SQLMVED：基于多变体执行的 SQL 注入运行时防御系统

马博林[1]，张铮[1]，刘浩[2]，邬江兴[1]
（1. 信息工程大学，河南 郑州 450001；2. 网络通信与安全紫金山实验室，江苏 南京 211100）

摘 要：SQL 解析过程中利用随机化进行 SQL 注入攻击（SQLIA）防御的有效性是建立在攻击者不了解当前系统采用的具体随机化方法的基础上，因此，攻击者一旦掌握了当前系统的随机化形式，便能够实施有效的 SQLIA。为了解决该问题，基于多变体执行设计出一种 SQL 注入运行时防御系统，多变体间采用互不相同的随机化方法，攻击者注入的非法 SQL 无法同时被所有变体解析成功，即使在攻击者掌握了随机化方法的情况下，非法 SQL 也最多只能被某一变体解析成功，利用表决机制对多变体的响应结果或解析结果进行表决，及时发现异常，阻断 SQLIA 的攻击路径。面向 Web 服务实现了原型系统 SQLMVED，实验证明该系统能够有效抵御 SQLIA。

关键词：SQL 注入攻击；运行时防御；多变体执行；随机化
中图分类号：TP393
文献标识码：A
DOI: 10.11959/j.issn.1000−436x.2021046

SQLMVED: SQL injection runtime prevention system based on multi-variant execution

MA Bolin[1], ZHANG Zheng[1], LIU Hao[2], WU Jiangxing[1]
1. Information Engineering University, Zhengzhou 450001, China
2. Purple Mountain Laboratories, Nanjing 211100, China

Abstract: The effectiveness of combining SQL statement parsing with randomization to defend against SQL injection attack (SQLIA) was based on the fact that attackers did not know about the current method of randomization adopted by system. Therefore, once attackers had mastered the current method of randomization who can launch effective SQLIA. In order to solve this problem, a SQL injection runtime prevention system based on multi-variant execution was designed, the multi-variant apply randomization methods from any other, so that illegal SQL statements could not be parsed successfully by all variants. Even if attackers had mastered the method of randomization, illegal SQL statements could only be parsed successfully by a certain variant at most, meanwhile the parsing results of multiple variants were voted to find the abnormality in time and block attack path. The prototype system SQLMVED is implemented for Web services and experiments show that the prototype can effectively defeat SQLIA.

Keywords: SQL injection attack, runtime prevention, multi-variant execution, randomization

1 引言

随着“互联网+”新业态的快速发展，传统行业向网络服务发生转变，海量的数据存储在数据库中，互联网用户可以通过结构化查询语句（SQL, structured query language）随时随地访问网络服务数据。数据库遵循国际标准化组织（ISO, International Organization for Standardization）、国际电工委员会

收稿日期：2020−09−22；**修回日期**：2021−01−28
通信作者：张铮，ponyzhang@126.com
基金项目：国家自然科学基金资助项目（No.61521003）；国家重点研发计划基金资助项目（No.2018YFB0804003）
Foundation Items: The National Natural Science Foundation of China (No.61521003), The National Key Research and Development Program of China (No.2018YFB0804003)

（IEC, International Electrotechnical Commission）等发布的统一标准，虽然提高了网络服务间数据格式的一致性，但易被利用的特性也使网络服务数据面临越来越多的安全威胁。

目前，SQL 注入攻击（SQLIA, SQL injection attack）是危害网络服务数据安全的主要威胁之一，“OWASP TOP 10”项目最近三次发布的 *The Ten Most Critical Web Application Security Risks* 报告显示，SQLIA 在所有统计的安全威胁分类中连续多年排名首位。

文献[1]提出了一种基于指令集随机化的 SQLIA 防御方法，通过对 SQL 进行随机化变化，使攻击者不能预知应用程序的 SQL 形式，无法完成 SQLIA。该文献推动了 SQLIA 防御方法由 SQL 运行前的检测技术向运行时的主动防御技术转变，其中的随机化方法在黑盒环境下能够有效抵御 SQLIA，但是攻击者一旦通过社会工程学或者在白盒环境下掌握了随机化方法，调整注入代码，便能实现有效的 SQLIA。

为了解决该问题，本文改进文献[1]的技术思路，提出了一种基于多变体执行技术的 SQLIA 防御方法。结合多变体执行与随机化防御思路，构建 SQLIA 运行时防御系统架构，并根据数据的读、写操作设计针对性的表决方法，基于 Web 服务实现了原型系统 SQLMVED（SQL multi-variant execution defense）。安全性评估和实验测试表明，该方法无论在攻击者是否掌握了防御机制的情况下，均能够有效抵御 SQLIA。

2 SQL 注入攻击研究

2.1 SQL 注入攻击原理

SQL 是用于数据库查询、更新和管理的高级非过程化编程语言，最初由 IBM 公司研制开发，目前广泛地应用于程序设计开发中。图 1 给出了典型的 Web 服务架构[2]。客户端通过应用层协议向服务器端发送请求（步骤 1)）。服务器通过 CGI/FastCGI 格式将请求转发至 CGI（common gateway interface）应用程序进行解析处理（步骤 2)）。执行目标程序，目标程序若执行回调功能代码，则重复步骤 3)；若执行数据库操作功能代码，则进行步骤 4)，例如程序调用 PHP-CGI 中的 mysqli_query、oci_execute 等函数；若执行具有系统命令调用功能的代码，则进行步骤 5)。最终，通过步骤 6)和步骤 7)将服务器端的响应结果返回至客户端。

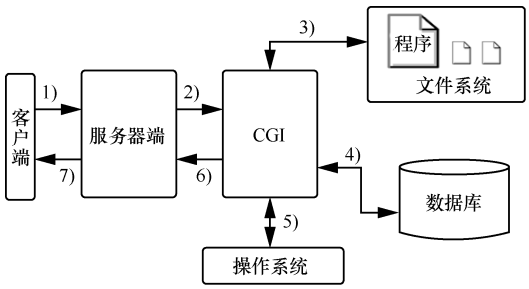


图 1 典型的 Web 服务架构

SQLIA 的语句执行在步骤 4)，攻击者在用户输入中构造恶意 SQL，由 CGI 将其与目标程序中的 SQL 拼接后发送至数据库执行，从而实现非法操作。相较于发生在步骤 3)的代码注入攻击和发生在步骤 5)的命令注入攻击，SQLIA 不依赖于操作系统以及应用程序运行环境，在漏洞存在的情况下攻击实施的门槛较低。

2.2 SQL 注入攻击威胁

SQLIA 是针对数据库的攻击技术，攻击者利用应用程序开发阶段的漏洞或缺乏对输入验证，向应用程序中注入恶意 SQL，从而达到查询、更改或删除数据的目的。根据网络空间安全[3]的框架划分，SQLIA 主要是利用应用层的漏洞或后门，破坏数据层的安全性，SQLIA 主要将以下网络服务特性或功能作为攻击目标。

1) 绕过身份验证。网络服务通常设置身份验证过程，以用户输入与数据库查询结果比对的方式实现，攻击者通过 SQLIA 可绕过比对过程，使用非法用户成功绕过身份验证。

