L2Fuzz: Discovering Bluetooth L2CAP Vulnerabilities Using Stateful Fuzz Testing

52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'22)

Haram Park

Carlos Kayembe Nkuba

Seunghoon Woo

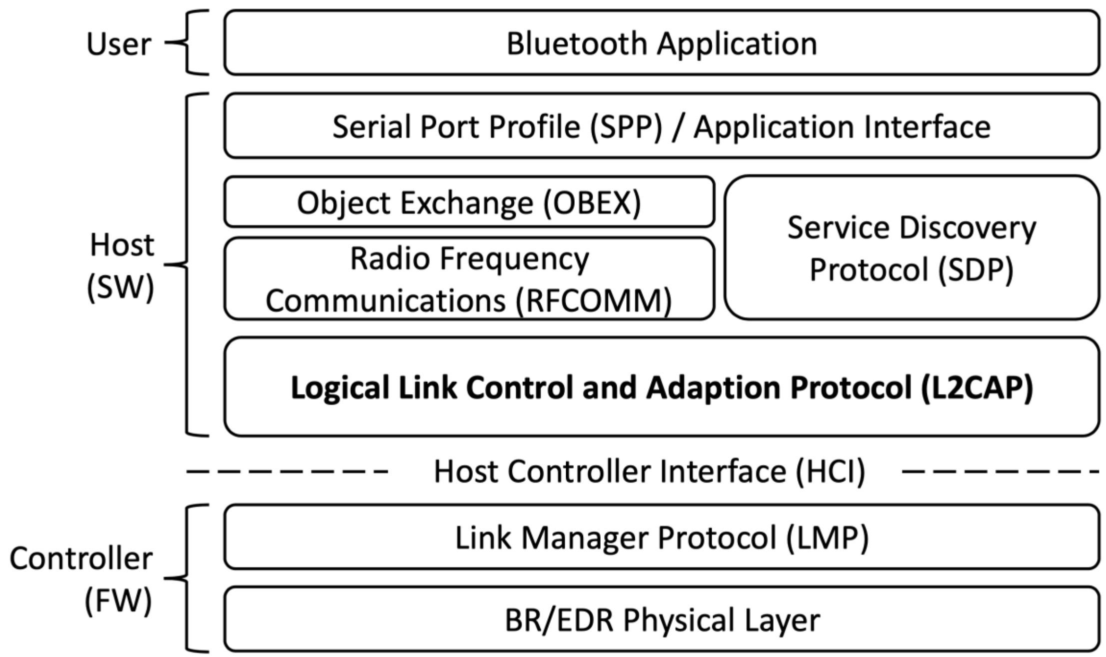
Heejo Lee

Computer & Communication Security Laboratory, Korea University

Background

- Bluetooth Basic Rate/Enhanced Data Rate (BT Classic)

1) Wireless communication technology which is adopted by billions of devices.
   → A vulnerability can attack billions of devices.
2) To use Bluetooth application, a L2CAP connection between devices is needed.
   → Using L2CAP vulnerability, critical attacks are possible.



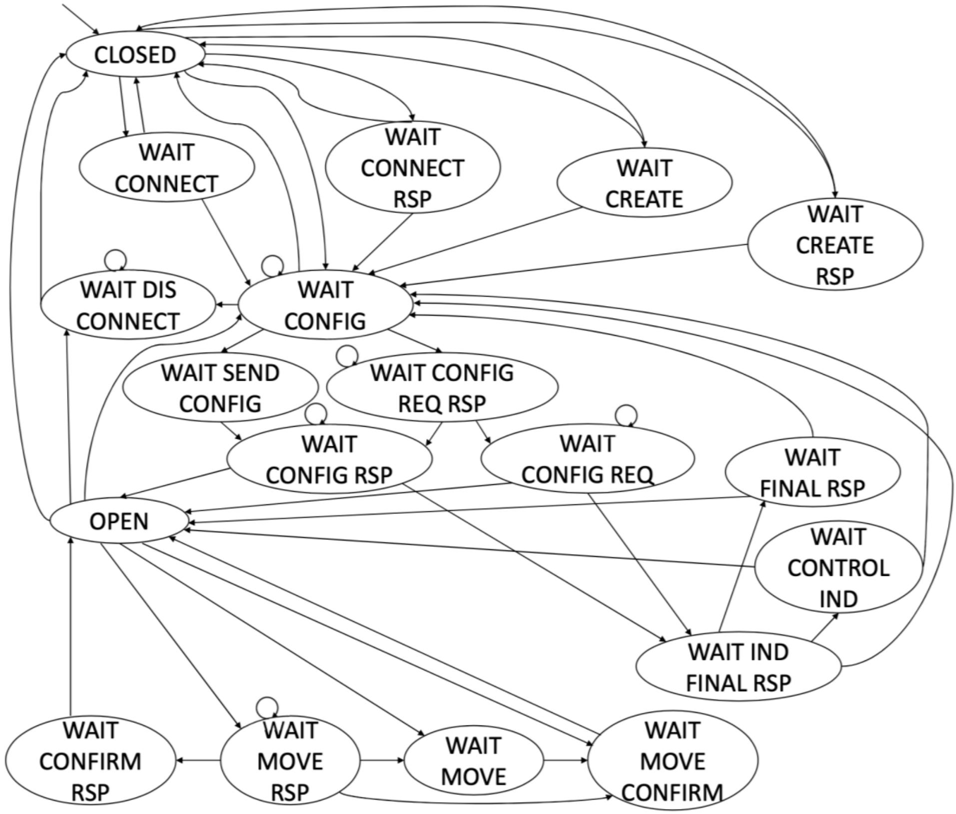
Challenge for fuzzing: Increasing the L2CAP state coverage

