LBM: A Security Framework for Peripherals within the Linux Kernel

Dave (Jing) Tian*^, Grant Hernandez*, Joseph Choi*, Vanessa Frost*, Peter Johnson**, and Kevin Butler*

*University of Florida, Gainesville, FL
^Purdue University, West Lafayette, IN
**Middlebury College, VT

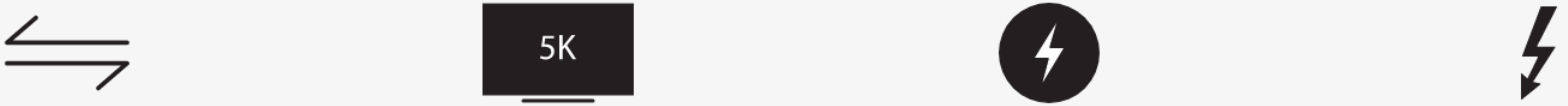
May 22, 2019

Peripherals

Modern Peripherals

NFC™

Modern Peripherals



Up to 40Gb/s data transfer

Supports up to two 5K displays

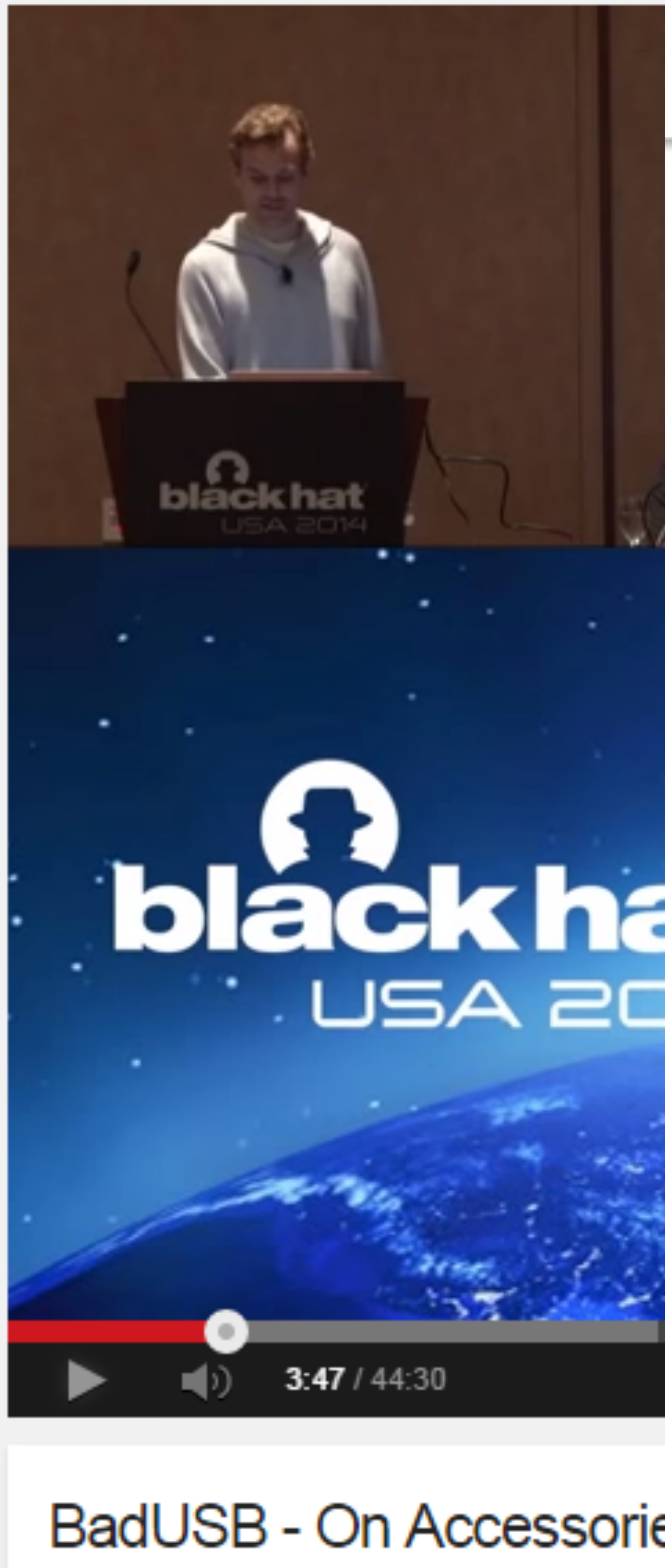
Charge and provide power from any port

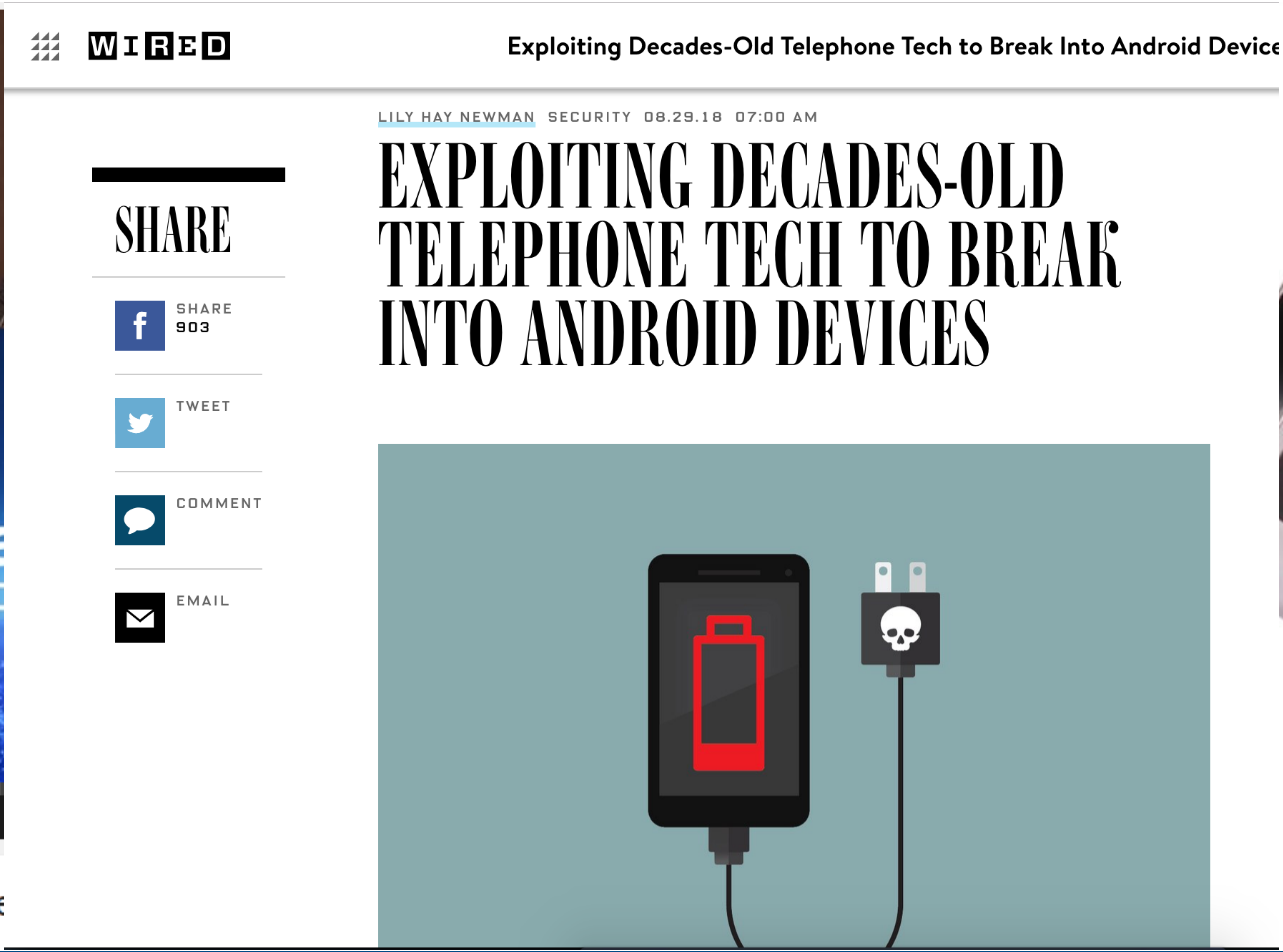
Up to four Thunderbolt 3 ports

Modern Peripherals

What could possibly go wrong??

Malicious Peripherals




WIRED
Exploiting Decades-Old Telephone Tech to Break Into Android Device

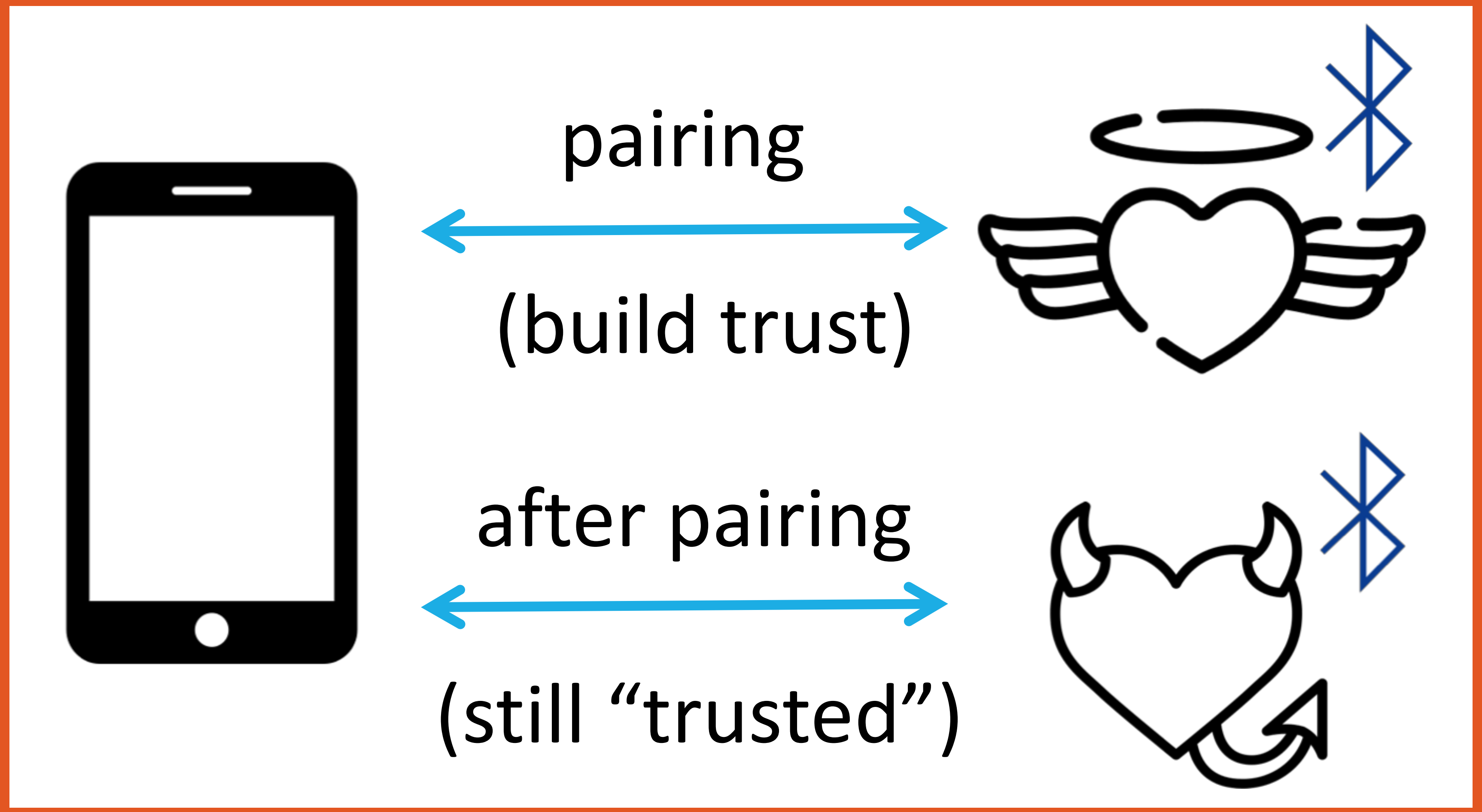
LILY HAY NEWMAN SECURITY 08.29.18 07:00 AM

