一种数据流相关过滤器自动插入的注入入侵避免方案

尹中旭 张连成

(数学工程与先进计算国家重点实验室 郑州 450002)

摘 要 注入类漏洞是动态Web应用程序中广泛存在的漏洞。文中对注入漏洞产生和利用的必要条件进行分析,并利用相关方法针对注入变量的不同类型(数字型、字符型和搜索型)进行区分防范;对宿主语言和对象语言进行分析,定位出了SQL语句中的查询变量及其类型;在控制流图的基础上,构建了包含source点和sink点的数据依赖关系子图;针对该子图,设计了过滤器插入算法,定义了不同输入数据类型和查询类型的过滤策略;随后,实现了基于数据流分析以及在相关数据库操作之前自动插入过滤器的方案;最后对提出的方案进行了分析测试,结果验证了所提方案的有效性。

关键词 SQL注入,程序分析,数据流分析,入侵避免,平衡字

中图法分类号 TP393.08 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.01.031

SQL Injection Intrusion Avoidance Scheme Based on Automatic Insertion of Dataflow-relevant Filters

YIN Zhong-xu ZHANG Lian-cheng

(State Key Laboratory of Mathematical Engineering & Advanced Computing, Zhengzhou 450002, China)

Abstract SQL injection is a widespread vulnerability in dynamic Web applications. This paper analyzed the necessary conditions for the production and exploitation of injection vulnerabilities, and made a distinctive protection for different types (digital type, character type and search type) of injection variables. Then, this paper dissected both the host language and object language to locate the query variables and their types in the SQL statement, and constructed the data dependency subgraph including source point and sink point on the basis of control flow graph. Aiming at this subgraph, this paper designed a filter insertion algorithm and defined filter policies according to different input and query types. Meanwhile, this paper implemented a dataflow analysis based scheme which automatically inserts filters before relevant database operation. At last, this paper analyzed and tested the proposed scheme. The results suggest the effectiveness of the proposed scheme.

Keywords SQL injection, Program analysis, Dataflow analysis, Intrusion avoidance, Banlance word

1 引言

Web程序具有灵活方便的用户界面,基于Web程序所开发的信息共享、交互反馈以及服务查询等动态页面应用已成为目前互联网服务的常见形式和通信量的重要来源。云平台以及网络设备的管理界面都采用了Web接口。SQL注入通过在用户输入中安排好附加的数据库查询操作语句,在缺乏可靠的过滤手段的情况下,由Web应用程序转交数据库服务器执行,达到入侵数据库系统的目的。

通常,SQL注入漏洞可以通过对用户数据过滤等安全编程方法避免。由于Web应用建立在较为抽象的脚本语言基础上,在Web2.0和前端技术日趋复杂的前提下,开发者更多地关注于界面效果和功能实现,变量的传递流程更加复杂。对于Web平台应用体系结构基础上的漏洞,如果缺乏有效的过滤和限制,注入漏洞必然存在,因此在Web应用程序被广泛应用的条件下,该类漏洞成为了当前网络安全的一大隐患。

SQL注入漏洞广泛存在于Web应用中,其产生于Web应用和数据库相互连接时。在OWASP发布的十大安全漏洞威胁报告(OWASP Top 10)中,SQL注入仍然排名第一[1]。

360发布的《2016年中国互联网安全报告》指出,SQL注入漏洞占到了所有检出的网站漏洞的16%[2]。由此可见,SQL注入漏洞一直是网络应用中较为严重的安全威胁。

对于SQL注入,通常使用的安全编程方法是过滤掉单引号和数据库查询关键字,对数位进行约束等,但往往只采用一种方法且意图不明确,对注入的防范不够彻底。例如,过滤单引号或关键字的方法对注入利用变量为数字型的方式不起作用。在注入语句中,特殊字符或关键字的部分可以通过改变注入数据的大小写或使用二进制编码方法进行替代,以绕过过滤。

