锁闭保护:基于程序行为分析的非预期执行攻击阻断

杨佳庚[1] 方滨兴[1),2)] 冀甜甜[2)] 张云涛[2)] 王 田[2)] 崔 翔[3)] 王媛娣[4)]

[1)](哈尔滨工业大学(深圳)计算机科学与技术学院 广东 深圳 518055)

[2)](北京邮电大学可信分布式计算与服务教育部重点实验室 北京 100876)

[3)](中关村实验室 北京 100194)

[4)](积至(海南)信息技术有限公司 北京 100029)

摘 要 投递恶意代码以调用安全敏感服务是网络攻击中实施窃取、损毁、致瘫攻击的必要行为,使网络空间面临严重威胁. 本文将此类攻击称为非预期执行攻击,现有的防御技术难以检测以合法载体实施的这类攻击. 本文提出了一种称为锁闭保护结构的安全防护机制,作为现有防御技术的补充和安全底线,是阻断恶意行为实施的最后一道防线. 通过分析目标程序针对安全敏感服务的预期行为,监控程序实际行为,阻断与预期行为不一致的服务执行,实现对非预期执行攻击的防御. 基于对影响服务行为的关键要素的观察,本文提出了锁闭保护模型,作为阻断非预期执行攻击的理论支撑. 然后,在Linux实验环境下实现了一个锁闭保护原型系统,使用真实的高级持续性威胁攻击样本、内核权限提升漏洞以及流行的应用程序进行了有效性验证,并评估了其产生的性能开销. 实验结果表明,该原型系统能成功抵御典型的非预期执行攻击,仅引入不超过5%的性能开销.

关键词 安全敏感服务;非预期执行攻击;锁闭保护;程序行为监控;攻击阻断

中图法分类号 TP309 **DOI号** 10.11897/SP.J.1016.2024.01697

Lockdown-Protection: Unintended Execution Attack Prevention Based on Program Behavior Analysis

YANG Jia-Geng[1)] FANG Bin-Xing[1),2)] JI Tian-Tian[2)] ZHANG Yun-Tao[2)]

WANG Tian[2)] CUI Xiang[3)] WANG Yuan-Di[4)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong* 518055)

[2)](*Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education, Beijing University of Posts and Telecommunications, Beijing* 100876)

[3)](*Zhongguancun Laboratory, Beijing* 100194)

[4)](*Geedge Networks Ltd., Beijing* 100029)

Abstract Delivering malware to invoke security-sensitive services is a necessary step in cyber-attacks to implement theft, destruction and denial-of-service attacks, putting cyberspace at serious risk. Malware that calls security-sensitive services performs sensitive operations such as file read or write, access control, and system management, posing a direct and significant threat to system security. For instance, in the WannaCry ransomware incident that began in 2017 involved attackers spreading ransomware to victims' devices through vulnerabilities. This ransomware encrypted and overwrote files by invoking the system's file writing services, aiming for ransom. In this paper, we define such attacks as unintended execution attacks, which are

收稿日期:2023-08-31;在线发布日期:2024-04-18. 本课题得到海南省方滨兴院士工作站资金资助. **杨佳庚**,博士研究生,主要研究方向为网络安全. E-mail: yangjiageng@stu.hit.edu.cn. **方滨兴**(通信作者),博士,教授,博士生导师,中国工程院院士,主要研究领域为计算机体系结构、计算机网络、网络安全. **冀甜甜**(通信作者),博士,博士后,主要研究方向为网络安全. E-mail: jitiantian0728@bupt.edu.cn. **张云涛**,博士,博士后,主要研究方向为主要研究方向为网络安全. 王田,博士研究生,主要研究方向为主网络安全. **崔 翔**,博士,教授,博士生导师,主要研究领域为网络安全. **王媛娣**,学士,主要研究方向为网络安全.

difficult to detect by existing techniques. The unintended execution attacks can lead to system crashes, data leakage, or destruction, with serious implications for personal privacy, business operations, and national security. Analyzing the MITRE ATT&CK attack matrix, we conclude that strategies closer to the end of the attack chain are more technically necessary. The invocation of security-sensitive services with high access and execution privileges at the end of the attack chain is a prerequisite for achieving attack objectives, making unintended execution attacks inevitable.

Existing defense techniques primarily focus on mitigating and detecting the process of malicious code injection and exploitation, preventing attackers from executing malicious code injection, control flow hijacking, and privilege escalation through code injection or exploiting vulnerabilities in programs. However, with the emergence of new types of vulnerability exploitation techniques, such as code reuse attacks, and considering the performance overhead of mitigation techniques in commercial systems, practical defense solutions are vulnerable to being bypassed. In particular, when attackers enter the system in an unknown manner and run malicious software with privileged accounts, existing defense techniques fail to detect such unintended execution attacks in real time. Therefore, researching countermeasures against unintended execution attacks, serving as the last line of defense to block malicious activities by attackers, is of significant importance for ensuring the baseline security of critical information systems. By a deep understanding of the mechanisms and behavioral characteristics of unintended execution attacks, this paper designs and implements a lockdown protection mechanism, which complement existing defense techniques and act as a security bottom line, which is the last line of defense to intercept malware execution. This mechanism includes three key steps to detect and block unintended execution attacks: (1) expected behavior analysis, (2) actual behavior monitoring, and (3) blocking unexpected behaviors. Specifically, the lockdown protection structure analyzes the expected behaviors of a target program with respect to security-sensitive services at compile time, monitors the actual behaviors of the program at runtime, and verifies that the actual behavior of the program matches the expected behavior prior to service execution. If inconsistencies are detected, the invocation is identified as an unintended execution attack. This defense approach is rooted in the observation that the program's execution behavior during an unintended execution attack deviates from its expected behavior. Based on the observation of critical elements that could affect service behavior, we propose a lockdown-protection model as a theoretical model for defending against unintended execution attacks. We then implement a prototype of lockdown-protection structure on Linux platform. We utilize real-world APT malware, kernel privilege escalation exploits, and popular applications to evaluate the effectiveness of our prototype. We also evaluate the performance overhead of the prototype. Experiment results show that our prototype is effective in defending against typical unintended execution attacks, introducing a performance overhead of less than 5%.

Keywords security-sensitive service; unintended execution attack; lockdown-protection; monitoring program behavior; attack prevention

1 引 言

近年来,网络空间已延伸到生产生活的方方面面,而网络空间面临的安全威胁也日益严峻. 例如,在俄乌冲突中,数据擦除软件(wiper)被大量投递到敌方关键信息基础设施中,在非预期执行后破坏系统关键数据导致服务中断. 安全敏感服务是指操作

系统所提供的具有直接访问或修改系统关键资源、执行特权操作的接口或系统调用,如与文件操作相关的 write 系统调用,与程序执行相关的 execve 系统调用等. 这些服务所执行的敏感操作,如文件读写、访问控制、系统控制等,与系统的安全性有着直接且重要的联系. 梳理历年的重大安全事件,本文发现投递恶意代码以调用安全敏感服务,使其执行在预期之外的状态下,是实现攻击目标的重要途径. 例如,始于 2017 年的 WannaCry 勒索事件[1],攻击者通过漏洞传播 WannaCry 勒索软件至受害者设备,该勒索软件通过调用系统的写文件服务加密并覆盖文件以实现勒索目的. 本文将这类攻击定义为非预期执行攻击,它能导致系统崩溃、数据泄露或数据损毁等危害,对个人隐私、企业运营和国家安全造成严重影响.

根据对 MITRE ATT&CK 攻击矩阵①的分析,在攻击链中越靠后的策略,其技术必要性越大. 而在攻击链末端调用具有高访问和执行权限的安全敏感服务是实现攻击目标的必要条件,因此非预期执行攻击具有必然性. 根据攻击必然性设计一种底线确保的防护机制是十分必要且可行的,有望实现对非预期执行攻击的完全阻断. 因此,本文通过深入理解非预期执行攻击的机制和行为特征,设计并实现了一种锁闭保护机制,可实现在攻击威胁到来时将系统置于一种安全状态,阻止非预期执行的发生. 这一机制作为现有防御技术的补充和安全底线,不仅可以为系统提供强大的防御能力,还可减少潜在的损失,确保信息安全和数据保护.

按照攻击实施方法的差异,本文将非预期执行攻击分为代码注入和漏洞利用两类攻击场景. 针对这两类攻击场景,研究人员提出了许多缓解、阻断非预期执行攻击实施过程的防御方法. 为防止代码注入类攻击,提出了数据执行保护、栈不可执行、指令随机化等技术. 数据执行保护[2](Data Execution Prevention,DEP)、栈不可执行等通过将内存的特定区域标记为仅用于数据,防止攻击者通过数据输入注入恶意代码. 指令随机化技术[3]通过对执行的指令进行随机化而减轻 shellcode 注入的攻击行为. 虽然上述的防御技术有效地阻断了攻击者以非法途径向目标程序中注入恶意代码的行为,但攻击者仍可以通过其他载体以合法方式实现代码注入,此时尚不存在有效的防御手段. 例如,使用带有恶意附件的钓鱼邮件或者恶意软件植入等攻击载体向系统内注入恶意代码,它仍然是 APT 攻击常用的攻击手段之一.

