DOI:10.11830/ISSN.1000-5013.201606110

动静结合的二阶 SQL 注入漏洞检测技术

李鑫[1,2],张维纬[1,2],郑力新[1,2]

(1. 华侨大学 工学院,福建 泉州 362021;
2. 华侨大学 工业智能化技术与系统福建省高校工程研究中心,福建 泉州 362021)

摘要: 为了有效检测应用中的二阶结构化查询语言(SQL)注入漏洞,提出一种动静结合的检测方法.通过静态分析获取持久存储信息,解决动态分析无法处理的 Web 应用多阶段间逻辑联系问题.通过动态分析获取元数据,解决静态分析无法定位污点信息持久存储位置的问题.通过模糊测试验证疑似漏洞,降低误报率.实验结果表明:该检测方法能够有效检测应用程序中存在的二阶 SQL 注入漏洞;相比于传统静态分析,检测精度高、误报率低;相比于传统动态分析,实现对多阶漏洞的检测,优于已有二阶 SQL 注入漏洞检测技术.

关键词: 漏洞检测;二阶结构化查询语言;静态分析;动态分析;污点分析

中图分类号: TP 393 **文献标志码:** A **文章编号:** 1000-5013(2018)04-0600-06

Vulnerability Detection Using Second-Order SQL Injection Combining Dynamic and Static Analysis

LI Xin[1,2], ZHANG Weiwei[1,2], ZHENG Lixin[1,2]

(1. College of Engineering, Huaqiao University, Quanzhou 362021, China;
2. Universities Engineering Research Center of Fujian Province Industrial Intelligent Technology and Systems, Huaqiao University, Quanzhou 362021, China)

Abstract: In order to detect the vulnerability of the second-order structured query language (SQL) injection in the Web application, a detection method based on static and dynamic analysis is proposed in this paper. By analyzing persistent data stores during static analysis, we track tainted information flow in different orders, which solves the problem that traditional dynamic detection can't relate different orders. By dynamic analysis to obtain mate data, solving the problem that traditional static analysiscan't find persistent data stores. Furthermore, we dynamically verify the suspected vulnerabilities to reduce the false positive by fuzzing. The experimental results show that our approach can effectively detect the second-order SQL injection vulnerability in application. Compared with the traditional static analysis, our approach can find more vulnerabilities with lower false positive and high detection accuracy. Compared with the traditional dynamic analysis, our approach can detect multiple order vulnerabilities. Our detection method is better than the existing methods for the detection of the second-order SQL injection vulnerability.

Keywords: vulnerability detection; second-order structured query language; static analysis; dynamic analysis; taint analysis

收稿日期: 2016-06-02

通信作者: 张维纬(1982-),男,讲师,博士,主要从事信息安全、云计算的研究. E-mail:weiweizh@hqu.edu.cn.

基金项目: 福建省自然科学基金资助项目(2015J05125);福建省科技厅专项资助项目(2013H2002);泉州市科技计划项目(2014Z112);华侨大学研究生科研创新能力培育计划资助项目(1400422005);华侨大学科研基金资助项目(13BS415)