2) 窃取数据。网络服务中的大量数据存储在数据库中，如身份证号、手机号、银行卡号等信息，攻击者通过 SQLIA 获取敏感信息，是窃取数据的主要途径。

3) 破坏可用性。攻击者通过 SQLIA 不仅可以窃取数据，还可以更改或删除数据，破坏数据完整性，甚至停止数据库服务，达到拒绝服务的目的。

4) 数据库指纹识别。攻击者通过 SQLIA 识别数据库类型、版本等信息，为发起针对性的 0day 或 APT（advanced persistent threat）攻击做准备。

5) 绕过输入检查。网络服务为了抵御 SQLIA，选择性地部署外挂式防御措施，对用户输入检查是否存在注入语句，攻击者通过构造特殊的 SQLIA 绕过输入检查。

2.3 SQL 注入攻击分类

本节主要介绍目前已知的 SQLIA 类型，由于

无法穷举每种攻击类型的所有变化形式，表 1 以用户查询为场景，为每种 SQLIA 类型提供了代表性的示例语句。真实的网络对抗中，攻击者通常组合使用多种 SQLIA 类型来完成对特定目标的攻击。

1) 重言式。此类攻击在条件语句中构造注入代码，使 SQL 的判定结果永远为真，从而绕过条件语句中的验证过程。

2) 批量查询。此类攻击利用查询分隔符，注入额外的 SQL，使原始语句完成操作后，继而执行注入的批量语句。

3) 错误回显。数据库错误消息通常包含有用信息，此类攻击向服务端注入语法错误、类型转换错误或逻辑错误的 SQL，利用错误回显获得数据库关键信息。更进一步地，攻击者将错误语句与查询语句拼接，利用错误信息回显查询结果。

4) 联合查询。此类攻击注入 UNION 语句，改变返回的数据结果，从而绕过验证过程或获得敏感数据。

5) 存储过程。此类攻击属于数据库中设置的额外抽象层，可以由开发人员进行编程，因此同样遭受 SQLIA 威胁。攻击者可以注入 SHUTDOWN、DROPTABLE 等语句，造成数据破坏或拒绝服务。

6) 盲注。当应用程序隐藏了数据库的不安全错误消息而不再产生错误回显时，攻击者既可以采用布尔盲注方法，利用应用程序的正确或错误响应，判断注入语句的执行结果；也可以采用时间盲注方法，利用 sleep、benchmark 等函数配合 if-then 语句，通过响应时间判断注入语句的执行结果，从而获得有效信息。

7) 交替编码。攻击者使用交替编码（例如十六进制、ASCII、Unicode）修改注入查询语句，从而绕过输入过滤器的检测。

2.4 SQL 注入攻击防御方法

根据防御发生的时机，Shar 等[4]将 SQLIA 防御技术在广义上分为防御性编码、运行前分析和运行时防御 3 类。

防御性编码是在开发阶段约束开发者的代码编写习惯，要求开发者在实现过程中更多地采用参数化查询或存储过程转义所有用户提交的参数、数据类型校验、白名单筛选等防御性编码方式。Mcclure 等[5]和 Cook 等[6]创新地提出了 SQL 开发模式，开发者通过提供的 API，能够自动实现具有数据类型校验、输入过滤和转义的 SQL，消除大量可能导致 SQLIA 发生的编码问题。此类防御技术尽管是有效的，但需要开发者使用新的编程方式，并且不会对已存在的应用程序提供安全防护，应用范围狭窄。因此，防御性编码类的 SQLIA 防御技术未受到国内外学者的持续关注，研究成果有限。

运行前分析一般是通过对应用程序进行漏洞扫描、自动化测试等来发现 SQLIA 漏洞，从而确保应用程序是安全可靠的，或者是在 SQL 执行之前利用数据挖掘、黑白名单等技术手段对输入检查，来保证用户输入不含有 SQLIA 代码。在漏洞测试方面，Kiezun 等[7]开发了 ARDILLA 工具，能够根

表 1　　SQLIA 类型及示例

攻击类型	攻击目标	示例
重言式	1)、2)	SELECT account FROM users WHERE login = ’zhangsan’ or 1=1-- ’ AND pass=’1234’;
批量查询	1)、2)、3)	SELECT account FROM users WHERE login = ’zhangsan’ AND pass=’ ’; drop table users -- ’;
错误回显	2)、3)、4)	SELECT account FROM users WHERE login = ’zhangsan’ AND (SELECT 1 FROM (SELECT count(*),concat((SELECT account FROM users WHERE login = ’zhangsan’),floor(rand(0)*2))x FROM users group by x)a)-- ’ AND pass=’1234’;
联合查询	1)、2)	SELECT account FROM users WHERE login = ’zhangsan’ UNION SELECT * FROM users WHERE login = ’zhangsan’ -- ’ AND pass=’1234’;
存储过程	1)、2)、3)	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
盲注	2)、4)	布尔盲注(bool blind injection) SELECT account FROM users WHERE login=’zhangsan’ AND SELECT length(database())>n-- ’ AND pass=’1234’; 时间盲注(time blind injection) SELECT account FROM users WHERE login=’zhangsan’ AND if(length(database())>=8,sleep(5),1)-- ’ AND pass=’1234’;
交替编码	5)	SELECT account FROM users WHERE login=’zhangsan’; exec(char(0x73687574646f776e)) -- AND pass=’’;

据应用程序自动生成 SQLIA 语句，从而检测应用程序是否存在 SQLIA 漏洞；孙歆等[8]设计了基于缺陷注入的模糊测试方法，服务代理获取用户请求后转发至模糊测试引擎，引擎采用 DOM4J 解析 XML 伪码配置，生成测试用例进行注入测试。针对输入检测，防御方法通常设计在防火墙层面[9-12]或 CGI 层面[13-15]。在防火墙层面的相关研究较丰富，Kar 等[9]提出的 SQLiGoT 采用令牌图和支持向量机能有效地识别 SQLIA 语句；韩宸望等[10]提出了基于 SQL 语法树的过滤方法，在用户请求进入服务端前对比输入语句的 SQL 语法树特征是否与合法特征一致，有效抵御 SQLIA；赵宇飞等[11]基于长度、连接频率和特征串对网络流量进行分析，有效检测 SQLIA。在 CGI 层面，张慧琳等[13]提出利用编码值判断 SQL 中敏感字符的来源、转义非可信敏感字符，基于 PHP 的 Zend 引擎实现了原型系统 PHPGate，阻止 SQLIA 发生。运行前分析类的 SQLIA 防御技术，无论是通过生成测试用例进行自动化测试，还是构建合法或非法的语句特征进行输入检测，其核心思想是在掌握恶意输入规则、语法漏洞集合或者建立有效白名单的前提下实施有效防护，当新型的攻击出现时，此类技术保证的攻守平衡会被打破，继而再努力解决新的问题，产生新的研究成果。

