ToothPicker
Apple Picking in the iOS Bluetooth Stack

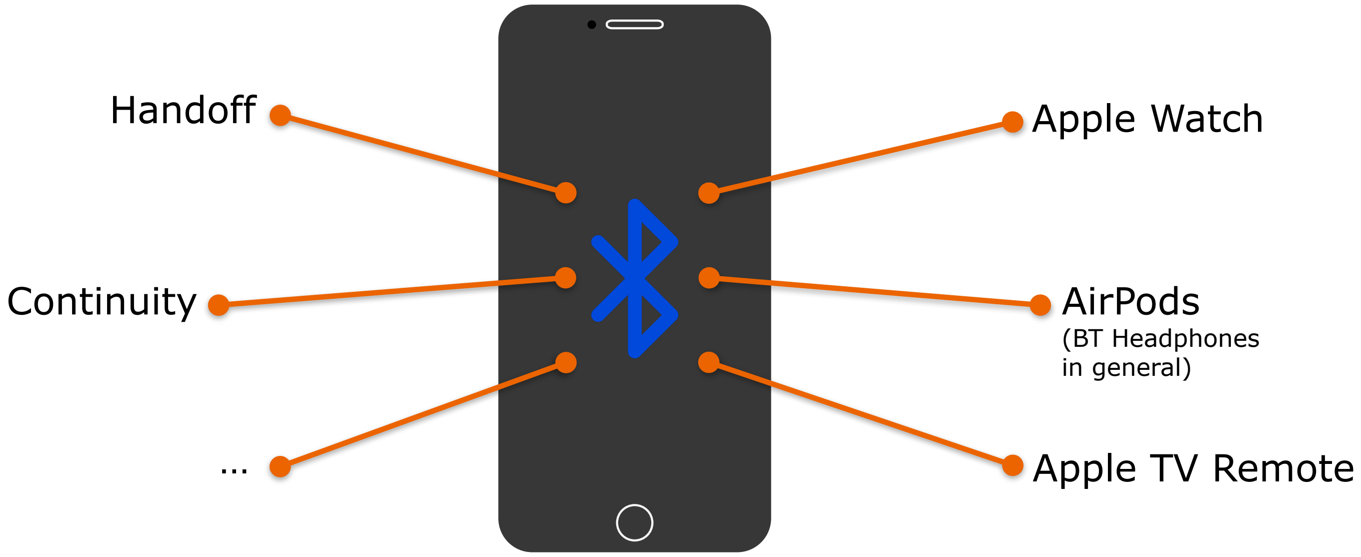
Dennis Heinze
Technische Universität Darmstadt
Secure Mobile Networking Lab - SEEMOO
ERNW Enno Rey Netzwerke GmbH

Jiska Classen, Matthias Hollick
Technische Universität Darmstadt
Secure Mobile Networking Lab - SEEMOO

Bluetooth in the Apple Ecosystem

Bluetooth in the Apple Ecosystem

The Apple ecosystem encourages turning on Bluetooth…

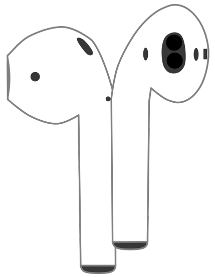


Bluetooth in the Apple Ecosystem

Three different Bluetooth stack implementations:

macOS

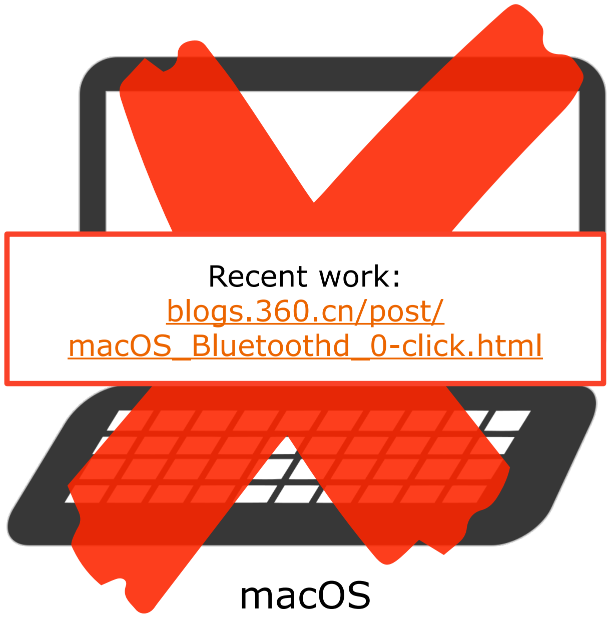
iOS



RTKit
(AirPods, Siri Remote, …)

Bluetooth in the Apple Ecosystem

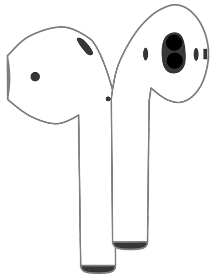
Three different Bluetooth stack implementations:



Recent work:
blogs.360.cn/post/macOS_Bluetoothd_0-click.html

macOS

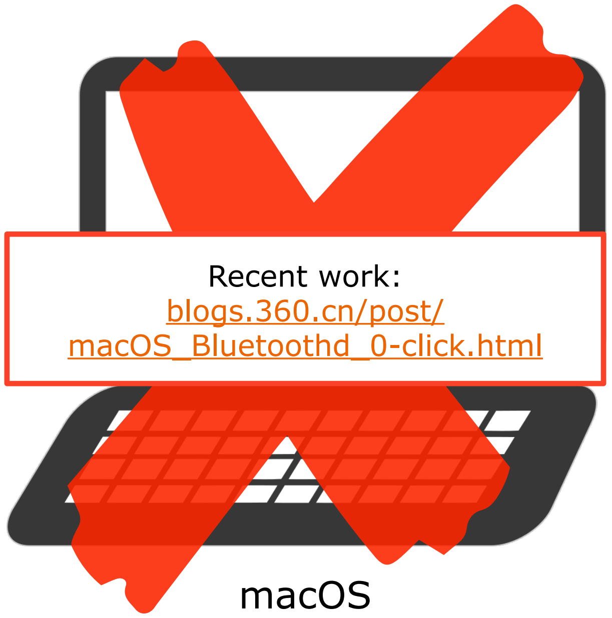
iOS



RTKit
(AirPods, Siri Remote, ...)

Bluetooth in the Apple Ecosystem

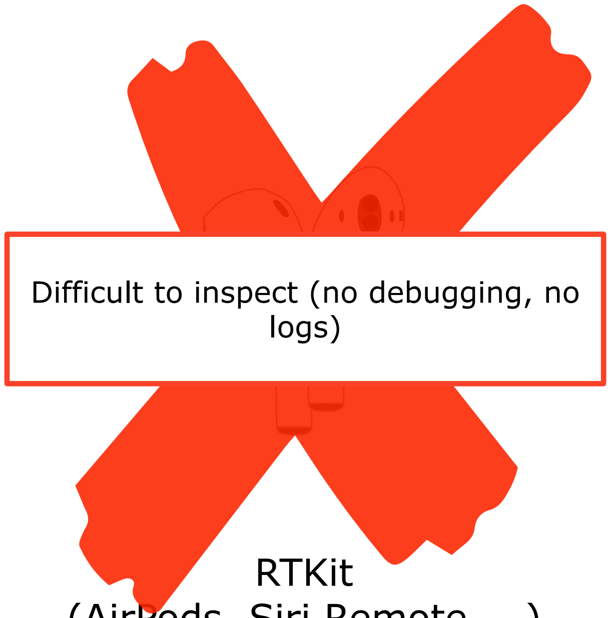
Three different Bluetooth stack implementations:



