rx hardening
& udp gso

willem de bruijn

Network rx stack hardening

PoD

- [redacted (3x)]

Local Priv

- CVE-2017-1000112: ufo overwrite skb_shared_info
- CVE-2017-1000111: packet_reserve use-after-free
  - user namespaces

Network rx stack hardening

- Prevent
  - syzkaller
    - tun napi_gro_frags + eth_get_headlen
- Reduce attack surface
  - Flow dissector
  - GRO and frags
  - BUG_ON
- Big red button
  - XDP

Flow dissector

- in rx stack, so under adversary ctrl
  - eth_get_headlen
  - __skb_get_hash
    - get_rps_cpu
    - packet fanout
    - tpacket_v3
  - cls_flower

- reduce attack surface
  - FLOW_DISSECTOR_KEY_…
  - safe parser language
    - skb_set_hash from a BPF hook
      - XDP? ...

Flow dissector: XDP?

- extract {headlen, rxhash}
- before GRO => run on each MSS
  - though skb->hash will make GRO fail faster (0b4cec8c2e87)

- far-out: could have single FD + GRO pass?
  - for GRO, also extract {NAPI_GRO_CB(skb)->{count, last, ..}, gro_result_t}
    - napi_gro_skb: 20 fields

GRO and napi_gro_frags

- same approach: turn off rarely exercised edge cases
  - majority of traffic from small set of protocols
    - exclude extension headers, frags, …
  - under administrator control
    - apply fix without reboot
    - modify parse graph (sysctl?)
      - else per-packet opt-out (XDP)
- dev_add_offload
  - {ip, ipv6, eth, vlan, mpls_[um]c, nsh}
  - net_offload.flags & INET6_PROTO_GSO_EXTHDR
- inet[6]_add_offload
  - {tcp, udp, ipip/sit/.., esp, gre, sctp, v6_routing, v6_dstopts (, v6_nexthop)}
  - packet_offload.priority < THRESHOLD

BUG_ON

egrep -nrl '(\<BUG\>|\<BUG_ON\>)'

- net: 810
  - net/core: 76
  - net/ipv4: 81
  - net/ipv6: 34

low hanging fruit: just a lot of drudgery work..

BUG_ON: skb_copy

```
/**
 *      skb_copy        -       create private copy of an sk_buff
 [..]
 *      data to alter. Returns %NULL on failure or the pointer to the buffer
 [..]
struct sk_buff *skb_copy(const struct sk_buff *skb, gfp_t gfp_mask)
{
 [..]
        if (!n)
                return NULL;

        /* Set the data pointer */
        skb_reserve(n, headerlen);
        /* Set the tail pointer and length */
        skb_put(n, skb->len);

        BUG_ON(skb_copy_bits(skb, -headerlen, n->head, headerlen + skb->len));

        copy_skb_header(n, skb);
        return n;
}
EXPORT_SYMBOL(skb_copy);
```

UDP GSO: motivation

QUIC: send multiple datagrams with same UDP header

```
# tcp
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4 -t
 1123 MB/s  8991 msg/s 8991 calls/s
     5,759,912,562      cycles

# connected udp
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4
 1142 MB/s  9141 msg/s 813549 calls/s
    15,884,155,703      cycles

# connectionless does: does NOT saturate link
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4 -u
  940 MB/s  7527 msg/s 669903 calls/s
    16,426,226,536      cycles

# tcp with tso off gso off: does NOT saturate link either
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4 -t
  887 MB/s  7100 msg/s 7100 calls/s
    16,865,336,172      cycles
```

UDP GSO: benchmark

QUIC use-case: batch datagrams, replicate UDP header

```
# tcp
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4 -t
 1123 MB/s  8991 msg/s 8991 calls/s
     5,759,912,562      cycles

# connected udp
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4
 1142 MB/s  9141 msg/s 813549 calls/s
    15,884,155,703      cycles

# connected udp + gso *
$ perf stat -a -C 4,5 -e cycles ./sendstream -4 -C 4 -D $DST -l 4 -U -u
 1147 MB/s  9183 msg/s 27549 calls/s
    10,602,866,911      cycles
```

(*) zero checksum in initial hack

UDP GSO: design

- amortize packet traversal
  - xmit_more
  - protocol stack
    - batching of individual expensive operations
      - skb_set_owner_w call in sendmmsg
      - enqueue skb chain when qdisc lock held
    - simpler: segmentation offload

- GSO: write large packet of N segments (including L7 headers, evenly spaced)
  - proof-of-concept
    - IP_PMTUDISC_INTERFACE
      - builds > mtu
        - do not fragment locally
        - do not set DF bit
    - set gso_{type, size}
  - needs better production API
    - setsockopt/cmsg gso_size OR
    - infer from sendmmsg + MSG_BATCH
      - SO_BATCH (SO_STRICT?)

UDP GSO: zerocopy

remaining cost after gso:

#perf record -a -C 4,5 -e cycles /export/hda3/tmp/sendstream -4 -C 4 -D $DST -l 4 -U -u

```
    36.10%          perf  [kernel.kallsyms]  [k] copy_user_generic_string
     9.70%      swapper  [kernel.kallsyms]  [k] bnx2x_start_xmit
     4.01%      swapper  [kernel.kallsyms]  [k] skb_segment
     1.76%         perf  [kernel.kallsyms]  [k] __ip_append_data.isra.48
     1.67%      swapper  [kernel.kallsyms]  [k] bnx2x_free_tx_pkt
     1.53%         perf  [kernel.kallsyms]  [k] skb_segment
     1.50%         perf  [kernel.kallsyms]  [k] bnx2x_start_xmit
     1.38%      swapper  [kernel.kallsyms]  [k] __memcpy
```

With gso, MSG_ZEROCOPY feasible for UDP.

UDP GSO: scale to servers

- don't burst send 45 MSS at a time: pace

- connected sockets is feasible on client (android)
- but not on many-connection server
  - no 4-tuple udp port hashtables, unlike tcp

- pacing + fairness for connectionless udp
  - 4-tuple SFQ fallback in FQ, or
  - cmsg SCM_TX_DEADLINE
    - connectionless SO_MAX_PACING_RATE
    - see also time based packet transmission patchset
  - optional: BQL support
    - tso_segment on dequeue to send out fewer MSS
  - optimize: maximize xmit_more effectiveness for gso chain