运行时防御在运行前分析的基础上，一方面在程序运行时进行监控，确保应用程序的行为始终在信任状态，Halfond 等[16]提出了 AMNESIA 工具，该工具运行在 CGI 层，将静态分析和运行时防御相结合，在静态部分创建语句模型，然后在运行时监测这些动态生成的语句，一旦不符合设置的合法语句模型，该工具将阻止语句发送至数据库；何成万等[17]提出了一种基于 AOP（aspect-oriented programming）和动态污点分析的检测方法，通过污点标记方法区分可信与非可信数据源，在 JDBC 中解析发现非可信语法。另一方面改变攻击者对目标系统的认知，使注入代码失效，Boyd 等[1]基于 SQL 的随机化设计了 SQLRand 系统，对 SQL 关键字进行随机化处理，例如，原始未随机化的 SQL 为 SELECT account FROM users WHERE login=“AND pass=”，采用 key 为“123”随机化后的 SQL 为 SELECT123 account FROM123 users WHERE123 login=“AND123 pass=”，这样可以使攻击者不能预知应用程序的 SQL 形式，SQLRand 系统架构如图 2 所示。在 CGI 与数据库之间增加代理，用于解析语句和去随机化处理，如果解析语句发现含有未随机化的 SQL 关键字，则进行异常处理，不予执行，从而实现 SQLIA 的有效防御。

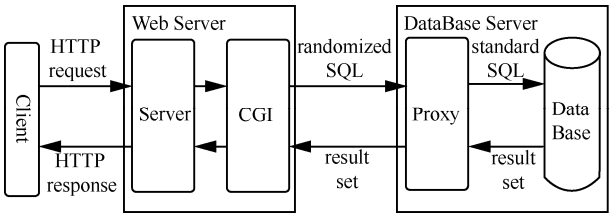


图 2 SQLRand 系统架构

SQLRand 系统在黑盒环境下能够有效抵御 SQLIA，但是存在被猜解的威胁，目前通过 hashcat[18] 工具暴力破解，在 GPU 破解速度为 9×10^9 次/s 的条件下，每秒可以尝试 900 万种组合，攻击者在秒级时间内就可以完成对 4 位标签的猜解。而攻击者在白盒环境下能够准确掌握 SQL 关键字的随机化方法，随即调整注入代码，立刻实现有效的 SQLIA。本文提出了一种基于多变体执行的 SQLIA 运行时防御方法，多变体间采用互不相同的随机化方法，即使在攻击者掌握了随机化方法的情况下，非法 SQL 也最多只能被某一变体解析成功，利用表决机制对多变体的响应结果或解析结果进行表决，实现 SQLIA 的有效防御。

3 基于多变体执行的 SQLIA 防御方法

3.1 多变体执行技术研究

程序冗余执行的技术思路最早应用在程序调试、错误容忍等保证程序可靠性的领域中，后来研究者在程序冗余执行的基础上，将冗余执行的软件进行多样化设计，从而解决软件的安全问题，因此在软件安全领域将冗余执行的多样化程序称为多变体。多变体执行最早由 Knowlton[19]提出，分别将 2 个功能逻辑等价程序的实现代码分成小片段，然后利用跳转指令，在保证程序语义不变的前提下，对代码片段进行重组。2 个程序并行执行时，由 CPU 检查程序的执行语义是否相等，从而可以预防控制转换越界、错误使用野指针等安全问题。

Cox 等[20]提出了较完整的软件多变体执行架构 N-Variant，如图 3 所示。该架构通过内存地址空间随机化和指令集随机化技术生成冗余变体，在不关注攻击方式的情况下，实现了对信息泄露攻击的有

效防御，N-Variant 奠定了多变体执行技术的基础，并总结出该技术面临着变体生成、监控方法等关键问题。

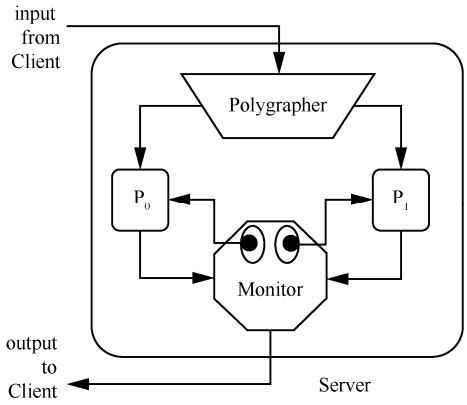


图 3　N-Variant 系统架构

有关多变体执行技术的众多研究中，Berger 等[21]提出了 DieHard，其架构如图 4 所示。该架构对 init_heap、malloc、free 函数进行重新设计，采用新的 init_heap 和 malloc 函数结合，使用随机化技术为多变体生成不同的堆对象布局。多变体以独立的进程运行，通过管道从主进程接收输入，然后在新的 free 函数中将输出写入共享空间的缓冲区中，调用表决进程对所有变体的输出进行比较，从而有效抵御内存错误，保证内存安全。该团队又在 Diehard 的基础上，扩展内存分配机制，设计了 Dieharder[22]系统架构，使变体能够在不连续的内存页面上随机分配，解决了 Diehard 中单个堆块溢出后覆盖其他堆块空间的问题。

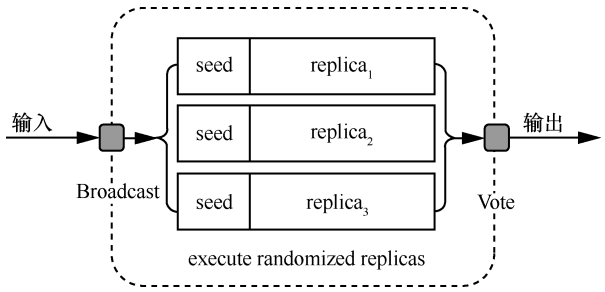


图 4　DieHard 系统架构

Novark 等[23]在 DieHard 的基础上，扩展提出了 Exterminator，如图 5 所示。该架构改善了 Diehard 的表决算法，能够表决发现内存错误位置，并在运行时生成补丁进行适应性的修补，并且 Exterminator 首次为多变体执行架构增加了动态反馈机制。

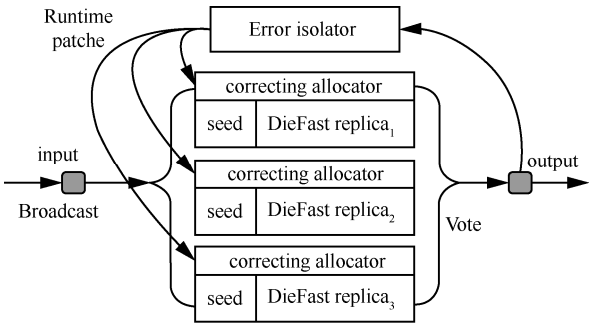


图 5　Exterminator 系统架构

以 N-Variant、Diehard、Dieharder、Exterminator 等架构为代表进行分析，多变体执行是在多样化技术的基础上，利用冗余、动态机制，解决软硬件同质化带来的安全问题。多变体只要在受保护的攻击面中存在更小的重叠范围，便可以作为变体的“源”来构建冗余执行架构。

我国邬江兴院士[24]提出了由构造决定安全的拟态防御理论，研究了冗余、异构、动态特性解决内生安全问题的有效性，系统地构建了动态异构冗余架构，如图 6 所示。该架构由输入代理、异构组件池、异构执行体集、调度器、表决器组成。动态异构冗余架构下同时运行多个功能等价、结构相异的执行体，由于网络攻击对于目标环境的依赖性，不同执行体对具有攻击行为的输入会产生不一致的输出，通过表决就能够发现执行体产生的异常结果。因此，动态异构冗余架构能够在不依赖先验知识的情况下，有效防御针对已知或未知漏洞后门发起的攻击。