Recent work:
blogs.360.cn/post/
macOS_Bluetoothd_0-click.html

macOS

iOS

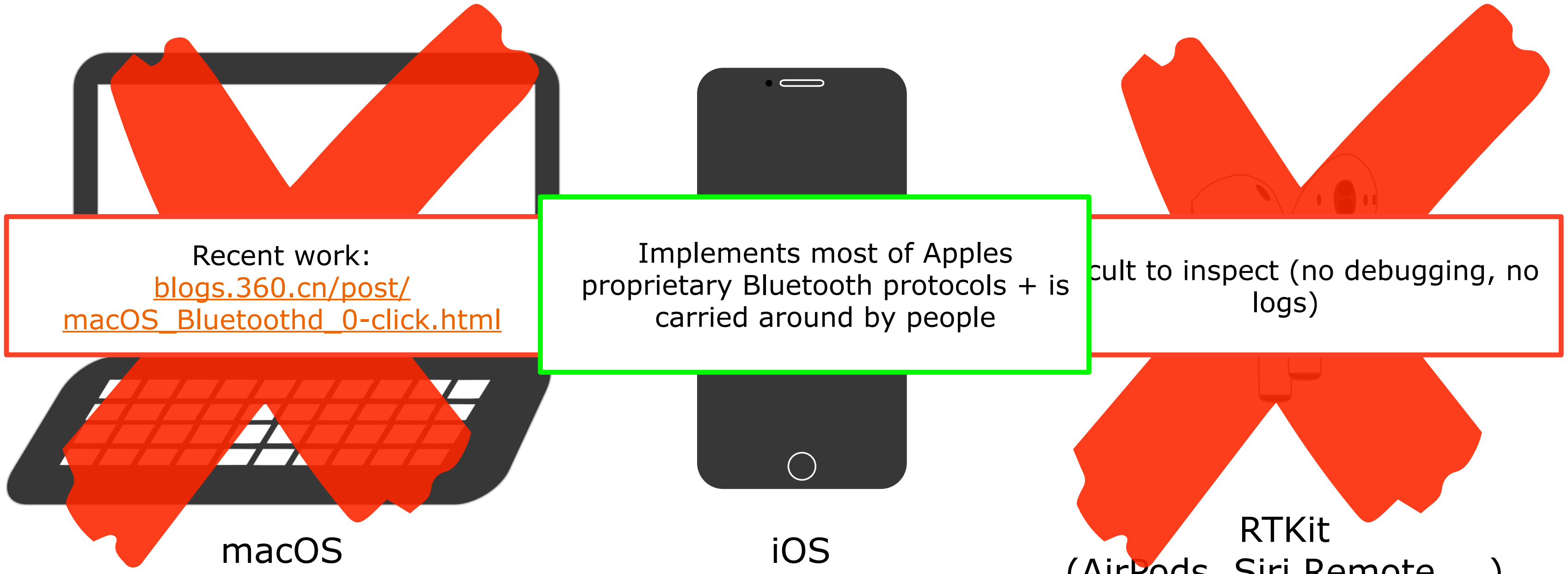


Difficult to inspect (no debugging, no logs)

RTKit
(AirPods, Siri Remote, …)

Bluetooth in the Apple Ecosystem

Three different Bluetooth stack implementations:



Recent work:
blogs.360.cn/post/
macOS_Bluetoothd_0-click.html

Implements most of Apples proprietary Bluetooth protocols + is carried around by people

cult to inspect (no debugging, no logs)

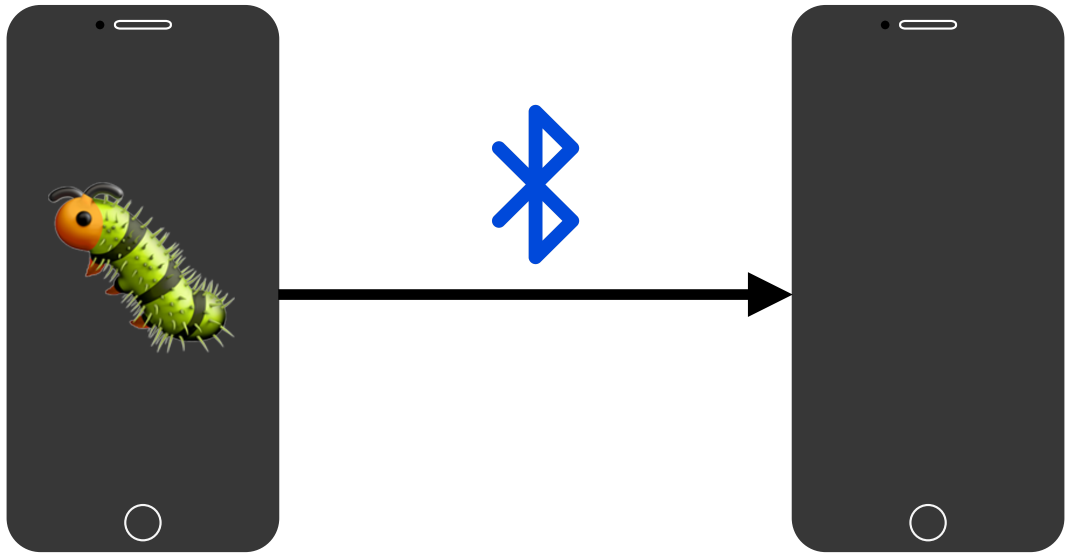
macOS

iOS

RTKit
(AirPods, Siri Remote, …)

Bluetooth on iOS

While it's not a "remote" zero-click attack surface for targeted attacks, Bluetooth RCEs are easily worm-able

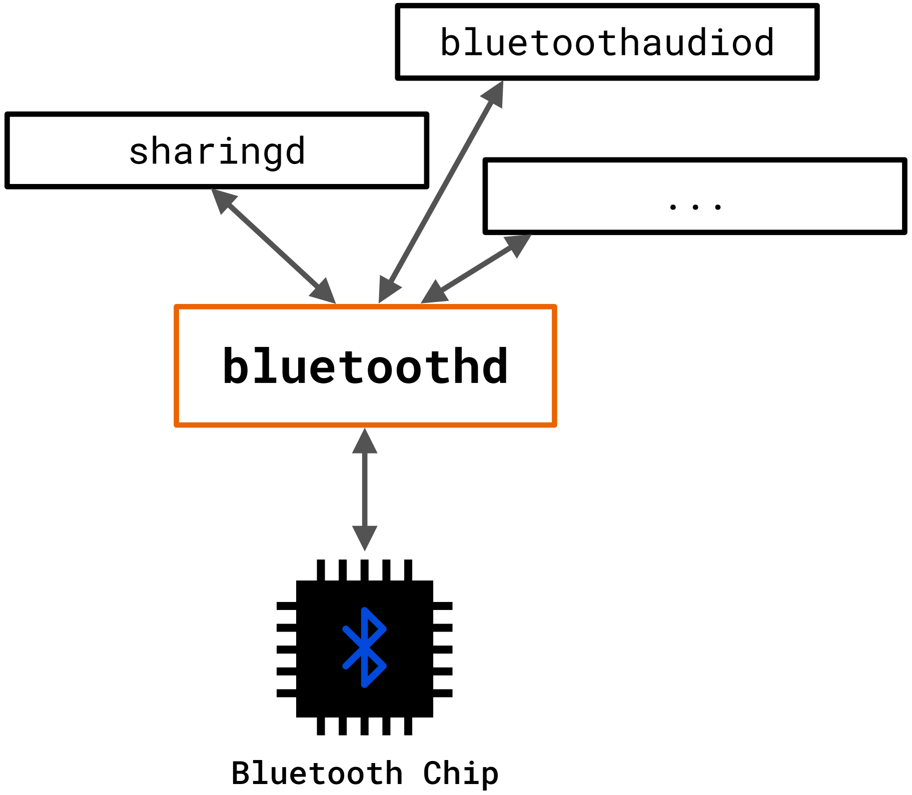


Proprietary Bluetooth Protocols

Category	Protocol	iOS	macOS	RTKit
Fixed L2CAP Channels	MagicPairing	✓	✓	✓
	Magnet	✓	✓	-
	LEA{P,S}	✓	-	✓
	FastConnect Discovery	✓	✓	✓
	DoAP	✓	✓	✓
L2CAP Channels	ExternalAccessory	✓	✓	✓
	AAP	✓	✓	✓
	Magnet Channels	✓	✓	-
	FastConnect	✓	✓	✓
	Apple Pencil GATT	✓	-	✓
Other	BRO/UTP	-	-	✓
	USB OOB Pairing	-	✓	-

