2017

COMPANY
INTRODUCTION OF 360

360公司

Linux 内核漏洞的分析与利用

——360 CERT 罗军

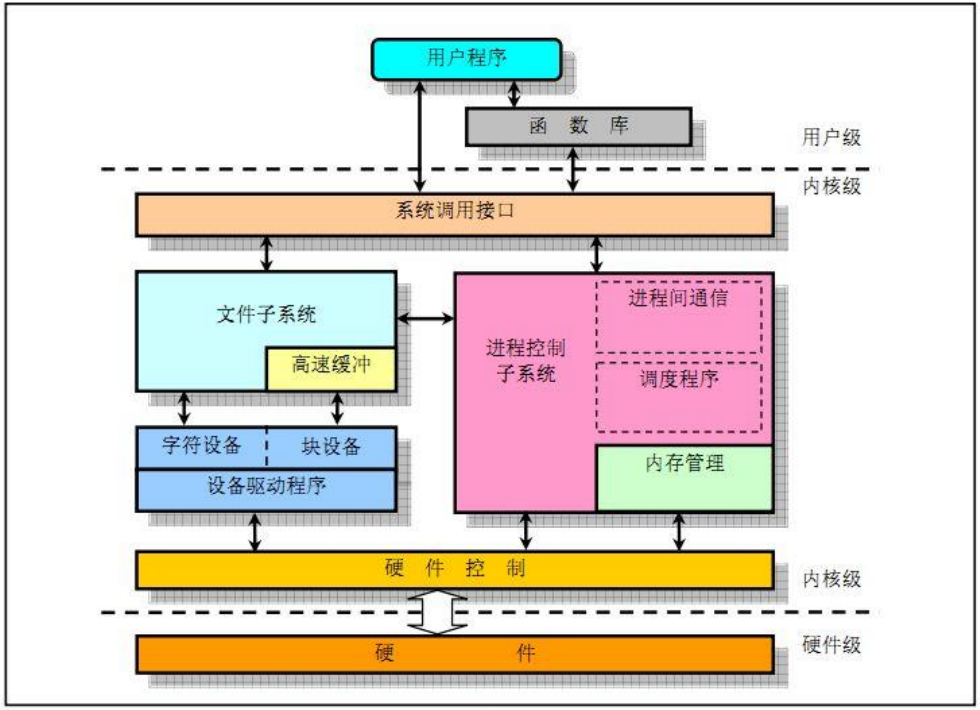
议题大纲

- LINUX内核架构
  - linux内核架构图
  - linux内核源码目录结构
- LINUX内核安全
  - linux内核漏洞现状
  - linux内核的攻击面
  - linux内核的漏洞类型
  - linux内核的防护措施
- 漏洞分析
  - 漏洞背景
  - 漏洞形成原因分析
  - 调试过程
  - 补丁分析
  - 影响版本
  - 提权演示
- 议题总结
  - 实际分析总结
  - 一些资料
  - 部门介绍
  - Q&A

一、LINUX内核架构

- LINUX内核架构图
- LINUX内核源码目录结构

Linux内核架构图



备注：图片来自网上

Linux内核源码目录结构

```
➜  linux-4.13 ls
arch            drivers   kernel           net          usr
block           firmware  lib              README       virt
built-in.o      fs        MAINTAINERS      samples      vmlinux
certs           include   Makefile         scripts      vmlinux-gdb.py
COPYING         init      mm               security     vmlinux.o
CREDITS         ipc       modules.builtin  sound
crypto          Kbuild    modules.order    System.map
Documentation   Kconfig   Module.symvers   tools
```

include/ ---- 内核头文件，需要提供给外部模块（例如用户空间代码）使用。
kernel/ ---- Linux内核的核心代码，包含进程调度子系统，以及和进程调度相关的模块。
mm/ ---- 内存管理子系统
fs/ ---- VFS子系统
net/ ---- 不包括网络设备驱动的网络子系统
ipc/ ---- IPC（进程间通信）子系统。

