
# Please Make a Dentist Appointment ASAP: Attacking IOBluetoothFamily HCI and Vendor-specific Commands

wang yu

## About me

wangyu_wy@didiglobal.com

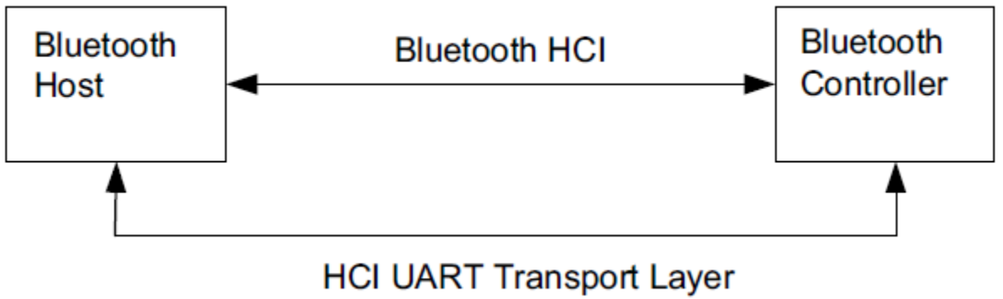
## Background of this research project

# Bluetooth Host Controller Interface (HCI) and Apple's IOBluetoothFamily Subsystem

# Bluetooth Host Controller Interface (HCI)

Standardized communication between the host stack (e.g., a PC or mobile phone OS) and the controller (the Bluetooth integrated circuit (IC)).
This standard allows the host stack or controller IC to be swapped with minimal adaptation.

https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#HCI

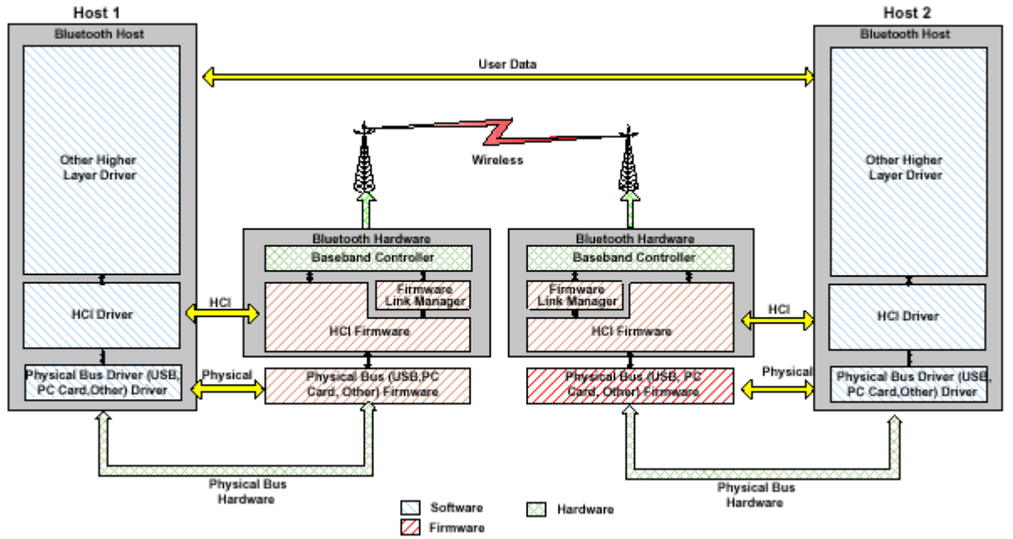


# Bluetooth HCI functional entities

The HCI is functionally broken up into three separate parts:

- HCI driver (Host)
- HCI firmware (Controller)
- Host controller transport layer

The HCI provides a uniform command method of accessing the Bluetooth hardware capabilities.




https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/hci.html

# Bluetooth HCI command group

| OGF | OpCode range | Command group |
|---|---|---|
| 0x01 | 0x0400 - 0x07FF | Link control |
| 0x02 | 0x0800 - 0x0BFF | Link policy |
| 0x03 | 0x0C00 - 0x0FFF | Host controller and baseband |
| 0x04 | 0x1000 - 0x13FF | Informational parameters |
| 0x05 | 0x1400 - 0x17FF | Status parameters |
| 0x06 | 0x1800 - 0x1BFF | Testing |
| 0x3F | 0xFC00 - 0xFFFF | Reserved for vendor-specific commands |

TI's vendor-specific commands:
https://software-dl.ti.com/simplelink/esd/simplelink_cc13x2_sdk/1.60.00.29_new/exports/docs/ble5stack/vendor_specific_guide/BLE_Vendor_Specific_HCI_Guide/hci_interface.html#vendor-specific-commands

# Vendor-specific commands (VSCs)

Only a small number of vendor-specific commands are documented.

The meaning of vendor-specific commands on different hardware platforms can be different, even if their OpCodes are the same.

- TI
  0xFC0A: HCI_EXT_ModemTestRxCmd
- Broadcom
  0xFC0A: BroadcomARMMemoryPeek/BroadcomARMMemoryPoke

# Inquiry command

*This command causes the BR/EDR controller to enter inquiry mode. Inquiry mode is used to discover other nearby BR/EDR controllers.*



| Command | OCF | Command Parameters | Return Parameters |
|---|---|---|---|
| HCI_Inquiry | 0x0001 | LAP,<br>Inquiry_Length,<br>Num_Responses | |

Bluetooth Core Specification Revision 5.2
https://www.bluetooth.com/specifications/bluetooth-core-specification/

# Parameters of inquiry command

Command parameters include:

- LAP (3 bytes)
- Inquiry_Length (1 byte)
- Num_Responses (1 byte)

*LAP:* *Size: 3 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXX | The LAP from which the inquiry access code should be derived when the inquiry procedure is made; see Assigned Numbers.<br>Range: 0x9E8B00 to 0x9E8B3F |

*Inquiry_Length:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| N = 0xXX | Maximum amount of time specified before the Inquiry is halted.<br>Range: 0x01 to 0x30<br>Time = N * 1.28 s<br>Range: 1.28 to 61.44 s |

*Num_Responses:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x00 | Unlimited number of responses. |
| 0xXX | Maximum number of responses from the Inquiry before the Inquiry is halted.<br>Range: 0x01 to 0xFF |

Bluetooth Core Specification Revision 5.2
https://www.bluetooth.com/specifications/bluetooth-core-specification/

# Implementation of hci_inquiry on different platforms

Lightweight Bluetooth (LwBT)

Linux kernel Bluetooth HCI

macOS IOBluetoothFamily kernel extension

# hci_inquiry on Lightweight Bluetooth (LwBT)

From https://github.com/lwalkera/lwBT/blob/refactoring/lwbt/hci.c#L731

to https://github.com/lwalkera/lwBT/blob/refactoring/lwbt/hci.c#L738

```
0712 err_t hci_inquiry(u32_t lap, u8_t inq_len, u8_t num_resp, ...)
0713 {
....
0730          /* Assembling command packet */
0731          p = hci_cmd_ass(p, HCI_INQUIRY_OCF, HCI_LINK_CTRL_OGF, HCI_INQUIRY_PLEN);
0732          /* Assembling cmd prameters */
0733          ((u8_t *)p->payload)[4] = lap & 0xFF;
0734          ((u8_t *)p->payload)[5] = lap >> 8;
0735          ((u8_t *)p->payload)[6] = lap >> 16;
0736          //MEMCPY(((u8_t *)p->payload)+4, inqres->cod, 3);
0737          ((u8_t *)p->payload)[7] = inq_len;
0738          ((u8_t *)p->payload)[8] = num_resp;
....
0742          return ERR_OK;
0743 }
```

# hci_inquiry on Linux kernel

From https://elixir.bootlin.com/linux/v5.10-rc3/source/net/bluetooth/hci_core.c#L1295
to https://elixir.bootlin.com/linux/v5.10-rc3/source/net/bluetooth/hci_core.c#L1298

```
1283 static int hci_inq_req(struct hci_request *req, unsigned long opt)
1284 {
1285         struct hci_inquiry_req *ir = (struct hci_inquiry_req *) opt;
1286         struct hci_dev *hdev = req->hdev;
1287         struct hci_cp_inquiry cp;
....
1294         /* Start Inquiry */
1295         memcpy(&cp.lap, &ir->lap, 3);
1296         cp.length  = ir->length;
1297         cp.num_rsp = ir->num_rsp;
1298         hci_req_add(req, HCI_OP_INQUIRY, sizeof(cp), &cp);
1299
1300         return 0;
1301 }
```

# hci_inquiry on macOS IOBluetoothFamily

Reverse engineering shows that the code is full of strange but human readable indicators.

```
1  __int64 __fastcall IOBluetoothHostController::BluetoothHCIInquiry(__int64 a1, unsigned int a2, __int64 a3, unsigned __int8
2  {
3    unsigned int v8; // er13
4    unsigned int v9; // eax
...
11   v9 = (*(*a1 + 0xC50LL))(a1, a2, 0LL, 0xFFFFLL);
12   if ( v9 )
13   {
14     v8 = v9;
15     _os_log_internal(
16       &dword_0,
17       &_os_log_default,
18       0LL,
19       "REQUIRE_NO_ERR failure: 0x%x - file: %s:%d\n",
20       v9,
21       "/AppleInternal/BuildRoot/Library/Caches/com.apple.xbs/Sources/IOBluetoothFamily_kexts/IOBluetoothFamily-7005.4.6/C"
22       "ore/Family/HCI/HostControllers/IOBluetoothHostController.cpp",
23       25509LL);
24     return v8;
25   }
26   return (*(*a1 + 0xCA0LL))(a1, a2, 0x401LL, a5, a6, "HbTbb", 0x401LL, 5LL, a3, a4, a5);
27 }
```