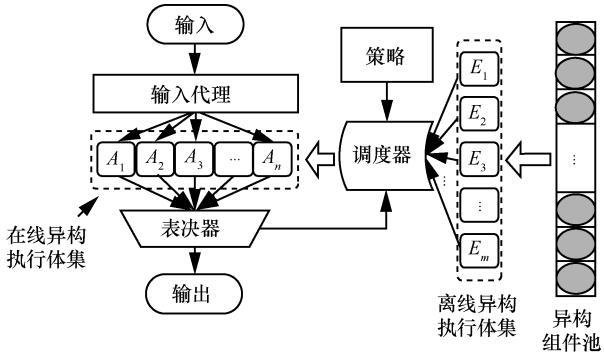


图 6　拟态防御动态异构冗余架构

拟态防御相较于多变体技术，具有完备的理论体系，已发行多版中英文著作[25]。对比两者的技术思路可以发现，多变体技术属于拟态防御作为内生安全构造技术解决软件安全的范畴。

本文基于多变体执行技术，设计实现了 SQLIA

运行时防御原型系统 SQLMVED。基于本节的研究内容，SQLMVED 的衍生路线如图 7 所示。

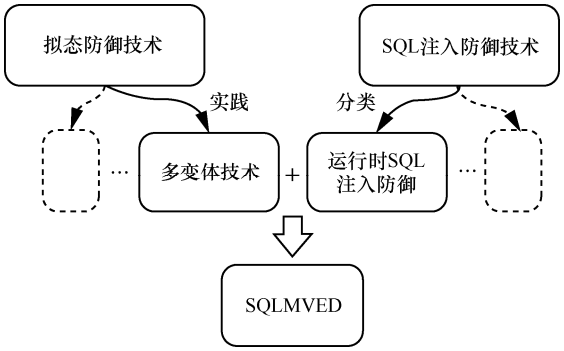


图 7 SQLMVED 的衍生路线

3.2 SQLMVED 系统设计

大部分多变体执行架构采用 Leader-Follower 或者 Master-Slave 设计，针对不予表决的系统调用，监控器将其分派至 Leader/Master 变体执行，再将结果同步至 Follower/Slave 变体中，这种方法能够有效降低假阳性，却增加了安全威胁，Leader/Master 变体一旦被控制，多变体执行架构的防御有效性将大打折扣。本文提出基于多变体执行的 SQLMVED 系统，为避免 Leader-Follower/Master-Slave 模式的安全威胁，采用同步模式，系统架构如图 8 所示，系统由用户请求代理、负责处理请求的多变体、数据库代理，以及数据库组成。

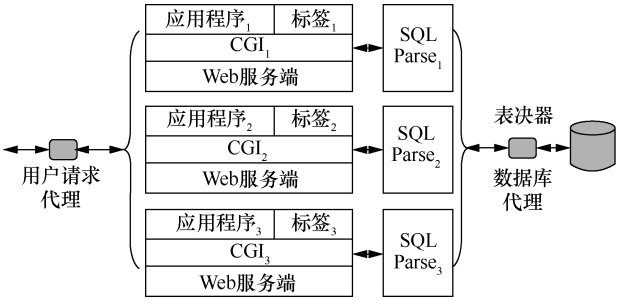


图 8 SQLMVED 系统架构

用户请求代理是网络服务的出入口[26]，当接收到用户发起的请求时，其将请求复制 3 份再转发至多变体；当接收到多变体的响应时，进行表决，若存在多数一致的响应，则返回至用户，否则拒绝响应。用户请求代理基于 Nginx 实现，利用 ngx_http_upstream_module 模块的负载功能和本文 3.3 节设计的算法 1，实现同一用户请求向多变体的转发，以及同一用户响应的表决。

多变体由 Web 服务端、CGI 以及部署的应用程序组成，其中应用程序采用随机化方法对 SQL 进行随机化变化，且变体之间采用的标签是不同的。

数据库代理由独立地服务于各个变体的 SQL 语法解析器 SQLParse 和负责比较其解析结果的表决器组成。SQL 语法解析器基于 sqlparse/ keywords.py 文件，使 tokens.Keyword 与随机化标签保持一致，完成解析随机化 SQL 和去随机化的过程。由于只改变 tokens.Keyword 的表现形式，不改变 SQL 解析的正常流程，因此本文提出的 SQL 随机化方法不会影响 SQL 解析器的正常功能。例如，变体 E_1 采用的标签为“123”，其对应的语法解析器为 SQLParse_1；变体 E_2 采用的标签为“456”，其对应的语法解析器为 SQLParse_2。以 SQLSELECT a FROM b WHERE c='?';（其中?为占位符）为例，当攻击者掌握了随机化标签，调整输入为 admin' or123 '1' = '1 时，SQLParse_1 解析的结果，在未经去随机化处理时的结构如图 9 所示。由于攻击者注入的“or123”符合变体 E_1 的变化方法，因此在变体 E_1 中注入攻击成功。

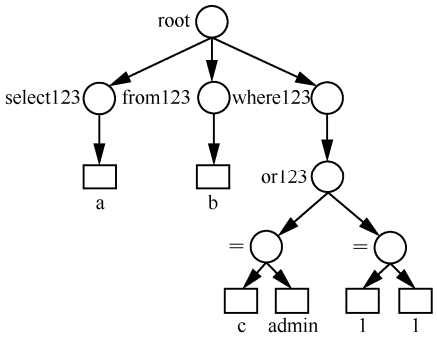


图 9 注入成功时的 SQL 语法树

SQLParse_2 解析的结果，在未经去随机化处理时的结构如图 10 所示。在变体 E_2 中，攻击者注入的“or123”不符合其变化方法，因此注入攻击失败。同理，在变体 E_3 中，注入攻击也失败。

变体 E_1、E_2、E_3 中的 SQL 经去随机化处理后由数据库执行，变体 E_2 和变体 E_3 的执行结果一致，且不同于注入成功的变体 E_1 的执行结果。

3.3 SQL 表决方法设计

表决作为多变体执行架构中的关键技术，决定着多变体系统能否发现异常。在多变体系统中，表决点要么设置在变体执行前对输入进行表决，要么设置在变体执行后对输出进行表决。考虑到数据库服务中数据读、写操作的不同影响，SQLMVED 系

统针对数据的读、写操作采取的处理方式也不同。

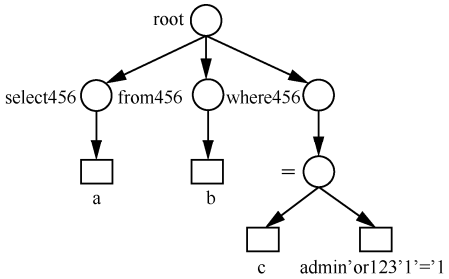


图 10　注入失败时的 SQL 语法树

读操作。数据的读操作不会引起数据的变化，读操作的表决点设置在用户请求代理处，用户请求代理对变体返回用户的输出进行表决，充分利用协议代理功能，保证参与一次表决的变体返回结果的同源性。