Linux内核源码目录结构

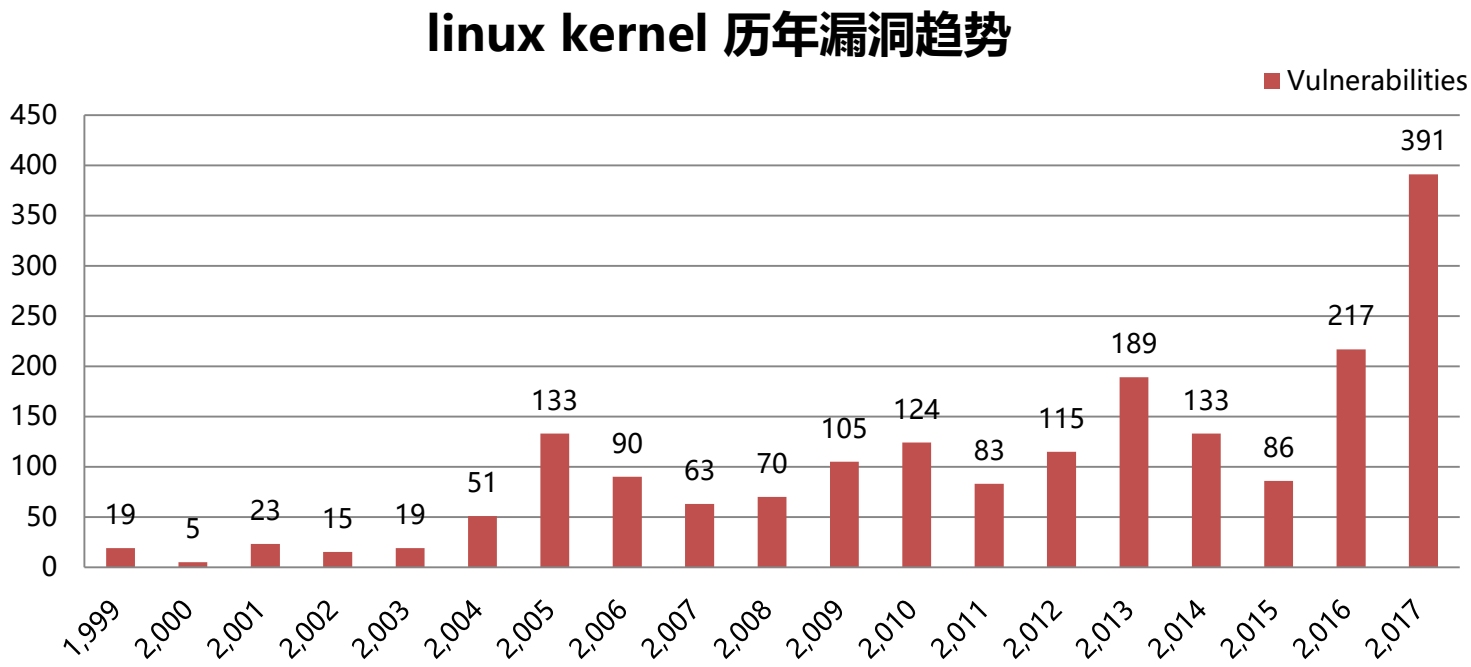
arch/ ---- 体系结构相关的代码，例如arm, x86等等。
init/ ---- Linux系统启动初始化相关的代码。
block/ ---- 提供块设备的层次。
sound/ ---- 音频相关的驱动及子系统，可以看作“音频子系统”。
drivers/ ---- 设备驱动（在Linux kernel 3.10中，设备驱动占了49.4的代码量）。
lib/ ---- 实现需要在内核中使用的库函数，例如CRC、FIFO、list、MD5等。
crypto/ ----- 加密、解密相关的库函数。
security/ ---- 提供安全特性（SELinux）。
virt/ ---- 提供虚拟机技术（KVM等）的支持。
usr/ ---- 用于生成initramfs的代码。
firmware/ ---- 保存用于驱动第三方设备的固件。
samples/ ---- 一些示例代码。
tools/ ---- 一些常用工具，如性能剖析、自测试等。
Kconfig, Kbuild, Makefile, scripts/ ---- 用于内核编译的配置文件、脚本等。
COPYING ---- 版权声明。
MAINTAINERS ----维护者名单。
CREDITS ---- Linux主要的贡献者名单。
Documentation, README ---- 帮助、说明文档。

二、LINUX内核安全

- LINUX内核漏洞现状
- LINUX内核的攻击面
- LINUX内核的漏洞类型
- LINUX内核的防护措施

Linux内核的漏洞现状

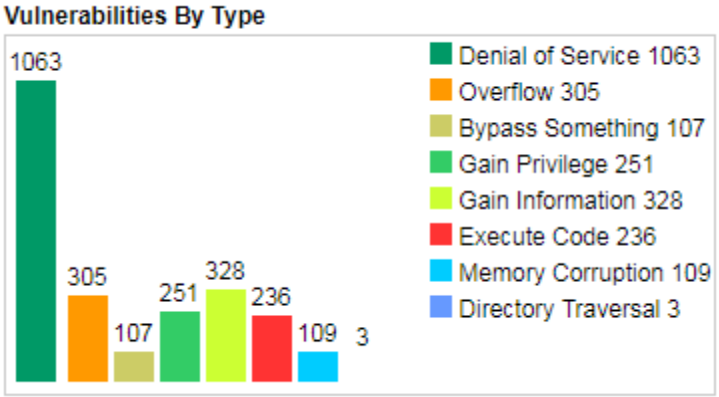
参考链接[1]