IOBluetoothHostController::BluetoothHCIInquiry

```
db 'HbHH',0                ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'Hb^b',0                ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'Hb^N',0                ; DATA XREF: IOBluetooth
db 'Hb^',0                 ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'Hb^bn',0               ; DATA XREF: IOBluetooth
db 'HbH',0                 ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'Hbb',0                 ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'Hb^bbH',0              ; DATA XREF: IOBluetooth
db 'Hb^bbb',0              ; DATA XREF: IOBluetooth
db 'Hb^W',0                ; DATA XREF: IOBluetooth
db 'Hb^NN',0               ; DATA XREF: IOBluetooth
db 'HbHWWHHbH',0           ; DATA XREF: IOBluetooth
db 'Hb^WWHHbH',0           ; DATA XREF: IOBluetooth
db 'HbHWW%%HHWW%%HHbbbbbbbbHHb',0
                           ; DATA XREF: IOBluetooth
db 'Hb^WW%%HHWW%%HHbbbbbbbbHHb',0
                           ; DATA XREF: IOBluetooth
db 'HbHHH',0               ; DATA XREF: IOBluetooth
                           ; IOBluetoothHostControl
db 'HbHHHHH',0             ; DATA XREF: IOBluetooth
db 'HbHbbWWWW',0           ; DATA XREF: IOBluetooth
db 'HbHHHH',0              ; DATA XREF: IOBluetooth
db 'HbHbbbWWWW',0          ; DATA XREF: IOBluetooth
db 'Hb8',0                 ; DATA XREF: IOBluetooth
0008CEDA 000000000008CEDA: __cstring:aDeviceNull
```

macOS Catalina 10.15.5 (19F96)

# Pack and unpack HCI requests

These indicators guide the work of serialization and deserialization routines PackDataList and UnpackDataList.

| Bluetooth HCI handler name | OpCode | Indicator |
|---|---|---|
| IOBluetoothHCIUserClient::DispatchHCIInquiry | 0x401 | "HbTbb" |
| IOBluetoothHCIUserClient::DispatchHCIInquiryCancel | 0x402 | "Hb" |
| IOBluetoothHCIUserClient::DispatchHCIPeriodicInquiryMode | 0x403 | "HbHHTbb" |
| IOBluetoothHCIUserClient::DispatchHCI..... | | |

- "H" is the size of the HCI request header/OpCode (2 bytes)
- "b" is the size of the HCI request body (1 byte)
- "Tbb" means LAP (3 bytes), Inquiry_Length (1 byte) and Num_Responses (1 byte)

# Summary of different HCI implementations

1. The implementation of IoT related Bluetooth HCI represented by LwBT is the simplest, and the design of macOS IOBluetoothFamily is the most complicated.

2. Complex design often means more attack surface.

3. Please also keep in mind, if the design and implementation are too simple, it usually does not mean security. For example, the design of state machine and the implementation of exception handling.

# Dive into IOBluetoothFamily

# Where to start?

We have gone through document reading, source code reading and reverse engineering of Bluetooth HCI. What should we do next?

We may also need HCI sniffer, fuzzer, code coverage methods and KASAN.

Attack surface assessment.

- From daemons to HCI
  Routine IOBluetoothHCIUserClient::SimpleDispatchWL
- From controller to HCI and daemons
  Routine IOBluetoothHostController::ProcessEventDataWL

# From Kemon to IOBluetoothFamily's sniffer and fuzzer

Kemon: An Open Source Pre and Post Callback-based Framework for macOS Kernel Monitoring
https://github.com/didi/kemon
https://www.blackhat.com/us-18/arsenal/schedule/index.html#kemon-an-open-source-pre-and-post-callback-based-framework-for-macos-kernel-monitoring-12085

The practice of kernel inline hooking:
https://www.blackhat.com/us-19/arsenal/schedule/#ksbox-a-fine-grained-macos-malware-sandbox-15059

# IOBluetoothFamily HCI request sniffer

```
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCISendRawCommand(0x62/98), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : --- raw command opcode=0xfd4c "Broadcom VSC -- LE Set Extended Scan Response Data".
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCISendRawCommand(0x62/98), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : --- raw command opcode=0xfd4b "Broadcom VSC -- LE Set Extended Advertising Data".
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCISendRawCommand(0x62/98), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : --- raw command opcode=0xfd4a "Broadcom VSC -- LE Set Extended Advertising Parameters".
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCISendRawCommand(0x62/98), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : --- raw command opcode=0xfd4d "Broadcom VSC -- LE Set Extended Advertising Enable".
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCILESetScanEnable(0xc4/196), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
```

Kemon-based IOBluetoothFamily HCI request sniffer

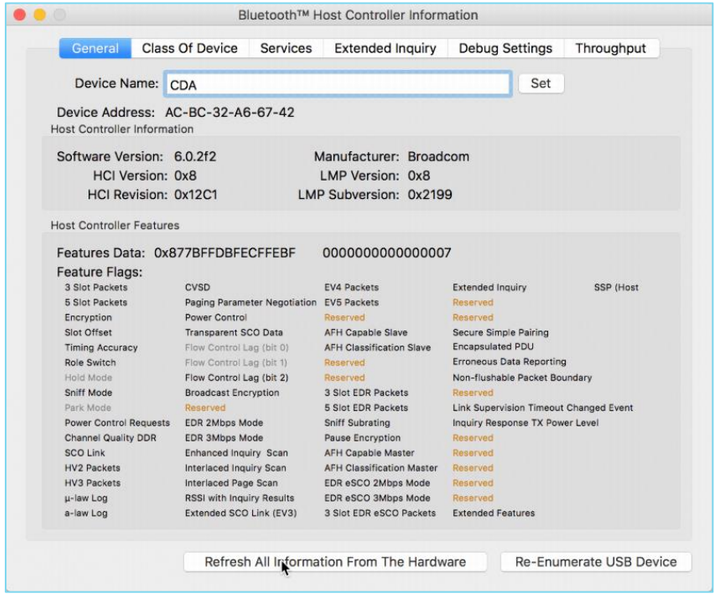
# DEMO

IOBluetoothFamily HCI request sniffer

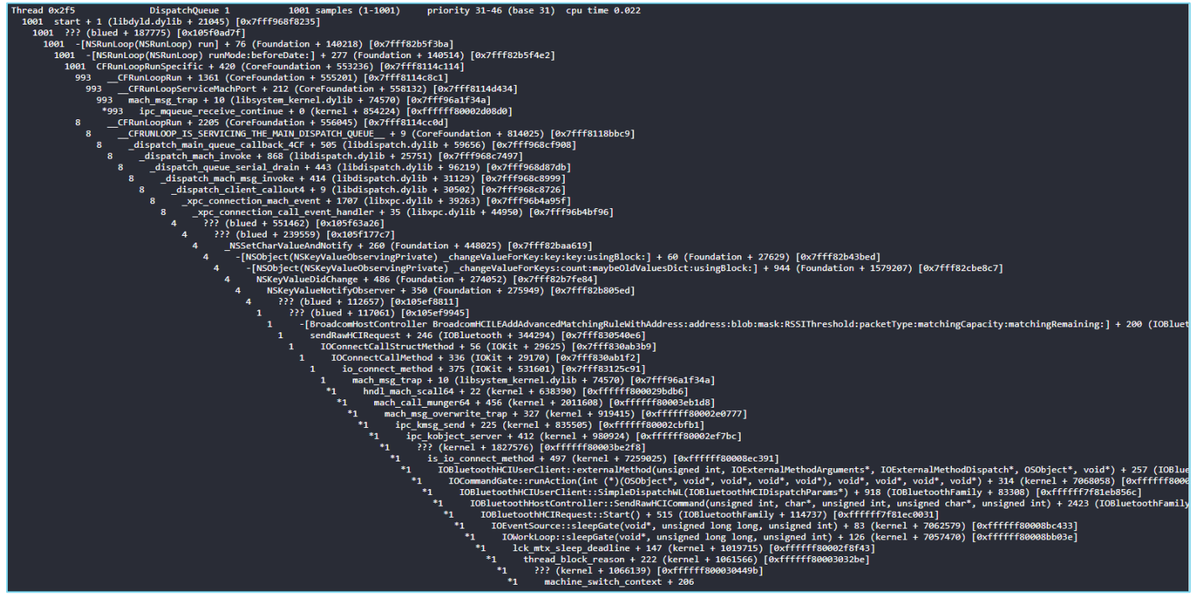
# Practice of IOBluetoothHCIUserClient::DispatchHCIChangeLocalName

## Hacking IOBluetooth

http://colemancda.github.io/2018/03/25/Hacking-IOBluetooth




BluetoothHCIWriteLocalName




Backtrace of bluetoothd

# IOBluetoothFamily HCI gadgets

Follow the calling sequence below:

1. DispatchHCIRequestCreate
2. DispatchHCIReadLocalName
3. DispatchHCIChangeLocalName
4. DispatchHCI.....
5. DispatchHCIRequestDelete

```
yuwang@Yus-MacBook-Pro-2 Desktop % ./local_name
DispatchHCIRequestCreate - request id = 0x3c.
Getting the local name of the Bluetooth adapter.
                                        -*> MEMORY DUMP <*-
+--------------------+-------------------------------------------------+------------------+
|       ADDRESS      |  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F | 0123456789ABCDEF |
| -------------------+-------------------------------------------------+------------------ |
|  0x00007ffee32318e0 | 59 75 e2 80 99 73 20 4d  61 63 42 6f 6f 6b 20 50 | Yu...s MacBook P |
|  0x00007ffee32318f0 | 72 6f 20 28 32 29 00 00  00 00 00 00 00 00 00 00 | ro (2).......... |
|  0x00007ffee3231900 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231910 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231920 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231930 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231940 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231950 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231960 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231970 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231980 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231990 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319a0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319b0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319c0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319d0 | 00 00 00 00 00 00 00 00                          | ........         |
+--------------------+-------------------------------------------------+------------------+
Changing the local name of the Bluetooth adapter.
                                        -*> MEMORY DUMP <*-
+--------------------+-------------------------------------------------+------------------+
|       ADDRESS      |  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F | 0123456789ABCDEF |
| -------------------+-------------------------------------------------+------------------ |
|  0x00007ffee32318e0 | 2a 2a 2a 2a 2a 2a 2a 2a  2a 2a 2a 2a 2a 2a 2a 2a | **************** |
|  0x00007ffee32318f0 | 2a 20 48 65 6c 6c 6f 20  4b 65 72 6e 65 6c 20 2a | * Hello Kernel * |
|  0x00007ffee3231900 | 2a 2a 2a 2a 2a 2a 2a 2a  2a 2a 2a 2a 2a 2a 2a 2a | **************** |
|  0x00007ffee3231910 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231920 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231930 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231940 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231950 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231960 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231970 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231980 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee3231990 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319a0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319b0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319c0 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffee32319d0 | 00 00 00 00 00 00 00 00                          | ........         |
+--------------------+-------------------------------------------------+------------------+
DispatchHCIRequestDelete.
```