- **Bluetooth L2CAP follows a specific state machine.**

- Vulnerabilities are highly likely to occur in

1) the state transition process
2) the functions of each state

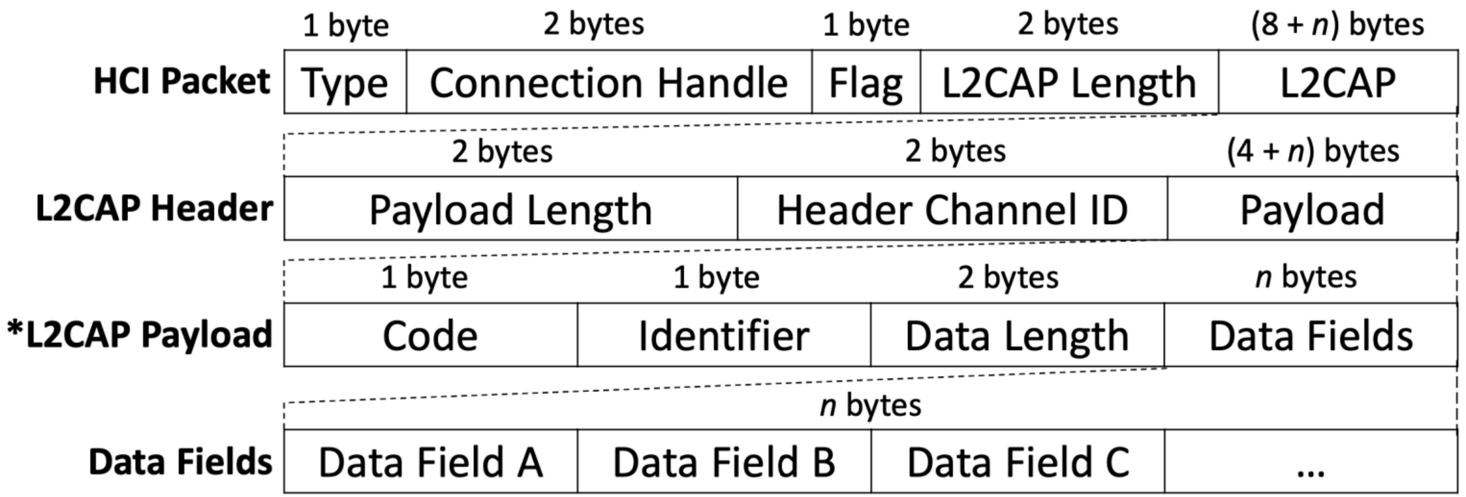
→ We need to <u>test as many states as possible.</u>



<Bluetooth 5.2 L2CAP state machine>

Challenge for fuzzing: Generating valid malformed packets

- **Payload can have multiple Data Fields depending on the command code.**



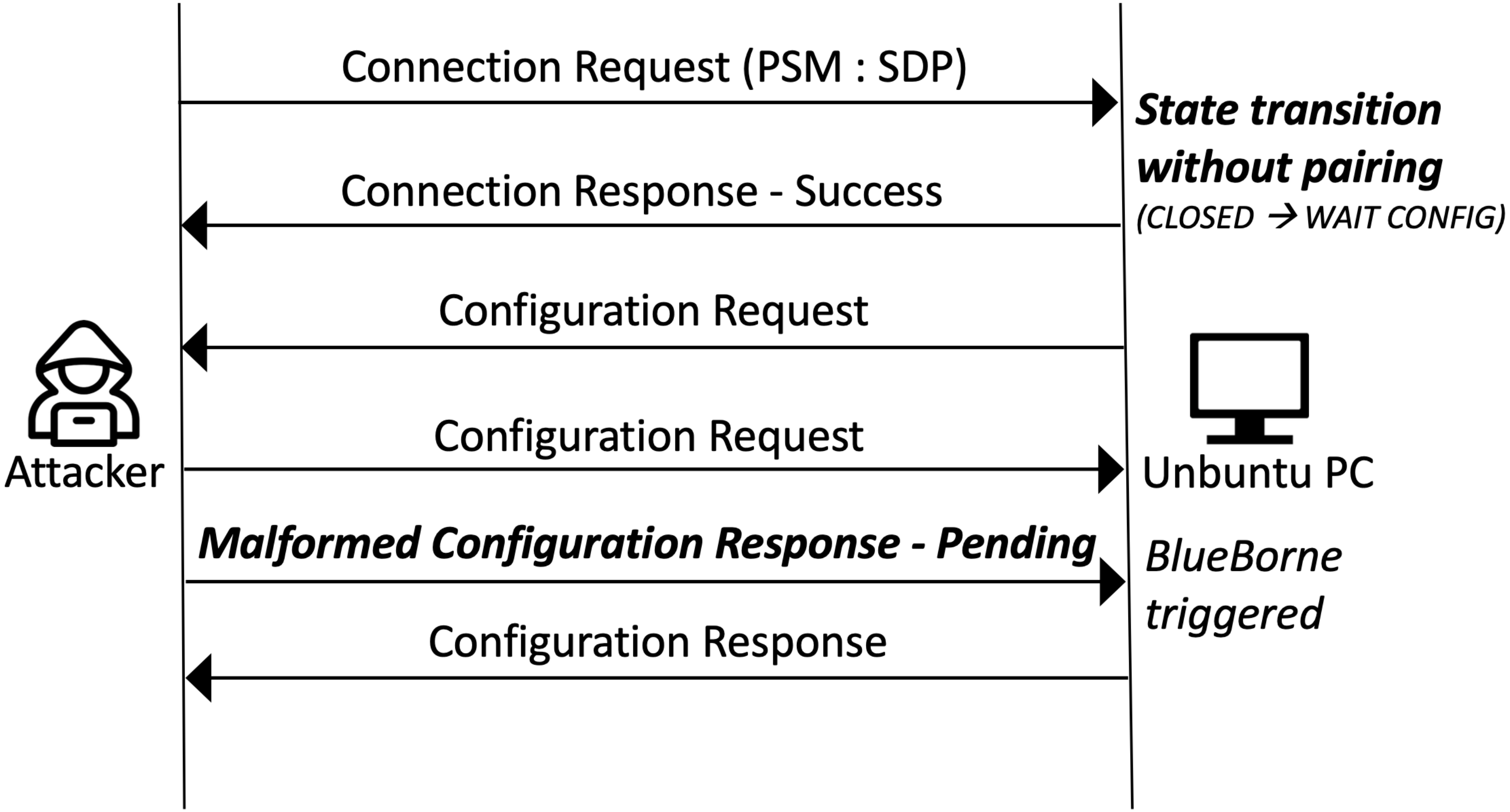
*L2CAP Payload can be up to 65,535 bytes.

