空天信息安全与可信计算
教育部重点实验室

Linux分析与安全设计

第8章 Linux安全研究进展

信息安全四大顶级会议

- S&P-IEEE Symposium On Security and Privacy
- USENIX Security Symposium
- CCS-ACM Conference on Computer and Communications Security
- NDSS-Network and Distributed System Security Symposium

其它著名安全会议和杂志

- TDSC（ IEEE Transactions on Dependable and Secure Computing ）-CCF A
- TIFS（IEEE Transactions on Information Forensics and Security ）-CCF A
- Blackhat
- Defcon
- RSA

目录

- **U**nderstanding Linux Malware （S&P 2018）
- **K-Miner**: Uncovering Memory Corruption in Linux （NDSS 2018）
- **HFL**: Hybrid Fuzzing on the Linux Kernel （NDSS 2020）
- **PeX**: A Permission Check Analysis Framework for Linux Kernel （USENIX Security 2019）
- **KEPLER**: Facilitating Control-flow Hijacking Primitive Evaluation for Linux Kernel Vulnerabilities （USENIX Security 2019 ）
- **LBM**: A Security Framework for Peripherals within the Linux Kernel （S&P 2019）
- **WireGuard**: Next Generation Kernel Network Tunnel （NDSS 2017）

Understanding Linux Malware

- 嵌入式设备的激增和物联网正在迅速改变恶意软件的格局。嵌入式系统依赖于各种不同的体系结构。这导致大量此类系统运行Linux操作系统的某些变体，从而促使恶意行为者制造出 Linux恶意软件。目前还没有全面的研究，分析和理解Linux恶意软件
- 该论文介绍了专门针对Linux恶意软件量身定制的第一个恶意软件分析系统的设计和实现细节。然后，介绍了对10,548个恶意软件样本（在一年的时间范围内收集）进行的首次大规模测量研究的结果，记录了详细的统计数据和见解。

S&P 2018

Understanding Linux Malware

面临的挑战：

- **目标多样性**

基于Linux的恶意软件可以针对多种目标，例如Internet路由器，打印机，监控摄像头，智能电视或医疗设备

- **静态链接**

大多数Linux恶意软件使用静态链接，很难定位二进制文件中具体使用了哪一个库函数

- **分析环境**

大多数Linux恶意软件需要Root权限执行，因此缺乏可用的沙箱分析环境

Understanding Linux Malware

- 设计和实现了第一个专门针对Linux恶意软件量身定制的分析框架

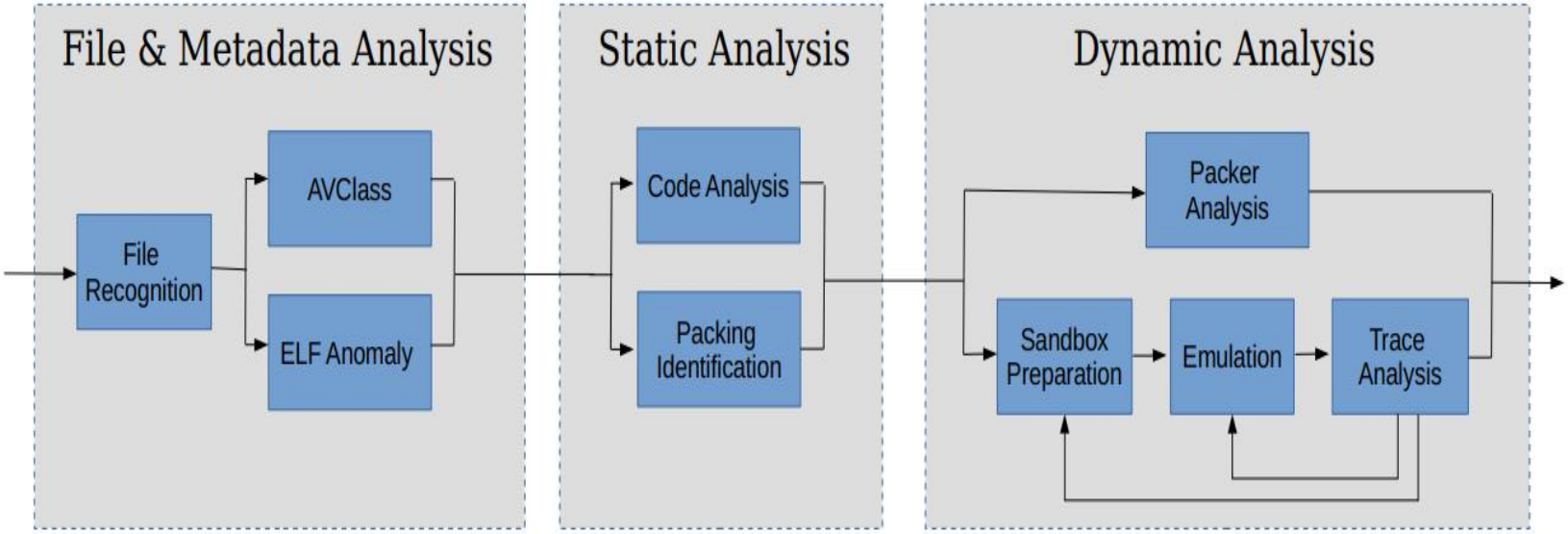


Fig. 1. Overview of our analysis pipeline.

Understanding Linux Malware

- 记录了支持Linux恶意软件分析的几种工具的设计和实现，并讨论了处理这种特殊类型的恶意文件时所涉及的挑战。
- 恶意软件开发人员经常篡改ELF标头，以欺骗分析师或使通用分析工具崩溃，面对这种情况时不同分析工具的表现。

TABLE III
ELF SAMPLES THAT CANNOT BE PROPERLY PARSED BY KNOWN TOOLS

Program	Errors on Malformed Samples
readelf 2.26.1	166 / 211
GDB 7.11.1	157 / 211
pyelftools 0.24	107 / 211
IDA Pro 7	- / 211

Understanding Linux Malware

- 论文提供了对一年内获得的10,548个Linux恶意软件样本进行的首次大规模实证研究。
- 论文提到的一些恶意软件使用的技术及使用情况

样本采用的ELF header修改技术

TABLE II
ELF HEADER MANIPULATION

Technique	Samples	Percentage
Segment header table pointing beyond file data	1	0.01%
Overlapping ELF header/segment	2	0.02%
Wrong string table index (e_shstrndx)	60	0.57%
Section header table pointing beyond file data	178	1.69%
Total Corrupted	211	2.00%

Understanding Linux Malware

样本采用的持久化策略

TABLE IV
ELF BINARIES ADOPTING PERSISTENCE STRATEGIES

Path	Samples	
	w/o root	w/ root
/etc/rc.d/rc.local	-	1393
/etc/rc.conf	-	1236
/etc/init.d/	-	210
/etc/rcX.d/	-	212
/etc/rc.local	-	11
systemd service	-	2
~/.bashrc	19	8
~/.bash_profile	18	8
X desktop autostart	3	1
/etc/cron.hourly/	-	70
/etc/crontab	-	70
/etc/cron.daily/	-	26
crontab utility	6	6
File replacement	-	110
File infection	5	26
Total	1644 (21.10%)	

Understanding Linux Malware

重命名自身的样本选择的名字

TABLE V
ELF PROGRAMS RENAMING THE PROCESS

Process name	Samples	Percentage
sshd	406	5.21%
telnetd	33	0.42%
cron	31	0.40%
sh	14	0.18%
busybox	11	0.14%
other tools	22	0.28%
empty	2034	26.11%
other *	973	12.49%
random	618	7.93%
Total	4091	52.50%

* Names not representing system utilities

K-Miner: Uncovering Memory Corruption in Linux

- K-Miner，设计是一种可有效分析大型操作系统内核（如Linux）的新框架
- 利用了内核代码的高度标准化的接口结构，以实现可扩展的指针分析并进行全局的，上下文相关的分析。
- 通过过程间分析，论文表明K-Miner可以系统并可靠地发现了几类不同的内存损坏漏洞，例如悬空指针，UAF，二次释放等漏洞。

NDSS 2018

K-Miner: Uncovering Memory Corruption in Linux

问题：

- 内核，如Linux，是用低级编程语言编写的，这些语言仅提供有限的类型和内存安全保证，从而使攻击者能够利用内存损坏漏洞对内核发起复杂的运行时攻击。
- 已经提出的许多防御措施例如控制流完整性（CFI）仅保护运行时的操作系统，而不是消除根本原因，即内核代码中的错误。
- 现有的静态分析方法都限于局部，过程内检查，并且由于庞大的内核代码库而面临严重的可扩展性挑战。

