文章编号:1000-5862(2016)04-0386-06

面向用户意图的 SQL 注入检测方法

毛辰宇,郭 帆*,叶继华
(江西师范大学计算机信息工程学院,江西 南昌 330022)

摘要: Web 程序安全的首要威胁是 SQL 注入攻击,动态分析技术可有效防御此类攻击. 提出面向用户意图的检测方法,在程序发布前预先定义 Web 程序期望的所有数据库操作,在运行时拦截提交至数据库的操作,阻止不符合意图的操作. 设计并实现描述数据库操作意图的语言 SQLIDL,将开发者提供的允许操作集合解释为以确定有限自动机(DFA)表示的字符串集合,并支持表名、列名、列值及存储过程名的正则表示. 在 SecuriBench 测试集的实验表明,该方法可有效检测现有 SQL 攻击模式且运行开销较小.

关键词: SQL 注入; 动态分析; 有限自动机; 攻击模式
中图分类号: TP 311 **文献标志码:** A **DOI:** 10.16357/j.cnki.issn1000-5862.2016.04.11

0 引言

随着互联网的迅猛发展,Web 应用越来越普及,随之带来的网络安全问题也越发严重,OWASP(Open Web application security project)在 2013 年十大关键 Web 应用安全漏洞报告中将 SQL 注入攻击(SQL Injection Attack,SQLIA)列在第 1 位[1]. SQLIA 指攻击者利用 Web 程序未对输入数据进行有效检查的漏洞,构造符合 SQL 语法但违背 Web 程序设计语义的恶意 SQL 语句,从而执行未经授权的数据库操作.

SQLIA 的防御技术主要分为静态和动态防御 2 大类,静态防御技术主要有 2 种:(i)利用字符串分析技术直接获取执行数据库操作时可能的 SQL 语句集合,若该集合与现有攻击模式不存在交集则可认为 Web 程序不存在 SQLIA[2];(ii)应用污点分析技术跟踪 Web 输入,若存在执行路径使得该输入未经检查并传播到数据库操作语句,则报警可能有 SQLIA[3-4]. 字符串分析存在精确度较低问题,而污点分析存在虚警过多问题,D. Ray 等[5]指出只有动态防御才可能精确阻止 SQLIA 等注入攻击.

动态防御技术包括动态污点分析[6-7]、对比分析由合法输入与攻击输入生成的不同 SQL 语句[5,8-9]、基于上下文无关文法(CFG)的分析等[10]. 动态污点分析指运行时跟踪 Web 输入(污点),若污点随着程序执行传播到 SQL 语句则实时报警. 对比分析主要以 2 次不同输入执行程序,比较生成的 SQL 语句的语法结构,如果结构存在差异表明该程序存在 SQLIA. 基于文法的分析利用字符串分析技术生成上下文无关文法(Context-Free Grammar,CFG)形式表示的 SQL 语句的值集,将输入标记为 CFG 的终结符,程序执行后,检查生成的 SQL 语句是否满足该 CFG,如果不满足则说明存在 SQLIA.

动态分析技术是从 SQL 语句语法分析的角度来判定是否存在 SQLIA,无法理解 Web 程序设计者和用户的数据库操作语义(本文称为用户意图). 本文提出一种面向用户意图的动态 SQLIA 检测模型,设计一种用户意图描述语言,将用户意图转换成有限自动机(DFA)表示的字符串值集合,在程序提交给数据库前动态截取 SQL 语句,检测该语句属于用户意图还是违背用户意图,违背用户意图即为 SQLIA.

1 相关工作

动态污点分析需要对程序进行插桩以跟踪污点输入的传播,文献[6]需要修改 PHP 解释器以跟踪输入的每个比特,而文献[7]对程序插桩跟踪所有的常量字符串和经过验证的输入,对程序插桩往往带来较大的运行开销,同时插桩代码的语义正确性无法保证,本文方法无需对代码进行插桩.

