计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):86−91 [doi: 10.15888/j.cnki.csa.006145]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Chopping 的 Web 应用 SQL 注入漏洞检测方法①

尤　枫, 马金慧, 张雅峰

(北京化工大学 信息科学与技术学院, 北京 100029)
通讯作者: 马金慧, E-mail: mjhttkl8899@163.com

摘　要: 随着 Web 应用的不断普及, 其安全问题越来越显突出, 特别是 SQL 注入漏洞攻击, 给用户的安全体验造成了巨大的威胁. 针对二阶 SQL 注入漏洞, 本文提出了一种基于 chopping 技术的二阶 SQL 注入漏洞检测方法. 首先通过对待测应用程序进行 chopping, 获取到一阶 SQL 注入疑似路径; 然后对一阶 SQL 注入疑似路径中的 SQL 语句进行分析, 确定二阶 SQL 注入操作对, 进而得到二阶 SQL 注入疑似路径; 最后通过构造攻击向量并运行, 确认二阶 SQL 注入疑似路径中漏洞是否实际存在. 实验结果表明, 本方法能够有效地检测出二阶 SQL 注入漏洞.

关键词: 二阶 SQL 注入; 污点分析; Web 应用; 漏洞检测; chopping; 程序切片

引用格式: 尤枫,马金慧,张雅峰.基于 Chopping 的 Web 应用 SQL 注入漏洞检测方法.计算机系统应用,2018,27(1):86–91. http://www.c-s-a.org.cn/1003-3254/6145.html

Web Application SQL Injection Detection Method Based on Chopping

YOU Feng, MA Jin-Hui, ZHANG Ya-Feng

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: With the wide use of Web application in recent years, its security has seriously affected the experience of relevant users. In particular, the SQL injection vulnerability attack is the most commonly used technique in attacking Web applications. In this study, a chopping-based method is proposed to detect the second-order SQL injection. Firstly, the static analysis based on chopping had been used to find the suspected first-order SQL injection paths. Secondly, the SQL statements in those suspected paths should be used to get the second-order operation pairs, which had been used to get the suspected second-order SQL injection paths. Finally, the attack vector is constructed to confirm the existence of the vulnerabilities which are hiding in Web applications. The experimental results show that the method we propose in this study can effectively detect the second-order SQL injection vulnerabilities thus prevent Web applications from SQL injection attack.

Key words: second-order SQL injection; taint analysis; Web application; vulnerability detection; chopping; program slicing

1　概述

随着互联网的普及与发展, Web 技术以其广泛性、交互性、快捷性和易用性等特点迅速风靡全球, 已渗透到社会生活的各个领域. 与此同时, Web 应用的安全也越来越受到人们的关注, SQL 注入、XSS 等 Web 安全问题层出不穷[1]. 作为 OWASP (Open Web Application Security Project) Top 10 Project[2]中排名靠前的 Web 应用安全隐患, SQL 注入漏洞一直困扰着广大的 Web 应用用户. 目前, 针对 SQL 注入漏洞检测的研究可以为两类: 一类是通过静态分析程序源代码, 匹

① 基金项目: 国家自然科学基金 (61672085, 61472025)
收稿时间: 2017-03-28; 修改时间: 2017-04-20; 采用时间: 2017-05-02; csa 在线出版时间: 2017-12-22

配漏洞的编码规则查找漏洞[3,4]; 另一类是通过动态爬虫爬取所有的网页链接, 逐个网页查找注入点, 提交攻击, 使用黑盒的方法进行检测[5,6]. 目前, 大多数相关研究主要是针对一阶 SQL 注入, 二阶 SQL 注入由于其原理的复杂性及发作的潜伏性, 较难发现和检测, 因此, 针对二阶 SQL 注入的研究较少, 且效果不理想.

Dahse 和 Holz[7]提出了一个较完整的针对 PHP 语言应用程序的二阶 SQL 注入检测方法. 该方法将数据库、session 以及文件等数据持久存储介质作为污点信息的输入源, 能够检测存储型 XSS、SQL 注入等几种二阶漏洞. 但由于 PHP 是解释执行的服务器脚本语言, 采用面向过程的开发方法, 所以该文提出的检测方法对 Java 等其他语言开发的应用程序并不完全适用. Yan L 等人[8]提出先查找源代码中的 SQL 语句, 然后为 SQL 语句的每个操作字段建立数据项, 通过对比数据项将二阶 SQL 注入的存入和触发过程查找出来, 以达到检测二阶 SQL 注入的目的. 但是该方法的静态分析阶段未实现自动化, 检测效果也并不理想. W. G. Halfond 和 A. Orso[9]提出通过对正常和异常查询代码建模, 与运行时查询动态建模进行对比, 以检测当前查询是否异常. 该方法检测的准确性取决于建模的准确性, 且运行时系统消耗太大.