K-Miner: Uncovering Memory Corruption in Linux

K-Miner系统架构

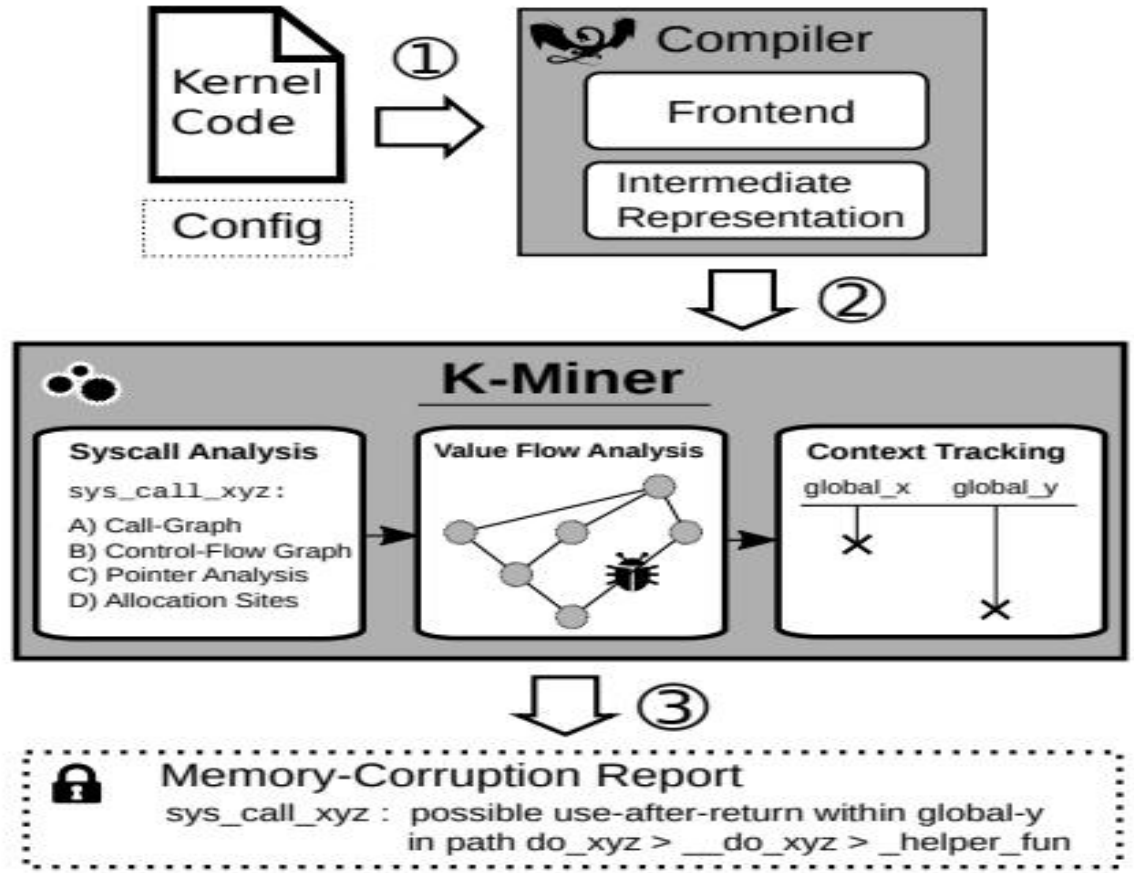


Figure 2: Overview of the different components of K-Miner.

K-Miner: Uncovering Memory Corruption in Linux

K-Miner实现概述

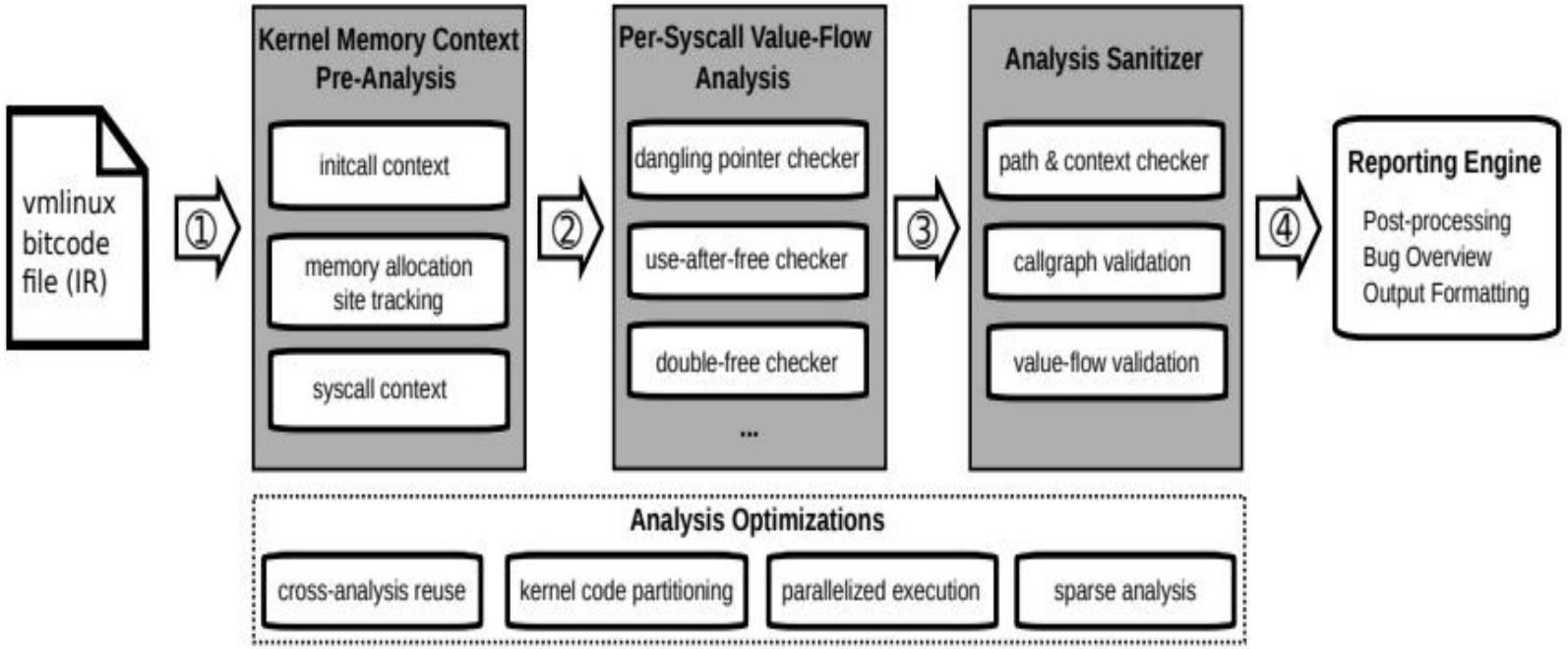


Figure 3: Overview of the K-Miner implementation: we conduct complex data-flow analysis of the Linux kernel in stages, re-using intermediate results.

K-Miner: Uncovering Memory Corruption in Linux

系统调用中的悬空指针漏洞示例

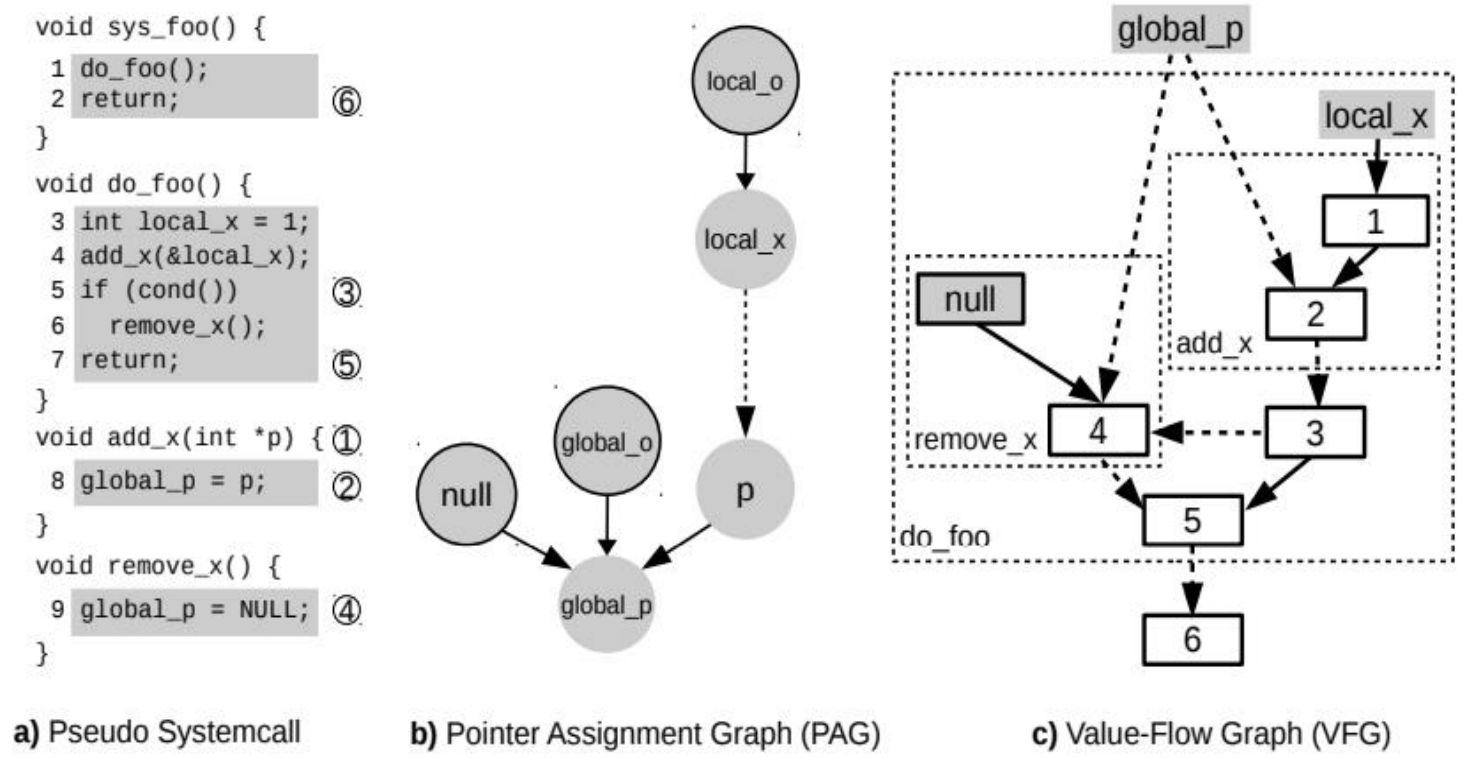


Figure 4: Example of a Dangling Pointer vulnerability in a (simplified) system call definition.

