◎热点与综述◎

竞态漏洞检测方法综述

赵世斌,周天阳,朱俊虎,王清贤
ZHAO Shibin, ZHOU Tianyang, ZHU Junhu, WANG Qingxian

数字工程与先进计算国家重点实验室,郑州 450002
State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China

ZHAO Shibin, ZHOU Tianyang, ZHU Junhu, et al. Survey on race condition detection. Computer Engineering and Applications, 2018, 54(3):1-10.

Abstract: Race condition induced by resource race threatens operating system a lot in parallized execution environment. Attackers usually take actions such as remote command execution, local priviledge exploitation etc. By using this kind of vulnerabilities. This paper proposes a general race condition detection framework after analysing vulnerability mechanism in different conditions, and summarizes the development process of race condition detection method in user-mode and kernel-mode separately. The bottleneck of dection efficiency and its solution is also discussed. And the recent development trend and problems need to be solved is pointed in the end, combining newest technique application.

Key words: race condition detection; happens-before; lock-set; shared resource operation trace

摘 要: 并行化程序运行环境中资源竞争导致的竞态漏洞是当今操作系统安全的重要威胁之一,攻击者常常间接利用竞态漏洞实施诸如远程命令执行、本地提权等攻击行为。分析了不同条件下竞态漏洞的产生机理及其相互关系,提出了竞态漏洞检测基本范式和通用框架,分别综述了用户态和内核态竞态漏洞检测方法的技术思想和发展脉络,讨论了制约检测效率的瓶颈问题以及可能的解决方法,结合最新技术应用指出了未来发展趋势和亟待解决的问题。

关键词: 竞态漏洞检测;happens-before;lock-set;共享资源操作轨迹
文献标志码: A **中图分类号:** TP311 doi:10.3778/j.issn.1002-8331.1711-0015

1 引言

现代操作系统功能结构和程序逻辑复杂,开发人员难以为所有共享资源提供完善的保护机制,导致了由于设计缺陷和实现不当产生的逻辑缺陷时有发生。这些逻辑缺陷被称为竞态漏洞,其不仅会导致程序本身运行出错和系统崩溃,还常被攻击者恶意利用,实现远程命令执行和本地提权等攻击效果[1]。目前,Web服务、杀毒软件等应用程序,Linux、Android和IOS操作等系统内核中潜藏的竞态漏洞被频繁曝光,竞态漏洞日益成为威胁网络信息系统安全的严重隐患。

竞态漏洞与堆栈溢出、格式化字符串等传统漏洞相比,具有隐蔽性强、难以调试、攻击效果明显等特点。目前,学者主要从程序竞争性行为分析以及由此触发的破坏性结果两个方面着手进行研究。前者主要针对竞态漏洞的发生条件进行分析,后者主要针对漏洞触发后漏洞利用过程中的程序异常行为进行分析。因为程序异常行为分析和竞态漏洞的成因关系不密切,本文主要面向竞态漏洞的发生条件进行研究。1992年,Netzer和Miller论证指出,由竞态程序行为导致的漏洞发现与定位是一个NP难问题[2]。

在漏洞挖掘与检测领域,如何快速有效地检测未知竞态漏洞是一项极具挑战性的研究工作。近年来,研究人员从不同系统特权级对竞态漏洞的产生机理及其检测技术展开了富有成效的研究,主要分为用户态和内核

基金项目: 国家自然科学基金(No.61502528,No.61402525,No.61402526)。
作者简介: 赵世斌(1993—),男,硕士研究生,研究领域为漏洞检测技术,E-mail:zsbpro@163.com;周天阳(1979—),男,博士研究生,讲师,研究领域为虚拟化技术;朱俊虎(1974—),男,博士,教授,研究领域为网络安全;王清贤(1960—),男,教授,博士生导师,研究领域为网络安全。
收稿日期: 2017-11-02 **修回日期:** 2017-12-25 **文章编号:** 1002-8331(2018)03-0001-10

态两个方向。其中,用户态竞态漏洞检测研究重点是访问共享资源的程序行为的逻辑关系[3-10],以及如何在有限条件下平衡检测效率和准确率之间的关系[11-17];内核态竞态漏洞检测研究则侧重于解决程序语义视图缺失问题,同时尽量减少侵入式检测方法对内核平稳高效运行的影响。随着系统虚拟化技术不断成熟和应用,研究人员尝试利用虚拟机监控器(hypervisor)的特权级优势,从系统底层对操作系统程序行为进行动态监控和调试分析[18],取得了较好的检测效果,为竞态漏洞检测研究探索了新方向。

本文从竞态漏洞定义出发,分析了不同类型程序竞态缺陷以及由此引发竞态漏洞的作用机理;在综合研究各类竞态漏洞检测原理与实现技术的基础上,提出了竞态漏洞检测基本范式和通用框架,深入分析了竞态漏洞检测涉及的关键技术环节及其需要解决的核心问题;依据不同检测目的和实现方法,分别论述了用户态、内核态两大类竞态漏洞检测研究中的技术思想及其发展脉络,指出了制约检测效率的瓶颈问题以及可能的解决方法。本文对竞态漏洞检测技术的总结展望将为当前及未来研究提供借鉴与参考。

2　竞态漏洞检测技术概述

如图1所示,竞态漏洞检测技术是一个数据提取和分析的过程。进行竞态漏洞检测首先需要选择数据的类型,即共享资源操作轨迹,包括共享内存读写、共享磁盘读写、信号量的使用、锁的使用、设备的使用等;然后根据需要的数据类型采取合适的数据提取方法,包括日志记录、插桩、虚拟化技术、Intel PT技术等;最后通过设计合适的数据分析方法对提取的数据进行分析,得到正确的判决结果。其中操作轨迹提取和漏洞触发识别组成了竞态漏洞检测的基本框架。

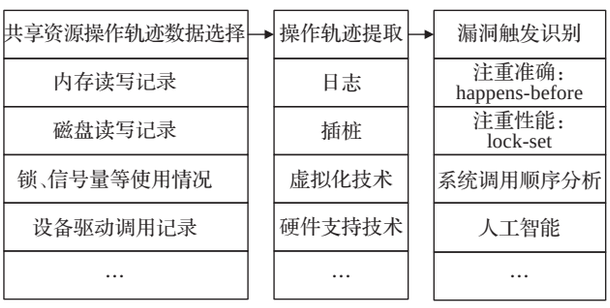

图1　竞态漏洞检测整体框架示意图

2.1　竞态漏洞定义与分类

竞态条件(race condition)是指这样一种情形——多个线程或进程在读写一个共享数据时结果依赖于它们执行的相对时间。竞态条件发生在多个进程或者线程读写数据时,其最终的结果依赖于多个进程的指令执行顺序。例如,在多个任务系统中,两个连续的操作作用在同一个文件时,系统会按照竞争条件的原则做出决定。此时,两个连续操作之间存在一定的时间间隔,如果攻击者利用这个时间间隔对文件进行了恶意篡改,则产生了竞态漏洞利用。