关系型数据库被广泛用于 Web 应用之中，然而它所带来的安全问题一直是威胁 Web 安全的主要因素之一[1]. 二阶结构化查询语言(structured query language，SQL)注入是一种新型 Web 漏洞，同一阶 SQL 注入技术一样，能够威胁客户端、服务器上的数据和系统的安全[2]. 传统检测方法不能有效对其进行检测. 因此，具有极强的隐蔽性，广泛存在于 Web 应用中. 近些年，二阶 SQL 注入逐渐替代传统 SQL 注入技术成为黑客行为的突破口. 因此，对于二阶 SQL 注入漏洞的检测成为了研究的热点[3-5]. Web 漏洞检测技术主要有动态、静态和动静结合 3 种方法. 动态分析的主要技术为模糊测试，如 Swarup 等[6]对通过爬虫找到的可控参数发送大量测试用例，并分析应用的异常检测漏洞. 动态分析虽然实施部署简单，误报率低，但也存在测试效率低、覆盖度不高等问题. 并且这种针对单一注入点进行检测的方式无法有效处理 Web 应用多阶段之间的联系，不能检测出二阶 SQL 注入漏洞. 早期的静态分析技术，如 ITS4[7]只是简单地在源代码中寻找危险函数的调用，误报率非常高，需要大量的人工分析其检测结果. 以 Graudit[8]为代表的通过正则表达式匹配寻找漏洞的技术，虽然一定程度上增加了检测的灵活性，但是依然需要大量人工参与. 基于数据流的污点分析技术，如 Pixy[9]是静态分析检测 Web 漏洞技术成熟的标志. Dashies 等[4]实现了二阶 SQL 注入漏洞检测工具 RIPS，但是在检测二阶 SQL 注入漏洞时，仍然存在 2 个问题：无法准确定位污染数据的中间存储位置和无法判断污染数据到达危险函数前是否经过有效过滤. 静态分析有覆盖面广、效率高的优点，但是误报率和漏报率高，尤其是不能准确检测多阶漏洞. 动静态结合是一种新兴技术，很好地解决了一些在单独使用动态或静态分析时的问题. 潘古兵等[10]提出了一种基于静态分析和动态检测的方法，但是检测方法没有考虑到多阶漏洞的情况，不适用于二阶 SQL 注入漏洞的检测. 闫露[5]提出了一套基于动静态结合分析技术的二阶 SQL 注入漏洞检测系统，但是这种检测方式只适合把完整 SQL 语句写在一条语句中，并且同时拼接了污点信息的情况. 因此，存在过高的漏报率，且需要大量人工分析. 本文提出一种动静结合的检测方法，可以有效检测 Web 应用中的二阶 SQL 注入漏洞.

1　相关技术

文中检测对象二阶 SQL 注入漏洞，是一种兼具污点传播和多阶段的特性的新型漏洞，这也是现有方法不能对其进行有效检测的原因.

1.1　二阶 SQL 注入漏洞

SQL 注入漏洞的应用允许攻击者把 SQL 代码插入到用户的输入参数中，再将这些参数传递给后台数据库服务器解析并执行. 二阶 SQL 注入则把用户输入的 SQL 代码先存储到计算机磁盘中，再间接传递给数据库服务器. 二阶 SQL 注入是污点传播型漏洞，即污点信息从用户输入流向安全敏感函数.

相比于传统 SQL 注入有多阶特性，即在应用程序内部污点信息不会直接流向安全敏感函数，而是流经计算机上的持久存储(persistent data stores，PDS). PDS 是计算机磁盘上的文件，包括数据库、session 和其他格式的文件. 二阶 SQL 注入漏洞的检测难点在于数据库形式的 PDS. SQL 语法允许以显式或隐式的方法(语句是否包含字段名)读写数据库. 如果源程序混合使用了隐式的读取或写入，就会造成数据的存储与读取程序点之间仅有逻辑上而无源代码中明确的数据流联系. 因此，使用静态分析技术检测二阶 SQL 注入漏洞时，污点信息流向持久存储后就会断掉，不能完整地跟踪污点传播的整个过程.

1.2　数据流分析技术

控制流图(control flow graph，CFG)是一种反映应用程序逻辑控制流程的有向有根图[11]，可以表示为

$$G=(N,E,\text{entry},\text{exit}).$$

上式中：N 为应用程序中所有基本块组成的集合；E 为控制流程图中边的集合；entry 和 exit 分别表示控制流程图中唯一的入口和出口节点，表示控制流的开始和结束. 基本块是由顺序执行的指令组成的最长序列，每条边连接两个基本块，代表控制流从一个基本块到另一个基本块的转移.

数据流分析是一种获取沿着应用程序执行路径流动信息的技术[12]. KILDALL[13]提出数据流分析框架，目前已经发展成为数据流分析的核心理论，其理论基础是格理论，应用到数据流分析是半格理论.

定义 1　半格 L 是一个 3 元组，$L=(V,\leqslant,\cup)$. 其中，V 是一个偏序集；$\leqslant$ 是定义在 V 上的一个偏

序关系,满足自反性、反对称性和传递性;$\cup$是定义在V上的一个二元交汇运算,满足等幂、可交换和结合律.半格有一个顶元素,记为T,且$\forall x\in V,\exists T\forall:T\cup x=x$;半格有一个底元素,记为$\perp$,且$\forall x\in V,\exists\perp:\perp\cup x=\perp$.

