Fuzzing del firewall di Linux con syzkaller

Whoami

1.

Davide Ornaghi

Offensive Security Specialist – Co-founder @ Betrusted

- Penetration Tester
- Linux Vulnerability Researcher
- Occasional conference speaker
- No Hat fellow
- (Syzkaller contributor)

Betrusted

Introducing syzkaller

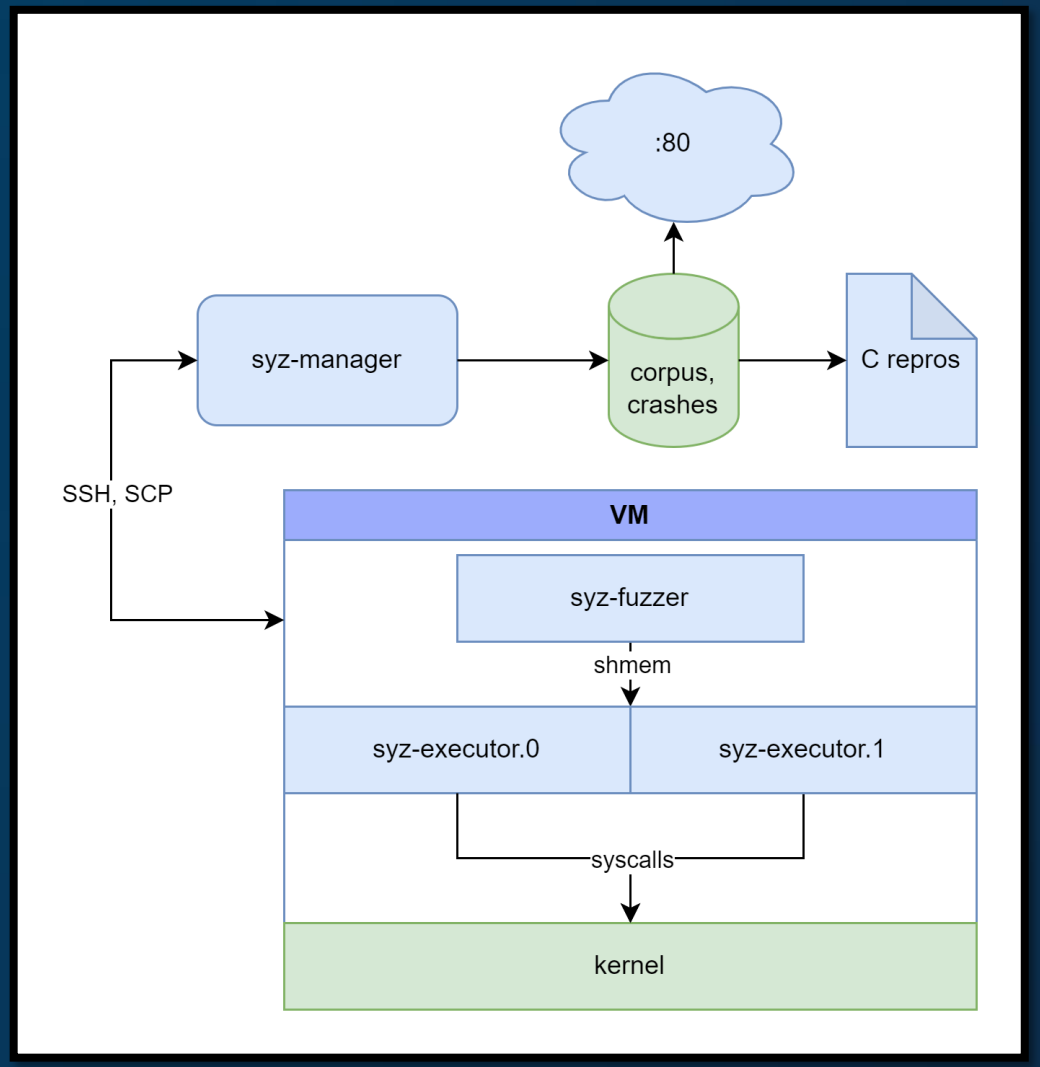
2.

What is syzkaller?

- The most popular (best) kernel fuzzer
- Unsupervised (?)
- Coverage-guided
- Supports Linux, *BSD, Android, ...
- Multiple modules written in C and Go

What does it look like?

