



**操作系统内核**

**(期末论文)**

**任课教师 杨文川**

**姓 名 林于翔**

**学 院 网络空间安全学院**

**专 业 网络安全**

**学 号 2020211919**

**2022年 12 月**

# 操作系统内核模糊测试技术

**摘要**：目前，针对操作系统内核漏洞挖掘的一个重要方法就是模糊测试。内核模糊测试的应用促进了操作系统内核和驱动程序安全防护水平的显著提升。本文对现有的内核模糊测试方法进行研究，介绍了内核模糊测试的发展情况和技术理论，并对内核模糊测试进行分类，总结了近年来内核模糊测试中使用的新技术。最后讨论了目前模糊测试研究中的问题，并对内核模糊测试未来发展趋势进行了展望。

关键词：操作系统内核；模糊测试；漏洞挖掘；驱动程序

## 引言

操作系统作为当今社会电子设备中最为基础的软件设施，从其诞生之初就产生了安全问题，其安全性一直是计算机安全研究的热点，是软件安全最早的研究对象之一[1]。随着近年来技术的发展，针对Windows 内核和Linux 内核等主流PC 和服务器操作系统的漏洞挖掘研究也越来越多，产生了大量研究成果。自2007年以来， iOS 和Android 等移动终端系统不断发展壮大，其安全研究也成为一个重要关注的对象。

内核漏洞有其独特的性质。首先，一般的操作系统平台都有着数量巨大的用户群，这使得操作系统内核漏洞出现重复的频率很高。其次，操作系统内核代码极其庞大，有着大量的遗留代码和纷繁复杂的子模块，大大增加了漏洞发生的可能。再者，由于内核特权级高，内核漏洞可以获取内核资源的访问权限、获取超级用户权限、关闭安全防护功能等高危攻击行为，并且能够为rootkit 等后门驻留打开突破口。最后，内核重视运行性能，默认情况下安全防护机制较少。尤其是在实际工作中还需要维持生产环境的稳定，甚至需要关闭很多内核防御机制，给攻击方留下了很大的可利用空间。

模糊测试作为一种效果很好的漏洞挖掘方法，在内核漏洞挖掘领域得到了广泛的应用。 以虚拟化、硬件调试器、云计算为代表的新技术都被应用到了内核模糊测试漏洞挖掘框架中，主流操作系统都有专门针对其内核的模糊测试漏洞挖掘工具。从发展历史来看，内核漏洞模糊测试从一开始简单的随机生成测试例，发展到输入规范化、虚拟化运行系统、覆盖率反馈、多系统支持、云平台集群化运行等诸多新特性[2]。

本文主要对近年来内核模糊测试的发展进行介绍，总结内核模糊测试的技术特点和基本框架。对典型工具的技术思想进行综述，并专门讨论了驱动模糊测试技术。最后对本文进行总结，讨论了内核模糊测试今后的发展方向。

## 内核模糊测试技术概述

### 模糊测试

模糊测试（Fuzzing），是一种通过向目标系统提供非预期的输入并监视异常结果来发现软件漏洞的方法。它作为目前较为有效的软件漏洞挖掘工具，在新世纪以来得到了广泛应用。模糊测试的核心思想是自动化或半自动的生成输入数据到目标程序中，监测目标程序运行是否发生异常。通过对造成程序运行失常的输入执行过程进行分析，从而找出目标程序中的隐藏缺陷。在模糊测试中，用随机坏数据（也称做 fuzz）攻击一个程序，然后等着观察哪里遭到了破坏。模糊测试的技巧在于，它是不符合逻辑的：自动模糊测试不去猜测哪个数据会导致破坏（就像人工测试员那样），而是将尽可能多的杂乱数据投入程序中。从测试例生成来说，模糊测试可以划分为基于生成( Generation-based) 和基于变异( Mutation-based) 两种技术路线，也有将其称为基于语法( Grammar-based) 和基于反馈( Feedback-based) 的模糊测试。从引导方式上来说，模糊测试可以划分为基于随机的模糊测试( blindfuzzing) 和基于覆盖率引导的模糊测试。在实践中，模糊测试确实能够挖掘出内存错误、数据竞争、逻辑错误等多种类型的漏洞。