# Built-in vendor-specific commands

| Built-in VSC handler name | OpCode | Indicator |
|---|---|---|
| DispatchHCIVendorSpecificBroadcomSetTransmitPower | 0xFC26 | "HbHb" |
| DispatchHCIVendorSpecificBroadcomBFCSuspend | 0xFCB5 | "HbH" |
| DispatchHCIVendorSpecificBroadcomBFCResume | 0xFCB6 | "HbH^n" |
| DispatchHCIVendorSpecificBroadcomBFCSetParams | 0xFCC2 | "HbbbbbbHHHbbb" |
| DispatchHCIVendorSpecificBroadcomBFCReadParams | 0xFCC3 | "Hb" |
| DispatchHCIVendorSpecificBroadcomBFCCreateConnection | 0xFCD0 | "Hb^H" |
| DispatchHCIVendorSpecificBroadcomBFCWriteScanEnable | 0xFCFA | "Hbb" |
| DispatchHCIVendorSpecificBroadcomReadLocalFirmwareInfo | 0xFCFE | "Hbb" |
| DispatchHCIVendorSpecificBroadcomBFCReadScanEnable | 0xFD01 | "Hb" |
| DispatchHCIVendorSpecificBroadcom..... | | |

# Raw vendor-specific commands

Sending raw vendor-specific commands through the HCI handler IOBluetoothHCIUserClient::DispatchHCISendRawCommand.

```
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestCreate(0x0/0), args number=4, output result size=0x4/4, output size=0x4/4.
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCISendRawCommand(0x62/98), args number=3, output result size=0x0/0, output size=0x0/0.
[Kemon.kext] : --- raw command opcode=0xfd4c "Broadcom VSC -- LE Set Extended Scan Response Data".
[Kemon.kext] : process(pid 100)=bluetoothd, routine=IOBluetoothHCIUserClient::DispatchHCIRequestDelete(0x1/1), args number=1, output result size=0x0/0, output size=0x0/0.
```

Kemon-based IOBluetoothFamily HCI request sniffer

- 0xFD4A: LE Set Extended Advertising Parameters
- 0xFD4B: LE Set Extended Advertising Data
- 0xFD4C: LE Set Extended Scan Response Data
- 0xFD4D: LE Set Extended Advertising Enable

# Firmware *gadgets*

macOS Bluetooth Analysis Suite (mBAS)
https://www.blackhat.com/us-20/arsenal/schedule/index.html#macos-bluetooth-analysis-suite-mbas-19886

- 0xFC4C: Broadcom VSC Write SoC RAM
- 0xFC4D: Broadcom VSC Read SoC RAM
- 0xFC4E: Broadcom VSC Launch SoC RAM

| Type | Handle | Decoded Packet |
|---|---|---|
| HCI COMMAND | | ▶ [FC4D] Vendor Specific Command [FC4D] - Read RAM - Address: 0x00200000 |
| HCI EVENT | | ▼ [FC4D] Command Complete - Broadcom VSC Event - Read RAM |
| | | [FC4D] Command Complete - Broadcom VSC Event - Read RAM |
| | | Parameter Length: 244 (0xF4) |
| | | Status: 0x00 - Success |
| | | Num HCI Command Packets: 0x01 |
| | | Opcode: 0xFC4D (OGF: 0x3F OCF: 0x4D) - [Vendor Specific] Vendor Specific Command [FC4D] |
| | | Broadcom VSC Event - Read RAM |
| HCI EVENT | | ▼ 00000000: 0ef4 014d fc00 bf39 eecb 18e8 76bd dd8a ...M...9....v... |
| | | 00000000: 0ef4 014d fc00 bf39 eecb 18e8 76bd dd8a ...M...9....v... |
| | | 00000010: ba03 475d 13a4 9680 0b68 5132 c051 1f2c ..G].....hQ2.Q.,<br>00000020: 25b9 05eb 6396 c554 dd15 e528 b0ee b13f %...c..T...(...?<br>00000030: eaad c6d4 ace5 8d39 753c bb6e ed92 be96 .......9u<.n....<br>00000040: cfb1 0a9a ece3 1e2a 6afc 785d 9f60 ec1d .......*j.x].`..<br>00000050: ac46 2454 b2c7 3517 0c7f 63b3 ab8f 7277 .F$T..5...c...rw<br>00000060: 3945 be3c 3cc0 289b a214 09f2 8209 d6b1 9E.<<.(.........<br>00000070: 1e12 8c87 a006 7baa 2d38 4504 53f5 e264 ......{.-8E.S..d<br>00000080: b437 d737 4c84 cfcc 3b41 b90a 1066 7352 .7.7L...;A...fsR<br>00000090: 1e38 4171 855b 3356 661f 5b7f 52eb d03f .8Aq.[3Vf.[.R..?<br>000000a0: 526d 2189 635e 7863 672a 3834 72b0 08c3 Rm!.c^xcg*84r...<br>000000b0: d4d2 1254 82aa 1864 cd86 68b0 5768 0275 ...T...d..h.Wh.u<br>000000c0: db84 bf82 b3f8 1d78 0884 3ede 2335 3859 .......x..>.#58Y<br>000000d0: 1d9f 4a3c 2804 e9b5 1ef8 7356 0770 6e68 ..J<(.....sV.pnh<br>000000e0: 344d cd8a 5155 6d68 eaad e65a aa18 6962 4M..QUmh...Z..ib<br>000000f0: daec aea5 f6d1 ......
| HCI COMMAND | | ▶ [FC4D] Vendor Specific Command [FC4D] - Read RAM - Address: 0x002000F0 |
| HCI EVENT | | ▼ [FC4D] Command Complete - Broadcom VSC Event - Read RAM |
| | | [FC4D] Command Complete - Broadcom VSC Event - Read RAM |
| | | Parameter Length: 244 (0xF4) |
| | | Status: 0x00 - Success |
| | | Num HCI Command Packets: 0x01 |
| | | Opcode: 0xFC4D (OGF: 0x3F OCF: 0x4D) - [Vendor Specific] Vendor Specific Command [FC4D] |
| | | Broadcom VSC Event - Read RAM |
| HCI EVENT | | ▼ 00000000: 0ef4 014d fc00 5903 8370 e43e 731f a9b1 ...M..Y..p.>s... |
| | | 00000000: 0ef4 014d fc00 5903 8370 e43e 731f a9b1 ...M..Y..p.>s... |
| | | 00000010: 4ddd ce2a 3458 4ffa bc2c 8573 3148 e153 M..*4XO..,.s1H.S<br>00000020: 628f 0358 96e9 6cb9 81a2 e26f 8649 a2f5 b..X..l....o.I..<br>00000030: 44a7 b5c6 2e8c 5af6 ed51 0ef1 d92a 7e29 D.....Z..Q...*~)<br>00000040: 7fde 8854 9425 2b5c 7175 d6b0 d4cf 6aa4 ...T.%+\qu....j.<br>00000050: b2a0 7fd6 eb78 fe86 f867 c223 9a0f d64b .....x...g.#...K<br>00000060: 7aed 6db7 0b9a c2e9 27be a325 2692 4b1f z.m.....'..%&.K.<br>00000070: 273c ceea c2d0 2edc 1bfb e8d6 4c87 af7e '<.........L..~<br>00000080: 2bbc 457e 168e 84ab 6a2b 8058 3d04 fa05 +.E~....j+.X=...<br>00000090: 0e08 5210 0ba4 d1eb fad0 95cc 1694 6521 ..R...........e!<br>000000a0: 12d7 9682 ecae 6ef7 6ab6 7cc2 973d 4580 ......n.j.\|..=E.<br>000000b0: 9277 42b7 261b 25ce 159b 248a aab6 dbd1 .wB.&.%...$.....<br>000000c0: 2fb6 efb4 067a f375 ffb4 7682 5069 501a /....z.u..v.PiP.<br>000000d0: fe96 8dc2 11a6 fc48 85d4 be75 2bae 1e59 .......H...u+..Y<br>000000e0: c299 cfd6 bd2f 0034 33a0 ddcc 2b91 6eec ...../.43...+.n.<br>000000f0: c68a ff8c 8a95 ......
| HCI COMMAND | | ▶ [FC4D] Vendor Specific Command [FC4D] - Read RAM - Address: 0x002001E0 |
| HCI EVENT | | ▶ [FC4D] Command Complete - Broadcom VSC Event - Read RAM |

# Fuzzing

Passive fuzzing based on IOBluetoothFamily HCI request sniffer, and active fuzzing based on the gadgets like mBAS.

Combining the two fuzzing methods.

IOBluetoothFamily HCI code coverage analysis based on Kemon's kernel inline hook engine.

# DEMO

IOBluetoothFamily subsystem fuzzer
on macOS 11.0 Big Sur

# Summary of macOS Bluetooth engineering

1. IOBluetoothFamily command and event sniffer.
2. IOBluetoothFamily passive and active fuzzer.
3. Kemon-based code coverage analysis and KASAN solution.

# IOBluetoothFamily HCI
# Latest Zero-day Vulnerability Case Studies

# Continual improvement

*In recent years, binary auditing and fuzzing against macOS Bluetooth kernel extensions such as IOBluetoothFamily have never stopped. We can also prove this from the output of IDA Pro Hex-Rays.*

*IOBluetoothHCIUserClient::ValidParameters has increased tenfold.*

```
310             }
311             LOBYTE(v14) = v14 + 1;
312           }
313           while ( v14 < a3 );
314         }
315         result = v29 != 0;
316       }
317     }
318   }
319   return result;
320 }
0001196F __ZN24IOBluetoothHCIUserClient15ValidParameters
```

macOS High Sierra 10.13.5 (17F77)
IOBluetoothHCIUserClient::ValidParameters

```
3553       v12);
3554     v34 = *(v33 + 216);
3555     if ( v34 )
3556     {
3557       v35 = strlen(v32);
3558       (*(*v34 + 2240LL))(v34, 250LL, v32, v35);
3559     }
3560     IOFree(v32, 511LL);
3561   }
3562   return 0;
3563 }
0001197F __ZN24IOBluetoothHCIUserClient15ValidParameters
```

macOS Catalina 10.15.5 (19F96)
IOBluetoothHCIUserClient::ValidParameters

# Binary auditing and vulnerability hunting

So far, the total number of kernel vulnerabilities I have reported is twenty-three.

