HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Sigurnosni nedostaci u ispuni Ethernet paketa

CCERT-PUBDOC-2003-04-12

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Otkriveno je da u implementaciji prijenosnog sloja (*engl. link layer*) TCP/IP stoga protokola postoje nedostaci koji napadaču omogućavaju pregled dijelova ranije poslanih paketa ili dijelova memorije jezgre operativnog sustava (*engl. kernel memory*). Upravljački programi (*engl. drivers*) mrežnih kartica vrlo često su loše napisani, te ne poštuju sve odgovarajuće RFC zahtjeve, što rezultira brojnim nedostacima, odnosno mogućnošću nekontroliranog curenja informacija. Između ostalog, jedan od tih nedostataka proizlazi iz neadekvatne ispune (*engl. padding*) Ethernet paketa. Naime, često se događa da se okteti potrebni za ispunu Ethernet paketa kopiraju direktno iz prethodno korištenog spremnika bez njegova čišćenja. Ovaj nedostatak vrlo je jednostavno iskoristiti, dok posljedice tog iskorištavanja mogu vrlo ozbiljne. Kroz dokument će biti opisano nekoliko inačica ovog nedostatka.
Pokazuje se da su mnogi operativni sustavi, uključujući Linux, NetBSD i Windows sustave, osjetljivi na ovaj nedostatak, a vjerojatno je da pogreške u implementaciji prijenosnog sloja postoje i u drugim operativnim sustavima.

2. Općenito o Ethernetu

Ethernet je razvio Xerox 1973. godine. Nakon nekoliko godina Ethernet je standardiziran i komercijaliziran u prvom 10Mbps Ethernet standardu. Nakon toga standard je doživio još neke revizije i promjene dok 1985. godine IEEE nije objavio "*Carrier Sense Multiple Access with Collision Detection Access Method and Physical Layer Specifications*", poznatiji pod imenom 802.3 CSMA/CD standard. Do današnjeg dana standardizirane su razne Ethernet tehnologije kao što su 100Mbps Fast Ethernet, Gigabit Ethernet i druge.
Ethernet je protokol prijenosnog sloja koji se koristi za prijenos podataka iz viših slojeva između računala. Prijenos podataka formira se prospajanjem paketa. Paketi moraju zadovoljavati jedan od dva formata definiranih IEEE 802.3 standardom. Unatoč nekim tehničkim razlikama između tih formata, oni posjeduju nekoliko polja koja su identična za oba formata. U tim poljima sadržane su adrese izvorišta i odredišta okvira, podaci koji osiguravaju integritet paketa, same podatke, te neke druge informacije.
Polje namijenjeno za same podatke je spremnik promjenjive duljine koji sadrži enkapsulirane podatke iz viših slojeva. Ograničenja unutar Ethernet standarda određuju veličinu tog polja između minimalno 46 okteta i maksimalno 1500 okteta. Ukoliko podaci iz viših slojeva imaju manju duljinu od minimalne, polje mora biti ispunjeno do ispravne veličine.
Spomenuti formati okvira razlikuju se prvenstveno u mehanizmu koji koriste za opisivanje enkapsuliranih podataka. Prvi format predstavlja enkapsulaciju tipa, dok drugi predstavlja enkapsulaciju duljine. Enkapsulacija tipa opisuje sadržaj podatkovnog polja prema tipu paketa protokola višeg sloja kojeg enkapsulira. Enkapsulacija duljine opisuje duljinu podatkovnog polja, i koristi opcionalno LLC zaglavlje za opis vrste protokola višeg sloja.