文献[9]基于现有 SQLIA 攻击模式与正常输入进行对比,将正常输入替换为攻击串,若产生的 SQL 语句的语法分析树结构不同,则存在对应的 SQLIA 漏洞. 文献[8]使用不同字符集分割 SQL 语句中的

收稿日期:2015-08-07
基金项目:国家自然科学基金(61562040)和江西师范大学科研计划(7177)资助项目.
通信作者:郭 帆(1977-),男,江西南昌人,副教授,博士,主要从事网络与信息安全的研究.

用户输入和程序自身产生的字符串,使用攻击模式生成的 SQL 语句的语法结构如果有所不同,且这些不同之处包含有"可执行代码"[5],根据所属不同字符集判断这些"可执行代码"由用户输入产生,则存在 SQLIA. 此类方法的一个主要问题在于没有明确什么样的输入才是合法输入,实际上是把所有不属于攻击模式的 SQL 语句都当作合法操作,而本文方法仅将符合用户意图即满足程序操作数据库语义的 SQL 语句作为合法输入.

文献[10]首先使用静态字符串分析技术获得以 CFG 形式表示的 SQL 语句值集,且将输入变量表示为该文法的终结符,该集合是实际可执行的 SQL 语句集合的超集,然后使用其它字符集标记和跟踪输入变量,当程序运行到提交数据库操作时,判断由输入变量产生的子串能否归约为终结符,从而判定实际执行的 SQL 语句是否属于该 CFG 所表达的语言,若不属于则为 SQLIA. 本文方法与文献[10]相似,但是本文方法使用 DFA 表示 SQL 语句值集,且由用户明确定义该集合,相比文献[10]基于字符串分析技术得到的 SQL 串集合精确性更高.

2　检测模型

面向用户意图的检测模型体系结构如图 1 所示,在 Web 程序向数据库提交请求之前动态截获请求的 SQL 串,判定该串是否满足预先定义的以自动机形式表示的用户意图,若不满足则报警.

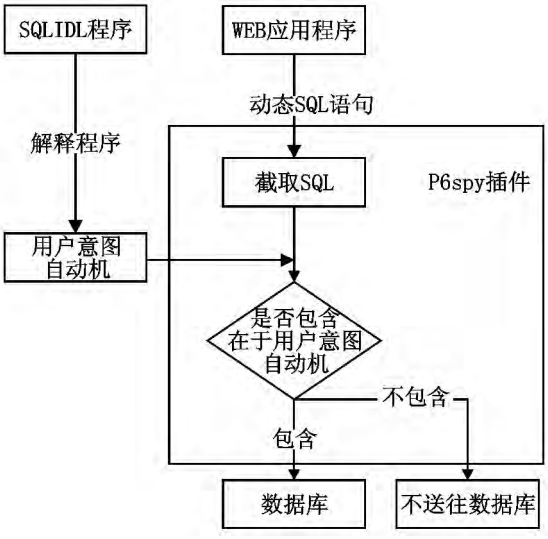

图 1　SQLIA 动态防御框架

本文设计一种描述用户数据库操作意图的语言(SQL Intention Description Language,SQLIDL),支持用户表达针对不同表和不同列的不同数据库动作. 使用 SQLIDL 语法编写期望执行的数据库操作语句序列,该序列即为用户期望的数据库操作意图. 模型将 SQLIDL 语句序列解释为由 DFA 表示的字符串集合,在运行时对提交的 SQL 串进行对比分析,检测该串是否满足 DFA 表示的用户意图,只有符合 DFA 的 SQL 串才会被提交给数据库. 目前,原型系统中截取 SQL 串并插入自动机对 SQL 进行检测的过程由修改后的 Ps6spy 插件实现[11].