为防御基于漏洞利用的攻击行为,地址随机化、控制流完整性检测等漏洞利用缓解技术在过去三十年中被陆续提出. 一种漏洞利用缓解技术的思路是在进程内存中引入随机性和秘密,增加攻击者漏洞利用的难度,例如地址空间布局随机化(Address Space Layout Randomization,ASLR)②. 但这类防御方案面临着信息泄露攻击所导致的去随机化和暴露密钥进而被绕过的风险,无法保证对具有完全内存控制权的攻击者的有效防御. 另一类缓解技术的思路是检测程序执行过程是否与编译时确定的控制流/数据流一致,例如控制流完整性检测[4](Control Flow Integrity,CFI). 但这类防御方案往往面临着过高的性能开销,部署在商用系统的粗粒度方案却因检测粒度过粗而存在被绕过的风险. 例如,CFI 是一种防御控制流劫持的有效方法,但为减少其带来的性能开销,部署在 Windows 上的 CFI 方案——CFG(Control Flow Guard)只进行粗粒度的完整性检查,而存在被绕过的风险[5].

综上所述,现有防御技术关注的重点是恶意代码注入过程和漏洞利用过程的缓解和检测,防止攻击者通过代码注入或利用程序中的漏洞实施恶意代码注入、控制流劫持、权限提升等攻击行为. 但由于代码复用攻击等新型漏洞利用技术的不断涌现以及缓解技术在商用系统实施时要考虑的性能开销问题,实际应用的防御方案均存在相关的绕过技术. 特别是当攻击者以未知方式进入系统后,以特权账户身份运行恶意软件实施攻击行为时,现有防御技术无法实时检测这类非预期执行攻击. 因此,研究针对非预期执行攻击的对抗技术,作为阻断攻击者实施恶意行为的最后一道防线,对于确保重要信息系统的底线安全具有重要意义.

基于对网络攻击中调用安全敏感服务实施非预期执行攻击的观察,本文提出了一种称为锁闭保护结构的安全防护机制. 锁闭保护结构对非预期执行攻击的检测和阻断分为三个关键步骤:1)预期行为分析;2)实际行为监控;3)非预期行为阻断. 具体而言,锁闭保护结构在编译时分析目标程序针对安全敏感服务的预期行为,在运行时监控目标程序的实际行为,在服务执行前检验程序的实际行为是否与

① MITRE ATT&CK. https://attack.mitre.org 2023,8,15

② Address space layout randomization. https://pax.grsecurity.net/docs/aslr.txt

预期行为相一致. 如果不一致,则认为此次调用行为是非预期执行攻击.

锁闭保护结构的防御思路源于非预期执行攻击时程序执行行为与预期行为不一致的观察,本文归纳总结了影响安全敏感服务行为的关键要素,提出了锁闭保护模型,作为锁闭保护结构检测阻断非预期执行攻击的理论支撑.

综上,将本文的贡献总结如下:

(1)通过分析网络攻击中对安全敏感服务的调用行为,本文定义了面向安全敏感服务的非预期执行攻击,从而更具体地刻画这类攻击行为;

(2)通过观察非预期执行攻击对程序行为的影响,归纳总结了用于映射程序行为的关键要素,在此基础上提出了锁闭保护模型,为检测阻断非预期执行攻击提供了防御思路;

(3)基于锁闭保护模型,本文提出了一种自适应、跨平台的锁闭保护结构,为防御各类场景下的非预期执行攻击提供了一种可行的解决方案;

(4)在 Linux 系统上实现了一个锁闭保护原型系统,使用真实 APT 攻击样本、内核权限提升漏洞等典型攻击场景以及流行的应用程序进行了有效性验证及性能测试. 实验结果表明,该原型系统能够检测并成功防御两类典型的非预期执行攻击,仅引入不超过 5%的额外性能开销;

本文的第 2 节总结非预期执行攻击,给出本文的威胁模型;第 3 节提出了锁闭保护模型作为防御非预期执行攻击的总体思路;第 4、5 节对锁闭保护结构的设计与 Linux 环境下锁闭保护原型系统的实现进行了讨论;第 6 节介绍了测试数据集和实验评估;第 7 节概述了防御非预期执行攻击的相关工作;最后对锁闭保护的防御方案进行总结与展望.

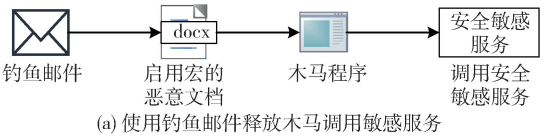
2 非预期执行攻击和威胁模型

2.1 非预期执行攻击

为理解非预期执行攻击的实施方法,本文分析了 MITRE ATT&CK 攻击矩阵中与非预期执行攻击紧密相关的三类技术(初始访问、执行、权限提升)并统计相关的 32 项子技术在实施非预期执行攻击的方法可分为代码注入(24/32 项)和漏洞利用(8/32 项)[6]. 因此,按照实施方法,非预期执行攻击可分为代码注入和漏洞利用两大类攻击场景.

代码注入类攻击场景多发生在具有用户输入功能、可执行系统命令的应用程序中,例如 Web 应用程序、数据库应用等;或其他解析并执行可能携带恶意代码的文件载荷的应用程序中,例如启用宏的文档(如图 1 (a))、恶意网页链接等. 其原理是攻击者向目标应用程序提供恶意输入数据,应用程序未对输入进行充分的过滤和验证,导致恶意输入数据被应用程序解释为系统命令(例如,SQL 注入[7]);或应用程序在解析文件时触发并执行了恶意代码,最终导致恶意代码以合法应用程序身份调用系统内的安全敏感服务.

漏洞利用类场景多发生在存在内存损坏漏洞的应用程序,如以 C/C++等内存不安全语言编写的应用程序. 其原理是利用程序的内存损坏漏洞,例如缓冲区溢出漏洞,来修改程序执行过程的控制数据(如函数返回地址、函数指针等)并按照攻击目的完成相关内存布局,这使得攻击者能够将程序的执行顺序跳转至非预期的、安全敏感服务的调用位置,执行符合其攻击目的的安全敏感服务.



(a) 使用钓鱼邮件释放木马调用敏感服务

```
movl   %esp,%ecx                pushl  $0x3f
movb   $0x66,%al                popl   %eax
int    $0x80      ❶             int    $0x80      ❹
……                              ……
pushl  $0x66                    pushl  %eax
popl   %eax                     pushl  $0x68732f2f
int    $0x80      ❷             pushl  $0x6e69622f
……                              ……
movb   $0x04,%bl                movl   %esp,%ecx
movb   $0x66,%al                movb   $0x0b,%al
int    $0x80      ❸             int    $0x80      ❺
```

(b) msf框架建立reverse_shell的shellcode关键代码

图 1 两种非预期执行攻击场景实例

如图 1(b)所示,本文以 Metasploit 框架生成的建立反向 shell 的载荷为例,说明攻击者如何实施非预期执行攻击以完成反向 shell 的建立的. 标签 1 和 2 处调用安全敏感服务 *socketcall* 建立 socket 连接;标签 3 处调用安全敏感服务 *sendto* 发送数据;标签 4 处调用 *dup*2 系统调用将标准输入流、标准输出流重定向到 socket 客户端;在标签 5 处调用安全敏感服务 *execve* 系统调用来执行"/bin/bash"程序,达到运行一个 shell 程序的效果. 因此,在漏洞利用代码中,安全敏感服务是攻击者实现攻击目标的主要途径,通过非预期执行攻击可实现远程控制、数据窃取等严重威胁信息系统安全性的攻击行为.

2.2 威胁模型

本文仅涉及用户态下针对安全敏感服务的非预

期执行攻击,即攻击者操纵应用程序来执行偏离应用程序原始设计目标的安全敏感服务. 这类攻击的目的是使应用程序执行在安全、预期的执行上下文中未涉及的对安全敏感服务的调用. 此类攻击主要包括两类场景:1)攻击者注入的恶意代码调用安全敏感服务;2)攻击者利用漏洞使应用程序的执行跳转至给定上下文中一个非预期的对安全敏感服务的调用处,例如攻击者劫持控制流绕过应用程序中对system()调用所对应的安全过滤代码而直接以攻击者提供的参数调用 system(). 仅涉及应用程序中所属数据的数据流攻击(例如,面向数据编程——DOP "Data-Oriented Programming")和仅涉及合法程序的 LotL(Living-off-the-Land)类攻击不在本文的讨论范围.

在本文中,本文假设一个强大但现实的攻击者:具有对应用程序进程内存的完全控制权,可以实施基于代码注入的攻击和基于漏洞利用的攻击. 攻击者可以利用应用程序的零日漏洞而实施任意程序内存的读写. 但是假定操作系统、硬件、应用程序加载过程是可信的,则攻击者无法对可执行内存进行写入,因为可执行内存对应的页被操作系统和内存管理单元标记为可读且不可写. 这些假设确保了在编译时插桩的锁闭保护结构代码的完整性,锁闭保护功能的可用性,这些是模拟攻击者利用零日漏洞控制系统的攻击场景的合理假设.

3　防御思路及模型

3.1　影响服务行为的三要素

本论文所讨论的针对安全敏感服务的非预期执行攻击的实质是攻击者为实现攻击目标而非法调用安全敏感服务的行为. 例如,通过代码注入的方式调用 system 函数执行攻击命令. 防御此类攻击的关键是区分安全敏感服务的行为是应用程序为功能实现而进行的预期调用,或在应用程序预期功能之外的行为. 以此为基本思想,本文提出了锁闭保护模型,旨在区分安全敏感服务的某次执行是应用程序的正常行为或异常行为(非预期执行攻击).