- **Mutating any or all fields causes packet rejection by the target devices.**

→ <u>We need effective mutating to avoid packet rejection and discover the vulner abilities.</u>

Motivating Example

- BlueBorne Attack (CVE-2017-1000251)
  - RCE attack through L2CAP vulnerability.



L2Fuzz

- Stateful fuzzer for detecting Bluetooth L2CAP vulnerabilities

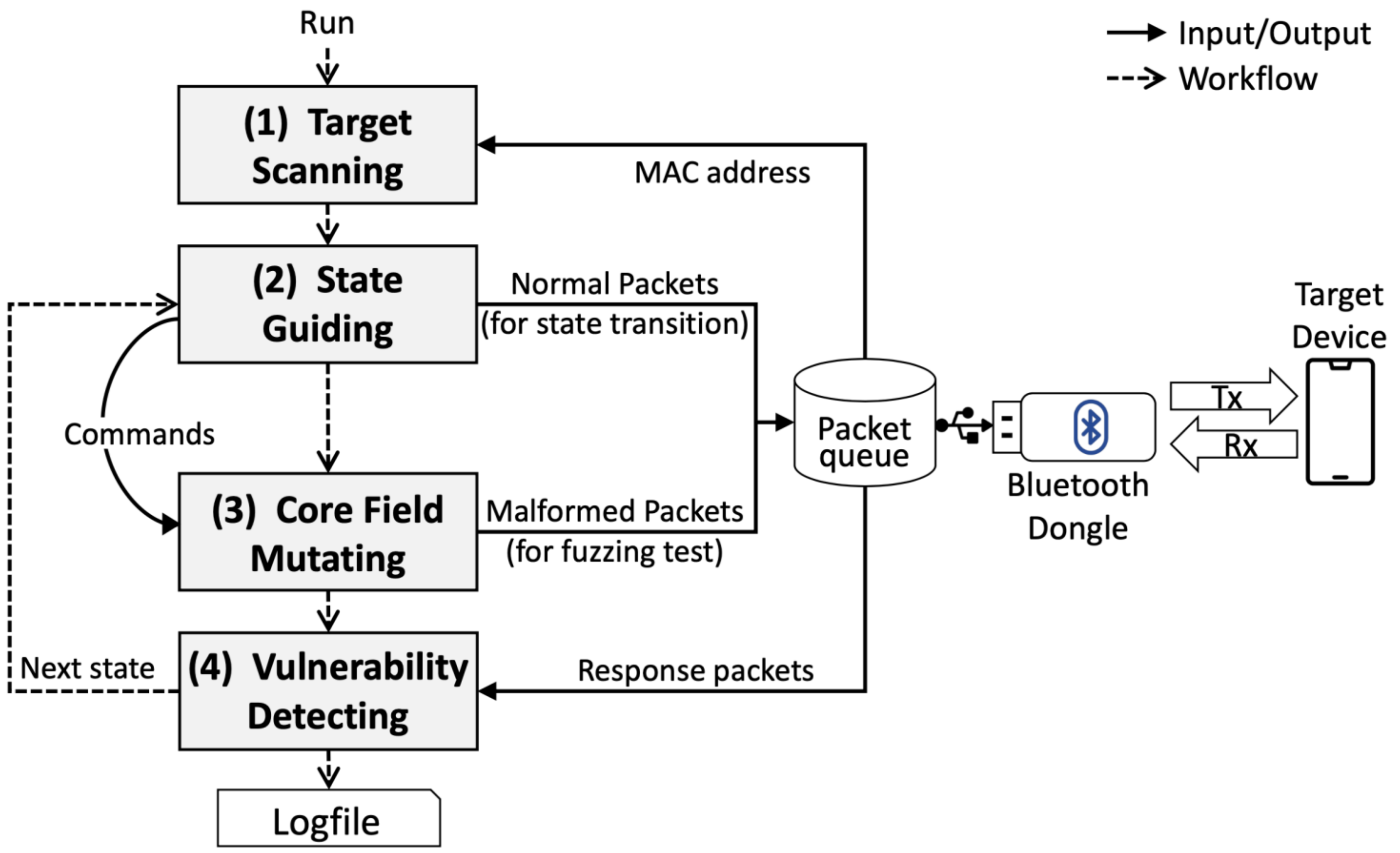
Key techniques

✓ **State Guiding**

- To increase state coverage

✓ **Core Field Mutating**

- To generate malformed packets that are less likely to be rejected



Process 1: Target Scanning

- **Scanning the target device's information**

1) MAC address : to establish L2CAP socket.