MODULE	ROLE
SYZ-MANAGER	Manages VMs and communicates via RPC to detect and reproduce crashes.
SYZ-FUZZER	Generates new test programs and mutates the existing ones via workers.
SYZ-EXECUTOR	Configures the fuzzing VMs, runs syzlang programs. It also collects coverage and sends it to syz-manager through syz-fuzzer.
SYZ-PROG2C	Translates syzlang programs into C sources.
SYZ-COVER	Allows dumping current coverage to file by converting raw coverage data into html or json.



Fuzzing the network stack

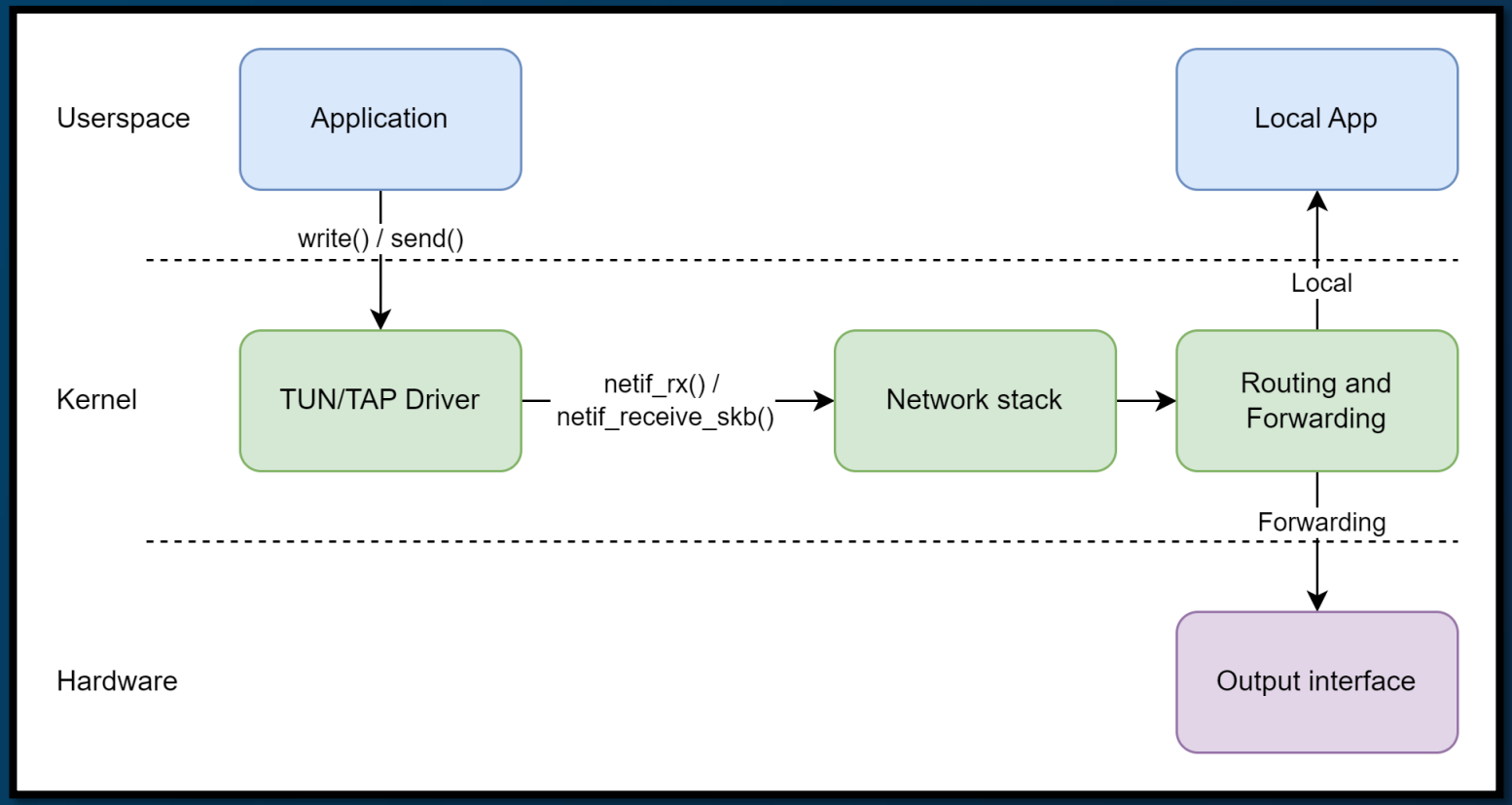
3.

