基于 AOP 和动态污点分析的 SQL 注入行为检测方法

何成万,叶志鹏
(武汉工程大学计算机科学与工程学院,湖北武汉 430205)

摘 要: Web 应用程序时刻面临着来自网络空间中诸如 SQL 注入等代码注入式攻击的安全威胁. 大多数针对 SQL 注入攻击的检测方法执行效率较低,检测精度也不够高,特别是实现方法不易被重用. 根据注入型脆弱性特征提出了一种基于 AOP(Aspect-Oriented Programming)和动态污点分析的 SQL 注入行为检测方法,并通过方面(aspect)模块化单元对污点分析过程进行了封装,使得安全这类典型的程序横切关注点从基层子系统中分离,提高了检测代码的可重用性. 在污点汇聚点结合通知(advice)机制动态加载各类检测组件实现在运行时执行检测代码,从而应对 SQL 注入这类典型的针对 Web 应用程序的代码注入攻击方式. 实验表明,该方法能够在不修改应用程序执行引擎及源码的前提下实现自保护过程,有效防御重言式(tautologies)、逻辑错误查询(logically incorrect queries)、联合查询(union query)、堆叠查询(piggy-backed queries)、存储过程(stored procedures)、推理查询(inference query)、编码转换(alternate encodings)等 7 种典型的 SQL 注入攻击类型.

关键词: Web 安全; SQL 注入; 污点分析; 面向方面编程; 漏洞检测
中图分类号: TP309.2 **文献标识码:** A **文章编号:** 0372-2112 (2019)11-2413-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.11.025

SQL Injection Behavior Detection Method Based on AOP and Dynamic Taint Analysis

HE Cheng-wan, YE Zhi-peng
(*School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, Hubei* 430205, *China*)

Abstract: Web applications are constantly exposed to security threats from code injection attacks such as SQL injection in cyberspace. At present, most detection methods against SQL injection attacks have low execution efficiency and low detection accuracy, and are not easy to be reused. According to the characteristics of injection vulnerability, a SQL injection behavior detection method based on aspect-oriented programming and dynamic taint analysis is proposed, the taint analysis process is encapsulated by the aspect unit, so that the typical program crosscutting-concerns are separated from the base system, which improves the reusability of detection code. The Advice mechanism is used to dynamically load the various detection component implementations to execute the detection code at runtime to counter typical code injection attacks such as SQL injection against Web applications. Experiments show that this method can realize the self-protection process without modifying the application execution engine and source code, so as to effectively defend against seven typical types of SQL injection attacks such as tautologies, logically incorrect queries, union query, piggy-backed queries, stored procedures, inference query, alternate encodings, and so on.

Key words: Web security; SQL injection; taint analysis; aspect-oriented programming; vulnerability detection

1 引言

Web 应用系统经常面临外部攻击而导致的安全威胁,其安全性已成为网络空间安全领域的焦点问题[1]. 安全威胁产生的根本原因在于漏洞的存在,也称为软件脆弱性(vulnerability)[2]. 黑客通过构造各种未经验

收稿日期:2018-12-04;修回日期:2019-03-20;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.61272115)

证的畸形外部输入数据使得软件实体违背设计者初衷,从而改变程序控制流并窃取程序控制权[3]. SQL 注入[3]作为其中一种典型利用方式,是当前所面临的最为严重的 Web 安全威胁之一.

面向方面编程(AOP, Aspect-Oriented Programming)[4,5]提供了支持在源代码级别上对程序横切关注点模块化的编程方法和工具. 针对注入型脆弱性与接收外部输入行为密切相关这一特征,本文提出一种基于 AOP 和动态污点分析的检测及防御模型,通过引入方面(aspect)这一模块化单元对污点分析过程进行了封装,使得安全这类典型的程序横切关注点(crosscutting-concerns)从基层子系统中分离,提高了检测代码的可重用性,并结合通知(advice)机制动态加载各类检测组件实现在运行时的动态检测,从而应对 SQL 注入这类典型的针对 Web 应用程序的代码注入攻击方式.