2) Service ports : to test the port that does not require pairing.

   a. attackers often exploit without pairing (*e.g.*, BlueBorne)

   b. fuzzing after pairing is meaningless (appropriate privilege escalation)

   c. for ports that require pairing, sending test packets without pairing causes the device to reject packets

Process 2: State Guiding

- **State Classification.**

1) Clustering states into "Job" based on the *event, functions* and *action*.

ex) WAIT CONNECT : Connection Request (*event*), Connection (*functions*), Connection Response (*action*)
WAIT CONNECT RSP : Connection Response (*event*), Connection (*functions*), Configuration Request (*action*)
WAIT CONNECT and WAIT CONNECT RSP → states related to "Connection Job"

Job	States
Closed	{CLOSED}
Connection	{WAIT CONNECT, WAIT CONNECT RSP}
Creation	{WAIT CREATE, WAIT CREATE RSP}
Configuration	{WAIT CONFIG, WAIT CONFIG RSP, WAIT CONFIG REQ, WAIT CONFIG REQ RSP, WAIT SEND CONFIG, WAIT IND FINAL RSP, WAIT FINAL RSP, WAIT CONTROL IND}
Disconnection	{WAIT DISCONNECT}
Move	{WAIT MOVE, WAIT MOVE RSP, WAIT MOVE CONFIRM, WAIT CONFIRM RSP}
Open	{OPEN}

Process 2: State Guiding(Cont.)

- **State Classification.**

2) Identifying the commands used for each Job.

ex) WAIT CONNECT accepts Connection Request.
WAIT CONNECT RSP accepts Connection Response.
Connection Request and Connection Response → Valid commands for Connection Job

Event	Action	State transition?
Connect Req	Connect Rsp	WAIT CONFIG
Connect Rsp	Reject	No
Config Req	Reject	No
Config Rsp	Reject	No
Disconnect Rsp	Reject	No
Create Channel Req	Reject	No
Create Channel Rsp	Reject	No
Move Channel Req	Reject	No
Move Channel Rsp	Reject	No
Move Channel Confirm Req	Reject	No
Move Channel Confirm Rsp	Reject	No

ex) WAIT CONNECT state's events and actions.

Process 2: State Guiding(Cont.)

- **State Classification.**

3) Mapping the valid commands to each job

Job	Valid commands
Closed	All commands
Connection	Connect Req/Rsp
Creation	Create Channel Req/Rsp
Configuration	Config Req/Rsp
Disconnection	Disconnect Req/Rsp
Move	Move Channel Req/Rsp, Move Channel Confirmation Req/Rsp
Open	All commands

- **State transition.**

- With the valid commands, L2Fuzz generates normal packet for state transition.

Process 3: Core Field Mutating

- **Field Classification.**

1) Segmenting L2CAP(L) into fixed(F), dependent(D), and mutable fields(M).

$$L = F \cup D \cup M$$

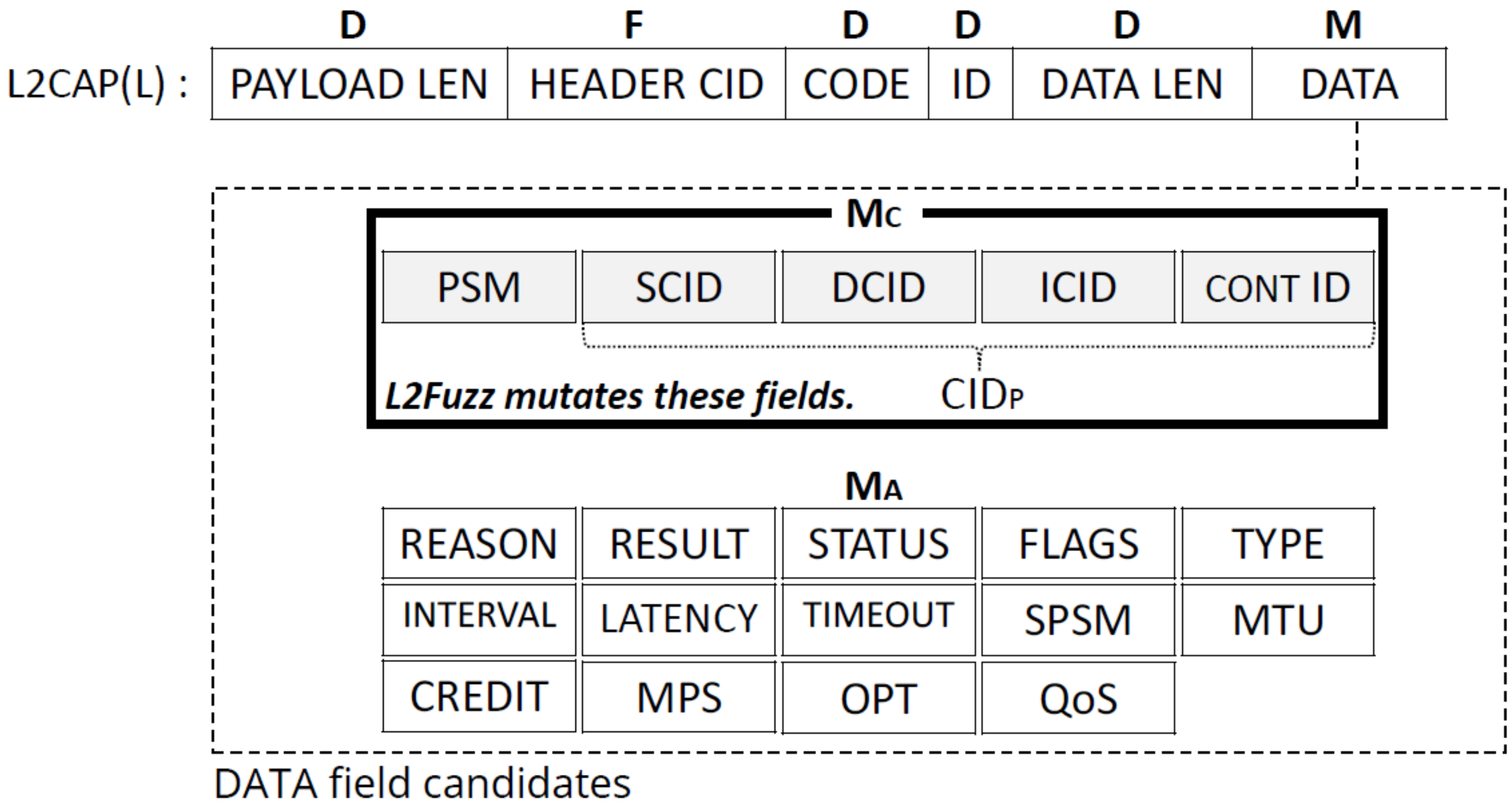
2) Classifying mutable fields(M) into mutable core fields(M_C) and mutable application fields(M_A).

$$M = M_C \cup M_A$$

Process 3: Core Field Mutating(Cont.)