Fuzzing iOS bluetoothd

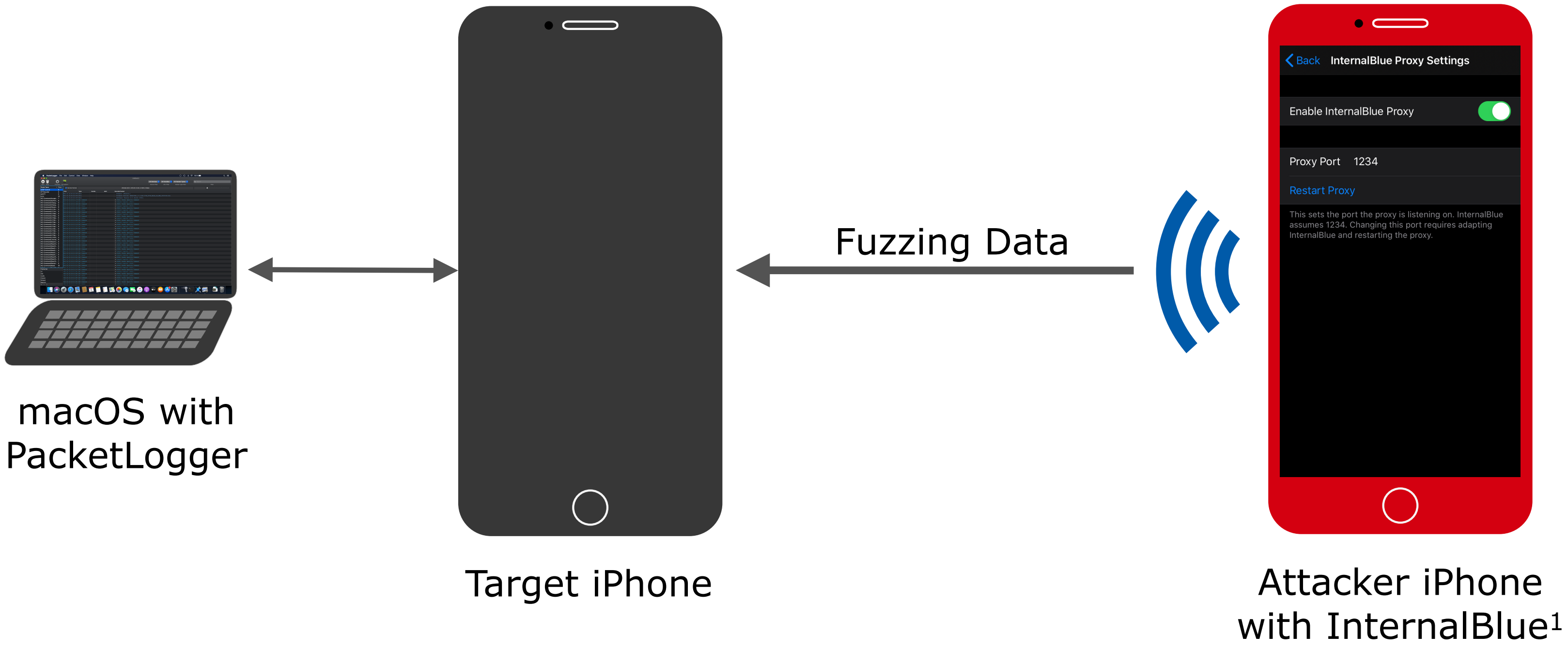
Bluetooth on iOS



- Lots of interaction with different system daemons
- Constant interaction with the Bluetooth Chip
- Multiple Threads
  - StackLoop (for HCI[1])
  - RxLoop
  - TxLoop
  - ...
- Huge binary file
- (Almost) no symbols

1: Host Controller Interface, interface to interact with BT Chip

Over-the-Air Fuzzing



Back InternalBlue Proxy Settings

Enable InternalBlue Proxy

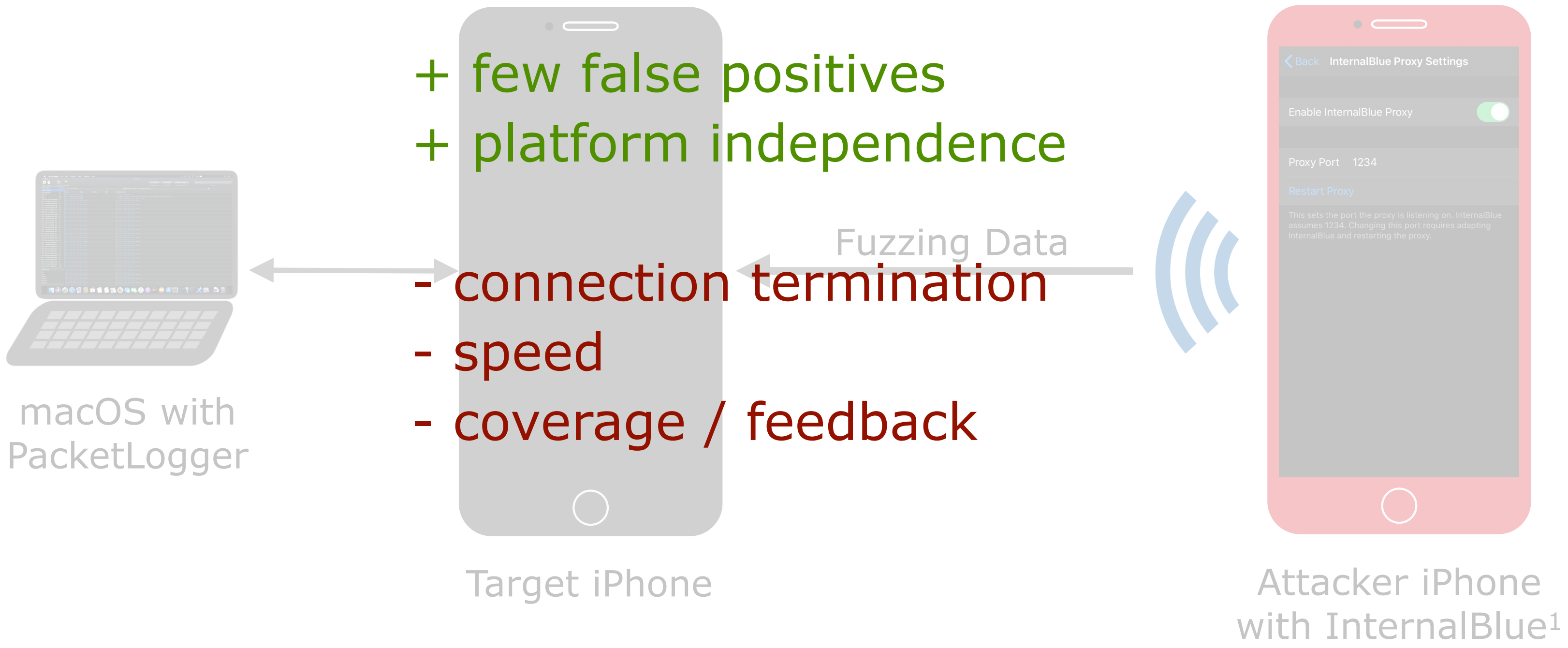
Proxy Port 1234

Restart Proxy

This sets the port the proxy is listening on. InternalBlue assumes 1234. Changing this port requires adapting InternalBlue and restarting the proxy.

1: https://github.com/seemoo-lab/internalblue

Over-the-Air Fuzzing


macOS with PacketLogger
Target iPhone
Fuzzing Data
Attacker iPhone with InternalBlue[1]
Back
InternalBlue Proxy Settings
Enable InternalBlue Proxy
Proxy Port 1234
Restart Proxy

This sets the port the proxy is listening on. InternalBlue assumes 1234. Changing this port requires adapting internalBlue and restarting the proxy.