实验部分使用 AspectJ 语言开发了基于探针形式的 Web 应用程序自保护原型系统,该框架易于采用 Servlet 组件的服务器端部署使用. 由方面封装的安全规范检测代码经编织器(Weaver)自动织入到基层子程序中,这一过程针对使用不同编程语言的应用程序具有较强的可移植性. 经对比实验验证,这是一种低侵入性、轻量级、高效的代码注入攻击检测方法,能够在不修改应用程序执行引擎及源码的前提下实现自保护过程,有效防御重言式(tautologies)、逻辑错误查询(logically incorrect queries)、联合查询(union query)、堆叠查询(piggy-backed queries)、存储过程(stored procedures)、推理查询(inference query)、编码转换(alternate encodings)等 7 种已知 SQL 注入攻击类型.

2 相关工作

软件漏洞检测技术是提升安全性及保证软件质量的基本手段与重要方法[6]. Seixas 等[7]识别和分类了 Web 程序中大多数通用安全漏洞,Ray 等[8]最终定义了 Web 应用程序和代码注入攻击方式. Shar 等[9]将 SQL 注入防御技术广义上划分为三类:防御性编码、静态(编译时)分析和运行时保护. Shinetal 的 SQLUnitGen[10]通过自动化测试定位风险点并生成检测报告,开发人员根据扫描结果人工修改缺陷代码从而修复软件漏洞. 赵宇飞等[11]以网络流量作为训练集,从真实网络环境中提取恶意请求特征检测 SQL 注入.

研究人员尝试根据一定策略推理 SQL 语句中可信与不可信部分[12],但此方式的缺陷是无法保证准确性. 文献[13]设计了一种内嵌在应用程序中的动态污点分析模型,通过封装基本字符和字符串类型,采用影子内存的方式将变量中的每个字符用一个对应的标识符标记其污点状态.

近年来,基于污点分析[14,15]的检测方法取得了长足的进步. 污点分析通过污点标记、污点传播与无害处理精确区分可信与不可信部分. 污点分析可分为动态污点分析和静态污点分析. 在静态污点分析中,Lam 等[16]利用静态数据流分析技术检查污点标记的用户输入数据是否到达 SQL 语句执行点从而挖掘 SQL 注入漏洞. Wassermann 等[17]使用上下文无关文法抽象 SQL 语句构造过程,用静态污点分析方法判定文法非终止符是否与用户输入相关. Jovanovic 等人提出的 Pixy[18]采用流敏感、过程间和上下文敏感的静态数据流分析方法挖掘 Web 应用漏洞. 但由于静态污点分析需要借助传统程序分析方法并且无法很好处理 Web 应用程序动态特性,导致其产生较高的漏报和误报. 动态污点分析能更好地应对程序动态特性,并基于不可信源和可信源划分为消极污点标记和积极污点标记. 其中消极污点标记对不可信输入进行污点标记,并在程序运行时传播污点标记,例如 Nguyen[19]等人通过修改 PHP 执行引擎中 String 类的数据结构与内置函数,在字符级对不可信数据进行污点标记与传播,并在汇聚点(sink)分析 SQL 语句语法结构. 但由于不可信输入的来源与类型分布广泛[20],对不可信数据的漏标记则会产生污点传播的欠污染问题从而造成检测结果的漏报. 相比较而言,可信数据来源比较固定,如程序员写入程序中的硬编码字符串[21],基于该特性提出了积极污点标记方法:Halfond 等[22~24]设计了 WASP 对程序中硬编码字符串设计可信标记并动态追踪标记传播轨迹,由于所有不可信输入均以未标记形式存在,此方法能够有效避免漏报.

3 方法设计与系统实现

污点分析的定义可以抽象成 < sources, propagation, sinks > 三元组形式[25],其中 source 即污点源,代表直接引入不受信任的数据至信息系统中;propagation 即污点传播,代表不受信任数据在程序中的传播轨迹;sink 即汇聚点,代表 SQL 语句执行等直接产生安全的敏感操作. 当数据在程序中流动时“信任”与“非信任”数据最终将会融合[26].

