计算机科学
COMPUTER SCIENCE

基于有限状态机的内核漏洞攻击自动化分析技术

刘培文，舒辉，吕小少，赵耘田

引用本文

刘培文，舒辉，吕小少，赵耘田. 基于有限状态机的内核漏洞攻击自动化分析技术[J]. 计算机科学, 2022, 49(11): 326-334.

LIU Pei-wen, SHU Hui, LYU Xiao-shao, ZHAO Yun-tian. Automatic Analysis Technology of Kernel Vulnerability Attack Based on Finite State Machine[J]. Computer Science, 2022, 49(11): 326-334.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

ROP 漏洞利用脚本的语义还原和自动化移植方法

Semantic Restoration and Automatic Transplant for ROP Exploit Script

计算机科学, 2022, 49(11): 49-54. https://doi.org/10.11896/jsjkx.210900230

基于符号执行的 Return-to-dl-resolve 利用代码自动生成方法

Automatic Return-to-dl-resolve Exploit Generation Method Based on Symbolic Execution

计算机科学, 2019, 46(2): 127-132. https://doi.org/10.11896/j.issn.1002-137X.2019.02.020

基于代码覆盖的浏览器漏洞利用攻击检测方法

Web Browser Vulnerability Exploitation Attack Test Technology Based on Code Overriding

计算机科学, 2011, 38(Z10): 41-43.

动态信息流分析的漏洞利用检测系统

Dynamic Information Flow Analysis for Vulnerability Exploits Detection

计算机科学, 2010, 37(7): 148-151.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211200039

基于有限状态机的内核漏洞攻击自动化分析技术

刘培文[1]　舒　辉[2]　吕小少[2]　赵耘田[2]
1 郑州大学网络空间安全学院　郑州 450001
2 信息工程大学数学工程与先进计算国家重点实验室　郑州 450001
(2386107892@qq.com)

摘　要　内核漏洞攻击是针对操作系统常用的攻击手段,对各攻击阶段进行分析是抵御该类攻击的关键。由于内核漏洞类型、触发路径、利用模式的复杂多样,内核漏洞攻击过程的分析难度较大,而且现有的分析工作主要以污点分析等正向程序分析方法为主,效率较低。为了提高分析效率,文中实现了一种基于有限状态机的内核漏洞攻击自动化分析技术。首先,构建了内核漏洞攻击状态转移图,作为分析的关键基础;其次,引入反向分析的思路,建立了基于有限状态机的内核漏洞攻击过程反向分析模型,能够减小不必要的分析开销;最后,基于模型实现了一种内核漏洞攻击反向分析方法,能够自动、快速地解析内核漏洞攻击流程。通过对10个攻击实例进行测试,结果表明,反向分析方法能够准确得到关键代码执行信息,且相比传统正向分析方法,分析效率有较大提高。

关键词:内核漏洞;漏洞利用;提权攻击;反向分析;漏洞触发点定位
中图法分类号　TP393

Automatic Analysis Technology of Kernel Vulnerability Attack Based on Finite State Machine

LIU Pei-wen[1],SHU Hui[2],LYU Xiao-shao[2] and ZHAO Yun-tian[2]
1 School of Cyber Science and Engineering,Zhengzhou University,Zhengzhou 450001,China
2 State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract　Kernel vulnerability attack is a common attack way for operating systems,and the analysis of each attack stage is the key to defend against such attacks. Due to the complexity and variety of kernel vulnerability types,trigger paths,and exploit modes,it is difficult to analyze the attack process of kernel vulnerability. Moreover,the existing analysis work mainly focuses on forward program analysis methods such as taint analysis,and the efficiency is low. In order to improve the analysis efficiency,this thesis implements an automatic analysis technology of kernel vulnerability attack based on finite state machine. Firstly,the state transition diagram of kernel vulnerability attack is constructed as the key basis for analysis. Secondly,the idea of reverse analysis is introduced,and a reverse analysis model of kernel vulnerability attack process based on finite state machine is established,which can reduce the unnecessary analysis cost. Finally,based on the model,a reverse analysis method of kernel vulnerability attack is implemented,which can automatically and quickly analyze the kernel vulnerability attack process. By testing 10 attack samples,the results show that the reverse analysis method can accurately obtain the key code execution information,and compared with the traditional forward analysis method,the analysis efficiency is greatly improved.

Keywords　Kernel vulnerability,Vulnerability exploit,Privilege escalation attack,Reverse analysis,Vulnerability trigger point positioning

1　引言

内核漏洞一般潜藏于复杂的内核代码中,而内核代码工作在最高特权级,因此利用内核漏洞进行攻击能够提升本地权限,并绕过应用层的安全防护,从而对主机的安全造成严重威胁。当前沙盒保护技术能够有效地对IE浏览器、微软Office办公软件等客户端程序进行保护,但黑客组织使用Windows内核漏洞配合客户端程序漏洞,可以穿透沙盒保护,进一步实施隐匿性攻击。例如,黑客组织APT28利用Windows内核漏洞CVE-2017-0263和EPS类型混淆漏洞CVE-2017-0262进行组合攻击,突破了Office办公软件的沙盒保护,严重影响了法国大选等事件。同样地,黑客组织SandCat利用Windows内核漏洞CVE-2018-8611绕过主流Web浏览器的沙盒保护,并针对中东地区用户进行小范围的针对性

到稿日期:2021-12-03　返修日期:2022-04-22
基金项目:国家重点研发计划(2019QY1305)
This work was supported by the National Key R&D Program of China(2019QY1305).
通信作者:舒辉(shuhui123@126.com)

攻击，造成了一定影响[1]。

由于内核漏洞攻击涉及深层次的操作系统内核代码，内核漏洞攻击程序相比其他恶意程序更加复杂，主要体现在内核漏洞类型、触发路径以及利用模式的复杂多样。这给内核漏洞攻击程序的逆向分析工作带来了较大困难。

内核漏洞攻击过程存在多个攻击阶段，包括漏洞触发阶段、漏洞利用阶段等。而内核漏洞攻击状态[2]是从内核漏洞攻击的角度，重新解释各阶段下存在的程序执行状态，如在漏洞利用阶段，程序执行状态主要是利用状态。内核漏洞攻击状态转移则指攻击状态发生变化的过程。

内核漏洞攻击过程分析的关键在于解析重要中间攻击状态以及状态转移过程，实现对攻击过程重要信息的提取，从而了解程序的攻击流程。但在分析过程中，一些无用状态会降低分析的效率。例如对于漏洞触发点的定位，为了保证分析结果的准确全面，必须考虑不同漏洞类型的触发特征，从而导致在触发阶段存在多个中间状态。