Boyd等[3]提出了一种基于数据库连接代理的方案sql-

到稿日期:2018-01-17 返修日期:2018-03-12 本文受国家自然科学基金资助项目(61402526)资助。

尹中旭(1983-),男,博士生,讲师,CCF会员,主要研究方向为网络安全,E-mail:yinzhxu@163.com;**张连成**(1982-),男,博士,讲师,主要研究方向为网络安全,E-mail:liancheng17@gmail.com(通信作者)。

rand。在该方案中,应用程序通过代理间接连接数据库,应用程序的数据库查询关键字自定义为随机化的内容,任何注入到查询数据中的 SQL 语句将在代理中解码失败。该方案虽然解决了应用程序中的注入检测问题,但入侵者可以通过发掘自定义的关键字的方式绕过该机制。

Gould 等[4]提出了一种 SQL 注入误用检测模型,用动态与静态相结合的方法,对应用程序进行静态分析,提取每个查询语句的关键字序列,然后在执行查询前提取并检查该查询语句序列的特征是否是几种合法的类型之一。该方案的优点是可以实时检测出插入数据中的额外查询,但其会产生误报,且不能抵御二阶 SQL 注入攻击[5]。

Valeur 等[6]提出了一种 SQL 注入异常检测模型,它通过对应用程序所有的数据库查询语句进行提取,并只保留这些语句的查询关键字部分,来构造全局模式集合;同时针对这些查询模式进行训练,构建行为数据库。在运行时,用代理或截获系统调用的方式检查所有的数据库查询语句,对其轮廓部分用全局模式集检查,对数据部分用训练出的行为数据库进行检查。其缺点是有一定的漏报率和误报率,且不能防止行为模仿攻击[7]。

Web 应用防火墙(Web Application Firewall,WAF)深入到 Web 协议内部,对 Web 交互数据进行规则设定和过滤的防火墙技术。为了防护网站,一般将 WAF 部署在 Web 服务器集群前端,并采用反向代理的模式对经过 WAF 的 HTTP 请求包进行双向过滤[8]。

WAF 通过定义攻击特征码的方式对常见的 SQL 注入攻击进行过滤和防护。但是关系型数据的种类非常多,虽然它们有统一的 SQL 结构化数据查询语言,但是每个数据库的具体实现存在差异,这些差异使得攻击数据具有多样化的特征,因此也使得 WAF 系统在不熟悉数据库类型、命令、结构以及应用程序的上下文的情况下,仅仅通过分析网络数据包和定义一些数据库特殊字符黑名单,不足以防护多种多样的 SQL 注入攻击。一些改进方法通过机器学习进行签名规则学习,但存在数据集获取难度大以及人工标记工作量大等问题[9]。RASP(Runtime Application Self-Protection)[10]技术是对 WAF 的改进,其将保护代码注入到应用程序中,与应用程序融为一体,进行实时监测和阻断攻击,使程序自身拥有自保护的能力;并且应用程序无需在编码时进行任何修改,只需进行简单的配置即可。这种方法属于针对特定漏洞进行动态补丁防护的应急方案,还不能独立和全面地应对某一类安全威胁。

一些方法通过参数化标准查询来限制 SQL 注入语句的查询,但这些方法与具体的语言特性有较强的联系,通用性受限[11]。

从实际情况来看,一方面,Web 应用程序的安全性很大程度上依赖于开发人员良好的编程习惯,但想要避免该漏洞,就需要在已经存在注入点的基础上从不同的层面进行限制,对攻击手段的理解不同,开发人员就会采用不同的方法,如果采用的方法定位不准确,那么只会增加漏洞利用的难度,难以达到预防效果。另一方面,对现有应用程序从源码上进行改造的开销较大,同时在安全编程上缺乏一种统一且可靠的解决方案。本文结合对应用程序的自动分析,通过在 SQL 连接中插入过滤代码来实现应用层入侵检测和避免方案,将安全模块附加到现有体系结构上,以减轻应用开发人员的安全设计成本,同时解决现有应用程序的不安全问题。