SQL 注入行为检测模型设计如图 1 所示,其处理流程可以划分为 4 个阶段:首先,在污点源入口对提交的外部输入数据进行污点标记;其次,通过污点传播算法追踪污点数据在程序运行期间的传播轨迹,即标记数据流传递过程中途径的方法,例如字符串中内置操作函数等;再次,对经无害处理模块净化后的数据移除对应的污点标记,降低性能损失;最后,在污点汇聚点执行污点检查策略,从而判定是否存在 SQL 注入攻击行为.

3.1 识别污点源

不可信数据作为 HTTP 请求参数传入服务器[27],经由服务端代码渲染后生成 SQL 语句,并最终在数据

库中执行.本文选取了一种目前被广泛应用于 Web 应用程序构建的典型服务器端框架-Java EE(Java platform,enterprise edition)为代表进行论述.

我们为 JDK 中 String、StringBuffer、StringBuilder 三个字符串类扩展了一个布尔类型成员变量来标记污点状态信息.其中 String 类是包含字符数组 char[ ] value、数组偏移量(起始索引)offset、计数值 count 和 hashcode 等成员变量的不可变数据类型.String 新建对象 s 表示由 s.value[s.offset..s.offset + s.count-1]范围内的字符序列组成的字符串[28].基于 AOP 静态横切关注点中的类型间声明机制在不影响类继承结构的前提下声明类实现指定接口.