现有的研究工作，如基于虚拟机的动态分析技术、粗粒度的动态污点分析技术等，尝试利用程序分析方法提高自动化分析的能力，但不能消除无用状态，因此仍造成了不必要的开销，效率较低。

本文实现了一种基于有限状态机的内核漏洞攻击自动化分析技术。其中，基于有限状态机的内核漏洞攻击过程反向分析模型，可以减少无用状态；而基于此模型实现的反向分析方法，可以自动、快速地获取关键代码执行信息，从而帮助安全人员快速了解内核漏洞攻击流程。

本文的主要贡献包括以下 3 个方面：

(1)构建了内核漏洞攻击状态转移图。基于内核漏洞攻击原理，总结划分了触发、利用、提权 3 个主要阶段的各种攻击模式，并构建了攻击状态转移图。攻击图是内核漏洞攻击过程分析的关键基础。

(2)建立了基于有限状态机的内核漏洞攻击过程反向分析模型。为了解决正向分析效率较低的问题，引入了反向分析的思路，逆向解析攻击过程，定义了基于反向分析的有限状态机，并用状态节点形式化描述了 3 个重要状态转移过程，从而建立了反向分析模型。基于模型可以实现一种粗粒度的反向分析方法，减小无用状态的分析开销，自动解析内核漏洞攻击流程。

(3)实现了内核漏洞攻击反向分析方法。在反向分析模型的指导下，借助全系统动态分析平台，详细设计了反向分析方法的各步骤，实现了漏洞触发点、敏感系统调用函数、提权代码执行点等关键代码执行信息的自动快速获取。通过对 10 个攻击实例进行测试，验证了该方法能够准确得到内核漏洞攻击过程的关键代码执行信息，且相比一般细粒度正向污点分析方法具有更小的开销。

本文的研究为内核漏洞攻击过程的分析工作提供了新的途径，有助于自动、快速地解析内核漏洞攻击流程。

2 相关工作

利用内核漏洞进行攻击，本身就增加了攻击程序的复杂性。内核漏洞的类型多样，包括内核池溢出、栈溢出、整型溢出、空指针解引用等。而为了触发内核漏洞函数，一般需要执行大量和漏洞直接触发无关的内核代码，导致漏洞触发路径更加隐蔽。同时，内核漏洞的利用技术也在逐渐变得复杂，从传统的控制流劫持攻击到复杂的非控制数据攻击[3]，给安全防御机制带来了更大的冲击。DOP[4]是一种非控制数据攻击技术，可以绕过针对控制流异常的防御机制，如细粒度控制流完整性机制(Control-Flow Integrity，CFI)[5]。复杂多样的内核漏洞类型、触发路径、利用模式，使得内核漏洞攻击的分析工作面临巨大的挑战。

为了减少人工分析的工作量，采用程序分析技术实现自动快速的内核漏洞攻击过程分析，是分析工作的主要研究方向。

内核漏洞攻击程序的静态分析不能直接对内核代码进行分析，因此无法得到关键代码执行信息。基于虚拟机的动态分析技术[6]通过软件对系统资源进行虚拟抽象，能够准确捕获内核运行信息，常用的平台有 Digtool[7]，Bochspwn[8]，PANDA[9]等。基于已有研究，可使用一些低开销的程序分析方法，来提高分析效率。控制流分析对当前存在的非控制数据攻击不起作用，因此需要使用动态污点分析技术来应对。但由于内核漏洞攻击过程涉及大量的内核函数调用，传统动态污点分析技术会面临许多问题，如资源消耗过大、污点传播路径爆炸等，需要对相应技术进行改进。StraightTaint[10]实现了一种离线污点分析框架，在保证一定分析能力的基础上，完全解耦程序执行和污点分析，从而不需要与程序进行频繁的数据交互，具有更低的分析开销。OFFDTAN[11]实现了一种新型的离线动态污点分析框架，优化了污点传播策略，并构建了污点传播流图来回溯定位可疑数据，具有较好的性能。

大多数研究改进了现有程序分析方法，并通过牺牲精度来进行粗粒度的程序跟踪分析，从而提高了分析效率。但正向分析方法不能减少无用状态，仍需要花费额外资源来跟踪一些与攻击完全无关的执行流程，因此提高分析效率的能力有限。

反向分析技术一般用于根据程序崩溃点信息，反向定位造成崩溃的根源代码 Root Cause。在内核漏洞攻击中，Root Cause 可以指漏洞触发点。其通过自动逆向执行以及后向污点分析，重构导致程序崩溃的数据流来进行反向分析，从而定位 Root Cause。为了具体实现这个过程，CREDAL[12]结合程序源码信息分析程序崩溃转储点，能够有效辅助测试人员寻找 Root Cause。POMP[13]使用后向污点分析重构数据流，并结合控制流信息和程序崩溃转储信息，来识别出与程序崩溃有关的代码。RETRACER[14]能够对内核代码和数据进行后向污点分析，进而可以得到内核崩溃相关函数信息。虽然上述研究能够自动辅助定位 RootCause，但逆向执行以及后向污点分析存在较大开销，可以通过缩小定位范围来减小反向分析的开销。Zheng 等[15]构建了一个基于遗传算法的高度灵活框架 FSMFL，其能够定位软件的多个 Root Cause。Jiang 等[16]提出了一种基于路径分析和信息熵的 Root Cause 定位方法 FLPI，分析了所有执行路径中的数据依赖关系，提高了定位精度；并利用信息熵理论筛选可疑语句，提高了定位效率。DEEPVSA[17]使用深度学习的方法，改进了值集分析

工具(VSA),可以显著提高分析内存别名的能力,从而提高测试人员查明软件 Root Cause 的能力。AURORA[18]通过对比崩溃输入和不崩溃输入的程序运行信息来缩小定位范围,从而更快地自动定位程序的 RootCause。ARCUS[19]通过结合启发式算法,能够更有效地根据源码信息反向定位 Root Cause。

大多数反向分析的研究聚焦于定位造成程序崩溃的 Root Cause。目前暂没有利用反向分析相关技术,来有效定位内核漏洞攻击中漏洞触发点的研究。

本文实现了一种粗粒度的反向分析方法,能够实现内核漏洞攻击过程的自动分析。类似于从程序崩溃点定位 Root Cause,本文方法利用反向分析技术来定位漏洞触发点,再利用控制流信息回溯得到用户态系统调用的相关函数,从而实现内核漏洞攻击过程的反向分析。相比已有的正向分析方法,本文方法能够减小无用状态的分析开销,自动解析内核漏洞攻击流程。