2.1　SQLIDL 语法

SQLIDL 程序包括 3 类语句: 变量声明、变量赋值、操作语句. 程序的变量分为表变量、列变量、值变量和方法名变量,这些变量可以赋予常量字符串或者以正则表达式表示的串值,用于表示用户期望操作数据库中哪些表和列,可以根据表名、列名和列值来表达用户的操作意图,方法名变量表示可以调用哪些内置过程和存贮过程. 操作语句由关键字、变量和常量构成,SQLIDL 的关键字与 SQL 语言的关键字相同,用于表示用户支持的数据库操作以及查询条件等,变量必须是经过声明和赋值的变量,常量包括字符串和数字表示. 其抽象文法如图 2 所示.

```
Program ---> Variable Assignment*
             VariableDefinitions*
             Query*
Query ---> select|
           insert|
           update|
           Delete
Select ---> select (Distinct)? ColumnList
    from Table (Join)? (where Where)? (Order)? (Union select (Distinct)?
ColumnList from Table (Join)? (where Where)? (Order)?)?
Insert ---> insert into TableName ( [Column)?]) values (VarList)
Update ---> update TableName set UpdateAssignmentList where Where
Delete ---> delete from TableName where Where
Variable Assignment ---> (VarName (,VarName)*) : (TABLEVAR|
COLUMNVAR|VALUEVAR|ALIASESVAR|METHODVAR);
VariableDefinitions ---> VarName = Regex;
VarName ---> ([A-Za-z][0-9A-Za-z_]*)|('[A-Za-z][0-9A
-Za-z]*')
VarColumn ---> Varname(Column)
Distinct ---> distinct
JoinType ---> inner|left|right join
JoinTable ---> (Table JoinType JoinTable)|Table on JoinOn
Join ---> JoinType JoinTable
JoinOn ---> AliasesName. ColumnName Operators Regex (and|or Alias-
esName. ColumnName Operators Regex)*
Order ---> order by (AliasesName.)? ColumnName ASC|DESC
Where ---> (AliasesName.)? ColumnName Operators Regex (and|or (Alias-
esName.)? ColumnName) Operators Regex)*
Table ---> TableName|TableName as AliasesName|TableName AliasesName
UpdateAssignmentList ---> [ColumnName = VarName (,ColumnName =
VarName)*]
VarList ---> [VarName (,VarName)*]
ColumnList ---> (Column|VarColumn)*|*
Column ---> [ColumnName (,ColumnName)*] | [AliasesName.)?
ColumnName (,(AliasesName.)? ColumnName)*]
ColumnName ---> VarName
TableName ---> VarName
AliasesName ---> VarName
Operators ---> =|<>|>|<|<=|>=|like
Regex ---> 正则表达式
```

图 2　SQLIDL 的抽象文法

目前,SQLIDL 支持的数据库操作包括:(i) select 查询操作;(ii) insert 添加操作;(iii) update 更改操作;(iv) delete 删除操作;(v) order by 排序操作.

如用户希望仅对表 user1 和 user2 的列 id 进行查询,且仅访问 id 值超出某个数值 x 的行,x 由访问

网站的用户输入,使用如下 SQLIDL 程序表示为

```
t1: TABLEVAR;
v1: VALUEVAR;
t1 = user [12];
v1 = [0-9] +;
select 'id' from t1 where 'id' > v1;
```

声明表变量 $t1$,使用正则表达式"*user* [12]"表示用户期望访问的数据库表名,具体的列名 id 则用单引号常量串 'id' 表示,select 和 from 均为 SQL 语言的关键字. 变量 $v1$ 赋予正则表达式"[0-9] +"用于表达用户意图中的"大于某个值",$v1$ 的值限定了网站用户可以输入的查询条件必须是数字串.