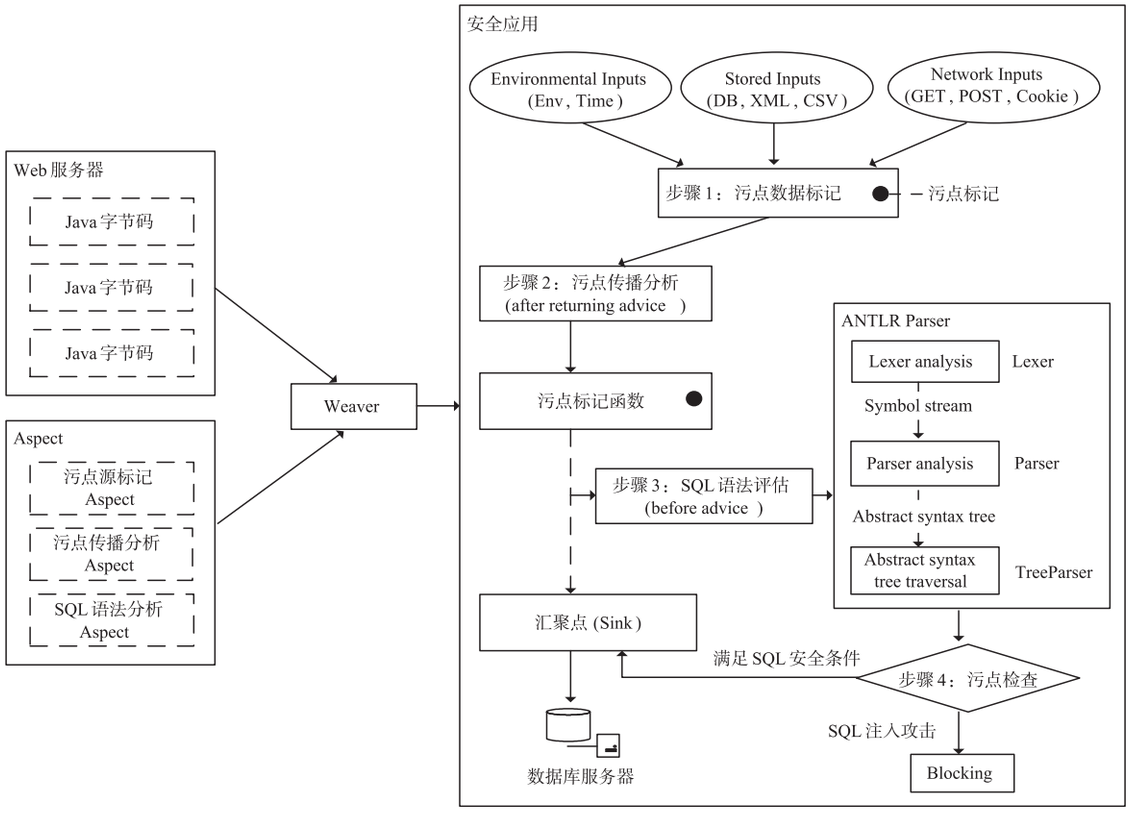

图1　动态污点分析检测模型

动态污点标记方法如图 2 所示,污点标记 Aspect 把用户的输入数据作为污点标记源,使用程序内部生成的 Hash 值作为起始头(starthash)和终结尾(endhash)标签对不可信数据进行首尾标记[29],并把拼接后的字符串 UserInputData 的 taint 属性设为 true.这种污点标记方法既保证了标记精度,又减少了性能开销.

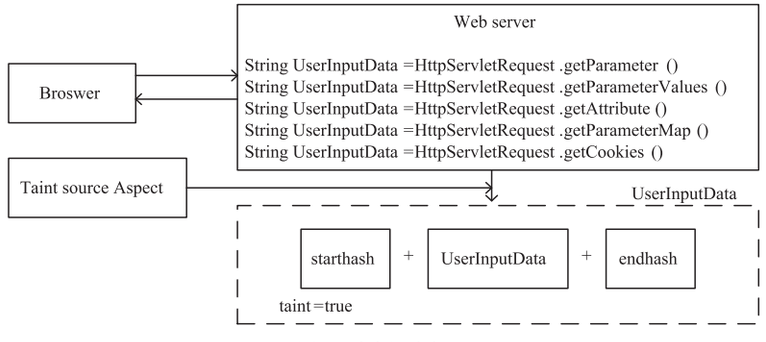

图2　动态污点标记

表 1 安全状态事件集[28]

名称	描述
StringData < isTainted, taintData >	isTainted:布尔类型成员变量 taintData:污点标记集
TaintedParamName < String >	监控函数集
TaintedInput < String, StringData >	不可信输入数据映射表
TaintedString < String, StringData >	污点传播事件映射表
SinkName < String >	敏感函数集

基于表 1 中所定义的安全状态(security state)概念[28],对报文中不同字段所引入信息系统的数据进行污点标记. 污点标记算法描述如算法 1 所示,若实参 param 是携带观测点的字符串实例变量,并通过赋值传播到指定污点引入方法 TaintedParamName 集合成员函数的形参列表中,则使用布尔型成员变量将函数返回值的污点状态信息标识为真,并作为污点标记源存储到不可信输入数据映射表 TaintedInput 中.

算法 1 输入事件污点标记算法

```
输入:方法返回值 tainted,方法参数 param.
sd← new StringData();
if param in TaintedParamName <String> then
  sd. isTainted← true;
  put <tainted,[starthash]sd. taintData[endhash]> in TaintedInput;
```

3.2 污点传播算法设计与实现

污点传播算法将直接影响整个污点传播流程的正确性,本文结合文献[28]和[29]的思想,设计了针对 Java 语言的污点传播算法(见算法 2).

污点传播分析需要考虑 3 类操作的传播方式:数据复制操作、数据运算操作及数据位操作. 由于携带首尾标记的观测点改变了字符串长度和字符位置,需要针对这类函数进行额外处理操作. 算法 2 中的污点传播规范指定了污点对象在方法调用过程中可能出现的 3 种情况,及针对污点对象出现的位置而分别定义的 3 类通知体代码执行流程:(1)tainted 表示污点对象出现在方法返回值中;(2)调用方法的实例对象 target 自身作为污点对象;(3)参数 args 表示污点对象传播到 TaintInput 或 TaintStrings 集合成员函数参数列表中.

Java 语言共有 216 个基础字符串函数,其中有 81 个函数具有污点传播功能,针对该部分函数需要逐一实现污点传播逻辑.