本文针对 Java Web 应用程序, 提出了基于 chopping 技术、综合利用静态分析和动态检测技术的二阶 SQL 注入漏洞检测方法, 以提高检测的自动化程度和检测效果.

2 相关背景介绍

2.1 污点分析

污点分析是一种针对源程序文件进行的分析方法, 广泛应用于软件测试等领域. 污点分析的过程是将所有外部输入程序的数据标记为污点数据, 识别污点数据在程序中的传播路径, 即在数据进入 source 点 (污点数据接收点) 时进行标记, 利用特定的规则跟踪分析污点数据在程序中的传播过程, 当数据传播到 sink 点 (对污点数据进行分析的点) 时检查关键操作是否会受到污点数据的影响.

污点分析有静态和动态之分, 静态污点分析是指不考虑特定输入, 所有污点传播过程中可能涉及到的语句都进行查找, 优点是能够快速定位污点在程序中的所有出现情况. 动态污点分析是在程序有特定输入的情况下, 对运行过程涉及到的数据流或控制流进行监控, 从而实现对数据传播、使用进行跟踪和检测.

2.2 SQL 注入漏洞

SQL 注入就是通过把故意设计的 SQL 语句插入到表单提交或页面请求的查询字符串中, 欺骗服务器执行恶意的 SQL 命令, 而不是按照合法用户的原本意图去执行 SQL 命令.

SQL 注入漏洞包括一阶 SQL 注入漏洞和二阶 SQL 注入漏洞.

对于一阶的 SQL 注入来说, 污点数据的传播路径为数据从 source 点输入, 经过路径传播到达 sink 点并执行 SQL 语句, 从而引发 SQL 注入攻击.

二阶 SQL 注入的传播路径为数据从 source 点输入, 经过路径传播到达 sink 点并存入数据库, 但不会立即执行, 而是等待下一个调用该数据的 SQL 语句执行, 数据再次从 sink 点流出并引发 SQL 注入攻击.

2.3 程序切片

程序切片最早是由 Mark W.提出的[10], 经过多年的不断研究, 程序切片技术从后向切片发展到前向切片, 也从一开始的面向过程的程序切片发展到面向对象的程序切片技术[11], 还衍生出程序削片 (dicing)[12]、程序砍片 (chopping)[13]等新型程序切片技术.

后向程序切片就是查找影响到某条语句的所有语句的程序切片的过程. 前向程序切片就是查找被某条语句所影响到的所有语句的程序切片的过程.

Chopping 是由 D. Jackson 等[14]提出的, chopping 准则由变量定义集和变量引用集两个变量集合组成, 对一个程序进行砍片的过程就是识别程序中所有说明在定义集中的变量影响引用集中变量值的语句子集的过程, 即查找定义集中的点的前向切片与引用集中的点的后向切片的交集的过程. 当与污点分析结合使用时, 定义集即是 source 点集, 引用集即是 sink 点集.

程序切片过程分为程序内切片和程序间切片过程. 程序内切片是指针对一个程序进行的切片, 程序间切片是指针对不同程序进行的切片.

3 Web 应用 SQL 注入漏洞检测

本文所提出方法的具体流程如图 1 所示, 主要分为四个部分. 首先进行程序预处理, 将 Java 代码进行词法和语法分析, 并转化为含有数据依赖边和控制依赖边的系统依赖图; 然后通过 chopping 规则获取含有疑

似一阶 SQL 注入数据传播路径的 chop; 再后通过对一阶疑似路径分析匹配, 获取二阶 SQL 注入疑似路径; 最后针对疑似路径, 自动生成攻击向量并提交运行, 验证二阶疑似路径是否为二阶 SQL 注入漏洞.