+ few false positives
+ platform independence

- connection termination
- speed
- coverage / feedback

1: https://github.com/seemoo-lab/internalblue

Fuzzing bluetoothd

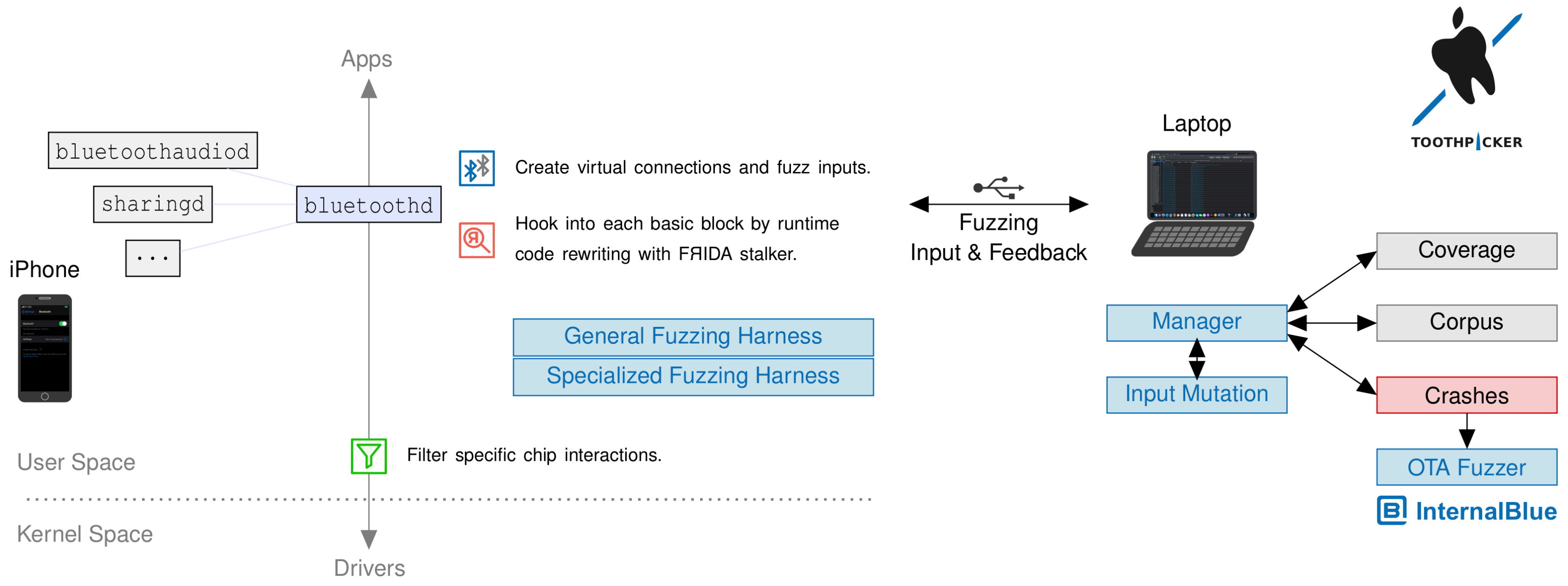
Coverage $\rightarrow$ FЯIDA Stalker

Feedback on crashes $\rightarrow$ FЯIDA Exception Handler

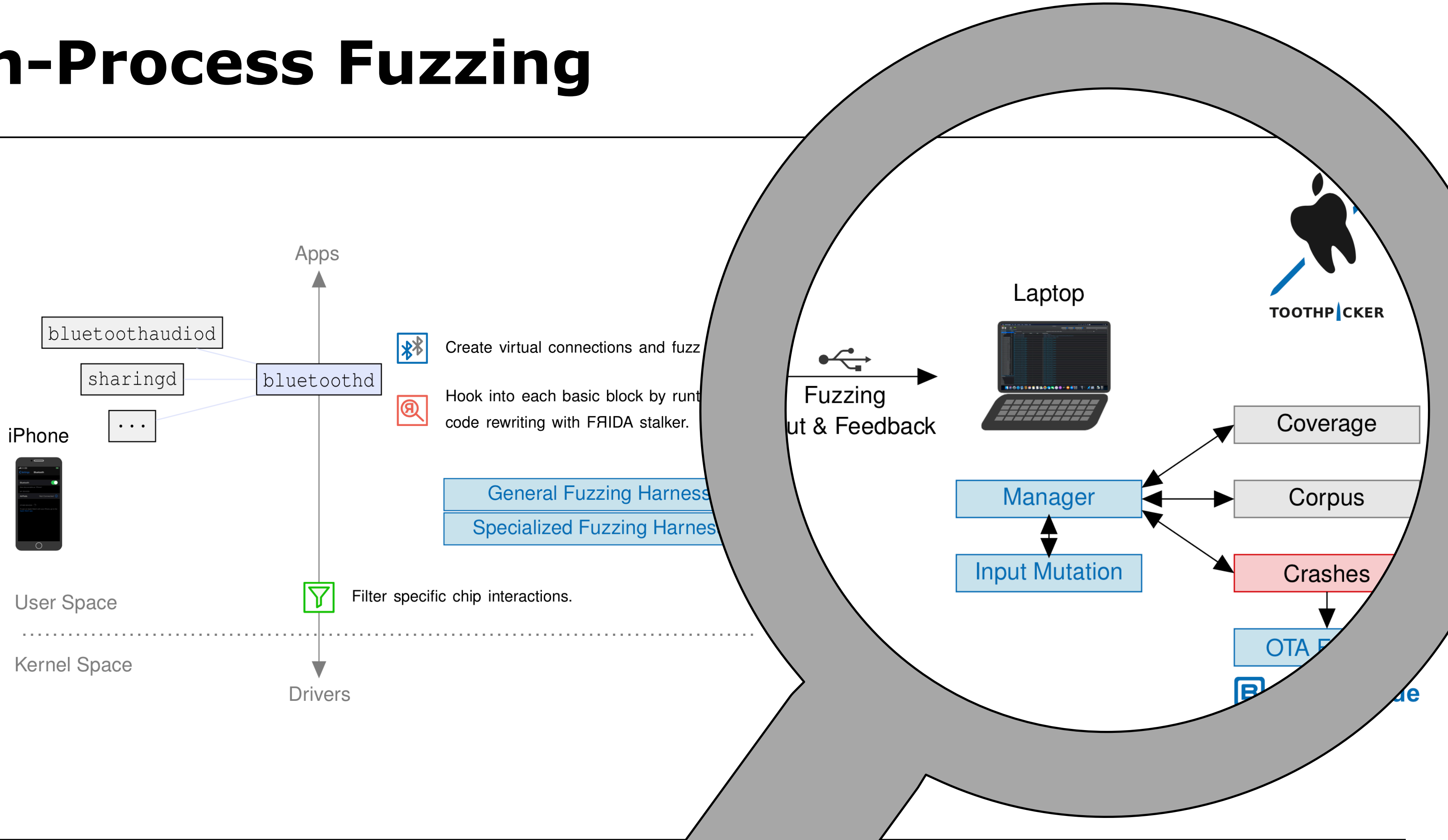
$$\left.\begin{array}{r}\text{No physical connection}\\ \text{No connection termination}\end{array}\right\} \rightarrow \begin{array}{l}\text{Virtual Connections by}\\ \text{code injection}\end{array}$$

ToothPicker

In-Process Fuzzing



In-Process Fuzzing

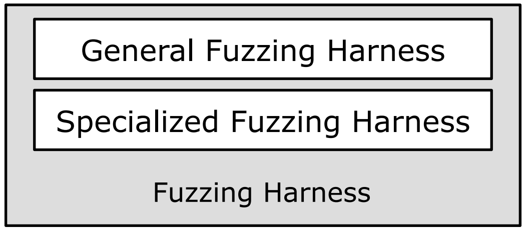


Create virtual connections and fuzz

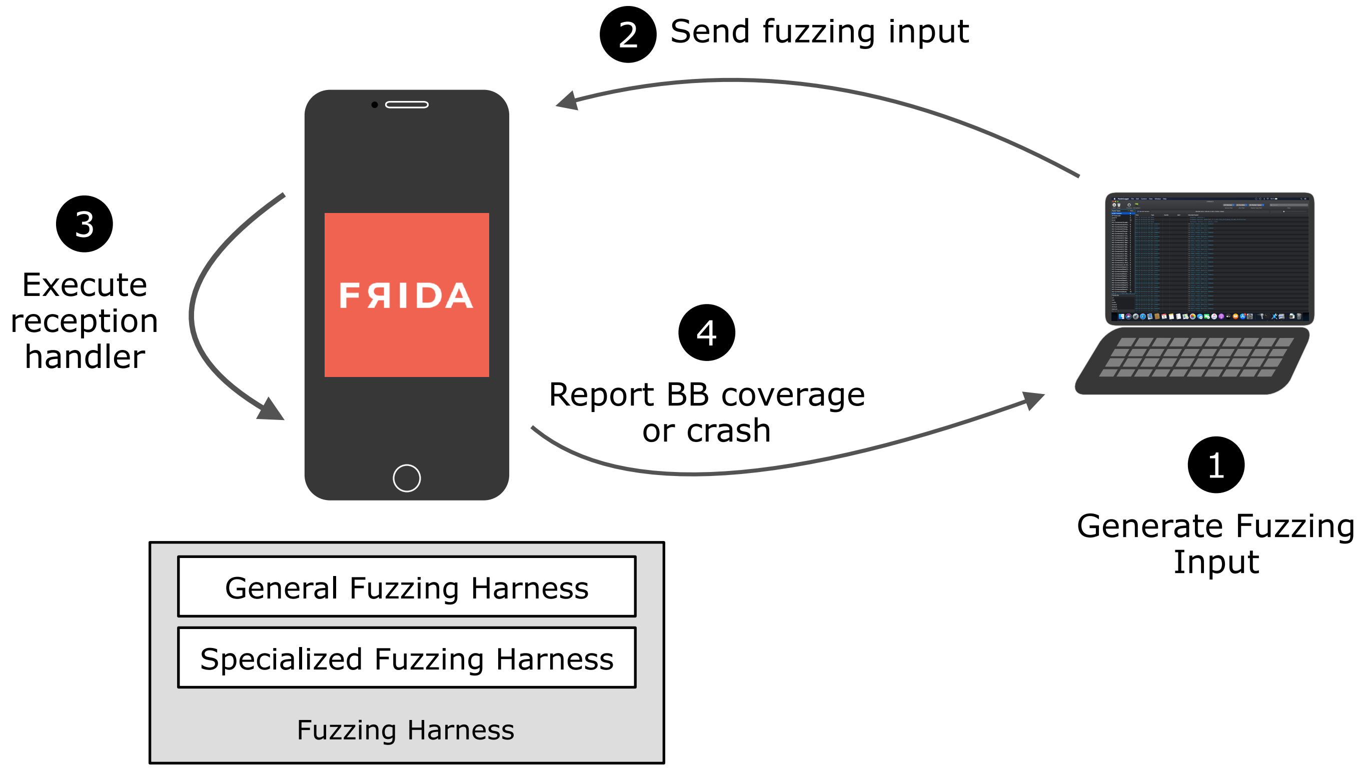
Hook into each basic block by runt
code rewriting with FRIDA stalker.