EXPLOITING DECADES-OLD TELEPHONE TECH TO BREAK INTO ANDROID DEVICES

SHARE
SHARE 903
TWEET
COMMENT
EMAIL

Malicious Peripherals

5.3 Billion Devices Affected



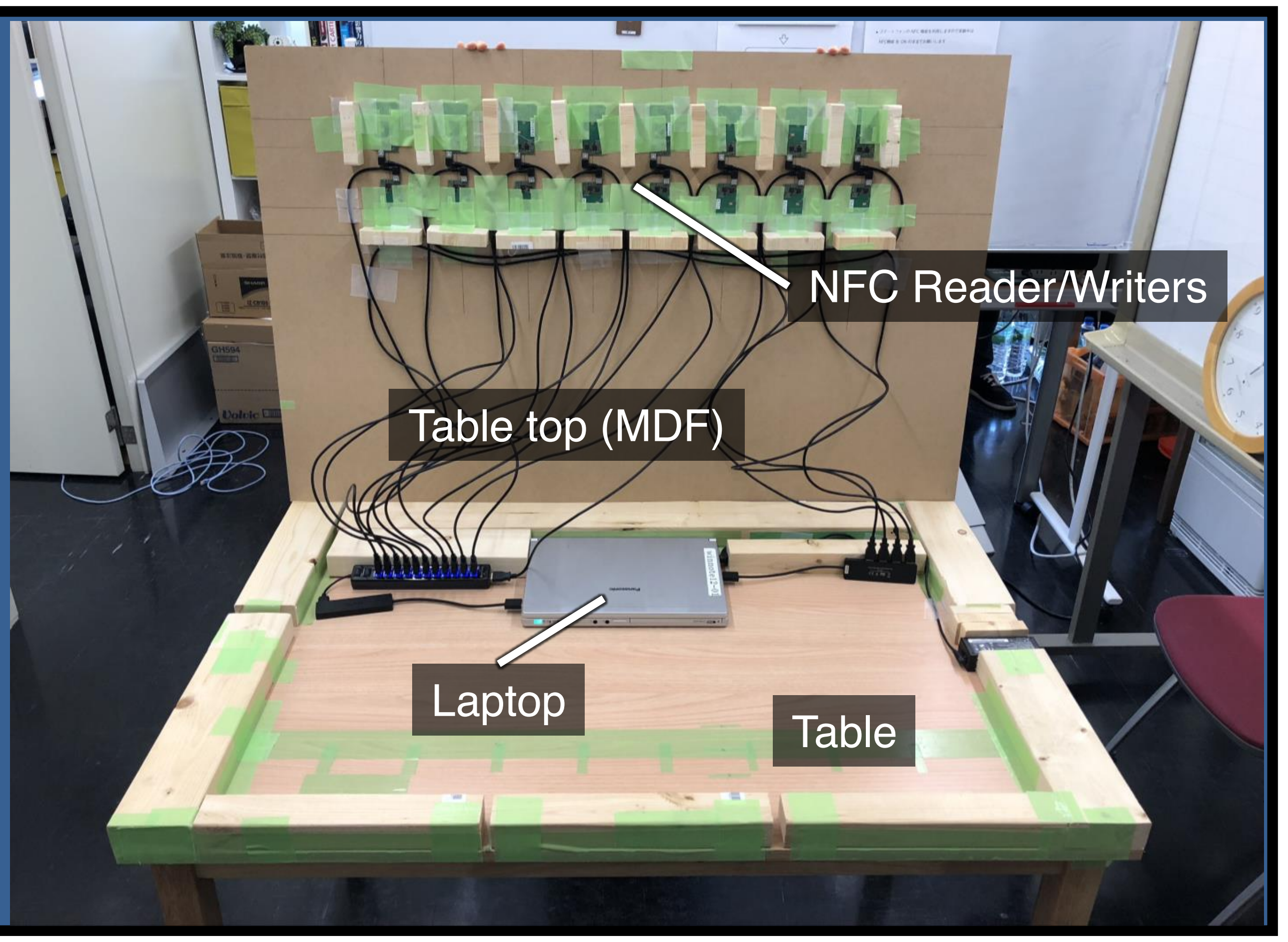
BadBluetooth Attacks

0day attacks over NFC!

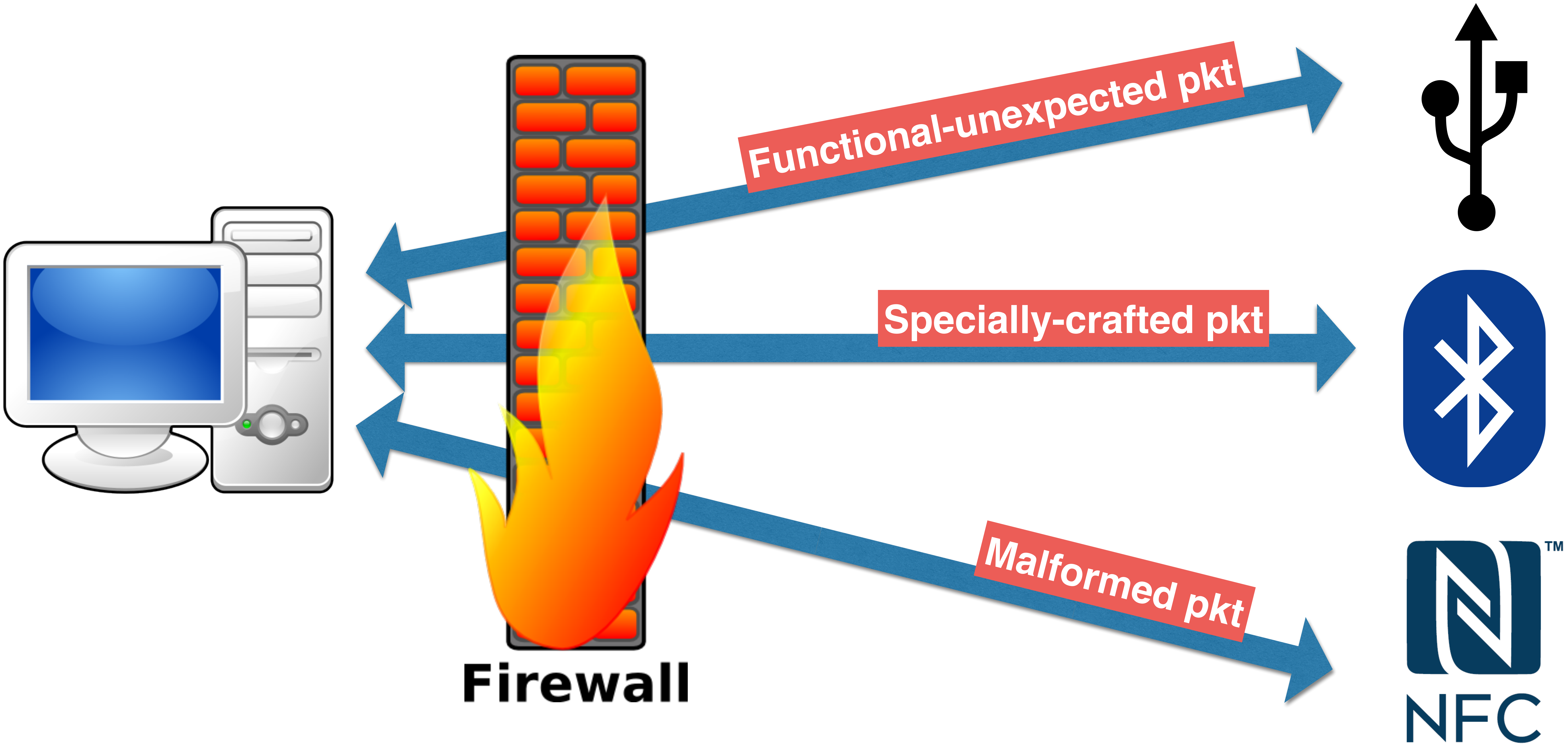
NFC

```
From: Suren Baghdasaryan <surenb@google.com>

When handling SHDLC I-Frame commands "pipe" field used for indexing
into an array should be checked before usage. If left unchecked it
might access memory outside of the array of size NFC_HCI_MAX_PIPES(127).

cc: Stable <stable@vger.kernel.org>
Signed-off-by: Suren Baghdasaryan <surenb@google.com>
Signed-off-by: Amit Pundir <amit.pundir@linaro.org>
Reviewed-by: Andy Shevchenko <andriy.shevchenko@linux.intel.com>
```

Tap’n Ghost

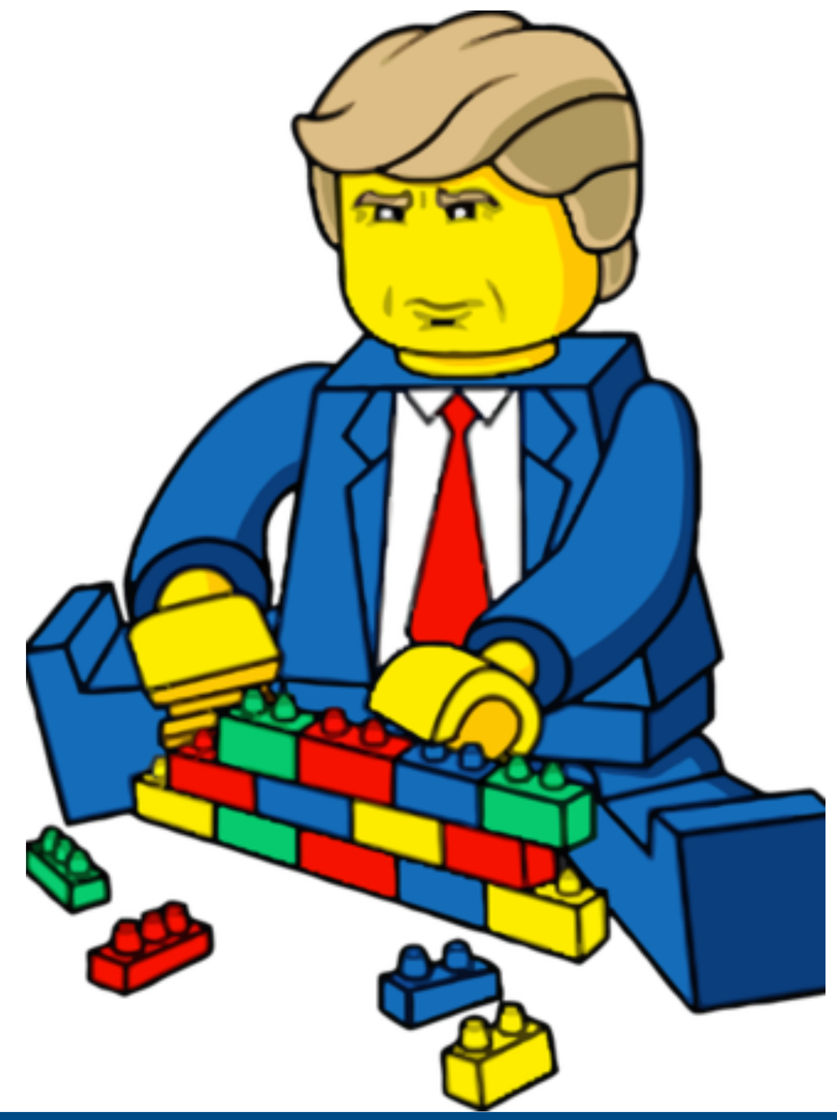


Solution?