Data flow

01 Generate «random» PDUs

02 Inject them into the stack

03 Collect coverage & repeat



https://xairy.io/articles/syzkaller-external-network

moca 2024

TUN/TAP Implementation

```
static void initialize_tun(void)
{
#if SYZ_EXECUTOR
    if (!flag_net_injection)
        return;
#endif
    tunfd = open("/dev/net/tun", O_RDWR | O_NONBLOCK);
    if (tunfd == -1) {
#if SYZ_EXECUTOR
        fail("tun: can't open /dev/net/tun");
#else
        printf("tun: can't open /dev/net/tun: please enable CONFIG_TUN=y\n");
        printf("otherwise fuzzing or reproducing might not work as intended\n");
        return;
#endif
    }
```

Uses TAP interface to process raw frames:

```
    strncpy(ifr.ifr_name, TUN_IFACE, IFNAMSIZ);
    ifr.ifr_flags = IFF_TAP | IFF_NO_PI;
    // Note: SYZ_ENABLE_NAPI_FRAGS is never enabled. This is cod
    // in case we figure out how IFF_NAPI_FRAGS works. With IFF_
    // don't reach destinations and bail out in udp_gro_receive
    // Also IFF_NAPI_FRAGS does not work with sandbox_namespace
#if ENABLE_NAPI_FRAGS
    ifr.ifr_flags |= IFF_NAPI | IFF_NAPI_FRAGS;
#endif
    if (ioctl(tunfd, TUNSETIFF, (void*)&ifr) < 0) {
```

Fuzzing Nftables

4.