写操作。数据的写操作必然使数据产生变化，若表决点继续设置在用户请求代理处，当其发现不一致时还需要对数据进行还原操作，增加了系统的复杂度。因此，写操作的表决点设置在数据库代理处，表决器对经过随机化语法解析器去随机化的语句进行表决，只有通过表决的写操作语句才会由数据库执行。

结合 3.2 节的 SQLMVED 系统设计，系统处理用户请求的流程如图 11 所示。

1) 用户请求代理表决算法

用户请求代理将请求复制，转发至冗余多变体执行并获取响应结果，对 HTTP 响应数据结构的响应体[27]进行表决，用户请求代理表决算法描述如算法 1 所示，若存在多数一致的响应，则将其中的响

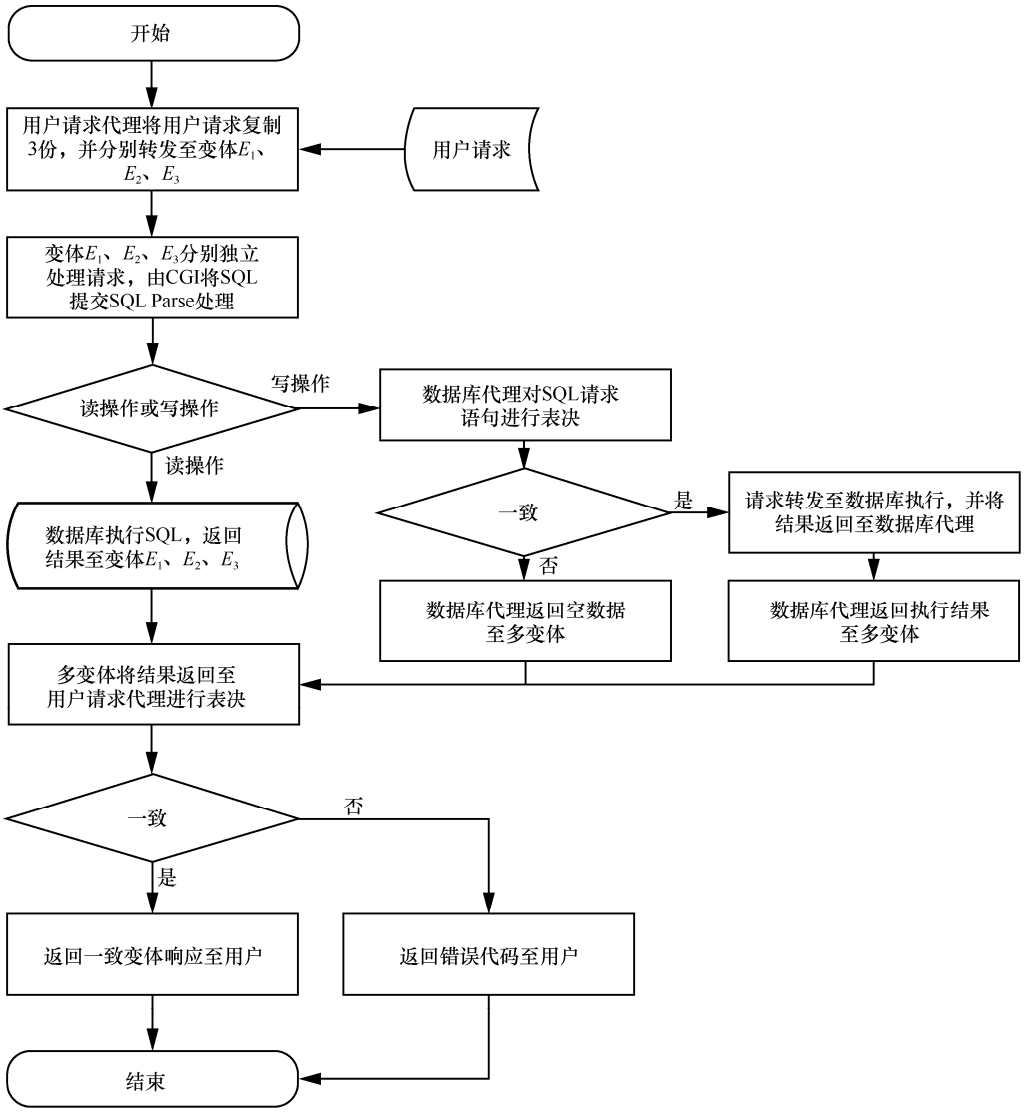


图 11　SQLMVED 系统流程

应结果返回至用户；若无法通过表决，则拒绝响应，返回错误状态码。

算法 1 用户请求代理表决算法

1) begin
2) request ← getrequst()//获取用户请求
3) if(request != null)
4) kernel_uuid←cat /proc/sys/kernel/ random/ uuid
5) uuid←strcat(kernel_uuid, time)
6) request.set_header('uuid', uuid)
7) for i=1 to 3
8) request$_i$←request
9) response$_i$←ngx(request$_i$)
10) end for
11) if(compare(response$_1$,response$_2$,response$_3$)== 1)
12) return response$_x$//返回多数一致响应
13) else
14) return errorcode//返回错误代码
15) end if
16)end if
17)end

2) 写操作表决方法

数据库代理负责将冗余的写操作合并成一次请求转发至数据库执行，因此无法以用户请求代理的方式处理。为保证参与表决的 SQL 是由同一请求复制分发而来的，算法 1 中的步骤 4)~步骤 6)采用唯一性标识码和时间戳在请求的头部对请求进行标记。在此基础上，通过修改应用程序源代码，将算法 1 中的标记信息传递至应用程序的 SQL 中，使数据库代理通过该标识能够识别不同变体的同一用户请求。如示例程序 1 所示，应用程序获取请求头部中的标记信息，之后通过变量将其以注释的形式拼接至 SQL 中。

示例程序 1 修改登录用户密码

1) begin
2) username←getinput()
3) passwd←getinput()
4) request←getrequst()//获取用户请求
5) if(request != null)
6) sql_uuid←request.get_header(uuid)
7) query←"UPDATE users SET pass = '".passwd."' WHERE login = '".username."' --".sql_uuid.";"
8) execute(query)
9) end if
10) end

数据库代理的表决器配置高速缓存，变体的 SQL 临时存储在缓存中，并设置超时时间，当未达到超时时间，若缓存中存在一致的 3 条语句，则消除缓存记录，转发至数据库执行一次；当达到超时时间，则直接消除缓存记录，不予转发至数据库执行，返回空数据至多变体。多变体在得到数据库代理返回的数据后，返回响应结果至用户请求代理，此时用户请求代理表决器再次对结果进行表决，由于写操作的请求已经在数据库代理处完成表决，因此多变体的响应结果在用户请求代理处表决是一致的。

3.4 安全性评估

根据 SQLMVED 系统的设计方法，在对其进行安全性评估时，需要满足以下几个条件。1) 多变体 E_1、E_2、E_3 同时处理请求，都具有独立执行 SQL 的能力；2) 多变体 E_1、E_2、E_3 在不遭受 SQLIA 时，均能够正确地处理 SQL；3) 多变体 E_1、E_2、E_3 任意之间不存在协同或协作的联系，假设用户请求代理和数据库代理的设计实现中没有漏洞或后门，攻击者无法以此为跳板协同多变体；4) 假设表决算法的设计实现中也不存在漏洞或后门，由此做出以下形式化推论。