3 内核漏洞攻击状态转移图

内核漏洞攻击状态转移图作为分析工作的关键基础,需要考虑不同攻击阶段存在的各种攻击模式。

整个内核漏洞攻击过程主要有 3 个阶段:漏洞触发、漏洞利用、系统提权。在漏洞触发阶段,通过调用敏感函数,将精心构造的用户态数据传入内核,从而引发内核漏洞函数的执行,并造成一定的破坏效果;在漏洞利用阶段,进一步增强漏洞触发的破坏效果,实现控制流劫持或者关键内存数据的修改;在系统提权阶段,篡改当前进程权限信息,最终实现提权。

本文主要研究内存破坏型内核漏洞攻击程序,这类攻击程序一般利用内核漏洞有目的性地破坏内存数据,进而实现提权。同时,本文的研究工作不涉及漏洞触发细节,因此仅以漏洞类型为漏洞触发阶段的攻击模式划分依据,主要包括 UAF、空指针解引用、堆栈溢出等。下面对漏洞利用和系统提权两个阶段划分攻击模式,以构建内核漏洞攻击状态转移图。

3.1 漏洞利用阶段的攻击模式

为了实现提权,需要对内核漏洞破坏的内存数据进行进一步利用。内核漏洞可修改的内存数据包括控制数据和非控制数据,而不同内核漏洞对数据的利用过程存在差异,一般划分为以下几种常见的利用模式,如图 1 所示。

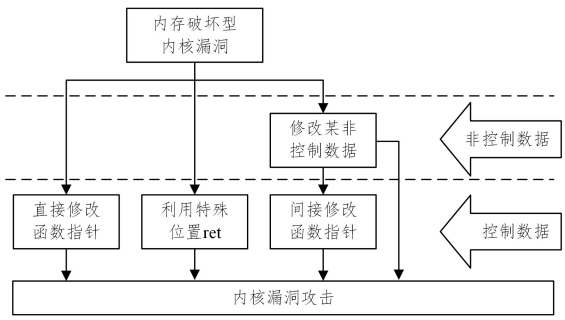

图 1 内存破坏漏洞利用模式

Fig. 1 Memory corruption vulnerability exploitation mode

(1)修改一个或者多个非控制数据,实现攻击目的。

(2)修改控制数据实现控制流劫持,有以下 3 种形式:1)在不破坏函数指针的情况下,需要修改特殊内存位置的数据,如栈中存储的返回地址;2)在破坏函数指针的情况下,直接修改某一对象的函数指针成员域,或者内核函数表中的函数指针;3) 在破坏函数指针的情况下,修改一个或者多个非控制数据后,再间接修改函数指针。

内存破坏型内核漏洞的利用模式可以等价于对控制数据或非控制数据的操作。而对于存在多个利用数据的情况,需要考虑利用数据之间的依赖关系。

多级利用的常见形式是利用内存破坏漏洞,破坏特殊对象指针成员域,使得原本封闭的内核内存结构可被用户态操控,进一步可以利用特殊对象的相关读写 API 对可控内存地址的数据域进行多次操作。目前被公开利用的特殊对象有 Bitmap, Palette, tagWnd 等,这些对象的相同点是将结构存储在内核空间,且能够用读写 API 函数修改自身的数据区域。

经过研究,这类多级利用存在数据依赖 DE,即每一级的利用数据,将直接或间接影响下一级利用数据的内存地址,而这些利用数据来源于用户态。因此,在分析过程中,对于任意利用数据,除了关注其实际的数据值,还要关注其内存地址是否被刻意构造。

3.2 系统提权阶段的攻击模式

利用阶段结束之后,攻击程序已经具备了控制流转移或者任意内存写的能力。Windows 权限访问机制规定,在一个继承了当前用户权限信息 TOKEN 的进程访问某一资源时,需要根据该资源访问控制列表(Access Control List, ACL)中的访问控制项目(Access Control Entry, ACE),来判断该进程是否有访问权[20]。通过有目的性地修改 TOKEN 或者 ACE,能够实现提权攻击。但完整性机制规定,在修改 ACE 之前,必须先修改 TOKEN 中有关完整性机制的成员域内容。因此,判定程序存在提权行为,需要监控 TOKEN 结构相关内存数据的异常修改,如指向 TOKEN 结构的指针被替换、TOKEN 结构关键成员域置位等。

结合当前攻击程序的能力,并考虑 Supervisor 模式执行保护(Supervisor Mode Execution Protection, SMEP),划分了如下 4 种提权模式,如图 2 所示。

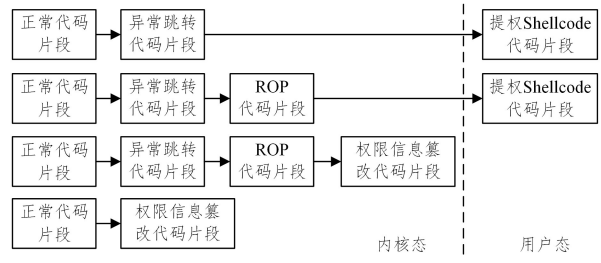

图 2 系统提权模式

Fig. 2 System escalation of privilege mode

(1)已具备内核态执行流转移能力,转移至用户态提权 Shellcode。

(2)已具备内核态执行流转移能力,先执行能够关闭 SMEP 的 ROP 代码片段,再转移至用户态提权 Shellcode。

(3)已具备内核态执行流转移能力,转移至 ROP 代码片段,直接利用多个内核模块代码片段实现提权。

(4)已具备任意内存读写能力,直接篡改 TOKEN 结构

相关的内存数据。

总结上述3个主要阶段的攻击模式，构建出如图3所示的内核漏洞攻击状态转移图。攻击图是内核漏洞攻击过程分析的关键基础。可以看出，图中存在一些无用状态，其对内核漏洞攻击过程分析没有帮助，如无关的初始状态以及漏洞触发中间状态。而大多数正向分析过程为了保证分析的准确、全面，不能忽略这些状态，导致不能简化攻击图，分析的开销也无法明显减小。