定义2 数据流分析框架DF是一个3元组,DF$=(D,L,F)$,其中,D是数据流传播方向,分为沿着控制流向前分析和逆着控制流向后分析;L是一个半格;F是在半格上从V到V的传递函数族.

由遍历控制流图可以获得体现程序所有可能执行路径的程序点序列.每一个程序点对应一个传递函数,抽象应用执行当前程序点后,数据流分析中数据流值发生改变.所有程序状态的集合对应半格中的偏序集V.基本块是控制流图的节点,每一个基本块可能有多个前驱和后继基本块.记基本块B的任意前驱和后继分别为B_{pre}和B_{succ},记进入和离开基本块B的数据流值分别为IN$[B]$和OUT$[B]$.在向后的数据流分析中,基本块B的出口数据流值是所有后继基本块入口状态的并,传递函数f_{B}模拟基本块对入口数据流的影响计算出口数据流.由此可以得出一组数据流约束方程为

$$\text{OUT}[B]=\begin{cases}\varnothing, & B=\text{exit},\\ \bigcup \text{IN}[B_{\text{succ}}], & \text{其他},\end{cases}$$
$$\text{IN}[B]=f_{\text{B}}(\text{OUT}[B]). \tag{1}$$

数据流分析就是求所有节点的数据流约束方程组的一组满足约束条件的最优解.

2 检测方法

PHP是目前主要的Web编程语言之一,由于其灵活性,存在较多的漏洞.因此,文中以PHP作为检测语言,但所提出的检测方法也同样适用于其他编程语言.

2.1 系统模型

二阶SQL注入漏洞检测模型,如图1所示.模型由静态分析和动态分析两部分组成.静态分析是检测系统中的主要部分,包括前端生成应用程序源代码的CFG和后端的查找注入点、PDS、触发点,以及通过污点分析定位二阶SQL的注入漏洞,并生成漏洞报告.动态分析起到辅助作用,包括前期信息搜集获取的元数据和后期漏洞的验证.

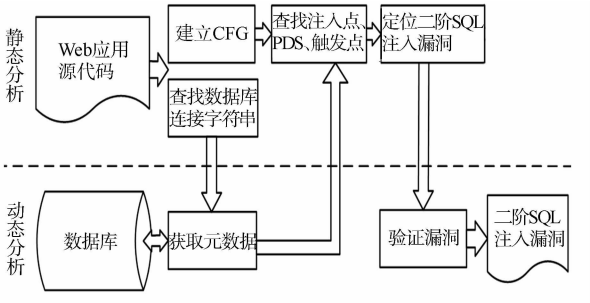

图1 检测系统模型

Fig. 1 Detection system model

2.2 静态分析部分

检测模型主要针对PHP语言,选用开源工具PHP-Parser[14],通过语法分析、语义分析等步骤生成抽象语法树(abstract syntax tree,AST),并且该工具提供一套对AST的遍历方法,方便生成CFG.

在生成CFG的同时,对当前基本块中语句进行抽象处理,并将信息保存到对应的基本块摘要中,该文件中所有基本块的摘要组成当前文件的摘要.抽象处理包括分析赋值语句、引用语句、常量/变量定义、方法、用户自定义函数/方法和系统函数.

如果当前节点是用户自定义函数/方法,则对其进行过程间分析,生成控制流图,并获得函数/方法的摘要.如果函数/方法体中含有安全敏感函数,则对其参数进行向后分析.如果参数值最终来自于当前函数或方法的形参,则将函数/方法及其形参加入到全局的用户自定义安全敏感函数变量中,并在调用函数点处进行污点分析.如果参数值来自于污染源,则向检测结果中添加一条漏洞信息.其他情况视作安全,不做任何处理.对于文件引入操作,如require,include等的分析类似于对函数/方法的过程间分析,只是检测到安全敏感函数.当参值来源于一个未知变量时,在返回的结果中,这个未知变量代替函数/方法中的形参,文件的分析结果保存到全局变量中的文件摘要中.