算法 2 污点传播事件算法

```
输入:方法参数 param,调用对象 target,方法传播值 propagatedData.
sd1,sd2,sd3 ← new StringData();
if <param,sd1> in TaintedStrings then
  if sd1. isTainted then
    if <target. sd2> in TaintedStrings then
      sd2. isTainted ← sd1. isTainted;
      concat([starthash]sd1. taintData[endhash],sd2. taintData);
      put <propagatedData,[starthash]sd2. taintData[endhash]>
        in TaintedStrings;
    else
      put <propagatedData,[starthash]sd1. taintData[endhash]>
        in TaintedStrings;
  else if <target,sd3> in TaintedStrings then
    put <propagatedData,[starthash]sd3. taintData[endhash]>
      in TaintedStrings;
if <param,sd1> in TaintedInput then
  if sd1. isTainted then
    if <target. sd2> in TaintedStrings then
      sd2. isTainted ← sd1. isTainted;
      add param in sd2. taintedData;
      put <propagatedData,[starthash]sd2. taintData[endhash]>
        in TaintedStrings;
        else
      sd ← new StringData();
      sd. isTainted ← sd1. isTainted;
      add param in sd. taintedData;
      put <propagatedData,[starthash]sd. taintData[endhash]>
        in TaintedStrings;
  else if <target,sd3> in TaintedStrings then
    put <propagatedData,[starthash]sd3. taintData[endhash]>
      in TaintedStrings;
if <target,sd3> in TaintedStrings then
  put <propagatedData,[starthash]sd3. taintData[endhash]>
    in TaintedStrings;
```

3.3 污点检查策略

标准 Servlet 程序中所有和数据库交互的相关操作都封装于 JDBC 库中,SQL 注入汇聚点 SinkName 包含了能够访问安全敏感数据库资源的切点方法,如 executeQuery()、executeUpdate()等. 使用 before 通知按照方法签名在运行时动态捕获程序中所匹配连接点中参数,对 SQL 语句作语法解析处理.

SQL 语法解析处理将 SQL 字符串递归解析为分别对应于关键字(keywords)、操作符(operators)和字面量(literals)的三类词法符号. 若发现经解析后的 SQL 语句中不包含非可信语法相关字符,表明该语句不存在 SQL 注入行为. 若经解析后 SQL 的任一词法符号携带污点标记,这些非可信语法相关字符可能破坏原 SQL 语句逻辑结构,造成 SQL 注入攻击危害. 为降低污点传播性能损失,上述经数据库执行后的函数返回值不再携带污点信息,并移除观测点中的首尾标记信息. SQL 注入攻击自防御算法描述如算法 3 所示.

算法 3　SQL 注入攻击自防御算法

```
输入:SQL 方法返回值 Value,调用敏感方法实参 param.
if <param,sd> in SinkName <String> then
  foreach tokens in param do
    if tokens are not tainted then
      SQL_exec(param);
    else
      prepend "\" before sd.taintData[starthash..endhash];
      SQL_exec(param);
```

4　实验与分析

为验证本文提出的 SQL 注入攻击检测模型的有效性,本节在 VMware Workstation 虚拟机平台中搭建了实验环境,分别对原型系统的功能指标和性能指标进行了评价. 测试环境选取 WAVSEP(Web Application Vulnerability Scanner Evaluation Project)[30]及 WebGoat[31]靶场系统,该靶场环境包含多种漏洞类型的 Web 应用程序. 实验分别选取了 4000 个正常样本和恶意样本,其中正常样本包含富文本字面量及无 SQL 语义字符串;恶意样本从 Github 中搜集了包含 7 种常见的 SQL 注入攻击.

4.1　功能测试

通过手工及自动化工具在 WAVSEP 和 WebGoat 两个靶场环境的每个注入点下频繁发送注入行为请求,模拟攻击者行为,并对检测结果进行了统计分析. 表 2 选取了其中具有代表性的 7 个 SQL 注入攻击实例,分别测试了本文与其余 5 种方法对 SQL 注入攻击的防御效果,每个测试用例分别针对一种类型的 SQL 注入攻击[24]. 表 2 中的 No. 1 ~ No. 6 都是按照每种攻击类型的规则构建的,具有一般性,第 7 种类型(编码转换)涉及到具体的字符编码,给出的测试用例一般性稍差. 表 3 中实验结果表明:对照组方法针对 7 种已知注入类型会产生不同程度漏报,我们的方法能够检测出已知的重言式(tautologies)、逻辑错误查询(logically incorrect queries)、联合查询(union query)、堆叠查询(piggy-backed queries)、存储过程(stored procedures)、推理查询(inference query)、编码转换(alternate encodings)等 7 种 SQL 注入攻击类型.