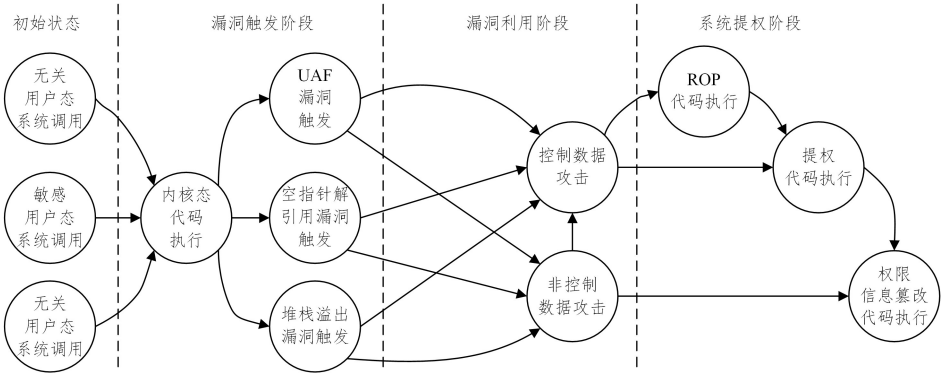

图3 内核漏洞攻击状态转移图

Fig. 3 Kernel vulnerability attack state transition diagram

4 基于有限状态机的内核漏洞攻击过程反向分析模型

本节引入反向分析的思路，重新解释内核漏洞攻击过程。首先，从内核漏洞攻击中的提权点开始，经过反向分析过程，定位漏洞触发点；然后，从漏洞触发点回溯控制流信息，确定用户态系统调用函数；最终，实现内核漏洞攻击过程的反向分析。

4.1 基于反向分析的有限状态机

定义1 基于反向分析过程的有限状态机 $M=\{Q,\Sigma,\delta,q_0,E\}$。其中，$Q=\{q_0,q_1,q_2,q_3\}$为状态集合，各元素分别代表如下含义：$q_0$为提权状态，$q_1$为利用状态，$q_2$为触发状态，$q_3$为用户态构造状态。$\Sigma=\{P_0,P_1,P_2,P_3\}$为输入字母表，各元素分别代表如下含义：$P_0$为提权数据，$P_1$为利用数据，$P_2$为触发数据，$P_3$为其他数据。$\delta$为状态转移函数，并定义如下内容：$\delta(q_0,P_0)=q_1$，$\delta(q_1,P_1)=q_2$，$\delta(q_2,P_2)=q_3$。$q_0$为初始状态，$E$为终止状态（即状态$q_3$）。

同时，可以得到如图4所示的反向分析状态转移图。

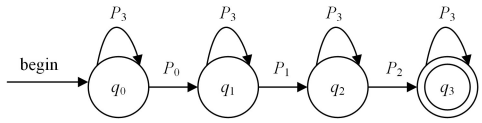

图4 反向分析状态转移图

Fig. 4 Reverse analysis of state transition diagram

4.2 状态节点

为了详细描述有限状态机M的3个状态转移过程，下面引入状态节点以及状态转移节点的相关概念。

定义2 状态节点为Nq，可表示某一状态下的指令信息，状态节点的五元组元素如表1所列。

表1 状态节点Nq的五元组元素

Table 1 Quintuple element of state node Nq

元素名称	元素符号表示	描述
时间信息	$COUNT(Nq)$	表示基本块计数，间接表示程序运行进度
指令信息	$INS\_addr(Nq)$	表示指令地址
	$INS\_behavior(Nq)$	表示指令行为，可取值$\{read,write,unknow\}$
	$INS\_bb(Nq)$	表示指令所属基本块
数据信息	$DATA\_addr(Nq)$	表示指令操作数据的地址
	$DATA\_value(Nq)$	表示指令操作数据的值
数据映射	$SrctoDes\_addr1(Nq)$	表示数据来源地址
	$SrctoDes\_addr2(Nq)$	表示数据现存地址
节点状态信息	$STATE(Nq)$	表示当前节点的状态，可取值$\{eop,exp,trigger,user\}$

定义3 状态转移节点为Nq_t，状态转移节点是特殊的状态节点，表示状态发生转移的关键临界节点。同理，状态转移节点存在与状态节点相同的五元组元素。

在反向分析状态机中，状态用来反映内核漏洞攻击各阶段的执行代码段，状态节点用于反映攻击阶段中的指令信息，而状态转移节点用于反映攻击阶段发生变化的关键指令信息。

状态节点Nq的数据映射元素，不能直接通过分析当前执行指令的上下文得到，因此进一步引入如下定义。

定义4 辅助节点集为$n=\{n_i \mid i=0,\cdots,m\}$，可表示读写同一数据的节点集合，用于辅助得到状态节点的数据映射元素。n_i有与状态节点类似的元素，但不存在数据映射和节点状态信息两个元素。

定义5 内存读写节点流$Seq=\langle n_0,n_1,\cdots,n_m\rangle$，表示$Seq$是一个由辅助节点$n$组成的有序序列。

同理，对于状态节点Nq的数据信息元素，也存在相应的内存读写流Seq，并规定如下准则$GetSeq$：对于任意的Nq，存在$Seq=\langle n_0,n_1,\cdots,n_m\rangle$，且$DATA_value(n_i)=DATA_value$

(Nq), $COUNT(n_m)=COUNT(Nq)$。

基于上述内容，设计了数据溯源算法，具体过程如算法 1 所示。该算法的主要思想是：依据某一状态节点的数据信息、基本块计数，得到并解析相应的内存读写流，最终得到节点较为完整的信息。