竞态条件漏洞,又名竞争条件漏洞,简称竞态漏洞,其定义为事件的执行时间或顺序影响程序正确性的一种缺陷[2]。竞态漏洞的定义是从程序执行产生的结果进行定义的,只有竞态条件发生且造成正确性缺失的情况才会被称为竞态漏洞;当竞态条件发生,但是没有对程序正确性造成影响,不能称之为竞态漏洞。

竞态漏洞的定义可以看出,竞态漏洞检测的关键在于:一是及时发现并定位竞争性的资源访问(可简称为竞态条件的产生);二是已构成竞态条件时危害程序正确性的异常行为。由于危害程序正确性的行为复杂多样,包括但不限于命令执行、信息泄露、拒绝服务、本地提权等各类漏洞利用行为,难以通过单一的检测方法实现所有异常行为的检测,因此研究人员更倾向于从竞态漏洞的产生原因(即竞态条件)来进行研究。不同原因产生的竞态漏洞具有不同特点,利用这些特点进行检测更有针对性。目前,针对竞态条件的研究主要包括:数据竞争和TOCTTOU(Time Of Check To Time Of Use)两类。

2.1.1　数据竞争

一个内存访问操作定义 e 为一个四元组 (m,t,L,a),其中:m 为内存访问操作的内存地址;t 为标识内存访问操作的线程;L 为操作所属线程拥有的锁集合;a 为内存访问操作的类型(READ或WRITE)。

数据竞争 $IsRace(e_i,e_j)$ 是满足以下条件的两个内存访问操作[19]:(1)内存访问位置相同;(2)两者并发执行;(3)至少有一个为写操作;(4)未使用“互斥”的同步机制约束。表示为:

$$\begin{aligned} IsRace(e_i,e_j)\Leftrightarrow &(e_i\cdot m=e_j\cdot m)\wedge\\ &(e_i\cdot t\neq e_j\cdot t)\wedge(e_i\cdot L\cap e_j\cdot L=\varnothing)\wedge\\ &(e_i\cdot a=WRITE\vee e_j\cdot a=WRITE) \end{aligned}\tag{1}$$

2016年被曝光的DirtyCow漏洞(CVE-2016-5195)是最为典型的数据竞争。该漏洞是由于Linux内核在处理写时复制的过程中没有限制内存的竞争性访问,任意文件在写时复制中可被恶意篡改,进而被攻击者用来进行提权操作[20]。DirtyCow漏洞存在于Linux内核2.6.22以上的所有版本中,危害范围广泛,包括绝大多数Linux服务器、主机以及安卓手机[21]。

如图2所示,正常的程序流程三次调用了 *faultin_page* 函数完成了三个步骤,其中第二次进入 *faultin_page* 主要是处理写权限的页错误问题,要求的写权限标志会被去掉,即去掉FOLL_WRITE标志位,第三次调用 *faultin_page* 时已经成功得到cow后的页面,且flags已

经去掉FOLL_WRITE,因此不会再产生写错误的处理,可以直接写入cow的页。但是如果在上述流程即第二次页错误处理结束时,在一个新的线程调用madvise,会unmap掉前面cow的页面,又进入缺页处理,这里不同的是在do_fault调用时,由于没有了写权限的要求,直接调用了do_read_fault读取映射文件的内存页,而不是内存页副本,后续即可实现越权写操作。在DirtyCow漏洞中,两个线程竞争的资源是内存页,并且造成了程序正确性的影响,所以DirtyCow漏洞是数据竞争。

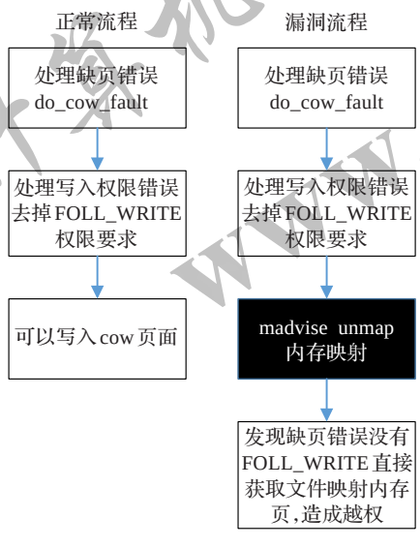

图2 DirtyCow漏洞触发流程图

由定义分析可知,数据竞争是竞态漏洞产生的必要条件,但不是充分条件。这是因为某些数据竞争发生时并不会影响程序的正确性,例如当两个进程竞争的内存共享资源和两个进程的逻辑完全无关,则认为构成数据竞争但是不构成竞态漏洞。由此可见,只有那些影响程序正确性的竞争性数据访问才可构成竞态漏洞产生的充分条件,DirtyCow正是由于在数据竞争发生之后导致任意文件被恶意篡改的危害性结果,才被研究人员归结为竞态漏洞。

2.1.2 TOCTTOU

TOCTTOU是竞态条件的一种,指计算机系统中的资源与权限等状态在检查(安全授权)和使用这个检查结果之间,因为检查结果(如授权状态)在这段时间发生了改变而造成的漏洞产生[22]。TOCTTOU通常发生在文件系统的访问时,特别是在UNIX操作系统中较为常见,文件系统访问一般会要求对文件先检查再写入,这就导致检查与读写操作之间存在时间间隔,攻击者可利用这种时间间隔对文件系统展开攻击,如写入恶意代码等。2010年爆出的KDE桌面本地提权漏洞(CVE-2010-0436),就发生在kdebase-workspace-4.1.4/kdm/backend/ctrl.c的openctrl功能中,根据漏洞产生原因与实现机理研究人员将其归结为一种TOCTTOU型竞态漏洞[23]。

目前,针对类UNIX系统的竞态漏洞检测主要是利用其文件系统的TOCTTOU特点展开研究[24]。类Unix文件系统中存在TOCTTOU缺陷的根本原因在于文件名和文件对象之间的映射是可变的,如果symlink等操作的时机准确,那么可以在文件的检查和使用之间替换文件,从而导致错误发生。