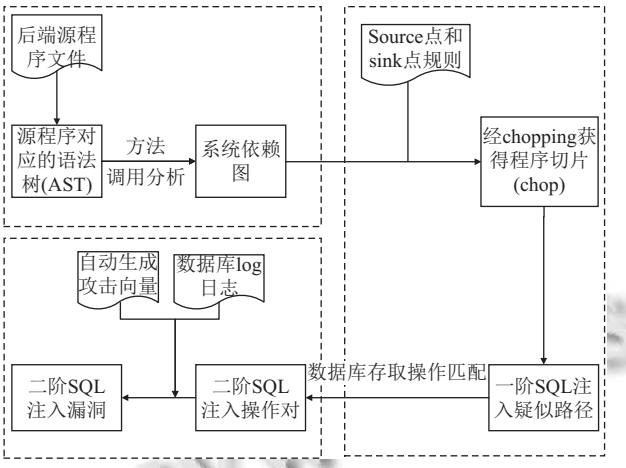

图 1 系统总体框图

3.1 预处理

JavaCC (Java Compiler Compiler)[15]是一个用 Java 开发的词法和语法分析生成器. 通过使用由 JavaCC 生成的词法分析和语法分析器对 Java Web 应用程序的后端代码进行分析, 根据得到的抽象语法树进行数据依赖分析、控制依赖分析以及函数调用关系分析生成系统依赖图 (SDG, System Dependence Graph)[16]. 一个系统依赖图由程序中的所有程序依赖图 (PDG, Program Dependence Graph) 组成, 其反映的是系统内程序之间的调用关系. 程序依赖图是有向图 G=(N, E), 其中 N 是表示给定程序的语句的节点集合, E 是控制依赖边和数据依赖边的集合.

3.2 一阶 SQL 注入疑似路径获取

针对 Java 语言的特点, 本文对所有接收外部输入和操作数据库的函数进行了详细分析, 归纳形成与漏洞相关的 source 点集和 sink 点集, 作为后续分析的基础.

根据 chopping 规则对系统依赖图进行 chopping, 获取 source 点和 sink 点之间的污点数据传播路径, 经过 chopping 之后获得的程序片段 (chop) 即为一阶 SQL 注入疑似路径, 也就是符合一阶 SQL 注入污点数据传播方式的路径, 数据在这些路径上传播时, 有可能会触发一阶 SQL 注入. 一阶 SQL 注入疑似路径获取算法如算法 1 所示.

算法1. 一阶SQL注入疑似路径获取算法

输入: SDG =(V, E)

输出: 一阶SQL注入疑似路径(chop)

1. S、T为给定的chopping准则, S是source点, T为sink点, $S \subset V$, $T \subset V$
2. $t \in T$, $s \in S$
3. while T不为空
5. 从T删除一个元素t
6. for each $j \to t \in E$
7. if j尚未标记
8. then将j添加到到t后向切片集, 为j添加后向访问阶段标记, 继续添加该路径的后向访问边节点
9. 查找所有的后向切片集$BS(t) = \{j \in N | * j \to t\}$
10.end
11.while S不为空
12. 从S删除一个元素s
13 for each $s \to j \in E$
14. if j未被标记前向阶段标志
15. then将j添加到到s前向切片集, j添加前向阶段标记, 继续添加该路径上的前向访问边节点
16. 查找所有的前向切片集$FS(s) = \{j \in N | s \to * j\}$
17. end
18. 求交集chop(s,t) = $FS(s) \cap BS(t)$
19. return chop(s, t)

Chopping 过程采用由 source 点进行前向切片和 sink 点进行后向切片求交集的方法来获取. 区别于单纯基于 sink 点进行后向切片, 或者基于 source 点进行前向切片, 程序 chop 是后向切片集合和前向切片集合的交集, 能够过滤掉一部分从 source 出发不能到达 sink 点以及从 sink 点回溯不能到达 source 点的无效路径, 在接下来的检测中, 能够减少相当一部分的审计工作, 对于程序分析和漏洞查找提供更精确的结果, 降低分析成本. 由于 SDG 中包含了同一项目不同文件之间的函数调用关系, 在分析过程中会涉及到程序内切片和程序间切片.

3.3 二阶 SQL 注入疑似路径查找

对一阶 SQL 注入疑似路径进行分析, 根据路径中 SQL 语句所操作的数据库表对语句进行归类, 将有相同操作对象的 SQL 语句进行匹配分析, 从而查找出对相同数据库表进行存、取操作的路径.