如图 2 所示,安全敏感服务的行为取决于三个元素:服务调用者、服务定义者以及服务的上下文. 以下分别介绍这三个元素,并说明三者的关系.

(1)服务调用者:调用某个指定服务的实体,调用行为可能是直接调用(例如,call funcA)也可能是间接调用(例如,call addr1/reg1);

(2)服务定义者:用于实现服务指定功能的实体,通常是一个不可分割的、连续的代码片段(例如,函数或系统调用);

(3)服务的上下文:服务调用者和定义者执行时的必要环境和相关要素(例如,调用者和定义者的函数栈帧、输入变量等).

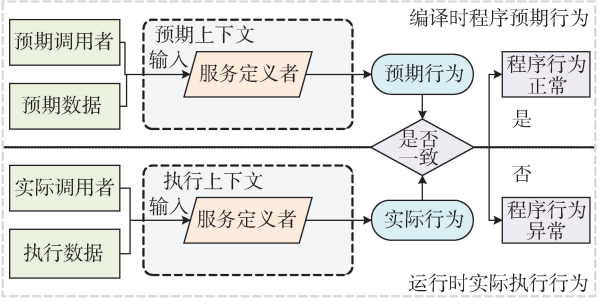


图 2　面向安全敏感服务的锁闭保护模型

在应用程序的进程空间中,程序执行至代码段中服务调用者处,服务调用者根据服务上下文的信息跳转至服务定义者处,服务定义者根据服务上下文的信息执行指定的预期功能. 即服务上下文会影响服务调用者和服务定义者的行为,服务调用者决定了服务定义者在怎样的服务上下文中执行,而服务调用者和定义者的行为也会影响服务上下文,三者相互作用共同决定了服务的实际行为.

3.2　锁闭保护模型

鉴于非预期执行攻击对应用程序和系统行为带来的严重威胁,针对此类威胁的检测和阻断机制具有十分重要的现实意义. 现有的漏洞利用缓解技术,例如数据执行保护和控制流完整性检测技术,它们通过限制程序内存的写入区域和执行区域的方式来缓解非预期执行攻击. 但是,缓解技术为了实现防御和部署的通用性和适配性而在部署实现上采用折衷的思路,未聚焦于非预期执行攻击中最关键的攻击目的——安全敏感服务的调用和执行行为. 因此,这些缓解技术都存在一定的绕过方式.

本文提出了一种称为锁闭保护的防御模型,通过观测应用程序对安全敏感服务的调用行为,来检测和阻断攻击者操纵应用程序实施非预期执行攻击的威胁行为. 锁闭保护模型基于以下两个观察:

(1)应用程序对安全敏感服务的调用行为是由应用程序的预期功能逻辑确定的,服务和程序在特定的执行上下文具有固定的调用和执行模式;

(2)攻击者实施非预期执行攻击时,程序对安全敏感服务的调用行为会偏离正常的服务调用和程序

执行模式,或偏离特定的执行上下文.

因此,本文提出的锁闭保护模型旨在观测应用程序在运行时的实际执行行为是否偏离编译时确定的程序预期行为,从而检测和阻断非预期执行攻击.如图 2 所示,锁闭保护模型基于前文提出的三要素来表示应用程序在编译时的预期行为和运行时的实际行为.具体而言,在编译时,可通过提取出应用程序的工作逻辑中的安全敏感服务的调用者和定义者,在运行时观察到应用程序执行预期功能时执行上下文的元数据.由程序代码逻辑中包含的预期调用者、服务定义者和预期上下文对应的元数据共同构成了程序的一次预期行为.

类似地,在运行时,锁闭保护模型监控安全敏感服务的实际调用者和其所处的执行上下文来映射成应用程序的一次实际行为,通过判断实际行为是否偏离模型在编译时所确定的预期行为,从而检测应用程序是否正在遭受非预期执行攻击.如果程序行为异常,系统将中断安全敏感服务定义者的执行,进而阻断非预期执行攻击.这样,即使攻击者利用内存损坏漏洞控制了应用程序的内存,也无法操纵应用程序实施偏移应用程序正常逻辑的非预期执行攻击行为.

4 设 计

要设计和实现基于锁闭保护模型的针对非预期执行攻击检测和阻断系统,需要解决以下问题:

(1)如何自动提取应用程序对安全敏感服务的调用逻辑和其预期执行模式;

(2)如何在应用程序运行时监控和记录与敏感服务执行行为相关的元数据;

(3)如何检测应用程序实际执行过程中偏离其预期行为的非预期执行行为.

为解决上述问题,本文提出了锁闭保护结构,这是一种基于锁闭保护模型的检测并阻断可疑的、非预期行为的安全防护方案.如图 3 所示,锁闭保护结构由三个主要组件组成:预期行为分析模块、执行行为监控模块和非预期行为检测模块.

(1)预期行为分析模块.应用程序的预期行为由实现其预期功能的代码逻辑确定.具体而言,预期行为由代码逻辑中影响服务行为的三要素(即服务调用者、定义者和服务的上下文)确定.获取这三个要素需要先在程序中标定这三要素的位置.因此,此模块在编译时通过静态分析的方式识别影响预期行为的代码位置,并插桩用于获取并记录元数据合法数值的代码片段(在下文中,称为“应用壳”).

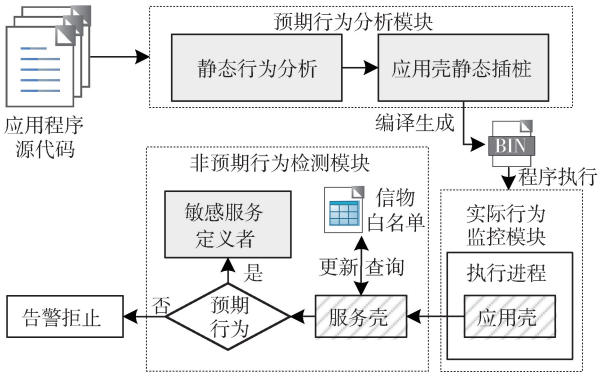


图 3 锁闭保护结构总体架构

(2)执行行为监控模块.该模块为插桩到应用程序中的应用壳,其目的是用于监控、捕获表示预期行为和实际行为的三要素.具体的三要素为安全敏感服务调用者的地址、所在的代码段上下文信息和调用者与定义者运行时的执行上下文元数据,并使用动态哈希算法将三要素映射为唯一的哈希值.

(3)非预期行为检测模块.该模块为插桩到安全敏感服务定义者处的代码片段(在下文中,称为“服务壳”),其目的是检测应用程序对安全敏感服务的调用行为是否偏离其预期行为.服务壳将预期行为三要素所映射的唯一哈希值作为安全信物,接收来自应用壳的实际行为三要素所映射的唯一哈希值并将其与安全信物进行比较,如果一致则放行此次执行;否则,认为此次对安全敏感服务的调用行为是可疑的、非预期的调用行为,服务壳拒止此执行行为并告警.

本文将锁闭保护系统插桩在用户态和内核态的代码片段称为“应用壳”和“服务壳”.这种方式与传统的软件加壳(Software Packing)技术在系统实现上有一定的相似性,而在功能与安全机制上有着明显的差异.从系统实现的角度讲,二者都涉及到在不改变应用程序原有逻辑的情况下添加额外的保护代码,从而提高程序的安全性.从功能目标的角度讲,“应用壳”与“服务壳”侧重于实时监控应用行为,防止可疑的异常行为;软件加壳侧重于防止应用被破解或非法分发.从安全机制的角度讲,“应用壳”根据影响服务行为的三要素生成安全信物,再由“服务壳”进行校验来判断偏移预期行为的状态,软件加壳则是使用加密、混淆和虚拟化技术使得对程序的分析和破解变得困难.

4.1　预期行为分析

非预期执行攻击的目标是实施偏离应用程序预期行为的、对安全敏感服务的非预期调用行为,因此分析应用程序对安全敏感服务的预期调用行为是识别非预期执行攻击的关键. 在应用程序中,对安全敏感服务的调用行为可分为直接调用行为和间接调用行为. 对安全敏感服务的直接调用行为指在程序中使用安全敏感服务的函数名称并提供所需的参数来执行该函数的过程. 在直接调用的过程中,程序的控制流将从当前执行位置转移到函数内部执行,并在函数执行完毕后返回结果及控制流. 如图 4(a)所示,第一行表示的是 C 语言格式程序对安全敏感服务 *write* 的直接调用行为,通过直接调用函数名称"*write*"并提供相应的实参来调用并执行 *write* 函数;第 2 行表示的是该函数调用行为转换成 LLVM IR 格式下的表现形式,由 *call* 指令直接调用函数名称"*@write*"并提供相应的实参来调用安全敏感服务. 因此,基于上述对安全敏感服务直接调用行为的观察,本文通过识别函数名称标识符以及调用函数类型是否与安全敏感服务相一致的思路来确定针对安全敏感服务的直接调用行为.

```
1: count = write(fd, buffer, strlen(buffer));
2: %20 = call i64 @write(i32 %16, i8* %17, i64 %19)
```

(a) C语言格式与LLVM IR格式下对安全敏感服务*write*的直接调用行为

```
1: typedef ssize_t (*write_func_ptr)(int, void*, size_t);
2: write_func_ptr ptr = write;
3: count = ptr(fd, buffer, strlen(buffer));
4: write_func_ptr addr = &write;
5: count = addr(fd, buffer, strlen(buffer));
```