如图3所示是TOUCTTOU的典型例子Binmail。Binmail是一个setuid-to-root程序,在普通用户权限下可以调用执行root用户权限的操作。在正常的读取邮件的过场中,Binmail首先通过*lstat*函数查看文件mail的信息,如果mail文件是正常文件,不是符号链接则执行*open*函数打开邮件。但是由于*lstat*和*open*函数不是原始操作,所以如果在*lstat*函数检查完毕后,另外一个线程或者进程可以通过unlink和symlink操作将mail文件替换为指向系统关键文件/etc/passwd等文件的链接,那么*open*的文件将会是替换后的系统关键文件,实现了任意文件读取。

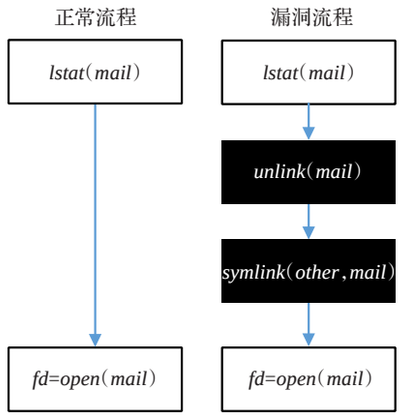

图3 Binmail漏洞触发流程图

2.1.3 竞态漏洞产生原因相互关系

由于不同进程拥有各自不同的内存、寄存器等资源,多进程执行产生竞态条件继而引发漏洞利用的可能性较小。大部分不同进程之间发生竞态漏洞的情况都是针对文件系统等多进程间共享资源TOUTTOU类型的竞态漏洞[25]。线程因为共有内存的缘故,使得大部分竞态漏洞都是线程之间资源竞争导致的。在现有CVE披露的竞态漏洞中,线程之间竞态漏洞最主要的原因是内存中数据竞争类型的竞态漏洞[26]。

如韦恩图(图4)所示,因为数据竞争并不一定会对程序的正确性造成影响,所以数据竞争和竞态漏洞是交集的关系。在韦恩图中,存在同时属于数据竞争和竞态漏洞的区域,指数据竞争影响了程序正确性并造成了危害,可以被视为竞态漏洞;导致程序存在属于数据竞争但是不属于竞态漏洞的部分,指数据竞争没有对程序的正确性造成影响;也存在不属于数据竞争但是属于竞态漏洞的部分,指不满足数据竞争的四个条件但是却影响

了程序的正确性,如死锁。在文献[3]中,特别地讨论了数据竞争中有害和无害的区分方法。TOCTTOU中检查的过程是为了确保数据不会被更改而诞生的过程,如果在此之后对数据进行了更改会违背检查的目的,从而影响程序的正确性,所以TOCTTOU是竞态漏洞的子集。

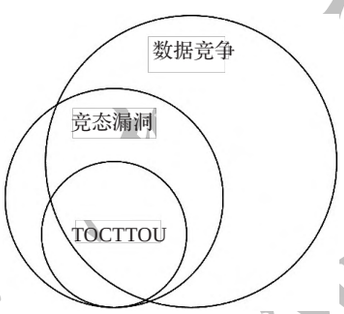

图4　竞态漏洞分类韦恩图

在进行竞态漏洞检测方法的研究过程中,不会研究对程序正确性无法造成影响的数据竞争,研究人员主要研究的竞态漏洞产生原因为数据竞争的有害部分和TOCTTOU。

2.2　竞态漏洞检测技术基本框架

竞态漏洞与栈溢出、堆溢出、格式化字符串等传统漏洞产生机理存在差异,是在开发之前由系统实现过程中对多进程、多线程相互干扰问题考虑不当而造成的安全隐患[27]。因此,竞态漏洞具有较强的隐蔽性,模糊测试、符号执行等传统漏洞分析手段难以将其有效检测出来[28];同时,由于竞态漏洞主要发生在多进程或多线程相互作用之时,竞争性的程序运行环境给调试带来很多挑战。

根据对现有文献研究,竞态漏洞检测基本框架主要包括两个部分:操作轨迹提取和漏洞触发识别[29]。操作轨迹提取是指从多进程或多线程的运行过程中提取共享资源的程序操作轨迹;漏洞触发识别则是根据共享资源操作轨迹识别和判断漏洞是否被触发。不同竞态漏洞的操作轨迹和漏洞触发机理具有不同特点,相应的检测思路也不尽相同,特别是在不同系统特权级环境下,检测方法的实现方式差异较大。目前,在竞态漏洞检测框架的两部分研究中,具体需要解决三个关键问题,分别是:数据种类选取、数据提取与管理以及检测性能与准确性平衡。如图5所示,在竞态漏洞检测基本框架下,这三个关键问题彼此影响,相互制约。

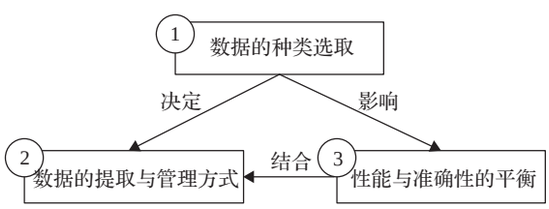

图5　竞态漏洞检测技术基本框架面临的问题

2.2.1　数据种类选取

面向不同检测对象,在不同操作系统特权级环境中,依据不同检测策略选取的数据种类在竞态漏洞检测中发挥不同的作用。系统调用数据是最基础的数据种类,监控操作系统调用,为竞态漏洞检测提供分析数据,是现在内核态竞态漏洞检测中的基本方法[30];函数级别的数据是高语义级别的数据,通常只能通过侵略式插桩的方式获取,选取合适的函数进行监控,可以生成完整的共享资源操作轨迹供检测系统分析,是现在用户态竞态漏洞检测中较为成熟的方法。操作类型也是数据种类提取需要考虑的问题,在完整共享资源操作轨迹提取方法中,通常会将内存、文件等共享资源操作区分为读、写、锁[31]三种操作,按照逻辑顺序排列后作为共享资源操作轨迹的记录[32]。

2.2.2　数据提取与管理方式

数据的提取和管理方式是连接数据和分析系统的桥梁,是竞态漏洞检测技术基本框架需要研究的重要问题。数据提取的关键是对系统操作通过侵略式插桩[33]的方式进行调试,常见的侵略式插桩的方式有动态插桩技术、重编译技术、HOOK技术。动态插桩技术指在待分析二进制程序中插入探针,通过探针的执行进行数据的提取与分析;重编译技术指在源码层面加入数据提取的代码,将程序重新编译的方式进行数据的提取;HOOK技术指通过改写需要检测的关键函数,使得程序首先运行数据提取指令再恢复正常程序流程的方式实现数据提取。随着硬件调试、虚拟化自省技术的快速发展,在系统外进行数据提取的方式逐渐成熟起来[34],在特定场景和条件下取得了不错的效果。数据管理是数据提取后需要解决的重要问题。对提取的数据进行存储有助于提高检测的准确率,但数据管理本身会产生额外性能开销,需要进一步展开研究,主要包括:数据存储的数据结构设计及其空间复杂度,数据的存、查、删等操作策略及其时间复杂度等。