定义 1. 二阶注入操作对: 由一个将攻击代码存入数据库的 SQL 操作和一个从数据库中将攻击代码取出使用的 SQL 操作组成的 SQL 操作对.

数据写入: 攻击者将攻击性代码写入数据库的过程.

此时的数据传播路径是: source→sink→数据库. 恶意数据写入数据库之后不会被立刻使用.

数据取出再使用: 将数据写入阶段已存储在数据库中的攻击代码取出来, 拼接到新的 SQL 语句中, 随后运行, 导致执行结果出现异常.

此时的数据传播路径是: 数据库→sink→数据使用, 攻击“发作”, 造成数据库恶意访问.

查找二阶 SQL 注入漏洞的关键就是查找二阶注入操作对.

通过分析一阶 SQL 注入疑似路径, 分别查找数据写入部分和数据取出再使用部分, 通过对一阶 SQL 注入疑似路径中构造的 SQL 语句进行分析, 若两条一阶疑似路径中的 SQL 语句所操作的数据库表相同, 且两条语句分别为存操作和取操作, 则将这两个 SQL 语句匹配为二阶注入操作对, 这两条一阶的疑似路径匹配为一条二阶 SQL 注入疑似路径. 具体实施步骤如下:

步骤 1. 将一阶疑似路径中所包含的 SQL 语句根据所操作对象及数据库表进行汇总.

步骤 2. 确定每个 SQL 语句的操作类型, 若有嵌套查询, 则以外层为基准. 通过判断 SQL 语句中的是否有 from 子句以及 where 子句判断 SQL 语句是属于数据写入还是取出再使用, 分别标记操作类型为存或取.

步骤 3. 将操作对象相同且操作类型不同的 SQL 语句结合为二阶 SQL 注入操作对, SQL 语句所在的两条一阶的疑似路径匹配为一条二阶 SQL 注入疑似路径.

通过一阶 SQL 注入疑似路径获取二阶 SQL 注入疑似路径的算法如算法 2 所示.

算法2. 二阶SQL注入疑似路径获取算法

输入: 一阶SQL注入疑似路径FP

输出: 二阶SQL注入疑似路径SP

```
1. 分析FP中所有fp的数据库操作对象表db(fp)
2. while FP不为空
3.    if db(fp1)=db(fp2)
4.    then分析fp1和fp2的数据库操作类型m(fp1)和m(fp2)
5.    if fp中含有from子句
6.       if fp中含有where子句
7.          fp标记为存
8.       else fp标记为取
9.       if (m(fp1)=存AND m(fp2)=取)OR(m(fp1)=取AND m(fp2)=存)
10.       fp1和fp2匹配为一个二阶SQL注入疑似路径sp(fp1, fp2)
11.       SP=SP∪sp(fp1, fp2)
12. end
13. return SP
```

3.4 二阶 SQL 注入漏洞的验证

对于查找到的二阶 SQL 注入疑似路径, 需要进行实际验证, 确认是否是真正的二阶 SQL 注入漏洞.

3.4.1 攻击向量的设计与执行

攻击向量是指包含异常攻击字段的 SQL 语句. 攻击字段的设计方法有七类[17], 分别是: 大小写混合、替换关键字、使用编码、使用注释、等价函数与命令、特殊符号、整合绕过, 这些攻击向量同样适用于二阶 SQL 注入漏洞.

对收集到的攻击向量进行整理分类, 并在其基础上按照规则进行变异处理.

变异遵循以下规则: 在保证逻辑关系合理的情况下, 分别进行字母大小写替换、编码处理、数字和字符串值随机化、添加多个重复的标签、添加“+”“–”或“.”等用于连接字符的符号. 经过组合及变异后的攻击向量的多样性会得到很大提升.

BurpProxy 是浏览器代理工具 BurpSuite 的核心部分, 是一个拦截 HTTP/HTTPS 请求的代理服务器, 作为一个在浏览器和目标应用程序之间的中间代理, 允许拦截、查看和修改两个方向上的原始数据流.

通过使用 BurpProxy 运行异常的 SQL 代码; 截获浏览器的 http 请求, 修改参数, 将攻击代码嵌入到请求参数中, 最终提交到数据库.

