上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CVE-2010-4342

学号: 522031910439　姓名: 梁俊轩

2025 年 6 月 13 日

1　漏洞概述

1.1　CVE 描述

在启用 Econet 模块的 Linux 内核（2.6.37-rc6 之前版本）中，net/econet/af_econet.c 的 aun_incoming() 函数存在空指针解引用漏洞。当处理特殊构造的 Acorn Universal Networking (AUN) 协议 UDP 数据包（类型码为 2）时，内核会因未校验 skb->dev 的空指针状态，直接访问其 ec_ptr 成员，导致内核 OOPS（空指针解引用）和系统拒绝服务。

1.2　影响的软件/组件及版本

在 2.6.37-rc6 之前的 Linux 内核。

1.3　分析调试环境

操作系统：Ubuntu 14.04
Linux 内核版本：2.6.34

2　漏洞分析

2.1　漏洞成因

套接字缓冲区 (skb) 来源于/usr/src/linux-2.6.34/net/econet/af_cont.c 的 aun_data_available 函数：

```
1 static void aun_data_available(struct sock *sk, int slen)
2 {
3     int err;
4     struct sk buff *skb;
5     unsigned char *data;
6     struct aunhdr *ah;
7     struct iphdr *ip;
8     size_t len;
9     while((skb=skb_recv_datagram(sk,0,1,&err))== NULL){
```

```
10          if(err == -EAGAIN){
11                  printk(KERN_ERR "AUN: no_data_available?!");
12                  return;
13          }
14          printk(KERN_DEBUG "AUN: recvfrom() error_%d\n",-err);
```

上述代码第九行的 skb_recv_datagram() 从套接字接收队列（sk->sk_receive_queue）中提取 skb，该函数位于/usr/src/linux-2.6.34/net/core/datagram.c，其通过位于/usr/src/linux-2.6.34/net/core/socket.c 的 sock_queue_rcv_skb 将 skb 加入队列，而该函数会显式将 skb->dev 设置为 NULL。

```
1  struct sk_buff *skb_recv_datagram(struct sock *sk, unsigned flags
       ,
2                                    int noblock, int *err)
3  {
4          .......
5          return __skb_recv_datagram(sk, flags | (noblock ?
               MSG_DONTWAIT : 0),
6                                     &peeked, err);
7  }
8
9  struct sk_buff *__skb_recv_datagram(struct sock *sk, unsigned
       flags,
10                                     int *peeked, int *err)
11 {
12         .....
13
14         do {
15                 /* Again only user level code calls this function
                       , so nothing
16                  * interrupt level will suddenly eat the
                        receive_queue.
17                  *
18                  * Look at current nfs client by the way...
19                  * However, this function was corrent in any case
                        . 8)
20                  */
21                 unsigned long cpu_flags;
22
23                 spin_lock_irqsave(&sk->sk_receive_queue.lock,
                       cpu_flags);
24                 skb = skb_peek(&sk->sk_receive_queue);
```

```
25                  if (skb) {
26                          *peeked = skb->peeked;
27                          if (flags & MSG_PEEK) {
28                                  skb->peeked = 1;
29                                  atomic_inc(&skb->users);
30                          } else
31                                  __skb_unlink(skb, &sk->
                                        sk_receive_queue);
32                  }
33                  spin_unlock_irqrestore(&sk->sk_receive_queue.lock
                        , cpu_flags);
34
35                  if (skb)
36                          return skb;
37
38                  /* User doesn't want to wait */
39                  error = -EAGAIN;
40                  if (!timeo)
41                          goto no_packet;
42
43          } while (!wait_for_packet(sk, err, &timeo));
44
45          return NULL;
46
47  no_packet:
48          *err = error;
49          return NULL;
50  }
```

```
1  int sock_queue_rcv_skb(struct sock *sk, struct sk_buff *skb)
2  {
3          .........
4          skb->dev = NULL;
5          .........
```

当 aun_data_available 函数接收到数据报文 (ah->code == 2) 时，会调用/usr/src/linux-2.6.34/net/cont/af_cont.c 中的 aun_incoming 函数处理套接字缓冲区 (skb)。aun_incoming 函数起始代码如下：

```
1  static void aun_incoming(struct sk_buff *skb, struct aunhdr *ah,
       size_t len)
2  {
```

```
3       struct iphdr *ip = ip_hdr(skb);
4       unsigned char stn = ntohl(ip->saddr) & 0xff;
5       struct sock *sk = NULL;
6       struct sk_buff *newskb;
7       struct ec_device *edev = skb->dev->ec_ptr;
```