2.2.3　性能与准确性的平衡

竞态漏洞数据分析过程中的关键问题是性能和准确性的平衡问题。竞态漏洞是由两个或两个以上进程/线程竞争资源造成的,因此数据分析需要对不同进程或线程的操作进行比较。操作系统内部对共享资源的访问操作具有频率高、种类多的特点。为提高检测的准确性,应尽可能比对所有相关操作,但这会造成检测开销大的问题。因此在设计检测方法时,必须考虑比较不同操作的性能开销。针对不同的检测环境和检测目标,在现有的竞态漏洞检测方法中出现了注重性能和注重准确性两种不同的理念。

3　用户态竞态漏洞检测

用户态竞态漏洞检测的本质是动态地监控并行程

序多个线程的内存访问操作并进行数据分析。检测程序通过监控用户态程序在并发执行过程中发生的内存访问操作和其使用的同步保护机制,使用算法推断竞态漏洞的发生,不同的检测手段使用了不同的检测与推断的方法,而评价方法的三个最重要的因素是:漏报率、误报率和额外性能开销。漏报率指检测方法所不能检测到的竞态漏洞占所有竞态漏洞的比例;误报率指检测结果中不被确认为竞态漏洞占检测结果数量的比例;额外性能开销指引入检测方法后系统运行同样的程序增长的时间占原运行时间的比例。目前,用户态竞态漏洞检测分为两种思路:happens-before 方法[2]和 lock-set 方法[11]。happens-before 方法注重准确性,误报率低,但是无法检测出所有潜在的竞态漏洞,漏报率高,额外性能开销大,研究人员的研究重点围绕着性能优化开展;lock-set 方法注重效率,会产生虚警,误报率高,系统额外开销低,研究人员的研究重点围绕着准确性提高开展。

3.1 happens-before 方法

happens-before 方法就是通过检测两个操作是否满足 happens-before 关系来实现竞态漏洞的检测。如果在程序执行过程中任意两个操作都满足 happens-before 关系则认为不存在竞态漏洞,反之则认为存在。

happens-before 关系最初在 1978 年 Lamport 提出,用来描述程序事件的一种偏序关系[2]。在两个事件中,在一个事件发生前能够确定另外一个事件的结果,则称两个事件满足 happens-before 关系。如图 6 所示,当一个线程对共享资源 v 的操作使用了锁,并且在之后一个不同的线程中对共享资源 v 再次申请了锁并且使用完后释放,那么率先使用锁对共享资源 v 操作的线程和后续使用锁的线程满足 happens-before 关系。通过设置 happens-before 规则,在每个多线程进程中检测任意两个操作之间是否满足 happens-before 关系。如果存在两个操作不满足 happens-before 关系,则判定存在竞态漏洞。因为 happens-bofore 方法能够确保所有操作之间不会存在结果不确定的情况,所以 happens-before 方法误报率低。20 世纪 90 年代初期,Netzer 对竞态漏洞进行了详细地分类与说明[29],预估了检测竞态漏洞的难度[2],将竞态漏洞检测问题抽象成一个数学模型进行了检测方法的探讨,并且率先基于 Lamport 的 happens-before 方法,使用信号量同步实现了竞态漏洞的检测[35]。

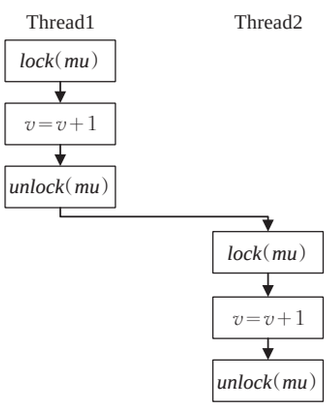

图 6 happens-before 方法检测案例

理想化的 happens-before 检测模型需要记录大量指令集操作并执行推断操作,以实现较高的检测精度,但这会大大增加实际的程序开销,严重影响检测效率。检测系统在实现过程中会使用向量时钟(Vector Clock,VC)来记录每个线程的共享资源操作轨迹数据,来进行 happens-before 关系的判断。如果被测程序有 n 个线程,那么每个 VC 都需要 $O(n)$ 的存储空间,每个 VC 操作需要 $O(n)$ 的时间进行算法计算,而在操作系统中 VC 操作的数量每秒都可能数以千计,所以时间复杂度和空间复杂度都在 $O(n)$ 的情况下会对系统造成极大的额外性能开销,严重影响检测效率和用户体验。

为了降低额外性能开销,研究人员采用降低记录 VC 操作数量的方法对 happen-before 进行改进。一种有效的思路是进行数据采样,通过降低监测分析数据量,提高检测性能,例如,文献[4]利用微软通用语言运行库(Common Language Runtime,CLR)设计了一种基于 happens-before 方法的检测系统 RaceTrack,该系统的每个检测单元均维护了一个访问历史数据库,为减少性能开销,系统设置了检测粒度和日志存储量,通过对关键数据进行记录,不关键的数据进行抛弃的方式有效地提升了检测效率。文献[3]则将数据竞争条件区分为有害和无害两种类型,多线程执行记录分析[36]提高检测效率。Maino 等人对采样方式进一步优化,只记录小于 2%访存操作,可以检测出 70%以上的竞态漏洞[37]。

happens-before 使用采样方法进行优化时,采样的时机直接影响了性能和准确率。为了准确把握 happens-before 的识别时机,Flanagan 设计了 FastTrack 系统[5],改进了笨重的 vectorclocks,采用了一种轻量级的 vectorclocks,在没有精度损失的情况下将监控操作的时间复杂度从 $O(n)$ 级别几乎降至 $O(1)$ 级别。Bond 等人[6]设计的 Pacer 提供了高速的采样机制,结合 FastTrack 实现了检测的速率近似于取样的速率,并将采样产生的额外开销降至 1%~3%。后期研究人员继续进行采样方法的创新和优化。Poluri 等人[8]使用 DJIT+结合 FastTrack 的方式进行了创新;Wilcox 等人[9]为了进一步优化,基于 FastTrack 使用了 Array Shadow State 进行了数据存取性能的优化,与 FastTrack 相比提升了约 35%的运行效率。文献[38]中分别针对漏洞研究者和软件使用者分别设计了高效的 RaceChaser 和精确的 Caper,区分了注重效率和注重性能的应用场景。