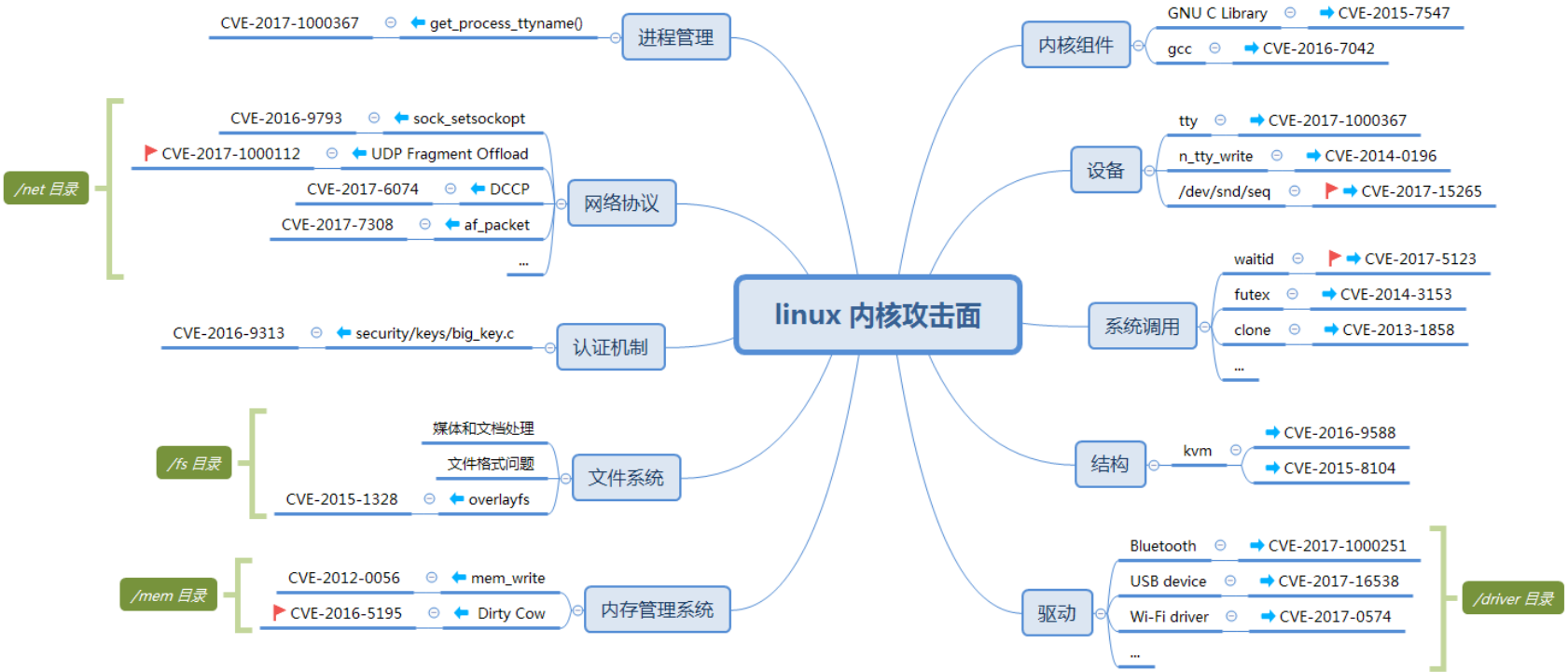
Linux内核漏洞现状

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	19	7		3						1		2			
2000	5	3										1			
2001	23	7													
2002	15	3													
2003	19	8													
2004	51	20													
2005	133	90													
2006	90	61													
2007	63	41													
2008	70	44													
2009	105	66													5
2010	124	67													5
2011	83	62													1
2012	115	83													
2013	189	101													7
2014	133	89													10
2015	86	55													
2016	217	153	5	38	18					12	35	52			1
2017	391	103	167	37	18			1		14	84	31			
Total	1931	1063	236	305	109			3		107	328	251			29
% Of All		55.0	12.2	15.8	5.6	0.0	0.0	0.2	0.0	5.5	17.0	13.0	0.0	0.0	

