql* 和 **null**；最后，对 SQL 参数进行分析，其中，参数 *sql* 在第 3 行通过字符串 *username* 和 *password* 拼接而来，而 *username* 和 *password* 是应用登录界面用户输入的用户名和密码，因此 SQl 参数 *sql* 与外部输入有关，且在 SQL 方法 rawQuery 执行前，程序并没有对 SQL 参数 *sql* 进行合法性检查，因此，我们可以判断 SQL 方法 rawQuery 处存在 SQL 注入漏洞。若攻击者在用户名和密码处均输入“ 1’ **OR** ‘1’ = ‘1 ”，则会通过 SQL 方法 rawQuery 中的 SQL 参数 *sql*, 将外部输入传递到后台数据库中。此时执行的 SQL 查询语句的形式为：

SELECT * from USER **WHERE** USERNAME = ‘1’ **OR** ‘1’=‘1’ **AND** PASSWORD = ‘1’ **OR** ‘1’=‘1’;

在这种情况下，该 SQL 查询语句的结果恒为真，将会成功利用 SQL 注入漏洞进行攻击，攻击者无需正确的用户名和密码即可直接登录。

```
1.  String username = UserView.getText().toString();
2.  String password = PassView.getText().toString();
3.  String sql = “ select * from USER where USERNAME=
    ‘ “ +username+“ ‘and PASSWORD=‘ “+password+“ ‘ “;
4.  SQLiteDatabase db = m_dbhelper.getWritableDatabase();
5.  Cuesor m_cursor = db.rawQuery(sql, null);
6.  String result = m_cursor.getString(0);
7.  ShowView.setText(result);
```

Fig. 1 An example of SQL injection
图 1 SQL 注入示例

3 基于污点分析的 Android 应用 SQL 注入漏洞静态检测技术

针对缺少专用的 Android 应用 SQL 注入漏洞检测工具，而现有的通用检测工具用于检测 Android 应用 SQL 注入漏洞时精确度较低的问题，本文提出了一种静态检测 Android 应用中潜在 SQL 注入漏洞的方法。如图 2 所示，首先，通过静态分析处理程序的结构，获得程序的控制流图和数据流图，并在程序中定位 SQL 方法和 SQL 参数；然后，对程序进行静态污点分析，判断 SQL 方法中的参数是否来自外部输入；最后，对于使用了污点数据的 SQL 方法，要判断应用中是否对其中的 SQL 参数进行了合法性检查，若不存在合法性检查，则认为这一条 SQL 方法存在 SQL 注入漏洞。

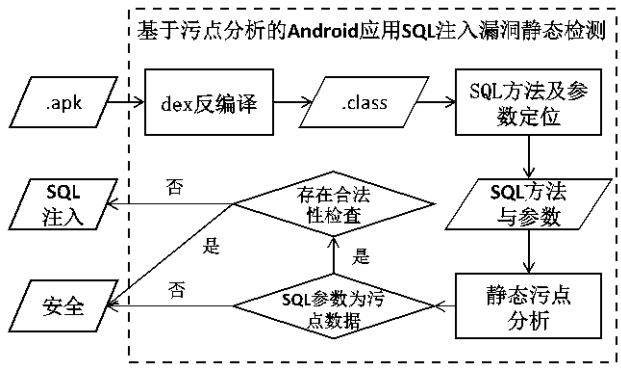

Fig. 2 Flowchart of static detection technology of SQL injection vulnerability in Android applications based on taint analysis
图 2 基于污点分析的 Android 应用 SQL 注入漏洞静态检测技术流程图

3.1 基于静态分析的 SQL 方法及参数定位

为了能够对代码进行静态检测，需要对程序的结构进行静态分析，使用合适的数据结构来描述代码。为扩大方法的适用范围，静态分析的对象采用了 Android 应用的字节码文件而非源代码。首先对程序进行静态分析，通过控制流分析、数据流分析等技术剖析程序的结构，然后在程序中定位 SQL 注入漏洞模型中的 SQL 方法及相应 SQL 参数。图 3 为通过静态分析定位 Android 应用 SQL 注入相关 SQL 方法及 SQL 参数的流程图。对于应用的 class 文件以及依赖的 jar 文件，首先需要对字节码进行控制流分析。控制流表示程序的单个语句、指令或者函数调用的顺序，通过有向图表示控制流，形成控制流图。控制流图表示了程序执行过程中所有可能遍历的路径，它的每个结点是一个基本块，有向边指明了基本块间的执行关系。控制流只能从基本

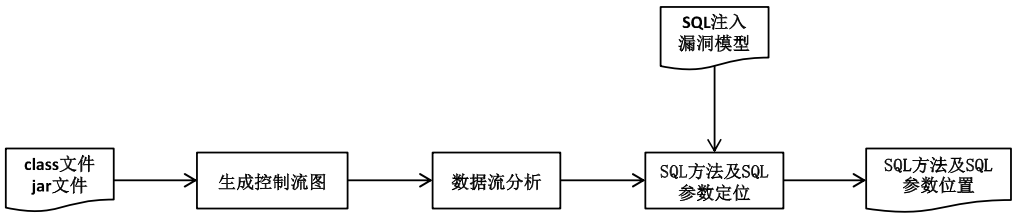

Fig. 3 Process of locating SQL method and parameter based on static analysis
图 3 基于静态分析的 SQL 方法及参数定位流程