在代码中，add_x一定会被调用，而remove是有条件调用，那么可能存在do_foo函数返回后，global_p仍保存了do_foo中的局部变量指针的情况，而此时该指针已是悬空指针。

K-Miner: Uncovering Memory Corruption in Linux

贡献

- 实现对内核代码的全局静态分析：K-Miner框架允许系统地分析内核的用户空间接口并检测可能的内存损坏。为了实现对现代操作系统内核的精确的过程间静态分析研究者解决了许多挑战，例如处理大型代码库，复杂的相互依赖关系以及全局内核指针和内存对象之间的别名关系。
- 原型框架实现：论文提出的框架是可扩展的，K-Miner包括一个基于Web的用户界面，以简化报告和协作。 它还提供了基于图表的广泛分析概述以及有关警报数量和性能的统计信息。作者还发布了基于LLVM构建的K-Miner开源项目
- 广泛的评估： 论文通过将其应用于许多不同Linux版本的所有系统调用中，来严格评估Linux内核的静态分析框架实现，并通过详细的报告和统计数据突出显示框架的有效性。论文通过发现一些已知的内存损坏漏洞。研究者们报告了K-Miner在内核中发现的两个used-after-Return漏洞。

HFL: Hybrid Fuzzing on the Linux Kernel

- 本文提出了HFL，它不仅将模糊与符号执行相结合以实现混合模糊，而且还通过以下三个独特功能解决了特定于内核的模糊挑战：
  - 1）将间接控制转换为直接控制；
  - 2）通过推断系统调用顺序建立一致的系统状态，
  - 3）识别系统调用的嵌套参数类型。

NDSS 2020

HFL: Hybrid Fuzzing on the Linux Kernel

挑战：

将符号执行和模糊测试结合在一起的混合模糊化方法是发现漏洞的一种有前途的方法，因为这两种方法可以相互补充。但是，混合模糊测试应用于内核测试是具有挑战性的，因为内核的以下独特特征使混合模糊测试的简单应用效率低下：

1）（内核）具有由系统调用参数确定的间接控制转移

2）通过系统调用控制和匹配内部系统状态，

3）推断用于调用系统调用的嵌套参数类型

现有的内核测试方法要么单独使用每种技术，而没有处理此类挑战，要么仅通过静态分析来不精确地处理部分挑战

HFL: Hybrid Fuzzing on the Linux Kernel

贡献：

第一个混合内核模糊器：提出一种混合内核模糊测--HFL，利用了随机模糊测试和符号测试技术的优势

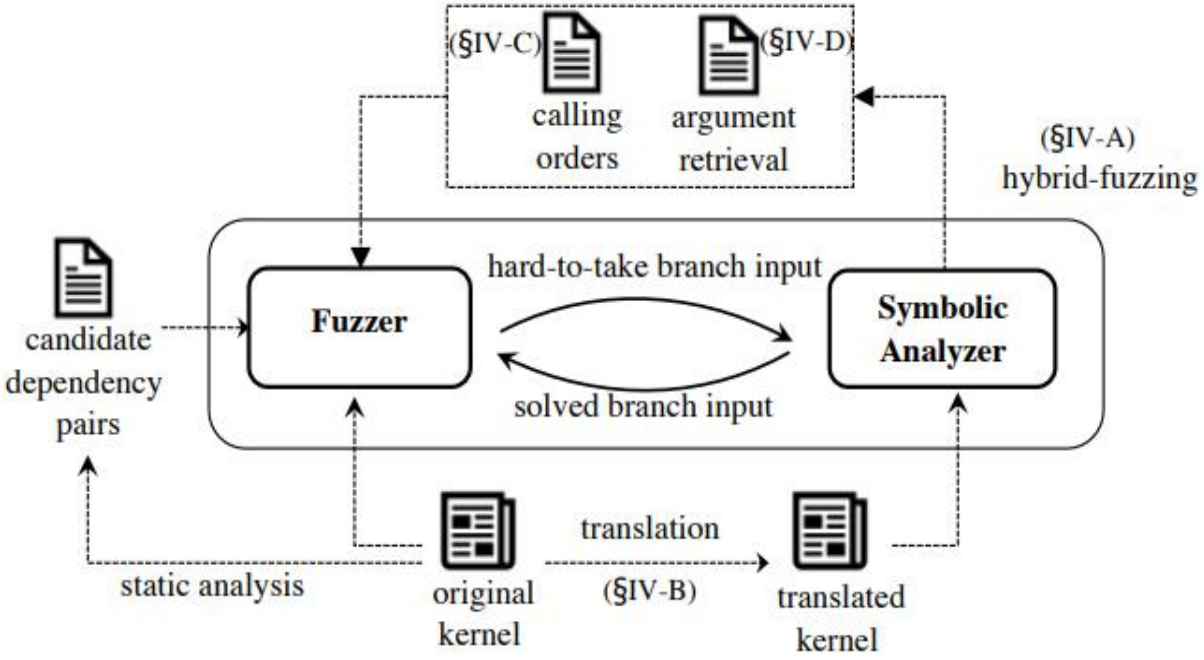


Fig. 5: Overview of HFL.

HFL: Hybrid Fuzzing on the Linux Kernel

- 贡献：

处理了特定与内核的挑战：将间接控制流转换为直接控制流，通过推断HFL过程中的系统调用顺序来保持内核内部状态的一致性。此外，有效地推断了系统调用参数的接口

```
1 // before translation
2 ret = ucma_table[hdr.cmd](...);
3
4 // after translation
5 if (hdr.cmd == RDMA_CREATE_ID)
6   ret = ucma_create_id (...);
7 else if (hdr.cmd == RDMA_DESTROY_ID)
8   ret = ucma_destroy_id (...);
9 ...
```

Fig. 6: Conversion to direct control-flow

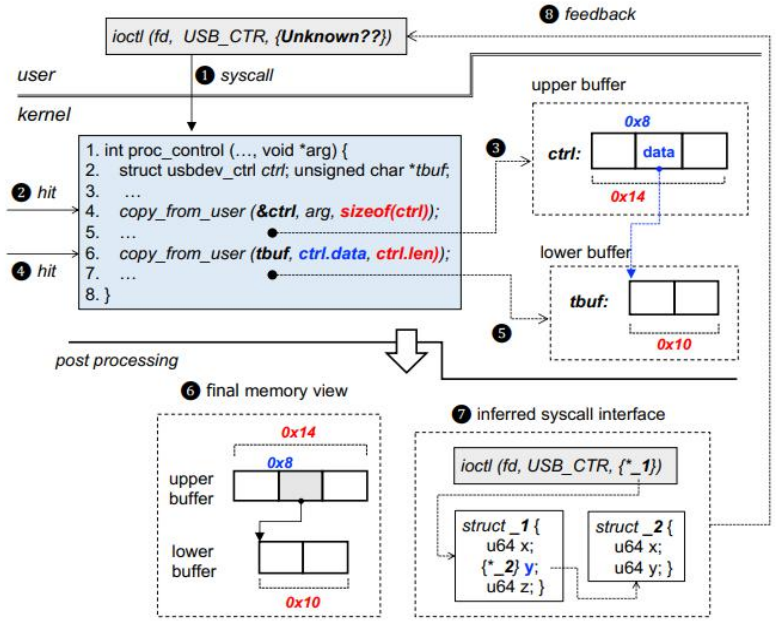


Fig. 8: Workflow of nested syscall argument retrieval.