除了努力提高检测性能,研究人员还着重研究了检测准确性和检测范围等问题。Huang 等人通过控制流抽象的方式提升了 happens-before 的完备性[10],而不是提

高性能。通过提高准确性和检测范围,happens-before方法的性能虽然会降低但是却可以更好地为漏洞挖掘等不需要性能的应用服务。

3.2 lock-set方法

lock-set方法最早由Savage等人在Eraser系统中提出[11]。这种方法的基本思想是提取并记录不同线程所申请的锁的集合,当每个线程访问共享资源时,取不同线程针对该共享资源锁的交集,如果两个线程对该共享资源没有使用同一个锁保护,那么两个线程就可能同时对该共享资源进行操作,认为存在竞态漏洞。

如图7所示,在两个线程中对相同的共享资源使用了不同的锁进行保护,存在竞态漏洞,会被lock-set方法检测出来。lock-set方法的本质是检测由不同进程或线程发出的没有被相同锁保护的冲突操作,其选取的数据种类是共享资源操作轨迹中的锁操作,具有较快的数据提取效率;lock-set方法对数据进行分析时采用的是集合的操作,具有较快的数据分析效率。因此,lock-set方法与happen-before方法相比,同时具有更高效的数据提取和分析效率,可以有效降低性能开销,被看作是happen-before的替代方法。当然,由于选取的数据片面,集合的判断逻辑简单,该方法存在虚警问题。由于lockset具有优异的检测效率,后续学者对其进行了更深入的研究,例如Praun等人[12]针对多线程的JAVA程序,利用面对对象的特殊属性,将变量级别的检测提升到了对象级别,优化了检测的性能,减少了16%~129%的即时开销,和75%以上的空间开销;Nishiyama[13]则是通过read-barrier-based方法减少检测数据竞争的必须检查的内存位置数量,针对SPEC基准测试,开销分别为57.8%至735.7%,对于Java Grande基准测试则为0.6%至6 012.5%。

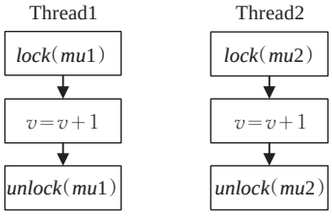

图7　lock-set方法检测案例

研究学者发现很难纯粹通过算法优化来同时获得准确率和性能的大幅度地提升,于是采取了与其他检测手段或方法结合的方式来辅助lock-set方法进行竞态漏洞的检测。Choi等人通过定义“weaker-than”关系过滤了一些冗余的访存记录,降低了访存监控开销[5]。两个已经发生的访存操作 e_i 和 e_j 满足“weaker-than”关系,指对于任意将要发生的访存操作 e_k 来说,如果 $IsRace(e_j,e_k)$ 发生可以推理出 $IsRace(e_i,e_k)$ 发生,那么可以说 e_i “weaker-than” e_j ,即可以不考虑 e_i 事件,将其从访存记录中删除。美国伊利诺斯大学采取体系结果扩展技术在硬件平台上实现了锁集合方法,将集合操作转换为硬件的位图操作,提高了效率,但是这种方法的误报率仍然高达0.89[14]。

为了能够有效地减少误报率,降低虚警,研究人员开始通过牺牲一部分性能来换取准确率。例如2007年出现的开源项目Helgrind[15],通过与happens-before结合的方式有效地提升了准确率。Helgrind是Valgrind的一部分,项目的目的是针对C/C++和Fortran程序设计一款检测数据竞争的工具,其采取的是Eraser中的lock-set算法,并使用了happens-before来减少虚警。其通过将二进制文件转换为中间语言,并使用即时编译器执行,并通过插件来进行分析。Helgrind出现后,Jannesari和Tichy基于Helgrind实现了多线程程序中数据竞争检测的方法[16]。

还有一部分学者将注意力集中在继续提高lock-set的性能上。例如,来自清华的Tianwei Sheng等人基于lock-set设计了RACEZ,其通过硬件性能监测装置透明地获取指令记录[17],性能优异,平均只有2.8%的额外开销;而后在2015年,同一个团队使用了CPU硬件的PMU功能进行监控,识别对内存的受保护的访问和不受保护的访问,对lock-set的性能进一步进行了优化。

3.3 未来发展

在现有研究中,研究人员针对happens-before方法和lock-set方法做了大量的性能优化和准确性提升的工作,取得了很多不错的结果。但是在已有的方法中,仍然存在可以继续深入研究和发展的空间。

3.3.1 智能精确的数据选取

在现有的方法中,通过取样的方式进行happens-before方法的性能优化是优秀的性能、准确性平衡的手段,但是取样的方法是值得研究的问题,使用自适应的方式进行取样是值得借鉴的一种尝试。未来的检测技术的研究手段中,可以使用机器学习、数据挖掘等更为智能的方法进行数据的选取,精确地选取必要的数据,以较小的数据量获得较大的信息量,提高检测的性能和准确性[39]。

3.3.2 透明高效的数据提取

用户态可以通过侵略式插桩的方式提取到高语义的共享资源操作轨迹数据,但是提取方式不透明、效率低。不透明指在进行数据提取的过程中,恶意进程可能会发现检测的痕迹从而和检测系统形成对抗。效率低指数据提取的性能开销大,影响程序的正常运行。

针对以上问题,探索新的共享资源操作轨迹数据的提取方式是研究人员关心的问题。Sheng等人[17]、Kistler[40]等人的研究都对数据提取方式进行了一些新的尝试,从效果来看,基于硬件的方法取得了不错的效果。在虚拟化技术飞速发展的今天,使用硬件虚拟化[41]、虚拟机自省技术[42]进行高效率、高透明共享资源

操作轨迹数据提取是一个很好的思路,赵跃华等人的工作[18]给出了利用硬件虚拟化技术进行数据提取的尝试。基于虚拟化自省技术、硬件虚拟化技术,检测系统可以通过设置客户操作系统从ring3进入ring0的调用作为VM Exit的条件实现低语义层级的系统调用监控,在客户机不知情的情况下访问虚拟机系统的内部状态,包括内存、CPU寄存器、IO设备标志;监测特定事件,包括特权寄存器更改、设备状态更改等;对监测中的虚拟机进行控制,如暂停、快照、重启,这为程序调试分析特提供了极大便利[43]。

进行漏洞检测的时候,对共享资源操作轨迹数据存取和查询的效率影响了检测的性能。针对硬盘存储的优化,如何对内存存储部分、硬盘存储部分的数据结构进行设计,选取合适的数据结构能够兼顾数据的存取和查询是未来的研究需要重点解决的问题[44]。