块中的第一个指令进入该块，除了基本块的最后一个指令，控制流在离开基本块之前不会停机或者跳转[14]。确定了基本块之后，就可以根据条件或无条件转移指令来识别基本块间的前后流通关系，以此建立字节码文件的控制流图。

有了控制流图之后，可以对程序进行进一步的数据流分析。数据流分析是一种用于获取程序中数据是如何沿着程序执行路径进行流动的信息的技术。在数据流分析中，通过对一组约束求解，就可以得到每个点上的数据流值，可分为基于语句语义约束和基于控制流约束。本文采用的是基于控制流约束方法，即在基本块之内时，每条语句输出的数据流就是下一条语句输入的数据流。在基本块之间时，每条控制流边都对应着新的约束，通过反复计算使系统到达稳定。

根据第 2 节中定义的 SQL 注入漏洞模型，遍历所有方法的控制流图，关注其中用于方法调用的指令，定位使用 SQL 方法的位置信息。然后，对 SQL 方法的参数进行数据流分析，定位每个 SQL 参数的位置、参数标识等信息。最后，我们可以定位程序中调用的所有 SQL 方法，以及每个 SQL 方法对应的 SQL 参数信息。定位 SQL 方法和 SQL 参数后，调用静态污点分析功能，分析对应的 SQL 参数是否来自外部输入。对于使用了外部输入作为参数的 SQL 方法，需要进一步验证程序中是否针对其 SQL 参数进行过合法性检查。

3.2 面向 SQL 注入漏洞的污点分析

本文提出的静态检测 SQL 注入漏洞方法的核心就是对程序进行静态污点分析。本文使用的程序静态分析部分是基于 FindBugs 使用的 BCEL 框架，可对应用的字节码进行处理。静态污点分析通常可分为过程内污点传播和过程间污点传播两类。考虑到部分应用会使用特定的数据结构来存储数据，如数组、Map 等，我们增加了特殊数据结构污点传播。下面将分别对 3 种静态污点传播规则进行介绍：

3.2.1 特殊数据结构污点传播规则

对于外部输入的数据，很多程序都是选择直接通过字符串进行存储和修改。但是，除此之外，有时会使用特定的数据结构来存储数据，如数组、Map 等。下面分别为数组结构、实现 Collection 接口和 Map 接口的类中的污点传播规则。

1. 数组结构

对于数组结构，可以将其视为一个整体进行处理。如果将一个污点数据传递给数组的一个元素，那么就将整个数组都标记为污染，而不是只将该索引位置标记。然后，对污染数组的所有读写操作都会将污染传播。

```
1. String[] spreadArray = new String[2];
2. spreadArray[0] = “constant”;
3. spreadArray[1] = taintStr;
4. String str = spreadArray[0];
```

Fig. 4 Array taint propagation
图 4 数组污点传播

如图 4 中所示的代码片段，字符串 *taintStr* 为污染数据，在第 3 行将其赋值给数组 *spreadArray* 的第 2 个元素时，将整个 *spreadArray* 标记为污点数据。然后在第 4 行将 *spreadArray* 的第一个元素赋值给字符串 str 时，虽然之前并没有将污点数据赋值给 *spreadArray*[0]，但是整个 *spreadArray* 数组空间都被标记为污染了，所以 *str* 也会被标记为污点数据。

2. Collection 接口和 Map 接口

Java 语言中定义了 Collection 和 Map 作为所有集合和 Map 类的接口，开发者可以通过实现这些接

口的子类来更加方便的组织复杂的数据，如 List、HashSet、TreeMap 等。因此，本方法也要对这些结构体进行污点传播处理。这些结构体的污点传播规则与数组类型相似，也是将结构体当作一个整体进行污点标记,即只要将一个污染数据传递给结构体，就将整个结构体视为污染。然后对于点运算符“.”，取最左边对象的污染信息,如 *a.b.c* 的污染信息即为 *a* 的污染信息。

3.2.2 过程内污点传播规则

过程内污点传播是以方法为单位进行的。图 5 为过程内污点分析的流程，对于每个方法，需要一个污点数据集实时记录方法中的污点信息。我们先初始化污点数据集，并将参数污点信息和与外部输入有关的方法、变量加入污点数据集。然后遍历程序，根据指令类型进行相应的操作：对于变量读取指令，记录变量的污点信息；对于普通赋值指令，若之前读取了污点数据，则该指令把污点数据赋值给当前变量，并将该变量其加入污点数据集；对于特殊结构体读写指令,进行特殊数据结构污点传播；对于方法调用指令，若该指令调用子方法，则进行过程间污点传播，若其对结构体进行操作，则进行特殊数据结构污点传播，若其是预先定义好的配置文件中的方法，表示其会把污点标记从参数传播到方法返回值，则将返回值加入污点数据集；对于方法返回指令，其是方法中最后一条指令，我们在记录方法的参数和返回值的污点信息后，输出整个污点数据集。其算法描述如算法 1。