表 2　SQL 注入测试用例集

No.	类型	测试用例
1	重言式(tautologies)	SELECT balance FROM acct WHERE password = ′OR 1 = 1 – – ′
2	逻辑错误查询 (logically incorrect queries)	SELECT accounts FROM users WHERE login = ′AND pass = ′AND pin = convert (int,(select top 1 name from sysobjects where xtype = ′u′))
3	联合查询(union query)	SELECT accounts FROM users WHERE login = ′ UNION SELECT cardNo from CreditCards where acctNo = 10032 – – AND pass = ′AND pin =
4	堆叠查询 (piggy-backed queries)	SELECT accounts FROM users WHERE login = ′doe′AND pass = ′;drop table users – – ′AND pin = 123
5	存储过程 (stored procedures)	CREATE PROCEDURE DBO.isAuthenticated @userName varchar2,@pass varchar2,@pin int AS EXEC("SELECT accounts FROM users WHERE login = ′" + @userName + "′and pass = ′" + @password + "′and pin = " + @pin);GO
6	推理查询(inference query)	′legalUser′and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 – – ′
7	编码转换(alternate encodings)	SELECT accounts FROM users WHERE login = legalUser′;exec(char(0x736875746461776e)) – – AND pass = ′AND pin =

表 3　检测结果

	1	2	3	4	5	6	7
CSSE[20]	√	×	√	√	×	×	×
Joza[12]	√	√	×	√	√	√	×
CANDID[14]	√	√	√	√	×	√	×
WASP[23]	√	√	√	×	×	×	×
SQLCHECK[17]	√	×	√	√	×	×	×
本文方法	√	√	√	√	√	√	√

4.2　性能测试

首先测试污点传播算法对字符串类型不同操作间的性能损失影响. 平均运行时间越小,其执行速度越快,故对性能损失影响小. 表 4 中对比了本文与 PHP-Gate[3]和 WASP[23]两种方法间性能损失. 字符串操作性能损失的计算公式如下:

字符串操作性能损失 =(操作时间(开启保护后) – 操作时间(开启保护前))/(操作时间(开启保护前))(%)

本方法通过数组偏移量与计数值这两部分首尾标记在字符串粒度级别记录污点信息,结合 AOP 机制在确保精度的前提下一定程度上降低了污点传播的性能损失.

表 4 字符串操作性能比较

方法	字符串操作函数	性能损失/%
WASP[23]	String. Concat(String)	170.83
	StringBuilder. Append(StringBuilder)	7100.00
	StringBuilder. Append(String)	2500.00
PHPGate[3]	Stringcat	10.91
	concat(.)	12.78
本文方法	String. Concat(String)	8.60
	StringBuffer. Append(String)	9.70
	StringBuilder. Append(String)	9.20

5 结束语

本文针对 Web 应用程序中注入型脆弱性漏洞机理,提出了基于 AOP 和动态污点分析的检测方法及其在防御 SQL 注入攻击时的应用. 该方法使用 AOP 封装了污点标记、污点传播分析、污点检查等方法能有效检测 7 种已知 SQL 注入攻击类型. 由方面封装的安全规范检测代码经编织器织入到字节码中,其过程无须修改应用程序执行引擎和源码,针对使用不同编程语言的应用程序具有较强的可移植性. 我们计划在将来的工作中进一步增强方法的鲁棒性,有针对性地研究适用于其它 Web 安全威胁的检测方法.