设系统存在变体集合 $E=\{E_i\}_{i=1}^{n=3}$，输入语句集合 $I=\{I_k\}_{k=1}^{l}$，变体 E_i 使用的随机化方法集合 $V_i=\{V_{ij}\}_{i=1,j=1}^{n=3,m}$，变体 E_i 解析输入语句 I_k 时，得到的解析结果为 $R_{ijk}=E_i(V_{ij},I_k)$，变体在图 12 展示的 IEO 模型中，针对输入语句集合 I，解析结果集合为 $O=\{R_{ij}\}_{i=1,j=1}^{n=3,m}$。

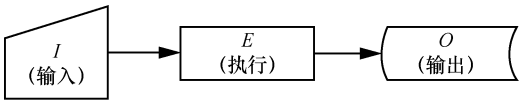


图 12 IEO 模型

对于 i,r=1,2,3 且 $i\neq r$，存在随机化方法 $\text{key}\in V_i\bigcap V_r$，攻击者采用非法输入语句 I_k 对 E_i,E_r 实施注入攻击，攻击成功的解析结果满足

$$P\{R_{ijk}=R_{rsk}\}=P\{E_i(V_{ij},I_k)=E_r(V_{rs},I_k)\}=$$
$$P\{V_{ij}=\text{key},V_{rs}=\text{key}\}=P\{V_{ij}=\text{key}\}P\{V_{rs}=\text{key}\} \quad (1)$$

E_i 使用的随机化方法集合 V_i 中随机化方法个数记为 $\text{card}(V_i)$，$\exists\omega=\bigcap_{i=1}^{n=3}V_i$，且 $\text{card}(\omega)\leqslant\text{card}(V_i)$，$\tau$

为集合 ω 中的元素。假设对于任意一个变体 $E_i \in E$，采用随机化方法集合 V_i 中每个方法元素的概率是相等的。根据式(1)可得出，SQLMVED 系统在某种随机化设计下被注入成功的概率为

$$P=\sum_{\tau\in\omega}P\{V_{1j}=\tau,V_{2s}=\tau,V_{3t}=\tau\}=$$

$$\sum_{\tau\in\omega}\prod_{i=1}^{n=3}P\{V_{ix}=\tau,V_{ix}\in V_i\}=\prod_{i=1}^{n=3}\frac{\mathrm{card}(\omega)}{\mathrm{card}(V_i)} \qquad (2)$$

由此可见，SQLMVED 系统的防御能力取决于变体之间随机化方法的差异性，由于 key_1、key_2、key_3 之间互不相同，因此变体之间不存在任何相互一致的随机化方法，也就是 card(ω)=0，因此攻击者无法有效实施 SQLIA。

4　实验与分析

为验证本文设计的 SQLMVED 系统的有效性，本节主要从防御有效性测试和性能测试对其进行实验与分析。

4.1　防御有效性测试

防御有效性测试中，采用 VMware Workstation 虚拟化环境搭建实验平台，普通未防护的环境为 LAMP（Linux、Apache、MySQL、PHP）架构，施加防护的环境为本文提出的 SQLMVED 架构，用户请求代理、变体（E_1、E_2、E_3）、数据库代理、数据库均采用虚拟机（VM, virtual machine）进行部署，具体配置如表 2 所示。

表 2　防御有效性测试环境

架构		操作系统	服务器	中间件	数据库
LAMP		CentOS	Apache	php-fpm	MySQL
本文架构	用户请求代理	CentOS	Nginx	python	—
	E_1 变体	CentOS	Apache	php-fpm	—
	E_2 变体	CentOS	Apache	php-fpm	—
	E_3 变体	CentOS	Apache	php-fpm	—
	数据库代理	CentOS	—	sqlparse python	—
	数据库	Windows10	—	—	MySQL

选取 bWAPP、DVWA、SQLI-LABS 这 3 种靶机应用程序中未实施安全防护策略的部分，采用 sqlmap 工具与人工构造注入结合的方式分别对部署在 LAMP 和 SQLMVED 架构环境中的以上 3 种靶机应用程序进行测试，测试结果如表 3 所示。LAMP 架构环境对 3 种靶机应用程序的全部 40 个注入点没有任何防御能力，而 SQLMVED 架构环境能够有效防御针对其中 36 个注入点的攻击。其中 bWAPP 2.2 中 SQLite、Stored SQLite、Blind SQLite 注入点无法防御是因为 SQLMVED 基于

表 3　防御有效性测试环境

应用程序	注入点	LAMP 架构	SQLMVED 架构
bWAPP 2.2(Low)	GET/Search	×	√
	POST/Select	×	√
	Login Form/Hero	×	√
	Drupal	×	×
	Stored User-Agent	×	√
	Blind Time-Based	×	√
	GET/Select	×	√
	AJAX/JSON/jQuery	×	√
	Login Form/User	×	√
	Stored Blog	×	√
	Stored XML	×	√
	Blind SQLite	×	×
	POST/Search	×	√
	CAPTCHA	×	√
	SQLite	×	×
	Stored SQLite	×	×
	Blind Boolean-Based	×	√
	Blind Web Services/SOAP	×	√
DVWA 1.9(Low)	sqli	×	√
	sqli_blind	×	√
SQLI-LABS (Less 1~ 20)	Error Based- String(GET)	×	√
	Error Based- DoubleQuotes String(GET)	×	√
	Dump into Outfile(GET)	×	√
	Blind- Time based- Double Quotes- String(GET)	×	√
	Double Injection- String- with twist(POST)	×	√
	Blind- Time Based- Double quotes- String(POST)	×	√
	Header Injection- Referer- Error Based- string(POST)	×	√
	Error Based- Intiger(GET)	×	√
	Double Query- Single Quotes- String(GET)	×	√
	Blind- Boolian- Single Quotes- String(GET)	×	√
	Error Based- String(POST)	×	√
	Double Injection- Double quotes- String(POST)	×	√
	Update Query- Error based - String(POST)	×	√
	Cookie Injection- Error Based- string(POST)	×	√
	Error Based- String with Twist(GET)	×	√
	Double Query- Double Quotes- String(GET)	×	√
	Blind- Time based- Single Quotes-String(GET)	×	√
	Error Based- Double quotes- String(POST)	×	√
	Blind- Boolian Based- String(POST)	×	√
	Header Injection- Error Based- string(POST)	×	√

MySQL 数据库设计，暂且无法应用于其他类型数据库；Drupal 属于应用框架，目前 SQLMVED 还未适用所有框架，因此无法有效防御此注入点。实验表明，SQLMVED 架构能够有效防御针对以上 3 种靶机应用程序的大部分 SQLIA。

4.2 性能测试

SQLMVED 架构环境在变体 E_1、E_2、E_3 中部署具有数据读写操作的测试页面，读写测试语句如表 4 所示，在 LAMP 架构环境中也部署同样的测试页面。

表 4 读写测试语句

类型	测试语句
读数据	SELECT account FROM users WHERE login='test' AND pass='1234';
写数据	UPDATE users SET account=92 WHERE login='test' AND pass='1234';