算法 1 过程内污点传播算法

输入： CFG *cfg*， File *config* *//污染源配置文件*
JavaClass *javaclass* *//字节码文件*
输出： HashSet *TaintSet* *//方法污点数据集*
过程：

```
1)   init TaintSet; //初始化 TaintSet
2)   if(Pam is tainted) //判断参数的污点信息
3)     TaintSet.add(Pam);
4)   TaintConfig(TaintSet, config); //配置 TaintSet
5)   for(每条指令 ins){
6)     if(ins 是对变量的读取)
7)       记录读取的变量是否为污点数据;
8)     else if(ins 是对普通的赋值语句)
9)       if(之前读取了污点数据集中的数据)
10)        TaintSet.add(当前变量 data);
11)    else if(ins 是特殊结构体的读写)
12)      doSpeTaint(ins); //特殊结构污点传播
13)    else if(ins 是方法调用指令){
15)      if(ins 调用子方法)
16)        执行过程间污点传播;
17)      else if(ins 是结构体操作方法)
18)        doSpeTaint(ins);
19)      else if(ins in config)
20)        TaintSet.add(ins 返回值);
21)    }else if(ins 是方法返回指令){
24)      记录参数及返回值的污点信息;
25)      return TaintSet;
26)    }
27)  }
```

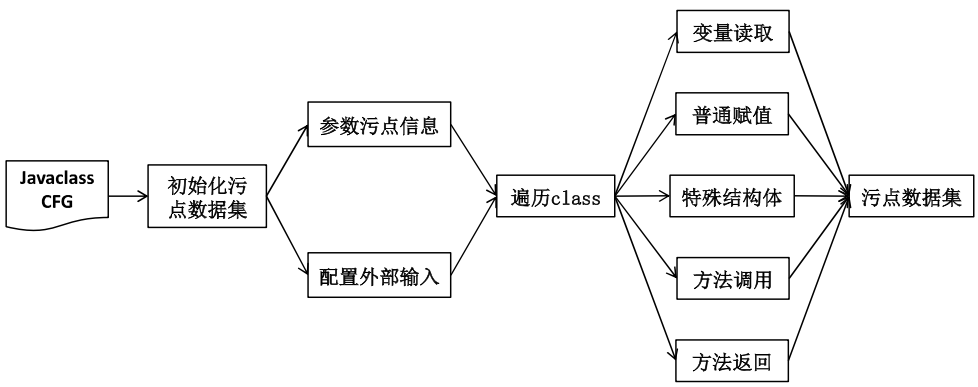

Fig. 5 In-process taint propagation
图 5 过程内污点传播

3.2.3 过程间污点传播规则

本文采用的过程间污点传播规则为：对于进行了过程内污点传播的方法，传播结束时会记录此方法的参数和返回值污点信息。因此，我们首先分析被调用方法参数，若是之前扫描过该方法，且参数污点信息相同，则直接获取扫描时返回值的污点信息；否则，将参数的污点信息传递给被调用方法，对其进行过程内污点传播，然后将其返回值的污点信息返回给调用该方法的位置。其算法描述如算法 2 所示。

算法 2 过程间污点传播算法

输入： Method *call //源方法*
Method *called //子方法*
HashMap *AnalysisedMap*<String *pamInfo*, String *resultInfo*> *//记录进行过过程内污点分析的方法的参数和返回值污点信息，pamInfo 为方法调用时参数污点信息，resultInfo 为方法返回值污点信息*

输出： String *resultinfo //子方法返回值污点信息*

过程：

```
1)  获取子方法 called 参数污点信息 paminfo
2)  if(AnalysisedMap.containsKey(paminfo)) //之
        前调用过该方法且参数污点信息一致
3)    resultinfo = AnalysisedMap.get(paminfo)
4)  else{
5)    对 called 做过程内污点传播
6)    resultinfo = AnalysisedMap.get(paminfo);
7)  }
8)  return resultinfo;
```

3.3 合法性检查

对于使用了来自外部输入参数的数据库执行方法，还无法确定其中确实存在 SQL 注入漏洞。若应用中对数据库查询语句的参数进行过合法性检查，就可以避免出现 SQL 注入的问题。因此，在进行过污点分析以后，还要判断应用中是否对使用了污点数据的 SQL 方法中的 SQL 参数进行过合法性检查。常见的合法性检查的方法有以下几种：

1. 参数化方法：很多数据库都提供了参数化的方法，使用这些方法执行 SQL 语句时，数据库服务器先是对 SQL 指令进行编译，然后再将参数代入，而不是将参数也当成语句的一部分。因此，即使参数中存在恶意片段，也不会被数据库执行。比如 Android 应用中可以使用 compileStatement 方法，将其中外部输入都用 “？”代替，然后在通过相应的 bind 方法对参数进行配置，这样就不会存在 SQL 注入问题。

2. 白名单验证：若开发者已经知道某些形式的参数不会导致 SQL 注入漏洞，则可以在执行数据库操作前将参数与这些安全的形式进行验证。比如通过 Pattern.matcher 与事先定义好的模式进行匹配，或者通过 String 提供的方法判断参数中是否存在敏感字符等。而本文在扫描前两层基本块中，若是执行到调用了白名单验证使用的方法，且其参数为正在检测的数据库执行语句的污点数据参数，就认为程序中对这条 SQL 方法进行过合法性检查。

3. 过滤、转义敏感字符：实现过滤或转义敏感字符的一个常用方法就是使用 replace 之类的方法对表格 1 中的敏感字符进行操作，如图 5 中第 6 行使用的 Matcher.replaceAll 方法。当扫描的前两个基本块中包含对敏感字符的过滤或者转义处理，也能认为应用中进行了合法性检查。