2.1. Sučelje prema TCP/IP stogu

Enkapsulacija IP paketa za prijenos preko Ethernet mreža definirana je u dva RFC dokumenta; RFC 894 ("*A Standard for the Transmission of IP Datagrams over Ethernet Networks*") i RFC 1042 ("*A Standard for the Transmission of IP Datagrams over IEEE 802 Networks*"). Oba dokumenta daju naputke kako rukovati IP paketima manjim od minimalne duljine podatkovnog polja definirane Ethernet standardima. Prema njima je za osiguranje minimalne duljine Ethernet okvira potrebno podatkovno polje ispuniti oktetima koji sadrže nul vrijednosti. Ova ispuna nije dio IP paketa i ne bi trebala biti uključena u polju za ukupnu duljinu definiranu u IP zaglavlju, već je to jednostavno dio prijenosnog sloja.
Nedorečenosti u tim dokumentima ostavljaju nejasnim tko je odgovoran za ispunjavanje Ethernet okvira, tako da je to ostalo ovisno o implementaciji. Neki sustavi implementiraju funkciju ispune u sklopovlju mrežnog sučelja, drugi pak kroz odgovarajuće upravljačke programe, a neki čak i u drugom sloju (IP sloj).
Pokazuje se da unutar raznih implementacija ispune Ethernet okvira postoje nepravilnosti. Često se za ispunu koriste podaci iz spremnika za manipulaciju podacima bez prethodnog postavljanja na nul

vrijednost, kako je definirano u RFC dokumentima. Na taj način ispunu definira sadržaj odgovarajućeg spremnika.
U nastavku dokumenta opisani su neki primjeri ovakve vrste nedostataka, te rješenja i mogućnosti ispravljana ovakvih nedostataka.

3. Sigurnosni nedostatak u ispuni Ethernet okvira

Sigurnosni nedostatak u ispuni Ethernet okvira pojavljuje se zbog programerske pogreške u implementaciji. Kroz nekoliko primjera programskog kôda biti će analizirane posljedice svake inačice ove pogreške, te uhvaćeni okviri koji potječu od ranjivih implementacija.

3.1. Pojednostavljeni prikaz

Sljedeći primjer ilustrira osnovnu programersku pogrešku koja može uzrokovati curenje informacija.

```
void xmit_frame(char *frame_buf, int frame_len)
{
int length;

if (frame_len < MIN_FRAME_SZ)
length = MIN_FRAME_SZ;
else
length = frame_len;

copy_to_tx_buf(frame_buf, length);

return;
}
```

Funkcija `xmit_frame()` kao argumente uzima dva parametra, prvi je spremnik koji sadrži okvir koji valja poslati, dok drugi parametar definira duljinu okvira unutar spremnika. Unutar kôda sadržana je i provjera da li okvir zadovoljava minimalnu duljinu definiranu Ethernet standardom. Ukoliko je duljina manja od minimalne, tada se varijabla `length` inicijalizira na minimalnu vrijednost, dok je u drugim slučajevima toj varijabli pridružena stvarna veličina okvira `frame_len`. Vrijednost varijable `length` koristi se za kopiranje spremnika okvira u spremnik sa slanje na mrežnom uređaju. U tom trenutku okvir duljine manje od minimalno dozvoljene biti će ispunjen od `frame_len` do `length` sa prethodnim sadržajem spremnika. Ta ispuna je ustvari mjesto curenja informacija.
Da bi se ovakav nedostatak ispravio, dovoljno je oktete za ispunu postaviti na nul-vrijednost. To je moguće učiniti na sljedeći način:

```
...
if (frame_len < MIN_FRAME_SZ) {
length = MIN_FRAME_SZ;
memset(frame_buf + frame_len, 0, length - frame_len);
} else
...
```

Na ovom primjeru vrlo je lako uočiti programerski pogrešku, no iznenađuje činjenica da mnogi upravljački programi u više ili manje sličnom obliku sadrže sličnu pogrešku u svom kôdu. Pogreška je u načelu ista u svim idućim primjerima, jedino su njene posljedice drukčije, obzirom na lokaciju spremnika u kojem se okvir nalazi prije slanja. Potencijalno postoje tri lokacije spremnika odakle okteti za ispunu mogu biti preuzeti:

- dinamički spremnik jezgre operativnog sustava,
- statički spremnik upravljačkog programa za upravljanje mrežnim sučeljem,
- sklopovski spremnik za slanje mrežnog sučelja.