Functional-unexpected pkt
Specially-crafted pkt
Malformed pkt
Firewall
NFC

Challenges

- **Peripheral Diversity**
  - USBFILTER (USENIX Sec'16), USBFirewall (ACSAC'17)
  - Bluetooth, NFC, etc.
- **Filtering (Rule) Complexity**
  - Programmability vs. Usability
  - Extensibility

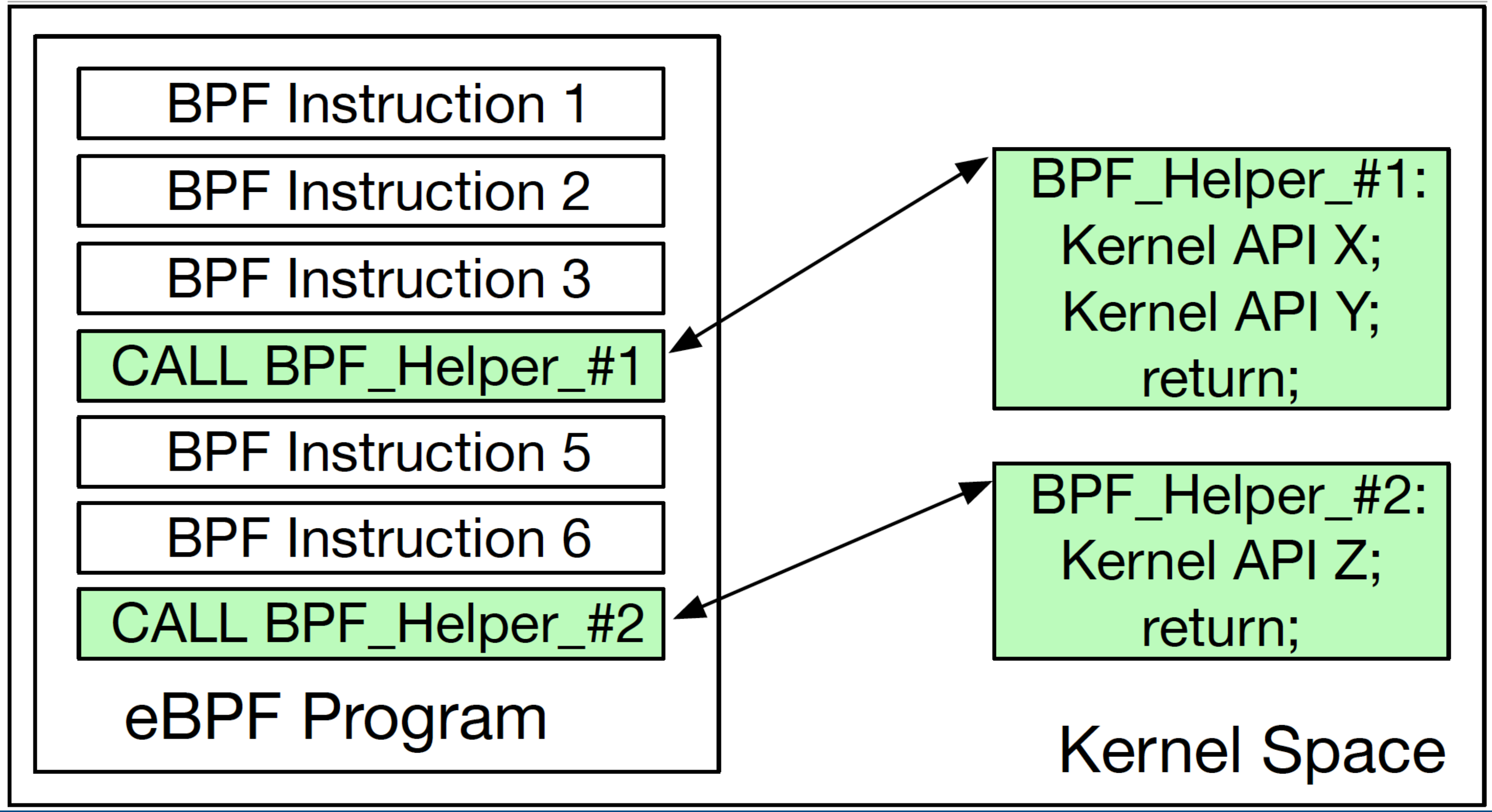


Linux (e)BPF Modules (LBM)

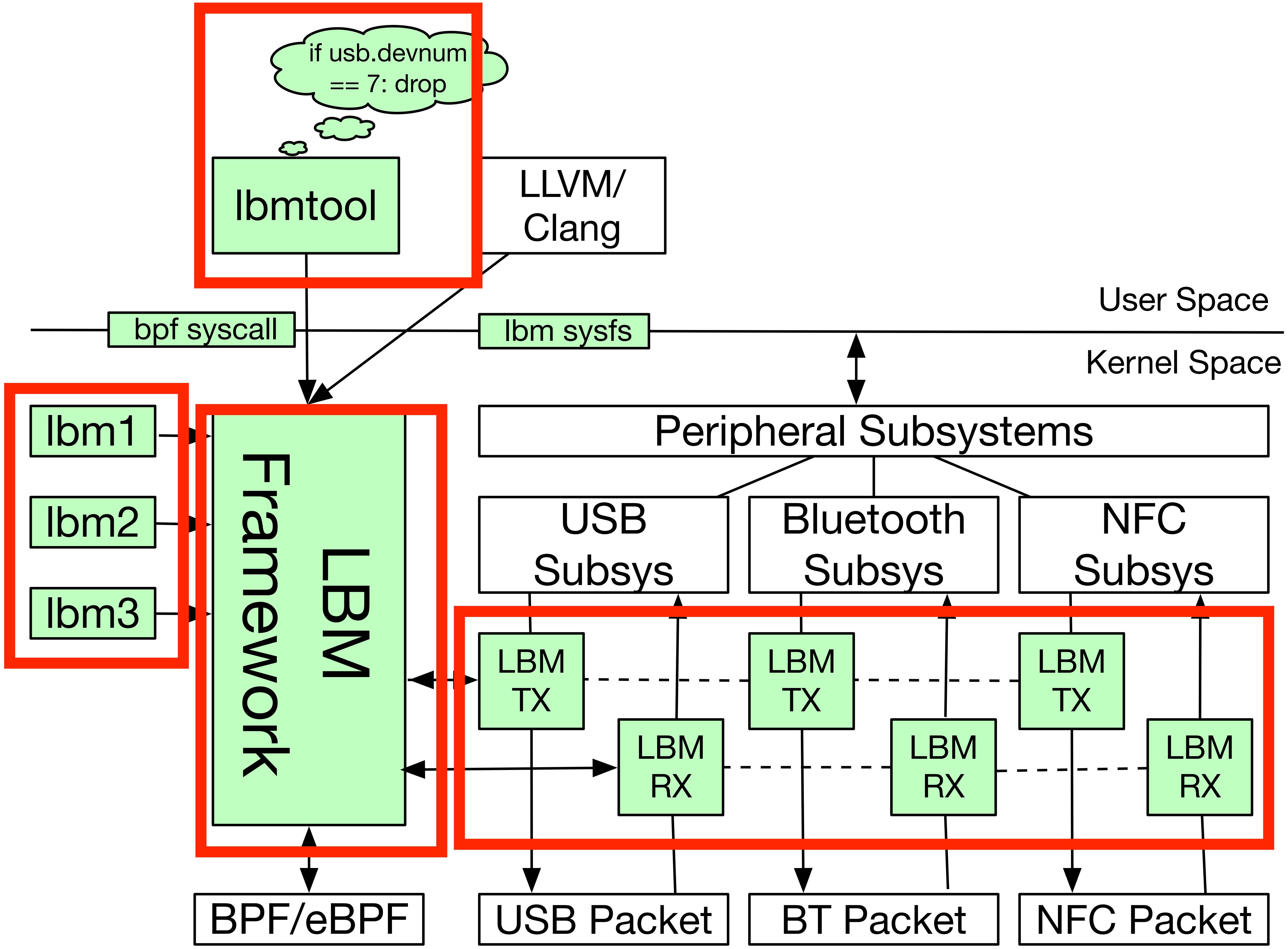
- **A generic security framework for peripherals**
  - Peripheral agnostic
  - LBM hooks
  - eBPF
  - Filter DSL
  - Module extension
  - USB, Bluetooth, NFC