模糊测试主要的缺点是覆盖率较低，生成的测试例一般只能覆盖小部分目标程序代码，并且大量测试例基本上是无效的。实践中，模糊测试在漏洞挖掘领域能取得良好效果，已经成为业界主流的漏洞挖掘方法。

### 内核模糊测试基本框架

内核模糊测试的威胁模型将用户态或硬件输入到操作系统内核的数据看作是威胁来源，即从低权限的用户空间、外部硬件到高权限的内核空间中的输入是不可信的。目前，用户态输入作为威胁来源的模糊测试已经有了比较多的研究，但将硬件输入作为威胁来源的研究还较少[3]。

内核模糊测试将生成的错误数据输入到被测内核，并期望内核异常运行。然而，由于操作系统内核的特殊性，内核模糊测试在测试用例生成、输入、内核操作监控等方面需要更高级别的技术实现。除了直接使用物理机器和内核代码库的用户模式模糊测试外，其他内核模糊测试工具也应用虚拟化来运行相同级别的目标系统。一般来说，内核模糊测试的主要评估标准包括内核代码的覆盖范围、识别漏洞的能力以及内核监控的额外开销。

## 内核模糊测试技术分类和典型工具

除了传统软件测试中基于对目标软件知识依赖程度而划分的白盒、灰盒及黑盒测试，内核模糊测试还有很多的分类标准。系统内核的攻击面基本上可以认为是传入内核的各种数据接口，包括系统调用参数、硬件输入、用户态共享到内核态的内存对象等。 从这些攻击面来看，内核模糊测试可以分为面向系统调用和面向系统数据处理接口两种。 从引导方式来看，内核模糊测试工具也可以分为随机模糊测试( blind fuzzing)和覆盖率引导的模糊测试。 除此以外，也可以用针对的目标操作系统、漏洞类型将内核模糊测试工具进行分类。内核模糊测试工具可以根据工作原理分为基于随机生成、基于类型知识、基于钩子以及基于反馈驱动四种类型[4]。另外，还有一类是基于内核漏洞状态感知的模糊测试工具，这种模糊测试的测试例输入可以是运行正常的程序，通过对内核状态的监控，进而捕获信息泄露、内存错误等类型的漏洞。值得注意的是，随着移动互联网和物联网的发展，模糊测试也成为挖掘这些平台漏洞的通用方法。

## 内核模糊测试中的关键技术

在操作系统内核模糊测试系统的搭建中，针对目标操作系统内核，需要整体设计测试目标接口、测试用例生成和变异方法、测试用例输入执行方式、反馈机制等方面。内核模糊测试系统的搭建与普通应用程序模糊测试比较类似，有研究将用户态模糊器作为输入生成器，也有部分直接使用基于系统调用接口知识的方式生成测试例。 为了实现高效的模糊测试，还需要重点考虑被测内核运行方式、内核漏洞触发状态感知、以及实现代码执行信息反馈引导等几个方面的问题[5]。

### 虚拟化技术

一般来说，操作系统内核只能运行在计算机体系特权级较高的层级上。2015年以前的内核模糊测试研究[6]主要为利用HOOK技术对Windows、MacOS 等操作系统部分函数进行修改，实现输入测试例和监控系统崩溃功能。但为了实现对内核状态的方便监控，就需要采用虚拟化技术将让被测内核降级到用户态运行。在内核模糊测试中一般使用软件虚拟化和硬件虚拟化平台运行被测内核，常见的平台有QEMU和KVM等。 Panda研究了利用虚拟化技术记录漏洞生命周期，完整收集漏洞状态语义，极大的方便了内核漏洞的调试。另外，基于感知的内核模糊测试工具基本上就是围绕虚拟化技术展开研究的。从开始的系统加载到漏洞状态检测，此类工具都需要透过虚拟机或模拟器对内核的运行状态进行监控。