HFL: Hybrid Fuzzing on the Linux Kernel

系统调用顺序推断工作流程:

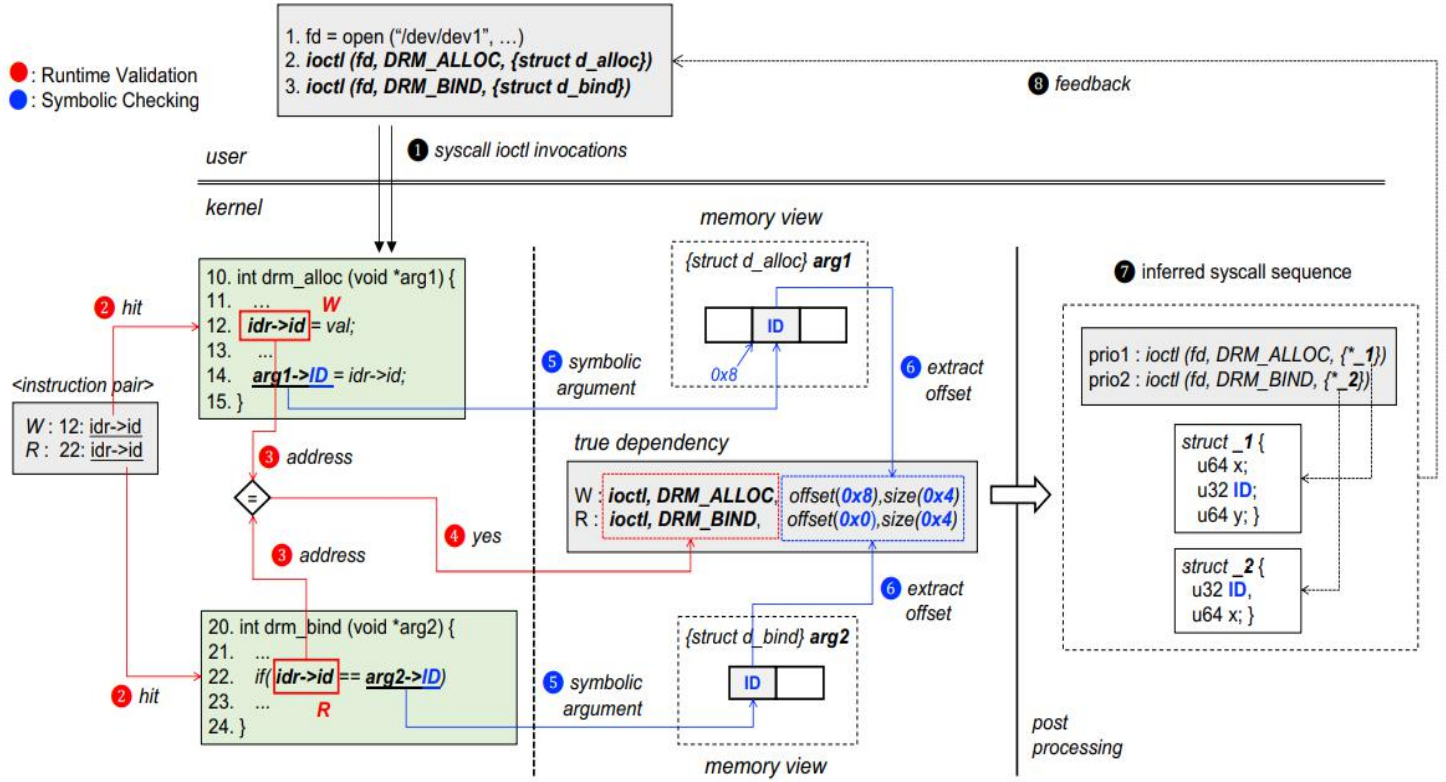


Fig. 7: Workflow of syscall sequence inference.

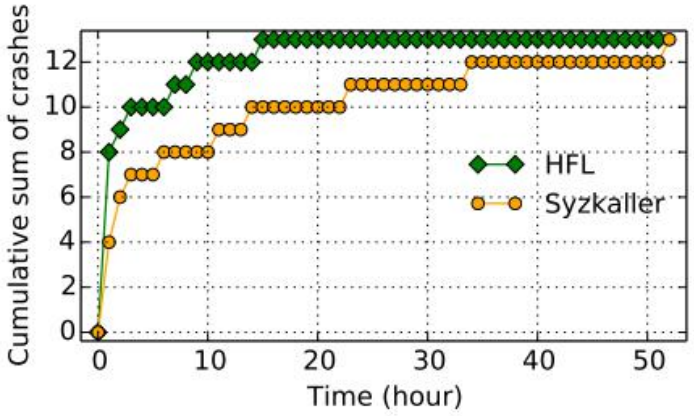
HFL: Hybrid Fuzzing on the Linux Kernel

贡献：

- 错误发现和测试：HFL在最近的Linux内核中发现了24个以前未知的漏洞。此外，使用相同数量的资源（CPU，时间等），HFL的代码覆盖率分别比Moonshine和Syzkaller高出15%和26%，超过kAFL / S2E / TriforceAFL甚至四倍。关于漏洞发现性能，HFL发现13个已知漏洞，比Syzkaller快三倍以上

	Coverage
HFL	10.5%
Moonshine	9.0%
Syzkaller	7.9%
kAFL	1.9%
TriforceAFL	0.9%
S2E	0.8%

覆盖率



HFL发现13个已知漏洞，比Syzkaller快三倍以上

HFL: Hybrid Fuzzing on the Linux Kernel

Linux内核中发现的24个以前未知的漏洞

Crash type	Description	Kernel	Subsystem	Status	Impact
integer overflow	undefined behaviour in mm/page_alloc.c	4.19-rc8	memory	patched	likely exploitable
integer overflow	undefined behaviour in net/can/bcm.c	4.19.13	network	patched	DoS
integer overflow	undefined behaviour in drivers/input/misc/uinput.c	4.19.13	drivers	patched	DoS
uninitialized variable access	undefined behaviour in fs/f2fs/extent_cache.c	5.0.7	file system	patched	DoS
memory access violation	use-after-free Read in ata_scsi_mode_select_xlat	4.17.19	drivers	confirmed	likely exploitable
memory access violation	use-after-free Read in raw_cmd_done	4.19-rc2	drivers	confirmed	likely exploitable
memory access violation	warning in pkt_setup_dev	4.19-rc2	drivers	confirmed	DoS
uninitialized variable access	uninit-value in selinux_socket_bind	4.19-rc8	network	confirmed	likely exploitable
memory access violation	undefined behaviour in drivers/block/floppy.c	4.19-rc8	drivers	confirmed	likely exploitable
integer overflow	undefined behaviour in drivers/net/ppp/ppp_generic.c	4.19-rc8	drivers	confirmed	DoS
uninitialized variable access	uninit-value in selinux_socket_connect_helper	4.19-rc8	network	confirmed	likely exploitable
integer overflow	undefined behaviour in ./include/linux/ktime.h	4.19.13	timer	confirmed	DoS
integer overflow	undefined behaviour in drivers/input/mousedev.c	4.20.0	drivers	confirmed	DoS
integer overflow	undefined behaviour in drivers/pps/pps.c	4.20.0	drivers	confirmed	DoS
memory access violation	general protection fault in spk_ttyio_ldisc_close	4.20.0	drivers	confirmed	likely exploitable
integer overflow	undefined behaviour in net/ipv4/ip_output.c	5.0-rc2	network	confirmed	DoS
integer overflow	undefined behaviour in drivers/scsi/sr_ioctl.c	5.0-rc2	drivers	confirmed	DoS
memory access violation	use-after-free Write in vgacon_scroll	4.17-rc3	drivers	reported	likely exploitable
memory access violation	use-after-free Write in do_con_write	4.17-rc3	drivers	reported	likely exploitable
task hang	task hung in drop_inmem_page	4.17.19	file system	reported	DoS
memory access violation	null-ptr-deref Write in complete	4.17.19	drivers	reported	DoS
integer overflow	undefined behaviour in fs/xfs/xfs_ioctl.c	4.19.19	file system	reported	DoS
memory access violation	undefined behaviour in fs/jfs/jfs_dmap.c	4.19.19	file system	reported	DoS
task hang	task hung in reiserfs_sync_fs	4.19.19	file system	reported	DoS

TABLE III: List of 24 previously unknown vulnerabilities in the Linux kernels discovered by HFL.

PeX: A Permission Check Analysis Framework for Linux Kernel

- 本文提出了PeX，它是Linux的静态权限检查错误检测器，它以内核源代码为输入，检查并报告缺失，不一致和冗余的权限检查问题。

- 本文还提出了KIRIN（基于内核接口的间接调用分析），这是一种新颖，精确且可扩展的间接调用分析技术。

USENIX Security 2019

PeX: A Permission Check Analysis Framework for Linux Kernel

- 内核权限检查错误示例:

```
1  int scsi_ioctl(struct scsi_device *sdev, int cmd,
   ↪  void __user *arg)
2  {
3    ...
4    case SCSI_IOCTL_SEND_COMMAND:
5      if (!capable(CAP_SYS_ADMIN) ||
       ↪  !capable(CAP_SYS_RAWIO))
6        return -EACCES;
7      return sg_scsi_ioctl(sdev->request_queue, NULL,
       ↪  0, arg);
8    ...
9  }
```

(a) sg_scsi_ioctl (Line 7) is called **with** CAP_SYS_ADMIN and CAP_SYS_RAWIO capability checks (Line 5). arg is user space controllable.

```
1  int scsi_cmd_ioctl(struct request_queue *q, ...,
   ↪  void __user *arg)
2  {
3    ...
4    case SCSI_IOCTL_SEND_COMMAND:
5      ...
6      if (!arg)
7        break;
8      err = sg_scsi_ioctl(q, bd_disk, mode, arg);
9      break;
10   ...
11   return err;
12 }
```

(b) sg_scsi_ioctl (Line 8) is called **without** capability checks. arg is user space controllable.

```
1  int sg_scsi_ioctl(struct request_queue *q, struct
   ↪  gendisk *disk, fmode_t mode, struct
   ↪  scsi_ioctl_command __user *sic)
2  {
3    ...
4    err = blk_verify_command(req->cmd, mode);
5    ...
6    return err;
7  }
8
9  int blk_verify_command(unsigned char *cmd, fmode_t
   ↪  mode)
10 {
11   ...
12   if (capable(CAP_SYS_RAWIO))
13     return 0;
14   ...
15   return -EPERM;
16 }
```

(c) sg_scsi_ioctl calls blk_verify_command, which checks CAP_SYS_RAWIO capability.