而 skb->dev 始终为 NULL，导致上述代码中最后一行 struct ec_device *edev = skb->dev->ec_ptr; 在函数执行实际逻辑前立即触发错误，即使用了空指针。

2.2 漏洞复现（触发路径）

本人使用的 Linux 内核版本为 2.6.34。

复现步骤：

1. 启用 Econet 模块：modprobe econet_aun_udp
2. scapy.send(IP(dst=target)/UDP(dport=519)/Raw(load=b'02<AUN_payload>'))
3. 触发结果：内核日志出现"Unable to handle kernel NULL pointer dereference" OOPS

具体如图1所示。

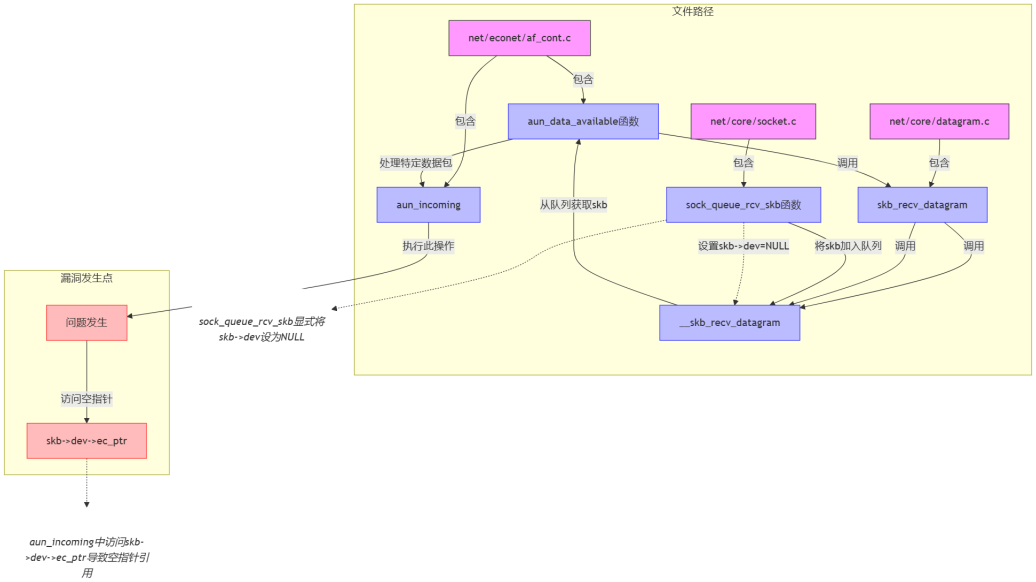

图 1 漏洞触发路径

2.3 漏洞可利用性分析

CVSS 向量：CVSS:2.0/AV:N/AC:M/Au:N/C:N/I:N/A:C

具体如表1所示。

指标	含义	值	解读
AV	攻击向量 (Attack Vector)	N (Network)	攻击者可通过网络访问目标系统
AC	攻击复杂度 (Attack Complexity)	M (Medium)	攻击需要特定条件，实现有一定难度
Au	所需权限 (Authentication)	N (None)	攻击者不需要身份验证
C	机密性影响 (Confidentiality Impact)	N (None)	攻击不会导致机密性损失
I	完整性影响 (Integrity Impact)	N (None)	攻击不会导致完整性损失
A	可用性影响 (Availability Impact)	C (Compromised)	攻击会导致可用性部分受损

表 1 CVSS 向量解读

2.4 攻击者利用步骤

远程攻击者通过 UDP 恶意构造 Acorn Universal Networking（AUN）数据包并发送，触发空指针解引用，引发内核错误（OOPS），导致目标系统出现异常，进而造成服务中断，实现拒绝服务攻击（具体详见附录的 oops）。

3 修复方案

3.1 已有的修复方案代码片段及分析

官方的修改方式在 bug 报告的信件中，官方补丁通过引入间接指针校验解决空指针问题，直接通过 skb_dst() 函数获取 skb 的 dev。若 dev 不为空，才会去获取 dev 的 ec_ptr。具体 commit 的提交记录已完全丢失：

https://marc.info/?l=linux-netdevm=129186011218615w=2

```
1  static void aun_incoming(struct sk_buff *skb, struct aunhdr *ah,
       size_t len)
2  {
3      struct iphdr *ip = ip_hdr(skb);
4      unsigned char stn = ntohl(ip->saddr) & 0xff;
5      struct dst_entry *dst = skb_dst(skb);
6      struct ec_device *edev = NULL;
7      struct sock *sk = NULL;
8      struct sk_buff *newskb;
9
10     if (dst)
11         edev = dst->dev->ec_ptr;
```