3.3.3 并行化共享资源操作轨迹数据分析

用户态竞态漏洞检测的方法自19世纪80年代开始经历了30多年的推敲,很难再有大幅度的性能提高,利用现有云计算、集群计算、并行计算发展的计算优势设计并行化数据分析算法,提高数据分析的性能,是优化竞态漏洞检测性能的一种方法,然而复杂的计算机系统环境给实现并行化的算法分析带来了难题。

4 内核态竞态漏洞检测

操作系统内核与用户态程序运行环境相比更加复杂,用户态竞态漏洞检测方法无法直接应用于内核态静态漏洞检测。一方面,内核中系统调用频繁,通过侵入性插桩记录共享资源操作轨迹的检测方法将导致大量额外开销;另一方面,内核程序语义更底层,以happen-before方法为例,很难从两个内核操作直接判断两者之间的逻辑关系,因为同一个内核线程的上下文执行程序有多种可能——用户模式进程代码、设备中段服务例程或是内核延迟过程调用[45]。正如文献[46]所论述的:通过理解内核中复杂的同步机制原语语义来推断happens-before关系或统计lock-set设置点都是相当繁重的工作。目前,内核态竞态漏洞检测的主要方法是对系统调用进行简单逻辑推理。

4.1 现有方法

现有内核态竞态漏洞检测方法主要分为两种,一种是针对文件系统特别是UNIX风格文件系统的TOCTTOU漏洞检测方法,另外一种是通过实现部分用户态竞态漏洞检测方法的轻量级方法。虚拟化技术出现之后,出现了基于虚拟化技术的竞态漏洞检测技术,在原有检测方法的基础上,对竞态漏洞检测提供了高效透明的数据提取支持。

文件系统的竞态检测较为简单,观测方便,特别的是,UNIX风格的文件系统中都有检查的过程,很多学者针对TOCTTOU进行了检测方法的研究。Tsyrklevich等人最先提出了一种通过监控系统调用检测利用竞态漏洞改写文件的方法[47]。Uppuluri在监视文件操作的基础上,设置时间窗口来进行竞态漏洞的检测[48],将虚警降到了3%,性能开销小于8%,是对检测性能和准确率的显著地提升。Wei等人基于Unix风格系统设计一种防御机制EDGI[49],通过枚举所有的文件系统调用并建立检测模型,防止攻击者在特定步骤对文件进行篡改。攻击者如果想要攻击,那么在TOCTTOU即检查和使用之间,需要修改文件路径名到逻辑磁盘的映射。EDGI通过在这个特定步骤防止篡改,即可完成检测。

因为内核态状态下很难实现完整的happens-before或者lock-set方法,研究人员通过实现部分的happens-before或者lock-set实现内核态竞态漏洞动态检测。例如,Jeter等人借鉴lock-set的思想,通过一个全局的锁来管理所有的共享内存进行检测[50]。Erickson等人则是借鉴了happens-before与采样的思想,设计了DataCollider[19]。DataCollider是一款优秀的内核态动态竞态漏洞检测工具,能够同时用于漏洞检测和漏洞挖掘,并且针对Windows 7的数据竞争进行了实现。DataCollider没有使用传统的程序分析方法,而是使用了硬件结构中的调试寄存器对复杂的内核代码进行了数据竞争的检测。DataColloder吸取了happens-before结合采样来优化性能的方法,通过采样少量的访存操作来降低系统运行时开销。DataCollider利用硬件断点采样一些内存访问指令,在这些指令断点触发的时候,取出其相应的访存地址,设置相应的数据断点,挂起线程,等待设置的内存断点被触发。一旦触发,则表示存在访问冲突,即数据竞争。这种方法在性能与准确率之间做了一个平衡,是一种优秀的内核态动态竞态漏洞检测方法,并且由于其采用的硬件断点,对上层操作系统具有一定的透明性,作者利用此方法挖出了若干Windows的竞态漏洞。但是由于该工具基于闭源的Windows操作系统,给其应用带来了不便。随后,Jiang等人将DataCollider进行了Linux版本的移植,并将之命名为DRDDR[30]。

现有竞态漏洞检测方法的瓶颈取决于内核态提取的数据效率低、语义低,VT-X硬件虚拟化技术出现之后其对客户机操作系统的高效透明的数据提取方式给现有的内核态竞态漏洞检测方法带来了很大的便利,有效地解决了效率问题。赵跃华,邓渊浩使用了VT-X硬件虚拟化技术进行了竞态漏洞检测的尝试[18]。其通过硬件虚拟化技术监控中断监视系统调用,设置关键内存缺页实现内存监控,又结合符号文件分析,从函数和栈中参数获取内存读写位置,通过判断两次对同内存访问的时间差判定内核竞态漏洞的存在性。使用硬件虚拟机化技术进行竞态漏洞检测,可以高效且透明地进行数据

采集和系统调用监控,给vector clock的生成提供了良好的支持,有效地解决了侵略性插桩的问题;但是使用硬件虚拟化技术,同样存在过于中断频繁,检测算法不宜复杂的问题。

4.2 未来发展

内核态程序运行数据提取的难度和性能要求是制约内核态竞态漏洞检测的主要原因。只有解决内核数据语义理解和数据分析效率等关键问题,内核竞态漏洞检测才可能获得突破性发展。

4.2.1 内核态数据提取方式优化

在内核态能够监控的语义层级低,而且系统调用频繁,频发触发检测事件对系统的性能开销过大。在未来的研究中,需要针对三个问题对内核态的数据提取进行优化,数据提取效率问题、数据提取透明度问题以及能够提取高语义层级数据的问题。

硬件虚拟化技术、虚拟机自省技术等虚拟机技术可以为数据提取的效率和透明度提供良好的解决方案。特别是在云虚拟平台,利用虚拟机技术可以在客户机不知情的情况下高效地提取出所需共享资源操作轨迹数据,为下一步分析提供良好的数据基础。但是使用此类方法提取的数据仍然是低语义级别的数据,仅为系统调用层级,使得检测方法的误报率高、检测漏洞的覆盖范围优先。

在未来的研究过程中,研发出新的内核层高语义数据提取方式是一个值得研究人员关心的方向,当能够提取的数据信息量足够支撑内核态运行用户态的检测方法,会对内核态竞态漏洞的检测准确性带来较大幅度的提升。在牺牲一部分透明性的前提下,通过侵略式插桩的方式实现某些关键函数的高语义级别的数据提取都是可行的方案。