一条路径a->c 检查了两次CAP_SYS_RAWIO和一次CAP_SYS_ADMIN，另一条只检查了一次CAP_SYS_RAWIO，说明其中一条路径检查冗余，另一条检查缺失。

PeX: A Permission Check Analysis Framework for Linux Kernel

KIRIN：基于内核接口的间接调用分析

- Linux内核非常频繁地使用间接调用，但是众所周知，其静态调用图分析非常困难，KIRIN利用内核抽象接口来实现精确而可扩展的间接调用分析。
- （1）Linux内核中几乎所有（95%）的间接调用都源自内核接口
- （2）内核接口的类型在LLVM IR中保留在其初始化位置（定义了函数指针）和间接调用位置（使用了函数指针）

PeX: A Permission Check Analysis Framework for Linux Kernel

- PeX以已知权限检查作为输入，以识别其基本权限检查，并利用它们作为查找其他权限检查包装程序的基础。此外，PeX提出了一种基于支配者分析的解决方案，以自动推断权限检查和特权功能之间的映射。

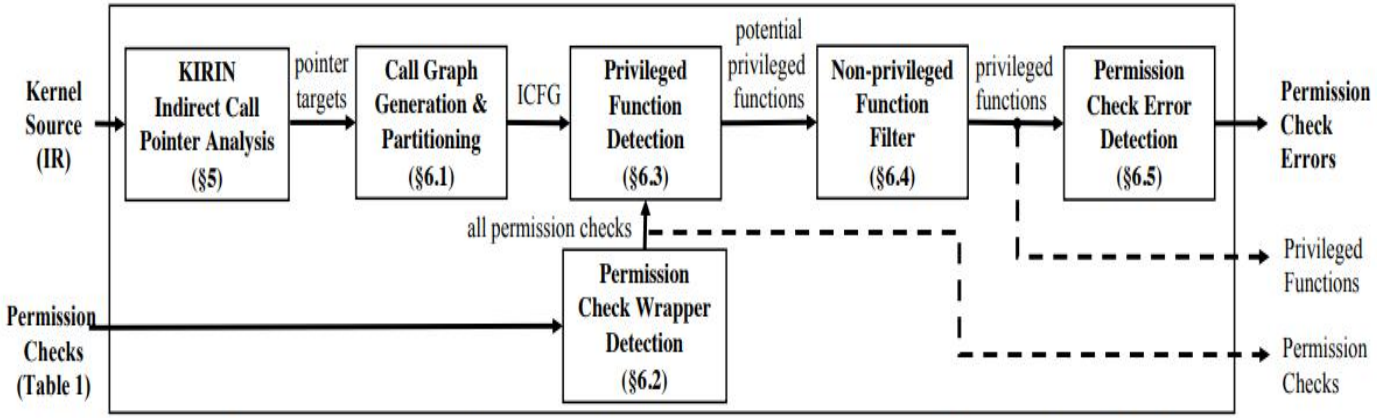


Figure 6: PeX static analysis architecture. PeX takes as input kernel source code and permission checks, and reports as output permission check errors. PeX also produces mappings between identified permission checks and privileged functions as output.

PeX: A Permission Check Analysis Framework for Linux Kernel

成果：

PeX报 告了36个DAC，Capabilities和LSM权限检查错误，其中14个已经由内核开发人员确认。如下表所示。

Table 6: Bugs Reported By PeX. Confirmed or Ignored.

Type-#	File	Function	Description	Status
DAC-1	fs/btrfs/send.c	`btrfs_send`	missing DAC check when traversing a snapshot	C
DAC-2	fs/ecryptfs/inode.c	`ecryptfs_removexattr(),_setxattr()`	missing `xattr_permission()`	C
DAC-3	fs/ecryptfs/inode.c	`ecryptfs_listxattr()`	missing `xattr_permission()`	C
CAP-4	drivers/char/random.c	`write_pool(), credit_entropy_bits()`	missing `CAP_SYS_ADMIN`	C
CAP-5	drivers/scsi/sg.c	`sg_scsi_ioctl()`	missing `CAP_SYS_ADMIN` or `CAP_RAW_IO`	I
CAP-6	drivers/block/pktcdvd.c	`add_store(), remove_store()`	missing `CAP_SYS_ADMIN`	I
CAP-7	drivers/char/nvram.c	`nvram_write()`	missing `CAP_SYS_ADMIN`	I
CAP-8	drivers/firmware/efi/efivars.c	`efivar_entry_set()`	missing `CAP_SYS_ADMIN`	C
CAP-9	net/rfkill/core.c	`rfkill_set_block(), rfkill_fop_write()`	missing `CAP_NET_ADMIN`	C
CAP-10	block/scsi_ioctl.c	`mmc_rpmb_ioctl()`	missing verify_command or `CAP_SYS_ADMIN`	I
CAP-11	drivers/platform/x86/thinkpad_acpi.c	`acpi_evalf()`	missing `CAP_SYS_ADMIN`	I
CAP-12	drivers/md/dm.c	`dm_blk_ioctl()`	missing `CAP_RAW_IO`	I
CAP-13	block/bsg.c	`bsg_ioctl`	inconsistent/missing `CAP_SYS_ADMIN`	C
CAP-14	kernel/sys.c	`prctl_set_mm_exe_file`	inconsistent capability check	I
CAP-15	kernel/sys.c	`prctl_set_mm_exe_file`	inconsistent capability and namespace check	I
CAP-16	block/scsi_ioctl.c	`blk_verify_command`	redundant check `CAP_SYS_RAWIO`	I
LSM-17	fs/ecryptfs/inode.c	`ecryptfs_removexattr(), _setxattr()`	missing `security_inode_removexattr()`	C
LSM-18	mm/mmap.c	`remap_file_pages`	missing `security_mmap_file()`	I
LSM-19	fs/binfmt_elf.c	`load_elf_binary()`	missing `security_kernel_read_file`	I
LSM-20	fs/binfmt_elf.c	`load_elf_library()`	missing `security_kernel_read_file`	I
LSM-21	fs/xfs/xfs_ioctl.c	`xfs_file_ioctl()`	missing `security_inode_readlink()`	C
LSM-22	kernel/workqueue.c	`wq_nice_store()`	missing `security_task_setnice()`	C
LSM-23	fs/ecryptfs/inode.c	`ecryptfs_listxattr()`	missing `security_inode_listxattr`	C
LSM-24	include/linux/sched.h	`comm_write()`	missing `security_task_prctl()`	C
LSM-25	fs/binfmt_misc.c	`load_elf_binary()`	missing `security_bprm_set_creds()`	I
LSM-26	drivers/android/binder.c	`binder_set_nice`	missing `security_task_setnice()`	I
LSM-27	fs/ocfs2/cluster/tcp.c	`o2net_start_listening()`	missing `security_socket_bind`	I
LSM-28	fs/ocfs2/cluster/tcp.c	`o2net_start_listening()`	missing `security_socket_listen`	I

PeX: A Permission Check Analysis Framework for Linux Kernel

Table 6: Bugs Reported By PeX. Confirmed or Ignored.