3.2. Dinamički spremnik jezgre operativnog sustava

Ethernet okviri obično se oblikuju oko enkapsuliranog paketa protokola više razine. Ti paketi se slažu unutar TCP/IP stoga jezgre operativnog sustava i njima se manipulira kroz spremnike alocirane za tu namjenu. Sljedeći primjer se temelji na Linux 2.4.18 jezgri koja koristi jedan spremnik po paketu. Taj

spremnik se dinamički alocira u memoriji namijenjenoj jezgri operativnog sustava, a njegov sadržaj, ukoliko nije ispravno inicijaliziran, može sadržati razne podatke iz memorije jezgre sustava.

3.2.1. Programski kôd upravljačkog programa

```
static int atp_send_packet(struct sk_buff *skb, struct net_device
*dev)
{
struct net_local *lp = (struct net_local *)dev->priv;
long ioaddr = dev->base_addr;
int length;
long flags;
length = ETH_ZLEN < skb->len ? skb->len : ETH_ZLEN;
netif_stop_queue(dev);
/* Disable interrupts by writing 0x00 to the Interrupt Mask
Register.
This sequence must not be interrupted by an incoming packet. */
spin_lock_irqsave(&lp->lock, flags);
write_reg(ioaddr, IMR, 0);
write_reg_high(ioaddr, IMR, 0);
spin_unlock_irqrestore(&lp->lock, flags);
write_packet(ioaddr, length, skb->data, dev->if_port);
lp->pac_cnt_in_tx_buf++;
if (lp->tx_unit_busy == 0) {
trigger_send(ioaddr, length);
lp->saved_tx_size = 0; /* Redundant */
lp->re_tx = 0;
lp->tx_unit_busy = 1;
} else
lp->saved_tx_size = length;
/* Re-enable the LPT interrupts. */
write_reg(ioaddr, IMR, ISR_RxOK | ISR_TxErr | ISR_TxOK);
write_reg_high(ioaddr, IMR, ISRh_RxErr);
dev->trans_start = jiffies;
dev_kfree_skb (skb);
return 0;
}
```

Struktura `skb` sadrži pokazivač na spremnik za oblikovanje okvira `skb->data` i veličinu okvira unutar spremnika `skb->len`. Programski kôd inicira varijablu `length` na veću od `ETH_ZLEN` ili `skb->len` vrijednosti. Ukoliko je duljina okvira manja od minimalne definirane Ethernet standardom (`ETH_ZLEN`), biti će postavljena na tu minimalnu vrijednost. Varijabla `length` se nakon toga koristi za kopiranje okvira iz `skb->data` spremnika na mrežni uređaj.

Sadržaj ispune ovisi o radu sustava. Zbog velikog adresnog prostora memorije jezgre operativnog sustava i malog broja okteta za ispunu, nije vjerojatno da će doći do curenja povjerljivih informacija. Unatoč tome, mjesto odakle potječu okteti za ispunu potencijalno može otkriti ključne informacije.

3.2.2. Hvatanje i pregled okvira

Sljedeći ispis uhvaćenih paketa ilustrira kako se sadržaj okteta za ispunu mijenja kada se za konstrukciju `ICMP echo` odgovora koriste različiti dijelovi memorije jezgre operativnog sustava.

```
14:18:48.146095 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 83f9 5206 2200 00
14:18:48.146393 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id 3747, len 29)
4500 001d 0ea3 0000 ff01 9668 0a32 0135
```

```
0a32 013c 0000 8bf9 5206 2200 0059 0100
6c02 0000 c006 0000 0600 0000 0010
14:18:49.146095 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 82f9 5206 2300 00
14:18:49.146387 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id 3749, len 29)
4500 001d 0ea5 0000 ff01 9666 0a32 0135
0a32 013c 0000 8af9 5206 2300 0000 4003
2300 4003 f101 1200 ed01 feff 0000
14:18:50.146093 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 81f9 5206 2400 00
14:18:50.146396 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id
3751, len 29)
4500 001d 0ea7 0000 ff01 9664 0a32 0135
0a32 013c 0000 89f9 5206 2400 0000 8002
1700 e002 1700 4003 9101 c001 0600
```