算法 1 数据溯源算法 tracksource

输入：Nq/*Nq 只有数据信息和时间信息*/
变量：X1，X2，Y1，Y2，Seq/*临时数据*/
输出：Nq/*Nq 有除节点状态外的完整信息*/
/*利用准则 GetSep，得到内存读写流 Seq*/
1. X1=DATA_value(Nq)，X2=COUNT(Nq)
2. Seq=GetSeq(X1,X2)
/*反向遍历 Seq，解析内存读写流，最终 Y1 存储数据来源节点，Y2 存储数据现存节点*/
3. for each Seq i=m:0
/*情况 1：当前节点为 write 操作*/
4. if INS_behavior(n_i)==write
/*赋初值：若当前节点是 Seq 中的第一个 write 节点，补全 Nq 信息，并开始求解数据映射*/
5. if first_write(n_i)
6. Nq=n_i
7. Y2=n_i，Y1=n_i
8. continue
9. end if
/*情况 1 的第一种判断：两个连续的写操作节点不存在数据的传播，忽略 write->write 这种情况*/
10. if INS_behavior(Y1)==write
11. continue
12. end if
/*情况 1 的第二种判断：write->read 的情况表示基本块间的值传递，共用同一内存*/
13. if INS_behavior(Y1)==read&&DATA_addr(Y1)==DATA_addr(n_i)&&DATA_value(Y1)==DATA_value(n_i)
14. Y1=n_i/*Y1 赋值为当前节点*/
15. end if
/*情况 2：当前节点为 read 操作*/
16. else if INS_behavior(n_i)==read
/*情况 2 的第一种判断：还没有找到首个 write 节点，或是忽略了两个连续的读操作节点 read->read 的情况*/
17. if Y1==NULL or Y2=NULL or INS_behavior(Y1)==read
18. continue
19. end if
/*情况 2 的第二种判断：read->write 的情况表示基本块内的值传递，数据在不同内存间转移*/
20. if INS_behavior(Y1)==write&&INS_bb(Y1)==INS_bb(n_i)&&DATA_value(Y1)==DATA_value(n_i)
21. Y1=n_i/* Y1 赋值为当前节点*/
22. end if
23. end if
24. end for
/*内存读写流解析完后，得到 Nq 数据映射信息*/
25. SrctoDes_addr1(Nq)=DATA_addr(Y1)
26. SrctoDes_addr2(Nq)=DATA_addr(Y2)
27. return Nq

下面对各攻击阶段存在的状态节点、状态转移节点以及函数进行符号定义，如表 2 所列。

表 2 基本符号

Table 2 Basic symbols

符号名称	符号	描述
利用状态转移节点	Nq_{1t}	表示转变为利用状态的临界状态节点
触发状态转移节点	Nq_{2t}	表示转变为触发状态的临界状态节点，即漏洞触发点
用户态构造状态转移节点	Nq_{3t}	表示转变为用户态构造状态的临界状态节点，即用户态系统调用函数
提权状态节点	Nq_0	$Nq_0=\{Nq_{0_i}\mid i=1,\cdots,n\}$，表示提权状态下的状态节点集合
利用状态节点	Nq_1	$Nq_1=\{Nq_{1_i}\mid i=1,\cdots,n\}$，表示利用状态下的状态节点集合
触发状态节点	Nq_2	$Nq_2=\{Nq_{2_i}\mid i=1,\cdots,n\}$，表示触发状态下的状态节点集合
用户态构造状态节点	Nq_3	$Nq_3=\{Nq_{3_i}\mid i=1,\cdots,n\}$，表示用户态构造状态下的状态节点集合
提权状态下的反向分析过程	EOP	从提权状态到利用状态的状态转移过程
利用状态下的反向分析过程	EXP	从利用状态到触发状态的状态转移过程
触发状态下的反向分析过程	$TRIGGER$	从触发状态到用户态构造状态的状态转移过程
反向分析全过程	F	整个内核漏洞攻击的反向分析过程
辅助函数 1	Min_COUNT	$Min_COUNT(N)$表示一个集合 N 中 $COUNT(N_i)$最小的元素
辅助函数 2	Max_COUNT	$Max_COUNT(N)$表示一个集合 N 中 $COUNT(N_i)$最大的元素

EOP，EXP，$TRIGGER$ 这 3 个状态转移过程是反向分析的重点。而通过定义状态节点 Nq，能够有效表示 3 个状态转移过程，从而把反向分析全过程 F 等价于各阶段状态转移节点的获取。具体过程如图 5 所示，从最初表示权限信息篡改的提权状态节点，反向分析至利用状态转移节点，转变为利用状态；经过多个利用状态节点，反向分析至触发状态转移节点，转变为触发状态；经过多个触发状态节点，反向分析至用户态构造状态转移节点，转变为用户态构造状态。

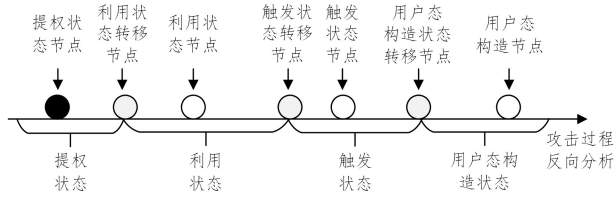

图 5 反向分析过程中的主要状态节点

Fig. 5 Main state nodes in reverse analysis

4.3 状态转移过程形式化描述

根据第3节总结划分的各阶段攻击模式，进一步利用状态节点形式化描述反向分析状态机的3个状态转移过程。

(1)从提权状态到利用状态的状态转移过程 EOP

存在如下关键状态节点：提权状态节点 Nq_{0_1} 可表示发生异常跳转的指令，如跳转至 Shellcode 或者 ROP 代码片段的指令；提权状态节点 Nq_{0_2} 可表示篡改 TOKEN 结构相关内存数据的指令；提权状态节点 Nq_{0_3} 可表示对应于 Nq_{0_1} 的最近一次相同内存数据写入指令，即写入异常跳转地址的指令；$Neop$ 可表示提权状态的关键状态节点集合。一般存在非控制流提权攻击状态节点集 $Neop=\{Nq_{0_2}\}$，或者控制流提权攻击状态节点集 $Neop=\{Nq_{0_3},Nq_{0_1},Nq_{0_2}\}$。

EOP 等价于获取状态转移节点 Nq_{1t}，可表示为 $EOP(Nq_{0_2})=Nq_{1t}$。

EOP 的具体过程为：

if $Neop==\{Nq_{0_2}\}$

$Nq_{1t}=Nq_{0_2}$

$STATE(Nq_{1t})=exp$

else if $Neop==\{Nq_{0_3},Nq_{0_1},Nq_{0_2}\}$

$Nq_{1t}=Nq_{0_3}$

$STATE(Nq_{1t})=exp$

(2)从利用状态到触发状态的状态转移过程 EXP

存在如下关键状态节点：$Nexp$ 可表示利用状态的关键状态节点集合。同时规定简单利用的 $Nexp$ 只存在一个状态节点，而多级利用的 $Nexp$ 存在两个及以上的状态节点。

为了获得 $Nexp$，设计了利用状态节点生成算法，具体过程如算法2所示。该算法的主要思想是：对于初始利用状态节点，依据数据依赖关系 DE，获取所有可能的利用状态节点，并加入 $Nexp$ 集合。

