FuzzBT

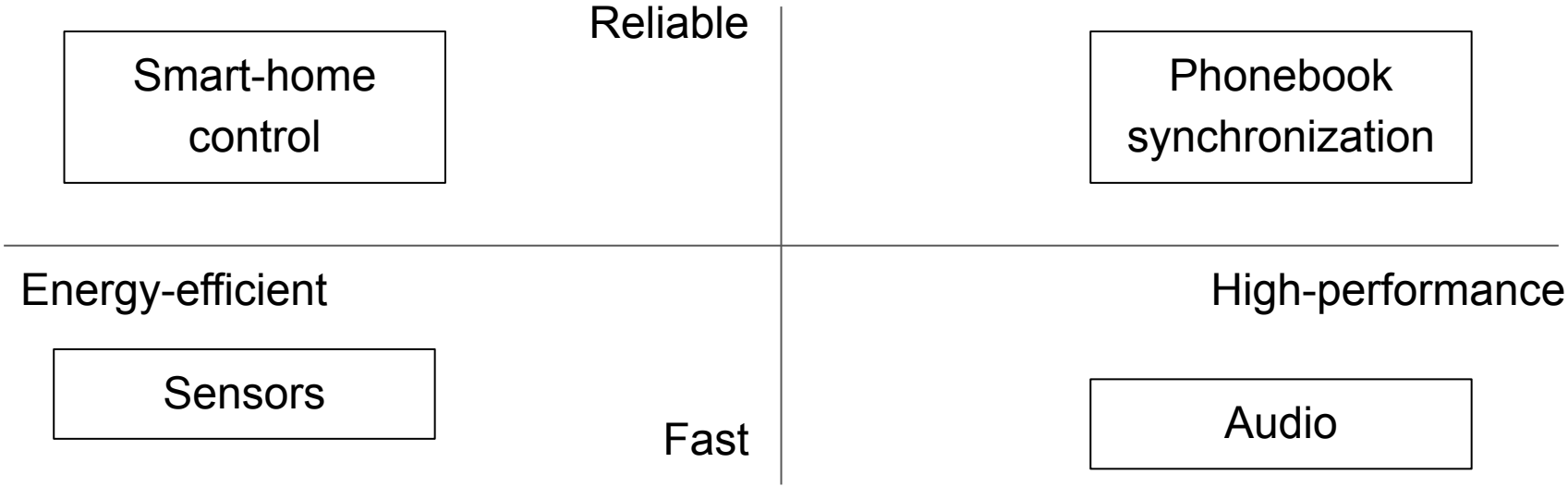
Holistic-State-Guided Fuzzing for Bluetooth Host Stack in Kernels

Sungwoo Kim, Hui Peng, Imtiaz Karim, Ruoyu Wu
Jianliang Wu, Elisa Bertino, Mathias Payer, Dave Tian

Key points

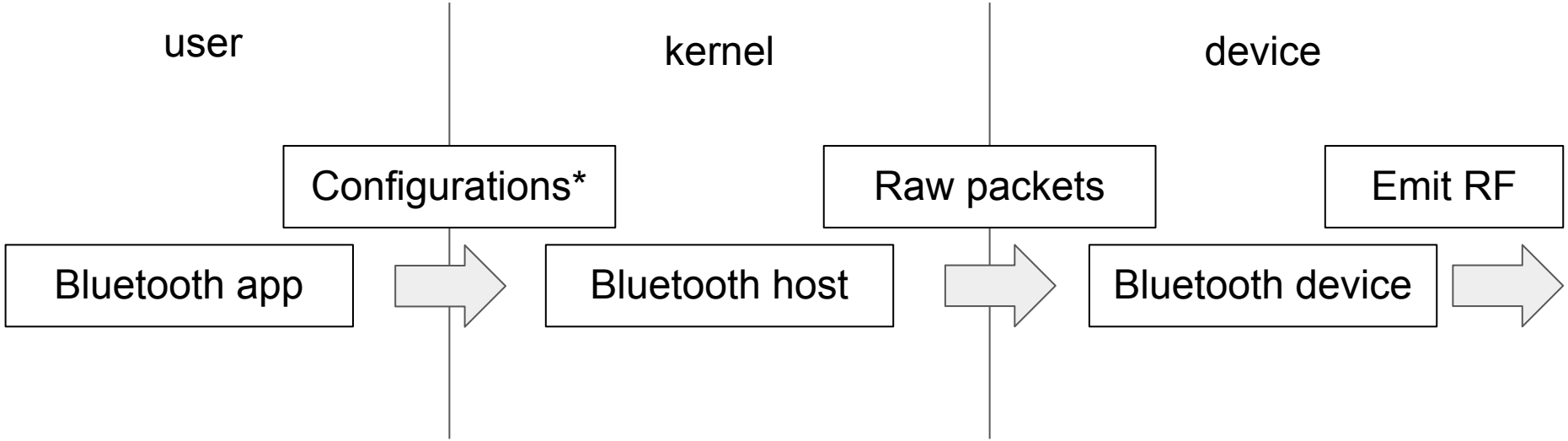
- Bluetooth designs (5 mins)
- Why we need another Bluetooth fuzzer (3 mins)
- Design & Evaluation of FuzzBT (7 mins)

Bluetooth is universal, but how?