The types of vulnerabilities include:

1. Uninitialized memory dereference
2. Kernel information disclosure
3. Heap data out-of-bounds access
4. Arbitrary memory access
5. Use-After-Free/Double free caused by race condition
6. Security Update 2020-002 patch bypass, etc.

# CVE IDs

CVE-2020-3892, CVE-2020-3893, CVE-2020-3905, CVE-2020-3907, CVE-2020-3908, CVE-2020-3912, CVE-2020-9779 and CVE-2020-9853
https://support.apple.com/en-us/HT211100

CVE-2020-9831
https://support.apple.com/en-us/HT211170

CVE-2020-9928 and CVE-2020-9929
https://support.apple.com/en-us/HT211289

Apple Product Security Follow-up IDs: 733637811, 734810171, 733658775, 733660424, 735099265, 735911525, 735912349, 735912935, 737656122, etc.

# Case #1 - kernel heap out-of-bounds read

CVE-2020-3907:
IOBluetoothHostController::BluetoothHCIWriteCurrentIACLAP (OpCode 0xC3A)
Out-of-bounds Read Vulnerability

CVE-2020-3908:
IOBluetoothHostController::BluetoothHCIWriteStoredLinkKey (OpCode 0xC11)
Out-of-bounds Read Vulnerability

Patched via Security Update 2020-002
https://support.apple.com/en-us/HT211100

# Case study of CVE-2020-3907

```
(lldb) memory read 0xffffff8055dca894 -c0x200
0xffffff8055dca894: 3a 0c fe ff ff ff ff ff ff ff ff ff ff ff ff ff  :.??????????????
0xffffff8055dca8a4: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ????????????????
0xffffff8055dca8b4: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ????????????????
......
0xffffff8055dca924: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ????????????????
0xffffff8055dca934: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ????????????????
0xffffff8055dca944: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ????????????????
0xffffff8055dca954: ff ff ff ff be ad de ef be ad de ef be ad de ef  ????????럭?럭??
0xffffff8055dca964: be ad de ef be ad de ef be ad de ef be ad de ef  ???럭?럭?럭??
0xffffff8055dca974: be ad de f1 57 00 00 00 00 00 00 d7 02 00 00 00  ????W......?....
0xffffff8055dca984: 00 00 00 96 eb 0a 1a 00 00 00 00 00 00 00 00 00  ....?..........
......

(lldb) bt
frame #0: 0xffffff7f91afabf6 IOBluetoothFamily`IOBluetoothHostController::SendHCIRequestFormatted + 984
frame #1: 0xffffff7f91b0a554 IOBluetoothFamily`IOBluetoothHostController::BluetoothHCIWriteCurrentIACLAP + 214
frame #2: 0xffffff7f91abf710 IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 1166
......
```

# Case study of CVE-2020-3908

```
(lldb) memory read 0xffffff8033ff0894 -c0x200
0xffffff8033ff0894: 11 0c fb 17 00 05 00 28 20 08 00 33 02 04 00 00  ..?....( ..3....
0xffffff8033ff08a4: 00 00 00 00 00 00 00 00 00 00 21 00 00 00 de ad  ..........!...
0xffffff8033ff08b4: be ef de ad be ef 00 00 00 00 00 00 00 00 00 00  ??..........??
0xffffff8033ff08c4: 00 00 00 00 00 00 6a 6f 72 70 6c 2e 69 68 de ad  ......jorpl.ih
0xffffff8033ff08d4: be ef de ad be ef de ad be ef 00 09 02 04 00 00  ??......????????
0xffffff8033ff08e4: 00 00 00 00 00 00 00 00 00 00 21 00 00 00 00 00  ..........!.....
0xffffff8033ff08f4: 00 00 00 00 00 00 00 00 00 00 21 00 00 00 de ad  ..........!...
0xffffff8033ff0904: be ef de ad be ef ff ff ff 80 31 da ed 80 11 17  ??...??1.?????
0xffffff8033ff0914: 47 b3 e2 8c 55 98 4d 43 fe d8 c8 82 2a d9 00 00  G??.U.MC???.*?..
0xffffff8033ff0924: 00 00 00 00 00 00 00 00 00 00 21 00 00 00 00 00  ..........!.....
0xffffff8033ff0934: 00 00 00 00 00 00 00 00 00 00 21 00 00 00 00 00  ..........!.....
......

(lldb) bt
frame #0: 0xffffff7f91afabf6 IOBluetoothFamily`IOBluetoothHostController::SendHCIRequestFormatted + 984
frame #1: 0xffffff7f91b0875d IOBluetoothFamily`IOBluetoothHostController::BluetoothHCIWriteStoredLinkKey + 269
frame #2: 0xffffff7f91abf766 IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 1252
......
```

# Summary of case #1

1. The root cause of these vulnerabilities is the lack of effective verification of user inputs, which leads to out-of-bounds reading.

2. The number of bytes out-of-bounds will be limited to 0x200 bytes.

3. This type of vulnerability can be easily captured by KASAN.

4. Similar vulnerabilities include:
   CVE-2020-9779, CVE-2020-9831, CVE-2020-9853, etc.

# Case #2 - kernel heap out-of-bounds access

CVE-2020-3912:
IOBluetoothHCIUserClient::DispatchHCISendRawACLData (sRoutine Index 0x63)
Out-of-bounds Access Vulnerability

Patched via Security Update 2020-002
https://support.apple.com/en-us/HT211100

# Roberto Paleari and Aristide Fattori

As far as I know, this is the third time in history that the same routine has been found vulnerable.

The first time can be identified as CVE-2014-8837.
Time to fill OS X (Blue)tooth: Local privilege escalation vulnerabilities in Yosemite
http://randomthoughts.greyhats.it/2015/01/osx-bluetooth-lpe.html
https://joystick.artificialstudios.org/time-to-fill-os-x-bluetooth-local/
https://support.apple.com/en-us/HT204244
https://www.exploit-db.com/exploits/35773

# Moony Li

*The second time can be identified as CVE-2015-3787.*
*The Bluetooth subsystem in Apple OS X before 10.10.5 allows remote attackers to cause a denial of service via malformed Bluetooth ACL packets.*
https://support.apple.com/en-us/HT205031

```
(lldb) bt
frame #0: 0xffffff80025f4e67 kernel.development`Debugger(message=<unavailable>) + 759
frame #1: 0xffffff80024e4ed1 kernel.development`panic(str=<unavailable>) + 209 at debug.c:383
frame #2: 0xffffff8002a3bb62 kernel.development`OSMetaClass::serialize + 18
frame #3: 0xffffff7f83ce9c64 IOBluetoothFamily`IOBluetoothHCIController::TransferACLPacketToHW + 1400
frame #4: 0xffffff7f83d1459e IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 830
frame #5: 0xffffff8002ab891e kernel.development`IOCommandGate::runAction + 462
frame #6: 0xffffff7f83d14245 IOBluetoothFamily`IOBluetoothHCIUserClient::externalMethod + 203
frame #7: 0xffffff8002ae2443 kernel.development`is_io_connect_method + 499
......
```

# Case study of CVE-2020-3912

```
(lldb) di -p
kernel`bcopy:
->  0xffffff8017998082 <+18>: rep   movsb      (%rsi), %es:(%rdi)
    0xffffff8017998084 <+20>: retq
    0xffffff8017998085 <+21>: addq  %rcx, %rdi

(lldb) register read rdi rsi rcx
General Purpose Registers:
       rdi = 0xffffff80560db7f0
       rsi = 0xffffff805699b000
       rcx = 0x0000000000003081

(lldb) bt
thread #1, stop reason = signal SIGSTOP
frame #0: 0xffffff8017998082 kernel`bcopy + 18
frame #1: 0xffffff8017c555a4 kernel`memmove + 20
frame #2: 0xffffff7f98e1cc00 IOBluetoothFamily`IOBluetoothMemoryBlock::writeBytes + 60
frame #3: 0xffffff7f98df93c3 IOBluetoothFamily`IOBluetoothHCIUserClient::DispatchHCISendRawACLData + 191
frame #4: 0xffffff7f98df129e IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 2886
......
```

# Summary of case #2

1. It is true that some complex routines will be repeatedly found vulnerable.
   - Routine AirPort_Athr5424::setSCAN_REQ, September 2007
     http://www.uninformed.org/?v=all&a=37&t=txt
   - Routine AppleBCMWLANCore::setSCAN_REQ, CVE-2020-9834, May 2020
     https://www.blackhat.com/us-20/briefings/schedule/#dive-into-apple-iofamilyv-20023

2. For such complex routines, it is better to have complete test cases to ensure that all branches can be covered.

3. Sometimes security patch can be bypassed. Learning the implementation of patches is usually meaningful and helpful.

# Case #3 - arbitrary memory access

CVE-2020-9929:
IOBluetoothHCIUserClient::DispatchHCIEnhancedSetupSynchronous-Connection (OpCode 0x43D), and
IOBluetoothHCIUserClient::DispatchHCIEnhancedAcceptSynchronous-ConnectionRequest (OpCode 0x43E)
Arbitrary Memory Access Vulnerabilities

Patched via Security Update 2020-004
https://support.apple.com/en-us/HT211289

# It could be changed

Reverse engineering shows that starting with macOS Catalina, the indicator of the routine IOBluetoothHCIUserClient::DispatchHCIEnhancedAcceptSynchronous-ConnectionRequest has been changed from "HbHWWHHbH" to "HbHWW%%HHWW%%HHbbbbbbbbHHb".

```
db 'HbHWW%%HHWW%%HHbbbbbbbbHHb',0
                        ; DATA XREF: IOBluetooth
db 'Hb^WW%%HHWW%%HHbbbbbbbbHHb',0
                        ; DATA XREF: IOBluetooth
db 'HbHHH',0            ; DATA XREF: IOBluetooth
                        ; IOBluetoothHostControl
db 'HbHHHHH',0          ; DATA XREF: IOBluetooth
db 'HbHbbWWWW',0        ; DATA XREF: IOBluetooth
db 'HbHHHH',0           ; DATA XREF: IOBluetooth
db 'HbHbbbWWWW',0       ; DATA XREF: IOBluetooth
db 'Hb8',0              ; DATA XREF: IOBluetooth
0008CEDA 000000000008CEDA: __cstring:aDeviceNull
```

macOS Catalina 10.15.5 (19F96)

# An IDA a day keeps the girls away

'%' is not an intuitive indicator, what does it represent?
In addition to '%', what other special indicators/symbols are supported?
Does the new indicator mean that the controller has changed?
Does such a modification have compatibility issues?
.....