算法2 利用状态节点生成算法 exp_gen

输入：(Nq_{1_i}，Nexp)/*初始利用节点集*/

输出：Nexp/*完整的利用节点集*/

/*依据数据依赖关系 DE，获取下一节点的部分信息，并利用算法1 tracksource 得到下一节点的完整信息*/

```
1. while(true)
2.   DATA_value(Nq_{1_{i+1}})=DATA_addr(Nq_{1_i})
3.   COUNT(Nq_{1_{i+1}})=COUNT(Nq_{1_i})
4.   Nq_{1_{i+1}}=tracksource(Nq_{1_{i+1}})
5.   STATE(Nq_{1_{i+1}})=exp
  /*循环结束的条件：直到出现一个不符合要求的节点*/
6.    if SrctoDes_addr1(Nq_{1_{i+1}})∈user_addr&&SrctoDes_addr2
      (Nq_{1_{i+1}})∈kernel_addr
7.      Nexp=Add_set(Nexp,Nq_{1_{i+1}})
8.      i=i+1
9.   else
10.     break
11.  end if
12. end while
13. return Nexp
```

EXP 等价于获取状态转移节点 Nq_{2t}，可表示为 $EXP(Nq_{1t})=Nq_{2t}$。

EXP 的具体过程为：

$Nexp=exp\_gen(Nq_{1t},Nexp)$

$Nq_{2t}=Min\_Count(Nexp)$

$STATE(Nq_{2t})=trigger$

其中，存在简单利用过程的特殊情况，$Nexp$ 的元素个数始终为1，即 $Nq_{2t}=Nq_{1t}$。

(3)从触发状态到用户态构造状态的状态转移过程 $TRIGGER$

存在如下关键状态节点：$Ntrigger$ 可表示触发状态的关键状态节点集合。依据控制流信息，可得到 $Ntrigger=\{Nq_{2_1},\cdots,Nq_{2_n}\}$，且 $Nq_{2_1}=Nq_{2t}$。

$TRIGGER$ 等价于获取状态转移节点 Nq_{3t}，可表示为 $TRIGGER(Nq_{2t})=Nq_{3t}$。

TRIGGER 的具体过程为：

$Nq_{3t}=Min\_Count(Ntrigger)$

$STATE(Nq_{3t})=user$

在有限状态机 M 的基础上，利用状态节点形式化描述3个状态转移过程后，可得到基于有限状态机的内核漏洞攻击过程反向分析模型。而整个反向分析过程可形式化表示为 $F=TRIGGER(EXP(EOP(Nq_0)))$，即表示从初始提权状态节点，经过反向分析，可最终得到用户态构造状态转移节点。

同时，可得到如图6所示的基于状态节点的反向分析状态转移图。相比第3节的内核漏洞攻击状态转移图，图6中有明确的初始状态，减少了触发中间状态，从而能够减小解析无用状态的开销。而且3个状态转移过程都不需要细粒度的解析指令，因此能够实现一种粗粒度的反向分析方法，进一步减小分析开销。

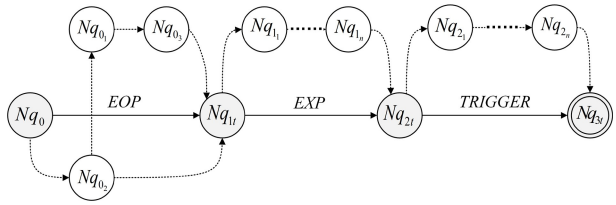

图6 基于状态节点的反向分析状态转移过程

Fig. 6 Reverse analysis of state transition process based on state node

5 内核漏洞攻击反向分析方法的具体实现

在第4节反向分析模型的指导下，可设计内核漏洞攻击反向分析方法的各步骤，实现内核漏洞攻击流程的自动解析。

反向分析的主要步骤包括提权分析、利用分析和触发分析，如图7所示。由第4节的形式化描述可知，各步骤实现的关键在于各状态节点的捕获。反向分析方法借助全系统动态分析平台 PANDA 实现。PANDA 是一个基于全系统仿真的动态分析平台，具有“全系统记录，一致性回放”的特点，可以实现各状态节点的准确捕获。

提权分析实现了利用状态转移节点的确定，需要捕获

异常跳转节点、异常跳转地址写入节点、权限信息篡改节点。异常跳转可通过分析相邻两个基本块的指令是否符合标准函数调用约定来判断，主要包括内核态非法跳转用户态、跳转ROP代码片段等情况；权限信息篡改可监控TOKEN结构相关内存数据的修改行为。在确定各节点后，根据反向分析子过程EOP，可得到提权信息和关键利用信息。

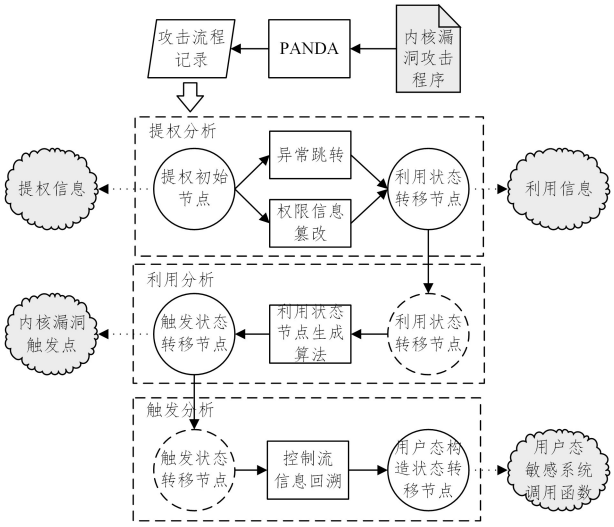

图7 反向分析流程

Fig. 7 Reverse analysis process

利用分析实现了触发状态转移节点的确定，需要借助动态分析平台获取利用数据的内存读写节点集。依据利用状态节点生成算法，可确定其余利用状态节点，并最终得到内核漏洞触发点。

触发分析实现了用户态构造状态转移节点的确定。在漏洞触发点，根据控制流信息，回溯确定各触发状态节点，最终得到用户态敏感系统调用函数。

利用反向分析方法能够自动得到内核漏洞攻击过程的用户态敏感系统调用点、内核漏洞触发点、中间利用点、提权代码执行点等关键代码执行信息，从而快速了解内核漏洞攻击流程。

6 实验评估

6.1 实验环境

本节使用实际的内核漏洞攻击样本，来对反向分析方法的有效性进行测试，表3列出了具体的实验环境。

表3 实验环境配置

Table 3 Experimental environment

名称	详细信息
主机环境	Ubuntu 18.04，2个CPU，8GB内存，100GB硬盘
客户机环境	Windows 7 x86 sp1，1个CPU，2GB内存，20GB硬盘
动态分析平台	PANDA2(基于QEMU2.9.1)
验证分析工具	WinDbg preview，IDA PRO 7.5