- **Field Classification.**

3) Applying to Bluetooth L2CAP Packet frame.



Process 3: Core Field Mutating(Cont.)

- **Packet mutating.**

1) No mutating : fixed(F), dependent(D).
2) Mutating : mutable core fields(M_C).
3) Default value : mutable application fields(M_A).
4) Adding garbage value.

D	F	D	D	D	M_C	M_A	M_A	
P-LEN	H-CID	CODE	ID	DATA LEN	DCID	FLAGS	MTU	
0C 00	01 00	04	06	08 00	40 00	00 20	01 02 00 04	

⬇

0C 00	01 00	04	06	08 00	**8F 7B**	**00 00**	**00 00 00 00**	**D2 3A 91 0E**
No mutating					**Mutating**	**Default Value**		**Garbage Value**

Process 4: Vulnerability detecting

- **Analyzing Target Device.**

1) Error message
   - ✓ *Connection Failed, Connection Aborted, Connection Reset, Connection Refused, and Timeout.*

2) Ping test
   - ✓ Whether the target device is responding.

3) Crash dump
   - ✓ Whether the crash dump was generated in the target device.

Evaluation

- **Experimental Setup.**

python™

scapy

Ubuntu 18.04

- **Baseline Fuzzers for comparison.**

Defensics	Bluetooth Stack Smasher	bfuzz
Synopsys	SecuObs	IoTcube

Evaluation(Cont.)

- **Target devices.**

✓ Testing 4 main general-purpose Bluetooth host stacks.

1) Android BlueDroid
2) Linux BlueZ
3) Apple Bluetooth stack
4) Windows Bluetooth stack

No.	Type	Vendor	Name	Year	Model	Chip	OS or FW	BT Stack	BT Ver.
D1	Tablet PC	Google	Nexus 7	2013	ASUS-1A005A	Snapdragon 600	Android 6.0.1	BlueDroid	4.0 + LE
D2	Smartphone	Google	Pixel 3	2018	GA00464	Snapdragon 845	Android 11.0.1	BlueDroid	5.0 + LE
D3	Smartphone	Samsung	Galaxy 7	2016	SM-G930L	Exynos 8890	Android 8.0.0	BlueDroid	4.2
D4	Smartphone	Apple	iPhone 6S	2015	A1688	A9	iOS 15.0.2	iOS stack	4.2
D5	Earphone	Apple	Airpods 1 gen	2016	A1523	W1	6.8.8	RTKit stack	4.2
D6	Earphone	Samsung	Galaxy Buds+	2020	SM-R175NZKATUR	BCM43015	R175XXU0AUG1	BTW	5.0 + LE
D7	Laptop	LG	Gram	2019	15ZD990-VX50K	Intel wireless BT	Windows 10	Windows stack	5.0
D8	Laptop	LG	Gram	2017	15ZD970-GX55K	Intel wireless BT	Ubuntu 18.04.4	BlueZ	5.0

Evaluation(Cont.)

- **Evaluation Metrics.**

1) *Mutation efficiency*
   - Minimum percentage of malformed packets transmitted without rejection.
   * It uses Malformed Packet Ratio and Packet Rejection Ratio.

$$Mutation\ efficiency = MP\ Ratio * (1 - PR\ Ratio)$$

- Malformed Packet Ratio

$$MP\ Ratio = \frac{\#Transmitted\ Malformed\ Packets}{\#Transmitted\ Packets}$$

- Packet Rejection Ratio

$$PR\ Ratio = \frac{\#Received\ Rejection\ Packets\ from\ Target}{\#Received\ Packets\ from\ Target}$$

2) *State Coverage.*
   - the number of L2CAP states to be covered.

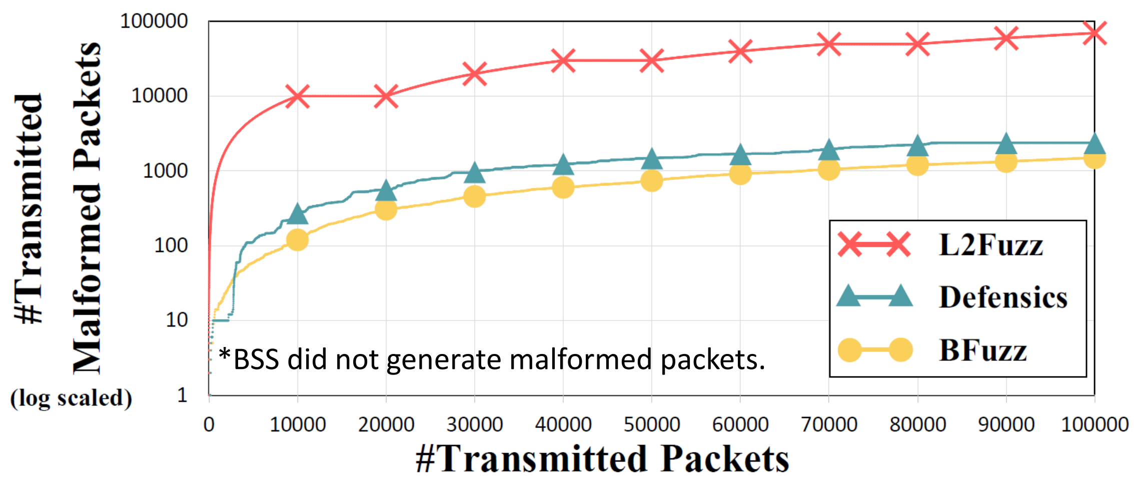
Mutation efficiency

- L2Fuzz shows the highest mutation efficiency.