在此基础上,通过基于数据流的污点分析追踪污点信息从污染源到安全敏感函数的流动,并借助元数据分析PDS补上被截断的数据流.数据流示意图,如图2所示.即在静态分析过程中遇到数据库查询函数(如mysql_query)时,分析SQL查询字符串的类型.如果向数据库中写入数据,且拼接了污点信息,则将当前查询字符串所指向的存储位置(表+字段)存入静态分析系统的PDS变量中,以备下次查

询使用. 如果是从数据库中读取数据，则获取当前查询字符串中数据的存储位置，验证该位置是否在 PDS 中，并将结果存入当前块摘要.

数据流分析框架的半格 L 中值域集合定义为

$$V=\{(\mathrm{var},\mathrm{pos},\mathrm{sta})\mid \mathrm{var}\in \mathrm{Variable},\ \mathrm{pos}\in \mathrm{Line},\mathrm{sta}\in \mathrm{Status}\}.$$

上式中：var 是定义的变量；pos 是变量的位置；sta 是变量的状态信息. 偏序$\leqslant$为$\subseteq$，集合交汇运算的底元素$\perp$为空，顶元素 T 为全集 V. 传递函数 f_{taint} 计算数据流为变量间的传递情况.

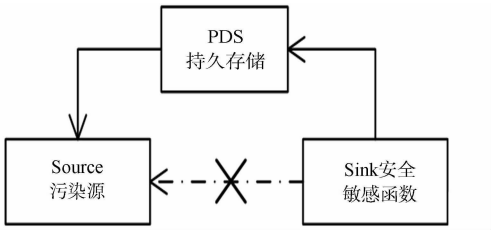

图 2　数据流示意图

Fig. 2　Schematic diagram of data flow

在向后的数据流分析中，入口数据流是基本块中产生的数据流和出口数据流减去基本块中杀死的数据流的并. 因此公式(1)中的入口数据流定义为

$$\mathrm{IN}[B]=f_{\mathrm{taint}}(\mathrm{OUT}[B])=\mathrm{GEN}_{\mathrm{B}}\cup(\mathrm{OUT}[B]-\mathrm{KILL}_{\mathrm{B}}).\tag{2}$$

对变量的赋值操作称为变量的定值，表现为变量出现在等号的左边，而出现在等号右边的变量则称为对该变量的使用. 在活跃变量分析中，对变量的定值相当于杀死该变量对应的数据流信息，记基本块 B 中所有被定值的变量集合为 $\mathrm{DEF}_{\mathrm{B}}$. 变量的使用相当于产生新的数据流信息，记基本块 B 中所有被使用的变量集合为 USE_{B}. 通过变量的使用和定值计算得到更精确的公式，式(2)重新定义为

$$\mathrm{IN}[B]=f_{\mathrm{taint}}(\mathrm{OUT}[B])=\mathrm{GEN}_{\mathrm{B}}\cup(\mathrm{OUT}[B]-\mathrm{DEF}_{\mathrm{B}}).\tag{3}$$

检测到危险函数调用时，开始通过迭代的方式进行活跃变量数据流分析. 以当前基本块作为分析开始的出口基本块，安全敏感函数中的危险参数信息作为出口数据流信息. 分析沿着控制流向后进行，当分析到入口基本块数据流值为空时，停止并返回安全；当数据流向后追踪到用户输入时，返回漏洞信息. 基于半格数据流分析框架的迭代算法如下.

```
TaintAnalyse ( currentBlock B,OUT[B] ){
    if B is not start block
        OUT[B]=∪IN[Bsucc];
    IN[B]=USE_B∪ (OUT[B] − DEF_B)
    if B ! =entry && IN[B]! =∅&& ! ( IN[B] exist in Source ){
        for each block Bsucc
            TaintAnalyse(Bsucc);
    }
    else{
        if IN[B] exist in Source
            return tainted;
        return safe;
    }
}
```

2.3　动态分析

动态分析在检测模型中主要起辅助功能，包括在模型初始化阶段查询数据库获取元数据(描述数据的数据)和通过模糊测试验证静态分析结果两部分.