为减小虚拟化的实验环境对测试结果的影响，以及主要体现读写过程的性能损耗，测试方法采用在连续的 100 次访问中分别记录测试页面的响应时间。

读数据测试结果汇总如图 13 和图 14 所示，SQLMVED 的平均响应时间为 47.48 ms，LAMP 的平均响应时间为 23.88 ms。SQLMVED 架构相较于 LAMP 架构，平均响应时间增加约一倍，大部分响应时间控制在 60 ms 以内，性能损耗可以接受。

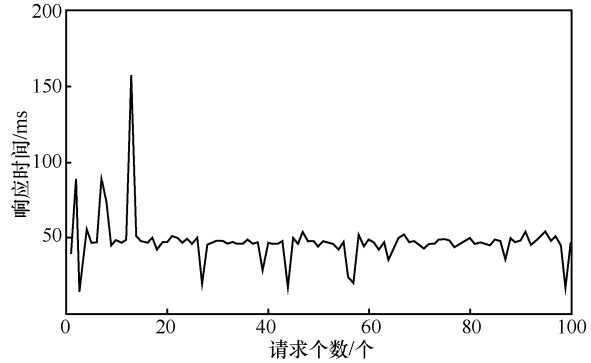


图 13 SQLMVED 读数据测试结果

写数据测试结果汇总如图 15 和图 16 所示，SQLMVED 平均响应时间为 4 870.63 ms，LAMP 的平均响应时间为 20.6 ms。SQLMVED 的写操作产生了较严重的性能损耗，这是由于 SQLMVED 中写数据的过程比读数据更复杂。但是，通过防御有效性测试可以发现，SQLIA 通常发生在读数据过程中，因此，若为减小性能损耗，即使只将读数据采用多变体执行的方法设计，也能够防御大部分 SQLIA。

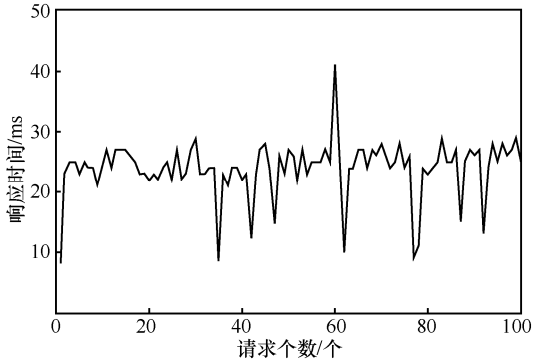


图 14 LAMP 读数据测试结果

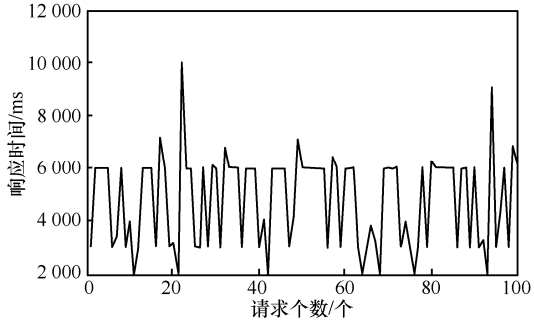


图 15 SQLMVED 写数据测试结果

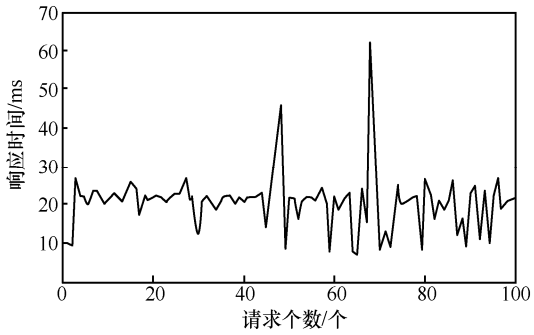


图 16 LAMP 写数据测试结果

此外，SQLMVED 原型系统在设计实现中可能存在缺陷，包括防御有效性测试中发现的不足，SQLMVED 的局限性将在第 5 节进行分析总结。

5 局限性分析

SQLMVED 原型系统的局限性主要体现在以下 2 个方面。

1) 首先，SQLMVED 的多变体中需要对应用程序的 SQL 进行随机化变化，而 SQL 具有跨代码段、跨函数、跨文件等多种拼接方式，如何准确地识别应用程序中的 SQL 并进行随机化变化，可能需要对程序分析技术进行深入研究。其次，本文使用的随机化方法不支持 Hibernate、Mybatis 等开发框架，因此 SQLMVED 的适用范围局限于使用原生 SQL

开发的应用程序。

应用程序以网站为例，根据 W^{3}Techs 对 Alexa 排名前 100 万网站的类型统计结果来看，使用原生 SQL 开发为主的 PHP 网站数量占比达到了 79%，说明 SQLMVED 虽然不能适用于所有的应用程序，但也具备可观的应用前景。

2) 本文在数据库代理处采用高速缓存解决数据库读操作的表决问题。但在高并发数据流的背景下，若把所有请求语句都保存在缓存中进行表决处理，将会大大增加表决过程对存储空间的消耗，本节基于 Bloom filter[28]提出 SQLMVED 的改进思路。

Bloom filter 是一种存储空间高效的散列结构，被广泛应用在 P2P 网络写作、网络缓存以及网络测量等领域。Bloom filter 可以对待表决的语句集合采用位串表示，利用数据流冗余消除方式对语句进行表决，当判定某请求语句是冗余重复的数据，就认为该语句为 3 冗余多变体中的多数一致语句，由数据库代理转发至数据库执行，否则丢弃该语句。数据流冗余消除算法设计的关键问题是需要实时删除达到最大计数值的过期元素，主要算法可以分为 3 类，分别是基准窗口模式、滑动窗口模式、跳跃窗口模式，需要采用何种方式还需要进一步研究。

6　结束语

本文在研究了现有多变体执行技术和 SQL 注入防御方法的基础上，结合两者的技术思路，提出了一种基于多变体执行的 SQL 注入运行时防御方法。首先为应用程序在运行时构建多变体执行架构；其次通过随机化方法对应用程序的 SQL 进行变化，保证变体间随机化方法的异构性；最后在用户请求代理和数据库代理处通过表决分别发现异常的数据读操作结果和数据写请求，使攻击者即使在掌握了变体随机化方法的情况下也无法实施有效的 SQLIA。基于该方法设计实现了原型系统 SQLMVED，并通过形式化推论证明了 SQLMVED 的防御能力。性能测试结果显示，由于该方法引入了请求的复制分发、语句解析、去随机化、表决等处理过程，会降低程序性能，因此如何减小对性能的影响将是未来工作的重点之一。防御有效性测试虽然采用靶机应用程序和自动化注入工具构造的都是已知漏洞攻击，但是该方法也可以有效防御利用未知漏洞的 SQLIA，只要攻击者构造的输入数据中具有 SQL 指令，就无法同时在多变体中正确执行，进而无法通过表决，防御利用未知漏洞的 SQLIA。

参考文献：

[1] BOYD S W, KEROMYTIS A D. SQLrand: preventing SQL injection attacks[C]//International Conference on Applied Cryptography and Network Security. Berlin: Springer, 2004: 292-302.