微软的增强缓解体验工具(Enhanced Mitigation Experience Toolkit，EMET)，可增加Windows 7下的可用保护措施，以便本文方法能够更好地适用于高版本的操作系统。

结合内核漏洞攻击的特征，主要考虑以下几种保护措施：DEP，SMAP，SMEP根据测试需求启用或关闭；ALSR，KALSR关闭，默认可以通过信息泄露的方式绕过相应保护措施，而且固定的模块基址有利于自动分析结果的验证。上述内容可通过微软的EMET实现。

6.2 有效性测试

为了验证本文方法的有效性，选取了Windows 7下的10个内核漏洞攻击程序进行测试。这里以CVE-2018-8120攻击程序分析流程为例。

首先通过PANDA的记录功能，生成CVE-2018-8120攻击过程的记录。通过提权分析，得到权限信息篡改节点、异常跳转节点、异常跳转地址写入节点，因此属于控制流提权攻击，异常跳转地址写入节点为利用状态转移节点；通过利用分析，依据利用状态节点生成算法，可以得到包含3个元素的关键利用状态节点集合，因此属于多级利用过程，并确定触发状态转移节点；通过触发分析，回溯控制流信息，可以得到关键触发状态节点集合，并最终确定用户态构造状态转移节点。上述过程的示例数据如图8所示。在快速获得上述信息后，通过符号表补充符号化内容，可以确定内核漏洞函数为win32k!NtUserSetImeInfoEx。

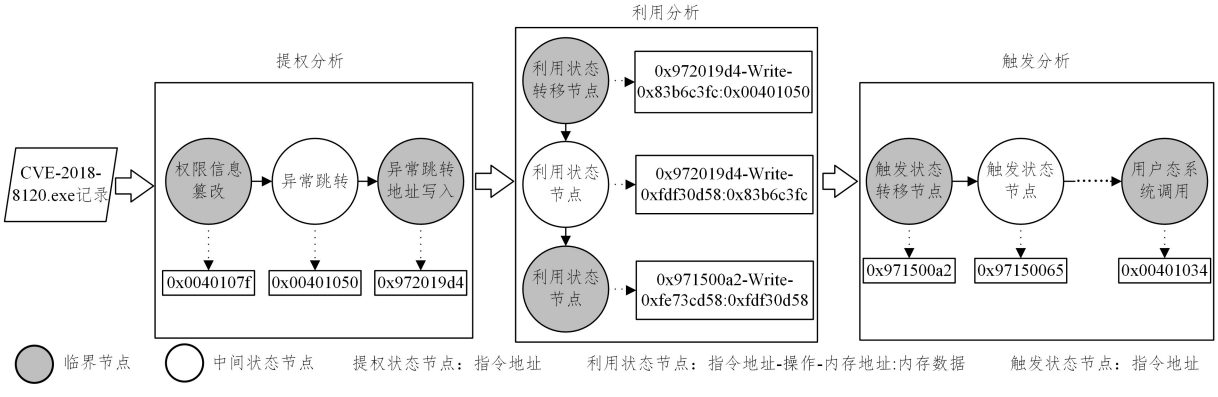

图8 CVE-2018-8120攻击程序分析

Fig. 8 CVE-2018-8120 attack program analysis

按照同样的分析流程，对其余攻击样本进行测试，实验结果如表4所列。其中，HEVD.sys的基址为0x8D982000，win32k.sys的基址为0x96210000，HEVD相关素材来源于开源项目HackSysExtremeVulnerableDriver，CVE攻击程序是

依据源码自构建的。

测试结果表明，利用反向分析方法能够有效得到大多数样本的关键代码执行信息。其中，CVE-2015-0003 和 CVE-2019-0808 的攻击样本不能直接得到实际的内核漏洞函数。出现该结果的原因是反向分析只是定位到了最初引发内存破坏的指令，对于一些成因复杂的漏洞，还需要进一步结合控制流信息和漏洞成因信息，才能得到实际的内核漏洞函数。但是，实验中得到的漏洞触发点能够辅助快速判断实际的内核漏洞函数。

表 4 测试样本反向分析结果

Table 4 Reverse analysis results of test samples

测试样本	实际漏洞类型及内核漏洞函数	反向分析得到的漏洞触发点及同一执行流关键内核函数
HEVD-1.exe	空指针解引用 HEVD! TriggerNullPointerDereference	0x8D986CF4 call dword ptr [esi+4] 0x8D986BE0 HEVD! TriggerNullPointerDereference
HEVD-2.exe	UAF HEVD! AllocateFakeObject	0x8D986593 rep movs dword ptr [edi],dword ptr [esi] 0x8D9864F8HEVD! AllocateFakeObject
HEVD-3.exe	栈溢出 HEVD! TriggerStackOverflow	0x83E787F3 rep movs dword ptr [edi],dword ptr [esi] 0x83E787C0 nt! memcpy 0x8D98662A HEVD! TriggerStackOverflow
HEVD-4.exe	池溢出 HEVD! TriggerPoolOverflow	0x83E787F3 rep movs dword ptr [edi],dword ptr [esi] 0x83E787C0 nt! memcpy 0x8D98612A HEVD! TriggerPoolOverflow
HEVD-5.exe	任意内存破坏 HEVD! TriggerArbitraryOverwrite	0x8D986B6B mov[ebx],eax 0x8D986B08 HEVD!TriggerArbitraryOverwrite
CVE-2018-8120.exe	空指针解引用 win32k! NtUserSetImeInfoEx	0x962200A2 rep movs dword ptr [edi],dword ptr [esi] 0x96220065 win32k! SetImeInfoEx 0x9621FFD8 win32k! NtUserSetImeInfoEx
CVE-2015-0003.exe	空指针解引用 win32k! FindSystemTimer	0x962C94F0 call dword ptr [esi+60h] 0x962C9347 win32k! xxxSendMessageTimeout 0x962C959Dwin32k! xxxSendMessage 0x9635B423 win32k! xxxFlashWindow 0x9635ACCE win32k! xxxSystemTimerProc
CVE-2019-0808.exe	空指针解引用 win32k! MNGetpItemFromIndex	0x9637E68D or dword ptr [esi+4],40000000h 0x9637E5D5 win32k! xxxMNSetGapState 0x9637E83Fwin32k! xxxMNUpdateDraggingInfo
CVE-2019-1458.exe	任意内存破坏 win32k! xxxPaintSwitchWindow	0x962C205D add [eax+8],ecx 0x962C2050 win32k! InflateRect 0x963779D9 win32k! xxxPaintSwitchWindow
CVE-2017-0101.exe	整数溢出 win32k! EngRealizeBrush	0x9624D78B mov [esi+3Ch],eax 0x9624D5A0 win32k! EngRealizeBrush