2 基于程序分析的 SQL 注入防御

本文实现的方案是将 SQL 语句和宿主程序进行结合分析,用数据流分析的方法找出所有的查询输入变量,并在变量的初始化和引用之间插入必要的 SQL 检查和过滤语句。目前 Web 应用程序所用的服务端脚本语言有很多种,如常用的 ASP,JSP 和 PHP 等,本文以 JSP 为目标,通过对其引用的 Java 包进行程序分析,对上述方法做验证性实现。图 1 给出了字符型 SQL 注入漏洞的代码片段。

```
1. String type=request.getParameter("type");
2. String auth=request.getParameter("auth");
3. if(type==1)
4. {response.sendRedirect("QRlogin.jsp");}
5. else
6. {String[] StrArray=auth.split(",");
7.  String u=StrArray[0];
8.  String p=StrArray[1];
9.  Statement sm=ct.createStatement();
10. ResultSet rs=sm.executeQuery("select * from usertable where uid="+
    u+" and password="p+"");
11. if(rs.next())
12. {response.sendRedirect("wel.jsp");}
13. else
14. {response.sendRedirect("passlogin.jsp");}
15. }
```

图 1 字符型 SQL 注入漏洞的代码片段

Fig. 1 Code fragment of character-typed SQL injection vulnerability

图 1 所示程序存在通过查询变量 u 引入数字型注入漏洞,以及通过 p 引入字符型注入漏洞。

2.1 污点策略定义和数据依赖分析

2.1.1 基于对象类定位 source 点和 sink 点

首先通过对 Web 的 GET,POST,COOKIE 等不同的输入数据来源进行定位,获得 JSP 上相关的污点源信息,如图 2 所示。同时,页面间的跳转关系决定了污点的传播逻辑,其中,ActionServlet 通过配置文件交互决定由哪个 Action 类处理对应的页面跳转。对由此引入的相关变量进行标记,并将之作为分析的 source 点。

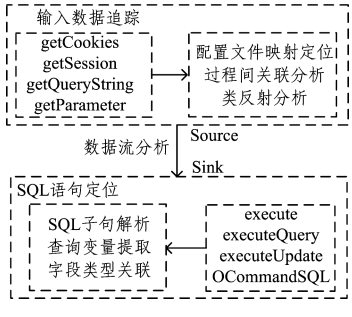


图 2 source 点与 sink 点的定位

Fig. 2 Location of source point and sink point

sink 点主要是通过 SQL 语句中的查询变量进行定位。

数据库查询语句一般是由应用程序动态生成。应用程序所用的脚本语言被称为宿主语言，由脚本语言动态生成的数据库查询语言（如 SQL 语句）是对象语言。对于待检测变量的定位和类型判定，需要先综合分析宿主语言和对象语言。

将要分析的关键程序点设置为执行查询操作的特征语句，即将 JSP 中的 execute，executequery 等查询语句作为特征语句，对 SQL 语句进行定位，进而对数据库查询字符串进行分析以定位查询变量。

查询类型一般分为数字型、字符型和搜索型 3 类。查询变量无法通过宿主语言进行定位，需要通过分析目标的 SQL 语言对查询变量进行定位。而对于查询变量的类型，需要定位查询变量所对应的数据库的列，通过列的类型来确定其具体类型。

2.1.2　依赖关系分析

针对查询变量进行数据流分析。对源码进行语法制导分析，在控制流分析中构造出程序各个过程内的控制流图和过程间的调用关系，在数据流分析中以控制流为基础分析出变量间及变量和输入参数间的数据定值引用和依赖关系，在过程内数据依赖关系的基础上构造全局依赖关系。图 3 为图 1 中代码片段所对应的数据依赖关系，其中节点数字表示行号，边上的标记(C,D)表示数据依赖或控制依赖。在词法分析及其后续阶段保留节点在源程序中的行列信息。