如算法 3 所示，本文只考虑使用了污染数据的 SQL 方法所在基本块及前两层的所有基本块，扫描这些基本块中是否对指定的 SQL 参数进行过合法性检查。如图 6 中的代码片段，第 10 行中的 SQL 方法 rawQuery 位于基本块 B4 中，且其参数 *sql* 为污点数据。则先检查 B4 在执行该方法前是否对 SQL 注入进行过合法性检查，若没进行过就依次扫描上一层基本块 B3、B2，若上一层仍未进行合法性检查，就再依次扫描两层前的基本块，该示例中仅有基本块 B1。最终，若 SQL 方法所在基本块及前两层基本块都未进行过合法性检查，就可以认为这个使用了污点数据的 SQL 方法存在 SQL 注入漏洞。在这个示例中，位于基本块 B3 的第 6 行代码进行了合法性检查，则第 10 行的数据库执行方法是安全的。

算法 3：合法性检查判断

输入： File *config* //*合法性检查相关方法配置文件*
String *pam* //*SQL 方法中使用的污染参数*
CFG *cfg*， Method *method*， Location *loc*
输出： Boolean //*是否进行过合法性检查*
过程：

```
1)  Block block=loc.getBasicBlock();//当前基本块
2)  if(checkLegit(block)) //检查 block 中指令是否
        调用了 config 中方法，且其参数为 pam
3)      return true;
4)  Block[] pb = block.getPrev();//当前基本块上一
                                层所有基本块
5)  for(preBlock : pb){
6)      if(checkLegit(preBlock))
7)          return true;
8)  }
9)  for(preBlock : pb){
10)     for(pre2Block : preBlock.getPrev()){
11)         if(checkLegit(preBlock))
12)             return true;
13)     }
14) }
15) return false;
```

4 工具实现与实验

4.1 原型工具实现

在 Java 代码静态分析工具 FindBugs 的基础上，本文实现了基于污点分析的 Android 应用 SQL 注入漏洞静态检测原型工具 SQLInj。SQLInj 首先静态分析 Android 应用的字节码文件，在程序中定位 SQL 注入漏洞的 SQL 方法和 SQL 参数，然后通过静态污点分析技术检测 SQL 语句中的污点数据，最后判断程序是否对 SQL 方法中使用的污点数据进行过合法性检查来报告是否存在 SQL 注入漏洞。如图 7 所示，SQLInj 的总体框架主要分为程序静态分析、污点分析及合法性检查 3 个模块。

4.1.2 程序静态分析

FindBugs 中实现了对字节码的控制流分析，在控制流图基础上，能得到每条指令的数据流信息。FindBugs 中提供了很多种数据流信息，如当前指令所活跃的数据是否为常量、空值等，本文使用 ConstantDataflow 和 ValueNumberDataflow 来判断参数是否为常量字符串及其名称，并对 FindBugs 中提供的 ValueNumberDataflow 相关方法进行了修改，使得在进行 ValueNumber 数据流分析后，能同时获取局部变量和全局变量的名称。

4.1.3 静态污点分析

静态污点分析功能需要两个配置文件：source.config 和 derivation.config。其中，source.config 文件中记录了与污染源有关的方法，在静态污点分析中，根据其配置污染源并初始化污点数据集。derivation.config 文件中是能够将污点标记从污染数据传播给新数据的方法。首先，按照配置文件识别污染数据；然后，根据污点传播规则将所有被外部输入影响的方法和变量进行标记；最后，判断当前的 SQL 方法是否使用了被污染的参数。

```
1. B1: String str = m_et.getText().toString();
2. B1: Pattern p = Pattern.compile(regEx);
3. B1: Matcher m = p.matcher(str);
4. B1: String sql = "";
5. B1: if(m.matches()){
6. B3:     sql = "SELECT * FROM usertable WHERE _id = '" +
       m.replaceAll("").trim() + "'";
7.     } else {
8. B2:     sql = "SELECT * FROM usertable WHERE _id = '" + str + "'";
9.     }
10. B4: Cursor cursor = m_db.rawQuery(sql , null);
```