Reverse engineering shows that the parser will read 5 bytes of data from the address submitted by the user, but the address is not verified before reading!

# Case study of CVE-2020-9929

```
(lldb) di -p
IOBluetoothFamily`PackDataList:
->  0xffffff7f81a20688 <+1013>: movb   -0x1(%rdi,%rax), %cl
    0xffffff7f81a2068c <+1017>: movb   %cl, (%rsi)
    0xffffff7f81a2068e <+1019>: incq   %rsi

(lldb) register read rdi r10
General Purpose Registers:
       rdi = 0xdeadcafedeadbeef
       r10 = 0xffffff7f81a7bf70  "%%HHWW%%HHbbbbbbbbbHHb"

(lldb) bt
thread #1, stop reason = EXC_BAD_INSTRUCTION (code=13, subcode=0x0)
frame #0: 0xffffff7f81a20688 IOBluetoothFamily`PackDataList + 1013
frame #1: 0xffffff7f81a457f8 IOBluetoothFamily`IOBluetoothHostController::SendHCIRequestFormatted + 1408
frame #2: 0xffffff7f81a520db IOBluetoothFamily`IOBluetoothHostController::
                                                BluetoothHCIEnhancedAcceptSynchronousConnectionRequest + 1139
frame #3: 0xffffff7f81a007d2 IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 2828
......
```

# Apple SDK

```
@function   IOBluetoothPackData
@abstract   Packs a variable amount of parameters into a buffer according to a printf-style format string.
@discussion Supported format characters:
            '1' Ptr to 1 byte of data.
            '2' Ptr to 2 bytes of data.
            '@' (shift-2) Ptr to 2 bytes of data to byte reverse.
            '3' Ptr to 3 bytes of data.
            '#' (shift-3) Ptr to 3 bytes of data to byte reverse.
            '4' Ptr to 4 bytes of data.
            '$' (shift-4) Ptr to 4 bytes of data to byte reverse.
            '5' Ptr to 5 bytes of data.
            '%' (shift-5) Ptr to 5 bytes of data to byte reverse.
            '6' Ptr to 6 bytes of data.
            '^' (shift-6) Ptr to 6 bytes of data to byte reverse.
            '7' Ptr to 6 bytes of data.
            '&' (shift-7) Ptr to 7 bytes of data to byte reverse.
            '8' Ptr to 6 bytes of data.
            '*' (shift-8) Ptr to 8 bytes of data to byte reverse.
            '9' Ptr to 6 bytes of data.
            '(' (shift-9) Ptr to 9 bytes of data to byte reverse.
            ......
```

# Summary of case #3

1. New features always mean new attack surface.

2. The change in the Pack and/or Unpack indicator of the HCI handlers actually indicates that the Bluetooth controller is changing, which may mean compatibility issues, and it may also mean potential attack surfaces.

3. The combination of reverse engineering and Apple SDK means a better life.

# Case #4 - kernel heap out-of-bounds access

Apple Product Security Follow-up ID 726569660:
IOBluetoothFamily`ParseVendorSpecificCommand and
Vendor-specific Command 0xFCE9 (Broadcom LE Meta VSC)
Out-of-bounds Access Vulnerability

# Confusion from LwBT exception handling

## *Lightweight Bluetooth (LwBT)'s hci.c*

```
243 //TODO: XXX??? DO WE SAVE NUMACL PACKETS COMPLETED IN LINKS LIST??
...
451 //TODO: MASTER SLAVE SWITCH??
...
597 case HCI_HARDWARE_ERROR:
598         LWIP_DEBUGF(HCI_EV_DEBUG, ("hci_event_input: Hardware Error\n"));
599         LWIP_DEBUGF(HCI_EV_DEBUG, ("Hardware_code: 0x%x\n\n", ((u8_t *)p->payload)[0]));
600         hci_reset();
601         //TODO: IS THIS FATAL??
602         break;
...
647 case HCI_DATA_BUFFER_OVERFLOW:
648         LWIP_DEBUGF(HCI_EV_DEBUG, ("hci_event_input: Data Buffer Overflow\n"));
649         LWIP_DEBUGF(HCI_EV_DEBUG, ("Link_Type: 0x%x\n", ((u8_t *)p->payload)[0]));
650         //TODO: IS THIS FATAL????
651         break;
```

# Exception handling and state machine

Exception handling and state machine are excellent fuzzing targets.

Attack surface assessment.

- From controller to HCI and daemons

  Routine IOBluetoothHostController::ProcessEventDataWL

# Vendor-specific command 0xFCE9 and customer specific features

```
HCI COMMAND [FCE9] Vendor Specific Command - Clear Matching Rules
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event - Clear Matching Rule

HCI COMMAND [FCE9] Vendor Specific Command - Add Matching Rule with Address for Type: 9
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event - Add Matching Rule with Address

HCI COMMAND [FCE9] Vendor Specific Command - Add Matching Rule with Address for Type: 5
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event - Add Matching Rule with Address

HCI COMMAND [FCE9] Vendor Specific Command - LE Meta VSC: LE Enable Customer Specific Feature: 0x1089
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event -
                                      LE Meta VSC: LE Enable Customer Specific Feature

HCI COMMAND [FCE9] Vendor Specific Command - LE Meta VSC: LE Adv Packet Filter Content Feature Section
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event -
                                      LE Meta VSC: LE Adv Packet Content Filter Feature Section

HCI COMMAND [FCE9] Vendor Specific Command - LE Meta VSC: LE Adv Packet Filter Service UUID
HCI EVENT   [FCE9] Command Complete - Broadcom VSC Event -
                                      LE Meta VSC: LE Adv Packet Content Filter Service UUID
```

lldb

```
kernel.development`bcopy:
->  0xffffff80157a4096 <+22>: rep    movsq      (%rsi), %es:(%rdi)
    0xffffff80157a4099 <+25>: movq   %rdx, %rcx
    0xffffff80157a409c <+28>: andq   $0x7, %rcx
    0xffffff80157a40a0 <+32>: rep    movsb      (%rsi), %es:(%rdi)
Target 0: (kernel.development) stopped.
(lldb) bt
* thread #6, name = '0xffffff804c476758', queue = '0x0', stop reason = EXC_BAD_ACCESS (code=1, address=0x3f467000)
  * frame #0: 0xffffff80157a4096 kernel.development`bcopy + 22
    frame #1: 0xffffff8015acb824 kernel.development`memmove(dst=0xffffff803aa20000, src=<unavailable>, ulen=<unavailable>) at loose_ends.c:578:2 [opt]
    frame #2: 0xffffff7f988c66a1 IOBluetoothFamily`ParseVendorSpecificCommand + 1229
    frame #3: 0xffffff7f988c4eb4 IOBluetoothFamily`ParseHCIEvent + 1144
    frame #4: 0xffffff7f988f6540 IOBluetoothFamily`IOBluetoothHostController::ProcessEventDataWL(unsigned char*, unsigned int, unsigned int) + 2696
    frame #5: 0xffffff7f9890e28b IOBluetoothFamily`BroadcomBluetoothHostController::ProcessEventDataWL(unsigned char*, unsigned int, unsigned int) + 349
    frame #6: 0xffffff7f988f5aae IOBluetoothFamily`IOBluetoothHostController::ProcessEventDataAction(IOBluetoothHostController*, unsigned char*, unsigned int, unsigned int) + 18
    frame #7: 0xffffff7f988ee644 IOBluetoothFamily`IOBluetoothHostController::DesyncIncomingDataAction(IOBluetoothHostController*, int (*)(IOBluetoothHostController*, unsigned char*, unsigned int, unsigne
d int), void*, unsigned int, unsigned int) + 92
    frame #8: 0xffffff7f988e24df IOBluetoothFamily`IOWorkQueue::executeWorkCall(IOWorkQueueCall*) + 51
    frame #9: 0xffffff7f988e248e IOBluetoothFamily`IOWorkQueue::checkForWork() + 42
    frame #10: 0xffffff7f988e2504 IOBluetoothFamily`IOWorkQueue::processWorkCallFromSeparateThread(IOWorkQueueCall*) + 30
    frame #11: 0xffffff7f988e277a IOBluetoothFamily`IOWorkQueue::ThreadCallMain(void*, int) + 126
    frame #12: 0xffffff8015920567 kernel.development`call_continuation + 23
(lldb)
```

Python

```
[ o d I t s z a P c ]
[                   ]
RIP FFFFFF80157A4096 | .@z..... | => `bcopy + 0x16`
RAX FFFFFFFFB5B925A | Z.[..... |
RBX FFFFFF803AA20000 | ...:.... |
RBP FFFFFF920EE9B7F0 | ........ | => 0xFFFFFF920EE9B860 => 0xFFFFFF920EE9B8D0 => 0xFFFFFF920EE9BC20 => 0xFFFFFF920EE9BED0 => 0xFFFFFF920EE9BEE0 => 0xFFFFFF920EE9BF20 => 0xFFFFFF920EE9BF40 => 0xFFFFFF920E>
RSP FFFFFF920EE9B7D8 | ........ | => 0xFFFFFF8015ACB824 => `memmove + 0x14`
RDI FFFFFF803AA20258 | X..:.... |
RSI FFFFFF803F466FFE | .oF?.... |
RDX 0000000000000800 | ........ |
RCX 00000000000000B5 | ........ |
R8  FFFFFF7F988D8AC4 | ........ | => `UnpackDataList + 0x63A`
R9  FFFFFF7F988D8B74 | t....... | => `UnpackDataList + 0x6EA`
R10 0000000000000001 | ........ |
R11 FFFFFF920EE9B837 | 7....... | => "

                                  "
R12 FFFFFF803F466DA6 | .mF?.... |
R13 0000000000000001 | ........ |
R14 FFFFFF920EE9B90C | ........ |
R15 FFFFFF920EE9B93E | >....... | => "
                                      IOBluetoothHostController::ProcessEventDataWL -- Found request"
CS  0000  DS  0000
ES  n/a   FS  FFFF0000
```

# Summary of case #4

1. The design of state machine and the implementation of exception handling are different for each operating system. Some designs and implementations do not fully comply with the official Bluetooth specification.