3.3. Statički spremnik upravljačkog programa sa upravljanjem mrežnim sučeljem

Neki upravljački programi prije slanja mrežnom sučelju kopiraju okvire u interne spremnike. Sljedeći primjer je kôd za Linux 2.4 jezgru koristi spremnik za poravnanje da bi osigurao da adresa okvira bude prihvatljiva mrežnom sklopovlju. RealTek 8139 chipset zahtijeva da adresa spremnika bude poravnata na 32-bitne riječi. Zbog tog razloga se spremnik za poravnanje ne čisti između uporaba, a prethodni sadržaj spremnika će biti iskorišten za ispunu.

3.3.1. Programski kôd upravljačkog programa

```
static int
rtl8129_start_xmit(struct sk_buff *skb, struct net_device *dev)
{
struct rtl8129_private *tp = (struct rtl8129_private *)dev->priv;
long ioaddr = dev->base_addr;
int entry;
if (netif_pause_tx_queue(dev) != 0) {
/* This watchdog code is redundant with the media monitor timer.*/
if (jiffies - dev->trans_start > TX_TIMEOUT)
rtl8129_tx_timeout(dev);
return 1;
}
/* Calculate the next Tx descriptor entry. */
entry = tp->cur_tx % NUM_TX_DESC;
tp->tx_skbuff[entry] = skb;
if ((long)skb->data & 3) { /* Must use alignment buffer. */
memcpy(tp->tx_buf[entry], skb->data, skb->len); outl(virt_to_bus(tp-
>tx_buf[entry]), ioaddr + TxAddr0 +
entry*4);
} else
outl(virt_to_bus(skb->data), ioaddr + TxAddr0 + entry*4);
/* Note: the chip doesn't have auto-pad! */
outl(tp->tx_flag | (skb->len >= ETH_ZLEN ? skb->len : ETH_ZLEN),
ioaddr + TxStatus0 + entry*4);
/* There is a race condition here -- we might read dirty_tx, take an
```

```
interrupt that clears the Tx queue, and only then set tx_full.
So we do this in two phases. */
if (++tp->cur_tx - tp->dirty_tx >= NUM_TX_DESC) {
set_bit(0, &tp->tx_full);
if (tp->cur_tx - (volatile unsigned int)tp->dirty_tx <
NUM_TX_DESC) {
clear_bit(0, &tp->tx_full);
netif_unpause_tx_queue(dev);
} else
netif_stop_tx_queue(dev);
} else
netif_unpause_tx_queue(dev);
dev->trans_start = jiffies;
if (tp->msg_level & NETIF_MSG_TX_QUEUED)
printk(KERN_DEBUG"%s: Queued Tx packet at %p size %d to slot
%d.\n",
dev->name, skb->data, (int)skb->len, entry);
return 0;
}
```

Ovaj kôd je sličan prethodnom primjeru, osim što se u ovom slučaju još provodi provjera da li je spremnik sa okvirom pravilno poravnat. Ukoliko adresa spremnika nije poravnata na 32-bitnu vrijednost, tada je okvir biti kopiran u spremnik koji alocira upravljački program i koji je poravnat kako se zahtijeva. Adresa poravnatog spremnika, koji može biti spremnik upravljačkog programa ili originalni spremnik prosljeđuje se mrežnoj kartici. Mrežna kartica dobiva adresu spremnika sa okvirom, isto kao i duljinu okvira. Veličinu okvira koji se šalje određuje veća od vrijednosti `ETH_ZLEN` ili `skb->len`.