Nftables concepts

TABLES

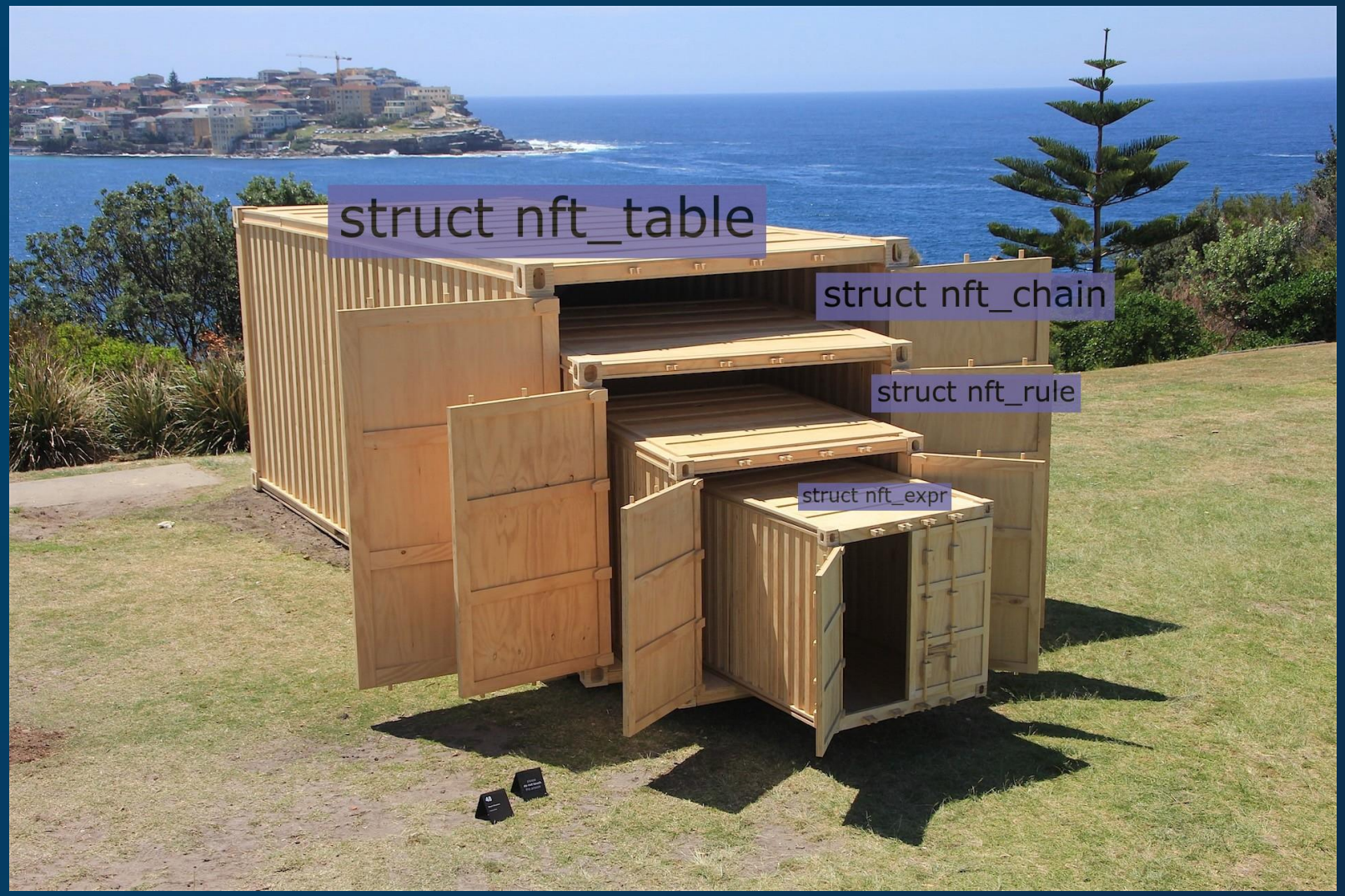
CHAINS

RULES

EXPRESSIONS

Fuzzing Nftables

https://blog.dbouman.nl/2022/04/02/How-The-Tables-Have-Turned-CVE-2022-1015-1016/#4-cve-2022-1015



moca 2024

Main issues

01.

TAP Interface

TAP interfaces take L2 frames as input, but nftables deals mainly with L3+ traffic.

02.

NFT complexity

For coverage to reach rule evaluation, a program must successfully setup a table, chain and a rule, and then send a packet to be filtered before the rule is discarded.

03.

Netfilter hooks

For a chain to be evaluated, its Netfilter hook must be reached by the packet path.
E.g. NF_INET_LOCAL_IN for a local packet.

Main issues

```
static noinline bool
nft_meta_get_eval_pkttype_lo(const struct nft_pktinfo *pkt,
                             u32 *dest)
{
        const struct sk_buff *skb = pkt->skb;

        switch (nft_pf(pkt)) {
        case NFPROTO_IPV4:
                if (ipv4_is_multicast(ip_hdr(skb)->daddr))
                        nft_reg_store8(dest, PACKET_MULTICAST);
                else
                        nft_reg_store8(dest, PACKET_BROADCAST);
                break;
        case NFPROTO_IPV6:
                nft_reg_store8(dest, PACKET_MULTICAST);
                break;
        case NFPROTO_NETDEV:
                switch (skb->protocol) {
                case htons(ETH_P_IP): {
                        int noff = skb_network_offset(skb);
                        struct iphdr *iph, _iph;
```

Results

```
__nft_cmp_offload                 ---        of 16
__nft_cmp_offload.cold            ---         of 1
nft_cmp16_fast_dump               100%        of 4
nft_cmp16_fast_init               93%        of 13
nft_cmp16_fast_init.cold          ---         of 1
nft_cmp16_fast_offload            ---         of 1
nft_cmp_dump                      100%        of 4
nft_cmp_eval                      ---         of 8
nft_cmp_fast_dump                 100%        of 4
nft_cmp_fast_init                 100%        of 5
nft_cmp_fast_init.cold            ---         of 1
nft_cmp_fast_offload              ---         of 1
nft_cmp_init                      100%        of 4
nft_cmp_offload                   ---         of 1
nft_cmp_select_ops                100%       of 13
nft_payload_n2h                   ---         of 5
-----------
SUMMARY                           98%        of 47
```

```
nft_ng_dump                       100%        of 5
nft_ng_inc_destroy                100%        of 1
nft_ng_inc_dump                   100%        of 1
nft_ng_inc_eval                   ---         of 5
nft_ng_inc_init                   75%         of 8
nft_ng_inc_reduce                 ---         of 1
nft_ng_random_dump                100%        of 1
nft_ng_random_eval                ---         of 1
nft_ng_random_init                100%        of 6
nft_ng_random_reduce              ---         of 1
nft_ng_select_ops                 100%        of 6
-----------
SUMMARY                           93%        of 28
```

Most evals are never tested!

https://syzkaller.appspot.com/upstream/manager/ci-qemu-upstream

moca 2024

My solution

01.