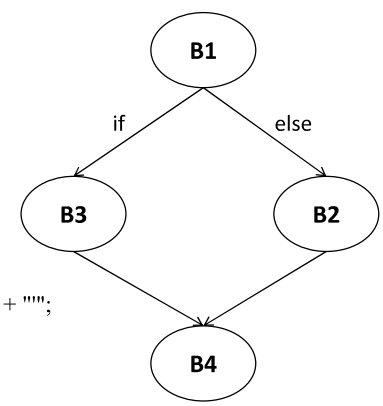

Fig. 6 Examples of legitimate check
图 6 合法性检查示例

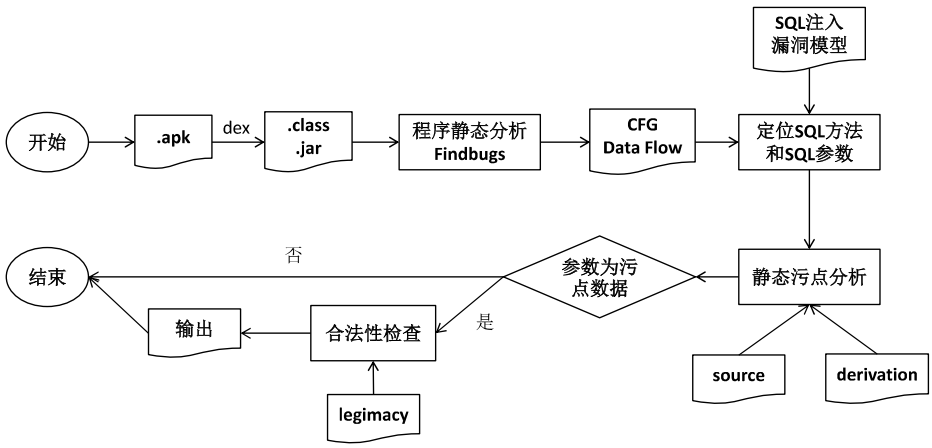

Fig. 7 General framework of SQLInj
图 7 SQLInj 总体框架

4.1.4 合法性检查

在配置文件 legimacy.config 中定义了 3.3 节中所介绍的合法性检查相关的方法。根据合法性检查判断规则，如果使用了污点数据的 SQL 方法所在的基本块以及前两层的基本块中使用过 legimacy.config 文件中定义的方法，即认为程序中对这条 SQL 方法进行过合法性检查。

4.1.5 检测结果报告

```
<terminated> FindBugs2 [Java Application] E:\Program\Java\jre1.8
H S SQL: com.example.sqlinject.MainActivity.SQL
InjectTest(String) passes a tainted data to met
hod: android.database.sqlite.SQLiteDatabase.raw
Query(Ljava/lang/String;[Ljava/lang/String;)Lan
droid/database/Cursor; At MainActivity.java:[li
ne 118]
```

Fig. 8 An example of SQLInj detect result
图 8 SQLInj 检测结果示例

当扫描完程序所有字节码文件后，SQLInj 会输出最终的 SQL 注入漏洞检测结果，报告所有 SQL 注入漏洞的 SQL 方法所在的类、方法、在源文件中的具体行数，以及具体是哪一个数据库执行方法等信息。图 8 为检测结果报告中的一条错误信息示例，从中能够看到该 SQL 注入漏洞发生在 MainActivity 类的 SQLInjectTest 方法中，漏洞产生的原因是在此方法中将一个污染数据传递给 SQL 方法 android.database.sqlite.SQLiteDatabase.rawQuery，该 SQL 方法位于源代码 MainActivity.java 的第 118 行。

4.2 实验设计与分析

使用 SQLInj 检测只需要 Android 应用的 APK 文件，但由于需要确认误报情况，因此实验选择了 SQLInject、sieve 等 6 个确定存在 SQL 注入漏洞的开源 Android 应用进行实验。如表 2 所示，6 个实验对象的规模从 500 行到 20000 行不等，表 2 中的第 3 列给出了应用功能的简要描述。

Table 2 Description of experimental objects
表 2 实验对象说明

应用名称	代码行数	应用功能
SQLInject	537	一个数据库查询应用，通过输入的 ID 显示数据库查询结果。
sieve	4563	一款密码管理器应用程序，通过身份验证后，管理设置的密码。
greenDAO	21510	一款对象关系映射工具，其能够将 java 对象映射到 SQLite 数据库中。
SQLiteMgr	4907	一个 Android 平台上的 SQLite 管理程序，其提供了各种 SQLite 数据库操作功能。
SimStuIfo	1188	一款 Android 设备中通过 SQLite 数据库管理学生信息的应用。
Creditcard	2963	一个基于 SQLite 的信用卡管理软件，能够管理信用卡并记录信用卡的还款和收款记录。

为评估本文所使用方法在检测 SQL 注入漏洞上的有效性，我们还扩展了 FindBugs 原有的 SQL 漏洞注入检测功能，在不改变其检测机制的基础上，使其能够支持检测 Android 应用 SQL 注入漏洞。将 FindBugs 运行的结果与 SQLInj 的结果作对比，可以验证使用污点分析及合法性检查判断对检测结果精度的影响。

4.3 实验结果分析

表 3 是 SQLInj 对 6 个应用的检测结果，其中已知漏洞为每个应用中已知存在 SQL 注入漏洞的 SQL 方法数量，污染方法是进行静态污点分析发现使用了污点数据的 SQL 方法的数量，疑似漏洞数是通过合法性检查判断后，存在 SQL 注入漏洞的 SQL 方法数量。然后，通过对每个应用中检测出的疑似漏洞进行判断，分析此处是否真实存在 SQL 注入漏洞。经分析确实存在的 SQL 注入漏洞为确认漏洞，未被检测到的 SQL 注入漏洞为漏报漏洞。表中第 6 列和第 7 列为 SQLInj 的漏报情况。实验结果表明，SQLInj 在 6 个 Android 应用中共检测出 35 个 SQL 注入漏洞。对检测出的 SQL 注入漏洞进行检查发现，对其中 4 个应用的分析未出现漏报，2 个应用的分析存在漏报，平均漏报率为 9.1%。对产生漏报的原因进行分析发现，在进行污点分析时，为了提高分析效率，采取了记录执行过的过程内污点分析方法的参数及返回值等污点信息，当再次调用同一方法时，不再进行过程内污点分析，直接采用之前的污点分析信息。这一措施导致若某方法使用的全局变量在首次调用时与外部输入无关，但在之后的调用中与外部输入相关时，会产生漏报。实验结果表明，本文所使用的方法能有效检测出 Android 应用中的 90%以上的 SQL 注入漏洞，漏报率较低。而 FindBugs 通过检测 SQL 方法的参数是否为常量字符串来判断此处是否存在 SQL 注入漏洞，所以基本不会发生漏报的现象，但精确度较低。