3.3.2. Hvatanje i pregled okvira

Sljedeći primjer temelji se na Linux 2.4.x jezgri, te Xircom PCMCIO mrežnoj kartici koja koristi `xirc2ps_cs.c` upravljački program.

Sljedeći tcpdump ispis na zoran način prikazuje problem curenja informacija. Informacije se temelje na HTTP baziranom prometu.

```
14:49:09.306008 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
id 577, len 28)
0x0000 4500 001c 0241 4000 8001 bb29 c0a8 de0b E....A@....).....
0x0010 c0a8 de19 0000 48f9 b406 0300 0400 0000 ......H.........
0x0020 0004 0650 5542 4c49 4303 4e4c 5307      ...PUBLIC.NLS.
14:50:46.313706 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 854, len 28)
0x0000 4500 001c 0356 4000 8001 ba14 c0a8 de0b E....V@.........
0x0010 c0a8 de19 0000 e7f8 b406 6400 4b43 4c41 ..........d.KCLA
0x0020 4e47 2e44 4c4c 3c00 0090 2787 ce24      NG.DLL<...'..$
14:51:02.315077 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 870, len 28)
0x0000 4500 001c 0366 4000 8001 ba04 c0a8 de0b E....f@.........
0x0010 c0a8 de19 0000 d7f8 b406 7400 7468 656d ..........t.them
0x0020 652e 646c 6c3c 0000 9027 87ce 2400      e.dll<...'..$.
11
14:51:11.315842 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 895, len 28)
0x0000 4500 001c 037f 4000 8001 b9eb c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 cef8 b406 7d00 743a 204d ..........}.t:.M
0x0020 6f7a 696c 6c61 2f35 2e30 2028 5769      ozilla/5.0.(Wi
```

```
14:51:50.318904 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1435, len 28)
0x0000 4500 001c 059b 4000 8001 b7cf c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 a7f8 b406 a400 6b65 6570 ............keep
0x0020 2d61 6c69 7665 0d0a 436f 6f6b 6965      -alive..Cookie
14:51:51.319076 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1436, len 28)
0x0000 4500 001c 059c 4000 8001 b7ce c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 a6f8 b406 a500 434f 500d ............COP.
0x0020 0a52 6566 6572 6572 3a20 6874 7470      .Referer:.http
14:52:03.320000 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1449, len 28)
0x0000 4500 001c 05a9 4000 8001 b7c1 c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 9af8 b406 b100 2057 696e ............Win
0x0020 646f 7773 204e 5420 352e 303b 2065      dows.NT.5.0;.e
14:52:04.320125 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1450, len 28)
0x0000 4500 001c 05aa 4000 8001 b7c0 c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 99f8 b406 b200 2f78 6d6c ............/xml
0x0020 2c61 7070 6c69 6361 7469 6f6e 2f78      ,application/x
14:52:05.320151 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1451, len 28)
0x0000 4500 001c 05ab 4000 8001 b7bf c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 98f8 b406 b300 742f 706c ............t/pl
0x0020 6169 6e3b 713d 302e 382c 7669 6465      ain;q=0.8,vide
14:52:06.320274 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1452, len 28)
0x0000 4500 001c 05ac 4000 8001 b7be c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 97f8 b406 b400 2c74 6578 ............,tex
0x0020 742f 6373 732c 2a2f 2a3b 713d 302e      t/css,*/*;q=0.
14:52:07.320319 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1453, len 28)
0x0000 4500 001c 05ad 4000 8001 b7bd c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 96f8 b406 b500 677a 6970 ............gzip
0x0020 2c20 6465 666c 6174 652c 2063 6f6d      ,.deflate,.com
14:52:08.320393 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1455, len 28)
0x0000 4500 001c 05af 4000 8001 b7bb c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 95f8 b406 b600 436f 6f6b ............Cook
0x0020 6965 3a20 4153 5053 4553 5349 4f4e      ie:.ASPSESSION
14:52:09.320478 192.168.222.11 > 192.168.222.25: icmp: echo reply
(DF) (ttl
128, id 1456, len 28)
0x0000 4500 001c 05b0 4000 8001 b7ba c0a8 de0b E.....@.........
0x0010 c0a8 de19 0000 94f8 b406 b700 3a20 6874 ............:.ht
0x0020 7470 3a2f 2f77 7777 2e63 7279 7374      tp://www.cryst
```