参考文献

[1] 仝青,张铮,张为华,等. 拟态防御 Web 服务器设计与实现[J]. 软件学报,2017,28(4):883 - 897.
TONG Qing,ZHANG Zheng,ZHANG Wei-hua,et al. Design and implementation of mimic defense web server[J]. Journal of Software,2017,28(4):883 - 897. (in Chinese)

[2] 诸葛建伟,陈力波,田繁,等. 基于类型的动态污点分析技术[J]. 清华大学学报(自然科学版),2012,52(10):1320 - 1328 + 1334.
ZHUGE Jian-wei, CHEN Li-bo, TIAN Fan, et al. Type-based dynamic taint analysis technology[J]. Journal of Tsinghua University (Science and Technology), 2012, 52(10):1320 - 1328,1334. (in Chinese)

[3] 张慧琳,丁羽,张利华,等. 基于敏感字符的 SQL 注入攻击防御方法[J]. 计算机研究与发展,2016,53(10):2262 - 2276.
ZHANG Hui-lin,DING Yu,ZHANG Li-hua,et al. SQL injection prevention based on sensitive characters[J]. Journal of Computer Research and Development, 2016, 53(10): 2262 - 2276. (in Chinese)

[4] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming[A]. European Conference on Object-Oriented Programming[C]. Jyvaskyla, Finland: Springer Verlag, 1997. 220 - 242.

[5] Kiczales G, Hilsdale E, Hugunin J, et al. An overview of AspectJ[A]. Proceedings of the 15th European Conference on Object-Oriented Programming[C]. Budapest, Hungary: Springer Verlag, 2001. 327 - 353.

[6] 李舟军,张俊贤,廖湘科,等. 软件安全漏洞检测技术[J]. 计算机学报,2015,38(4):717 - 732.
LI Zhou-jun,ZHANG Jun-xian,LIAO Xiang-ke,et al. Survey of software vulnerability detection techniques[J]. Chinese Journal of Computers, 2015, 38(4): 717 - 732. (in Chinese)

[7] Seixas, Fonseca, Vieira. Looking at web security vulnerabilities from the programming language perspective: a field study[A]. IEEE International Conference on Software Reliability Engineering[C]. Mysuru, Karnataka, India: IEEE Press, 2009. 129 - 135.

[8] Ray D, Ligatti J. Defining code-injection attacks[J]. ACM Sigplan Notices, 2012, 47(1): 179 - 190.

[9] Lwin Khin Shar, Hee Beng Kuan Tan. Defeating SQL injection[J]. Computer, 2013, 46(3): 69 - 77.

[10] Shin Y, Williams L, Xie T. SQLUnitGen: Test Case Generation for SQL Injection Detection[R]. North Carolina: North Carolina State University, 2006.

[11] 赵宇飞,熊刚,贺龙涛,等. 面向网络环境的 SQL 注入行为检测方法[J]. 通信学报,2016,37(2):89 - 98.
ZHAO Yu-fei, XIONG Gang, HE Long-tao, et al. Approach to detecting SQL injection behaviors in network environment[J]. Journal on Communications, 2016, 37(2):89 - 98. (in Chinese)

[12] Naderi-Afooshteh A, Nguyen-Tuong A, Bagheri-Marzijarani M, et al. Joza: Hybrid taint inference for defeating web application SQL injection attacks[A]. IEEE/IFIP International Conference on Dependable Systems and Networks[C]. Rio de Janeiro, Brazil: IEEE Press, 2015. 172 - 183.

[13] 董敏. 基于动态污点分析的 SQL 注入攻击检测问题的研究[D]. 北京:北京工业大学,2014.
DONG Min. Research on the Attack Detection of SQL Injection Based on Dynamic Analysis[D]. Beijing: Beijing University of Technology, 2014. (in Chinese)

[14] Bandhakavi S, Bisht P, Madhusudan P, et al. CANDID: Preventing SQL injection attacks using dynamic candidate evaluations[A]. ACM Conference on Computer and Communications Security[C]. Alexandria, Virginia, USA: ACM, 2007. 12 - 24.

[15] 刘豫,王明华,苏璞睿,冯登国. 基于动态污点分析的恶意代码通信协议逆向分析方法[J]. 电子学报,2012,40(4):661 – 668.
LIU Yu,WANG Ming-hua,SU Pu-rui,FENG Deng-guo. Communication protocol reverse engineering of malware using dynamic taint analysis[J]. Acta Electronica Sinica,2012,40(4):661 – 668. (in Chinese)

[16] Benjamin Livshits V,Monica S Lam. Finding security vulnerabilities in java applications with static analysis[A]. Proceedings of the 14th Conference on USENIX Security Symposium[C]. California,USA:ACM,2005. 18 – 18.