How can one protocol suite satisfy conflicting requirements?

How Bluetooth can be universal



A simple solution to support conflicting requirements: implement each feature and accumulate them.

*retransmission policy, security policy, power control, maximum transfer units, etc.

How Bluetooth can be universal

```
switch(configuration) {
  case BASIC:              do_basic(skb);                    break;
  case ERTM:               do_enhanced_retransmission(skb);  break;
  case STREAMING:          do_streaming(skb);                break;
  case LE_FLOWCTL:         do_flow_control(skb);             break;
}
```

The host stack implements each requirement.

→ The host stack is sum of different protocol state machines.

Research Gaps

Q. How many configurations are there?

A. 1800+ configurations, in the Linux kernel.

Q. Is there a previous work covering configurations?

A. No.

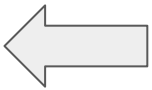
Goal: Finding vulnerabilities across the Bluetooth host stack configurations

Concrete example

```
1   /* HCI layer (lower) */
2   u8 hci_cc_le_read_buffer_size()
3   {
4     /* cross-protocol state propagation */
5     hdev->le_mtu = __le16_to_cpu(rp->le_mtu);
6   }
7
8   /* L2CAP layer (higher) */
9   struct l2cap_conn *l2cap_conn_add()
10  {
11    conn->mtu = hdev->le_mtu;
12  }
13
14  void l2cap_le_flowctl_init()
15  {
16    chan->mps = min(chan->imtu, conn->mtu - L2CAP_HDR_SIZE);
17    /* error: div-by-zero. */
18    chan->rx_credits =
19      (chan->imtu / chan->mps) + 1;
20  }
```

Concrete example

```
1   /* HCI layer (lower) */
2   u8 hci_cc_le_read_buffer_size()
3   {
4     /* cross-protocol state propagation */
5     hdev->le_mtu = __le16_to_cpu(rp->le_mtu);
6   }
7
8   /* L2CAP layer (higher) */
9   struct l2cap_conn *l2cap_conn_add()
10  {
11    conn->mtu = hdev->le_mtu;
12  }
13
14  void l2cap_le_flowctl_init()
15  {
16    chan->mps = min(chan->imtu, conn->mtu - L2CAP_HDR_SIZE);
17    /* error: div-by-zero. */
18    chan->rx_credits =
19      (chan->imtu / chan->mps) + 1;
20  }
```



bug (div-by-zero)

Concrete example

```
1   /* HCI layer (lower) */
2   u8 hci_cc_le_read_buffer_size()
3   {
4     /* cross-protocol state propagation */
5     hdev->le_mtu = __le16_to_cpu(rp->le_mtu);
6   }
7
8   /* L2CAP layer (higher) */
9   struct l2cap_conn *l2cap_conn_add()
10  {
11    conn->mtu = hdev->le_mtu;
12  }
13
14  void l2cap_le_flowctl_init()
15  {
16    chan->mps = min(chan->imtu, conn->mtu - L2CAP_HDR_SIZE);
17    /* error: div-by-zero. */
18    chan->rx_credits =
19      (chan->imtu / chan->mps) + 1;
20  }
```

requiring a specific config

bug (div-by-zero)

Concrete example

```
1   /* HCI layer (lower) */
2   u8 hci_cc_le_read_buffer_size()
3   {
4     /* cross-protocol state propagation */
5     hdev->le_mtu = __le16_to_cpu(rp->le_mtu);
6   }
7
8   /* L2CAP layer (higher) */
9   struct l2cap_conn *l2cap_conn_add()
10  {
11    conn->mtu = hdev->le_mtu;
12  }
13
14  void l2cap_le_flowctl_init()
15  {
16    chan->mps = min(chan->imtu, conn->mtu - L2CAP_HDR_SIZE);
17    /* error: div-by-zero. */
18    chan->rx_credits =
19      (chan->imtu / chan->mps) + 1;
20  }
```

requiring multiple host stack protocols to trigger, rather than one protocol

requiring a specific config

bug (div-by-zero)

Design goals

- Effectively explore configurations

- Effectively test multiple protocols in the host stack

Recall: How Bluetooth can be universal

```
switch(configuration) {
  case BASIC:          do_basic(skb);                    break;
  case ERTM:           do_enhanced_retransmission(skb);  break;
  case STREAMING:      do_streaming(skb);                break;
  case LE_FLOWCTL:     do_flow_control(skb);             break;
}
```

Q. Can the corpus for BASIC will effectively cover ERTM?

Configuration exploration

	Baseline fuzzer (Syzkaller)	Fuzzing multiple configurations by reusing corpus.
Unique bug	19.04	18.02
Code coverage	1949.58	1930.71