3.4. Sklopovski spremnik za slanje mrežnog sučelja

Neki upravljački programi imaju posebno ozbiljnu inačicu nedostatka prilikom ispune Ethernet paketa. Naime, ti upravljački programi mogu dati mrežnoj kartici neispravni broj okteta koji su postavljeni u sklopovski spremnik za slanje. Kada se tako nešto dogodi oktete za ispunu određuje sadržaj prethodnog okvira koji se nalazio u spremniku za slanje. Linux 2.4 `axnet_cs.c` upravljački program radi upravo na taj način. Programski kôd neispravno informira uređaj da je minimalni broj okteta kopiran u `Tx` spremnik, dok je ustvari tamo kopirana samo stvarna duljina okvira. Mrežna kartica tada šalje onoliko okteta koliko je navedeno u informaciji, odnosno kao ispunu okvira šalje oktete iz `Tx` spremnika.

3.4.1. Programski kôd upravljačkog programa

```
static int ei_start_xmit(struct sk_buff *skb, struct net_device
*dev)
{
long e8390_base = dev->base_addr;
struct ei_device *ei_local = (struct ei_device *) dev->priv;
int length, send_length, output_page;
unsigned long flags;
netif_stop_queue(dev);
length = skb->len;
/* Mask interrupts from the ethercard.
SMP: We have to grab the lock here otherwise the IRQ handler
on another CPU can flip window and race the IRQ mask set. We end
up trashing the mcast filter not disabling irqs if we dont lock
*/
spin_lock_irqsave(&ei_local->page_lock, flags);
outb_p(0x00, e8390_base + EN0_IMR);
spin_unlock_irqrestore(&ei_local->page_lock, flags);
/*
* Slow phase with lock held.
*/
disable_irq_nosync(dev->irq);
spin_lock(&ei_local->page_lock);
ei_local->irqlock = 1;
send_length = ETH_ZLEN < length ? length : ETH_ZLEN;
/*
* We have two Tx slots available for use. Find the first free
* slot, and then perform some sanity checks. With two Tx bufs,
* you get very close to transmitting back-to-back packets. With
* only one Tx buf, the transmitter sits idle while you reload the
* card, leaving a substantial gap between each transmitted packet.
*/
if (ei_local->tx1 == 0)
{
output_page = ei_local->tx_start_page;
ei_local->tx1 = send_length;
if (ei_debug && ei_local->tx2 > 0)
printk(KERN_DEBUG "%s: idle transmitter tx2=%d,
lasttx=%d, txing=%d.\n",
dev->name, ei_local->tx2, ei_local->lasttx, ei_local-
>txing);
}
else if (ei_local->tx2 == 0)
{
output_page = ei_local->tx_start_page + TX_1X_PAGES;
ei_local->tx2 = send_length;
if (ei_debug && ei_local->tx1 > 0)
```

```
printk(KERN_DEBUG "%s: idle transmitter, tx1=%d,
lasttx=%d, txing=%d.\n",
dev->name, ei_local->tx1, ei_local->lasttx,
ei_local->txing);
}
else
{ /* We should never get here. */
if (ei_debug)
printk(KERN_DEBUG "%s: No Tx buffers free! tx1=%d
tx2=%d last=%d\n",
dev->name, ei_local->tx1, ei_local->tx2,
ei_local->lasttx);
ei_local->irqlock = 0;
netif_stop_queue(dev);
outb_p(ENISR_ALL, e8390_base + EN0_IMR);
spin_unlock(&ei_local->page_lock);
enable_irq(dev->irq);
ei_local->stat.tx_errors++;
return 1;
}
/*
* Okay, now upload the packet and trigger a send if the transmitter
* isn't already sending. If it is busy, the interrupt handler will
* trigger the send later, upon receiving a Tx done interrupt.
*/
ei_block_output(dev, length, skb->data, output_page);
if (! ei_local->txing)
{
ei_local->txing = 1;
NS8390_trigger_send(dev, send_length, output_page);
dev->trans_start = jiffies;
if (output_page == ei_local->tx_start_page)
{
ei_local->tx1 = -1;
ei_local->lasttx = -1;
}
else
{
ei_local->tx2 = -1;
ei_local->lasttx = -2;
}
}
else ei_local->txqueue++;
if (ei_local->tx1 && ei_local->tx2)
netif_stop_queue(dev);
else
netif_start_queue(dev);
/* Turn 8390 interrupts back on. */
ei_local->irqlock = 0;
outb_p(ENISR_ALL, e8390_base + EN0_IMR);
spin_unlock(&ei_local->page_lock);
enable_irq(dev->irq);
dev_kfree_skb (skb);
ei_local->stat.tx_bytes += send_length;
return 0;
}
```