Type-#	File	Function	Description	Status
DAC-1	fs/btrfs/send.c	`btrfs_send`	missing DAC check when traversing a snapshot	C
DAC-2	fs/ecryptfs/inode.c	`ecryptfs_removexattr(),_setxattr()`	missing `xattr_permission()`	C
DAC-3	fs/ecryptfs/inode.c	`ecryptfs_listxattr()`	missing `xattr_permission()`	C
CAP-4	drivers/char/random.c	`write_pool(), credit_entropy_bits()`	missing `CAP_SYS_ADMIN`	C
CAP-5	drivers/scsi/sg.c	`sg_scsi_ioctl()`	missing `CAP_SYS_ADMIN` or `CAP_RAW_IO`	I
CAP-6	drivers/block/pktcdvd.c	`add_store(), remove_store()`	missing `CAP_SYS_ADMIN`	I
CAP-7	drivers/char/nvram.c	`nvram_write()`	missing `CAP_SYS_ADMIN`	I
CAP-8	drivers/firmware/efi/efivars.c	`efivar_entry_set()`	missing `CAP_SYS_ADMIN`	C
CAP-9	net/rfkill/core.c	`rfkill_set_block(), rfkill_fop_write()`	missing `CAP_NET_ADMIN`	C
CAP-10	block/scsi_ioctl.c	`mmc_rpmb_ioctl()`	missing verify_command or `CAP_SYS_ADMIN`	I
CAP-11	drivers/platform/x86/thinkpad_acpi.c	`acpi_evalf()`	missing `CAP_SYS_ADMIN`	I
CAP-12	drivers/md/dm.c	`dm_blk_ioctl()`	missing `CAP_RAW_IO`	I
CAP-13	block/bsg.c	`bsg_ioctl`	inconsistent/missing `CAP_SYS_ADMIN`	C
CAP-14	kernel/sys.c	`prctl_set_mm_exe_file`	inconsistent capability check	I
CAP-15	kernel/sys.c	`prctl_set_mm_exe_file`	inconsistent capability and namespace check	I
CAP-16	block/scsi_ioctl.c	`blk_verify_command`	redundant check `CAP_SYS_RAWIO`	I
LSM-17	fs/ecryptfs/inode.c	`ecryptfs_removexattr(), _setxattr()`	missing `security_inode_removexattr()`	C
LSM-18	mm/mmap.c	`remap_file_pages`	missing `security_mmap_file()`	I
LSM-19	fs/binfmt_elf.c	`load_elf_binary()`	missing `security_kernel_read_file`	I
LSM-20	fs/binfmt_elf.c	`load_elf_library()`	missing `security_kernel_read_file`	I
LSM-21	fs/xfs/xfs_ioctl.c	`xfs_file_ioctl()`	missing `security_inode_readlink()`	C
LSM-22	kernel/workqueue.c	`wq_nice_store()`	missing `security_task_setnice()`	C
LSM-23	fs/ecryptfs/inode.c	`ecryptfs_listxattr()`	missing `security_inode_listxattr`	C
LSM-24	include/linux/sched.h	`comm_write()`	missing `security_task_prctl()`	C
LSM-25	fs/binfmt_misc.c	`load_elf_binary()`	missing `security_bprm_set_creds()`	I
LSM-26	drivers/android/binder.c	`binder_set_nice`	missing `security_task_setnice()`	I
LSM-27	fs/ocfs2/cluster/tcp.c	`o2net_start_listening()`	missing `security_socket_bind`	I
LSM-28	fs/ocfs2/cluster/tcp.c	`o2net_start_listening()`	missing `security_socket_listen`	I
LSM-29	fs/dlm/lowcomms.c	`tcp_create_listen_sock`	missing `security_socket_bind`	I
LSM-30	fs/dlm/lowcomms.c	`tcp_create_listen_sock`	missing `security_socket_listen`	I
LSM-31	fs/dlm/lowcomms.c	`sctp_listen_for_all`	missing `security_socket_listen`	I
LSM-32	net/socket.c	`kernel_bind`	missing `security_socket_bind`	I
LSM-33	net/socket.c	`kernel_listen`	missing `security_socket_listen`	I
LSM-34	net/socket.c	`kernel_connect`	missing `security_socket_connect`	I
LSM-35	fs/ocfs2/cluster/tcp.c	`o2net_start_listening()`	redundant `security_socket_create`	C
LSM-36	fs/ocfs2/cluster/tcp.c	`o2net_open_listening_sock()`	redundant `security_socket_create`	C

KEPLER: Facilitating Control-flow Hijacking Primitive Evaluation for Linux Kernel Vulnerabilities

- 自动利用漏洞利用是一个充满挑战的问题。任务中具有挑战性的部分是将已识别的可利用状态（利用原语）连接到触发代码重用（例如ROP）有效负载的执行。控制流劫持原语是最常见的利用功能之一。

- 本文提出了KEPLER通过自动生成“single shot”漏洞利用链来促进漏洞利用的产生。 KEPLER接受控制流劫持原语作为输入，并通过利用普遍的内核编码样式和相应的工具来符号化利用链来引导任何内核ROP有效负载。与先前的自动漏洞利用生成技术和先前的内核利用技术进行比较，显示KEPLER有效地促进了Linux内核中控制流劫持原语的评估。

USENIX Security 2019

挑战：

1、漏洞利用缓解措施

新的硬件功能，编译器辅助工具，敏感数据对象保护和基于虚拟化的虚拟机管理程序等。

2、漏洞利用路径陷阱

漏洞利用原语的副作用可能会在中途终止漏洞利用。与漏洞利用原语一起发生的内存损坏可能使第二次尝试触发该原语受挫，因为漏洞利用路径不可避免地会包含触发漏洞利用的意外终止的指令。 这种终止可能是无效内存访问引起的内核panic或内核线程中的无限循环。

3、不合适的漏洞利用原代码。

缺少堆栈关键位置gadget（这是执行ROP攻击的关键步骤），并且对通用寄存器的控制不足会导致原始漏洞利用不当。

贡献：

内核“single shot”利用。 论文提出了一种code-reuse漏洞利用技术，该技术在现代Linux内核缓解措施和原语本身所构成的各种约束下，将单个不合适的控制流劫持原语转换为任意ROP有效载荷执行。所提出的技术利用了流行的内核编码风格和相应的gadget，因此很难被击败。该计算开发链的方法是可自动化的，因为gadget拼接问题可以在合理大小的搜索空间上转换为搜索问题。另外，此技术的“single shot”特性使它适合于易于意外终止的漏洞，因为它避免了多次强调控制流劫持原语的情况。

用于Linux内核的半自动漏洞利用生成器。 我们使用包括IDA SDK，QEMU / KVM和angr在内的一系列工具实施KEPLER。从用户提供的控制流劫持原语开始，KEPLER分析Linux内核二进制文件，跟踪有用的内核gadget，并自动生成许多gadget链，以启动“single shot”利用并绕过内核缓解措施。它不需要内核源代码，并且可以应用于剥离的内核映像。

KEPLER: Facilitating Control-flow Hijacking Primitive Evaluation for Linux Kernel Vulnerabilities

KERPLER设计以及一个内核gadget示例

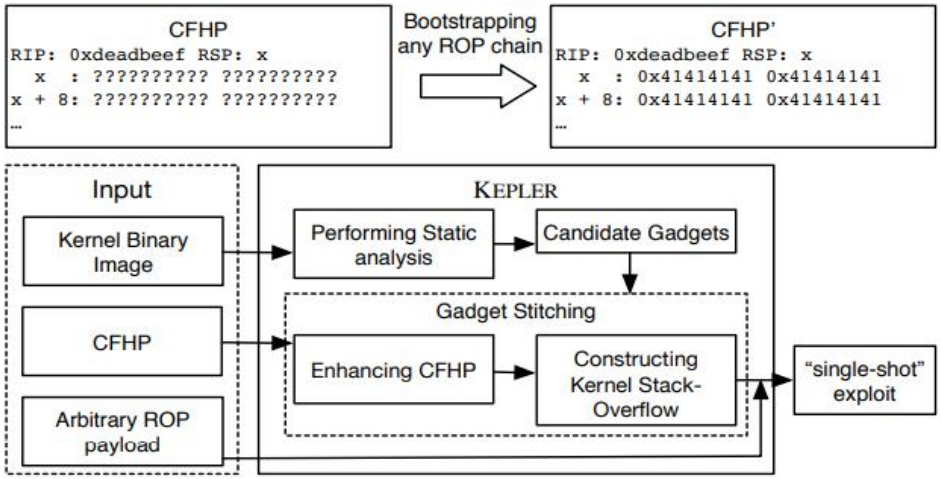


Figure 2: Overall of KEPLER's design.

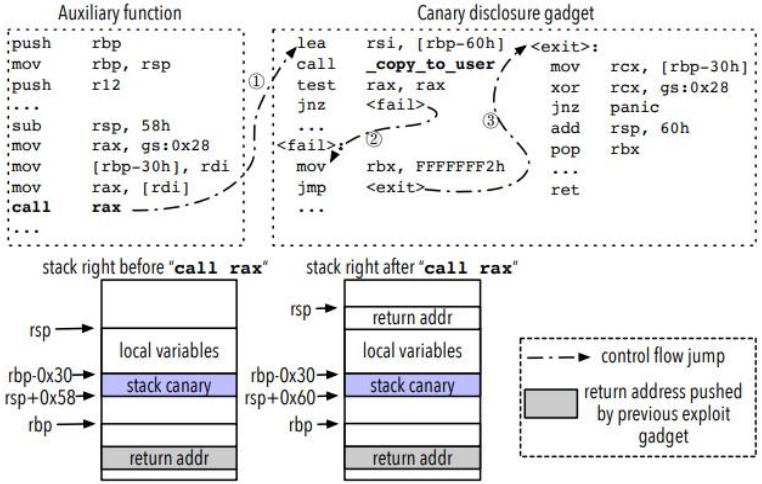


Figure 6: An example of canary disclosure gadget and its corresponding auxiliary gadget.

贡献：

实际影响。 我们使用16个实际内核漏洞和3个最近发布的CTF例子系统地评估了KEPLER的有效性和效率。 给定一个内核控制流劫持原语，我们证明KEPLER通常可以生成数以万计的不同利用链，并且能够绕过内核缓解并成功执行利用。 我们证明，KEPLER可以在不到约50分钟的时间内输出针对内核漏洞的第一个有效漏洞。

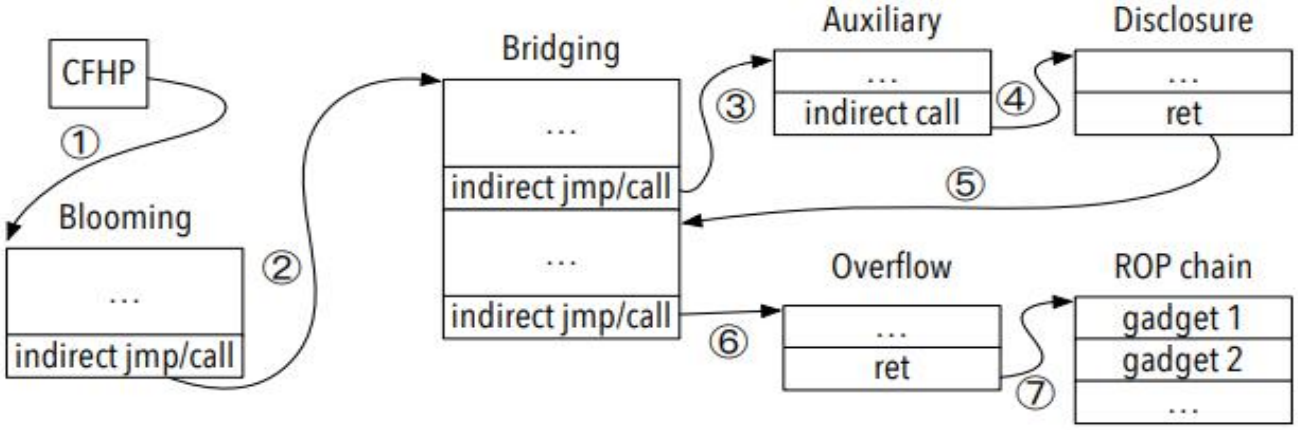


Figure 8: An illustration of how KEPLER stitches various kernel gadgets for ultimate exploitation.