TAP Interface

Create a TUN interface alongside TAP, and only use it for nftables fuzzing.

02.

NFT complexity

Define a new pseudo-syscall which performs all the common nft setup (tables, chains & hooks) statically, only leaving rule creation to the fuzzer, and immediately sends a random packet to be processed, thus triggering nf_hook.

03.

Netfilter hooks

Only register the NF_INET_PREROUTING hook so that both local and remote packets will be processed.

Implementation

- initialize_tun()

```
    strncpy(ifr.ifr_name, TUN_IFACE, IFNAMSIZ);
    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
    // Note: SYZ_ENABLE_NAPI_FRAGS is never enabled. This is cod
    // in case we figure out how IFF_NAPI_FRAGS works. With IFF_
    // don't reach destinations and bail out in udp_gro_receive
    // Also IFF_NAPI_FRAGS does not work with sandbox_namespace
#if ENABLE_NAPI_FRAGS
    ifr.ifr_flags |= IFF_NAPI | IFF_NAPI_FRAGS;
#endif
    if (ioctl(tunfd, TUNSETIFF, (void*)&ifr) < 0) {
```

```
pthread_t cleanup_thread;
debug("Creating cleanup thread");
if (pthread_create(&cleanup_thread, NULL, nft_cleanup, NULL) != 0) {
    fail("Failed to create thread");
}
```

```
nftfd = syscall(__NR_socket, /*domain=*/0x10ul, /*type=*/3ul, /*proto=*/0xc);
if (nftfd == -1)
    fail("Failed to create socket");
uint8_t init_syz0[] = {
    0x14, 0x00, 0x00, 0x00, 0x10, 0x00, 0x01, 0x00, 0x9e, 0x7e, 0xd5, 0x66, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0a, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x05, 0x04, 0x9f, 0x7e, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x09, 0x00, 0x01, 0x00, 0x73, 0x79, 0x7a, 0x30,
    0x00, 0x79, 0x05, 0x00, 0x40, 0x00, 0x00, 0x00, 0x03, 0x0a, 0x05, 0x04, 0xa0, 0x7e, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x09, 0x00, 0x01, 0x00, 0x73, 0x79, 0x7a, 0x30,
    0x00, 0x58, 0x21, 0x00, 0x09, 0x00, 0x03, 0x00, 0x73, 0x79, 0x7a, 0x30, 0x00, 0x00, 0x00, 0x00,
    0x14, 0x00, 0x04, 0x80, 0x08, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x02, 0x00,
    0xff, 0xff, 0xff, 0xff, 0x14, 0x00, 0x00, 0x00, 0x11, 0x00, 0x01, 0x00, 0xa1, 0x7e, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00};
uint8_t init_syz1[] = {
    0x14, 0x00, 0x00, 0x00, 0x10, 0x00, 0x01, 0x00, 0xc5, 0x82, 0xd5, 0x66, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0a, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x05, 0x04, 0xc6, 0x82, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x09, 0x00, 0x01, 0x00, 0x73, 0x79, 0x7a, 0x31,
    0x00, 0xd0, 0x1b, 0x00, 0x40, 0x00, 0x00, 0x00, 0x03, 0x0a, 0x05, 0x04, 0xc7, 0x82, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x09, 0x00, 0x01, 0x00, 0x73, 0x79, 0x7a, 0x31,
    0x00, 0x10, 0x00, 0x00, 0x09, 0x00, 0x03, 0x00, 0x73, 0x79, 0x7a, 0x31, 0x00, 0x00, 0x00, 0x00,
    0x14, 0x00, 0x04, 0x80, 0x08, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x02, 0x00,
    0xff, 0xff, 0xff, 0xff, 0x14, 0x00, 0x00, 0x00, 0x11, 0x00, 0x01, 0x00, 0xc8, 0x82, 0xd5, 0x66,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00};
static const struct sockaddr_nl snl = {
    .nl_family = AF_NETLINK};
sendto(nftfd, (const void*)init_syz0, 136, 0, (struct sockaddr*)&snl, sizeof(snl));
sendto(nftfd, (const void*)init_syz1, 136, 0, (struct sockaddr*)&snl, sizeof(snl));
```