3.2 可能的临时缓解方案

通过 modprobe -r econet_aun_udp 卸载模块，或在启动参数中添加 blacklist econet_aun_udp 禁止模块加载。

3.3 安全开发建议

1. **指针安全**：内核中访问指针前必须验证非空（尤其是 skb->dev 等易变成员）
2. **协议栈防护**：网络协议处理函数应假设外部数据不可信
3. **防御性编程**：使用类似宏增强校验

A 附录

A.1 oops

```
BUG: unable to handle kernel NULL pointer dereference at
    0000000000000240
IP: [<ffffffff813a303e>] aun_data_available+0xb3/0x28d
PGD e818067 PUD e819067 PMD 0
Oops: 0000 [#1] SMP
last sysfs file: /sys/devices/virtual/net/lo/operstate
CPU 0
Modules linked in:

Pid: 0, comm: swapper Not tainted 2.6.37-rc3 #39 /Bochs
RIP: 0010:[<ffffffff813a303e>]  [<ffffffff813a303e>]
    aun_data_available+0xb3/0x28d
RSP: 0018:ffff88000fc03b00  EFLAGS: 00010246
RAX: 0000000000000000 RBX: ffff88000fc03b3c RCX: ffff88000e808800
RDX: 0000000000000002 RSI: 0000000000000286 RDI: ffff88000e8060d4
RBP: ffff88000fc03b70 R08: 0000000000000003 R09: 0000000000000002
R10: ffff88000e69ac00 R11: 00000000ffffffff R12: ffff88000e80886a
R13: ffff88000e439500 R14: ffff88000e8060d4 R15: ffff88000e80884e
FS:  0000000000000000(0000) GS:ffff88000fc00000(0000) knlGS
    :0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 0000000000000240 CR3: 000000000e813000 CR4: 00000000000006f0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
Process swapper (pid: 0, threadinfo ffffffff81600000, task
    ffffffff8162b020)
```

```
Stack:
 ffff88000fc03ba0 ffffffff8131d614 ffff880000000010
     ffff88000e806000
 0200000100000000 0202000a00000000 ffffc90000086d48
     0000000000000246
 ffff88000fc03b70 ffff88000e806000 ffff88000e439500
     0000000000000000
Call Trace:
 <IRQ>
 [<ffffffff8131d614>] ? rt_intern_hash+0x5de/0x606
 [<ffffffff812f7839>] sock_queue_rcv_skb+0x168/0x187
 [<ffffffff81325641>] ip_queue_rcv_skb+0x45/0x4c
 [<ffffffff81031aa0>] ? try_to_wake_up+0x265/0x277
 [<ffffffff8133ee23>] __udp_queue_rcv_skb+0x50/0xb9
 [<ffffffff813403b0>] udp_queue_rcv_skb+0x1ad/0x26e
 [<ffffffff81340b67>] __udp4_lib_rcv+0x30a/0x50d
 [<ffffffff81340d7f>] udp_rcv+0x15/0x17
 [<ffffffff813206b1>] ip_local_deliver+0x12d/0x1d0
 [<ffffffff81320546>] ip_rcv+0x4f2/0x530
 [<ffffffff81300fb3>] __netif_receive_skb+0x34d/0x377
 [<ffffffff81301c1d>] netif_receive_skb+0x67/0x6e
 [<ffffffff812fb763>] ? __netdev_alloc_skb+0x1d/0x3a
 [<ffffffff8129a189>] cp_rx_poll+0x2e8/0x3ab
 [<ffffffff81007414>] ? nommu_map_page+0x0/0xa0
 [<ffffffff81302308>] net_rx_action+0xa7/0x215
 [<ffffffff8103c0f9>] __do_softirq+0xcd/0x18c
 [<ffffffff81002e4c>] call_softirq+0x1c/0x28
 [<ffffffff810042c3>] do_softirq+0x33/0x68
 [<ffffffff8103bc61>] irq_exit+0x36/0x38
 [<ffffffff810039a8>] do_IRQ+0xa3/0xba
 [<ffffffff813be8d3>] ret_from_intr+0x0/0xa
 <EOI>
 [<ffffffff810089a1>] ? default_idle+0x62/0x7a
 [<ffffffff813c1882>] ? atomic_notifier_call_chain+0x13/0x15
 [<ffffffff81001321>] cpu_idle+0x54/0xbe
 [<ffffffff813a3d49>] rest_init+0x6d/0x6f