6.3 对比测试

为了对比验证本文方法在分析效率上有提高，基于 PANDA 实现了一种细粒度的正向污点分析方法，通过该方法能够得到漏洞攻击过程的关键数据依赖关系。通过比较两种方法的分析时间和内存消耗，来验证本文方法在分析效率上有较大提高。

多级利用过程很大程度上影响了分析时间，因此准备了两组测试集，测试集 test1 包含 5 个简单利用的样本，测试集 test2 包含 5 个多级利用的样本。通过分别统计两个测试集中样本的平均分析开销，来表明两种方法的分析能力。

两种方法的平均分析时间和内存消耗情况如表 5 所列。

表 5 两种分析方法开销的比较

Table 5 Cost comparison between two analysis methods

测试样本集	细粒度正向污点分析		反向分析	
	平均时间消耗/s	平均内存消耗/%	平均时间消耗/s	平均内存消耗/%
test1	146	67	32	22
test2	193	83	56	24

可以看出，反向分析优于细粒度的正向污点分析。在细粒度正向污点分析测试中，测试集 test2 中的个别样本内存消耗超出了基础限制，导致分析终止。这表明，在有限资源下，细粒度的正向污点分析方法可能无法正常分析内核漏洞攻击程序。

结束语 内核漏洞攻击由于成因复杂、危害巨大、分析难度高，是现如今信息安全领域需要特别关注的。

本文引入反向分析的思路，依据内核漏洞攻击状态转移图，建立了基于有限状态机的内核漏洞攻击过程反向分析模型，实现了一种粗粒度的内核漏洞攻击反向分析方法，可自动快速解析内核漏洞攻击流程。实验结果证明了该方法可以准确高效地完成内核漏洞攻击程序的分析工作。

由于反向分析方法简化了漏洞触发的分析过程，无法直接得出漏洞成因的分析结果，因此今后的研究工作将致力于对漏洞成因的自动分析。

参 考 文 献

[1] Vulnerability and technical analysis of Windows local rights raising in APT activities [EB/OL]. https://paper.seebug.org/1753/#apt.

[2] ZHANG K, LIU J J. Network Attack Path Analysis Method-Based on Vulnerability Dynamic Availability[J]. Netinfo Security, 2021, 21(4): 62-72.

[3] MA M Y, CHEN L W, MENG N. A Survey of Memory Corruption Attack and Defense [J]. Journal of Cyber Security, 2017, 2(4): 82-98.

[4] Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks[C]// Symposium on Security and Privacy

(SP). 2016:969-986.
[5] JANG H, PARK M C, LEE D H. IBV-CFI: Efficient fine-grained control-flow integrity preserving CFG precision[J/OL]. Computers & Security. https://www.researchgate.net/publication/340442234_IBV-CFI_Efficient_fine-grained_control-flow_integrity_preserving_CFG_precision.
[6] LU S B, LIN Z C, ZHANG M. Kernel Vulnerability Analysis: A Survey[C]// 2019 IEEEFourth International Conference on Data Science in Cyberspace (DSC). Hangzhou, China, 2019: 549-554.
[7] PAN J F, YAN G L, FAN X C. Digtool: A virtualization-based framework for detecting kernel vulnerabilities[C]// 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC: USENIX Association, 2017: 149-165.
[8] JURCZYK M, COLDWIND G. Bochspwn: Exploiting Kernel Race Conditions Found via Memory Access Patterns[C]// The Syscan'12 Conference. 2013.
[9] BRENDAN D G, JOSH H, PATRICK H, et al. Repeatable Reverse Engineering with PANDA[C]// 5th Program Protection and Reverse Engineering Workshop (PPREW-5). Association for Computing Machinery, New York, NY, USA, 2015: 1-11.
[10] MING J, WU D H, WANG J, et al. StraightTaint: decoupled offline symbolic taint analysis[C]// the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16). 2016: 308-319.
[11] WANG X J, MA R, DOU B W, et al. OFFDTAN: A New Approach of Offline Dynamic Taint Analysis for Binaries[C]// Security and Communication Networks. 2018: 1-13.
[12] XU J, MU D L, CHEN P, et al. CREDAL: Towards Locating a Memory Corruption Vulnerability with Your Core Dump[C]// the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 2016: 529-540.
[13] XU J, MU D L, CHEN P, et al. POMP: Postmortem program analysis with hardware-enhanced post-crash artifacts[C]// the 26th USENIX Security Symposium. USENIX Association, 2017: 17-32.
[14] CUI W D, PEINADO M, CHA S K, et al. RETracer: Triaging Crashes by Reverse Execution from Partial Memory Dumps [C]// the 38th International Conference on Software Engineering (ICSE). 2016: 820-831.
[15] ZHENG Y, WANG Z, FAN X Y, et al. Localizing multiple software faults based on evolution algorithm[J]. The Journal of Systems & Software, 2018, 139: 107-123.
[16] JIANG S J, ZHANG X, WANG R C, et al. Fault Localization Approach Based on Path Analysis and Information Entrop[J]. Journal of Software, 2021, 32(7): 2166-2182.
[17] GUO W B, MU D L, XING X Y, et al. DEEPVSA: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis[C]// Proceedings of the 28th USENIX Security Symposium. Santa Clara: USENIX Association, 2019: 1787-1804.
[18] YAGEMANN C, PRUETT M, CHUNG S P, et al. ARCUS: Symbolic Root Cause Analysis of Exploits in Production Systems[C]// the 30th USENIX Security Symposium. 2021.
[19] BLAZYTKO T, SCHLOGEL M, ASCHERMANN C, et al. AURORA: Statistical Crash Analysis for Automated Root Cause Explanation[C]// the 29th USENIX Security Symposium. 2020.
[20] NI T, YE X. Privilege Escalation Technology of Kernel Vulnerabilities in Write What Where Mode[J]. Journal of Information Engineering University, 2014, 15(2): 232-236.

LIU Pei-wen, born in 1997, postgraduate. His main research interests include cyber security and reverse engineering.

SHU Hui, born in 1974, Ph. D, professor, Ph.D supervisor. His main research interests include cyber security and reverse engineering.

(责任编辑:喻藜)