Peripheral Diversity

Filtering Complexity



LBM: Architecture



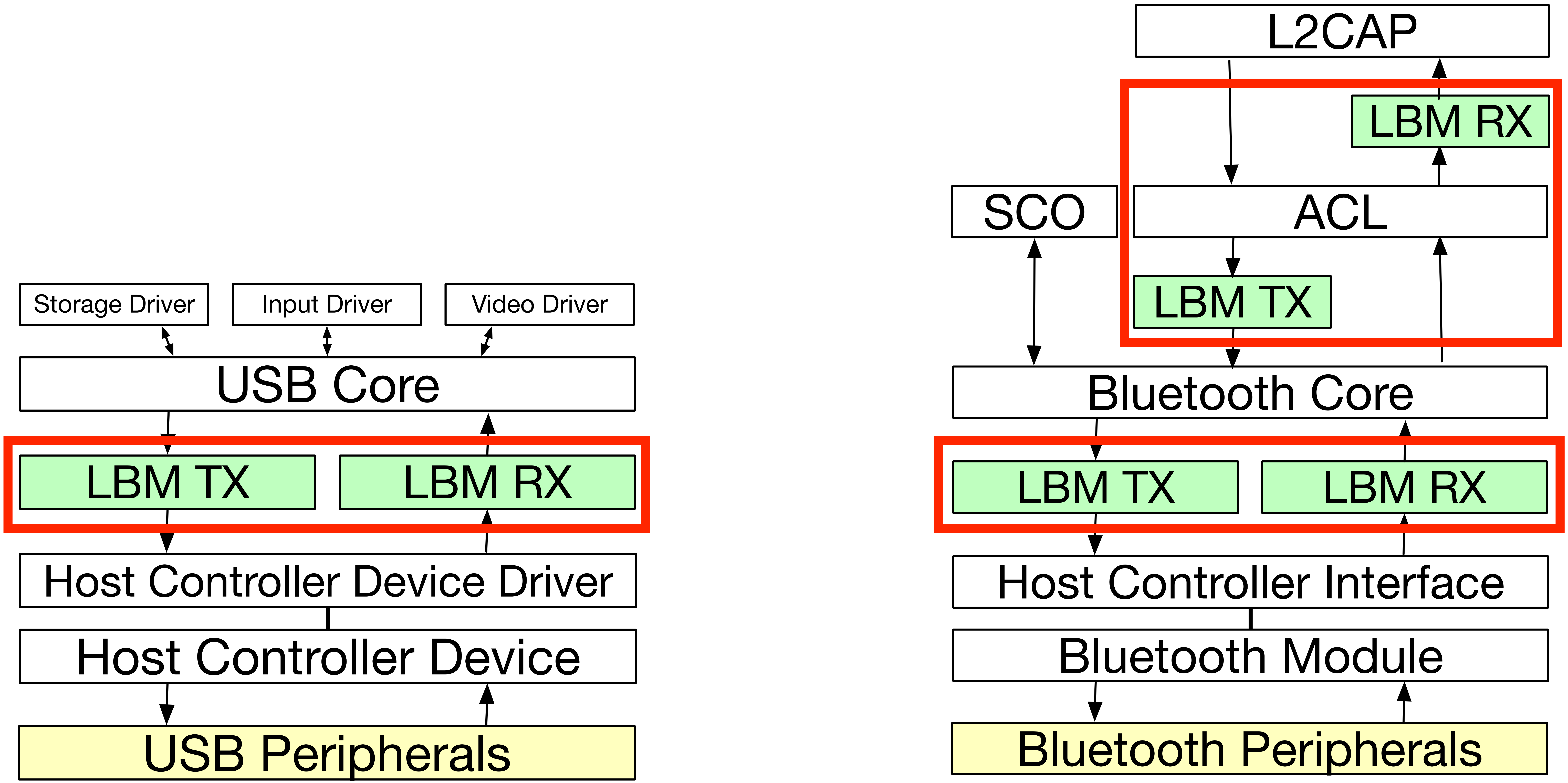
LBM: Hooks

- **Linux Security Modules (LSM)**
  - > 100 (kernel 4.13)
- **Linux (e)BPF Modules (LBM)**
  - 1

```
int lbm_filter_pkt(
        int subsys, int dir, void *pkt)
```

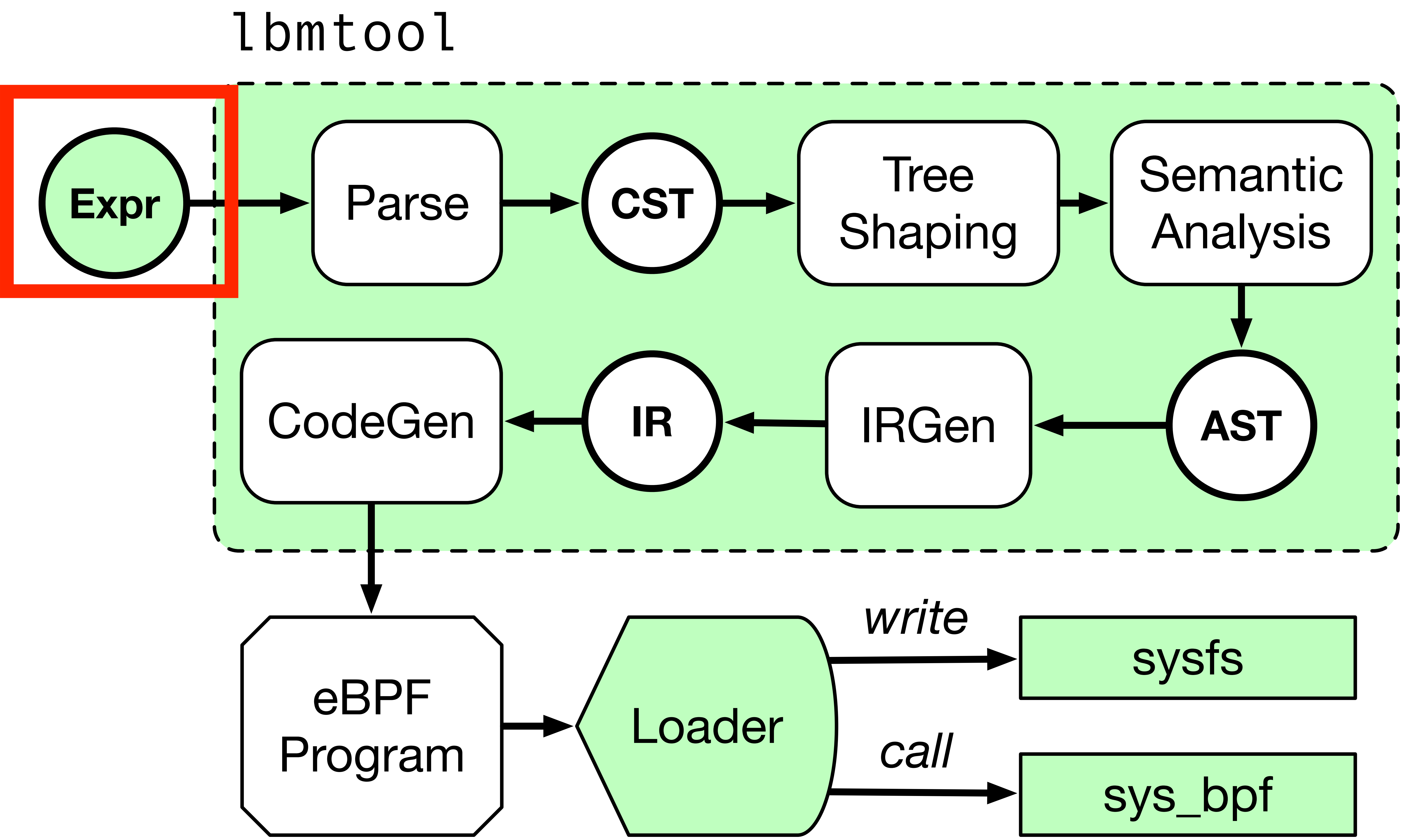
```
lbm_filter_pkt(LBM_SUBSYS_INDEX_USB, LBM_DIR_TX,
    (void *)urb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_USB, LBM_DIR_RX,
    (void *)urb);
```

LBM: Hook Placement



LBM: Filter DSL & lbmtool

usb.idVendor == 0x413c &&
usb.idProduct == 0x3010



eBPF Assembly

```
LSTART:
        MOV64_REG(REG_9, REG_1)
        MOV64_REG(REG_1, REG_9)
        CALL_FUNC(FUNC_lbm_usb_get_idVendor)
        MOV64_REG(REG_1, REG_0)
        MOV64_IMM(REG_6, 1)
        JMP_IMM(JEQ, REG_1, 16700, L1_)
        MOV64_IMM(REG_6, 0)
L1_:
        MOV64_REG(REG_1, REG_9)
        CALL_FUNC(FUNC_lbm_usb_get_idProduct)
        MOV64_REG(REG_2, REG_0)
        MOV64_IMM(REG_3, 1)
        JMP_IMM(JEQ, REG_2, 12304, L2_)
        MOV64_IMM(REG_3, 0)
L2_:
        JMP_IMM(JEQ, REG_6, 0, L3_)
        JMP_IMM(JEQ, REG_3, 0, L3_)
        MOV64_IMM(REG_4, 1)
        JMP_A(L4_)
L3_:    MOV64_IMM(REG_4, 0)
L4_:
        JMP_IMM(JNE, REG_4, 0, L5_)
L6_:    MOV64_IMM(REG_0, 0)
        EXIT_INSN()
L5_:    MOV64_IMM(REG_0, 1)
LEND:   EXIT_INSN()
```

LBM: Proof-of-Concept for NFC

Step I: Place hook

```
lbm_filter_pkt(LBM_SUBSYS_INDEX_NFC, LBM_DIR_TX,
     (void *)skb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_NFC, LBM_DIR_RX,
     (void *)skb);
```

Step II: Expose protocol fields

```
+BPF_CALL_1(lbm_nfc_nci_get_mt, struct sk_buff *, skb)
+{
+        return nci_mt(skb->data);
+}
+
+static const struct bpf_func_proto lbm_nfc_nci_get_mt_proto = {
+        .func           = lbm_nfc_nci_get_mt,
+        .gpl_only       = false,
+        .ret_type       = RET_INTEGER,
+        .arg1_type      = ARG_PTR_TO_CTX,
+};
```

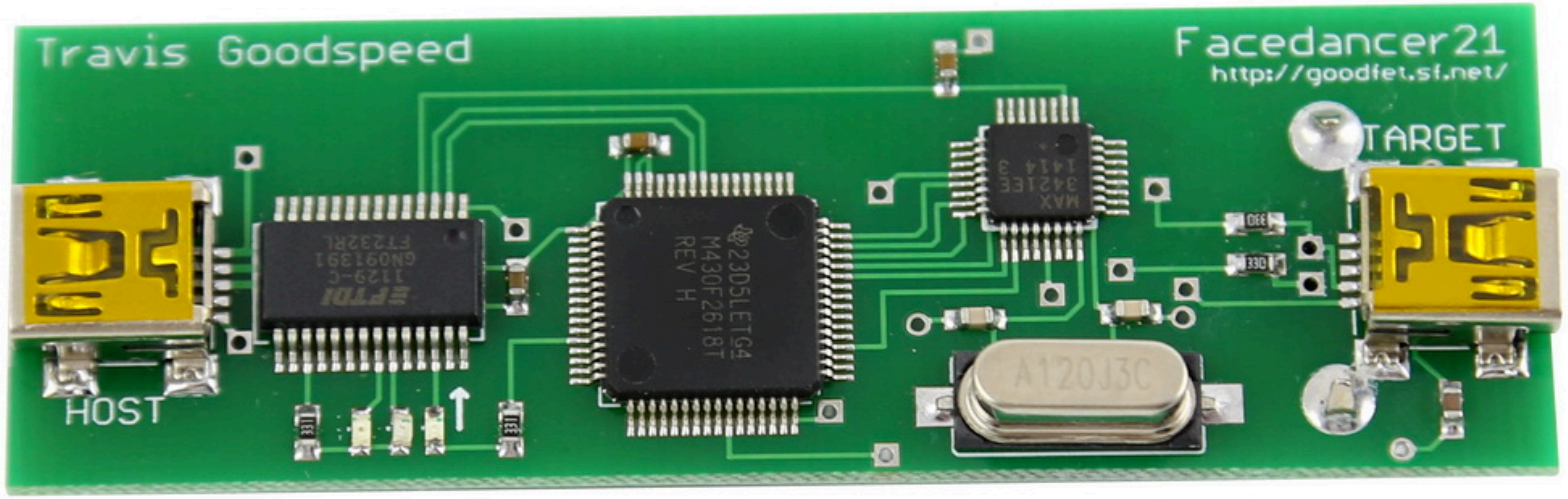
Step III: Extend lbmtool

```
+nfc_nci = {
+    "len" : SymbolContext(ty=Type.TY_INT_32, offset=0),
+    "mt"  :  SymbolHelper(ty=Type.TY_INT_32, name="lbm_nfc_nci_get_mt"),
+}
```