LBM: A Security Framework for Peripherals within the Linux Kernel

- 外围设备越来越多地通过少量标准化的通信协议进行连接，包括USB，蓝牙和NFC。主机操作系统负责管理这些设备。但是，恶意外围设备可以从OS请求附加功能，从而导致系统受损，或者可以伪造数据包来利用OS软件堆栈中的漏洞。迄今为止，针对恶意外围设备的防御仅部分覆盖外围设备的攻击面，并且仅限于特定协议（例如USB）。

- 本文提出了Linux (e)BPF Modules (LBM)，这是一个通用的安全框架，它提供了一个统一的API，用于对Linux内核中的恶意外围设备实施保护。 LBM利用eBPF数据包过滤机制来提高性能和可扩展性，并提供了高级语言来促进强大过滤功能的开发。

S&P 2019

LBM: A Security Framework for Peripherals within the Linux Kernel

➢ 贡献：

设计并实现LBM通用安全框架，以防御恶意外围设备。LBM核心被设计为基于eBPF的高性能数据包过滤框架。提供LBM挂钩以扩展对不同外围子系统的支持。

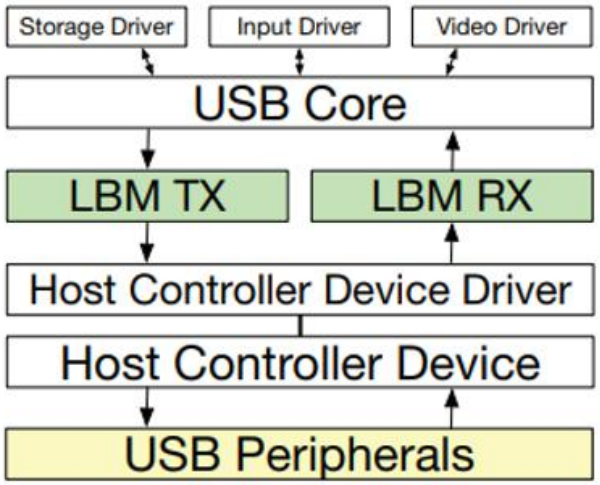


Figure 2: LBM hooks inside the USB subsystem.

USB子系统的LBM挂钩架构

LBM: A Security Framework for Peripherals within the Linux Kernel

LBM架构

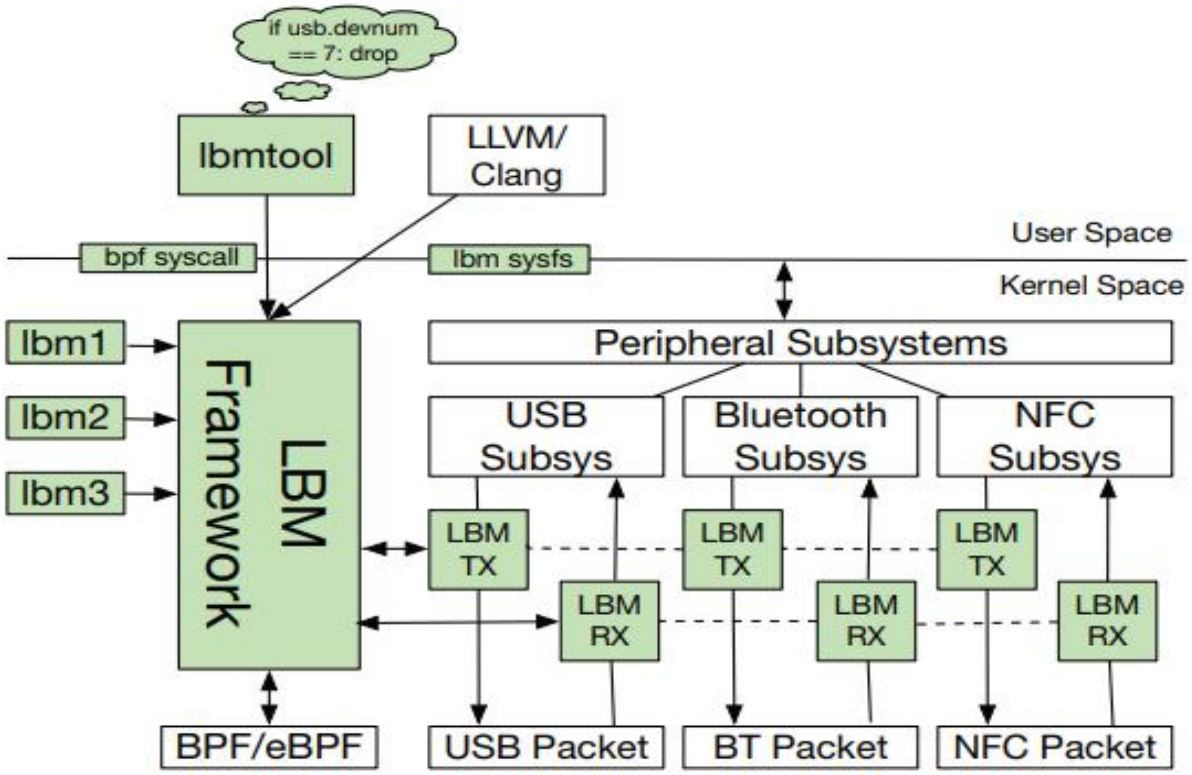


Figure 1: LBM Architecture.

LBM: A Security Framework for Peripherals within the Linux Kernel

设备高级过滤语言与工具：

开发高级过滤语言，以方便编写LBM规则。用户可以使用类似于PCAP的 高级语言编写LBM规则，以对外围数据包应用不同的策略，从而避免使用复杂的低级BPF汇编语言编写过滤器。我们的用户空间 LBM工具实用程序将 LBM规则转换为eBPF指令，并将其加载到LBM内核

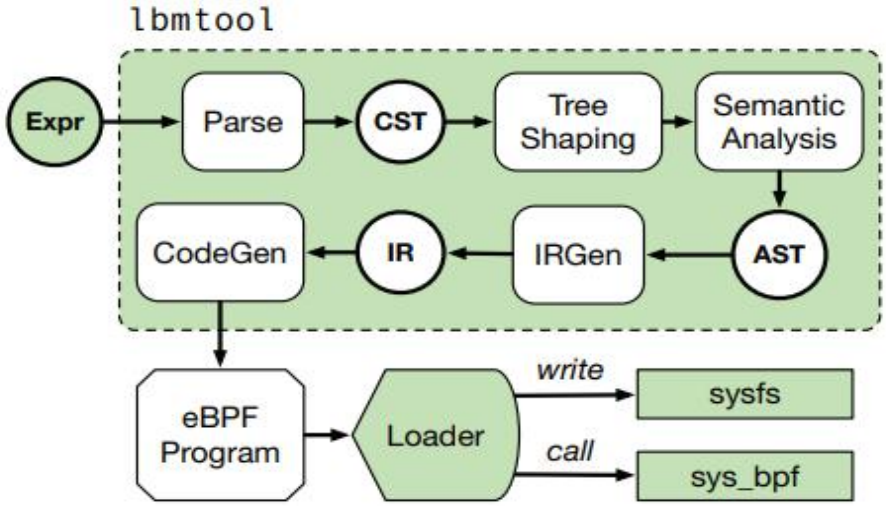


Figure 3: The flow of LBMTOOL in compiling LBM rules to eBPF programs and loading them into the running kernel.

LBM: A Security Framework for Peripherals within the Linux Kernel

➢贡献：

LBM支持USB，蓝牙和NFC。 通过将有用的协议字段暴露给用户空间并扩展，作者扩展了LBM以支持多种外围协议，扩展了LBM工具识别不同外围设备的LBM规则。通过在LBM框架下统一并完全实现USBFILTER和USBFirewall防御，我们展示了LBM的可扩展性。

Feature	USBFILTER	USBFirewall	LBM
Plugin Modules	✓		✓
Stack Protection		✓	✓
User-defined Rules	✓		✓
TX Path Mediation	✓		✓
RX Path Mediation		✓	✓
Multiple Protocols			✓

Table I: LBM compared to USBFILTER and USBFirewall. LBM unifies USBFILTER and USBFirewall, providing a superset their properties via extensible protocol support.