2.2 SQLIDL 工作机制

本节以某实验室网站的数据库操作示例说明 SQLIDL 的工作机制,该网站设计者期望数据库表 News 的增加、删除、修改和查询操作:(i) 查询操作,根据每行新闻的类型(列 type) 及显示标记(列 is_show) 决定是否可以显示该新闻,并且按照主键 id 升序排列;(ii) 修改操作,用户根据新闻的显示标记(is_show) 可对指定新闻(id) 修改其标题(title) 和类型(type);(iii) 添加操作,向表中增加一行新闻;(iv) 删除操作,根据 id 列值删除指定新闻. 根据该意图访问数据库的 SQL 语句示例如图 3 所示,对应的 SQLIDL 程序如图 4 所示.

```
select* from News where type = 2 and is_show = 1 order by id ASC;
update News set title = 'aa',type = 2 where is_show = 1 or id = 2;
insert into News( id,title,type,is_show) values( 1,'title',1,1) ; delete from News where id = 1;
```

图 3 网站访问数据库 SQL 语句示例

```
v1,v2,v3: VALUENAME;
v1 = [1-9] * $
v2 = [A-Za-z0-9] + $
v3 = [1-9] * $
select* from 'News' where 'type' = v1 and 'is_show' = 1 order by 'id' ASC;
update 'News' set 'title' = v2,'type' = v1 where 'is_show' = 1 or 'id' = v3;
insert into 'News' values( v3,v2,v1,v1) ;
delete from 'News' where 'id' = v3;
```

图 4 代表网站设计意图的 SQLIDL 程序

图 3 给出的程序严格限定了用户可以提交的数据库操作的语法格式,任何违背该程序可表示的 SQL 语句串都将被拒绝并报警,即使该语句可能并不是 SQL 注入攻击,但是该语句违背了用户意图即报警,因此该方法不仅可检测 SQLIA,也可以检测网站数据库操作的程序实现与用户期望的不一致性.

SQLIDL 程序解释器将图 4 给出的程序解释为如图 5 所示的字符串 DFA,其中省略号表示忽略了与 SQLIA 无关的部分.

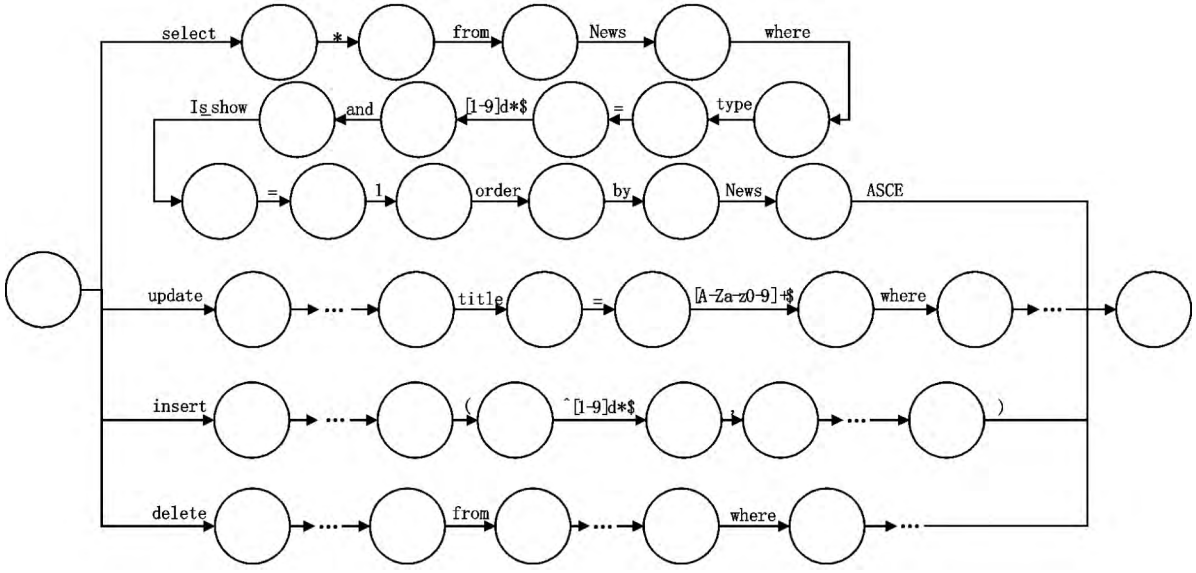

图 5 对应图 4 的字符串 DFA

图 5 生成的 DFA 可有效防御文献[5]总结的现有 11 种 SQLIA,表 1 给出本文方法与其它方法防御 SQLIA 的对比结果,SQLIDL 可以成功检测所有攻击模式.