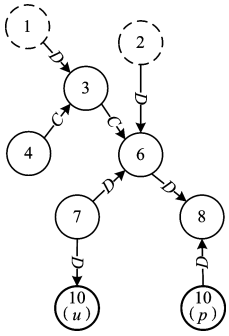


图 3　代码片段中的数据依赖关系

Fig. 3　Data dependencies of code fragment

静态污点分析是对源代码直接进行分析的一种代码分析方法，主要对源代码的控制流和数据结构进行分析，通过词法分析、语法分析、关键字符研究和参数过滤研究等，对存在缺陷的代码进行污点标记和污点追踪，最后根据污点传播的路径来检测代码的安全性和完整性。

2.2　基于静态数据流分析的过滤规则生成

过滤规则与 source 点和 sink 点都相关。对于 sink 点，根据对 SQL 语句的查询变量，结合数据库进行关联分析，以确定变量的类型及由此决定的可能的注入类型。

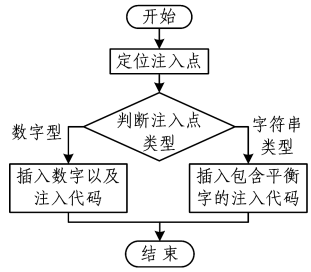
对于字符型和搜索型，由于在程序中构造查询时对这两类关键字都使用了引用符号来定界，如单引号“‘”“’”和双字符序列“’%”“%’”，这里从注入的角度把这两类符号和注释符号“－－”统称为平衡字。对这两种注入类型进行利用的必要条件是在字符串里输入平衡字，以便插入额外查询。因此，通过过滤输入查询关键字字符串变量中的平衡字，可以过滤这种类型的注入。

图 1 第 10 行中，变量 u 和 p 的值被单引号封闭，因此要实现注入攻击，需要输入平衡字将单引号封闭之后再引入其他的 SQL 语句，如“1‘or’ 1 ‘＝’1”。

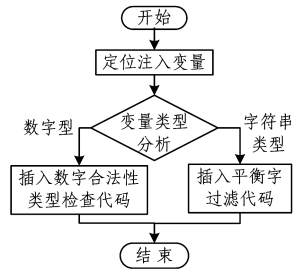
对于数字型注入，不需要输入平衡字，其方法是对数字字符的取值进行约束，需要检查输入的各位是否为数字。可以采用 Integer. parseInt()语句对字符串进行转换，如果能够转换为整数，则说明该字符串的内容为合法的数字类型。

根据不同的变量类型构造相应的语句，对数字型做数位检查和字符串到数字的转换，对一般字符型和搜索型进行平衡字判断和过滤。根据得到的源码位置信息，将语句插入到对输入变量进行初始化定义的语句之后。

根据要进行的查询对输入关键字依照类型进行区分，并分别进行检查，可以防范以上几类 SQL 注入问题，如图 4 所示。



(a)基于注入变量类型的注入攻击步骤



(b)基于注入变量类型的注入防护方案

图 4　基于注入变量类型的 SQL 注入攻击和防护

Fig. 4　SQL injection attack and protection based on injection variable type

2.3　过滤器设置的算法设计

首先对目标程序从 source 点到 sink 点进行数据流分析，对与污点数据节点无关的节点和边进行剪枝，对整个程序进行切片分析。下面给出过滤器插入算法的具体步骤。

算法 1　过滤器插入算法 P

输入：数据流子图 $G=\{V,E,\lambda,\mu\}$，source 点集合 V_{source}，sink 点集合 $V_{sink}(V_{source}\subset V, V_{sink}\subset V)$，$\lambda: E\rightarrow\Sigma$，其中 $\Sigma=\{C,D\}$