4.2.2 低语义级别数据智能分析

研究低语义级别数据智能分析方法是解决内核态竞态漏洞数据语义级别低的方法之一。低语义级别数据分析可以从两个方向进行研究,从低语义级别数据还原高语义级别数据是方法之一,可以通过模式匹配、规则推理等方式利用现有的低语义级别数据还原出高语义级别数据[51],研究的关键问题是还原算法的设计、还原算法的效率和准确率;直接从低语义级别进行数据智能分析也是方法之一。文献[52]通过VT-X硬件虚拟化技术提取了Windows操作系统的内核对象分配、内核内存访问、线程调度和函数调用数据,并通过固定的规则从提取数据中分析漏洞是否触发,通过这种方式发现了45个Windows内核漏洞。这种方式数据提取效率快,但是现在的方法只能通过简单的规则限定检测漏洞触发,漏报率和误报率都很高。未来可以通过神经网络、深度学习等方式,将可以提取的数据类型作为特征,使用已有的漏洞提取数据作为学习样本进行深度学习,训练后通过神经网络等方式智能推理实现漏报率误报率较低的漏洞检测方式,检测出竞态漏洞等更为复杂的漏洞。

5 总结与展望

现有的竞态漏洞检测技术在用户态和内核态存在显著的差异。用户态竞态漏洞检测经过充分的发展已经逐步成熟,内核态竞态漏洞检测因为研究手段单一、语义层级低、性能开销大等因素尚未形成完善的竞态漏洞检测方案,但是涌现了许多具有参考价值的尝试和创新。总结研究人员对竞态漏洞检测的各类方法,可以对竞态漏洞检测技术的发展带来指引和启迪。

5.1 竞态漏洞检测方法总结

如表1列出的竞态漏洞检测工具所示,现有的用户态竞态漏洞检测工具种类按照检测方法分为happens-before类别和lock-set类别。Racetrack[4]、LiteRace[37]、Fasttrack[5]、PACER[6]属于happens-before类别,此类工具是在happens-before基础上通过取样的方式进行了性能优化;Eraser[11]是属于lock-set类别的检测工具。内核态竞态漏洞检测工具均是通过系统调用的监控实现检测,包括RPS[53],DRDDR[30],RACEZ[17]和HARD[14]。

表1 竞态漏洞检测工具汇总

工具	特权级	方法	性能	误报率	漏报率
Eraser	ring 3	lock-set	快	高	低
RPS	ring 0	detect system call	快	高	高
Racetrack	ring 3	happens-before	中	低	高
HARD	ring 0	hardware support	快	高	高
LiteRace	ring 3	happens-before	中	低	高
FastTrack	ring 3	happens-before	快	低	高
PACER	ring 3	track all accesses in sampling period	快	—	—
RACEZ	ring 3	lock-set	快	高	低
DRDDR	ring 0	detect system call	快	高	高

用户态的竞态漏洞检测方法趋于成熟,检测技术整体框架完善;内核态的竞态漏洞检测面临两个挑战:如何实现内核态系统数据的提取,并保证数据提取操作的稳定性、透明性和数据的高语义;如何实现适合内核态数据的高效数据分析方法。目前来说,硬件虚拟化技术可以较好地解决第一个挑战中的部分问题,能够透明且高效地从客户机中提取信息,但是信息的语义层级仍然较低,仍处于系统调用级别;解决第二个挑战的关键是能否基于机器学习设计出适合内核态数据的分析算法。

5.2 竞态漏洞检测方法展望

利用虚拟机化技术进行数据提取是未来竞态漏洞检测技术的发展趋势。虚拟机自省技术、硬件虚拟化技术、云虚拟化技术都可以有效地提高数据提取效率,并且使得提取数据的方式更加隐蔽,提取的数据更为真

实,在漏洞利用和漏洞检测的对抗过程中获得更大的优势。

利用机器学习的方式智能化分析数据之间的逻辑关系,定位操作与操作之间的矛盾之处,发现存在于程序之中的竞态漏洞是未来竞态漏洞数据的分析方法。神经网络、深度学习等机器学习方法能够准确、高效地进行竞态漏洞数据分析。未来可以利用已公开的漏洞生成训练数据,分析系统在训练完成后,对虚拟化技术提取的数据进行分析,实现智能化的逻辑漏洞检测,弥补现有漏洞检测技术中逻辑漏洞检测的短板,改善目前严峻的内核安全生态。

参考文献:

[1] Raheja S,Munjal G.Analysis of Linux kernel vulnerabilities[J].Indian Journal of Science and Technology,2016,9(48).

[2] Netzer R H B,Miller B P.What are race conditions?:Some issues and formalizations[J].ACM Letters on Programming Languages and Systems(LOPLAS),1992,1(1):74-88.

[3] Narayanasamy S,Wang Z,Tigani J,et al.Automatically classifying benign and harmful data races using replay analysis[C]//ACM Sigplan Notices,2007,42(6):22-31.

[4] Yu Y,Rodeheffer T,Chen W.Racetrack:Efficient detection of data race conditions via adaptive tracking[C]//ACM SIGOPS Operating Systems Review,2005,39(5):221-234.

[5] Flanagan C,Freund S N.FastTrack:Efficient and precise dynamic race detection[C]//ACM Sigplan Notices,2009,44(6):121-133.

[6] Bond M D,Coons K E,McKinley K S.PACER:Proportional detection of data races[C]//ACM Sigplan Notices,2010,45(6):255-268.

[7] Smaragdakis Y,Evans J,Sadowski C,et al.Sound predictive race detection in polynomial time[J].ACM Sigplan Notices,2012,47(1):387-400.

[8] Poluri S V,Ramanathan M K.Deterministic dynamic race detection across program versions[C]//2015 IEEE International Conference on Software Maintenance and Evolution(ICSME),2015:181-190.

[9] Wilcox J R,Finch P,Flanagan C,et al.Array shadow state compression for precise dynamic race detection(T)[C]//30th IEEE/ACM International Conference on Automated Software Engineering(ASE),2016:155-165.

[10] Huang J,Meredith P O N,Rosu G.Maximal sound predictive race detection with control flow abstraction[J].ACM Sigplan Notices,2014,49(6):337-348.

[11] Savage S,Burrows M,Nelson G,et al.Eraser:A dynamic data race detector for multithreaded programs[J].ACM Transactions on Computer Systems(TOCS),1997,15(4):391-411.

[12] Von Praun C,Gross T R.Object race detection[C]//ACM Sigplan Notices,2001,36(11):70-82.

[13] Nishiyama H.Detecting data races using dynamic escape analysis based on read barrier[C]//Virtual Machine Research and Technology Symposium,2004:127-138.

[14] Zhou P,Teodorescu R,Zhou Y.HARD:Hardware-assisted lockset-based race detection[C]//IEEE 13th International Symposium on High Performance Computer Architecture,2007:121-132.