1) 与数据库通信获取元数据. 每一种数据库都有一个单独的数据库存储元数据信息，例如，在 MSSQL 中的 master 数据库、MySQL 中的 information_schema 数据库. 文中模型用到的元数据是应用程序数据库的所有表和字段信息. 通过获得数据库连接后向数据库服务器查询获得. 即 SELECT table_name,column_name,column_type FROM information_schema. columns WHERE table_schema= 'empirecms'.

返回的结果集中包括了每一个表的所有字段. 通过遍历，把该结果集转化为二维数组，存入全局变量 metadata 中，在静态分析时使用.

2) 漏洞的验证. 动态验证流程,如图3所示. 漏洞的验证指的是在检测流程的最后一步对静态分析的结果进行验证,主要使用模糊测试技术. 首先,读取分析结果,获得注入点与触发点. 漏洞的注入点选择用户输入文件,其作为HTTP请求中的地址,并依据用户输入的形式判定其存在于HTTP请求中的位置. 漏洞触发点选择安全敏感函数文件作为地址. 然后,在注入点发送合法请求,并获得正常响应. 注入测试用例到注入点,通过对比分析此时应用程序的响应与合法请求的响应,验证测试位置是否存在二阶SQL注入漏洞. 测试用例涵盖非法/逻辑错误查询、联合查询、附带查询、重言式、SQL盲注.

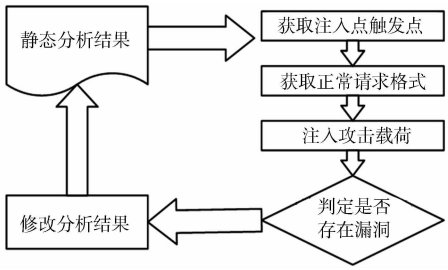

图3 动态验证流程图

Fig. 3 Dynamic verification flow chart

3 实验与分析

为验证文中模型对完整Web应用中漏洞检测效果,选择phpvulhunter、文献[5]中工具、RIPS[4]的最新版本作为对比的工具,选择Schoolmate,Webchess,phpyun v4.0三个开源应用作为检测样本. 在选择对比工具时,优先选择开源能够检测PHP语言中二阶SQL注入漏洞的工具. 在目前已知的检测工具中,只有文献[5]中工具和RIPS两个能够检测二阶SQL注入漏洞. 但是由于前者不对外开放,因此,为了与其进行对比,实验样本中包括了文献[5]中的两个开源应用. 工具phpvulhunter代表传统静态分析技术. 最后一个检测样本是国内流行的开源PHP应用phpyun,代表了通用的编程风格. 对文中模型工具对比实验结果,如表1所示.

表1 工具对比实验

Tab. 1 Comparison between tools

检测样本	phpvulhunter			文献[5]			RIPS			文中模型		
	TPDS	FP	TP	TPDS	FP	TP	TPDS	FP	TP	TPDS	FP	TP
SchoolMate	0	0	0	—	—	1	44	22	4	24	0	6
WebChess	0	0	0	—	—	0	21	5	1	8	0	1
phpyun	0	0	0	—	—	—	0	2	0	4	0	2

表1中:TPDS(taint persistent data stores)表示受污染的持久存储;FP(false positive)表示误报;TP(true positive)表示真实漏洞;数据已剔除重复报告.

由表1可知:工具phpvulhunter没有检测到3个样本中的二阶SQL注入漏洞,说明传统静态分析方法由于设计时的局限性,不能检测二阶SQL注入漏洞;文献[5]中仅检测出了SchoolMate中一个二阶SQL注入漏洞;RIPS检测4个二阶SQL注入漏洞;文中模型检测6个二阶SQL注入漏洞;文中模型对二阶SQL注入漏洞有着优于现有工具的检测效果.

TPDS连接二阶SQL注入漏洞的注入点和触发点,是检测的关键位置. 工具RIPS检测出了大量TPDS,但是误报率非常高. 因为在静态分析时,由于没有获取元数据,而不知道字段类型,把一些安全的字段也视作可以被污染. 例如,在对SchoolMate和WebChess两个样本的检测过程中,RIPS把大量作为主键的数字类型字段标记成了TPDS. 文中模型则通过引入动态分析获取元数据,判断字段是否可被污染,对TPDS的标记更加准确.