nci.len > 10 && nci.mt == 5

NFC	Kernel	lbmtool	Total
# of lines	85	12	97

LBM: FaceDancer Testing



LBM: Protocol Stack Protection

```
((usb.setup_packet != 0) && /* For enumeration */
 (usb.request[0] == 0x80) && /* Get_Descriptor */
 (usb.request[1] == 0x06) &&
 /* Make sure response contains at least 2 bytes
     */
 ((usb.actual_length < 2) ||
  /* Make sure the descriptor type matches */
  ((usb.request[3] != usb.data[1]) ||
  /* Device descriptor */
  ((usb.request[3] == 1) && ((usb.data[0] != 18)
      || (usb.actual_length != 18))) ||
  /* Configuration descriptor */
  ((usb.request[3] == 2) && ((usb.data[0] < 9)
      || (usb.actual_length < 9))) ||
  /* String descriptor */
  ((usb.request[4] == 3) && ((usb.data[0] < 4)
      || (usb.actual_length < 4))))))
```

LBM: USB Security

- **Defending against BadUSB**

```
((usb.pipe == 1) && /* INT (Keystroke) */
 ((usb.manufacturer != "X") ||
  (usb.product != "Y") ||
  (usb.serial != "Z") ||
  (usb.plugtime != 12345)))
```

- **Securing charging**

```
((usb.busnum == 1) && (usb.portnum == 1))
```

LBM: Bluetooth Security

- **Defending against BlueBorne**

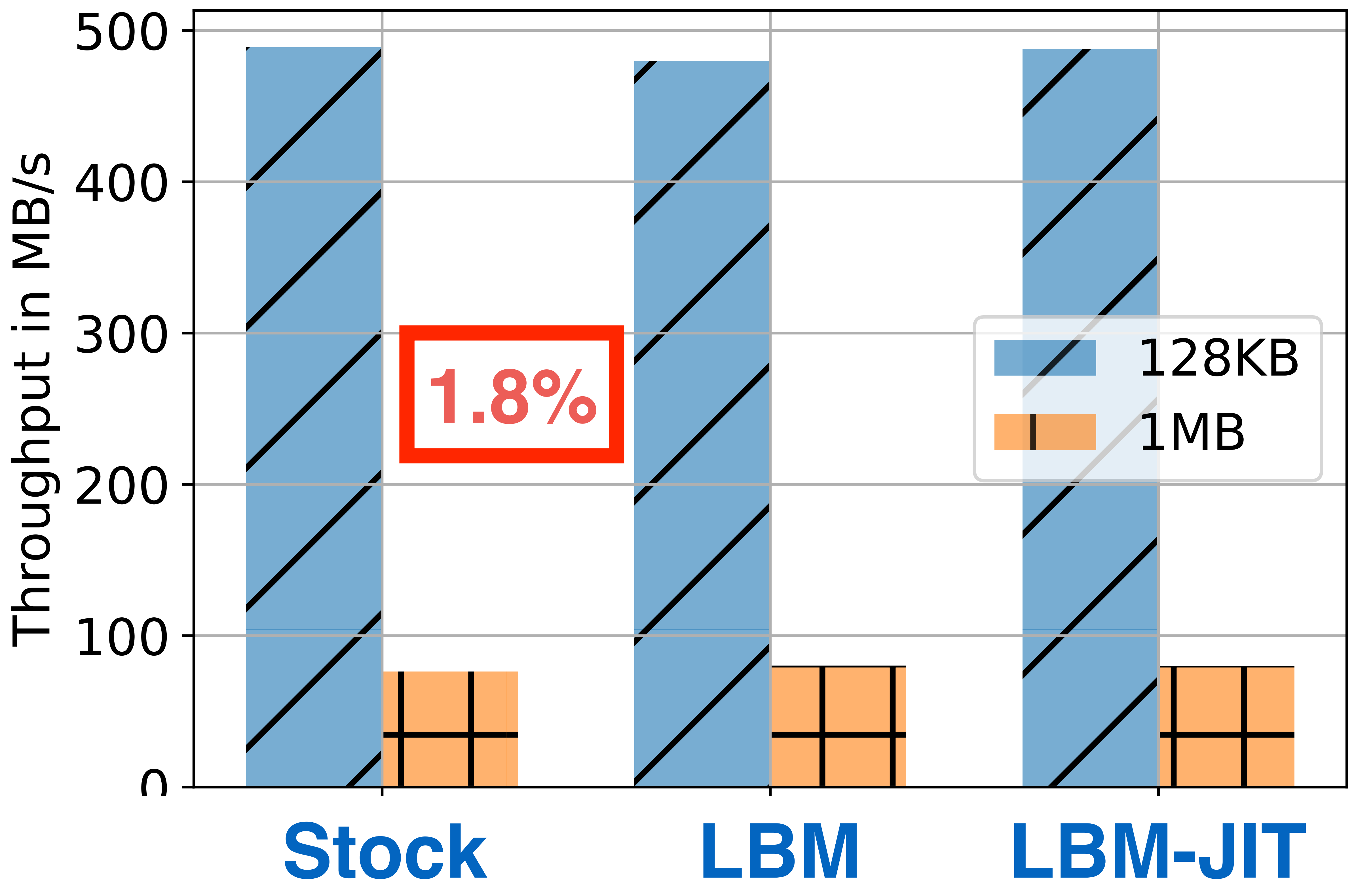
```
((bt.l2cap.cid == 0x1) && /* L2CAP Signaling */
 /* Configuration Response */
 (bt.l2cap.sig.cmd.code == 0x5) &&
 (bt.l2cap.sig.cmd.len >= 66))
```

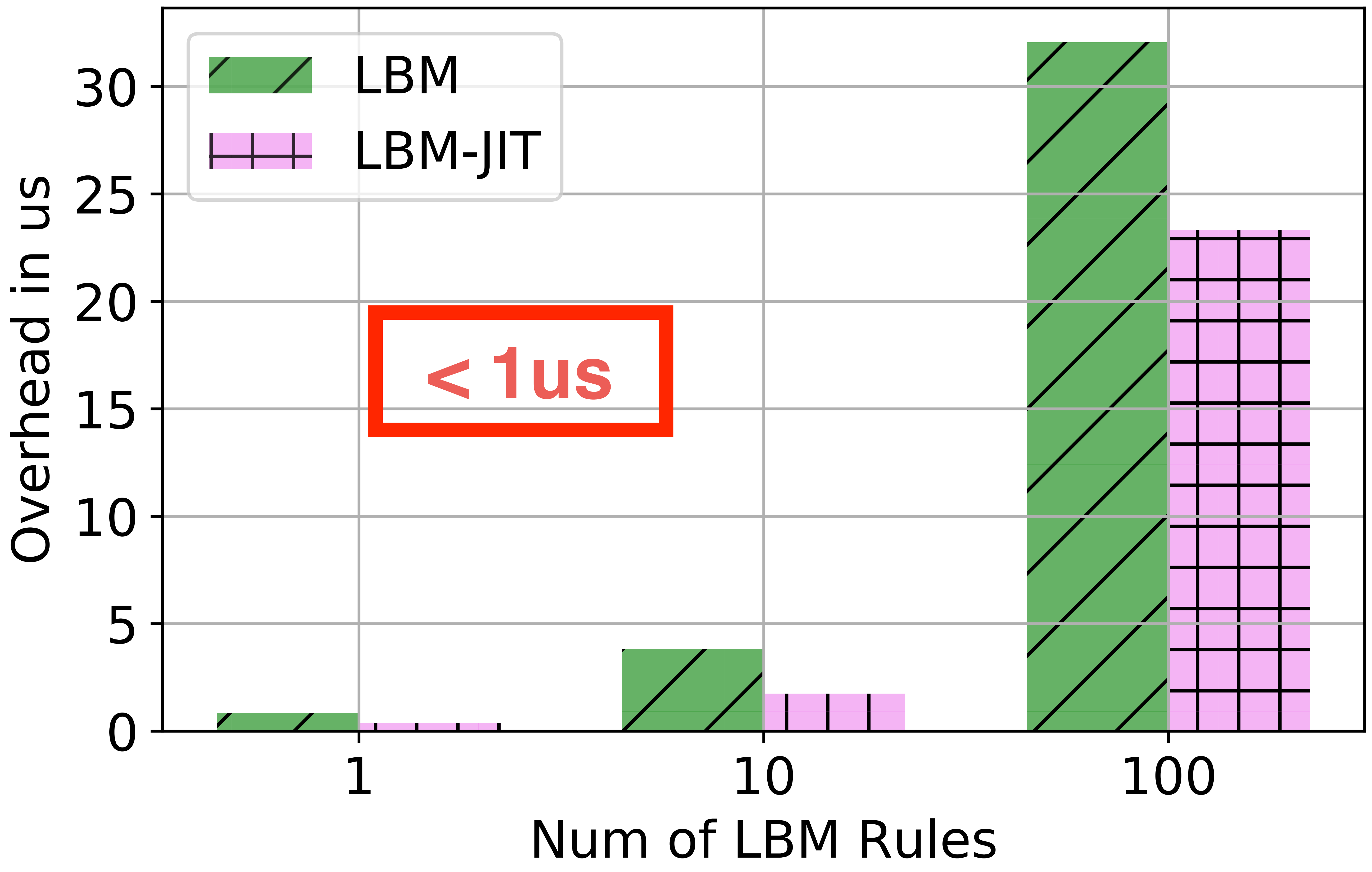
- **Defending against BleedingBit**

```
((bt.hci.conn == 1) && /* A link exists */
 (bt.hci.conn.type == 0x80)) /* BLE link */
```

Dynamic Kernel Patching

LBM: Benchmarks





LBM: Discussion

- BPF memory write
- LLVM support
- Stateless vs. Stateful policy
- DMA-oriented protocols

Conclusion

- Linux (e)BPF Module
- USB, Bluetooth, NFC
- Effectiveness and Minimum Overhead

https://github.com/fics/lbm

Q&A

https://davejingtian.org

Thanks!

Malicious Peripherals

What about wireless peripherals?