Table 4 Comparison of false positive
表 4 误报情况对比

应用名称	*FindBugs*		*SQLInj*	
	误报数	误报率	误报数	误报率
SQLInject	14	73.7%	0	0%
sieve	3	60%	0	0%
greenDAO	63	85.1%	3	25%
SQLiteMgr	16	59.3%	0	0%
SimStuIfo	7	63.6%	1	20%
Creditcard	4	33.3%	0	0%
合计	107	62.5%	4	7.5%

SQLInj 与 FindBugs 的误报情况对比如表 4 所示。使用 FindBugs 分析的误报率最低为 33.3%，最高的高达 85.1%，平均误报率为 62.5%。使用 SQLInj 的误报率最低为 0%，最高为 25%，平均误报率为 7.5%。这是因为只要数据库执行方法的参数为诸如方法返回值、外部传参、字符串拼接等，FindBugs 就会认为存在 SQL 注入漏洞，而 SQLInj 只有当 SQL 方法的参数与外部输入有关且未进行合法性检查时才认为存在 SQL 注入漏洞。SQLInj 存在误报的原因是 SQLInj 不能检测出应用在 SQL 方法前两层基

Table 3 Detection results of SQLInj
表 3 SQLInj 检测结果

应用名称	已知漏洞	污点变量	污染方法	疑似漏洞	确认漏洞	漏报漏洞	漏报率
SQLInject	5	33	7	5	5	0	0%
sieve	2	12	2	2	2	0	0%
greenDAO	11	89	15	12	9	2	18.2%
SQLiteMgr	11	128	11	7	7	4	36.4%
SimStuIfo	4	28	5	5	4	0	0%
Creditcard	8	42	8	8	8	0	0%
合计	41	332	48	39	35	6	9.1%

本块之前进行的敏感字符过滤等合法性检查，误报存在 SQL 注入漏洞。实验结果表明，本文所使用的基于污点分析的 SQL 注入漏洞静态检测方法有效降低了误报率。

此外，我们还使用了综合评价指标来评价 SQLInj 和 FindBugs 检测结果的综合性能。我们将检测结果的确认漏洞称为 *TP*，表示检测存在 SQL 注入漏洞且确实存在 SQL 注入漏洞的 SQL 方法数量。将疑似漏洞与确实漏洞的差称为 *FP*，表示检测存在 SQL 注入漏洞，但其实是安全的 SQL 方法数量。将已知漏洞与确实漏洞的差称为 *FN*，表示没有检测出但实际存在 SQL 注入漏洞的 SQL 方法数量。准确率 $P = TP/(TP+FP)$，表示被正确检测出的 SQL 方法中确实存在 SQL 注入漏洞的比例；召回率 $R = TP/(TP+FN)$，表示所有确实存在 SQL 注入漏洞的 SQL 方法中被正确检测出来的比例；综合评价指标 F 值 $F = 2PR/(P+R)$，表示准确率和召回率的加权调和平均，综合了准确率和召回率的结果，有利于对检测结果综合性能的评价[15]。

SQLInj 与 FindBugs 的综合评价标准 *F* 值对比结果如表 5 所示。FindBugs 的 *F* 值最低为 0.27，最高为 0.80，平均值为 0.53。SQLInj 的 *F* 值最低为 0.78，最高为 1，平均值为 0.91，比 FindBugs 高了约 38 个百分点。因此，本文所使用的基于污点分析的 SQL 注入漏洞静态检测方法的综合性能更好。

Table 5 Comprehensive contrast between FindBugs and SQLInj
表 5 FindBugs 与 SQLInj 综合对比情况

应用名称	*FindBugs*			*SQLInj*		
	P	**R**	**F**	**P**	**R**	**F**
SQLInject	0.26	1	0.41	1	1	1
sieve	0.4	1	0.57	1	1	1
greenDAO	0.15	1	0.27	0.75	0.82	0.78
SQLiteMgr	0.41	1	0.58	1	0.64	0.78
SimStuIfo	0.37	1	0.54	0.8	1	0.89
Creditcard	0.67	1	0.80	1	1	1
合计	0.38	1	0.53	0.93	0.91	0.91

4.4 有效性影响因素分析

本文提出的方法能够有效地检测出 Android 应用中存在的 SQL 注入漏洞，并降低了误报率。但是该方法的效果仍受到以下方面限制：

1. 实验数据集的代表性。为了验证 SQLInj 检测结果的正确性，我们选择使用开源的 Android 应用进行实验，而且样本数量也还不够，需要提高实验数据集的代表性。

2. 本方法仍存在漏报问题。主要原因为 FindBugs 按照调用树自底向上扫描方法所在 class 文件中的所有方法，这导致个别方法在被调用前就已经被扫描过了，当此方法被调用时若其所在类的全局变量曾被改变，将使得此方法返回值污染信息发生变化，但如果此方法参数的污点信息与第一次扫描时相同，不会再重新扫描，可能会产生漏报。此外，合法性检查不够精确也可能会导致漏报。我们通过扫描 SQL 方法前两层基本块中是否进行了合法性检查来判断是否存在 SQL 注入，若程序中进行了合法性检查，但检查条件并不准确，可能会导致检查后 SQL 参数中仍存在会导致 SQL 注入的字符串，从而引起漏报。