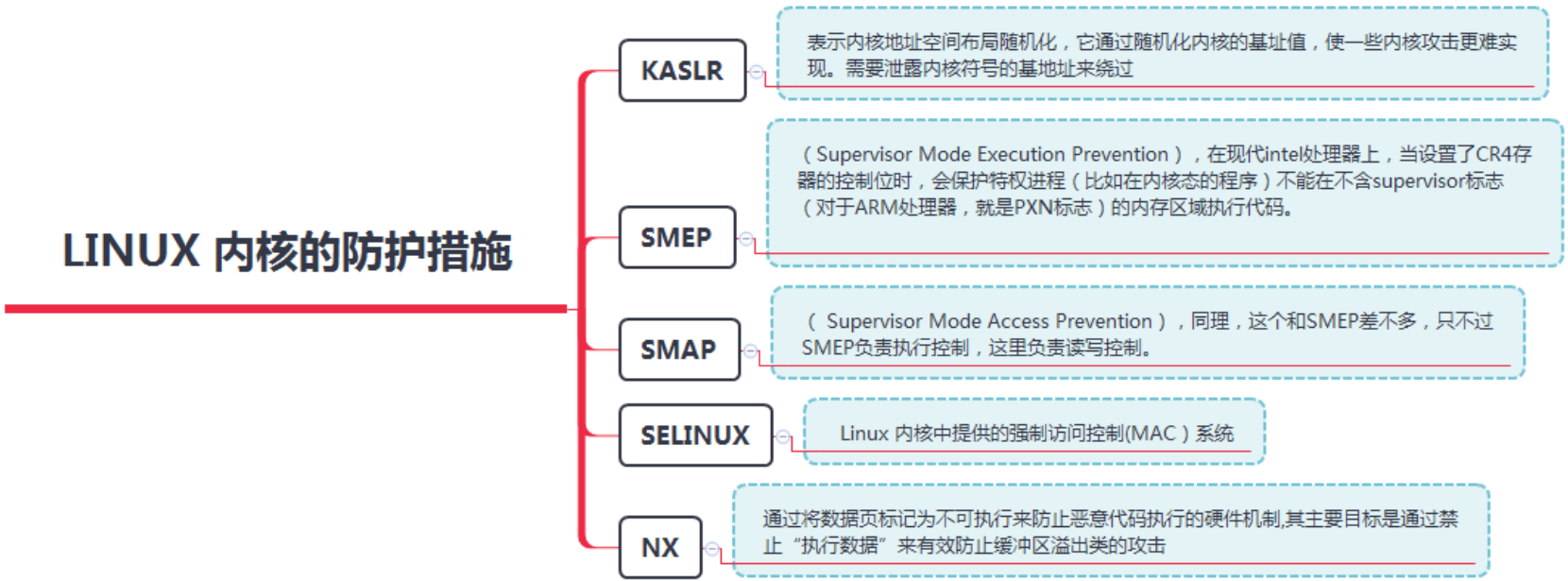


Linux内核的攻击面



Linux内核的漏洞类型

Linux内核的防护措施



Linux内核的防护措施

```
de4dcr0w@ubuntu:~/study/tools/checksec.sh-1.6$ uname -a
Linux ubuntu 4.13.0-16-generic #19-Ubuntu SMP Wed Oct 11 18:35:14 UTC 201
7 x86_64 x86_64 x86_64 GNU/Linux
de4dcr0w@ubuntu:~/study/tools/checksec.sh-1.6$ ./checksec --kernel
* Kernel protection information:

  Description - List the status of kernel protection mechanisms. Rather t
han
  inspect kernel mechanisms that may aid in the prevention of exploitatio
n of
  userspace processes, this option lists the status of kernel configurati
on
  options that harden the kernel itself against attack.

  Kernel config: /boot/config-4.13.0-16-generic

  Warning: The config on disk may not represent running kernel config!

  Vanilla Kernel ASLR:                    Full
  GCC stack protector support:            Enabled
  Strict user copy checks:                Disabled
  Enforce read-only kernel data:          Disabled
  Restrict /dev/mem access:               Enabled
  Restrict /dev/kmem access:              Enabled

* Selinux:                                Disabled

  SELinux infomation available here:
    http://selinuxproject.org/

* grsecurity / PaX:                       No GRKERNSEC

  The grsecurity / PaX patchset is available here:
    http://grsecurity.net/
```

版本：Ubuntu 17.10 64位
为当前最新版本，开启的防护如左图所示

内核版本为：
linux 4.13.0-16-generic

三、漏洞分析

- 漏洞背景
- 漏洞形成原因分析
- 调试过程
- 补丁分析
- 影响版本
- 提权演示

漏洞分析环境

（1） **gdb + qemu 调试内核**

只需要编译内核获得vmlinux 和 bzImage即可。打开一个窗口，进入内核目录，调试模式启动qemu：

```
qemu-system-x86_64 -kernel arch/x86_64/boot/bzImage -initrd  ../busybox-1.19.4/rootfs.img -append "root=/dev/ram rdinit=/sbin/init" -s  -S
```

（2）**双机调试内核**

编译内核时：勾选 Compile the kernel with debug info
Compile the kernel with frame pointers
4.0版本之前，关闭内核地址只读：Write protect kernel read-only data structures
4.0版本之后，修改arch/x86/Kconfig，关闭内核只读：CONFIG_STRICT_KERNEL_RWX
关闭KASLR，才能设置断点调试

漏洞背景

- **（1）漏洞背景**

2017年8月10日，oss-sec网站公布了CVE-2017-1000112，Andrey Konovalov 通过syzkaller发现了linux内核中关于UFO的代码中存在内存写越界的漏洞。

时间线：

2017.08.03 - Bug reported to security () kernel org
2017.08.04 - Bug reported to linux-distros@
2017.08.10 - Patch submitted to netdev
2017.08.10 - Announcement on oss-security@

漏洞背景

- **（2）什么是UFO?**

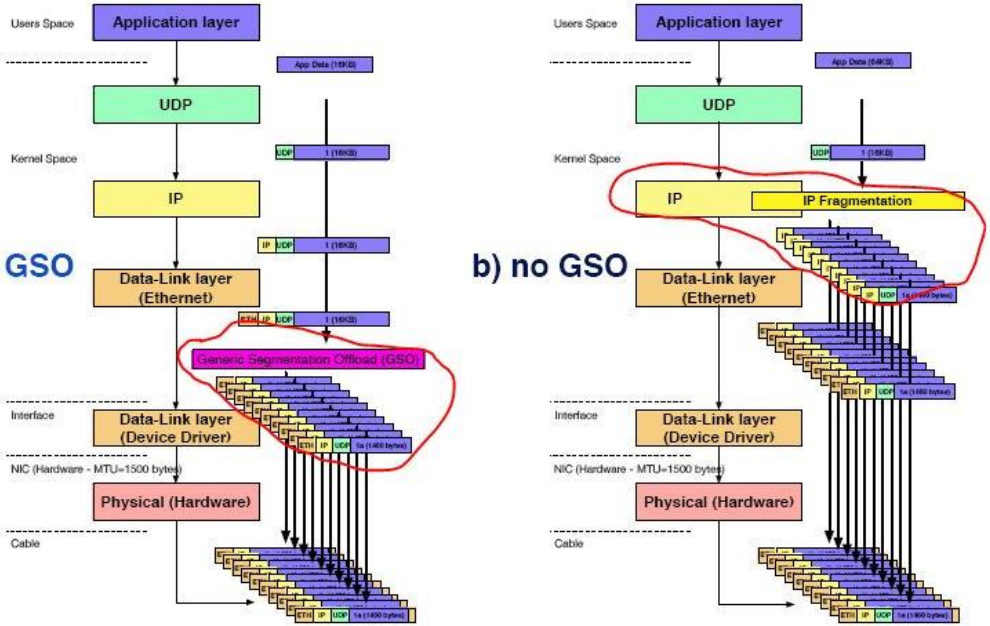
UFO(UDP Fragment Offload)是硬件网卡提供的一种特性，由内核和驱动配合完成相关功能。其目的是由网卡硬件来完成本来需要软件进行的分段(分片)操作用于提升效率和性能。减少Linux 内核传输层和网络层的计算工作，将这些计算工作offload（卸载）到物理网卡。UDP协议层本身不对大的数据报进行分片，而是交给IP层去做。因此，UFO就是将IP分片offload到网卡中进行。