5.3 Billion Devices Affected

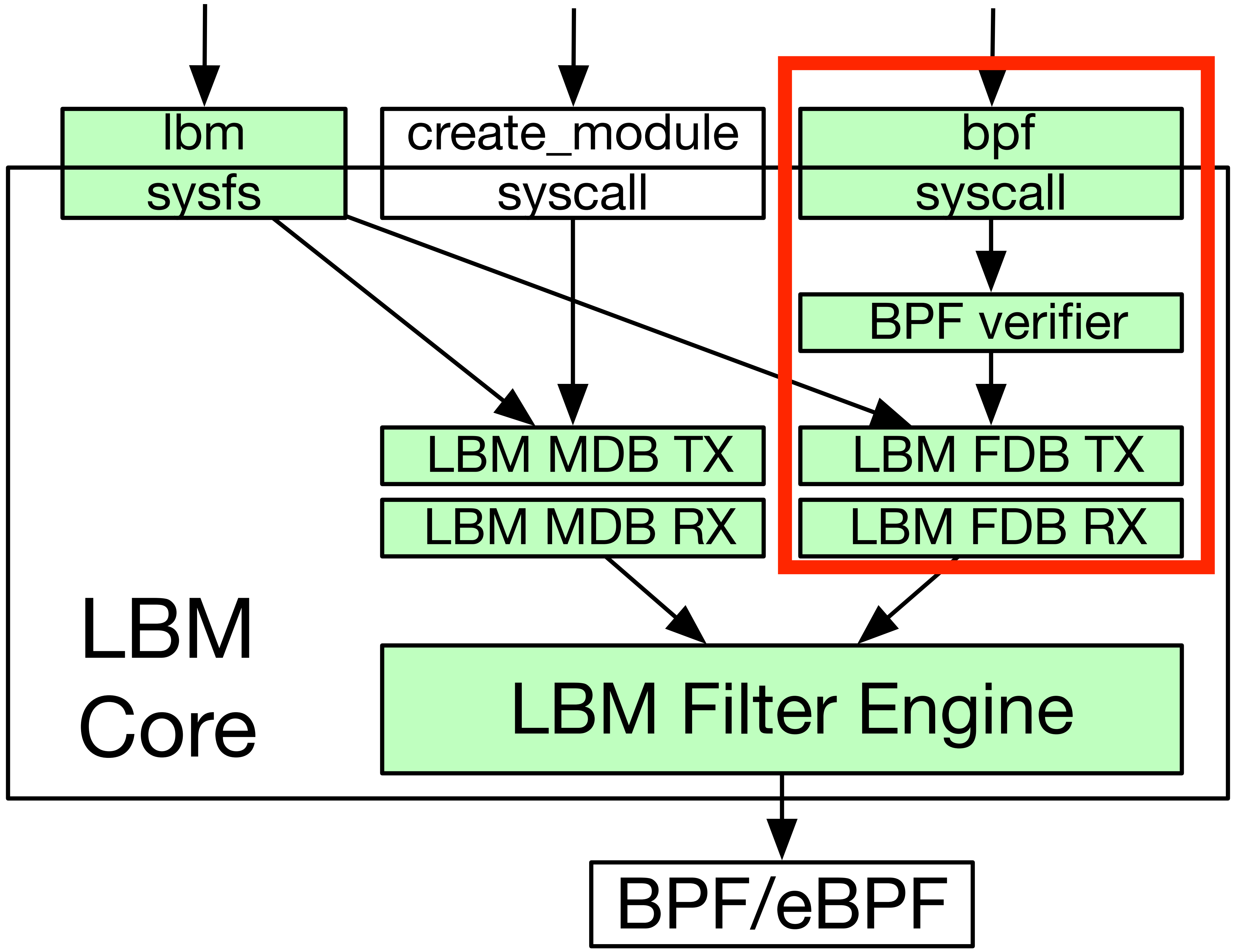
0day attacks over NFC!

NFC

```
From: Suren Baghdasaryan <surenb@google.com>

When handling SHDLC I-Frame commands "pipe" field used for indexing
into an array should be checked before usage. If left unchecked it
might access memory outside of the array of size NFC_HCI_MAX_PIPES(127).

cc: Stable <stable@vger.kernel.org>
Signed-off-by: Suren Baghdasaryan <surenb@google.com>
Signed-off-by: Amit Pundir <amit.pundir@linaro.org>
Reviewed-by: Andy Shevchenko <andriy.shevchenko@linux.intel.com>
```

LBM: Core Framework

- **An eBPF client**
  - LBM filter = eBPF program
- **Load LBM filters**
  - Subsystem / Path
- **Verify LBM filters**
  - Subsystem / No memory write
- **Store/Manage/Run LBM filters**
  - SysFS (/sys/fs/bpf, /sys/kernel/security/lbm)



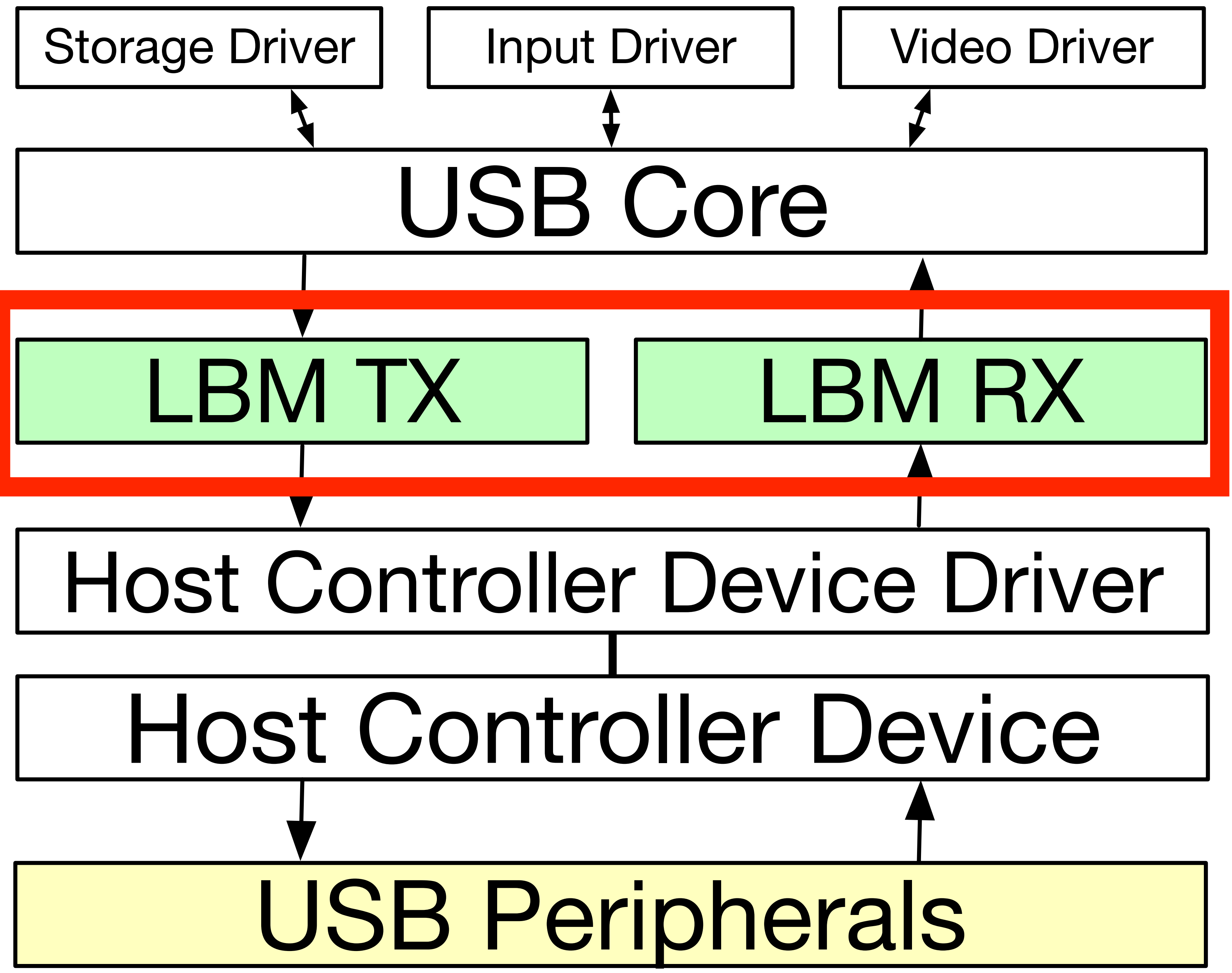
LBM: USB

- **LBM hooks**
- 34 protocol fields
- 31 BPF helpers
- 621 LoC

```
lbm_filter_pkt(LBM_SUBSYS_INDEX_USB, LBM_DIR_TX,
      (void *)urb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_USB, LBM_DIR_RX,
      (void *)urb);
```

```
"devnum" :
"bcdUSB" :
"bDeviceClass" :
"bDeviceSubClass" :
"bDeviceProtocol" :
"bMaxPacketSize0" :
"idVendor" :
"idProduct" :
"bcdDevice" :
"iManufacturer" :
"iProduct" :
"iSerialNumber" :
"bNumConfigurations" :
```

```
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_devnum"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bcdUSB"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bDeviceClass"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bDeviceSubClass"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bDeviceProtocol"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bMaxPacketSize0"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_idVendor"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_idProduct"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bcdDevice"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_iManufacturer"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_iProduct"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_iSerialNumber"),
SymbolHelper(ty=Type.TY_INT_32, name="lbm_usb_get_bNumConfigurations")
```