1. for all $v\in V_{Sink}$
2. $v'=v$
3. do
4. 取边$(v_1,v')\in E$ 且 $\lambda(v_1,v')=D$
5. if 存在 $v_2,(v_1,v_2)\in E$ 且 $\lambda(v_1,v')=D$ 且 $v_2\neq v'$或 $v_1\in V_{Source}$
6. 选择在边(v_1,v')上插入过滤器
7. else
8. $v'=v_1$
9. while $v'\notin v_{Source}$

图 3 即进行切片分析之后的数据流子图。过滤器插入的算法要考虑到多个 sink 点的情况，若某程序位置的数据流可以关联到多个 sink 点，则不能在该点进行插入。

在该算法中，从每个 sink 点开始，通过数据依赖关系边

的连接关系进行回溯，向 source 点遍历，进行数据流回溯分析。过滤器插入的目标边应该是到达某 sink 点的路径，而且只针对一个 sink 点。为了不影响程序执行的效率，插入的边应该为满足该条件的最接近于 source 点的边。如在图 3 中，节点 10(u)和节点 10(p)为 sink 点，节点 1 和节点 2 为 source 点。以 sink 点 10(u)为例，首先往前回溯到节点 7，然后再回溯到节点 6 后，若发现存在从节点 6 到其他 sink 点的数据依赖边(6,8)，则不能再向前回溯，在边(6,7)上插入过滤器。对于过滤器的选择，首先根据数据流分析的结果，定位节点 7 所对应的 source 节点 2，再根据 sink 点 10(p)的类型为数字型，查询表 1 对应的过滤规则，得到对应的过滤器算法。

表 1 不同变量类型的过滤规则

Table 1 Filtering rules of different variable types

	数字型	字符型	搜索型
get	url 编码检查＋数字位解析	url 编码解析＋引号字符过滤	url 编码解析＋‘%’字符过滤
post	CharacterEncoding 编码检查＋数字位解析	CharacterEncoding 编码检查＋引号字符过滤	CharacterEncoding 编码检查＋‘%’字符过滤
Session/application	会话变量解码＋数字位解析	会话变量解码＋引号字符过滤	会话变量解码＋‘%’字符过滤
cookie	cookie 变量解码＋数字位解析	cookie 变量解码＋引号字符过滤	cookie 变量解码＋‘%’字符过滤

3 系统实现和验证分析

3.1 查询变量的定位和类型分析

通过 String 工具[9]可以分析出 Java 程序中的特定字符串变量在某特定位置下的所有可能的赋值查询变量，以定位潜在 SQL 注入漏洞的 sink 点。基于 String 实现对变量的自动化分析。定义 SQL 查询关键词为标签，利用StringAnalysis 类进行分析，定位 SQL 语句。

```
private static String[] sigs={
"〈boolean execute(java.lang.String)〉",
"〈java.sql.ResultSet executeQuery(java.lang.String)〉",
"〈int executeUpdate(java.lang.String)〉"};
```

对于一般的注入攻击，分析的重点是 SQL 查询语句中的 WHERE 子句，而由于要防止二阶注入攻击[5]，因此需要对数据库插入操作语句的 INSERT 子句和更新语句的 SET 子句进行分析。

图 5 中的 SQL 语句实例概括了一次查询的语句的多数关键词和句法类型[13]。

```
SELECT C.Name,E.NameLast,E.NameFirst,E.Number,ISNULL(I.Description,'NA')
AS Description
    FROM tblCompany AS C    JOIN tblEmployee AS E    ON C.CompanyID=E.CompanyID    LEFT JOIN tblCoverage AS V    ON E.EmployeeID=V.EmployeeID    LEFT JOIN tblInsurance AS I    ON V.InsuranceID=I.InsuranceID    WHERE C.Name LIKE @Name
AND V.CreateDate > CONVERT(smalldatetime,'01/01/2000')
    ORDER BY C.Name,E.NameLast,E.NameFirst, ISNULL(I.Description,'NA')
```