Filter specific chip interactions.

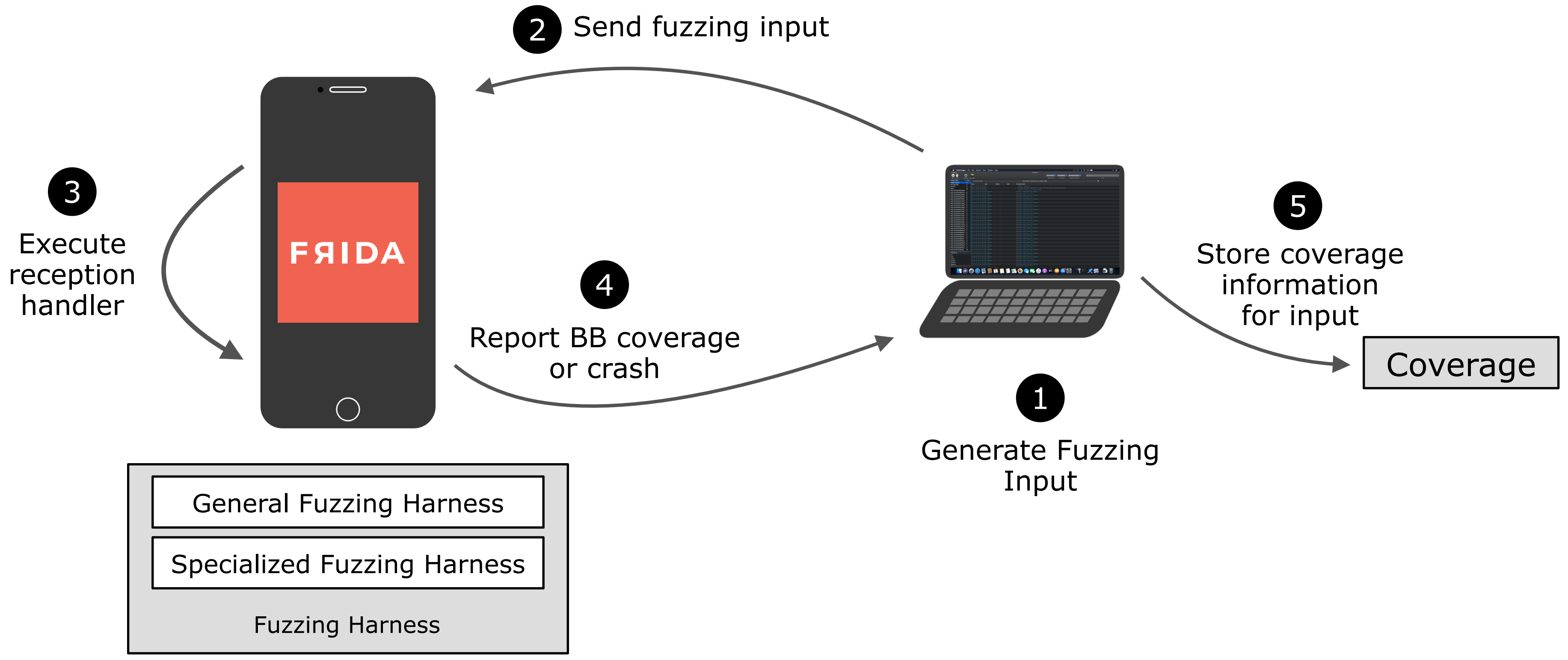
In-Process Fuzzing



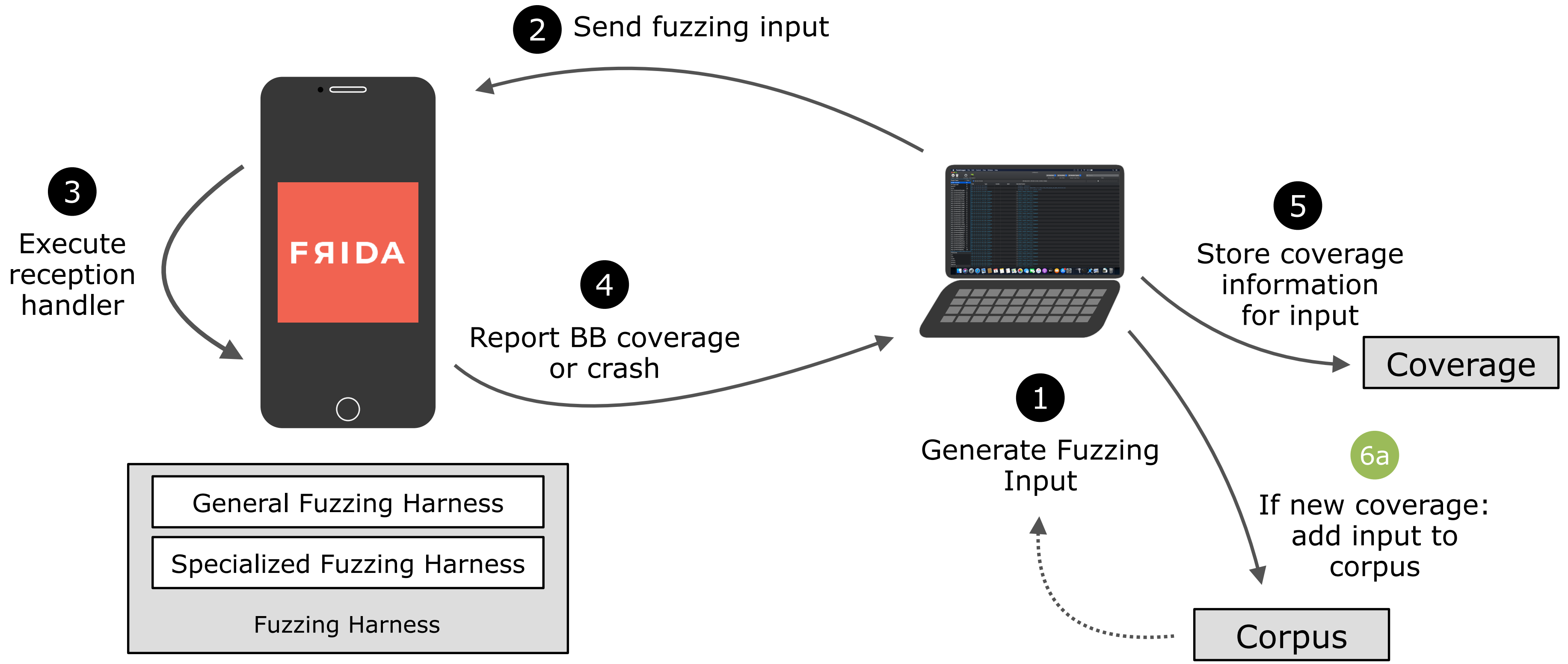
In-Process Fuzzing



In-Process Fuzzing

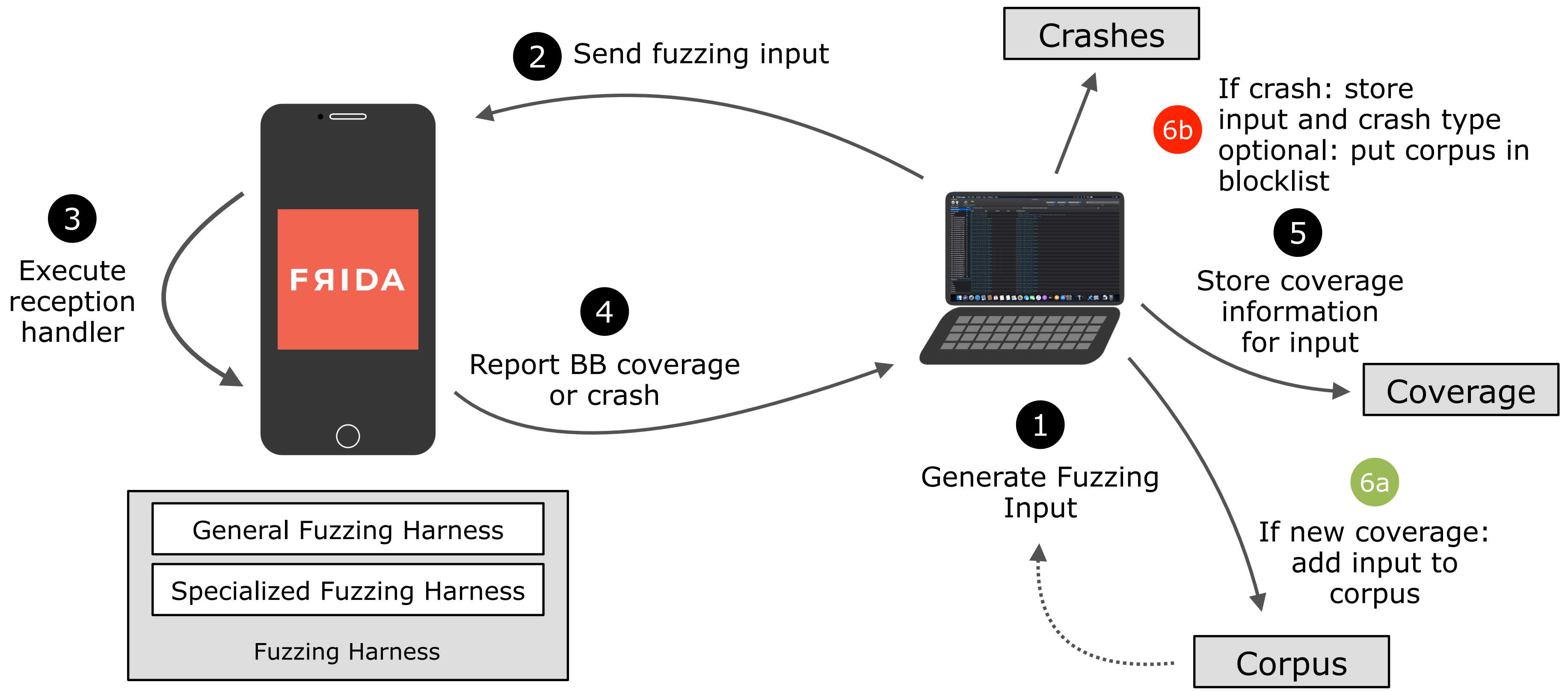

2 Send fuzzing input
3 Execute reception handler
FRIDA
4 Report BB coverage or crash
5 Store coverage information for input
Coverage
1 Generate Fuzzing Input