Option 1

- syz_batch_emit() + Output UDP

```c
static long syz_batch_emit(volatile long msg, int f, volatile long a0, volatile long a1)
{
    // syz_batch_emit(fd sock_nl_netfilter, msg ptr[in, msghdr_netlink[nft_batch_msg]], f flags
    int ret = sendmsg(nftfd, (const struct msghdr*)msg, f);
    if (ret < 0)
        return ret;

    const char* udpmsg = "AAAA";
    struct sockaddr_in servaddr;

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("0.0.0.0");
    servaddr.sin_port = htons(1337);
    sendto(sockfd, udpmsg, strlen(udpmsg) + 1, 0,
           (struct sockaddr*)&servaddr, sizeof(servaddr));
}
```

Option 2

- syz_batch_emit() + syz_emit_ethernet()

```
static long syz_batch_emit(volatile long msg, int f, volatile long a0, volatile long a1)
{
    // batch_and_emit(fd sock_nl_netfilter, msg ptr[in, msghdr_netlink[nft_batch_msg]], f
    int ret = sendmsg(nftfd, (const struct msghdr*)msg, f);
    if (ret < 0)
        return ret;

    return syz_emit_ethernet(a0, a1, 0);
}
```

Option 2: Implementation

- syz_batch_emit() + syz_emit_ethernet()

```
syz_batch_emit(msg ptr[in, msghdr_netlink[nft_batch_msg_new]], f flags[send_flags],
               len len[packet], packet ptr[in, ipv4_packet])

nft_batch_msg_new {
    begin   nft_nlmsghdr[NFNL_MSG_BATCH_BEGIN]
    msgs    netlink_msg_netfilter_t[NFNL_SUBSYS_NFTABLES, NFT_MSG_NEWRULE, nft_rule_policy]
    end nft_nlmsghdr[NFNL_MSG_BATCH_END]
} [packed]
```

Results

```
nft_counter_clone                  ---          of 5
nft_counter_destroy                100%         of 1
nft_counter_do_dump                100%         of 5
nft_counter_do_init                78%          of 9
nft_counter_dump                   100%         of 1
nft_counter_eval                   100%         of 1
nft_counter_fetch                  89%          of 9
nft_counter_init                   100%         of 1
nft_counter_init_seqcount          ---          of 5
nft_counter_obj_destroy            ---          of 1
nft_counter_obj_dump               ---          of 1
nft_counter_obj_eval               ---          of 1
nft_counter_obj_init               ---          of 1
nft_counter_offload                ---          of 1
nft_counter_offload_stats          ---          of 2
-----------
SUMMARY                            89%          of 27
```

```
nft_ng_dump                        100%         of 5
nft_ng_inc_destroy                 ---          of 1
nft_ng_inc_dump                    ---          of 1
nft_ng_inc_eval                    100%         of 5
nft_ng_inc_init                    100%         of 8
nft_ng_inc_reduce                  ---          of 1
nft_ng_random_dump                 100%         of 1
nft_ng_random_eval                 100%         of 1
nft_ng_random_init                 84%          of 6
nft_ng_random_reduce               ---          of 1
nft_ng_select_ops                  100%         of 6
-----------
SUMMARY                            97%          of 32
```

Evals are tested even locally!

moca 2024

Results

Stats 📈

VMs	4
candidates	0
corpus	832
coverage	10305
exec total	16668122 (240/sec)
reproducing	0
crash types	3
crashes	178
suppressed	0
syscalls	4
uptime	69253 sec

Crashes:

Description	Count	Last Time	Report
general protection fault in	100	2024/05/24 10:36	
general protection fault in	9	2024/05/24 08:06	
general protection fault in	46	2024/05/24 10:30	

Thanks!