Varijabli `length` pridružena je stvarna duljina paketa. Vrijednost varijable se uspoređuje sa minimalnom duljinom okvira, a veća od tih vrijednosti pridružuje se varijabli `send_length`. Ovisno

o tome koji od dva `Tx` spremnika je slobodan, uređaj se obavještava o duljini okvira. Uočava se da se prilikom kopiranja na uređaj koristi stvarna duljina okvira.
Okteti za ispunu uvijek će sadržati podatke iz prethodno poslanih okvira. Ovo je potencijalno najozbiljniji nedostatak pošto se podaci koji se nalaze u okvirima za ispunu odnose na neke od posljednje poslanih paketa.

3.4.2. Hvatanje okvira

Primjer u nastavku preuzet je sa računala sa Linux 2.4.x jezgrom, te PCMCIA mrežnom karticom i `axnet_cs.c` upravljačkim programom. Iz `tcpdump` ispisa lako je uočiti problem curenja informacija.

```
14:18:48.146095 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 83f9 5206 2200 00
14:18:48.146393 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id 3747, len 29)
4500 001d 0ea3 0000 ff01 9668 0a32 0135
0a32 013c 0000 8bf9 5206 2200 0059 0100
6c02 0000 c006 0000 0600 0000 0010
14:18:49.146095 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 82f9 5206 2300 00
14:18:49.146387 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id 3749, len 29)
4500 001d 0ea5 0000 ff01 9666 0a32 0135
0a32 013c 0000 8af9 5206 2300 0000 4003
2300 4003 f101 1200 ed01 feff 0000
14:18:50.146093 10.50.1.60 > 10.50.1.53: icmp: echo request (DF)
(ttl
64, id 0, len 29)
4500 001d 0000 4000 4001 240c 0a32 013c
0a32 0135 0800 81f9 5206 2400 00
14:18:50.146396 10.50.1.53 > 10.50.1.60: icmp: echo reply (ttl 255,
id 3751, len 29)
4500 001d 0ea7 0000 ff01 9664 0a32 0135
0a32 013c 0000 89f9 5206 2400 0000 8002
1700 e002 1700 4003 9101 c001 0600
```

4. Zaključak

U ovom dokumentu detaljno je prikazan nedostatak prilikom upravljanja ispunom Ethernet paketa. Ovaj nedostatak u prijenosnom sloju uočen je na više platformi, a pretpostavlja se da pogađa i mnoge druge. Važno je napomenuti da mnogi sustavi pogođeni ovim nedostatkom predstavljaju kritične točke mrežne infrastrukture u mnogim organizacijama. Ovog trenutka teško je uopće utvrditi točno koji sustavi su zahvaćeni ovim nedostatkom i do kolikog stupnja je zbog toga narušena opća razina sigurnosti pojedinih sustava.