General Fuzzing Harness
Specialized Fuzzing Harness
Fuzzing Harness

In-Process Fuzzing



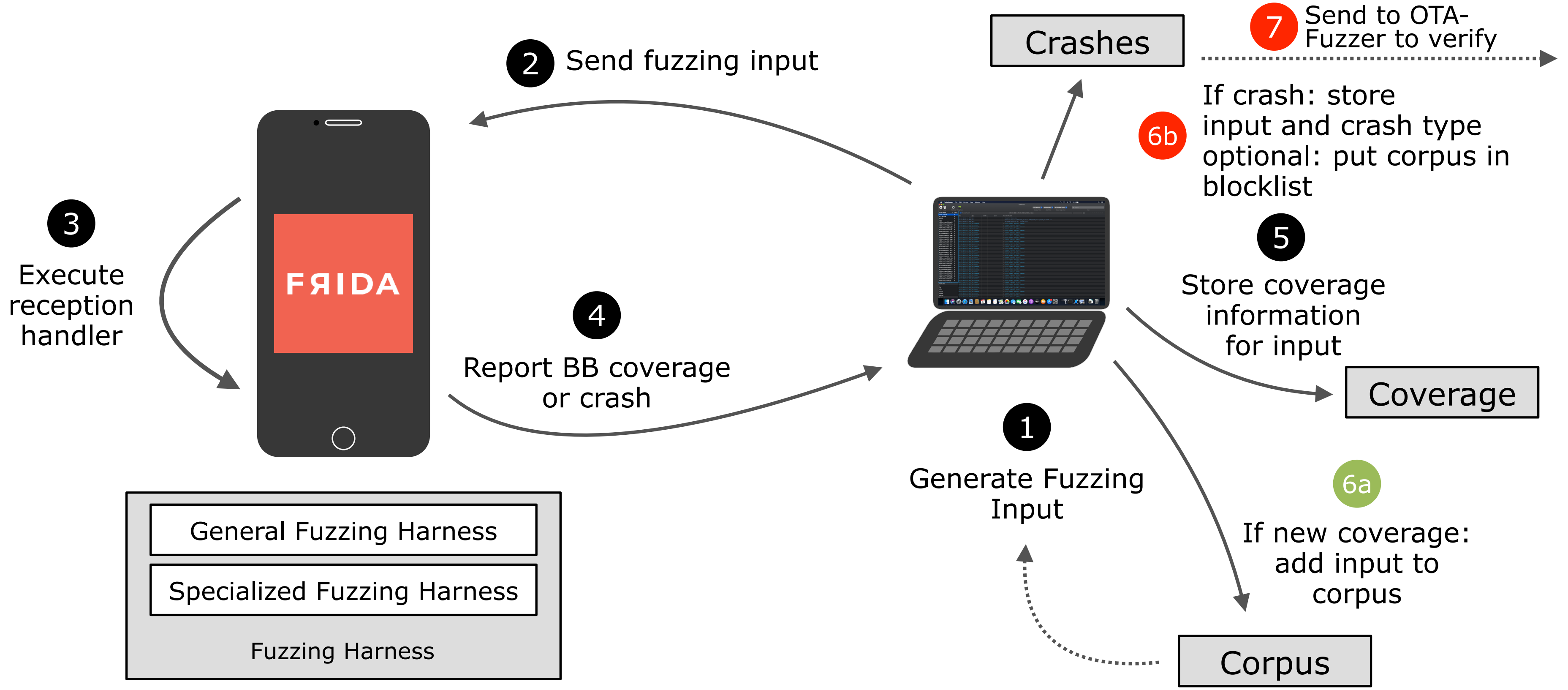
2 Send fuzzing input
3 Execute reception handler
FRIDA
4 Report BB coverage or crash
5 Store coverage information for input
Coverage
1 Generate Fuzzing Input
6a If new coverage: add input to corpus
Corpus
General Fuzzing Harness
Specialized Fuzzing Harness
Fuzzing Harness