2. A large number of vendor-specific commands are not documented.

3. What does it mean when an undocumented vendor-specific command superimposes a state machine that does not conform to the official specification?

# Case #5 - uninitialized memory dereference

CVE-2020-3892:
IOBluetoothHCIUserClient::SimpleDispatchWL
Uninitialized Kernel Memory Dereference Vulnerability

Patched via Security Update 2020-002
https://support.apple.com/en-us/HT211100

# The pattern of the vulnerability

*I discovered this vulnerability within twenty minutes of starting to reverse the IOBluetoothFamily kernel extension.*

*This is not because I am lucky, nor because I am good at macOS kernel reverse engineering, but because I made the same mistake several years ago.*

# Confusion from `if/else` statements

## What should we cover in the `else` branch?

```
if (condition_one || (condition_two &&
     0xc3 == condition_three) || ((condition_four &&
     !condition_five && 0xdeadbeef != condition_six) ||
     !inline_routine_one() && 0x7 <= condition_seven)) {

     ......

} else {

     ...... /* What should we cover here? */

}
```

# The simplest case

We need to cover in the `else` branch, including:

- !condition_one && condition_two
- condition_one && !condition_two
- !condition_one && !condition_two

```
if (condition_one && condition_two) {

    ......

} else {

    ...... /* Three cases have to be covered here */

}
```

# Case study of CVE-2020-3892

## Again, what should we cover in the `else` branch?

```
140             local_buffer = IOMalloc(input_length);
141             tmp_routine = routine;
142             *(this + 8 * index + 0x120) = local_buffer;
143             if ( local_buffer && routine->routine_input_buffer[index] )// "Do one thing and do that well." – Linux kernel coding style
144             {
145               bzero(local_buffer, routine->routine_input_length[index]);
146
147               local_descriptor = IOMemoryDescriptor::withAddressRange(
148                                    routine->routine_input_buffer[index],
149                                    routine->routine_input_length[index],
150                                    3uLL,
151                                    *(this + 0xE0),
152                                    v28);
153               *(this + 8 * index + 0xE8) = local_descriptor;
154               if ( !local_descriptor )
155               {
156                 ret_value = 0xE00002BD;
157                 goto LABEL_66;
158               }
```

macOS High Sierra 10.13.5 (17F77)
IOBluetoothHCIUserClient::SimpleDispatchWL and IOBluetoothHCIUserClient::ValidParameters

# Corner cases matter

*Unfortunately, there is almost nothing.*

```
140             local_buffer = IOMalloc(input_length);
141             tmp_routine = routine;
142             *(this + 8 * index + 0x120) = local_buffer;
143             if ( local_buffer && routine->routine_input_buffer[index] )// "Do one thing and do that well." - Linux kernel coding style
144             {
145               bzero(local_buffer, routine->routine_input_length[index]);
146
147               local_descriptor = IOMemoryDescriptor::withAddressRange(
148                                    routine->routine_input_buffer[index],
149                                    routine->routine_input_length[index],
150                                    3uLL,
151                                    *(this + 0xE0),
152                                    v28);
153               *(this + 8 * index + 0xE8) = local_descriptor;
154               if ( !local_descriptor )
155               {
156                 ret_value = 0xE00002BD;
157                 goto LABEL_66;
158               }
```

```
184             }
185             else
186             {
187               *(this + 8 * index + 0xE8) = 0LL;
188             }
189             if ( ++index >= args_number )
190               goto LABEL_36;
191           }
192         }
```

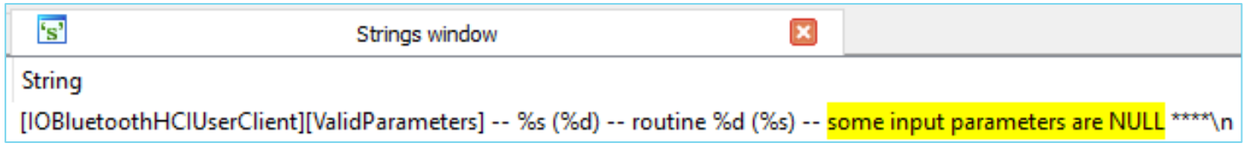
macOS High Sierra 10.13.5 (17F77)
IOBluetoothHCIUserClient::SimpleDispatchWL and IOBluetoothHCIUserClient::ValidParameters

# Uninitialized memory dereference

This means that attackers can bypass the heap initialization process of routine IOBluetoothHCIUserClient::SimpleDispatchWL by providing only the input length.

These uninitialized heaps are then passed to all HCI handlers!

Security Update 2020-002
https://support.apple.com/en-us/HT211100




macOS Catalina 10.15.5 (19F96)
IOBluetoothHCIUserClient::SimpleDispatchWL

# Summary of case #5

1. CVE-2020-3892 has been hidden in plain sight for a long time and affects all macOS Bluetooth HCI handlers.

2. Some traditional fuzzing methods are difficult to find this type of vulnerability.

3. "Do one thing and do that well." - Linux kernel coding style (Burn it)
   https://www.kernel.org/doc/html/v4.10/process/coding-style.html

# Case #6 - race condition

CVE-2020-3905:
IOBluetoothHCIUserClient::DispatchHCIWriteEncryptionMode (OpCode 0xC22)
Kernel Object Race Condition Vulnerability

Patched via Security Update 2020-002, but this patch can be bypassed.
https://support.apple.com/en-us/HT211100

CVE-2020-9928:
IOBluetoothFamily Kernel Object Race Condition Vulnerability
Triggered by Mixed HCI Commands

Patched via Security Update 2020-004
https://support.apple.com/en-us/HT211289

# A call stack from "Hacking IOBluetooth" (selected)

```
Thread 0x2f5        DispatchQueue 1        1001 samples (1-1001)        priority 31-46 (base 31) cpu time 0.022
 8    _xpc_connection_call_event_handler + 35 (libxpc.dylib + 44950) [0x7fff96b4bf96]
 4    ??? (blued + 551462) [0x105f63a26]
 4    ??? (blued + 239559) [0x105f177c7]
 4    _NSSetCharValueAndNotify + 260 (Foundation + 448025) [0x7fff82baa619]
 4    -[NSObject(NSKeyValueObservingPrivate) _changeValueForKey:key:key:usingBlock:] + 60 (Foundation + 27629) [0x7fff82b43bed]
 4    -[NSObject(NSKeyValueObservingPrivate) _changeValueForKeys:count:maybeOldValuesDict:usingBlock:] + 944 (Foundation + 1579207) [0x7fff82cbe8c7]
 4    NSKeyValueDidChange + 486 (Foundation + 274052) [0x7fff82b7fe84]
 4    NSKeyValueNotifyObserver + 350 (Foundation + 275949) [0x7fff82b805ed]
 4    ??? (blued + 112657) [0x105ef8811]
 1    ??? (blued + 117061) [0x105ef9945]
 1    -[BroadcomHostController BroadcomHCILEAddAdvancedMatchingRuleWithAddress:address:blob:mask:RSSIThreshold:packetType:matchingCapacity:matchingRemaining:] + 200
 1    sendRawHCIRequest + 246 (IOBluetooth + 344294) [0x7fff830540e6]
 1    IOConnectCallStructMethod + 56 (IOKit + 29625) [0x7fff830ab3b9]
 1    IOConnectCallMethod + 336 (IOKit + 29170) [0x7fff830ab1f2]
 1    io_connect_method + 375 (IOKit + 531601) [0x7fff83125c91]
 1    mach_msg_trap + 10 (libsystem_kernel.dylib + 74570) [0x7fff96a1f34a]
*1    hndl_mach_scall64 + 22 (kernel + 638390) [0xffffff800029bdb6]
*1    mach_call_munger64 + 456 (kernel + 2011608) [0xffffff80003eb1d8]
*1    mach_msg_overwrite_trap + 327 (kernel + 919415) [0xffffff80002e0777]
*1    ipc_kmsg_send + 225 (kernel + 835505) [0xffffff80002cbfb1]
*1    ipc_kobject_server + 412 (kernel + 980924) [0xffffff80002ef7bc]
*1    ??? (kernel + 1827576) [0xffffff80003be2f8]
*1    is_io_connect_method + 497 (kernel + 7259025) [0xffffff80008ec391]
*1    IOBluetoothHCIUserClient::externalMethod(unsigned int, IOExternalMethodArguments*, IOExternalMethodDispatch*, OSObject*, void*) + 257
*1    IOCommandGate::runAction(int (*)(OSObject*, void*, void*, void*, void*), void*, void*, void*, void*) + 314 (kernel + 7068058) [0xffffff80008bd99a]
*1    IOBluetoothHCIUserClient::SimpleDispatchWL(IOBluetoothHCIDispatchParams*) + 918 (IOBluetoothFamily + 83308) [0xffffff7f81eb856c]
*1    IOBluetoothHostController::SendRawHCICommand(unsigned int, char*, unsigned int, unsigned char*, unsigned int) + 2423 (IOBluetoothFamily + 327391) [0xffffff7f81ef3edf]
*1    IOBluetoothHCIRequest::Start() + 515 (IOBluetoothFamily + 114737) [0xffffff7f81ec0031]
*1    IOEventSource::sleepGate(void*, unsigned long long, unsigned int) + 83 (kernel + 7062579) [0xffffff80008bc433]
*1    IOWorkLoop::sleepGate(void*, unsigned long long, unsigned int) + 126 (kernel + 7057470) [0xffffff80008bb03e]
*1    lck_mtx_sleep_deadline + 147 (kernel + 1019715) [0xffffff80002f8f43]
*1    thread_block_reason + 222 (kernel + 1061566) [0xffffff80003032be]
*1    ??? (kernel + 1066139) [0xffffff800030449b]
*1    machine_switch_context + 206
```

# What can be read from the call stack

This is a complete call stack for sending raw vendor-specific command.