(b) C语言格式下两种对安全敏感服务*write*的间接调用行为

```
1: store i64 (i32, i8*, i64)* @write, i64 (i32, i8*, i64)** %6, align 8
2: %41 = load i64 (i32, i8*, i64)*, i64 (i32, i8*, i64)** %6, align 8
3: %42 = load i32, i32* %4, align 4
4: %43 = getelementptr inbounds [64 x i8], [64 x i8]* %2, i64 0, i64 0
5: %44 = getelementptr inbounds [64 x i8], [64 x i8]* %2, i64 0, i64 0
6: %45 = call i64 @strlen(i8* %44) #6
7: %46 = call i64 %41(i32 %42, i8* %43, i64 %45)
```

(c) LLVM IR语言格式下对安全敏感服务*write*的间接调用行为

图 4　针对安全敏感服务 *write* 的常见调用行为

与直接调用不同,对安全敏感服务的间接调用不直接使用函数名称来调用对应的函数,而是使用指向该函数的函数指针或者保存该函数地址的变量(例如,函数指针数组)作为调用目标. 间接调用可以在运行时动态地决定要调用的函数,在回调函数、动态库加载等场景下非常有用. 但因间接调用在编译时的不可见性,阻碍了分析针对安全敏感服务的间接调用行为. 本文主要考虑 C 语言中两种最为常见且覆盖了绝大多数情况的函数间接调用行为,即通过函数指针的间接调用和通过函数地址的间接调用. 如图 4(b)所示,第 1～3 行展示了 C 语言格式程序使用函数指针对安全敏感服务 *write* 的间接调用行为,指针变量 *ptr* 指向 *write* 函数处,程序通过传递实参至 *ptr* 并调用指针 *ptr* 的形式实现的间接调用;第 4～5 行展示的是程序使用存储 *write* 函数地址的变量 *addr*,通过传递实参至 *addr* 并调用地址变量 *addr* 而实现的间接调用.

在上述的两种间接调用方式中,程序并不是通过调用 *write* 函数名称的方式实现的调用行为,而是通过与 *write* 函数相关联的变量实现的调用,因此称为间接调用. 虽然这两种调用方式在 C 语言中的语法格式是不同的,但这两类间接调用方式在 LLVM IR 格式下,它们的表现形式是相同的.

图 4(c)展示了在 LLVM IR 格式下对安全敏感服务 *write* 的间接调用过程,共涉及三个动作:(1)相关函数变量的初始化与赋值,在第 1～2 行,程序将 *write* 函数变量存储(store 指令)到寄存器%6 中,然后将对应的变量从寄存器%6 加载(load 指令)到寄存器%41 中(即寄存器%41 是 *write* 函数的一个别名);(2)准备、加载函数调用所需的实参,在第 3～6 行,程序初始化并调用 *write* 函数所需的三个实参(i32 类型的寄存器%42,i8 数组类型的寄存器%43,以及 i64 类型的寄存器%45);(3)间接调用敏感服务,在第 7 行,程序使用 *call* 指令调用步骤(1)中加载到寄存器%41 中的函数变量,并将步骤(2)中加载的三个变量作为实参传递到此次调用.

基于上述对 LLVM IR 中安全敏感服务间接调用过程的观察,本文通过以下三个步骤在编译时识别潜在的对安全敏感服务的间接调用:(1)识别间接函数调用,确定 call 指令使用函数变量而非函数名称进行调用;(2)判断此次间接函数调用所加载的参数类型是否与安全敏感服务的形参类型所兼容;(3)函数变量的数据依赖性分析,通过上下文回溯的方式判断此次调用的函数变量是否为安全敏感服务的别名. 如果满足上述三个条件,则确定此调用行为是针对安全敏感服务的间接调用行为.

综上所述,本文基于对安全敏感服务直接调用和间接调用的观察,设计了面向应用程序安全敏感服务调用的预期行为分析算法. 如算法 1 所示,预期行为分析模块首先遍历程序的所有指令,目的是寻找所有 call/jmp 类指令(第 2～3 行). 如果跳转指令的目标属于安全敏感服务集合,则判定其为直接调用安全敏感服务行为,在此调用前注入应用壳(第 4～5 行). 如果跳转指令的目标是变量,则此处

调用为间接调用,将该指令加入到间接调用指令集合$\mathbb{C}$中(第 7～8 行),以进行间接调用目标分析. 遍历间接调用指令集合$\mathbb{C}$中的所有调用,判断此次间接调用所加载的参数类型是否与安全敏感服务的形参类型所兼容(第 14 行). 如果兼容,则进行更复杂的别名分析,判断此次间接调用的跳转目标是否为安全敏感服务的一个别名(第 15 行). 如果是,则判定其为间接调用安全敏感服务行为,在此调用前注入应用壳(第 16 行). 通过上述的算法,预期行为分析模块向所有涉及安全敏感服务的调用处注入应用壳,从而标记了应用程序针对安全敏感服务的预期调用行为.

算法 1. 应用程序的预期行为分析算法.

输入:程序源代码 I,安全敏感服务 S

输出:插桩后应用程序

```
1.  C=Ø
2.  FOR i∈I
3.    IF IsCallInst(i)
4.      IF i.dest∈S
5.        InstrumentCheck(i)//直接调用行为
6.      ELSE
7.        IF IsVar(i.dest)
8.          C←i
9.        ENDIF
10.     ENDIF
11.   ENDIF
12. ENDFOR
13. FOR  j∈C
14.   IF IsCompatbile(j)
15.     IF IsAlias(j.dest)
16.       InstrumentCheck(j)//间接调用行为
17.     ENDIF
18.   ENDIF
19. ENDFOR
```

4.2 实际行为监控

锁闭保护结构通过监控目标程序的运行时特征进行比对来检测非预期执行攻击. 如前所述,本文将服务调用者、服务定义者与服务的执行上下文作为表示程序预期行为的三元组. 针对确定的服务定义者(安全敏感服务本身),本文在编译时静态分析了服务定义者所对应的所有服务调用者,并在调用者处插桩了应用壳. 在本节中,选取敏感服务的随机偏移处的代码段上下文和运行时函数调用栈的栈帧内容作为安全敏感服务的执行上下文. 值得注意的是,锁闭保护模型所定义的执行上下文可以包含任意与敏感服务执行相关的变量,不限于本文所选取的执行上下文信息. 而不同的执行上下文信息会为锁闭保护模型带来不同的安全属性,例如选择控制流敏感的分支变量作为执行上下文会为锁闭保护模型引入控制流完整性的安全属性.

综上,服务调用者处的应用壳采集在运行时调用栈的元数据确定调用者位置,根据随机的偏移量获取随机的代码段上下文元数据. 应用壳使用上下文敏感的安全信物生成算法将上述元数据通过哈希方案映射为唯一的安全信物,此信物值作为表示程序中一次确定性安全敏感服务调用行为的标识符,与敏感服务的预期行为是一一对应的. 用于信物生成的元数据会随着应用壳的插桩位置的变化而产生差异性,因此所生成的安全信物是上下文敏感的. 具体而言,不同的应用壳使用算法 2 生成的安全信物是不同的,而同一个应用壳在不同的栈帧中所生成的安全信物也是不同的,所以安全信物可作为应用程序内不同行为调用者的唯一标识符.

算法 2. 上下文敏感的安全信物生成算法.

输入:应用壳检查器 *Checker*

输出:安全信物 *Key*

```
1.  S=Ø
2.  call_addr=ReturnAddr(Checker)
3.  FOR i∈ {call_addr-16, call_addr}
4.    S←i
5.  ENDFOR
6.  codeseg_addr=_start()
7.  <D, L>=Rand()
8.  FOR j∈{codeseg_addr+D, callseg_addr+D+L}
9.    S←j
10. ENDFOR
11. IF L<64
12.   Fill(S)
13. ENDIF
14. Key=SHA1(S)
```

算法 2 展示了上下文敏感的安全信物生成算法的整体流程. 算法 2 的输入为当前执行安全信物生成的应用壳检查器 *Checker*,用于获取当前栈帧的返回地址、确定对应的敏感服务调用者和获取调用者上下文信息(第 2～5 行). 然后,用户壳随机生成代码段偏移量 D 和上下文长度数据 L,并在相对于代码段基址偏移量为 D 处取长度为 L 的代码段上下文,代码段基址 *codeseg_addr*+D+L 的取值不应超出代码段范围(第 6～8 行). 如果 L 的长度小于 64 字节,算法 2 会填充元数据集合 S 至 80 个字节. 最后,算法 2 调用 *SHA*1 哈希方案计算元数据

集合 S 所映射的哈希值作为安全信物. 算法 2 采用的双随机化因素方案确保了足够的熵变,而产生安全信物的元数据是上下文敏感的编译时信息(代码段上下文)和运行时信息(检查器调用栈上下文),这确保了安全信物的不可逆和不可伪造的安全属性.

以图 5 中对 write 服务的直接调用行为为例,根据算法 2,将安全信物的计算与使用过程展示如下:首先,根据栈帧信息获取调用位置(图 5(a)第 4 行)上下文信息(图 5(b)中下划线标识的 16 个字节). 然后,随机生成代码段偏移量 D(示例中为 0x1d0)和代码段上下文长度 L(实例中为 0x40),根据 D 和 L 选取代码段上下文(图 5(c)中下划线标识的 64 个字节). 最后,使用 SHA1 算法计算上述 80 个字节的数据对应的哈希值作为安全信物,并通过安全通道传输至服务壳处.