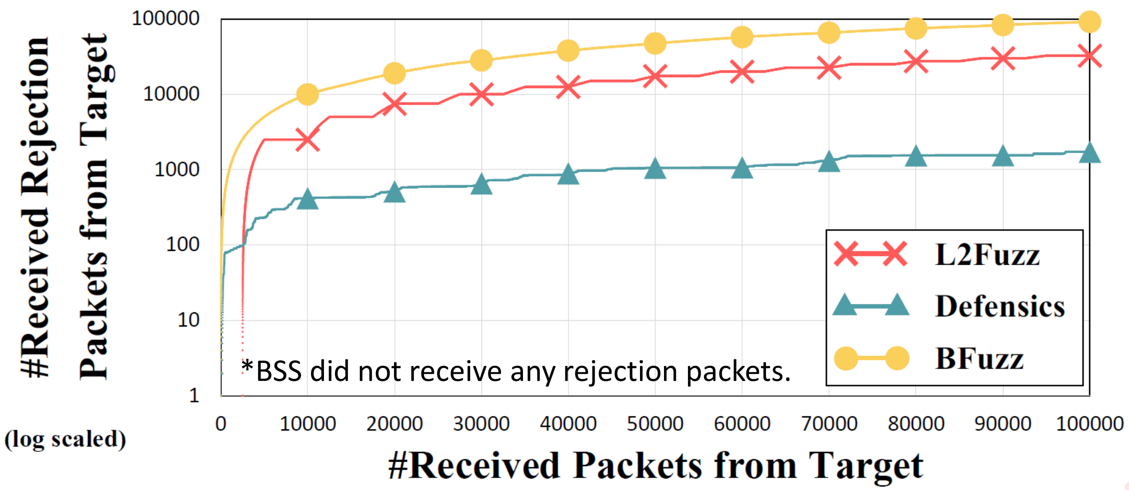
Fuzzer	MP Ratio	PR Ratio	Mutation efficiency
L2Fuzz	69.96%	32.49%	$\boxed{47.22\%}$
Defensics	2.38%	1.73%	2.33%
BFuzz	1.50%	91.60%	0.12%
BSS	0%	0%	0%

*MP Ratio = *Malformed Packet Ratio*
*PR Ratio = *Packet Rejection Ratio*
*Mutation efficiency = MP Ratio * (1 - PR Ratio)