The entry and exit of macOS IOBluetoothFamily HCI are routines IOBluetoothHCIUserClient::SimpleDispatchWL and IOBluetoothHCIRequest::Start.

How to ensure that Bluetooth-related data structures are safe in a multithreaded environment?

# IOCommandGate mechanism

Class IOCommandGate
Single-threaded work-loop client request mechanism.
https://developer.apple.com/documentation/kernel/iocommandgate

Routine IOCommandGate::runAction
Single thread a call to an action with the target work-loop.

Routine IOCommandGate::commandSleep
Put a thread that is currently holding the command gate to sleep.

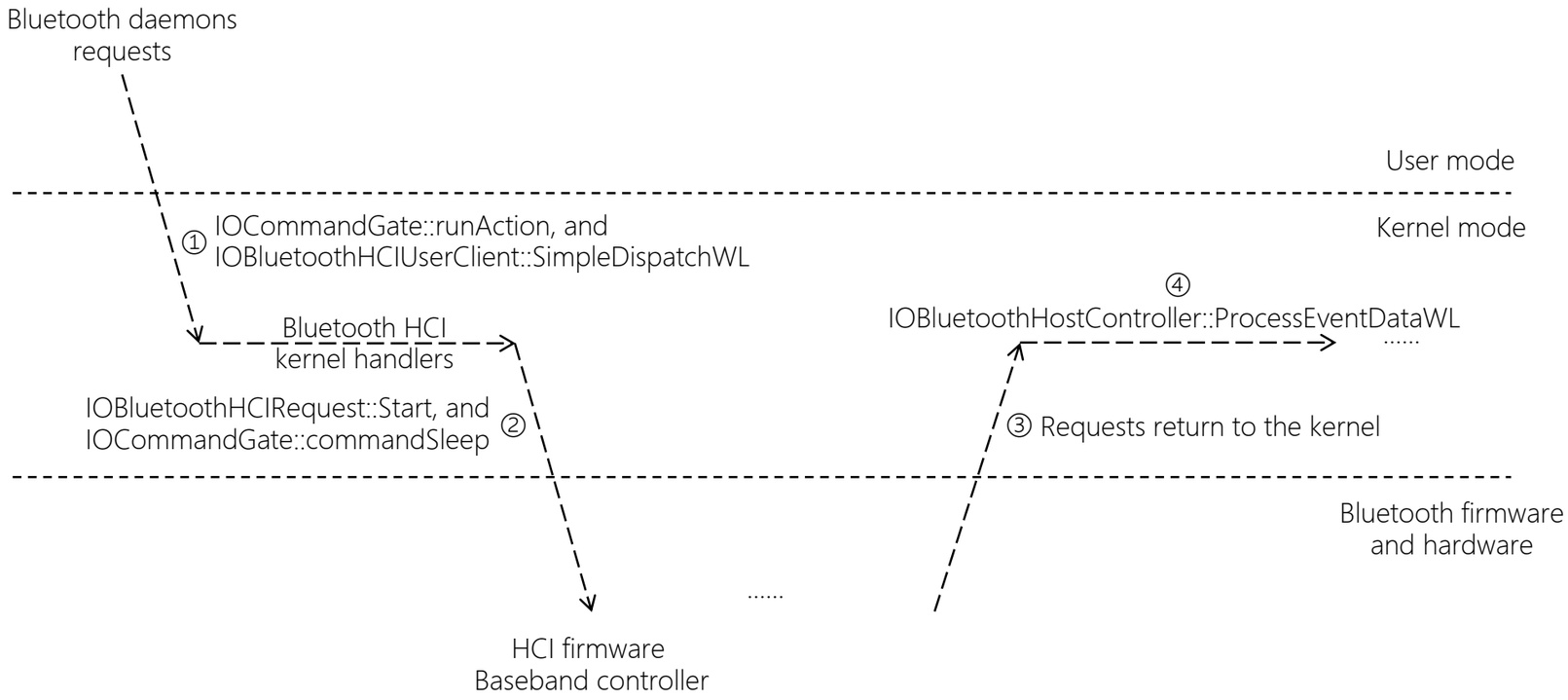
# Yes, you can sleep for a while

Routine IOCommandGate::commandSleep
Put a thread to sleep waiting for an event **but release the gate first**.

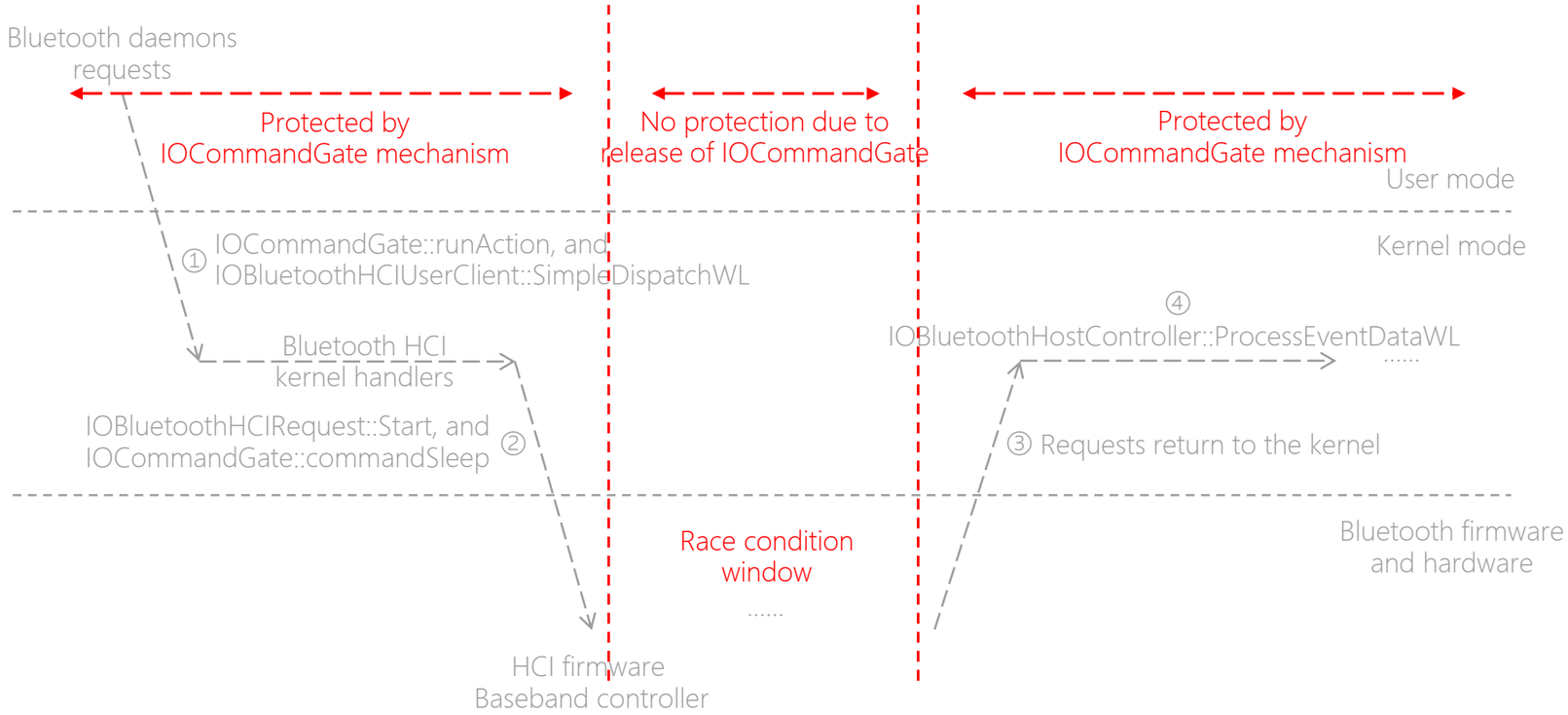
At this time, the HCI request is NOT completed by the Bluetooth controller.
So again, how to ensure the Bluetooth-related data structures are safe in this window?

Unfortunately, this issue has not been considered.