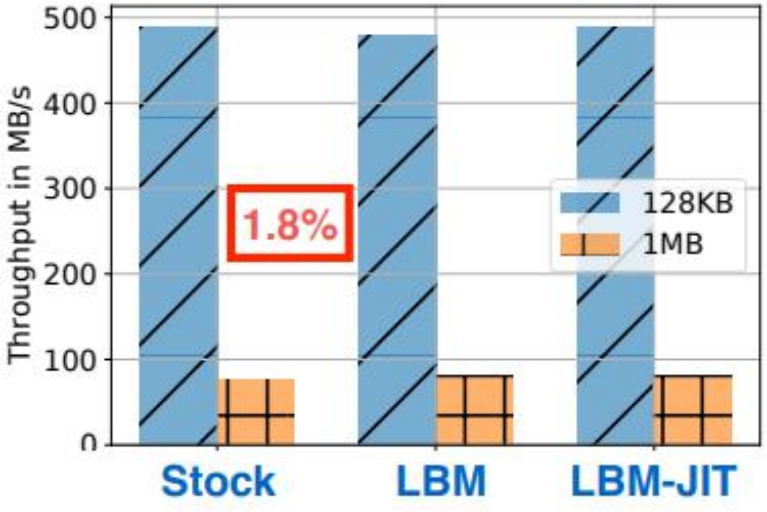
Feature	USBFILTER	USBFirewall	LBM
Filter Mechanism	C	C	eBPF
User-space DSL	CNF	N/A	PCAP DSL
Acceleration	Short Circuit	N/A	JIT

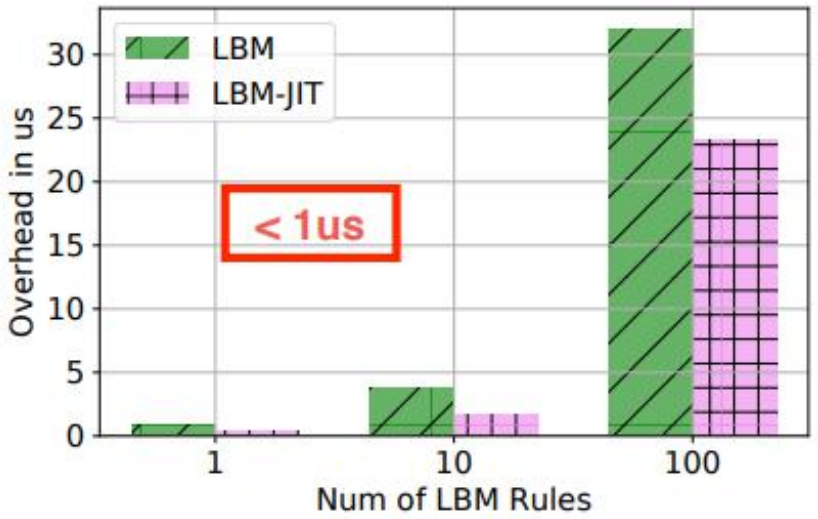
Table II: LBM vs. USBFILTER vs. USBFirewall, specifically with respect to filter design of each.

LBM: A Security Framework for Peripherals within the Linux Kernel

评估结果：

评估性能并分析针对外围攻击的覆盖范围。 通过应用适当的LBM规则，我们可以防御所有已知的外围设备攻击。我们的微观基准测试表明，在大多数情况下，LBM引入的一般开销在1微秒以内，宏基准测试表明LBM具有比其他解决方案更好的性能，对应用程序吞吐量的影响可以忽略不计。





WireGuard: Next Generation Kernel Network Tunnel

WireGuard是一个安全的网络隧道，在第3层上运行，为Linux实现内核虚拟网络接口，旨在替换大多数用例的IPsec以及流行的用户空间和/或基于TLS的解决方案（如OpenVPN），同时更加安全，性能更高且易于使用。

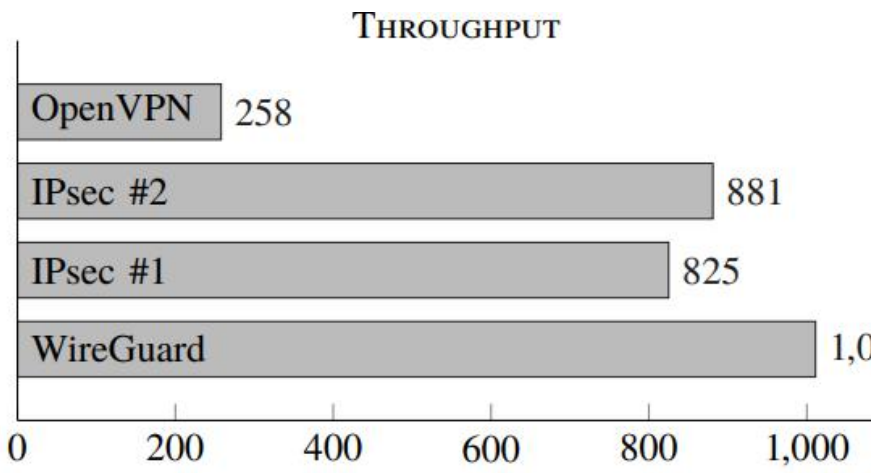
WireGuard可以轻松地用不到4,000行代码在Linux上实现，从而易于审核和验证

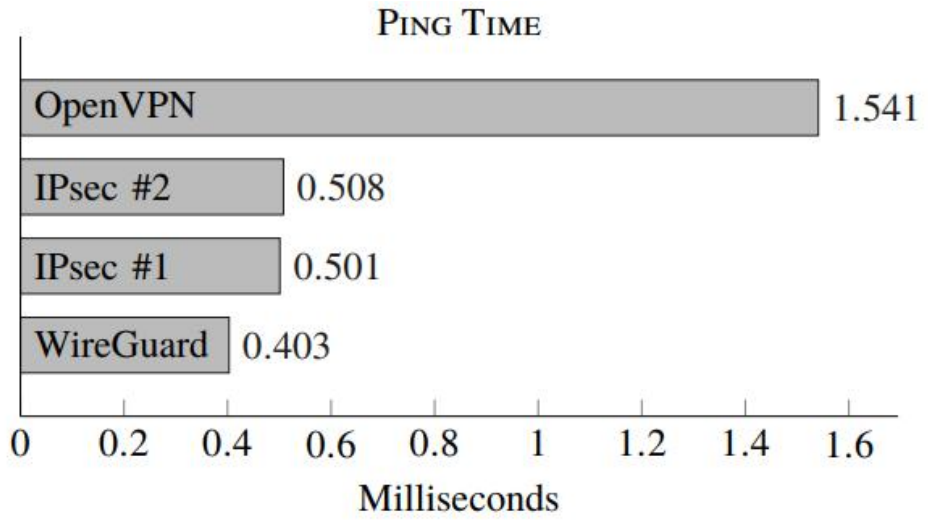
NDSS 2017

WireGuard: Next Generation Kernel Network Tunnel

- 虚拟隧道接口基于安全隧道的建议基本原理：对等公钥和隧道源IP地址之间的关联。
- 使用基于NoiseIK的单次往返密钥交换，并使用新颖的计时器状态机机制对用户透明地处理所有会话的创建。
- 像OpenSSH一样使用短预共享静态密钥（Curve25519点）进行相互身份验证
- 使用ChaCha20Poly1305身份验证加密将数据包封装在UDP中可以实现传输速度。
- IP绑定cookie的改进形式用于缓解拒绝服务攻击，从而大大改进了IKEv2和DTLS的cookie机制以添加加密和身份验证。

WireGuard: Next Generation Kernel Network Tunnel





对于这两个指标，WireGuard都优于OpenVPN和IPsec两种模式。在OpenVP N和IPsec的吞吐量测试期间，CPU利用率为100%，但在WireGuard的测试中并未完全利用CPU，这表明WireGuard能够完全实现千兆位以太网链路的加密通信。

参考文献

- Gens, D. , Schmitt, S. , Davi, L. , & Sadeghi, A. R. . (2018). K-Miner: Uncovering Memory Corruption in Linux. Network and Distributed System Security Symposium.

- Tong Zhang, Wenbo Shen, Dongyoon Lee, Changhee Jung, Ahmed M. Azab, Ruowen Wang: PeX: A Permission Check Analysis Framework for Linux Kernel. USENIX Security Symposium 2019: 1205-1220

- Cozzi, E. , Graziano, M. , Fratantonio, Y. , & Balzarotti, D. . (0). Understanding Linux Malware. Understanding Linux Malware. IEEE Computer Society.

- Donenfeld, J. A. . (2017). WireGuard: Next Generation Kernel Network Tunnel. Network & Distributed System Security Symposium.

参考文献

- Kim, K. , Jeong, D. R. , Kim, C. H. , Jang, Y. , & Lee, B. . (2020). HFL: Hybrid Fuzzing on the Linux Kernel. Network and Distributed System Security Symposium.

- Tian, D. J. , Hernandez, G. , Choi, J. I. , Frost, V. , & Butler, K. R. B. . (2019). LBM: A Security Framework for Peripherals within the Linux Kernel. 2019 IEEE Symposium on Security and Privacy (SP). IEEE.

- Wei Wu, Yueqi Chen, Xinyu Xing, Wei Zou: KEPLER: Facilitating Control-flow Hijacking Primitive Evaluation for Linux Kernel Vulnerabilities. USENIX Security Symposium 2019: 1187-1204