<Mutation efficiency results>

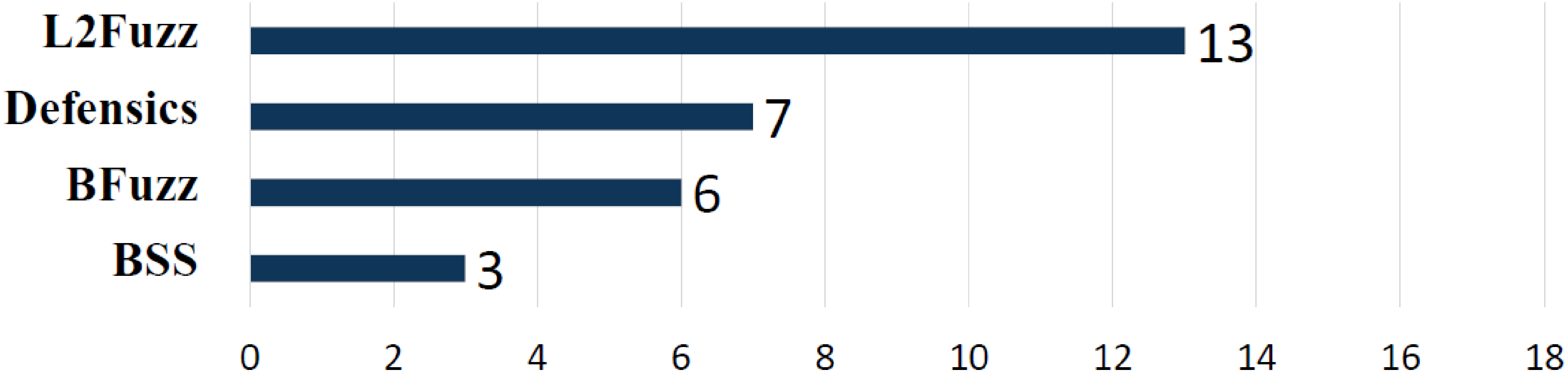


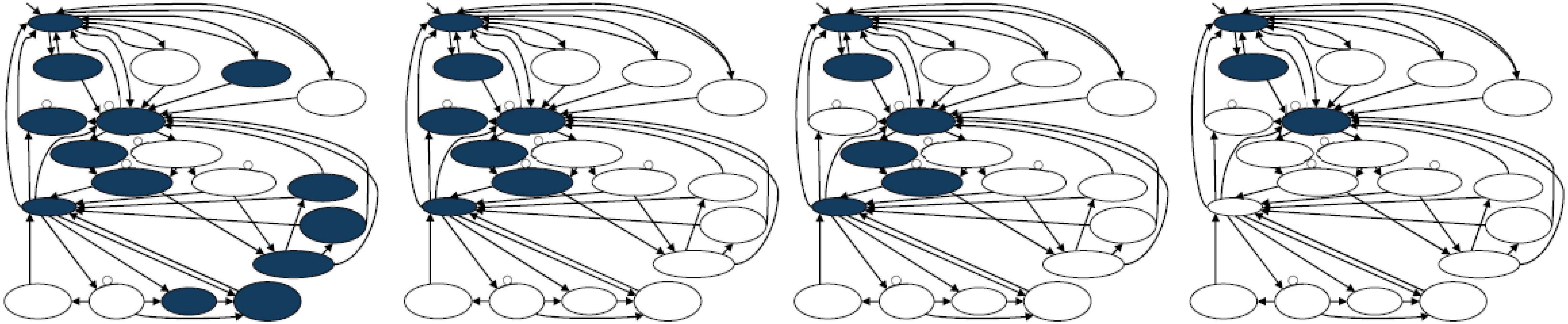
<MP Ratio measurement results>



<PR Ratio measurement results>

State Coverage





(a) L2Fuzz (b) Defensics (c) BFuzz (d) BSS

Vulnerability Detection Results

- **L2Fuzz detected five zero-day vulnerabilities.**

1) Nexus 7, Pixel 3, Galaxy 7 (Android): reported and discussing patch.
2) Airpods 1 gen (Apple's stack): reported and patched.
3) LG Gram (Ubuntu) : reported.

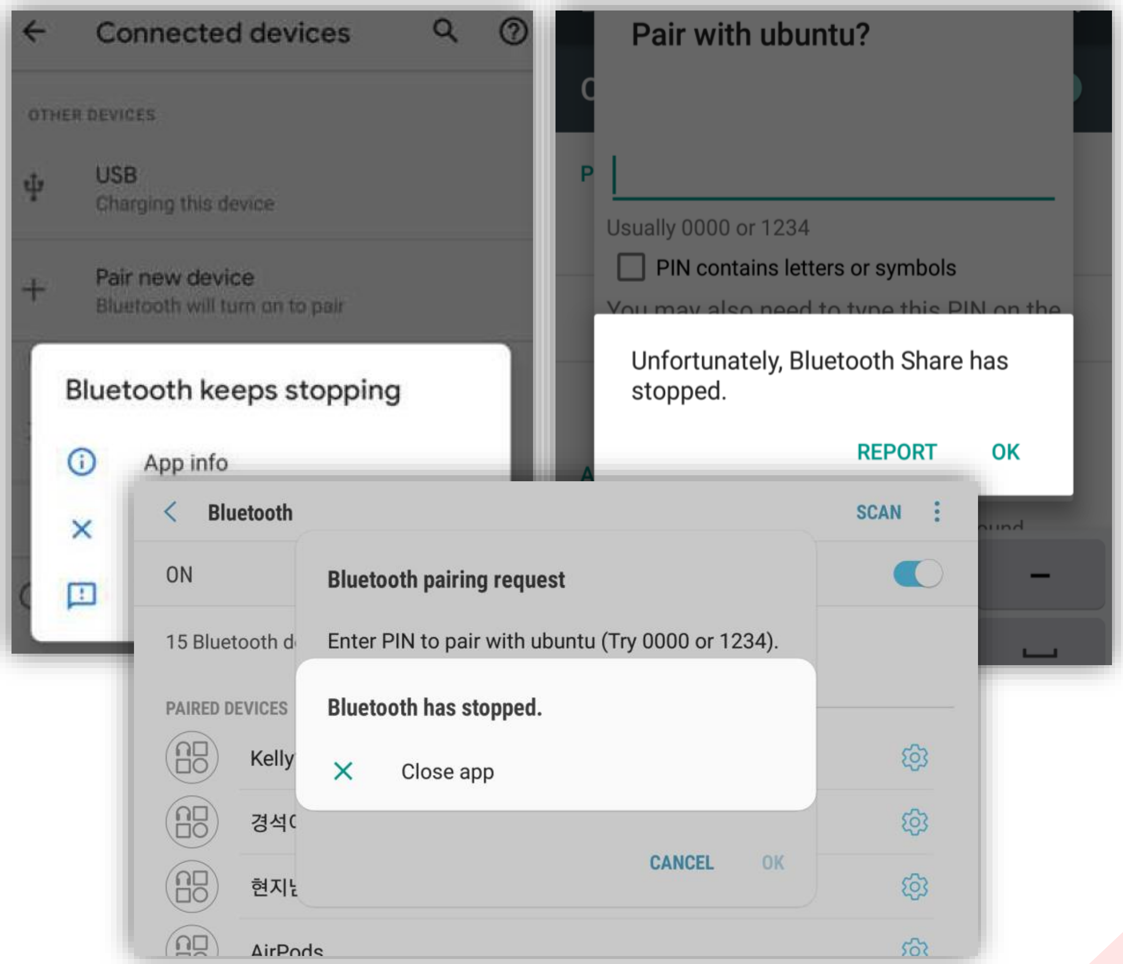
Type	Vendor	Name	Vuln?	Description	Elapsed Time	Reported to Vendors?
Tablet PC	Google	Nexus 7	**Yes**	DoS	1 m 32 s	Yes
Smartphone	Google	Pixel 3	**Yes**	DoS	1 m 25 s	Yes
Smartphone	Samsung	Galaxy 7	**Yes**	DoS	7 m 11 s	Yes
Smartphone	Apple	iPhone 6S	No	N/A	N/A	N/A
Earphone	Apple	Airpods 1 gen	**Yes**	Crash	40 s	Yes
Earphone	Samsung	Galaxy Buds+	No	N/A	N/A	N/A
Laptop	LG	Gram	No	N/A	N/A	N/A
Laptop	LG	Gram	**Yes**	Crash	2 h 40 m	Discussing