图 5 SQL 查询示例

Fig.5 Example of SQL query

通过 AND 或 OR 拆分 WHERE 后面的判断语句，将判断符号(LIKE,<,>,=等)前后的列名和变量名或常量值对应起来，然后将以表中列的类型决定变量类型。

更新语句的一般形式为：

UPDATE 〈table_name〉 SET 〈column_name〉=〈value〉 WHERE 〈search condition〉

将 set 子句后面的每一个“=”左右的列名和变量名或常量值对应起来，用于下一步分析。

插入语句的一般形式为：

INSERT [INFO] 〈table_name〉 [(column_list)] VALUES (value_list)

分析 column_list 和 value_list，将所插入行的每个单元的列名和变量名或常量值对应起来。

这样，对数据库的查询、插入和更改操作的列名和变量或常量就建立了对应关系，然后根据列名，通过查询数据库获取列的类型名，如利用查询结果得到变量的类型名在 mssqlserver 中的实现为：

```
select b.name from syscolumns as a JOIN systypes as b on a.xtype=b.xtype where a.name='列名'
```

3.2 数据流分析

自动化工具的程序分析，以 Soot 和 String 两个开源软件为基础来实现[12]。其中 Soot 是一个对 Java 字节码进行优化的工具，由于它根据 Java 字节码对目标程序结构进行了分析，因此可以得到目标程序的 jimple 中间语言描述形式[13]。

利用 Soot 对目标程序构建控制流图 UnitGraph，在此基础上继承 Soot 工程中的 BackwardFlowAnalysis 类，在 flowThrough 函数中定义数据流传播策略。从 sink 点开始，对数据依赖关系进行分析。在 Soot 分析结果的基础上，对 jimple 语句进行处理，即对目标字符串变量进行活动性分析(LivenessAnalysis)、别名分析和数据依赖关系分析，构建仅与 source 点和 sink 点相关的数据依赖子图[14]。

根据 2.3 节的过滤器插入算法，插入相应的过滤代码，以根据输入数据类型和查询变量类型分别进行过滤处理。例如，对图 1 中变量 u 进行分析并将其转换为数字型的语句为：

```
int intu=Integer.parselnt(u);
```

如果 str_count 包含非数字内容，则会出错。如果为字符型，则进行平衡字过滤处理，根据 sql 语法，将“--”“’”“’%”和“%’”等作为平衡字进行过滤。该方案比较完整地防范了常见的 SQL 注入攻击，同时通过处理数据库插入和更新语句，对二阶注入攻击也有良好的防范效果。

4 测试

将本文所实现的方法 TPSQLIA 与 WAF 和关键词过滤等方法进行对比。其中，WAF 选用开源的 JSP 网站防火墙 webcastellum[15]。实验通过尝试直接注入、关键字编码、二阶注入、等价函数替换等攻击方法测试防护效果，结果如表 2 所列。其中，二阶注入使用多重编码、替换等方式将载荷存储于数据库，而触发时通过正常请求数据库中的数据进行，因此一般的 WAF 方法无法检测出异常行为。对于特定的实现，关

键词随机化方案替换的关键词是确定的，虽然其能够限制二阶注入，但攻击者可以根据其替代方案重构攻击语句进行绕过。代码过滤的方法可以通过关键词多重编码的方法进行绕过。本文提出的 TPSQLIA 方法，通过区分具体的注入类型，对常见的数字型查询直接检查变量的数位，对字符型和搜索型的查询着重对插入额外语句必须的少数平衡字进行过滤，能有效防范关键字编码、等价函数替换等攻击方法，同时本方案通过程序分析方法直接在查询语句之前插入针对具体变量的过滤代码，可以有效对关键字编码和二阶注入方法进行防范。

表 2　SQL 注入防护的对比测试结果

Table 2　Comparison test results of SQL injection protection

测试方法	直接利用	关键字编码	二阶注入	等价函数替换
WAF	√	×	×	√
关键词随机化	√	√	√	×
代码过滤	√	×	×	√
TPSQLIA	√	√	√	√