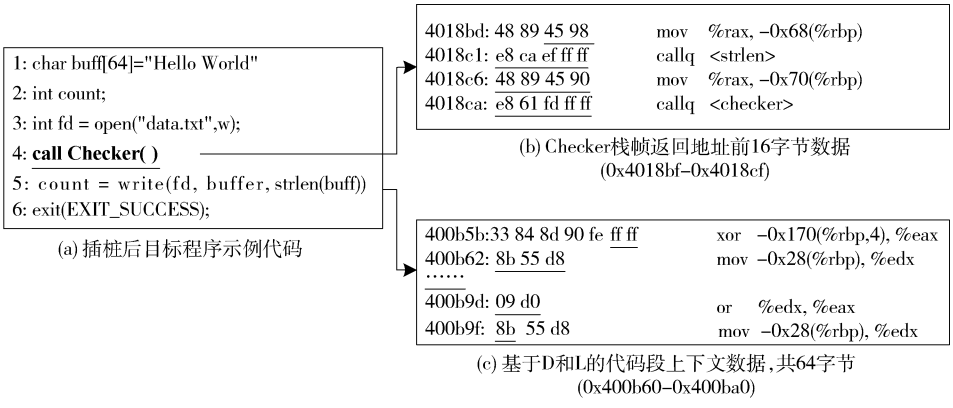


图 5　安全信物计算实例

4.3　非预期行为检测

非预期行为检测模块比较应用壳在运行时监控的实际行为是否与确定的应用程序预期行为相一致,进而判断是否有偏离程序预期行为的非预期执行攻击发生. 与应用壳监控运行时实际行为并计算安全信物的方案类似,服务壳的作用是计算调用行为所对应的确定性预期行为的安全信物、校验此信物与应用壳所传递的实际行为信物是否相同,并根据校验结果选择放行此次调用行为或告警.

在锁闭保护结构中,服务壳以内核态 hook 的方式阻断针对安全敏感服务的非预期调用并确保了锁闭保护结构的不可绕过属性. 位于用户态的应用壳在计算一次实际调用行为的安全信物后,将信物通过安全信道传递到内核态的服务壳处. 服务壳在接收到实际行为所对应的安全信物后,通过以下三个过程完成信物校验:

(1)内核态的服务壳获取此时调用敏感服务的用户态应用程序进程空间中调用者的用户态内存地址 callee. 根据 callee 的地址回溯其上方是否有插桩对应的锁闭保护应用壳. 如果此处 callee 并未存在应用壳,则说明此次调用一定是未通过锁闭保护校验的非法调用行为,返回错误并终止敏感服务;

(2)如果 callee 处存在应用壳,位于内核态的服务壳需要根据此时敏感服务调用者 callee 的栈帧信息和来自信物传递时的熵变值(D,L)重新获取调用者的上下文元数据并计算生成此处预期调用行为所对应的安全信物. 服务壳中安全信物的生成过程与应用壳中信物生成过程是类似的,同样采用 $SHA1$ 算法计算相关上下文元数据的哈希值;

(3)比较位于内核态的服务壳所生成的预期行为的安全信物与接收到的应用壳传递的安全信物是否相同. 如果相同,则校验通过,此次对敏感服务的调用过程是符合预期行为的合法调用;否则,校验不通过,终止此次敏感服务的调用并告警.

5　实　现

锁闭保护结构的实现方案如图 6 所示,主要包括在编译时编译器对目标程序的静态分析、插桩和编译过程,运行时插入到目标程序应用壳的运行时库和位于内核态作为安全敏感服务 hook 的服务壳.

5.1　编译时静态插桩

如 4.1 节所述,阻断非预期执行攻击的基础是自动提取应用程序对安全敏感服务的调用行为,并在服务调用者处插桩应用壳代码. 本文基于 LLVM

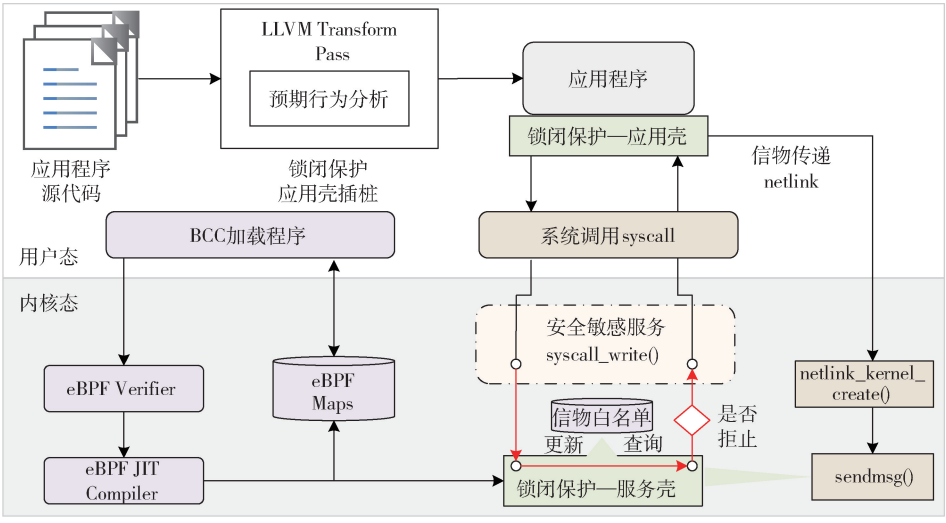


图 6 锁闭保护结构的实现方案

编译框架开发了针对安全敏感服务调用行为的编译时静态分析插件(即 LLVM Transform Pass),作为预期行为分析的具体实现. 首先,LLVM 框架将应用程序源代码转换成 LLVM IR 代码格式,本文开发的 LLVM 插件是针对 LLVM IR 语言的静态分析方案,因此具有语言无关性,可兼容 LLVM 架构支持的任意种类的编程语言. 其次,该插件根据算法 1 的步骤逐一遍历每条 LLVM IR 指令以寻找针对安全敏感服务的直接调用和间接调用行为. 最后,该插件通过在敏感服务调用处插桩对应用壳的调用指令改写目标程序源代码,并将应用壳的运行时库链接到目标程序,从而实现用户态的锁闭保护结构(应用壳).

5.2 应用壳运行时库

本文将锁闭保护应用壳实现为包含监视函数的静态库. 监视函数实现如 4.2 节所述的安全信物生成算法,并在信物生成后通过安全信道将其传递到位于内核态的锁闭保护服务壳处. 在运行时,应用壳监控函数的工作流程如下:

(1)应用程序执行至安全敏感服务调用处时会先触发应用壳监控函数,该函数根据代码段和栈帧长度生成一个合法的随机数作为随机偏移量;

(2)根据随机偏移量,监控函数在<代码段基址+随机偏移量>处获取代码段上下文信息以及当前栈帧地址前 16 字节的运行时信息作为元数据;

(3)监控函数根据上述采集的上下文信息的长度扩充元数据至 80 字节(符合 *SHA*1 哈希方案的源数据长度),使用 *SHA*1 算法将元数据映射为一个 64 字节的哈希值作为此次调用行为的安全信物;

(4)监控函数通过 *Netlink* 套接字将生成的安全信物和随机偏移量传递至内核态的服务壳处.

根据锁闭保护结构的设计方案,监控函数需要将信物通过安全信道传递给服务壳,从而证明此次敏感服务调用的合法性. 因为应用壳和服务壳分别位于用户态和内核态,因此信物的传递过程需要考虑用户态与内核态的通信方式. 如图 6,考虑到锁闭保护服务壳的实现为基于 eBPF 的内核 hook 方案,所以本文提出了基于 *Netlink* 通信的侧信道安全信物传递方案. 具体而言,在用户态,应用壳在生成安全信物后通过 *Netlink* 与 *netlink_kernel_create*() 创建的内核态的 *Netlink* 套接字进行通信,通信内容为生成的安全信物. 另一方面,在内核态,服务壳实现对 *sendmsg*()向内核态的 *Netlink* 套接字发送安全信物时,服务壳通过 *sendmsg*()的钩子函数获取到应用壳传递的安全信物,服务壳通过侧信道监听并接收了应用壳通过 *Netlink* 信道传递的安全信物.

5.3 安全敏感服务防护

锁闭保护服务壳的作用是校验安全敏感服务的调用行为是否为预期行为,并阻断所有非预期执行行为. 在 Linux 系统上,安全敏感服务以系统调用的方式为用户提供相关服务. 因此,为了阻断非预期执行攻击并防止攻击者绕过锁闭保护结构,本文将锁闭保护服务壳实现为安全敏感服务的内核层钩子函数,在安全敏感服务执行前判断调用行为的合法性并实施放行或阻断操作.