在误报率方面,phpvulhunter没有检测出漏洞,不存在误报率. 文献[5]中工具没有公布数据,RIPS相较于文中模型存在较高的误报率. RIPS是一种纯静态的检测工具,因此,对TPDS的错误标记会严重影响最终结果的准确性. 文中模型一方面可以获得相对精准的TPDS,另一方面采用动态验证静态分析结果,降低了误报率.

对phpyun应用的检测中,RIPS由于采用了只要数据来源代码中含有污染源就报告漏洞的方法,因此,在没有检测到TPDS的情况下,误报2个二阶SQL注入漏洞,并发现了12个安全敏感函数的PDS. 而对于应用中过滤条件if(is_numeric($v)),由于无法获知数据库字段类型,直接当作安全处理. 如果PDS的字段类型为字符串形式,以16进制形式表示的攻击载荷能够绕过该过滤条件,并在数据库中被自动还原为对应的字符,污染当前PDS. 文中模型由于能够通过动态获取字段类型标记被污染字

段，进而发现漏洞.

4　结束语

通过动静结合的方式检测二阶 SQL 注入漏洞，弥补了传统单独使用静态、动态分析时各自的不足.通过实验证明：文中模型有着优于现有技术的检测效果；虽然针对 PHP 语言中的二阶 SQL 注入漏洞设计，但是同样适用于其他编程语言的二阶污点传播型漏洞.下一步将进一步完善检测模型对 PHP 语法特性和动态验证中对 cookie 和 SESSION 的支持，完善测试用例，并可适当增加以编码、大小写转换等方式绕过输入验证的功能.

参考文献：

[1] OWASP F. OWASP top ten project[EB/OL]. [2016-05-23]. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[2] 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(增刊 1)：85-93. DOI：10.11959/j.issn.1000-436x.2015285.

[3] AMIT Y, BESKROVNY E, TRIPP O. Detection of second order vulnerabilities in web services: US, 20130167237A1[P]. 2013-06-27.

[4] DAHSE J，HOLZ T. Static detection of second-order vulnerabilities in web applications[C]//23rd USENIX Security Symposium. San Diego：USENIX，2014：989-1003.

[5] 闫璐. Web 应用二阶 SQL 注入漏洞检测方法研究[D]. 天津：天津大学，2014：1-39.

[6] SWARUP S，KAPOOR R K. Web vulnerability scanner (WVS)：A tool for detecting web application vulnerabilities [J]. International Journal of Engineering Research，2014，3(2)：126-131. DOI：10.17950/ijer/v3s2/219.

[7] VIEGA J，BLOCH J T，KOHNO Y，*et al*. ITS4：A static vulnerability scanner for C and C++ code[C]//16th Annual Conference of Computer Security Applications. New York：IEEE Press，2000：257-267.

[8] 克拉克. SQL 注入攻击与防御[M]. 北京：清华大学出版社，2013：100-101.

[9] JOVANOVIC N，KRYEGEL C，KIRDA E. Pixy：A static analysis tool for detecting web application vulnerabilities [C]//IEEE Symposium on Security and Privacy. New York：IEEE Press，2006：258-263. DOI：10.1109/SP.2006.29.

[10] 潘古兵，周彦晖. 基于静态分析和动态检测的 XSS 漏洞发现[J]. 计算机科学，2012，39(B6)：51-53.

[11] BRAVENBOER M，SMARAGDAKIS Y. Strictly declarative specification of sophisticated points-to analyses[J]. ACM SIGPLAN Notices，2009，44(10)：243-262. DOI：10.1145/1640089.1640108.

[12] 吴世忠，郭涛，董国伟，等. 软件漏洞分析技术[M]. 北京：科学出版社，2014：79-115.

[13] KILDALL G A. A unified approach to global program optimization[C]//Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York：ACM，1973：194-206. DOI：10.1145/512927.512945.

[14] 林姗，郑朝霞. 基于格的数据流分析研究与应用[J]. 武汉理工大学学报(信息与管理工程版)，2011，33(6)：932-935. DOI：10.3963/j.issn.1007-144X.2011.06.021

(责任编辑：陈志贤　　**英文审校：**吴逢铁)