In-Process Fuzzing

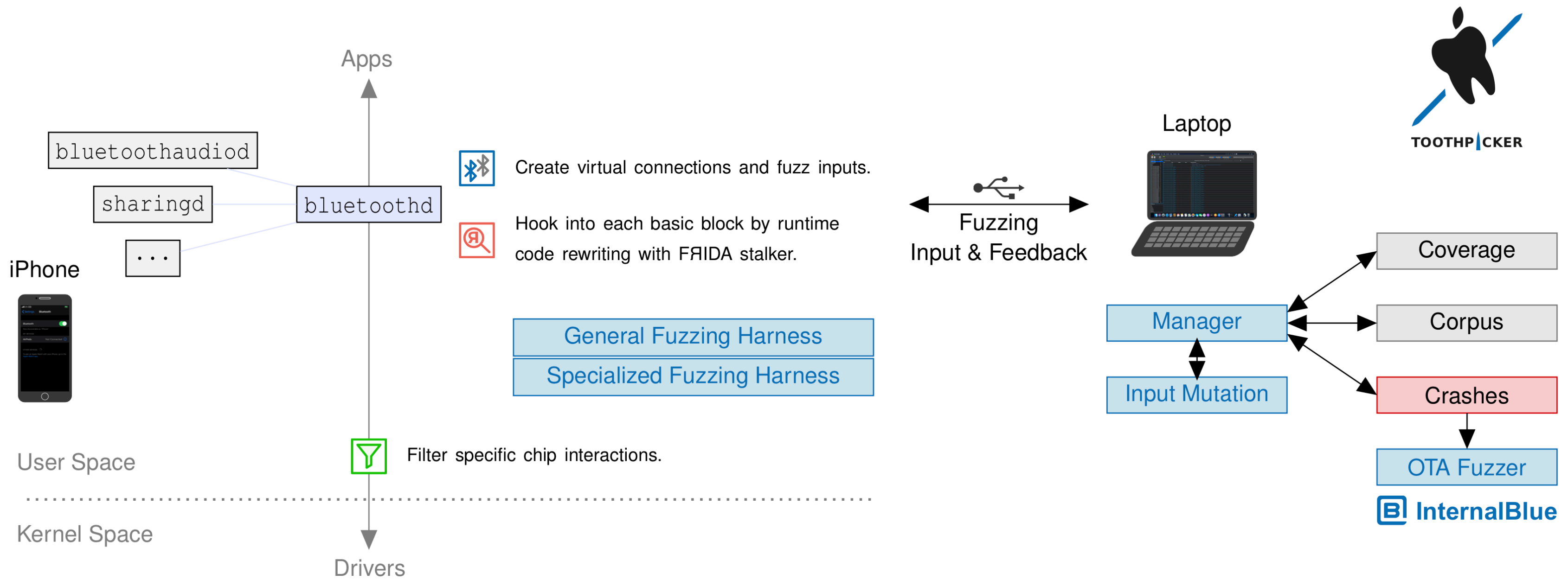


2 Send fuzzing input
Crashes
6b If crash: store input and crash type optional: put corpus in blocklist
3 Execute reception handler
FRIDA
5 Store coverage information for input
4 Report BB coverage or crash
Coverage
1 Generate Fuzzing Input
6a If new coverage: add input to corpus
General Fuzzing Harness
Specialized Fuzzing Harness
Fuzzing Harness
Corpus

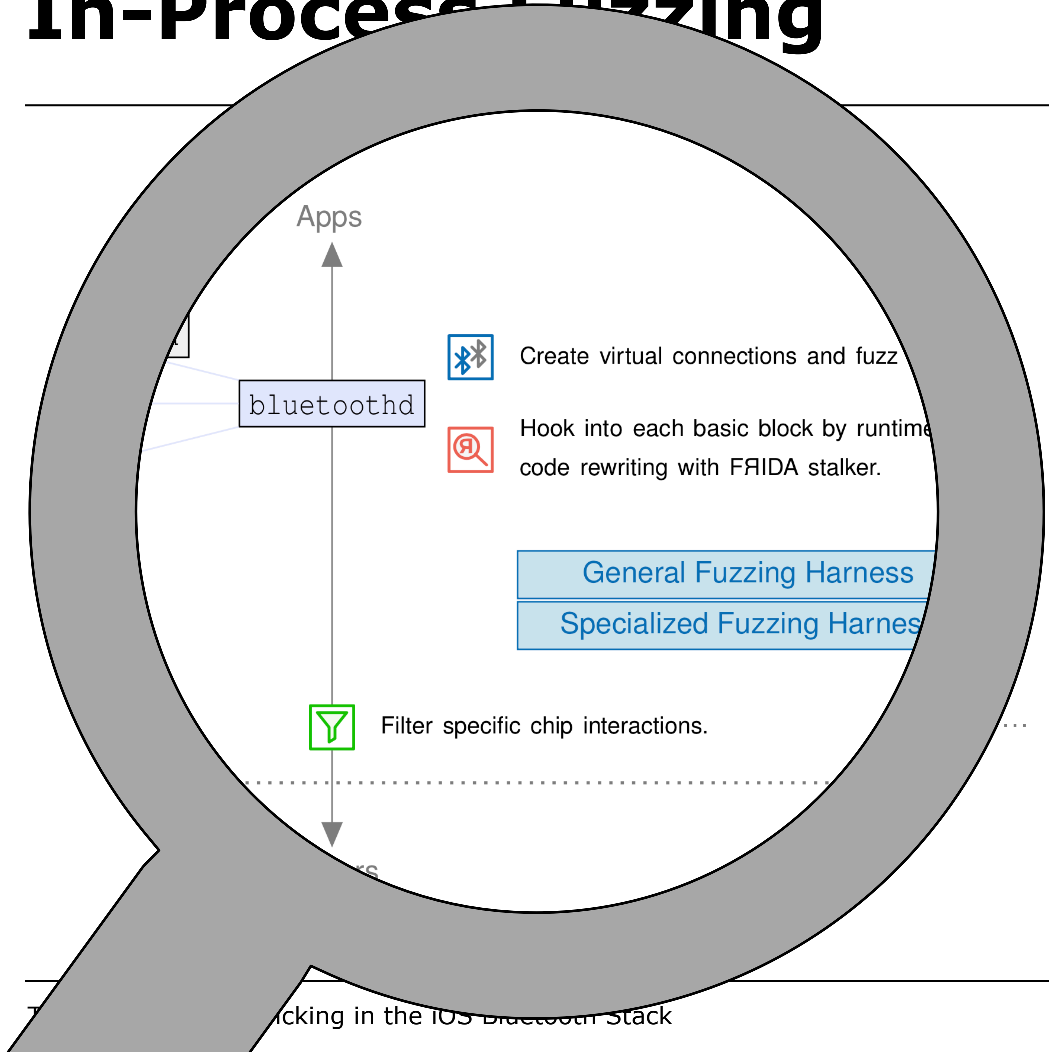
In-Process Fuzzing

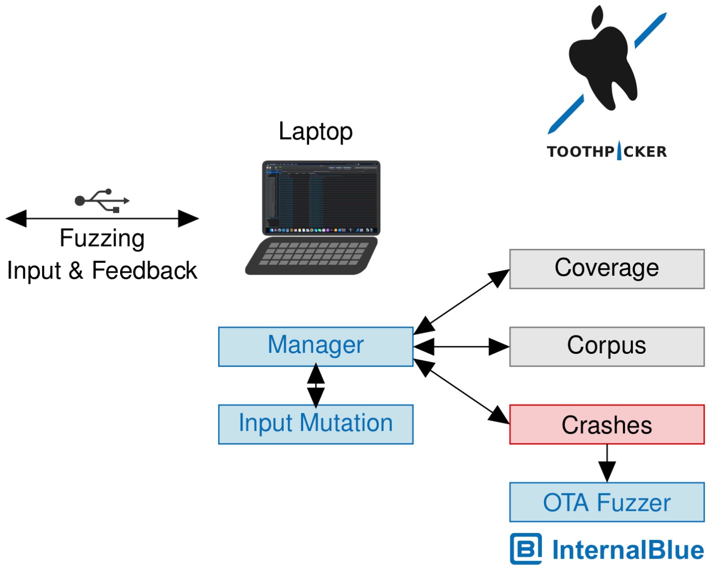

2 Send fuzzing input
Crashes
7 Send to OTA-Fuzzer to verify
6b If crash: store input and crash type optional: put corpus in blocklist
3 Execute reception handler
FRIDA
4 Report BB coverage or crash
5 Store coverage information for input
Coverage
1 Generate Fuzzing Input
6a If new coverage: add input to corpus
Corpus
General Fuzzing Harness
Specialized Fuzzing Harness
Fuzzing Harness