A. No, simply reusing corpus across configurations degrades the fuzzing performance*

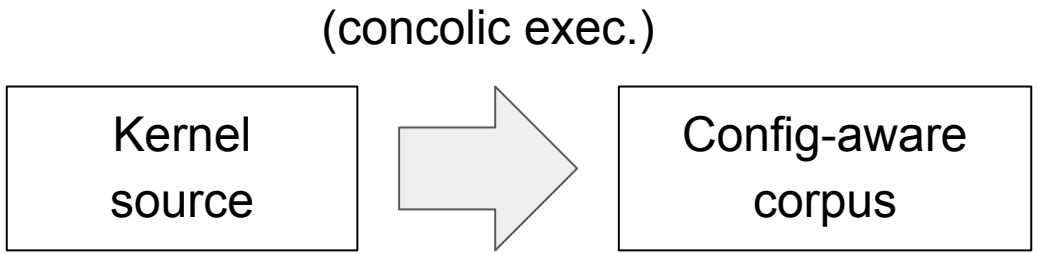
*On avaerage, N=20, fuzzed 50 hours for each. More details are in Table 7 (a)

Configuration exploration

Config-aware corpus

← We need this!

Configuration exploration



We perform concolic execution to extract config-aware fuzzing inputs.

*More technical details are in Section 5.1

Configuration exploration

	Baseline fuzzer (Syzkaller)	Fuzzing with config-aware corpus.
Unique bug	19.04	21.07
Code coverage	1949.58	1943.09

The config-aware corpus increased bug findings in our evaluation

*On avaerage, N=20, fuzzed 50 hours for each. More details are in Table 7 (a)

Configuration exploration

	Baseline fuzzer (Syzkaller)	Fuzzing with config-aware corpus.
Unique bug	19.04	21.07
Code coverage	1949.58	1943.09

The config-aware corpus increased bug findings in our evaluation, **but it did not improve code coverage**

*On avaerage, N=20, fuzzed 50 hours for each. More details are in Table 7 (a)

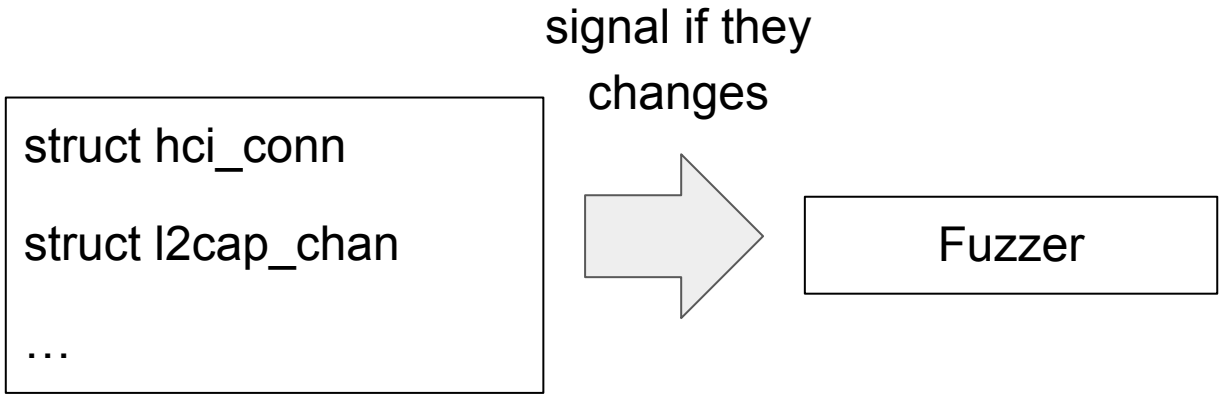
Recall: How Bluetooth can be universal

```
switch(configuration) {
  case BASIC:              do_basic(skb);                    break;
  case ERTM:               do_enhanced_retransmission(skb);  break;
  case STREAMING:          do_streaming(skb);                break;
  case LE_FLOWCTL:         do_flow_control(skb);             break;
}
```

Each configuration has its own protocol state machine

→ State coverage will be effective

State feedback for multiple protocol fuzzing



We monitor state variables via LLVM pass

*More technical details are in Section 5.2

State feedback for multiple protocol fuzzing

	Baseline fuzzer (Syzkaller)	State-guided fuzzing
Unique bug	19.04	23.21
Code coverage	1949.58	2097.29

State-guidance increased fuzzing performance in our evaluation.

*On avaerage, N=20, fuzzed 50 hours for each. More details are in Table 7 (a)

Bug findings

- Found 18 bugs with 9 CVEs*

- 15 out of 18 are configuration-specific

*# of bugs != # of CVE. The Linux kernel assigns a CVE per a patch, not per bug

Bug findings

This bug crashes the kernel even before starting a fuzzer.

A specific Bluetooth configuration is required to trigger.

```
@ struct hci_xxx amp_xxx[] = {
          ..., // function pointers
+         {}
  };
```

Missing an indicator at the end of array modified a program counter.

*CVE-2023-28866, 1.5 years remained undiscovered

Conclusion

- Vulnerabilities under various Bluetooth configurations have remain unexplored.

- We discovered 18 bugs from various configurations.

- Exploring configurations was critical for finding Bluetooth vulnerabilities.

Thank you

Also special thanks to Luiz, Marcel, the maintainers of the Linux Bluetooth subsystem, and our sponsors.