如图 6 所示,本文使用 BCC 工具集和 eBPF 技术构建锁闭保护服务壳的具体实现方案. eBPF 技

术允许用户在内核中执行自定义的代码片段且不影响内核代码,因此锁闭保护结构具有较好的安全性和可移植性,可兼容多版本的内核. 在用户态,BCC 加载程序启用位于内核态的锁闭保护服务壳,服务壳以 eBPF 的形式作为安全敏感服务 *sys_write*()和安全信道出口服务 *sendmsg*()的钩子函数. 按照 4.3 节的算法 2,服务壳实现并执行以下功能. 首先,服务壳监听 *sendmsg*()信道,接收来自安全信道的安全信物和随机偏移量. 然后,当用户态的应用程序调用 *sys_write*()进行写文件操作时,服务壳会钩住 *sys_write*()的执行,并根据随机偏移量和服务调用者、服务定义者和当前执行上下文查询信物白名单中预期行为对应的安全信物. 最后,服务壳将预期行为信物与来自应用壳的实际行为信物进行对比,如二者相同则放行此次对服务 *sys_write*()的执行;否则,终止对应的应用程序进程并告警.

6　评估实验

为验证锁闭保护结构对非预期执行攻击的防御效果,本文构造表 1 所示的实验环境应用于锁闭保护原型系统并构造相应的非预期执行攻击实例. 本文从防御有效性和性能开销 2 个方面对锁闭保护原型系统进行评估实验,旨在验证锁闭保护原型系统是否能防御攻击者实施非预期执行攻击行为、对识别应用程序中安全敏感服务调用行为的有效性,以及锁闭保护原型系统的效率和开销.

表 1　实验环境

软硬件	相关参数
操作系统	Ubuntu 18.04.6 x86 64
处理器	AMD Ryzen7950@5.40GHz
内核版本	Linux 5.4.0-120-generic

6.1　有效性验证和评估

如前文所述,非预期执行攻击分为两类场景:代码注入类、漏洞利用类. 攻击者通过代码注入实施的非预期执行攻击十分常见,例如攻击者通过鱼叉式钓鱼诱导受害者下载执行带有恶意程序的邮件附件,或者利用过滤器函数的漏洞通过数据输入接口将代码注入到程序解析器(例如,JavaScript 引擎等). 漏洞利用是攻击者利用程序中存在的漏洞实现其攻击目标的最为常见的攻击手法之一. 因此,本文设计了面向代码注入类和漏洞利用类的测试场景,以安全敏感服务 *ssize_t write*(*int fd*, *const void * buf*, *size_t count*)作为实验目标,验证这 2 类威胁场景下锁闭保护原型系统的有效性. 此外,为验证本文所提出的预期行为分析算法对安全敏感服务调用的有效性,本文选取了 Linux 常用的几类软件进行预期行为分析结果准确率和漏报率的评估. 值得注意的是,现有的防御技术主要针对攻击者实施漏洞利用的过程进行缓解而无法直接防御非预期执行攻击. 尤其是面对代码注入类攻击场景时,防御技术无法检测并阻断攻击者注入到系统中的恶意代码. 因此,在进行有效性验证实验时,本文并未选取常见的防御技术作为基准.

6.1.1　代码注入类验证实验

在真实 APT 攻击案例中,攻击者在初始访问后,通常会在受感染主机内下载并执行恶意程序以实现持久化、数据窃取/破坏等目的. 下载至受感染主机上的恶意软件作为攻击者注入到系统内的有效攻击载荷,调用系统内安全敏感服务以实施非预期执行攻击. 如表 2 所示,本文选取了 6 个真实 APT 攻击中使用的擦除器(wiper)样本作为实验目标,用于验证锁闭保护机制的有效性. 擦除器是 APT 攻击中一类常用的恶意软件,攻击者使用擦除器实施痕迹清理、数据破坏等攻击目的. 本文所选取的擦除器样本来源包括:(1)部分样本根据 APT 攻击报告中的恶意样本文件哈希值,以及通过恶意软件搜索网站,如微步云①、MalwareBazaar② 等下载获取;(2)部分样本未注明索引链接或样本未公开,本文根据 APT 攻击报告中公开的关键技术在本文的实验环境下复现了相关样本. 根据所选取的擦除器程序的功能和攻击目标,本文使用上述擦除器对实验环境的日志文件或特定文本文件(密钥)进行擦除,并重复擦除行为 10 次. 实验结果如表 2 最后一列所示,本文所实现的锁闭保护原型系统成功防御住了 6 个实验目标对不同文件的擦除攻击.

6.1.2　漏洞利用类验证实验

在实际网络攻击中,攻击者可利用系统中潜在的漏洞完成初始访问、权限提升等目的. 攻击者通过触发系统中的漏洞,使用已知或未知的漏洞利用手法调用系统内的安全敏感服务以实施非预期执行攻击. 为验证锁闭保护技术在检测阻断基于漏洞利用的非预期执行攻击的有效性,本文选取了 8 个影响

① 微步云沙箱. https://s.threatbook.com 2023,10,15

② MalwareBazaar. https://bazaar.abuse.ch/browse 2023,10,15

表 2 代码注入类验证实验结果

擦除器程序	描述	是否成功防御
WhisperGate	在某些国家冲突中广泛使用的擦除器,用于覆盖主引导记录 MBR	√
WIPERIGHT	用于清除基于 Linux 和 Unix 的系统上的特定日志条目	√
MIGLOGGLEARN	用于清除基于 Linux 和 Unix 的系统上的日志或从日志中删除某些字符串	√
Shamoon	于 2012 年首次发现的用于执行数据擦除任务的恶意擦除器软件	√
Uzapper	适用于基于 Linux 和 Unix 的系统的日志擦除工具	√
CoreEraser	通过覆盖文件内容并删除以实现无法恢复效果的文件损毁软件	√

范围较大的 Linux 内核提权漏洞作为实验目标,如表 3 所示.本实验使用的内核提权漏洞利用代码均来自 Exploit DB,它们通过越界写、双重释放等内存损坏型漏洞进行一系列的利用过程完成了对安全敏感服务的调用,最终实施了本地权限提升的攻击效果.

表 3 漏洞利用类验证实验结果

漏洞编号	影响组件	是否成功防御
CVE-2014-0196	tty	√
CVE-2019-13272	ptace_link	√
CVE-2021-3493	overlayfs	√
CVE-2021-4034	Polkit	√
CVE-2021-22555	Netfilter	√
CVE-2022-0847	pipe	√
CVE-2022-2588	opcua	√
CVE-2022-2639	openvswitch	√

根据所选定的内核提权漏洞和其所影响的组件,使用相应的漏洞利用脚本分别在未启用和启用锁闭保护机制的实验机上进行本地提权攻击复现.在未启用锁闭保护机制的实验环境下,上述 8 个漏洞利用脚本成功从普通用户提权至 root 用户;在开启锁闭保护机制后,锁闭保护机制检测并阻断了上述漏洞利用脚本对安全敏感服务的非预期执行攻击,成功防御内核漏洞导致的权限提升行为.

6.1.3 预期行为分析验证实验

在锁闭保护模型中,准确识别应用程序对安全敏感服务的预期调用行为是检测、阻断非预期执行攻击的前提,本文针对应用程序中对安全敏感服务的直接调用行为和间接调用行为设计了预期行为分析算法.为评估本文所提出的预期行为分析算法的有效性,本文选取 Linux 系统最为常用的工具集 GNU Binutils、流行的安全通信协议库 OpenSSL 以及开源的 XML 文件解析库 libxml2 作为实验目标.针对上述目标中对安全敏感服务 *ssize_t write(int fd, const void * buf, size_t count)* 和 *ssize_t send(int sockfd, const void * buf, size_t len, int flags)* 的直接调用和间接调用行为进行识别,并计算识别的准确率与误报率.

在本文所选取的 4 个目标程序中,使用本文所实现的预期行为分析模块在编译时对应用程序中对安全敏感服务 write 和 send 调用行为进行了统计和分析.如表 4 所示,通过对预期行为分析模块所识别的敏感服务直接调用和间接调用指令进行分析对比,该模块所识别的直接和间接调用行为均为实际存在的敏感服务调用行为,识别敏感服务调用行为的准确率为 100%,无误报发生.需注意,在 OpenSSL 中,存在一处对 write 间接调用的漏报.该处对 write 的间接调用是宏定义的 write 函数的包装器,该包装器仅包含 2 个参数,因此本文所提出的轻量级别名分析方案未识别到此处间接调用,如果使用重量级别名分析算法可以实现对这类间接调用的识别.本文仅是对基于锁闭保护原型系统的验证性实验,因此未采用高开销的别名分析算法.

表 4 预期行为分析验证实验及结果

应用程序	版本	直接调用(write)	间接调用(write)	直接调用(send)	直接调用(send)	准确率	漏报率
Binutils	2.40	51	2	0	0	100%	0
OpenSSL	3.1.2	11	3	4	0	100%	1/19
Libxml2	2.9.10	7	0	16	0	100%	0
c-ares	1.19.1	10	1	2	0	100%	0

6.2 性能评估

为了评估锁闭保护技术部署在真实系统中对系统总体性能的影响,本文使用 UnixBench 5.1.3 性能测试套件对启用锁闭保护原型系统的 Linux 系统

进行整体性能评估. UnixBench 是一个类 unix 系统性能测试工具,用于评估 Linux 系统下不同测试项的性能开销. 本文分别在实验环境下开启和关闭锁闭保护原型系统,共运行 UnixBench 性能测试 5 次,获取并计算各测试项分数的平均值. 以未开启锁闭保护时各测试项分数的平均值为基准,计算锁闭保护对系统不同功能(包括浮点运算、读写、进程创建、系统调用等)影响的性能开销占比.