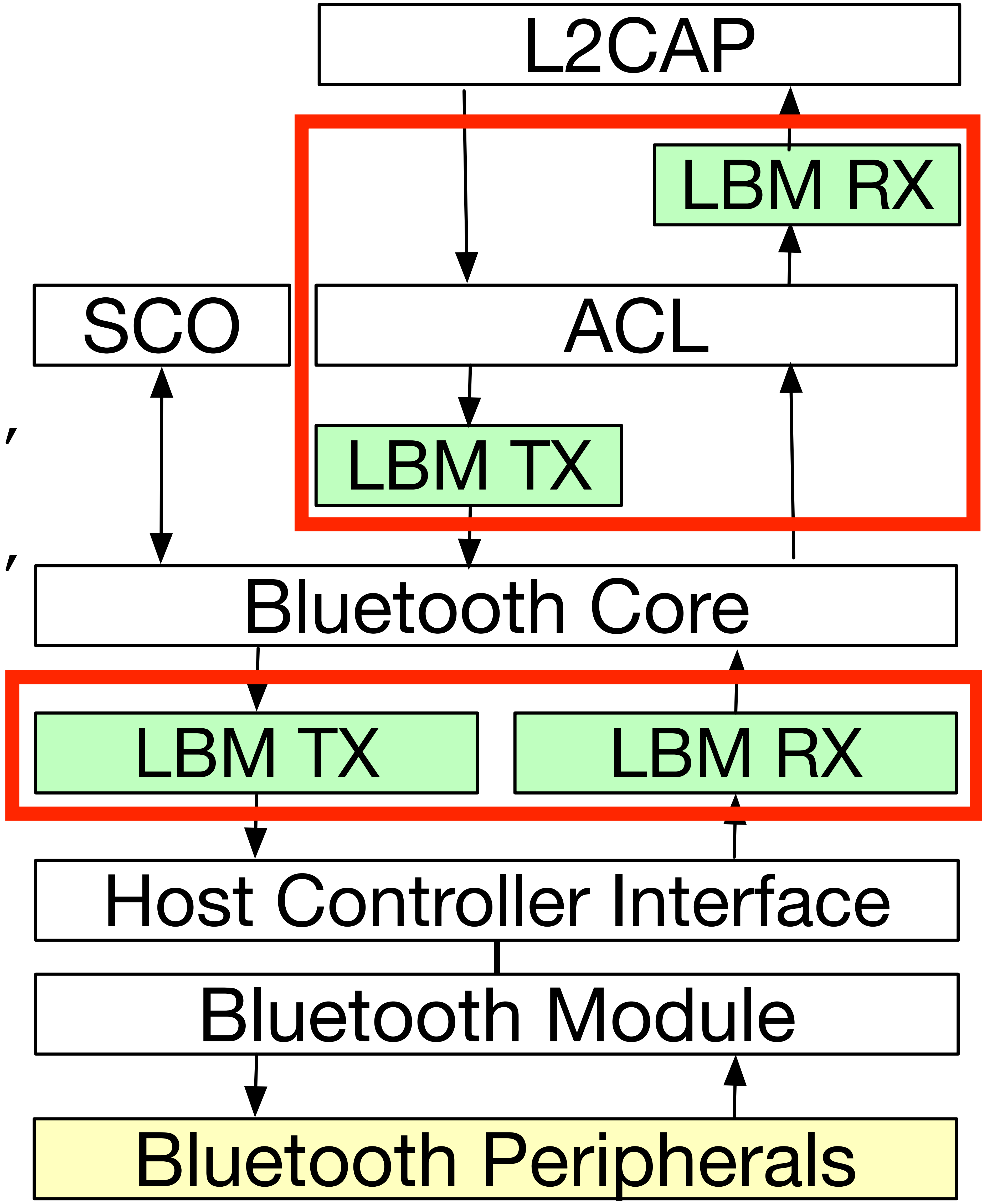


LBM: Bluetooth

- **LBM hooks**

```
lbm_filter_pkt(LBM_SUBSYS_INDEX_BLUETOOTH,
     LBM_DIR_TX, (void *)skb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_BLUETOOTH,
     LBM_DIR_RX, (void *)skb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_BLUETOOTH_L2CAP,
     LBM_DIR_TX, (void *)skb);
lbm_filter_pkt(LBM_SUBSYS_INDEX_BLUETOOTH_L2CAP,
     LBM_DIR_RX, (void *)skb);
```

- **HCI/L2CAP**
  - 30/28 protocol fields
  - 29/27 BPF helpers
  - 683/744 LoC



LBM: Protocol Stack Protection

```
/* HCI-CMD */
((bt.hci.type == 1) && (bt.hci.len < 3)) ||
 /* HCI-ACL */
 ((bt.hci.type == 2) && (bt.hci.len < 4)) ||
 /* HCI-SCO */
 ((bt.hci.type == 3) && (bt.hci.len < 3)) ||
 /* HCI-EVT */
 ((bt.hci.type == 4) && (bt.hci.len < 2)))
```

LBM: Filter DSL

usb.idVendor == 0x413c &&
usb.idProduct == 0x3010

Intermediate Representation

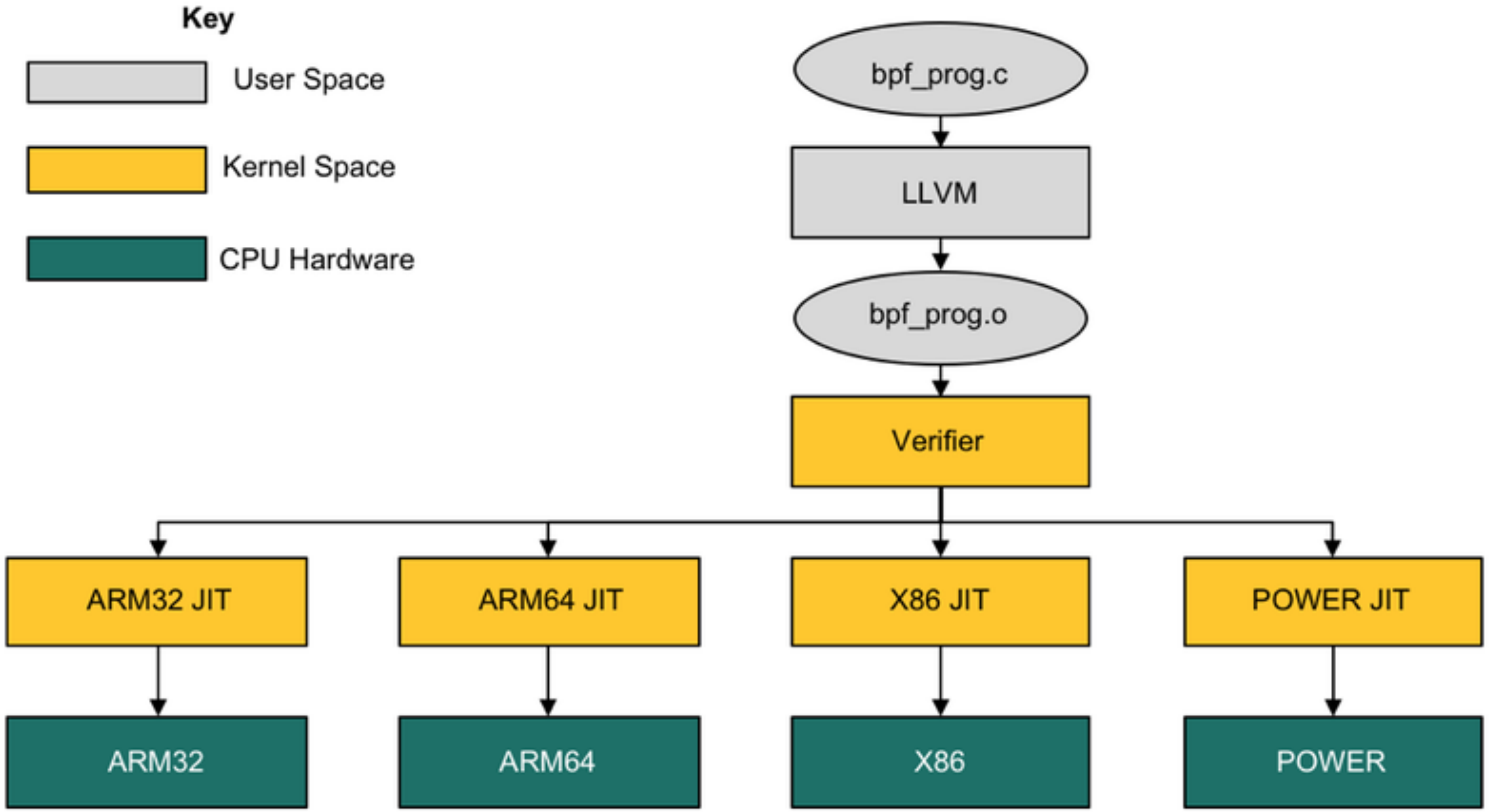
```
0: t1 := call(lbm_usb_get_idVendor)
1: t0 := binop(EQ, t1, 16700)
2: t3 := call(lbm_usb_get_idProduct)
3: t2 := binop(EQ, t3, 12304)
4: t4 := binop(AND, t0, t2)
```

eBPF Assembly

```
LSTART:
          MOV64_REG(REG_9, REG_1)
          MOV64_REG(REG_1, REG_9)
          CALL_FUNC(FUNC_lbm_usb_get_idVendor)
          MOV64_REG(REG_1, REG_0)
          MOV64_IMM(REG_6, 1)
          JMP_IMM(JEQ, REG_1, 16700, L1_)
          MOV64_IMM(REG_6, 0)
L1_:
          MOV64_REG(REG_1, REG_9)
          CALL_FUNC(FUNC_lbm_usb_get_idProduct)
          MOV64_REG(REG_2, REG_0)
          MOV64_IMM(REG_3, 1)
          JMP_IMM(JEQ, REG_2, 12304, L2_)
          MOV64_IMM(REG_3, 0)
L2_:
          JMP_IMM(JEQ, REG_6, 0, L3_)
          JMP_IMM(JEQ, REG_3, 0, L3_)
          MOV64_IMM(REG_4, 1)
          JMP_A(L4_)
L3_:      MOV64_IMM(REG_4, 0)
L4_:
          JMP_IMM(JNE, REG_4, 0, L5_)
L6_:      MOV64_IMM(REG_0, 0)
          EXIT_INSN()
L5_:      MOV64_IMM(REG_0, 1)
LEND:     EXIT_INSN()
```

LBM: Extended BPF (eBPF)

- 64-bit BPF architecture
- BPF helpers
- BPF maps
- BPF verifier
- BPF JIT



https://www.netronome.com/blog/bpf-ebpf-xdp-and-bpfilter-what-are-these-things-and-what-do-they-mean-enterprise/

What Went Wrong?

- **No Authorization!**
  - Devices are trusted by default
  - Devices can request any functionality
- **No Integrity!**
  - Device firmware can be hacked
  - Firmware modifications are invisible to host
- **No Authentication!**
  - Devices have no trustworthy notion of identity