### 内核漏洞状态感知技术

内核异常转态感知技术也是内核漏洞研究的一个重要方面，并且极大程度上决定着内核漏洞挖掘的效率。内核漏洞感知技术既需要精确地捕捉到内核漏洞触发时的寄存器状态、堆栈状态等底层硬件信息，还要对漏洞发生时相关内存对象分配释放信息、内核栈信息、线程信息等上层漏洞语义进行记录。Linux系统上实现了一系列的内核安全违例检查工具，这些工具都受到了PaX项目以及用户态程序漏洞检测技术的启发，利用雷区(Ｒed Zone)等技术对Linux 内核中内存读写、线程创建与销毁等可能导致漏洞的行为进行探测。其中以KASAN18 ( Kernel Address Sanitizer) 和UBSAN19 ( Undefined-Behavior Sanitizer) 最为成熟，已经可以直接应用到Linux内核中，可以通过对内存对象读写监控、内存对象周围监控、内存延迟回收等内核活动实现内存错误探测，发现内核中存在的堆栈移除、UAF( Use-after-free) 和越界读写漏洞，并打印输出漏洞信息。KMSAN20 ( Kernel Memory Sanitizer) 和KTSAN21( Kernel Thread Sanitizer) 还都处于原型开发阶段，主要用于未初始化内存使用和竞态条件错误的探测。 另外，针对信息泄漏漏洞，可以配合内核数据污染，采用污点分析的方法进行漏洞挖掘。

### 代码覆盖率计算技术

代码覆盖率，是一种通过计算测试过程中被执行的源代码占全部源代码的比例，进而间接度量软件质量的方法。它在保证测试质量的时候潜在保证实际产品的质量，可以基于此在程序中寻找没有被测试用例测试过的地方，进一步创建新的测试用例来增加覆盖率。按性质，它属于白盒测试的范畴，即主要依据源代码的内部结构来设计测试用例，通过设计不同的输入来测试软件的不同部分。

根据评价的标准和方法不同，代码覆盖率测试又可以细分为语句覆盖（statement coverage）、判定覆盖（decision coverage）、条件覆盖（condition coverage）、条件判定组合覆盖（condition decision coverage）、路径覆盖（path coverage）、多条件覆盖（multi-condition coverage）和修正条件判定覆盖（modified condition / decision coverage）等。

针对不同测试层次，代码覆盖率主要有单元级或架构级。前者较为基础，而且使用更为方便，所以应用非常广泛。单元级测试常用的指标主要有：语句覆盖、判定覆盖和修正条件判定覆盖。

语句覆盖：又称行覆盖(LineCoverage)，指已经被执行到的语句占总可执行语句（不包含类似C++的头文件声明、代码注释、空行等等）的百分比，这是最常用的也是要求最低的覆盖率指标，实际中通常会结合判定覆盖率或者条件覆盖率一起使用。判定覆盖：又称分支覆盖(BranchCoverage)，用以度量程序中每一个判定的分支是否都被测试到了，即代码中每个判定的"真"和"假"至少执行一次。条件覆盖：它度量判定中的每个子表达式结果true和false是否都被测试到了。

模糊测试中代码覆盖率指的是输入的测试例能够让目标程序的控制流途经代码的数量，一般计数的单位都是控制流上基本块。目前内核模糊测试代码覆盖率计算主要方式包括源码插桩、Intel PT控制流记录、单步执行、性能监控单元(PMU)取样、动态二进制翻译等方式[7]。KCOV 是Syzkaller项目组在GCC 编译器中实现的一种针对Linux内核的源码插桩覆盖率检测工具，专门配合Syzkaller等模糊测试工具进行覆盖率测量。通过开启特定的Linux内核编译选项，记录执行系统调用产生的内核代码覆盖率。KCOV 记录的数据包括内核代码基本块的地址和这些地址的数量。 通过将每一个地址与上一个地址进行哈希等运算生成控制流图中的边，然后以此计算整个测试例对内核代码的覆盖率。Intel ProcessorTrace( Intel PT) 是Intel 第五代( Broadwell) CPU上添加的一个新特性，实现了使用硬件对程序控制流进行记录，提高了代码跟踪的效率。通过使用Intel PT 对内核控制流进行记录，通过解码获取控制流覆盖率的基本块，就可以高效地记录测试例对应的内核代码覆盖率，不需要对源码进行处理，并且相对于动态二进制翻译极大的提高了效率。