Case Study: DoS in Android Bluetooth

- **Remote temporary device denial of service.**


```
"cmd": "Configuration Request",
"cmd_code": 4,
"raw": "b'\\x04\\x00\\x04\\x00\\xbbY\\x00\\x00'",
"summary": "<bound method Packet.summary of
<L2CAP_CmdHdr  code=conf_req |<L2CAP_ConfReq  dcid=22971 |>>>",
"state": "Wait Send Config State",
"sended?": "no",
"crash": "yes",
"crash_info": "TimeoutError"
```


<L2Fuzz logfile>



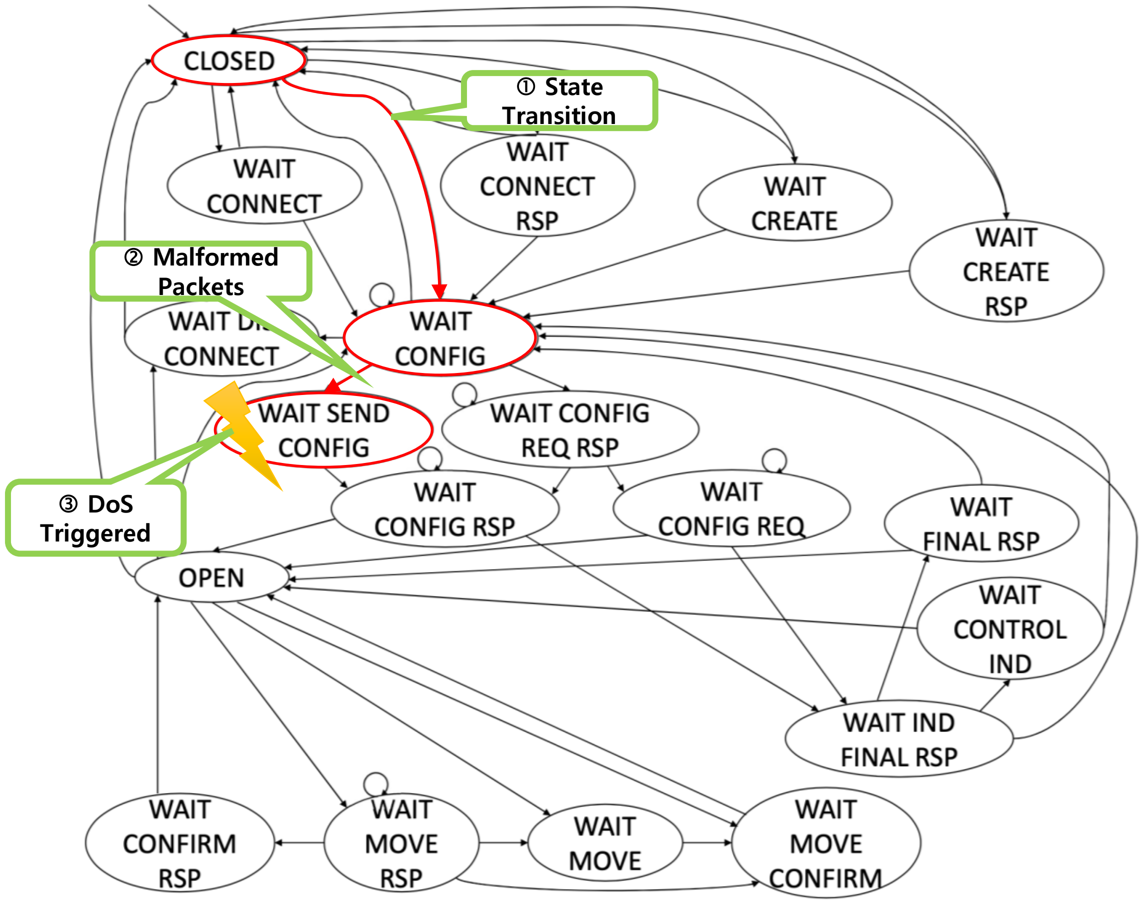
<DoS triggered in Android phones>

Case Study: DoS in Android Bluetooth(Cont.)

- **Remote temporary device denial of service.**

```
*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
Build fingerprint:
'google/blueline/blueline:11/RQ1D.210105.003/7005430:user/release-keys'
Revision: 'MP1.0'
ABI: 'arm64'
Timestamp: 2021-07-07 15:16:25+0900
pid: 1948, tid: 2946, name: bt_main_thread  >>> com.android.bluetooth <<<
uid: 1002
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x20
Cause: null pointer dereference
...
backtrace:
    #00 pc 0000000000378da0  /system/lib64/libbluetooth.so
(l2c_csm_execute(t_l2c_ccb*, unsigned short, void*)+3748) (BuildId:
3178e5a1f58c0a343c0d83be72d223da)
...
```

<ADB logfile – Google Pixel 3>



Discussion

- **Applicability to other protocols.**
  - RFCOMM, SDP, and OBEX
- **Countermeasures.**
  - Vendors are encouraged to update L2CAP layer.
- **Limitations and future works.**
  - Cannot test long-term.
  - Hard to analyze root cause immediately.
  - Cannot evaluate code coverage; because of closed-sources.
  - Cannot cover whole states.
- **Responsible vulnerability disclosure.**
  - All vulnerabilities are reported.
  - Several vulnerabilities are not disclosed due to the vendor's rejection.

Conclusion

- We present L2Fuzz, a stateful fuzzer for detecting Bluetooth L2CAP vulnerabilities.
- By State Guiding and Core Field Mutating, L2Fuzz can effectively detect vulnerabilities.
- With L2Fuzz, Developers can prevent risks in the Bluetooth host stack.

Q&A

- **Thanks for your attention.**
- L2Fuzz source code repository is (https://github.com/haramel/L2Fuzz).
- L2Fuzz will be available at (https://iotcube.net) as a part of BFuzz.

- **Contact**

- Haram Park (freehr94@korea.ac.kr)
- Computer & Communication Security Lab (https://ccs.korea.ac.kr)