3. 本方法仍存在误报问题。我们只扫描 SQL 方法前两层基本块中是否进行了合法性检查，当程序中合法性检查的位置与 SQL 方法距离较远时，可能会导致产生误报。

5 相关研究现状

由于 SQL 注入漏洞的风险性以及普遍性，如何检测 SQL 注入漏洞的问题引起了很多国内外学者的注意，目前已经做出了大量的研究工作，主要可分为静态分析和动态测试两大类。另外，因为本文方法主要基于污点分析技术，因此对使用污点分析的相关工作也进行了总结。

5.1 静态分析技术

程序静态分析不需要程序实际运行，而是在编译时就获得程序的相关属性，在此基础上对程序进行分析的技术。文献[16]中对近几年使用静态检测 SQL 注入的方法进行了分析。符号执行技术是静态分析的一种，用来决定是哪些输入导致程序每个片段执行的分析方法。当使用符号执行来分析程序时，

程序将符号作为值赋给变量，对程序的路径进行模拟执行，精确分析程序的代码属性[2][17]。文献[18][19]中使用静态方法检测 Web 应用中的 SQL 注入漏洞，文献[20][21][22]则是通过静态方法针对 Android 应用进行检测。

除此之外，Coverity[5]和 FindBugs[9]是使用较多的静态软件质量及安全漏洞检测工具。其中，Coverity 使用过程间静态分析、字节精度分析、虚假路径剪枝等技术对软件源码进行静态分析，并提供了完整的路径覆盖，确保每一行代码以及潜在的执行路径都能被测试；FindBugs 则是对应用的字节码文件进行数据流分析，并与漏洞缺陷模式进行匹配。但是，两者的 SQL 注入漏洞检测功能主要是针对 Web 应用的，无法对 Android 应用进行检测。

5.2 动态分析技术

动态分析技术是将程序在真实或虚拟机上运行，并对运行过程进行分析的方法。在静态分析中，有很多情况被认为是可能发生的，但在动态分析中，其分析的路径都是动态执行，确实发生了的，而且可以观察到程序运行时的状态。B. Chess 和 J. West 在 2008 年提出了一种检测漏洞的动态分析方法[23]。在 SQL 注入问题上，也有很多方法是使用动态分析技术检测的，如文献[24][25][26]是动态检测 Web 应用中的 SQL 注入漏洞，文献[27]则将动态技术应用到检测 Android 应用上的 SQL 注入。

5.3 污点分析技术

污点分析可以分为静态和动态污点分析两种。文献[28][29]中使用静态污点分析技术检测 Android 应用中的安全漏洞，文献[30]则是提出一种在 Java 虚拟机中的动态污点分析方法，并结合了静态分析方法来收集可到达的方法，以减少运行时开销。在 Android 应用污点分析中，FlowDroid[10]和 TaintDroid[31]是两个具有代表性的工具。FlowDroid 是一款针对 Android 应用的上下文、流、字段、对象敏感和生存周期感知的静态污点分析工具，但其不支持 SQL 注入漏洞的检测。TaintDroid 则是通过动态污点分析跟踪信息流的流动，但只会显示数据流动信息，而不会对漏洞进行检测。

6 结束语

本文针对 Android 应用中的 SQL 注入漏洞，提出一种基于污点分析的 SQL 注入漏洞静态检测方法。该方法首先对应用进行静态分析，在程序中定位 SQL 注入漏洞的 SQL 方法和 SQL 参数；然后通过静态污点分析方法检测 SQL 方法是否使用了污点数据；最后识别应用中是否存在对污点数据进行过合法性检查。基于上述方法，实现了原型工具 SQLInj，并对存在 SQL 注入漏洞的 Android 应用进行了实验。

在本文研究的基础上，我们计划进一步研究如何通过 JPF-Android 等工具自动产生 SQL 注入攻击，以验证 SQL 注入漏洞的存在，并在检测出 SQL 注入后，尝试在源码中进行漏洞修复。

References:

[1] OWASP. OWASP Top Ten Project [EB/OL].[2017-07-11]. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Main.

[2] Weng Zisheng, Wang Baosheng, Lin Jinbin. Array analysis in program symbol execution [J]. Journal of Yangtze University (Natural Science Edition), 2010, 7(1):225-228.

[3] ZHONG Fang-ting,LIU Chao,JIN Mao-zhong. Improvement of instrumentation in program dynamic analysis system [J]. Computer Engineering and Design, 2007, 28(19):4585-4588.

[4] Hewlett Packard Enterprise. Fortify Static Code Analyzer[EB/OL].[2017-07-01].https://saas.hpe.com/zh-cn/software/sca.

[5] Coverity. Coverity Development Testing Platform[EB/OL].[2017-07-12].http://www.coverity.com/html_cn/w-content/uploads/2014/01/DS_Coverity_Solutions_Overview.pdf

[6] CNNIC.China Statistical Report on Internet Development [EB/OL].[2017-07-10].http://www.cnnic.cn/hlwfzyj/hlwxzbg/hlwtjbg/201701/P020170123364672657408.pdf

[7] CNNIC. Report on Network Security Status of Chinese Mobile Internet Users in 2015 [EB/OL].[2017-07-10]. http://www.cnnic.cn/gywm/xwzx/rdxw/2016/201610/P020161010424290641484.pdf

[8] Veracode. Veracode state of software security report.[EB/OL].[2017-07-06].https://info.veracode.com/state-of-software-security-report.html

[9] Hovemeyer D, Pugh W. Finding bugs is easy[J]. Acm Sigplan Notices, 2004, 39(12):92-106.