[15] Serebryany K,Iskhodzhanov T.ThreadSanitizer:data race detection in practice[C]//Proceedings of the Workshop on Binary Instrumentation and Applications,2009:62-71.

[16] Jannesari A,Tichy W F.On-the-fly race detection in multithreaded programs[C]//Proceedings of the 6th Workshop on Parallel and Distributed Systems:Testing,Analysis,and Debugging,2008:6.

[17] Sheng T,Vachharajani N,Eranian S,et al.RACEZ:A lightweight and non-invasive race detection tool for production applications[C]//2011 33rd International Conference on Software Engineering(ICSE),2011:401-410.

[18] 赵跃华,邓渊浩.基于硬件虚拟化的内核竞态漏洞监测技术研究与实现[J].软件导刊,2015(5):161-164.

[19] Erickson J,Musuvathi M,Burckhardt S,et al.Effective data-race detection for the kernel[C]//Usenix Symposium on Operating Systems Design & Implementation,2010:151-162.

[20] Jimenez M,Papadakis M,Le Traon Y.Vulnerability prediction models:A case study on the linux kernel[C]//2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation(SCAM),2016:1-10.

[21] 倪昀泽.移动智能终端安全漏洞攻防技术综述[J].移动通信,2017,41(7):30-34.

[22] Payer M,Gross T R.Protecting applications against TOCTTOU races by user-space caching of file metadata[C]//ACM Sigplan Notices,2012,47(7):215-226.

[23] Wei J,Pu C.TOCTTOU vulnerabilities in UNIX-Style file systems:An anatomical study[C]//Conference on Usenix Conference on File & Storage Technologies,2015,29(8):12.

[24] Bratus S,D'Cunha N,Sparks E,et al.TOCTOU,traps,and trusted computing[C]//International Conference on Trusted Computing.Berlin Heidelberg:Springer,2008:14-32.

[25] Lamport L.Correctly executes multiprocess progranm[J].IEEE Transactions on Computers,1979,28(9).

[26] Li X,Chang X,Board J A,et al.A novel approach for software vulnerability classification[C]//2017 Annual

Reliability and Maintainability Symposium(RAMS),2017:1-7.

[27] Whyman P A.Automatic endpoint vulnerability detection of Linux and open source using the National Vulnerability Database[D].Colorado State University,2007.

[28] Koziol J,Litchfield D,Aitel D,et al.The Shellcoder's handbook[M].Indianapolis:Wiley,2004.

[29] Netzer R H B.Race condition detection for debugging shared-memory parallel programs[D].University of Wisconsin-Madison,1991.

[30] Jiang Y,Yang Y,Xiao T,et al.DRDDR:A lightweight method to detect data races in Linux kernel[J].The Journal of Supercomputing,2016,72(4):1645-1659.

[31] Jiang Y,Yang Y,Xiao T,et al.Kernel data race detection using debug register in Linux[C]//COOL Chips XVII,2014:1-3.

[32] Lamport L.Time,clocks,and the ordering of events in a distributed system[J].Communications of the ACM,1978,21(7):558-565.

[33] Flanagan C,Freund S N.Type-based race detection for Java[C]//ACM Sigplan Notices,2000,35(5):219-232.

[34] Gupta S,Sultan F,Cadambi S,et al.Using hardware transactional memory for data race detection[C]//IEEE International Symposium on Parallel & Distributed Processing,2009:1-11.

[35] Netzer R H B,Ghosh S.Efficient race condition detection for shared-memory programs with post/wait synchronization[D].University of Wisconsin-Madison,1992.

[36] Bhansali S,Chen W K,De Jong S,et al.Framework for instruction-level tracing and analysis of program executions[C]//Proceedings of the 2nd International Conference on Virtual Execution Environments,2006:154-163.

[37] Marino D,Musuvathi M,Narayanasamy S.LiteRace:effective sampling for lightweight data-race detection[C]//ACM Sigplan Notices,2009,44(6):134-143.

[38] Biswas S,Cao M,Zhang M,et al.Lightweight data race detection for production runs[C]//Proceedings of the 26th International Conference on Compiler Construction,2017:11-21.

[39] Witten I H,Frank E,Hall M A,et al.Data mining:Practical machine learning tools and techniques[M].[S.l.]:Morgan Kaufmann,2016.

[40] Kistler M,Brokenshire D.Detecting race conditions in asynchronous DMA operations with full system simulation[C]//2011 IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS),2011:207-215.

[41] Srivastava A,Giffin J.Automatic discovery of parasitic malware[C]//International Workshop on Recent Advances in Intrusion Detection.Berlin Heidelberg:Springer,2010:97-117.

[42] Garfinkel T,Rosenblum M.A virtual machine introspection based architecture for intrusion detection[C]//Proceedings of the Network & Distributed Systems Secuity Symposium,2003:191-206.

[43] 王丽娜,高汉军,刘炜,等.利用虚拟机监视器检测及管理隐藏进程[J].计算机研究与发展,2011,48(8):1534-1541.

[44] Singhal M,Kshemkalyani A.An efficient implementation of vector clocks[J].Information Processing Letters,1992,43(1):47-52.

[45] Gaur D,Connor P,Jenison L,et al.Driver having multiple deferred procedure calls for interrupt processing and method for interrupt processing:U.S.Patent Application 09/823,155[P].2001-03-29.

[46] Love R.Linux kernel development[M].[S.l.]:Novell Press,2005.

[47] Tsyrklevich E,Yee B.Dynamic detection and prevention of race conditions in file accesses[C]//Proceedings of Usenix Security Symposium,2003,4(2):17.

[48] Uppuluri P,Joshi U,Ray A.Preventing race condition attacks on file-systems[C]//Proceedings of the 2005 ACM Symposium on Applied Computing,2005:346-353.

[49] Wei J,Pu C.Modeling and preventing TOCTTOU vulnerabilities in Unix-style file systems[J].Computers & Security,2010,29(8):815-830.

[50] Jeter R E,Potter K H.Memory controller that tracks queue operations to detect race conditions:U.S.Patent 7,254,687[P].2007-08-07.

[51] Jiang X,Wang X,Xu D.Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security,2007:128-138.

[52] Pan J,Yan G,Fan X.Digtool:A virtualization-based framework for detecting kernel vulnerabilities[C]//26th {USENIX} Security Symposium({USENIX} Security 17),USENIX Association,2017.

[53] Park J,Lee G,Lee S,et al.RPS:An extension of reference monitor to prevent race-attacks[J].Advances in Multimedia Information Processing-PCM 2004,2005:556-563.