UnixBench 的测试结果如图 7 所示,本次实验所使用的性能测试套件共进行 11 项测试,包括字符串处理(dhry2reg)、双精度浮点数运算(whetstone)、excel 函数吞吐量(excel)、文件读写(filecopy1/2/3,对应不同读写文件大小)、管道吞吐量(pipe)、上下文切换(context)、进程创建(spawn)、shell 脚本测试(shell)、系统调用(syscall). 测试实验分别对单核性能开销和多核性能开销进行记录,实验结果表明开启锁闭保护机制所带来的单核平均性能开销和多核平均性能开销分别是 2.59% 和 4.45%. 其中,开启锁闭保护机制后,对单核性能影响最大的测试项为文件读写,带来的性能开销为 9.64%;对多核性能影响最大的测试项为进程创建,带来的性能开销为 12.17%.

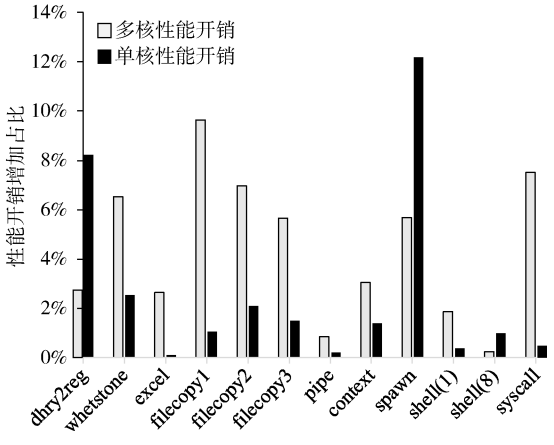


图 7　UnixBench 性能测试结果

7　相关工作

由 C、C++等类型不安全语言编写的应用程序容易出现内存安全漏洞. 在过去的几十年间,安全研究人员提出了大量的漏洞利用缓解技术,在缓解非预期执行攻击上发挥了重要的作用. 本文以非预期执行攻击的不同攻击场景为分类,描述现有相关工作对代码注入和漏洞利用两类场景的防御思路.

7.1　代码注入类防御

攻击者通过向信息系统内注入恶意代码,调用并执行安全敏感服务是实施非预期执行攻击的主要手段之一. 代码注入类攻击场景可分为利用程序的输入功能向程序进程注入恶意代码以及借助其他攻击载体向系统内注入恶意代码[8].

数据执行保护是部署最广泛的保护机制,它们通过将内存的某些区域标记为仅用于数据,这些区域是可写入但不可执行的,这样攻击者通过数据输入功能注入的恶意代码不具备执行权限,从而避免了恶意代码调用安全敏感服务的攻击. 现代处理器从硬件上支持了这两种保护机制,使得其带来的开销可忽略不计,很好地防御了代码注入类攻击.

在内存中引入秘密性是防止代码注入攻击的另一种防护思路[9,10]. 任何代码注入攻击成功的前提是攻击者了解并熟悉被攻击系统的语言,指令随机化技术通过对执行的指令进行随机化处理而增加成功注入 shellcode 的难度. 与之类似的防御技术还有指针加密技术. 2003 年,Cowan 等人[11]首次提出了指针加密方案 PointGuard,尝试对内存中的指针加密来防止栈溢出漏洞. 该方案对内存中所有指针进行加密,当需要加载加密指针至寄存器时进行解密,这限制了攻击者操纵指针变量的能力. 指针加密技术最为成熟的方案是 ARM 处理器于 2016 年新增的指针验证[12](Pointer Authentication,PA)功能. 与上述的两项工作类似,PA 在将指针变量存储至内存时创建指针验证码(Pointer Authentication Code,PAC),创建的 PAC 保存在对应指针变量的高位,当指针变量被加载时通过 PAC 验证该指针的完整性. PA 使用专有的高权限寄存器存储密钥;并增加了用于创建和验证 PAC 的指令,在硬件层面完成了指针验证功能. 虽然 PA 的安全性和兼容性得到了提升,但研究表明,PA 无法完全解决恶意 PAC 生成和重用攻击的问题,存在被绕过的风险[13].

综上,现有的代码注入攻击的防御技术聚焦于阻断利用数据输入注入恶意代码到程序进程内执行的攻击过程,一些成熟的防护方案得到了硬件支持并广泛部署到各类系统中,取得了很好地效果. 然而,在即时(Just-In-Time,JIT)编译等情况下,DEP 等安全策略无法完全执行[14];以及代码重用攻击[15]的出现使得这类防护技术存在被绕过的可能. 利用其他攻击载体向系统内注入恶意代码的攻击行为,攻击者往往会通过伪装的攻击载体而取得合法用户的信任,例如通过鱼叉式钓鱼邮件投递带有恶意宏

载荷的工作文档附件,如果用户在打开文档时启用宏,则恶意宏会以合法用户身份被解析执行. 这类代码注入是以合法用户身份执行的,现有的防护技术无法防御这类代码注入攻击. 锁闭保护作为现有防护技术的一种补充,通过分析程序实际行为是否与预期行为存在差异性来识别恶意代码,从而阻断以任何方式注入到系统内的恶意代码,实现对代码注入类攻击场景的防御.

7.2 漏洞利用类防御

攻击者利用程序中存在的漏洞(大多为内存损坏型漏洞),通过篡改关键内存数据(例如,函数返回地址、函数指针、权限相关元数据等),实施控制流劫持[16,17]、内存数据泄露[18]、权限提升[19]等攻击,最终实现对安全敏感服务的非预期执行攻击. 针对这类攻击场景,相关工作通过在内存空间中引入随机性、利用程序执行流的确定性等思路实现对非预期执行攻击的缓解,即概率性防御技术和确定性防御技术.

ASLR 是部署最广泛的地址空间随机化技术,它被证明是最全面、最有效的概率性防御方案之一. ASLR 的核心思想是随机化内存布局,包括堆、栈、代码段和库等. ASLR 在加载时将代码和数据随机地放置到不同的内存区域,使得攻击者难以推断出代码和数据的位置,从而无法稳定地操纵内存数据或劫持控制流. 虽然正确地实施 ASLR 能有效地防御各类内存攻击,但是在一些操作系统上 ASLR 的实现没有保证所有数据和代码区域的随机性(在一些 Linux 发行版上,ASLR 仅使库的代码位置是随机的),这使得 ASLR 存在被绕过的风险.

与概率性防御技术引入随机性进行防御的思路不同,确定性防御技术旨在通过在程序执行期间验证程序的控制流或数据流信息是否存在异常来防御漏洞利用行为. 控制流完整性检测(CFI)是一类部署非常广泛的确定性防御技术,其目标是确保每个间接控制流转移(indirect control transfer,ICT)指令跳转至合法目标来阻止控制流劫持攻击.

粗粒度的 CFI 方案,如 CCFIR[20]和 BinCFI[21],通过二进制重写技术检查控制流完整性,具有很强的兼容性和良好的性能. 一些细粒度 CFI 方案尝试引入硬件加速机制来优化 CFI 过程的性能开销,例如 PT-CFI[22]和 PathArmor[23]等通过使用 Intel PT 硬件机制来实施控制流完整性校验,但依然带来了不小的开销并引入了兼容性问题,无法广泛部署. μCFI[24]使用执行上下文来确保唯一跳转属性变量,这与锁闭保护校验信物的思路是类似的,既确保了控制流的完整性又降低了开销. 但 μCFI 方案要求对应用程序进行细粒度的复杂数据流分析,为部署该方案带来了挑战. 虽然细粒度 CFI 方案有更强的安全性,但维护校验每个 ICT 指令的合法目标相关的元数据会带来过高的性能开销,实际部署的 CFI 方案通常是妥协的、不精准的粗粒度 CFI 方案[25,26]. 但粗粒度的跳转目标集合使得每个 ICT 指令所允许的跳转目标太多,使得非法跳转成为可能,无法提供强大的安全保障来消除所有的控制流劫持攻击. Enes 等人[5]提出了绕过粗粒度 CFI 方案的攻击方法,验证了粗粒度 CFI 方案是存在缺陷的. 锁闭保护机制仅要求粗粒度的服务调用者识别,相比于细粒度 CFI 方案具有更好的实用性和兼容性. 同时,相比于粗粒度 CFI 方案,锁闭保护机制会检查所有对安全敏感服务的调用行为,使得攻击者难以绕过.

缓冲区边界检查机制也是一类常见的确定性防御技术. 针对 C/C++等低级语言的内存损坏问题,许多安全研究试图通过加强内存管理的安全性来确保应用程序无法读取或写入越界指针或悬空指针,从而阻断内存损坏漏洞的利用过程. Cyclone[27]和 CCured[28]扩展了 C 语言的实现特性,使用类型推断、运行时检查和类型静态验证保证指针是类型安全的,扩展在运行时检查指针的安全性,从而确保了防止常见的内存损坏问题. 但是,这类方案面临着一个问题,即存在非常庞大的现有代码,无法对所有使用 C/C++语言的代码都进行安全加固.

SoftBound[29]和 CETS[30]以增强 C 语言的空间安全性的编译时插桩的方式实现了完全的内存安全. SoftBound 将指针的基地址和边界信息记录为元数据,从而提供了空间安全性. CETS 为每个对象维护一个标识符,将元数据与不相交的指针相关联,检查内存对象在解引用时是否为有效的对象,从而提供时间安全性. 但它们引入了多高的性能开销和运行时开销,无法大量部署在已有系统中.