文献[5]认为不属于可执行代码的注入不是攻击. 如模式 7 的攻击语句"SELECT balance FROM acct WHERE password = TRUE",TRUE 值的类型虽然与 password 列的类型不符,但是它不是可执行代码,因此不认为是一个注入攻击. N. Tuong 等[6]通过扫描每个令牌标识符和运算符检测 SQL 注入,使得语句结构正常的 SQL 注入不能被检测出. S. Son 等[8]使用影子字符集跟踪输入串,若该串不是执行代码则不报警,同样对模式 7 不会报警. 文献[10]不能分析出类似于 flag = TRUE 这样语法结构正常

但是存在注入隐患的攻击类型.

表 1　SQLIDL 与现有研究效果对比

	1	2	3	4	5	6	7	8	9	10	11
SQLIDL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
文献[5]	Y	Y	Y	N	Y	Y	N	Y	Y	Y	N
文献[6]	Y	N	Y	Y	N	N	N	N	N	Y	Y
文献[8]	Y	Y	Y	N	Y	Y	N	Y	Y	N	N
文献[9]	Y	Y	Y	N	N	N	Y	N	N	N	Y
文献[10]	Y	N	Y	N	N	N	N	N	N	N	Y

2.3　SQLIA 检测

本节对现有各种攻击模式分析并讨论本文模型的检测能力.

模式 1　攻击者修改原有 SQL 语句,显式的将列 type 的值填充为 “OR 1 = 1 - -”,使表达式的逻辑永远为真,攻击语句为 “select * from News where type = ‘OR 1 = 1 - -’ and is_show = 1 order by id ASC”,图 6 为该语句的 DFA 表示,省略了注入无关部分,图 5 的 DFA 表示无法表示串 “or 1 = 1”,因此可以检测该攻击.

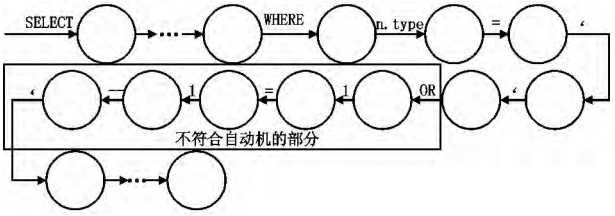

图 6　模式 1 的自动机表示

模式 2　攻击者注入特殊的函数,实现 SQL 注入攻击,如 “select * from News where type = exit( ) and is_show = 1 order by id ASC”,图 6 为该串的 DFA 表示,由于列 type 的取值范围为数字串,由值变量 $v2$ 的正则表达式表示,函数调用语句”exit( )”无法使用图 5 的 DFA 表示,如图 7 所示.

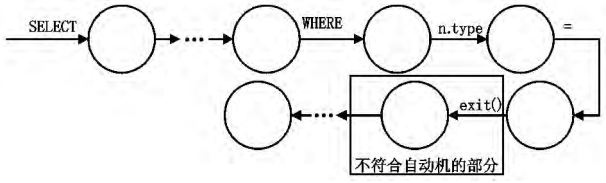

图 7　模式 2 的自动机表示

模式 3　使用 “flag = 1 000 > GLOBAL”注入模式可以使攻击者通过二分搜索有效的提取全局变量. 如”select * from News where type = 100 > GLOBAL and is_show = 1 order by id ASC”,与模式 2 类似,列 type 的注入语句” 100 > GLOBAL”无法由值变量 $v2$ 的正则表达式表示,如图 8 所示.