[10] Arzt S, Rasthofer S, Fritz C, et al. FlowDroid:precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[J]. Acm Sigplan Notices, 2014, 49(6):259-269.

[11] Chen Xiaobing, Zhang Hanyu, Luo Liming, Huang He. Resear ch on technique of SQL injection attacks and detection.Computer Engineering and Applications, 2007,11:150-152+203.

[12] Dong Min. Rrsearch on The Attack Detection of SQL Injection based on Dynamic Analysis [D].Beijing University of Technology,2014.

[13] Zhou Yan. Research and implementation of SQL injection detection method [D].Northwest University,2011.

[14] Aho A V, Sethi R, Ullman J D. Compilers: principles, techniques, and tools[J]. 1986.

[15] Yong L I, Huang Z, Fang B, et al. Using Cost-Sensitive Classification for Software Defects Prediction[J]. Journal of Frontiers of Computer Science & Technology, 2014, 8(12):1442-1451.

[16] Gupta M K, Govil M C, Singh G. Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey[C]// Recent Advances and Innovations in Engineering. IEEE, 2014:1-5.

[17] Xu Chen. A Program Verification and Testing Tool Based on Symbolic Execution and Constraint Solving [D]. Chinese Academy of Sciences (Institute of Software), 2002.

[18] Son S, Mckinley K S, Shmatikov V. RoleCast: Finding Missing Security Checks When You Do Not Know What Checks Are[J]. Acm Sigplan Notices, 2011, 46(10):1069-1082.

[19] Ernst M D, Lovato A, Macedonio D, et al. Boolean Formulas for the Static Identification of Injection Attacks in Java[M]// Logic for Programming, Artificial Intelligence, and Reasoning. Springer Berlin Heidelberg, 2015.

[20] Zhang M, Yin H. AppSealer: Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications[C]// Network and Distributed System Security Symposium. 2014.

[21] Jiang Y Z X, Xuxian Z. Detecting passive content leaks and pollution in android applications[C]//Proceedings of the 20th Network and Distributed System Security Symposium (NDSS). 2013.

[22] Guowei D, Meilin W, Shuai S, et al. Android application security vulnerability analysis framework based on feature matching[J]. Journal of Tsinghua University (Science and Technology), 2016, 65(5): 461-467.

[23] B. Chess, J. West. Dynamic Taint Propagation: "Finding Vulnerabilities without Attacking" Information Security Technical Report. Volume 13,Issue 1, 2008, 33-39.

[24] Jang Y S, Choi J Y. Detecting SQL injection attacks using query result size[J]. Computers & Security, 2014, 44(2):104-118.

[25] Doshi J C, Christian M, Trivedi B H. SQL FILTER – SQL Injection Prevention and Logging Using Dynamic Network Filter[J]. Communications in Computer & Information Science, 2014, 467:400-406.

[26] Kumar D G, Chatterjee M. MAC based solution for SQL injection[J]. Journal of Computer Virology and Hacking Techniques, 2015, 1(11): 1-7.

[27] Hay R, Tripp O, Pistoia M. Dynamic detection of inter-application communication vulnerabilities in Android[C]// International Symposium on Software Testing and Analysis. ACM, 2015:118-128.

[28] Yue Hongzhou, Zhang Yuqing, Wang Wenjie, et al. Android Static Taint Analysis of Dynamic Loading and Reflection Mechanism[J]. Journal of Computer Research and Development, 2017, 54(2):313-327.

[29] Wang Yunchao, Wei Qiang, Wu Zehui. Approach of Android Applications Intent Injection Vulnerability Detection Based on Static Taint Analysis[J]. Computer Science, 2016, 43(9):192-196.

[30] Zhao J, Qi J, Zhou L, et al. Dynamic Taint Tracking of Web Application Based on Static Code Analysis[C]// International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. 2016:96-101.

[31] Enck W, Gilbert P, Chun B G, et al. TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones[J]. Acm Transactions on Computer Systems, 2014, 32(2):1-29.

PAN Qiuhong was born in 1995. She received the B.S. degree in computer science from Nanjing University in 2017. She is a master candidate at Nanjing University. Her research interests include software security, vulnerability detection.

潘秋红(1995–)，女，江苏徐州人，2017 年于南京大学获得学士学位，现为南京大学硕士研究生，主要研究领域为软件安全，漏洞检测。

CUI Zhanqi was born in 1984. He received the Ph.D. degree in computer science from Nanjing University in 2011. He is an assistant professor at Beijing Information Science and Technology University. His research interests include software analysis and testing.

崔展齐(1984-)，男，贵州金沙人，2011 年于南京大学获博士学位，现为北京信息科技大学讲师，主要研究领域为软件分析和测试。

WANG Linzhang was born in 1973. He received the Ph.D. degree in computer software and theory from Nanjing University in 2005. He is a professor at Nanjing University. His research interests include software engineering, information security.

王林章(1973–)，男，江苏建湖人，2005 年于南京大学获得博士学位，现任南京大学计算机科学与技术系教授。 主要研究领域为软件工程，信息安全。