In-Process Fuzzing

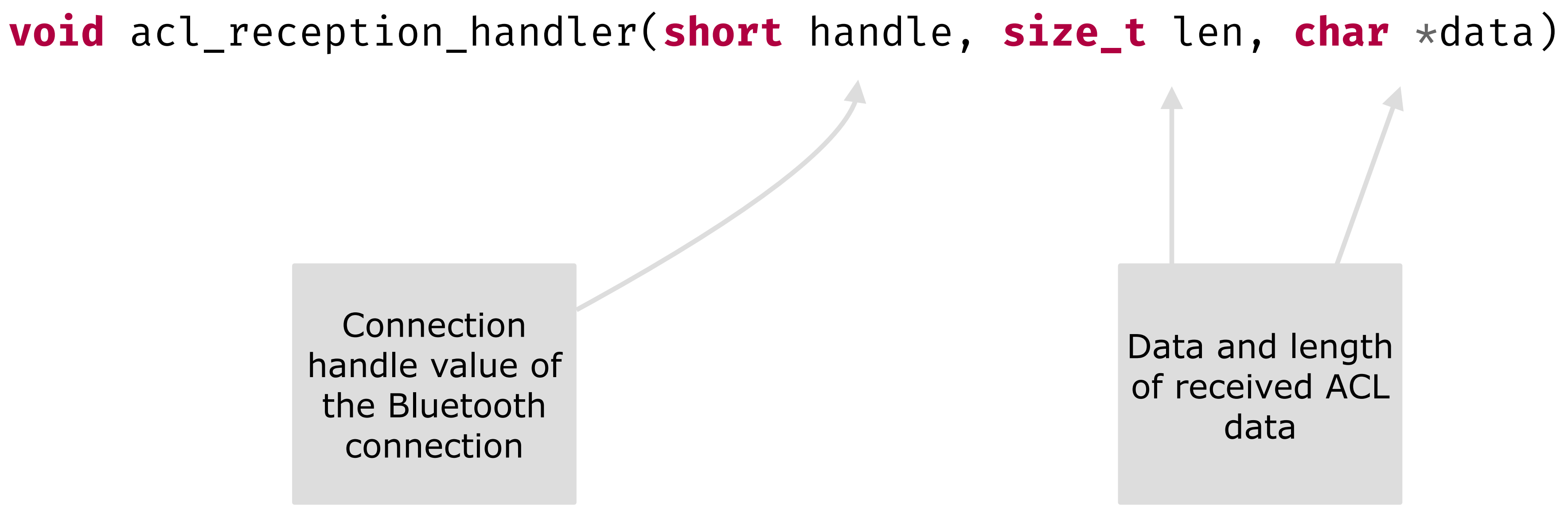


In-Process Fuzzing


Apps
bluetoothd
Create virtual connections and fuzz
Hook into each basic block by runtime
code rewriting with FRIDA stalker.
General Fuzzing Harness
Specialized Fuzzing Harnes
Filter specific chip interactions.



In-Process Fuzzing



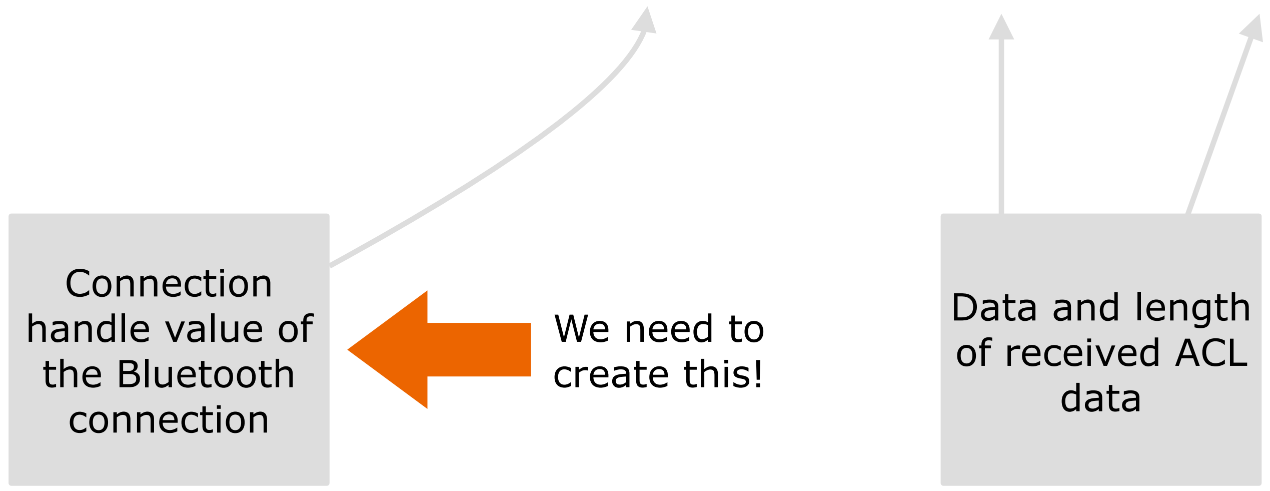
```
void acl_reception_handler(short handle, size_t len, char *data)
```

Connection handle value of the Bluetooth connection
Data and length of received ACL data

The functions and structures are named by us, Apple stripped all these symbols

In-Process Fuzzing

```
void acl_reception_handler(short handle, size_t len, char *data)
```


Connection handle value of the Bluetooth connection
We need to create this!
Data and length of received ACL data

The functions and structures are named by us, Apple stripped all these symbols

In-Process Fuzzing

```
bt_connection_t *allocate_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

The functions and structures are named by us, Apple stripped all these symbols

In-Process Fuzzing

```
bt_connection_t *allocate_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

Set the handle value of the connection:

```
*(short*)connection = 0x11;
```

The functions and structures are named by us, Apple stripped all these symbols

In-Process Fuzzing

```
bt_connection_t *allocate_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

Set the handle value of the connection:

```
*(short*)connection = 0x11;
```

Now we can call the reception handler with our fuzzing data

```
acl_reception_handler(0x11, len, data);
```

The functions and structures are named by us, Apple stripped all these symbols

In-Process Fuzzing

- Forge connection
  - Call `allocate_connection` to create connection object
  - Set handle value of the connection
- Filter BT Chip interaction
  - Overwrite other HCI-related functions that confuse `bluetoothd` (the connection is not real and the BT chip does not know the handle value)
- Stabilize Connection
  - Overwrite functions that force-disconnect the handle

➡ Similar process for BLE connections (more complex connection creation)

Results

Bluetooth Protocol Targets

Category	Protocol	iOS	macOS	RTKit	Accessibility	Proprietary	Knowledge	Target
Fixed L2CAP Channels	MagicPairing	✓	✓	✓	↑	✓	↑	✓
	GATT	✓	✓	(✓)	↑		↑	✓
	Signal Channel	✓	✓	✓	↑		↑	✓
	Magnet	✓	✓	?	-	✓	-	✓
	LEA{P,S}	✓		✓	-	✓	-	✓
	FastConnect Discovery	✓	✓	✓	↑	✓	↑	✓
L2CAP Channels	SDP	✓	✓	✓	↑		↑	✓
Other	ACL	✓	✓	✓	↑		↑	✓

Performance

- 25-30 messages per second
- bottlenecks:
  - FЯIDA Instrumentation
  - radamsa input mutation[1]
  - on never devices: Pointer Authentication
- Accumulated coverage: ~6.000 BBs of 153.620 BBs
  - coverage is only a small part of `bluetoothd`
  - however, ACL-based Bluetooth protocols prior to pairing are also only a small part of `bluetoothd`
  - hard to determine the exact number of BBs for these

1: https://gitlab.com/akihe/radamsa/-/issues/66

Results

ID	Description	Effect	Detection Method	OS	Disclosure	Status
MP1	Ratchet AES SIV	Crash	ToothPicker	iOS	Oct 30 2019	Not fixed
MP2	Hint	Crash	ToothPicker	iOS	Dec 4 2019	Not fixed
MP7	Ratchet AES SIV	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
MP8	Ratchet AES SIV	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
L2CAP2	Group Message	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
LEAP1	Version Leak	Information Disclosure	Manual	iOS	Mar 31 2020	Not fixed
SMP1	SMP OOB	Partial PC Control	ToothPicker	iOS	Mar 31 2020	Fixed in iOS 13.5: CVE-2020-9838
SIG1	Missing Checks	DoS	ToothPicker	iOS	Mar 31 2020	Fixed in iOS 13.6: CVE-2020-9931

Results

ID	Description	Effect	Detection Method	OS	Disclosure	Status
MP1	Ratchet AES SIV	Crash	ToothPicker	iOS	Oct 30 2019	Not fixed
MP2	Hint	Crash	ToothPicker	iOS	Dec 4 2019	Not fixed
MP7	Ratchet AES SIV	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
MP8	Ratchet AES SIV	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
L2CAP2	Group Message	Crash	ToothPicker	iOS	Mar 13 2020	Not fixed
LEAP1	Version Leak	Information Disclosure	Manual	iOS	Mar 31 2020	Not fixed
SMP1	SMP OOB	Partial PC Control	ToothPicker	iOS	Mar 31 2020	Fixed in iOS 13.5: CVE-2020-9838
SIG1	Missing Checks	DoS	ToothPicker	iOS	Mar 31 2020	Fixed in iOS 13.6: CVE-2020-9931

TOOTHPICKER

Twitter:
@ttdennis
@naehrdine
@seemoolab

github.com/seemoo-lab/toothpicker

dennis@bluetooth.lol
jiska@bluetooth.lol
mhollick@seemoo.de