 [<ffffffff8169cc85>] start_kernel+0x332/0x33d
 [<ffffffff8169c2a8>] x86_64_start_reservations+0xb8/0xbc
 [<ffffffff8169c39e>] x86_64_start_kernel+0xf2/0xf9
Code: 00 80 fa 04 0f 84 bb 01 00 00 80 fa 02 0f 85 c3 01 00 00 45
```

```
      8b bd a0 00 00 00 \
4e 8d 3c 39 41 8b 47 0c 0f c8 88 45 b7 49 8b 45 20 <4c> 8b b0 40
    02 00 00 4d 85 f6 0f \
84 4f 01 00 00 41 8a 46 01 48  RIP  [<ffffffff813a303e>] \
aun_data_available+0xb3/0x28d  RSP <ffff88000fc03b00>
CR2: 0000000000000240
---[ end trace 8e7c904f0da8a9a0 ]---
Kernel panic - not syncing: Fatal exception in interrupt
Pid: 0, comm: swapper Tainted: G      D      2.6.37-rc3 #39
Call Trace:
 <IRQ>  [<ffffffff813bc1b4>] panic+0x8c/0x18d
 [<ffffffff810374ff>] ? kmsg_dump+0x115/0x12f
 [<ffffffff813bf5a2>] oops_end+0x81/0x8e
 [<ffffffff81020b01>] no_context+0x1f7/0x206
 [<ffffffff81067af6>] ? handle_IRQ_event+0x52/0x117
 [<ffffffff81020c92>] __bad_area_nosemaphore+0x182/0x1a5
 [<ffffffff81069aa5>] ? handle_fasteoi_irq+0xd5/0xe0
 [<ffffffff81020cc3>] bad_area_nosemaphore+0xe/0x10
 [<ffffffff813c160a>] do_page_fault+0x1e3/0x3db
 [<ffffffff810039a8>] ? do_IRQ+0xa3/0xba
 [<ffffffff813be8d3>] ? ret_from_intr+0x0/0xa
 [<ffffffff8102c35c>] ? enqueue_task_fair+0x156/0x162
 [<ffffffff812fcb99>] ? __skb_recv_datagram+0x116/0x258
 [<ffffffff813beadf>] page_fault+0x1f/0x30
 [<ffffffff813a303e>] ? aun_data_available+0xb3/0x28d
 [<ffffffff8131d614>] ? rt_intern_hash+0x5de/0x606
 [<ffffffff812f7839>] sock_queue_rcv_skb+0x168/0x187
 [<ffffffff81325641>] ip_queue_rcv_skb+0x45/0x4c
 [<ffffffff81031aa0>] ? try_to_wake_up+0x265/0x277
 [<ffffffff8133ee23>] __udp_queue_rcv_skb+0x50/0xb9
 [<ffffffff813403b0>] udp_queue_rcv_skb+0x1ad/0x26e
 [<ffffffff81340b67>] __udp4_lib_rcv+0x30a/0x50d
 [<ffffffff81340d7f>] udp_rcv+0x15/0x17
 [<ffffffff813206b1>] ip_local_deliver+0x12d/0x1d0
 [<ffffffff81320546>] ip_rcv+0x4f2/0x530
 [<ffffffff81300fb3>] __netif_receive_skb+0x34d/0x377
 [<ffffffff81301c1d>] netif_receive_skb+0x67/0x6e
 [<ffffffff812fb763>] ? __netdev_alloc_skb+0x1d/0x3a
 [<ffffffff8129a189>] cp_rx_poll+0x2e8/0x3ab
 [<ffffffff81007414>] ? nommu_map_page+0x0/0xa0
```

```
98  [<ffffffff81302308>] net_rx_action+0xa7/0x215
99  [<ffffffff8103c0f9>] __do_softirq+0xcd/0x18c
100 [<ffffffff81002e4c>] call_softirq+0x1c/0x28
101 [<ffffffff810042c3>] do_softirq+0x33/0x68
102 [<ffffffff8103bc61>] irq_exit+0x36/0x38
103 [<ffffffff810039a8>] do_IRQ+0xa3/0xba
104 [<ffffffff813be8d3>] ret_from_intr+0x0/0xa
105 <EOI>  [<ffffffff810089a1>] ? default_idle+0x62/0x7a
106 [<ffffffff813c1882>] ? atomic_notifier_call_chain+0x13/0x15
107 [<ffffffff81001321>] cpu_idle+0x54/0xbe
108 [<ffffffff813a3d49>] rest_init+0x6d/0x6f
109 [<ffffffff8169cc85>] start_kernel+0x332/0x33d
110 [<ffffffff8169c2a8>] x86_64_start_reservations+0xb8/0xbc
111 [<ffffffff8169c39e>] x86_64_start_kernel+0xf2/0xf9
112 --
```