如大家所知，在网络上传输的数据包不能大于mtu，当用户发送大于mtu的数据报文时，通常会在传输层就会按mtu大小进行分段，防止发送出去的报文大于mtu，为提升该操作的性能，新的网卡硬件基本都实现了UFO功能，可以使分段(或分片)操作在网卡硬件完成，此时用户态就可以发送长度大于mtu的包，而且不必在协议栈中进行分片。

这就意味着当开启UFO时，可以支持发送超过MTU大小的数据报。

漏洞背景

（3） GSO（Generic Segmentation Offload，可以理解成在数据推送网卡前进行分片，相当于对UFO的一种优化



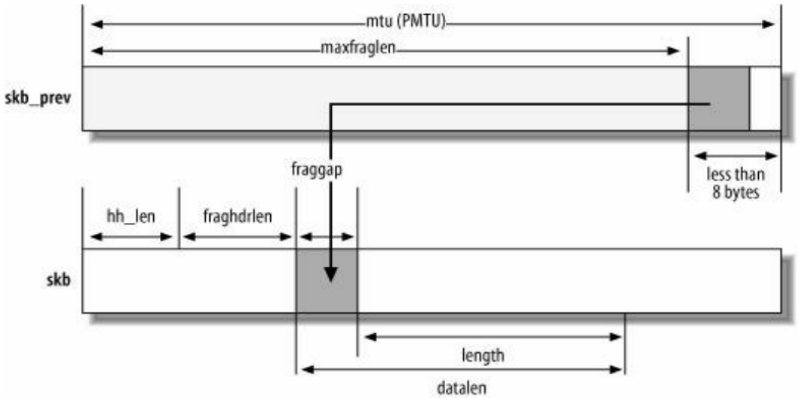
漏洞形成原因分析

漏洞发生点：/net/ipv4/ip_output.c 中的__ip_append_data函数

ip_append_data()主要是将接收到大数据包分成多个小于或等于MTU的SKB，为网络层要实现的IP分片作准备。

例如，假设待发送的数据包大小为4000B，先前输出队列非空，且最后一个SKB还没填满，剩余500B。这时传输层调用ip_append_data(),则首先会将剩余空间的SKB填满。进入循环，每次循环都分配一个SKB，通过getfrag将数据从传输层复制数据，并将其添加到输出队列的末尾，直至复制完所有待输出的数据。

Skb结构:



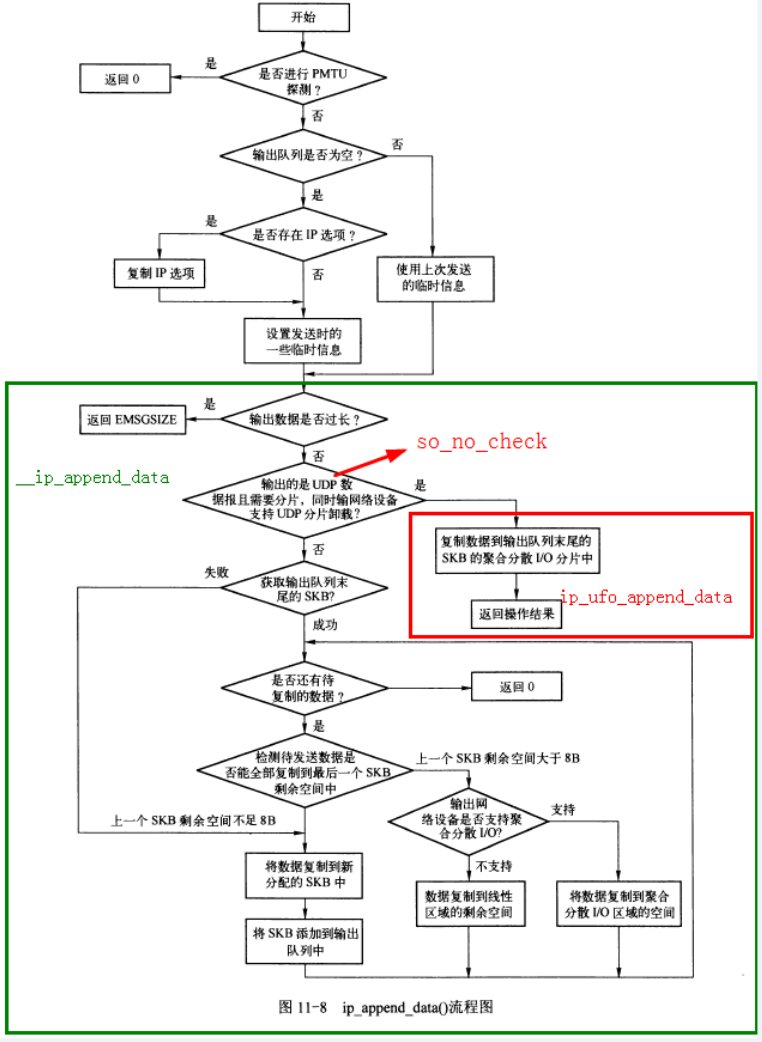
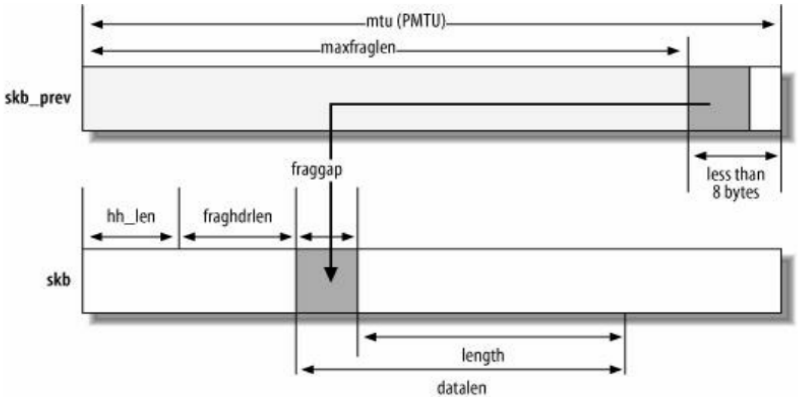