模式 4　使用自定义存贮过程注入. 如 “select * from News where n. type = ‘ and test. F1 = 1 - -’ and is_show = ´1´ order by id ASC”,一旦数据库执行

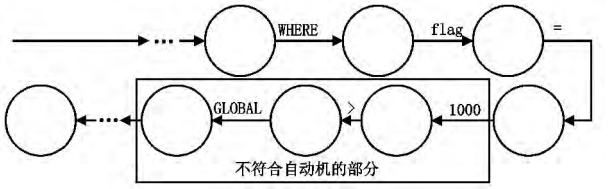

图 8　模式 3 的自动机表示

该语句,攻击者将触发存贮过程 test. F1 的执行,该攻击与模式 1 相似处理.

模式 5 和模式 6　攻击者调用不符合用户意图的内置过程或存贮过程,该模式与模式 2 不同之处在于,攻击者只能注入需要调用的过程名. 如 “select * from News where n. type = exit( ) and is_show = 1 order by id ASC”,图 4 的示例不支持此种模式,但 SQLIDL 可以通过方法名变量来限制用户可以调用的内置或者存贮过程名字,从而检测模式 5 和模式 6.

模式 7　使用不同类型的列值注入. 如 “select * from News where type = TRUE and is_show = 1 order by id ASC”,列值”TRUE”无法使用列 type 的数字串表示,因此可检测该模式.

模式 8 和模式 9　由于代码防御技术会通过查找常见字符或字符串判断 SQL 注入,攻击者会对构造的恶意字符串编码. 由于互联网应用中不同层次采用的编码方式不同,使用编码的参数很容易绕过特征字符串检测,如 “select * from News where type = ‘2’; exec(char(0x73687574646f776e)) - -’ and is_show = 1 order by id ASC”,图 9 为该串对应的自动机表示,圈出部分为注入内容,由于注入串无法使用图 5 的 DFA 表示,因此 SQLIDL 可轻松检测该模式,如图 9 所示.

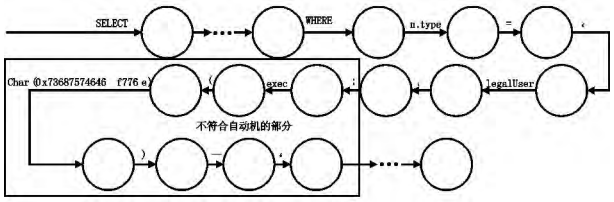

图 9　模式 8 和模式 9 的自动机表示

模式 10 和模式 11　在不修改原有 SQL 语句逻辑的情况下,用’; ’等语句结束符结束原 SQL 命令,并在后面构建新的 SQL 命令,让数据库一次收到多个 SQL 语句请求. 其中,第 1 个语句是正常请求,后面的 1 个或多个语句是攻击者构造的恶意请求. 例如攻击者可在参数处构造含有 SQL 命令结束符的值 “‘; drop table News - -”,则生成多语句 SQL 命令 select * from News where type = ‘1’; drop table News - ‘ and is_show = ´1´ order by id ASC.

图 10 为该攻击串的 DFA 表示,圈出部分为注

入内容,后续注入语句 "drop table News"无法被图 4 的 DFA 表示,即可检测该攻击模式.

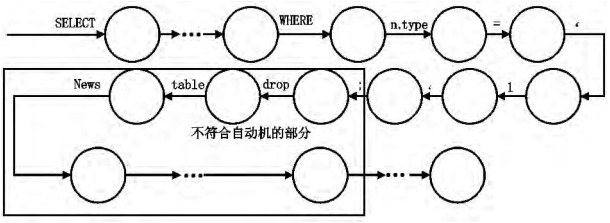

图 10 模式 10 和模式 11 的自动机表示

3 系统实现

Ps6spy [11] 是记录 JDBC 调用日志信息的工具,相当于一个 SQL 语句记录器,本文的检测模型通过修改 Ps6spy 实现. Ps6spy 工作在代理模式,对 JDBC 驱动拦截并转发,本文系统只需在数据库连接配置文件中将数据库驱动程序改成 Ps6spy 的拦截驱动程序即可完成部署.