[17] Wassermann Gary,Zhendong Su. Sound and precise analysis of web applications for injection vulnerabilities[J]. ACM Sigplan Notices,2007,42(6):32 – 41.

[18] Jovanovic N,Kruegel C,Kirda E. Pixy:a static analysis tool for detecting web application vulnerabilities[A]. IEEE Symposium on Security and Privacy[C]. Berkeley,USA:IEEE Press,2006. 258 – 263.

[19] Nguyen-Tuong A,Guarnieri S,Greene D,et al. Automatically hardening web applications using precise tainting[A]. IFIP 20th International Information Security Conference[C]. Chiba,Japan:Springer Verlag,2005. 295 – 307.

[20] Pietraszek T,Berghe C V. Defending against injection attacks through context-sensitive string evaluation[A]. International Conference on Recent Advances in Intrusion Detection[C]. Seattle,WA,USA:Springer Verlag,2006. 124 – 145.

[21] Mui R,Frankl P. Preventing web application injections with complementary character coding[A]. European Symposium on Research in Computer Security[C]. Leuven,Belgium:Springer Verlag,2011. 80 – 99.

[22] Halfond W G J,Orso A,Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks[A]. ACM Sigsoft International Symposium on Foundations of Software Engineering[C]. Portland,Oregon,USA:ACM,2006. 175 – 185.

[23] Halfond W,Orso A,Manolios P. WASP:Protecting web applications using positive tainting and syntax-aware evaluation[J]. IEEE Transactions on Software Engineering,2008,34(1):65 – 81.

[24] Halfond W G J,Viegas J,Orso A. A classification of SQL injection attacks and countermeasures[A]. Proceedings of the International Symposium on Secure Software Engineering[C]. Washington,USA:ACM,2006. 13 – 15.

[25] 王蕾,李丰,李炼,等. 污点分析技术的原理和实践应用[J]. 软件学报,2017,28(4):860 – 882.
WANG Lei,LI Feng,LI Lian,et al. Principle and practice of taint analysis[J]. Journal of Software,2017,28(4):860 – 882. (in Chinese)

[26] 马金鑫,李舟军,张涛,等. 基于执行踪迹离线索引的污点分析方法研究[J]. 软件学报,2017,28(9):2388 – 2401.
MA Jin-xin,LI Zhou-jun,ZHANG Tao,et al. Taint analysis method based on offline indices of instruction trace[J]. Journal of Software,2017,28(9):2388 – 2401. (in Chinese)

[27] 周颖,方勇,黄诚,等. 面向 PHP 应用程序的 SQL 注入行为检测[J]. 计算机应用,2018,38(1):201 – 206.
ZHOU Ying,FANG Yong,HUANG Cheng,et al. Detection of SQL injection behaviors for PHP applications[J]. Journal of Computer Applications,2018,38(1):201 – 206. (in Chinese)

[28] Chin E,Wagner D. Efficient character-level taint tracking for Java[A]. ACM workshop on Secure Web Services[C]. Chicago,IL,United States:ACM,2009. 3 – 11.

[29] 王溢,李舟军,郭涛. 防御代码注入式攻击的字面值污染方法[J]. 计算机研究与发展,2012,49(11):2414 – 2423.
WANG Yi,LI Zhou-jun,GUO Tao. Literal tainting method for preventing code injection attack in web application[J]. Journal of Computer Research and Development,2012,49(11):2414 – 2423. (in Chinese)

[30] Shay Chen. Web Application Vulnerability Scanner Evaluation Project[DB/OL]. https://github.com/sectooladdict/wavsep,2014 – 02 – 26.

[31] OWASP. WebGoat[DB/OL]. https://github.com/WebGoat/WebGoat,2019 – 02 – 10.

作者简介

何成万　男,1967 年生于湖北荆门. 现为武汉工程大学教授. 主要研究方向为基于复用的软件工程.
E-mail:hechengwan@hotmail.com

叶志鹏　男,1993 年生于湖北武汉. 现为武汉工程大学计算机科学与工程学院硕士研究生. 主要研究方向为数据安全、网络安全.
E-mail:googuo69@gmail.com