结束语　本文提出了一种结合代码静态分析技术在现有 Web 程序中自动添加针对注入的入侵防范语句的方案。首先分析 Web 程序，确定查询、更新、插入语句的位置；然后通过数据流分析确定相应查询或更新的操作字符串，分析该字符串以确定该字符串的操作性质以及操作对象，根据已经建立的防范模型确定防范语句，并辅助插入到程序中。本文以 JSP 网站为例进行了验证，所提方法通过针对性的防护，降低了过滤方法的复杂程度，可有效地防御常见的注入改进攻击方法。本文提出的方法主要针对 JSP 程序中的 Java 程序进行了分析，对宿主语言和系统框架等其他方面引入的注入没有进行综合防护，可能导致逃逸。为使该方法有效地针对其他类型的语言，需要进一步解决程序静态分析技术的通用化问题，对流敏感的数据流分析方法进行优化，提高分析精度。

参 考 文 献

[1] OWASP Top 10-2013[EB/OL]. https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf. 2013/2014-08-21.

[2] 2016 年中国互联网安全报告[EB/OL]. http://zt.360.cn/1101061855.php?dtid=1101062370&did=490280697.

[3] BOYD S W, KEROMYTIS A D. SQLrand: Preventing SQL Injection Attacks[M]. New Ywork: Springer Berlin Heidelberg, 2004: 292-302.

[4] GOULD C, SU Z, DEVANBU P. Static checking of dynamically generated queries in database applications[J]. Acm Transactions on Software Engineering & Methodology, 2004, 16(4): 645-654.

[5] LE D G, LI X, GONG S R, et al. Research on second-order SQL injection techniques [J]. Journal on Communications, 2015, 36(S1): 85-93. (in Chinese)
乐德广，李鑫，龚声蓉，等. 新型二阶 SQL 注入技术研究[J]. 通信学报，2015，36(S1)：85-93.

[6] VALEUR F, MUTZ D, VIGNA G. A Learning-Based Approach to the Detection of SQL Attacks[C]// International Conference on Detection of Intrusions & Malware. 2005: 123-140.

[7] GAURAV T, PHILIP K. CHAN. On the learning of system call attributes for host-based anomaly detection [J]. International Journal on Artificial Intelligence Tools, 2011, 15(6): 875-892.

[8] TORRANO-GIMENEZ C, PEREZ-VILLEGAS A, ÁLVAREZ G. An Anomaly-based Web Application Firewall[C]// Security and Cryptography. 2009: 23-28.

[9] UWAGBOLE S O, BUCHANAN W J, FAN L. Applied Web Traffic Analysis for Numerical Encoding of SQL Injection Attack Features[C]// Proceedings of the European Conference on Cyber Warfare and Security(Eccws 2016). 2016.

[10] ČISAR P, ČISAR S M. The framework of runtime application self-protection technology [C] // International Symposium on Computational Intelligence and Informatics. IEEE, 2017: 000081-000086.

[11] SENDIANG M, POLII A, MAPPADANG J. Minimization of SQL injection in scheduling application development[C]// International Conference on Knowledge Creation and Intelligent Computing. IEEE, 2017: 14-20.

[12] MØLLER A. The Big Manual for the Java String Analyzer: Latest release: version 2.1-1, November 30, 2009[J]. Nucleic Acids Research, 2012, 40(14): 6520-33.

[13] SHELDON R. Transact-SQL Formatting Standards (Coding Styles)[EB/OL]. https://www.red-gate.com/simple-talk/sql/t-sql-programming/transact-sql-formatting-standards-(coding-styles).

[14] YAN M M, MUY M, HE Y J, et al. The Analysis of Function Calling Path in Java Based on Soot[J]. Applied Mechanics & Materials, 2014, 568-570: 1479-1487.

[15] WebCastellum[EB/OL]. https://sourceforge.net/projects/webcastellum/2014/2015-07-15.