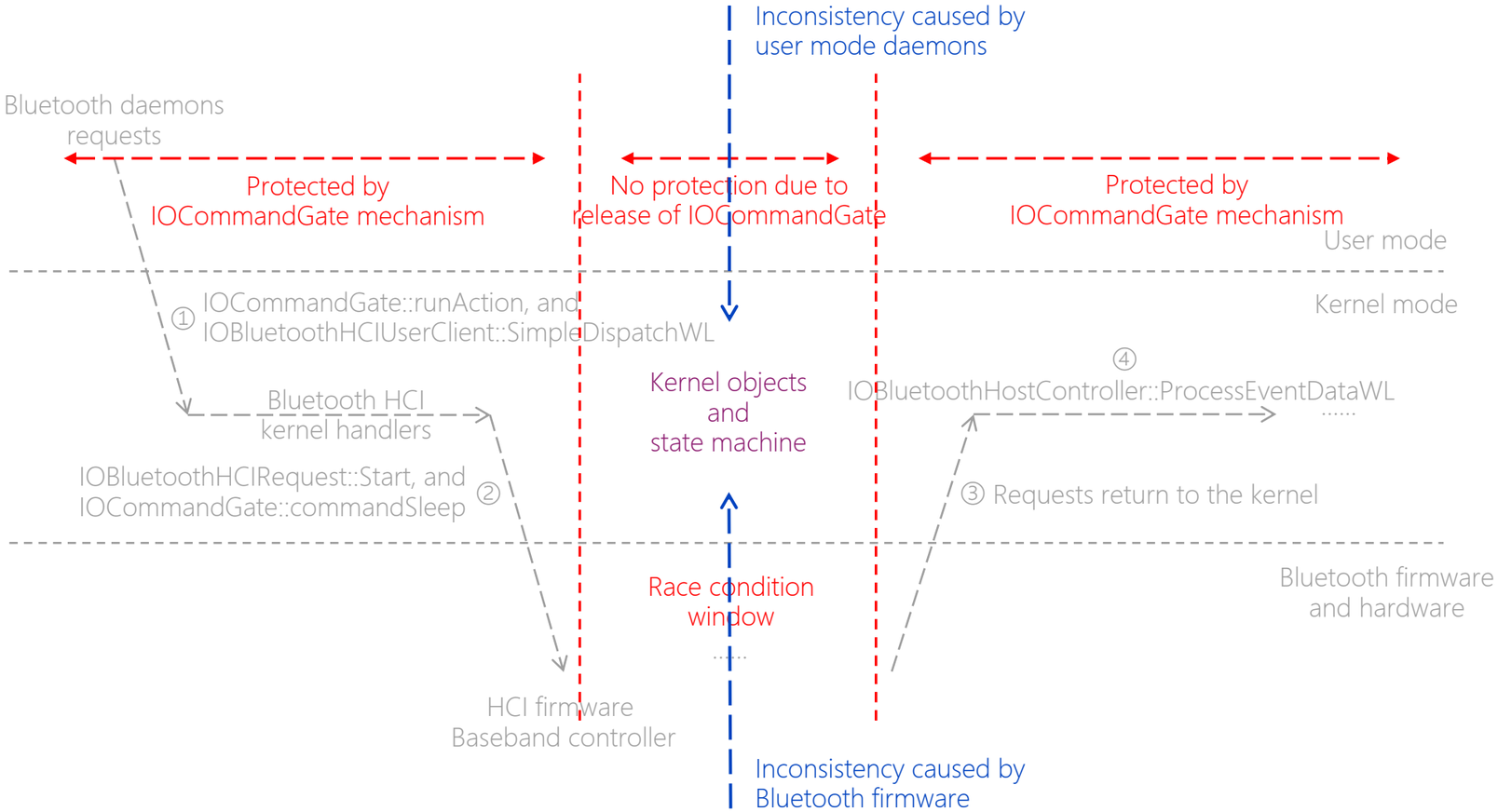
# IOBluetoothFamily HCI request flow



# Race condition window



# Data and state inconsistency



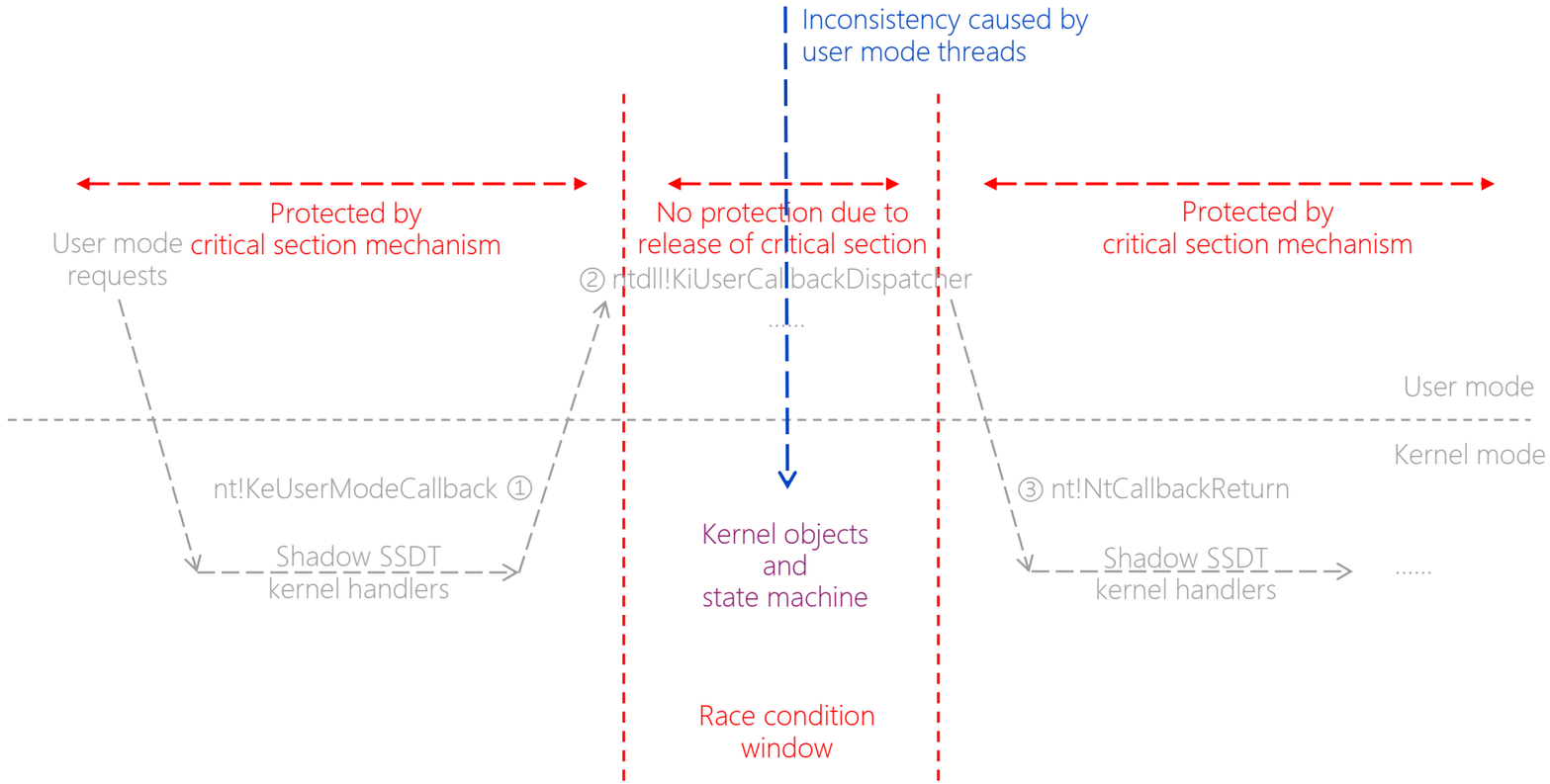
# Recall the Win32K user mode callback mechanism

Win32k cannot hold the lock when calling back to user mode. Releasing the lock means that there is a window in which the kernel data structures are not protected.

Reference counting and object lifecycle management are very important.

A New CVE-2015-0057 Exploit Technology
https://www.blackhat.com/docs/asia-16/materials/asia-16-Wang-A-New-CVE-2015-0057-Exploit-Technology-wp.pdf

# nt!KeUserModeCallback and nt!NtCallbackReturn



# Case study of CVE-2020-9928

```
(lldb) register read rdx rsi
General Purpose Registers:
       rdx = 0xffffff801270fcfa  ""Element %p from zone %s caught being freed to wrong zone %s\n"
@/BuildRoot/Library/Caches/com.apple.xbs/Sources/xnu/xnu-4570.61.1/osfmk/kern/zalloc.c:3528"
       rsi = 0xffffff8012749a40  "panic"

(lldb) bt
thread #1, stop reason = signal SIGSTOP
frame #0: 0xffffff8011f7c8ea kernel.development`panic_trap_to_debugger [inlined] current_cpu_datap
frame #1: 0xffffff8011f7c8ea kernel.development`panic_trap_to_debugger [inlined] current_processor
frame #2: 0xffffff8011f7c8ea kernel.development`panic_trap_to_debugger [inlined] DebuggerTrapWithState
frame #3: 0xffffff8011f7c8ba kernel.development`panic_trap_to_debugger
frame #4: 0xffffff8011f7c6bc kernel.development`panic(str=<unavailable>) at debug.c:611:2 [opt]
frame #5: 0xffffff8011fd5f09 kernel.development`zfree(zone=0xffffff80128c10d0, addr=0xffffff80403ae070)
frame #6: 0xffffff8011f89a69 kernel.development`kfree(data=0xffffff80403ae070, size=248)
frame #7: 0xffffff8012601739 kernel.development`::IOFree(inAddress=<unavailable>, size=248)
frame #8: 0xffffff7f94ebf90e IOBluetoothFamily`IOBluetoothHCIUserClient::SimpleDispatchWL + 1676
frame #9: 0xffffff801263eb58 kernel.development`IOCommandGate::runAction at IOCommandGate.cpp:217:11 [opt]
frame #10: 0xffffff7f94ebf266 IOBluetoothFamily`IOBluetoothHCIUserClient::externalMethod + 228
......
```

# Summary of case #6

1. Vulnerabilities like CVE-2020-9928 have been hidden in plain sight for a long time and affect all macOS Bluetooth HCI handlers.

2. Some traditional fuzzing methods are difficult to find this type of vulnerability.

3. Security Update 2020-002 can be bypassed.

# The End

# From the perspective of kernel development

1. State machine and exception handling, etc. need to be carefully designed.

2. Corner/Test cases matter.

3. "Do one thing and do that well."

# From the perspective of vulnerability research

1. Vulnerabilities like CVE-2020-3892 and CVE-2020-9928 have been hidden in plain sight for a long time and affect all macOS Bluetooth HCI handlers.

2. CVE-2020-3892 affects the validation and usage of HCI input parameters at a very early stage, while CVE-2020-9928 affects the synchronization design when HCI exits.

3. With this research as a starting point, I believe we can do more!

# From the perspective of security engineering

1. macOS Bluetooth HCI command and event sniffer helps us better understand the design of the HCI subsystem.

2. Passive and active fuzzing methods help us to hunt kernel vulnerabilities more efficiently.

3. Kemon-based code coverage analysis and kernel address sanitizer are both interesting and meaningful engineering attempts.

4. With the help of Kemon project, I believe we can do better!

# Q&A

wang yu

Didi Research America