原型系统分为 SQLIDL 解释器和自动机检测模块. SQLIDL 解释器由 SQLIDL 语法分析程序和自动机生成程序组成,语法分析程序使用 Java 语言实现,自动机生成程序基于 JSA [13] 中的组件开发.

SQL 检测部分通过修改 Ps6spy 插件实现,Ps6spy 主要在 com. Ps6spy. engine. logging 包的 P6LogStatement. java 和 P6LogPreparedStatement. java2 个类中执行数据库操作,本文实现主要对这 2 个类进行修改,将自动机检测模块部署在数据库操作执行之前. 在实时生成的 SQL 送往数据库执行之前,检测模块将该 SQL 与 SQLIDL 生成的自动机比对,根据比对结果判断实时 SQL 是符合用户意图.

系统实时检测函数流程处理为:(i) Ps6spy 插件拦截 Web 程序提交的 SQL 语句;(ii) 生成与之相对应的字符串 DFA;(iii) 将动态 SQL 语句和自动机比对. 如果不包含,则认为执行的 SQL 不符合程序意图,将其拦截;否则认为是一次正常的 SQL 语句请求,送往数据库执行.

4 实验分析

为了验证方法的有效性和可行性,本文针对 SecuriBench [12] 测试网站集合以及 BodgeIt 漏洞演示网站 [14] 展开 SQLIA 攻击测试,以检测本文方法的防御效果并展开性能分析. Web 服务器为 Tomcat6. 0,数据库采用 Mysql5. 6,JAVA 环境采用 JDK1. 6 以上. 实验在局域网内进行,客户端发起正常或恶意的 Web 请求,服务器端硬件为 CPU 2GHz,内存 4GB,系统为 Windows 7.

表 2 给出 SecuriBench [12] 测试集中的信息,并对比了应用本文方法前后对数据库操作性能的影响.

表 2 测试网站信息汇总及性能测试数据

Program	Version number	File Count	Line Count	SQL Injections	Uninstrumented Averagetime/ms	Instrumented Averagetime/ms
webgoat	5.4	77	19 440	8	3	119
personalblog	1.2.6	39	5 591	2	40	151
snipsnap	1.0-BETA-1	445	36 745	1	73	185
roller	0.9.9	276	52 089	0	8	122
jboard	0.30	90	17 542	0	35	152
bodgeit	1.4.0_51	39	1 835	13	2	97
Total		927	131 407	24		

表 2 的列值包括测试网站的版本号、文件数量、代码行数、存在的 SQL Injection 数量以及应用方法前后响应数据库请求的时间.

针对每一个测试网站的不同数据库操作点,分别执行 200 条正常和恶意数据库操作,作为统计数据库请求时间的样本集合. 实验表明本文方法对每个 SQL 请求的平均响应时间延迟 100 ms ~ 110 ms,主要的时间消耗位于匹配自动机的过程中.

为测试本文方法的精确性,实验针对 webgoat 中 SQLIA 模块编写对应的 SQLIDL 程序,并对每个模块执行 SQLIA 攻击. 实验结果如表 3 所示,其中 SQL Injection 列表示 webgoat 中存在的 SQLIA 漏洞,实验表明本文方法可以准确检测并拦截的所有 SQL Injection 攻击. 针对 SecuriBench 中的 personalblog 和 snipsnap 网站以及 bodgeit 网站的 SQLIA 漏洞进行实验检测,本文模型同样成功检测出所有攻击.

表 3 webgoat 中各种 SQLIA 类型的检测结果

SQL Injection	result	SQL Injection	result
NumericSQL Injection	√	Add Data with SQL Injection	√
String SQL Injection	√	Database Backdoors	√
LAB: SQL Injection	√	Blind Numeric SQL Injection	√
Modify Data with SQL Injection	√	Blind String SQL Injection	√

5　结语

本文提出了一种面向用户意图的动态 SQL 注入缺陷检测模型,设计一种数据库操作意图描述语言,将程序意图转换成有限自动机表示的字符串值集合,动态截取 SQL 语句字符串,检测该字符串值是否包含于自动机. 该方法不足之处在于无法区分网站不同用户的意图,如管理员和普通用户的访问意图可能不同,如果以管理员意图来设计自动机,就无法检测普通用户冒充管理员的情况.