综上,现有的漏洞利用缓解技术针对漏洞利用过程的各个关键步骤的目的实施不同的防护方案. 概率性防御方案通过在内存地址空间中引入随机性和秘密来增加攻击者获取/篡改关键内存数据的难度. 确定性防御方案利用程序执行流的确定性防止控制流劫持、确保内存安全. 虽然在理论上现有的防御技术能有效地防御漏洞利用过程,但在具体实施相关防护方案时,厂商必须考虑到防护技术的性能开销和兼容性. 广泛部署在商用系统上的防护方

案往往是粗粒度、低开销的实现,但粗粒度方案引入的脆弱性使得这些防护方案存在被绕过的风险. 锁闭保护既在信物中引入了随机性和秘密,又需要敏感服务调用满足确定性的预期行为;而在运行时仅需对安全敏感服务的调用行为进行动态校验,避免了细粒度检测带来的性能开销.

8 结　论

本文定义了面向安全敏感服务的非预期执行攻击,为了防御调用安全敏感服务的非预期执行攻击,提出了锁闭保护模型. 以该模型为理论支撑,本文提出了一种称为锁闭保护结构的安全防护机制. 锁闭保护结构通过预期行为分析、实际行为监控和非预期行为阻断这三个关键步骤检测并阻断任何偏离程序预期行为的安全敏感服务调用行为.

本文在 Linux 实验环境下实现了锁闭保护结构的原型系统,并使用真实 APT 攻击样本、内核权限提升漏洞、流行的应用程序进行了有效性验证. 实验结果表明,锁闭保护原型系统可有效地防御代码注入类场景和漏洞利用类场景的非预期执行攻击,且各项性能开销较低.

参 考 文 献

[1] Savita Mohurle, Manisha Patil. A brief study of wannacry threat: Ransomware attack 2017. International Journal of Advanced Research in Computer Science, 2017, 8(5): 1938-1940

[2] Takeshi Okamoto. SecondDEP: Resilient computing that prevents shellcode execution in cyber-attacks. Procedia Computer Science, 2015, 60: 691-699

[3] Gaurav S Kc, Angelos D Keromytis, Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization// Proceedings of the 10th ACM Conference on Computer and Communications Security. Washington, USA, 2003: 272-280

[4] Nathan Burow, Scott A Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, Mathias Payer. Control-flow integrity: Precision, security, and performance. ACM Computing Surveys (CSUR), 2017, 50(1): 1-33

[5] Enes Göktas, Elias Athanasopoulos, Herbert Bos, Georgios Portokalidis. Out of control: Overcoming control-flow integrity// Proceedings of the 2014 IEEE Symposium on Security and Privacy. 2014: 575-589

[6] Hira Shahzadi Sikandar, Usman Sikander, Adeel Anjum, Muazzam A Khan. An adversarial approach: comparing windows and linux security hardness using mitre ATT&CK framework for offensive security// Proceedings of the 2022 IEEE 19th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET), 2022: 022-027

[7] William G. Halfond, Jeremy Viegas, Alessandro Orso, Others. A classification of SQL-injection attacks and countermeasures// Proceedings of the IEEE International Symposium on Secure Software Engineering, 2006: 13-15

[8] Aleph One. Smashing the stack for fun and profit. Phrack magazine, 1996, 7(49): 14-16

[9] Sandeep Bhatkar, R. Sekar. Data space randomization// Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2008: 1-22

[10] Ali Jose Mashtizadeh, Andrea Bittau, Dan Boneh, David Mazires. CCFI: Cryptographically enforced control flow integrity// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York, USA, 2015: 941-951

[11] Crispin Cowan, Steve Beattie, John Johansen, Perry Wagle. PointGuard™: Protecting pointers from buffer overflow vulnerabilities// Proceedings of the 12th USENIX Security Symposium (USENIX Security 03), 2003: 7-7

[12] Hans Liljestrand, Thomas Nyman, Kui Wang, Carlos Chinea Perez, Jan-Erik Ekberg, N. Asokan. PAC it up: Towards pointer integrity using ARM pointer authentication// Proceedings of the 28th USENIX Security Symposium (USENIX Security 19). Santa Clara, USA, 2019: 177-194

[13] Joseph Ravichandran, Weon Taek Na, Jay Lang, Mengjia Yan. PACMAN: attacking ARM pointer authentication with speculative execution// Proceedings of the 49th Annual International Symposium on Computer Architecture. New York, USA, 2022: 685-698

[14] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Ahmad-Reza Sadeghi. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization// Proceedings of the 2013 IEEE Symposium on Security and Privacy. Berkeley, USA, 2013: 574-588

[15] Didier Stevens. Malicious PDF documents explained. IEEE Security & Privacy, 2011, 9(1): 80-82

[16] Minh Tran, Mark Etheridge, Tyler Bletsch, Xuxian Jiang, Vincent Freeh, Peng Ning. On the expressiveness of return-into-libc attacks// Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011. Menlo Park, USA, 2011: 121-141

[17] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)// Proceedings of the 14th ACM Conference on Computer and Communications Security. New York, USA, 2007: 552-561

[18] Shuo Chen, Jun Xu, Emre Can Sezer, Prachi Gauriar, Ravishankar K Iyer. Non-control-data attacks are realistic threats// Proceedings of the 14th USENIX Security Symposium (USENIX Security 05). Baltimore, USA, 2005: 146-146

[19] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Marcel Winandy. Privilege escalation attacks on android// Information Security: 13th International Conference, ISC 2010. Boca Raton, USA, 2011: 346-360

[20] Zhang Chao, Wei Tao, Chen Zhaofeng, Duan Lei, Szekeres Laszlo, Mccamant Stephen. Practical control flow integrity and randomization for binary executables// Proceedings of the 2013 IEEE Symposium on Security and Privacy. Berkeley, USA, 2013: 559-573

[21] Mingwei Zhang, R Sekar. Control flow and code integrity for COTS binaries: An effective defense against real-world ROP attacks// Proceedings of the 31st Annual Computer Security Applications Conference. New York, USA, 2015: 91-100

[22] Yufei Gu, Qingchuan Zhao, Yinqian Zhang, Zhiqiang Lin. PT-CFI: Transparent backward-edge control flow violation detection using intel processor trace// Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. New York, USA, 2017: 173-184

[23] Van Der Veen Victor, Andriesse Dennis, Gkta Enes, Gras Ben, Sambuc Lionel, Slowinska Asia, Bos, Herbert. Practical context-sensitive CFI// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015: 927-940

[24] Hong Hu, Chenxiong Qian, Carter Yagemann, Simon Pak Ho Chung, William R Harris, Taesoo Kim, Wenke Lee. Enforcing unique code target property for control-flow integrity // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018: 1470-1486

[25] Yubin Xia, Yutao Liu, Haibo Chen, Binyu Zang. CFIMon: Detecting violation of control flow integrity using performance counters// Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). Boston, USA, 2012: 1-12

[26] Vishwath Mohan, Per Larsen, Stefan Brunthaler, Kevin W Hamlen, Michael Franz. Opaque Control-Flow Integrity // Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2015: 27-30

[27] Trevor Jim, J Gregory Morrisett, Dan Grossman, Michael W. Hicks, James Cheney, Yanling Wang. Cyclone: a safe dialect of C// USENIX Annual Technical Conference, General Track. Monterey, USA, 2002: 275-288

[28] George C Necula, Scott Mcpeak, Westley Weimer. CCured: Type-safe retrofitting of legacy code// Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA, 2002: 128-139

[29] Santosh Nagarakatte, Jianzhou Zhao, Milo M K Martin, Steve Zdancewic. SoftBound: Highly compatible and complete spatial memory safety for C// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA, 2009: 245-258

[30] Santosh Nagarakatte, Jianzhou Zhao, Milo M K Martin, Steve Arthur Zdancewic. CETS: compiler enforced temporal safety for C// Proceedings of the 2010 International Symposium on Memory Management. New York, USA, 2010: 31-40

YANG Jia-Geng, Ph. D. candidate. His current research interest is network security.

FANG Bin-Xing, Ph. D., professor, Ph. D. supervisor, Academician of Chinese Academy of Engineering. His current research interests include computer architecture, computer network and network security.

JI Tian-Tian, Ph. D., postdoctoral scholar. Her current research interest is network security.

ZHANG Yun-Tao, Ph. D., postdoctoral scholar. His current research interest is network security.

WANG Tian, Ph. D. candidate. His current research interest is network security.

CUI Xiang, Ph. D., professor, Ph. D. supervisor. His current research interest is network security.

WANG Yuan-Di, bachelor. Her current research interest is network security.

Background

Invoking security-sensitive services is a necessary step for malware to achieve its attack goals. On the one hand, for exploitation-based invocation of security-sensitive services, existing defenses use probabilistic protection and deterministic protection to mitigate the process of exploitation. However, in order to reduce the overhead introduced by the defenses, the defense solutions applied in commercial systems usually use only coarse-grained security checks, which can be bypassed by exploitation. On the other hand, there is no effective defense to prevent attacks when malware delivered with legitimate payloads calls security-sensitive services. Unfortunately, using malware delivered to target machines to call security-sensitive services is one of the common APT techniques. In response to the above problems, this paper proposes a new defense mechanism called lockdown-protection which analyzes the run-time actual behaviors of an application whether deviate from the compile-time expected behaviors. Lockdown protection can detect and prevent illegal security-sensitive services calls in real time by analyzing exceptions in application behavior.