[2] 马博林, 张铮, 陈源, 等. 基于指令集随机化的抗代码注入攻击方法[J]. 信息安全学报, 2020, 5(4): 30-43.
MA B L, ZHANG Z, CHEN Y, et al. The defense method for code-injection attacks based on instruction set randomization[J]. Journal of Cyber Security, 2020, 5(4): 30-43.

[3] 方滨兴. 定义网络空间安全[J]. 网络与信息安全学报, 2018, 4(1): 1-5.
FANG B X. Define cyberspace security[J]. Chinese Journal of Network and Information Security, 2018, 4(1): 1-5.

[4] SHAR L K, TAN H B K. Defeating SQL injection[J]. Computer, 2013, 46(3): 69-77.

[5] MCCLURE R A, KRUGER I H. SQL DOM: compile time checking of dynamic SQL statements[C]//Proceedings of 27th International Conference on Software Engineering. Piscataway: IEEE Press, 2005: 88-96.

[6] COOK W R, RAI S. Safe query objects: statically typed objects as remotely executable queries[C]//Proceedings of 27th International Conference on Software Engineering. Piscataway: IEEE Press, 2005: 97-106.

[7] KIEYZUN A, GUO P J, JAYARAMAN K, et al. Automatic creation of SQL Injection and cross-site scripting attacks[C]//2009 IEEE 31st International Conference on Software Engineering. Piscataway: IEEE Press, 2009: 199-209.

[8] 孙歆, 姚一杨, 卢新岱, 等. 基于 HTTP 代理的模糊测试技术研究[J]. 网络与信息安全学报, 2016, 2(2): 75-86.
SUN X, YAO Y Y, LU X D, et al. Research and implementation of fuzzing testing based on HTTP proxy[J]. Chinese Journal of Network and Information Security, 2016, 2(2): 75-86.

[9] KAR D, PANIGRAHI S, SUNDARARAJAN S. SQLiGoT: detecting SQL injection attacks using graph of tokens and SVM[J]. Computers & Security, 2016, 60: 206-225.

[10] 韩宸望, 林晖, 黄川. 基于 SQL 语法树的 SQL 注入过滤方法研究[J]. 网络与信息安全学报, 2016, 2(11): 70-77.
HAN C W, LIN H, HUANG C. Research on the SQL injection filtering based on SQL syntax tree[J]. Chinese Journal of Network and Information Security, 2016, 2(11): 70-77.

[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): 88-97.
ZHAO Y F, XIONG G, HE L T, et al. Approach to detecting SQL injection behaviors in network environment[J]. Journal on Communications, 2016, 37(2): 88-97.

[12] APPELT D, PANICHELLA A, BRIAND L. Automatically repairing web application firewalls based on successful SQL injection attacks[C]//2017 IEEE 28th International Symposium on Software Reliability Engineering. Piscataway: IEEE Press, 2017: 339-350.

[13] 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 H L, DING Y, ZHANG L H, et al. SQL injection prevention based on sensitive characters[J]. Journal of Computer Research and Development, 2016, 53(10): 2262-2276.

[14] NGUYEN-TUONG A, GUARNIERI S, GREENE D, et al. Automatically hardening Web applications using precise tainting[C]//IFIP International Information Security Conference. Berlin: Springer, 2005: 295-307.

[15] PIETRASZEK T, BERGHE C V. Defending against injection attacks through context-sensitive string evaluation[C]//International Conference on Recent Advances in Intrusion Detection. Berlin: Springer, 2005: 124-145.

[16] HALFOND W G J, ORSO A. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks[C]//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2005: 174-183.

[17] 何成万, 叶志鹏. 基于 AOP 和动态污点分析的 SQL 注入行为检测方法[J]. 电子学报, 2019, 47(11): 2413-2419.
HE C W, YE Z P. SQL injection behavior detection method based on AOP and dynamic taint analysis[J]. Acta Electronica Sinica, 2019, 47(11): 2413-2419.

[18] HRANICKÝ R, ZOBAL L, RYŠAVÝ O, et al. Distributed password cracking with BOINC and hashcat[J]. Digital Investigation, 2019, 30: 161-172.

[19] KNOWLTON K C. A combination hardware-software debugging system[J]. IEEE Transactions on Computers, 1968, 100(1): 84-86.

[20] COX B, EVANS D, FILIPI A, et al. N-Variant systems: a secretless framework for security through diversity[C]//Proceedings of the 15th conference on USENIX Security Symposium. New York: ACM Press, 2006: 105-120.

[21] BERGER E D, ZORN B G. DieHard: probabilistic memory safety for unsafe languages[C]//ACM SIGPLAN Conference on Programming Language Design & Implementation. New York: ACM Press, 2006: 158-168.

[22] NOVARK G, BERGER E D. DieHarder: securing the heap[C]//Proceedings of the 17th ACM Conference on Computer and Communications Security. New York: ACM Press, 2010: 1-12.

[23] NOVARK G, BERGER E D, ZORN B G. Exterminator: automatically correcting memory errors with high probability[J]. ACM SIGPLAN Notices, 2007, 42(6): 1-11.

[24] 邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016, 1(4): 1-10.
WU J X. Research on cyber mimic defense[J]. Journal of Cyber Security, 2016, 1(4): 1-10.

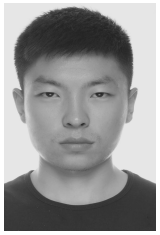
[25] WU J X. Cyberspace mimic defense[M]. Cham: Springer International Publishing, 2020.

[26] 张铮, 马博林, 邬江兴. web 服务器拟态防御原理验证系统测试与分析[J]. 信息安全学报, 2017, 2(1): 13-28.
ZHANG Z, MA B L, WU J X. The test and analysis of prototype of mimic defense in web servers[J]. Journal of Cyber Security, 2017, 2(1): 13-28.

[27] 马博林, 张铮, 刘健雄. 应用于动态异构 web 服务器的相似度求解方法[J]. 计算机工程与设计, 2018, 39(1): 282-287.
MA B L, ZHANG Z, LIU J X. Similarity calculation method applied to dynamic heterogeneous web server system[J]. Computer Engineering and Design, 2018, 39(1): 282-287.

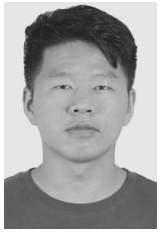
[28] 唐海娜, 林小拉, 韩春静. 基于移动指针的数据流冗余消除算法[J]. 通信学报, 2012, 33(2): 7-14.
TANG H N, LIN X L, HAN C J. Duplicate elimination algorithm for data streams with SKIP Bloom filter[J]. Journal on Communications, 2012, 33(2): 7-14.

[作者简介]



马博林（1993− ），男，河北吴桥人，信息工程大学博士生，主要研究方向为网络空间安全。

张铮（1976− ），男，湖北黄梅人，博士，信息工程大学副教授，主要研究方向为网络空间安全、高性能计算。



刘浩（1997− ），男，安徽阜阳人，网络通信与安全紫金山实验室工程师，主要研究方向为网络空间安全。

邬江兴（1953− ），男，浙江嘉兴人，中国工程院院士，信息工程大学教授，主要研究方向为通信与信息系统、网络空间安全。