图 11-8　ip_append_data()流程图

漏洞形成原因分析

具体的过程为UFO填充的skb大于MTU，导致在non-UFO路径上copy = maxfraglen-skb->len变为负数，触发重新分配skb的操作，导致fraggap=skb_prev->len-maxfraglen会很大，超过MTU，之后在调用skb_copy_and_csum_bits()进行复制操作时造成写越界。



漏洞形成原因分析

Shellcode的触发：覆写skb结构里的destructor_arg->callback，指向shellcode，在之后释放skb操作中，会调用skb_release_data函数：

```
static void skb_release_data(struct sk_buff *skb)
{
    struct skb_shared_info *shinfo = skb_shinfo(skb);
    int i;

    if (skb->cloned &&
        atomic_sub_return(skb->nohdr ? (1 << SKB_DATAREF_SHIFT) + 1 : 1,
                          &shinfo->dataref))
        return;

    for (i = 0; i < shinfo->nr_frags; i++)
        __skb_frag_unref(&shinfo->frags[i]);

    /*
     * If skb buf is from userspace, we need to notify the caller
     * the lower device DMA has done;
     */
    if (shinfo->tx_flags & SKBTX_DEV_ZEROCOPY) {
        struct ubuf_info *uarg;

        uarg = shinfo->destructor_arg;
        if (uarg->callback)
            uarg->callback(uarg, true);
    }

    if (shinfo->frag_list)
        kfree_skb_list(shinfo->frag_list);

    skb_free_head(skb);
} « end skb_release_data »
```

调试过程

（1）下断点：

```
(gdb) b __ip_append_data
Breakpoint 1 at 0xffffffff8170bf30: file net/ipv4/ip_output.c, line 864.
(gdb) b udp_send_skb
Breakpoint 2 at 0xffffffff81733bb0: file net/ipv4/udp.c, line 800.
```

（2）non-UFO路径下copy < 0

```
(gdb) n
922             if (((length > mtu) || (skb && skb_is_gso(skb))) &&
(gdb) n
924                 (rt->dst.dev->features & NETIF_F_UFO) && !rt->dst.header_len &&
(gdb) n
923                 (sk->sk_protocol == IPPROTO_UDP) &&
(gdb) n
924                 (rt->dst.dev->features & NETIF_F_UFO) && !rt->dst.header_len &&
(gdb) n
925                 (sk->sk_type == SOCK_DGRAM) && !sk->sk_no_check_tx) {
(gdb) n
926                     err = ip_ufo_append_data(sk, queue, getfrag, from, length,
(gdb) p length
$2 = 3492
```

调试过程

（3）查看buffer中覆写的skb

```
0xffff8800af14b674:     0x4242424242424242      0x4242424242424242
0xffff8800af14b684:     0x4242424242424242      0x4242424242424242
0xffff8800af14b694:     0x4242424242424242      0x4242424242424242
0xffff8800af14b6a4:     0x4242424242424242      0x4242424242424242
0xffff8800af14b6b4:     0x4242424242424242      0x0000ff0042424242
0xffff8800af14b6c4:     0x0000000000000000      0x0000000000000000
0xffff8800af14b6d4:     0x0000000000000000      0x0000000000000000
0xffff8800af14b6e4:     0x006052b000000000      0x0000000000000000
0xffff8800af14b6f4:     0x0000000000000000      0x0000000000000000
```

（4）查看rop链的入口：

```
(gdb) x /gx 0x6052b0
0x6052b0:       0xffffffff8104510a
```

补丁分析

Diffstat

```
-rw-r--r-- net/ipv4/ip_output.c  8
-rw-r--r-- net/ipv4/udp.c        2
-rw-r--r-- net/ipv6/ip6_output.c 7
```