## 总结与展望

内核模糊测试作为高效的内核漏洞挖掘工具，在近年来得到了较好的发展[8]，各种新技术新做法都被应用到内核模糊测试中，并且取得了良好的效果。目前，实践中内核模糊测试的主要局限性还是在于操作系统内核复杂性导致的一系列问题上。首先内核模糊测试还需要能力更强的漏洞检测方法。由于内核运行状态较为复杂，大量的子系统事务相互影响，目前比较成熟的KASAN 在检测内存错误也会存在无法准确还原漏洞语义的情况，而针对未初始化内存、数据竞争、未定义行为等漏洞的检测更是存在较大困难，这也使得漏洞调试、复现等环节收到了巨大的限制。 其次，目前的内核模糊测试工具除Syzkaller系列以外都没有正面面对操作系统内核规模庞大、接口众多的问题。Syzkaller虽然能够一定程度自动化提取系统调用、系统库函数等内核调用接口，但仍然需要人工编写测试例生成的模版。最后，模糊测试固有的测试例质量不高的问题。目前IMF、Moonshine 等工具虽然已经对这个问题进行了研究，但在解读内核中纷繁复杂的状态关系上仍然还有较大的改进空间。

由于技术实现难度等原因，内核模糊测试的发展相对于普通应用程序模糊测试还是有一定的滞后性，将普通应用程序模糊测试已有的方法应用在内核模糊测试上是一种比较经济的改进方法。 普通应用程序的模糊测试在AFL 等工具的基础上使用了多种方法改进了代码覆盖率的表示形式。另外，随着神经网络技术的发展，也有研究在模糊测试中应用深度学习技术提高模糊测试能力[9]。这些在普通应用程序模糊测试中行之有效的方法都可以尝试应用到内核模糊测试中。从长期来看，内核模糊测试未来除了可以在传统的虚拟化的基础上进行更加细致的研究，还可以采用类似于JANUS 的方法也是一种非常好的选择，将内核子系统或驱动抽离单独对其进行模糊测试，提高漏洞复现能力和测试效率。

## 参考文献

1. Anderson J P．Computer security technology planning study［Ｒ］．Anderson( James P) and Co Fort Washington PA，1972．
2. 李贺,张超,杨鑫,朱俊虎.操作系统内核模糊测试技术综述[J].小型微型计算机系统,2019,40(09):1994-1999.
3. Song D，Hetzelt F，Das D，et al． PeriScope: an effective probing and fuzzing framework for the hardware-OS boundary［C］/ / Proceedingsof the Network and Distributed Systems Security Symposium(NDSS) ， Internet Society， 2019: 1-15． fuzzer［C］
4. Han H S，Sang K C． IMF: inferred model-based / / ACM Sigsac Conference，ACM，2017: 2345-2358．
5. 杨鑫. Linux内核系统调用接口模糊测试技术研究[D].战略支援部队信息工程大学,2020.DOI:10.27188/d.cnki.gzjxu.2020.000081.
6. Ni Tao．Ｒesearch on key technologies for detection and exploitationof windows kernel vulnerabilities ［D］．Zhengzhou: Information Engineering University，2013．
7. Xu Yong-jian． Ｒesearch on detecting vulnerabilities in linux driver［D］．Beijing: Beijing University of Technology，2015．
8. Qin Bi-shi． Ｒesearch on detection and analysis of vulnerability of windows device drivers［D］Beijing: University of Chinese Academy of Sciences，2015．
9. Hu Z，Shi J，Huang Y H，et al．GANFuzz: a GAN-based industrial network protocol fuzzing framework［C］/ /Proceedings of the 15th ACM International Conference on Computing Frontiers，ACM，2018: 138-145．