3.4.2 攻击向量的执行结果分析

运行异常攻击向量, 查看数据库日志中的 SQL 语句执行记录, 然后提交正常的 SQL 命令, 再次查看数据库日志, 将两次运行结果进行对比, 若两条数据库结果不同, 则说明提交的异常请求发生了篡改, 确认程序存在漏洞.

4 实验结果及分析

4.1 实验设计

本实验的硬件配置为 2.20 GHz, 2.20 G RAM 笔记本计算机; 系统环境是 Windows7, 使用 JDK 1.8.0_91_、MySQL 5.6.28、Java 语言在 Eclipse 下完成.

根据本文提出的方法, 开发了一个二阶 SQL 注入漏洞检测程序, 实现对应用程序后台代码进行 chopping 以查找一阶 SQL 注入疑似路径以及进一步匹配查找二阶 SQL 注入疑似的路径. 最后通过生成大量的攻击向量并利用浏览器代理工具运行, 从而验证所查找到的二阶 SQL 注入疑似路径, 确定二阶 SQL 注入漏洞的存在.

下面针对 4 个 Java Web 应用程序进行实验.

4.2 实验结果及分析

本文实验基于 4 个 Java Web 应用程序, 基本信息如表 1 所示.

表 1 应用程序基本信息表

应用程序名称	应用程序说明	后台代码行数	文件数
宿舍管理系统 Dormitory Manager	安排学生入住及寝室调换等	5100	60
网上书店 BookStore	图书, 售价, 分类、订单信息等	4838	55
图书馆管理系统 JSPLibrary	图书档案管理、借阅归还管理	3352	33
校园精品课程网 Classnet	CMS系统, 新闻发布、课程管理、资源下载管理功能	12005	101

4.2.1 一阶 SQL 注入疑似路径查找

程序首先以四个应用程序后台代码作为输入, 进行预处理并针对 source 点和 sink 点进行 chopping, 四个应用程序查找到的一阶 SQL 注入疑似路径数如表 2 所示.

表 2 一阶疑似路径数列表

应用程序名称	一阶SQL注入疑似路径数
Dormitory Manager	6
BookStore	11
JSPLibrary	12
Classnet	17

4.2.2 确定有效的二阶 SQL 注入操作对

接下来, 程序将会对前一步所获取的四个应用程序中的一阶 SQL 注入疑似路径进行分析. 按照存、取操作的划分原则, 通过字符匹配, 对每个切片所操纵的数据库表进行归类, 确定每个切片操作的类型, 即存操作还是取操作, 操作对象相同且操作类型不同的匹配为二阶 SQL 注入操作对, 最终得到结果如表 3 所示.

表 3 中数据表示在一应用程序中, 操作对所在的文件名及操作针对的数据库以及操作类型.

由表 3 中数据可见, 四个应用程序中的二阶注入操作对个数分别为 1 个、3 个、4 个、5 个, 即四个应用程序中分别匹配出 1 个、3 个、4 个和 5 个二阶 SQL 注入疑似路径.

4.2.3 攻击向量验证实验

针对二阶注入操作对进行模拟攻击, 通过与数据库日志中的 SQL 语句记录进行对比, 验证攻击向量是否触发了二阶 SQL 注入.

表 3 二阶注入操作对分析结果

应用程序名称	操作对所在文件	操作的数据库表	操作类型
Dormitory Manage	StudentDao.java	student	存/取
BookStore	BookDaoImpl.java	d_book	存/取
	CategoryDaoImpl.java	d_category	存/取
	UserDaoImpl.java	d_user	存/取
JSPLibrary	BookDaoImpl.java	d_book	存/取
	CategoryDaoImpl.java	d_category	存/取
	ReceiveAddressDaoImpl.java	d_receive address	存/取
	UserDaoImpl.java	d_user	存/取
Classnet	ClazzDaoImpl.java	clazz_table	存/取
	NewsDaoImpl.java	news	存/取
	SourceDaoImpl.java	source_table	存/取
	TopicDaoImpl.java	Topic_table	存/取
	UserHomeworkDaoImpl.java	Userhoume-work_table	存/取

如下为针对程序 Dormitory Manager 自动生成的异常攻击向量的示例:

(1) http://localhost/sushe/StudentDel.action?Student_ID=6 uNIoNsELecT 1, 2, 3, 4

(2) http://localhost/sushe/StudentDel.action?Student_ID=6UNIunionONSELselectECT 1, 2, 3, 4

(3) http://localhost/sushe/StudentDel.action?Student_ID=6/*!u%6eion*//*!se%6cect*/ 1, 2, 3, 4…

(4) http://localhost/sushe/StudentDel.action?Student_ID=6%55nION/**/%53ElecT 1, 2, 3, 4

(5) select+id-1+1.from users.