3 files changed, 10 insertions, 7 deletions

```
diff --git a/net/ipv4/ip_output.c b/net/ipv4/ip_output.c
index 50c74cd..e153c40 100644
--- a/net/ipv4/ip_output.c
+++ b/net/ipv4/ip_output.c
@@ -965,11 +965,12 @@ static int __ip_append_data(struct sock *sk,
                csummode = CHECKSUM_PARTIAL;

        cork->length += length;
-       if ((((length + (skb ? skb->len : fragheaderlen)) > mtu) ||
-            (skb && skb_is_gso(skb))) &&
+       if ((skb && skb_is_gso(skb)) ||
+           (((length + (skb ? skb->len : fragheaderlen)) > mtu) &&
+           (skb_queue_len(queue) <= 1) &&
            (sk->sk_protocol == IPPROTO_UDP) &&
            (rt->dst.dev->features & NETIF_F_UFO) && !dst_xfrm(&rt->dst) &&
-           (sk->sk_type == SOCK_DGRAM) && !sk->sk_no_check_tx) {
+           (sk->sk_type == SOCK_DGRAM) && !sk->sk_no_check_tx)) {
                err = ip_ufo_append_data(sk, queue, getfrag, from, length,
                                         hh_len, fragheaderlen, transhdrlen,
                                         maxfraglen, flags);
@@ -1288,6 +1289,7 @@ ssize_t   ip_append_page(struct sock *sk, struct flowi4 *fl4, struct page *page,
                return -EINVAL;

        if ((size + skb->len > mtu) &&
+           (skb_queue_len(&sk->sk_write_queue) == 1) &&
            (sk->sk_protocol == IPPROTO_UDP) &&
            (rt->dst.dev->features & NETIF_F_UFO)) {
                if (skb->ip_summed != CHECKSUM_PARTIAL)
```

打补丁之前，进不进ufo路径主要由sk_no_check_tx决定。

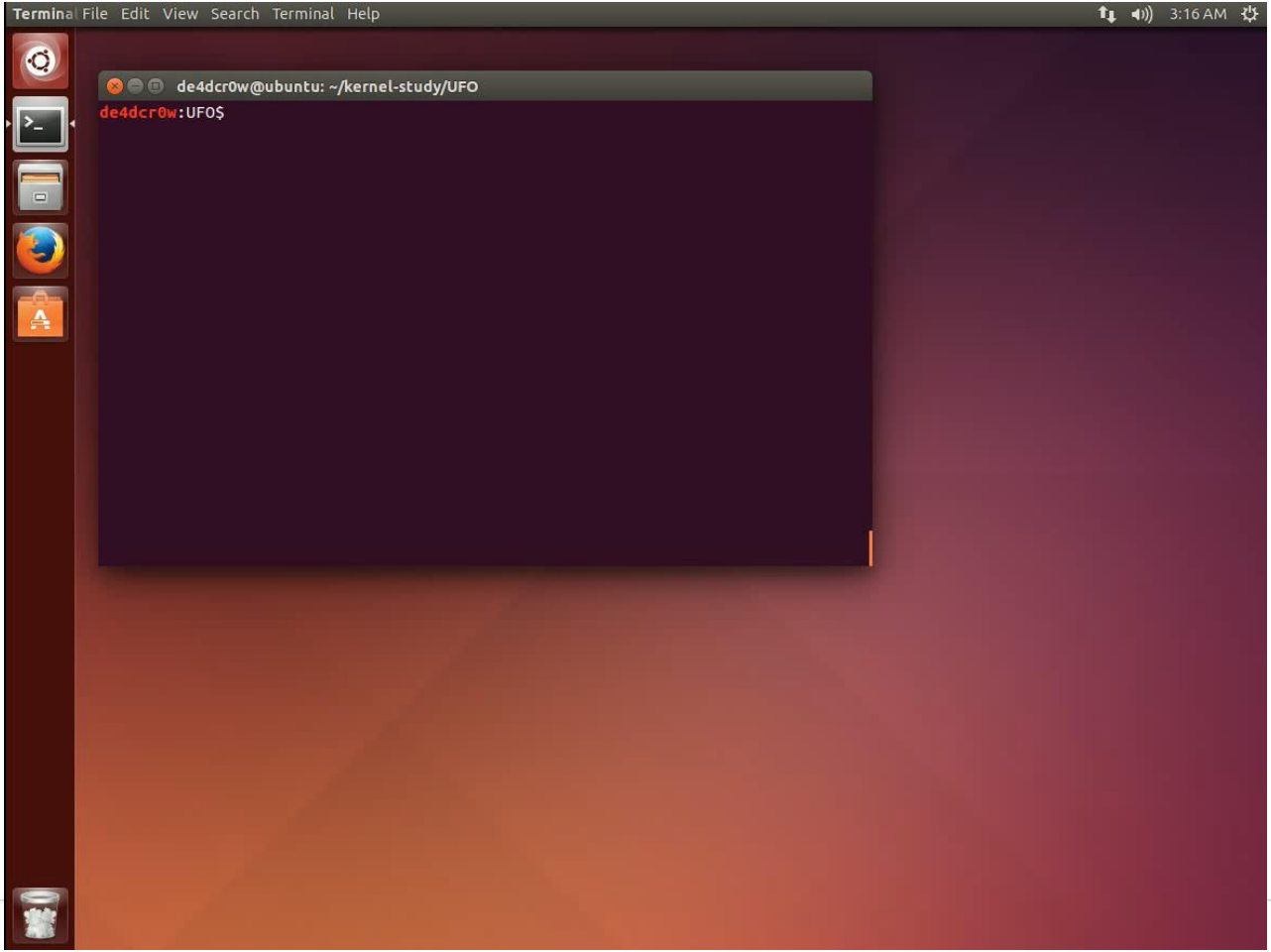
打补丁之后，即使设置了sk_no_check_tx，只要开启了gso（Generic Segmentation Offload，可以理解成在数据推送网卡前进行分片，相当于对UFO的一种优化），一样会进入ufo路径，这样漏洞就无法触发了。