该方法虽然尽可能的通过描述用户意图而实现对危险攻击的拦截,但是在实际程序编写中,代码长度和 Web 应用中数据库操作的数量成正比. 当数据库操作较多时,精确描述程序意图非常困难,因此未来工作之一是研究从实际的数据库操作语句中自动提取用户意图.

6　参考文献

[1] OWASP Top10-Open Web Application Security Project. Top ten Web application security risks [EB/OL]. [2015-03-14]. http://www.owasp.org.cn/owasp-project/2013top10.

[2] Wasserman G, Su Zhendong. Sound and precise analysis of Web applications for injection vulnerabilities [EB/OL]. [2015-03-14]. http://web.cs.ucdavis.edu/~su/publications/pldi07.pdf.

[3] Jovanovic N, Kruegel C, Kirda E. Static analysis for detecting taint-style vulnerabilities in web applications [J]. Journal of Computer Security, 2010, 18(5): 861-907.

[4] 黄强,曾庆凯. 基于信息流策略的污点传播分析及动态验证 [J]. 软件学报, 2011, 22(9): 2036-2048.

[5] Ray D, Ligatti J. Defining code-injection attacks [J]. Acm Sigplan Notices, 2015, 47(1): 179-190.

[6] Tuong N, Guarnieri A, Greene S, et al. Automatically hardening web applications using precise tainting [J]. Ifip Advances in Information & Communication Technology, 2010, 181: 372-382.

[7] 王溢,李舟军,郭涛. 防御代码注入式攻击的字面值污染方 [J]. 计算机研究与发展, 2012, 49(11): 2414-2423.

[8] Son S, McKinley K S, Shmatikov V. Diglossia: detecting code injection attacks with precision and efficiency [EB/OL]. [2015-02-17]. http://msr-waypoint.com/pubs/202166/diglossia-ccs-2013.pdf.

[9] Bandhakavi S, Bisht P, Madhusudan P. CANDID: preventing SQL injection attacks using dynamic candidate evaluations [EB/OL]. [2015-02-17]. https://www.cs.uic.edu/~pbisht/files/candid-sql-injection-ccs07.pdf.

[10] Su Zhendong, Wassermann G. The essence of command injection attacks in Web applications [J]. Acm Sigplan Notices, 2006, 41(1): 372-382

[11] GitHub Inc. P6spy [EB/OL]. [2015-02-17]. https://github.com/p6spy/p6spy.

[12] Benjam in Livshits. Stanford securi bench [EB/OL]. [2015-02-17]. http://suif.stanford.edu/~livshits/securibench/.

[13] Aarhus University. Java string analysis [EB/OL]. [2015-02-17]. http://www.brics.dk/JSA/

[14] GitHub Inc. BodgeIt [EB/OL]. [2015 - 07 - 11]. http://code.google.com/p/bodgeit.

The Intention-Oriented SQL Injection Defense

MAO Chenyu, GUO Fan*, YE Jihua

(College of Computer Information and Engineering, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: SQL injection attack(SQLIA) is the most serious threat to Web program security, while dynamic analysis may effectively defend SQLIA. An intention-oriented detection approach is proposed to represent all the database operations expected by Web users, to intecept the operations before the user submission and drop the unintentional operations. A language named SQLIDL is proposed to express the intention of database operations, to transform the SQL operations into string sets formalized by deterministic finite automata(DFA). SQLIDL currently implements the regular expression representation of table names, column names, values and store procedure names. The prototype implementation is evaluated on SecuriBench datasets and the results demonstrate all existing SQL attack patterns can be correctly detected with acceptable run-time overhead.

Key words: SQL injection; dynamic analysis; DFA; attack pattern

(责任编辑: 冉小晓)