经过运行攻击向量验证, 确认的二阶 SQL 注入漏洞个数如表 4 所示.

表 4 检测出的二阶 SQL 注入漏洞个数

应用程序名称	二阶SQL注入疑似漏洞数	攻击向量数	确认的二阶SQL漏洞数
Dormitory Manager	1	100	1
BookStore	3	200	2
JSPLibrary	4	200	2
Classnet	5	300	2

由表 4 中数据可知, 通过攻击实验验证, 一些二阶 SQL 注入疑似漏洞被证实为二阶 SQL 注入漏洞. 在实际中, 二阶 SQL 注入的检测难度较大, 完全检测出来或完全确认是很困难的. 本文的方法在相应的触发概率下, 能够有效地检测出二阶 SQL 注入漏洞.

5 结语

本文提出了一种基于 chopping 技术的静态分析 Web 应用程序源代码与设计攻击向量动态验证相结合的二阶 SQL 注入漏洞检测方法, 该算法将程序切片技术的变种形式程序 chopping 应用到静态分析过程, 通过获取污点传播路径并设计攻击向量运行的方式验证漏洞, 实现了二阶 SQL 注入漏洞的检测.

当应用程序规模较大时, 构造的系统依赖图规模庞大而且结构复杂, 不利于获取切片. 因此, 在今后的工作中可针对系统依赖图设计约减算法, 对本文方法进行优化.

参考文献

1 王青国. 浅析 Web 应用软件开发安全. 计算机系统应用, 2013, 22(2): 5–9.

2 OWASP Top10 object. https://www.owasp.org/index.php/Top_10_2013-Top_10, 2013.

3 Kim MY, Lee DH. Data-mining based SQL injection attack detection using internal query trees. Expert Systems with Applications, 2014, 41(11): 5416 –5430. [doi: 10.1016/j.eswa.2014.02.041]

4 Jang YS, Choi JY. Detecting SQL injection attacks using query result size. Computers & Security, 2014, 44: 104–118.

5 Lounis O, Guermeche SEB, Saoudi L, *et al*. A new algorithm for detecting SQL injection attack in Web application. Proceedings of 2014 Science and Information Conference (SAI). London, UK. 2014. 589–594.

6 Thomé J, Shar LK, Briand L. Security slicing for auditing XML, XPath, and SQL injection vulnerabilities. 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). Gaithersbury, MD, USA. 2015. 553–564.

7 Dahse J, Holz T. Static detection of second-order vulnerabilities in web applications. Proceedings of the 23rd USENIX Security Symposium. San Diego, CA, USA. 2014. 989–1003.

8 Yan L, Li XH, Feng RT, *et al*. Detection method of the second-order SQL injection in web applications. International Workshop on Structured Object-Oriented Formal Language and Method. Queenstown, New Zealand. 2013. 154–165.

9 Halfond WGJ, Orso A. AMNESIA: Analysis and monitoring for NEutralizing SQL-injection attacks. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York, NY, USA. 2005. 174–183.

10 Weiser MD. Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method[Doctoral Dissertation]. Michigan: University of Michigan, 1979.

11 CodeSurfer.http//www.grammatech.com/products/codesurfer.

12 Chen TY, Cheung YY. Dynamic program dicing. Proceedings of 1993 Conference on Software Maintenance. Montreal, Quebec, Canada. 1993. 378–385.

13 Krinke J. Barrier slicing and chopping. Proceedings 3rd IEEE International Workshop on Source Code Analysis and Manipulation. Amsterdam, Netherlands. 2003. 81–87.

14 Jackson D, Rollins EJ. A new model of progrm dependences for reverse engineering. ACM SIGSOFT Software Engineering Notes, 1994, 19(5): 2–10. [doi: 10.1145/195274]

15 Download JavaCC. http://javacc.org.

16 Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12(1): 26 –60. [doi: 10.1145/77606.77608]

17 Halfond WGJ, Viegas J, Orso A. A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering. Arlington, VA, USA. 2006, 1. 13–15.