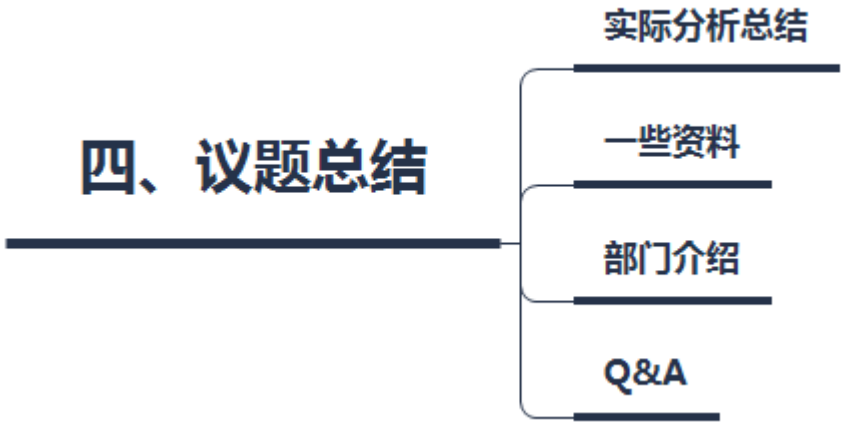
影响版本

Redhat Enterprise Linux 7
Linux kernel 4.12.3
Linux kernel 4.12.2
Linux kernel 4.11.9
Linux kernel 4.11.5
Linux kernel 4.11.4
Linux kernel 4.11.3
Linux kernel 4.11.2
Linux kernel 4.11.1
Linux kernel 4.11
Linux kernel 4.10.15
Linux kernel 4.10.13
Linux kernel 4.10.12
Linux kernel 4.10.10

Linux kernel 4.10.6
Linux kernel 4.10.4
Linux kernel 4.10
Linux kernel 4.9.13
Linux kernel 4.9.8
Linux kernel 4.9.4
Linux kernel 4.9.3
Linux kernel 4.7.4
Linux kernel 4.4.30
Linux kernel 4.4.29
Linux kernel 4.4.28
Linux kernel 4.4.27
Linux kernel 4.4.25
Linux kernel 4.4.24

...
...
...


Terminal File Edit View Search Terminal Help
3:16 AM
de4dcr0w@ubuntu: ~/kernel-study/UFO
de4dcr0w:UFO$


四、议题总结
实际分析总结
一些资料
部门介绍
Q&A

实际分析总结

（1）前期工作准备：
搭建环境，要找到对应的漏洞版本，漏洞组件，这一步要达到能复现漏洞
（2）进行分析工作:
查看源码，猜测漏洞可能发生的地方
进行调试工作，分析漏洞产生的具体细节
对比补丁，验证漏洞成因
（3）进阶分析：
编写成功率更高的poc
将poc转化成一个exp
绕过防护机制，完善exp
（4）整理分析报告

一些资料

一些有用的网站：

[1] http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
[2] https://www.exploit-db.com/ 各种exp
[3] https://www.kernel.org/pub/linux/kernel/ 内核版本下载
[4] https://github.com/SecWiki/linux-kernel-exploits 历年的linux-kernel-exp
[5] https://github.com/secfigo/Awesome-Fuzzing fuzz学习资源整合
[6] https://groups.google.com/forum/#!forum/linux.kernel linux-kernel google组
[7] https://bugzilla.redhat.com/buglist.cgi?quicksearch=kernel bug list
[8] http://ddebs.ubuntu.com/pool/main/l/linux/ 镜像文件下载
[9] http://elixir.free-electrons.com/linux/latest/source 在线查看内核源码
[10] https://github.com/torvalds/linux linux git 可以下载源码，对照补丁

一些资料

推荐书籍：

《A Guide to Kernel Exploitation Attacking the Core》
《Linux 内核源码剖析- TCP.IP 实现, 樊东东, 莫澜》
《Linux内核设计与实现》
《深入理解Linux内核》

一些工具：

Source Insight ：查看源码工具
Qemu ： 模拟仿真，搭建虚拟环境
Gdb : 调试工具
Fuzz工具：trinity、syzkaller

部门介绍

360CERT全称“360 Computer Emergency Readiness Team”我们致力于维护计算机网络空间安全，是360基于"协同联动，主动发现，快速响应"的指导原则，对全球重要网络安全事件进行快速预警、应急响应的安全协调团队。