#1: Peripheral Diversity

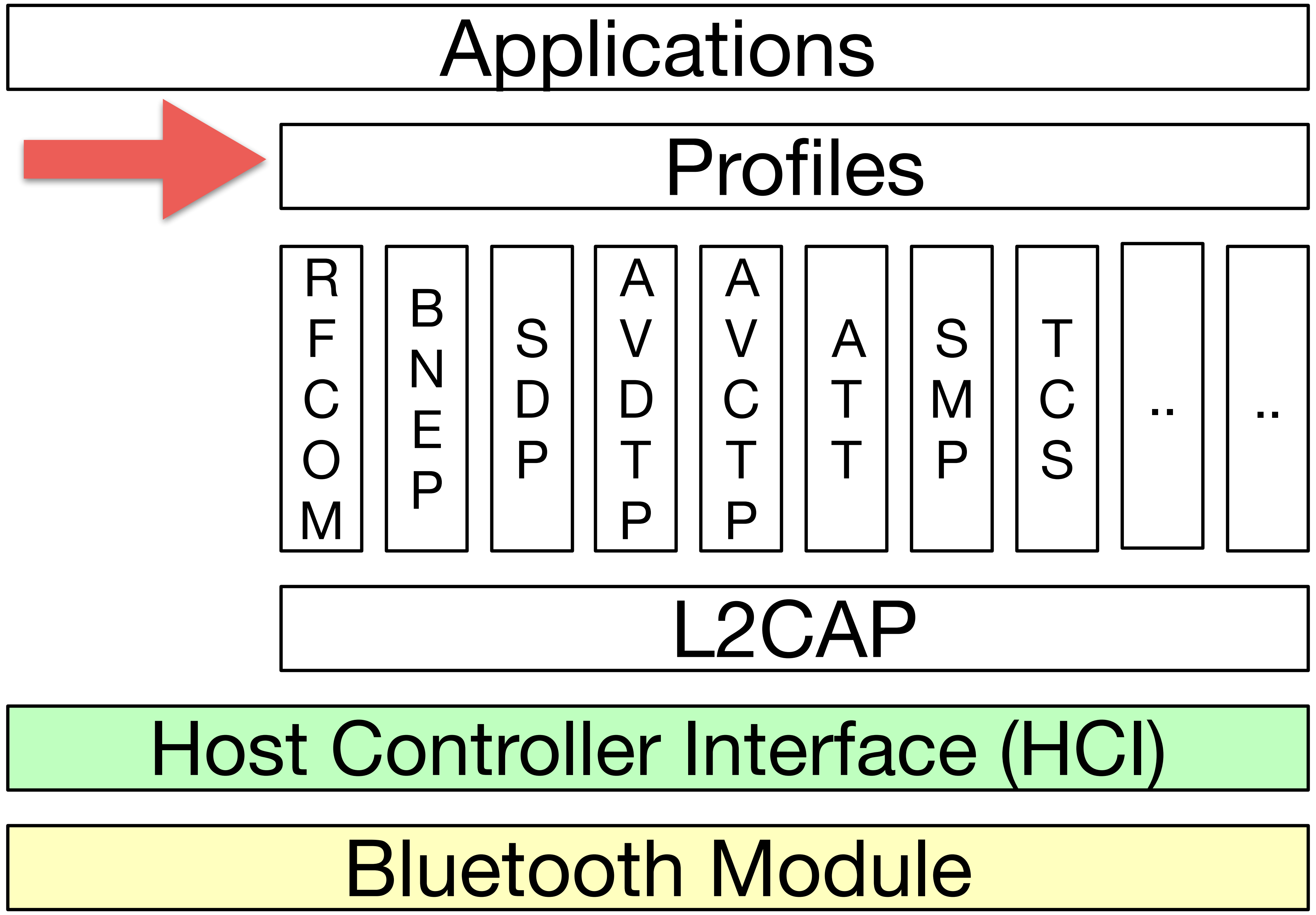
Q: How do we support all peripherals??

- ***USBFILTER*** *(USENIX Security'16)*
- ***Bluetooth-FW, NFC-FW, X-FW?***

A: Peripheral Agnostic -

- Separation between mechanism and implementation - **hooks**
- Separation between mechanism and policy - **generic packet filter**

#2: Hook Placement



Q: Where to place hooks??

- ***High layer?***
- ***Low layer?***
- ***In between?***

A: Reference Monitor Concept -

- Complete mediation
- Tamperproof / Verifiability

#3: Generic Packet Filter

- **Berkeley Packet Filter (BPF)**
- High-performance (IP) packet filtering
- In-kernel virtual machine (RISC)
- Just-In-Time (JIT) compilation
- Backend of tcpdump

Q: What is generic packet filter??

A: BPF for peripherals!

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh        [12]
(001) jeq        #0x800           jt 2    jf 18
(002) ld         [26]
(003) jeq        #0x7f000001      jt 6    jf 4
(004) ld         [30]
(005) jeq        #0x7f000001      jt 6    jf 18
(006) ldb        [23]
(007) jeq        #0x84            jt 10   jf 8
(008) jeq        #0x6             jt 10   jf 9
(009) jeq        #0x11            jt 10   jf 18
(010) ldh        [20]
(011) jset       #0x1fff          jt 18   jf 12
(012) ldxb       4*([14]&0xf)
```

#4: Programmability vs. Usability

```
iptables -A INPUT -s 15.15.15.51 -j DROP
iptables -A OUTPUT -p tcp --sport 22 -m
conntrack --ctstate ESTABLISHED -j ACCEPT
```

```
#include <linux/bpf.h>
#ifndef __section
# define __section(NAME)                  \
   __attribute__((section(NAME), used))
#endif
static int foo(void)
{
    return XDP_DROP;
}
__section("prog")
int xdp_drop(struct xdp_md *ctx)
{
    return foo();
}
char __license[] __section("license") = "GPL";
```

```
      ldh     [12]
      jeq     #ETHERTYPE_IP, l1, l2
      l1:     ret      #TRUE
      l2:     ret      #0
```

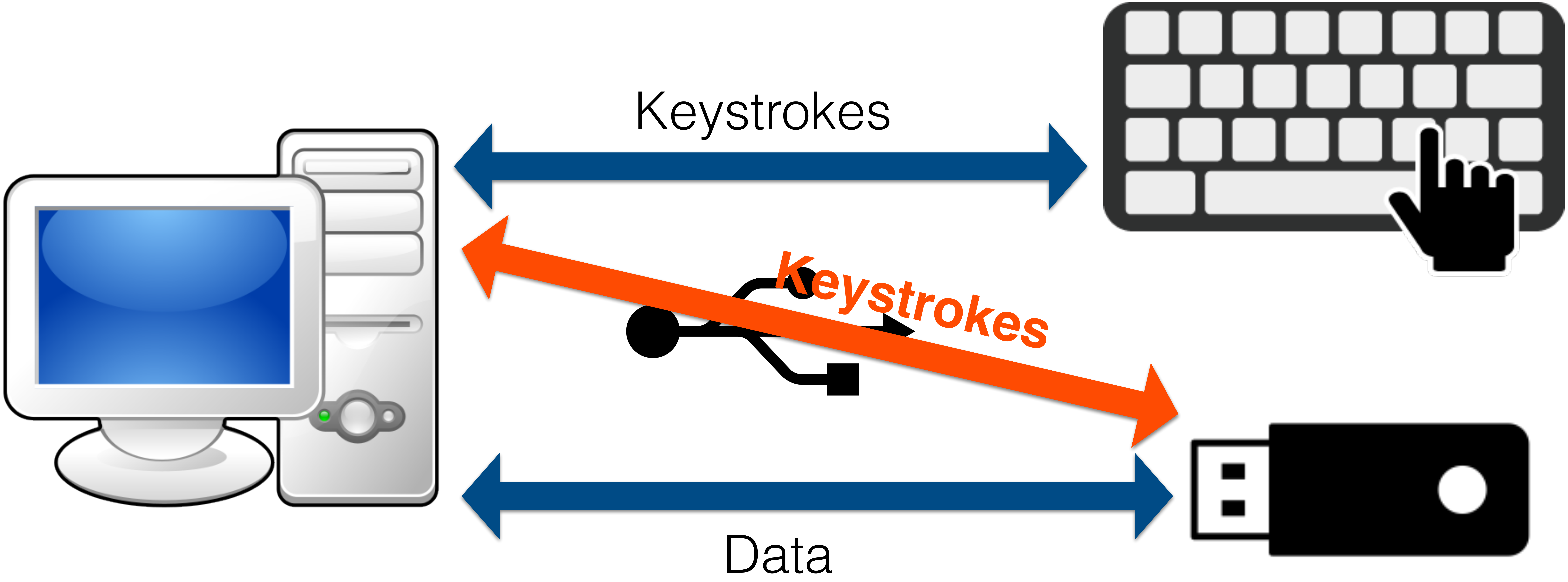
Q: Who writes filtering rules??

- ***End users?***
- ***Sysadmins?***
- ***Developers?***

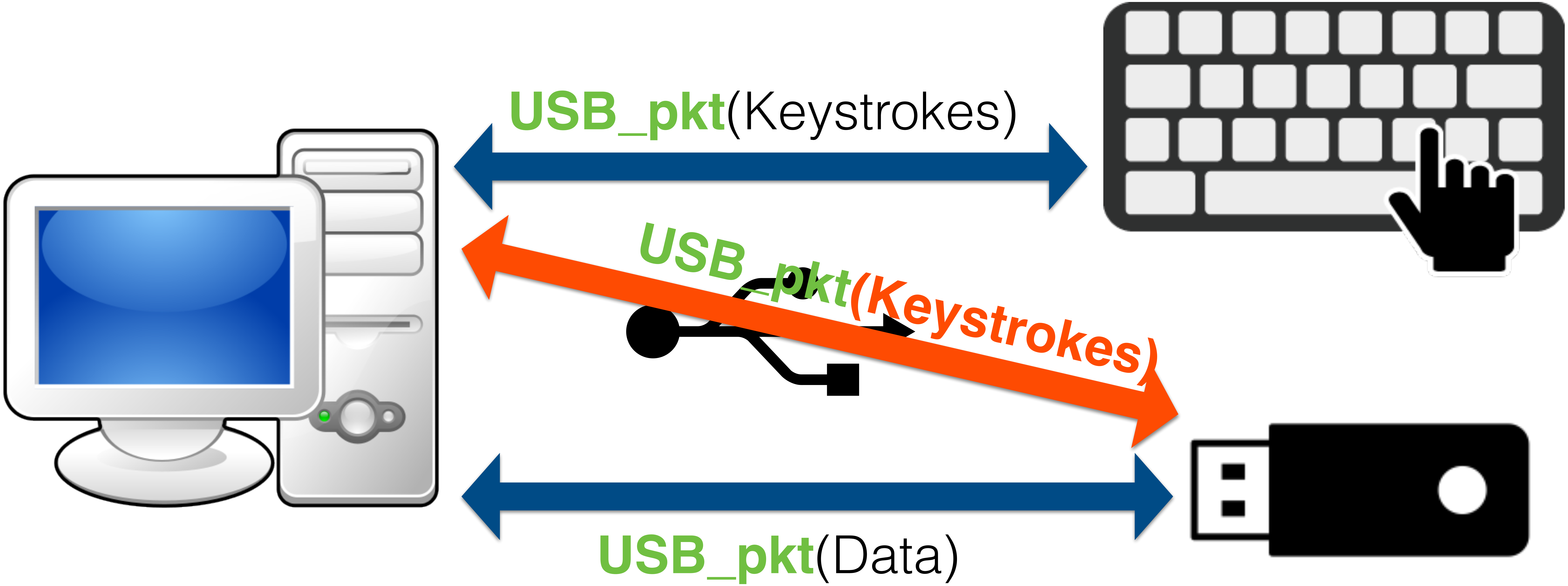
A: Everyone! -

- Users not enemy (Doh!)
- Peripheral agnostic (Again!)

BadUSB Attacks



BadUSB Attacks



BlueBorne Attacks

```
switch (result) {
        case L2CAP_CONF_SUCCESS:
                l2cap_conf_rfc_get(chan, rsp->data, len);
                clear_bit(CONF_REM_CONF_PEND, &chan->conf_state);
                break;

        case L2CAP_CONF_PENDING:
                set_bit(CONF_REM_CONF_PEND, &chan->conf_state);

                if (test_bit(CONF_LOC_CONF_PEND, &chan->conf_state)) {
                        char buf[64];

                        len = l2cap_parse_conf_rsp(chan, rsp->data,
                                                   buf, &result);
                        if (len < 0) {
                                l2cap_send_disconn_req(chan, ECONNRES
                                goto done;
                        }
```

buf length (64) is NOT passed here!