Finding New Bluetooth Low Energy Exploits via Reverse Engineering Multiple Vendors' Firmwares

Veronica Kovah

Dark Mentor LLC

Hello World!

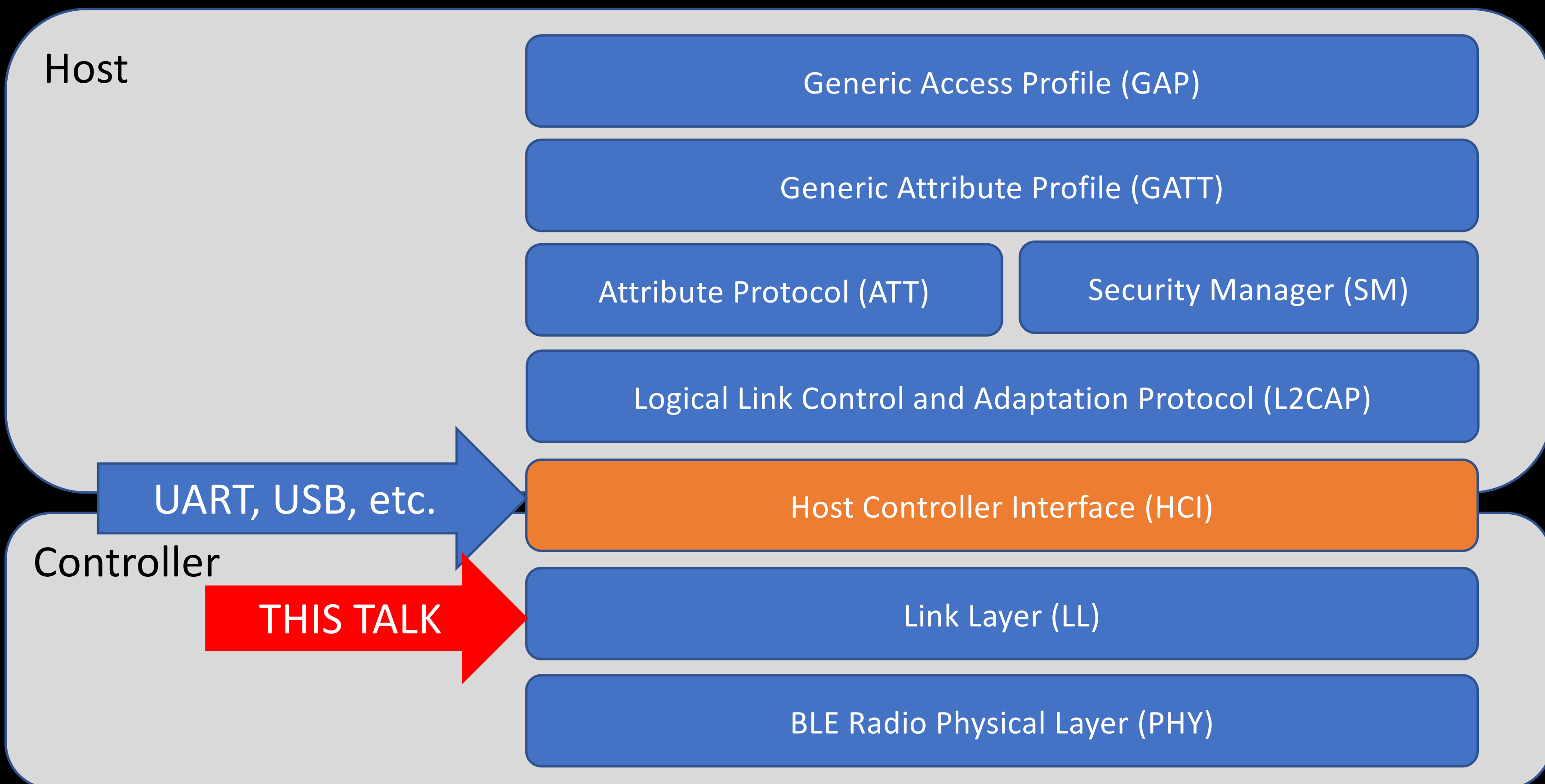
- Previously a security engineer for Tesla, NSA, MITRE, and Sourcefire
- Currently founder of Dark Mentor LLC, security consulting and education
- This talk is about sharing the journey from knowing almost nothing about Bluetooth to finding remote code execution vulnerabilities
- veronica@darkmentor.com, @VeronicaKovah

Starting from scratch…

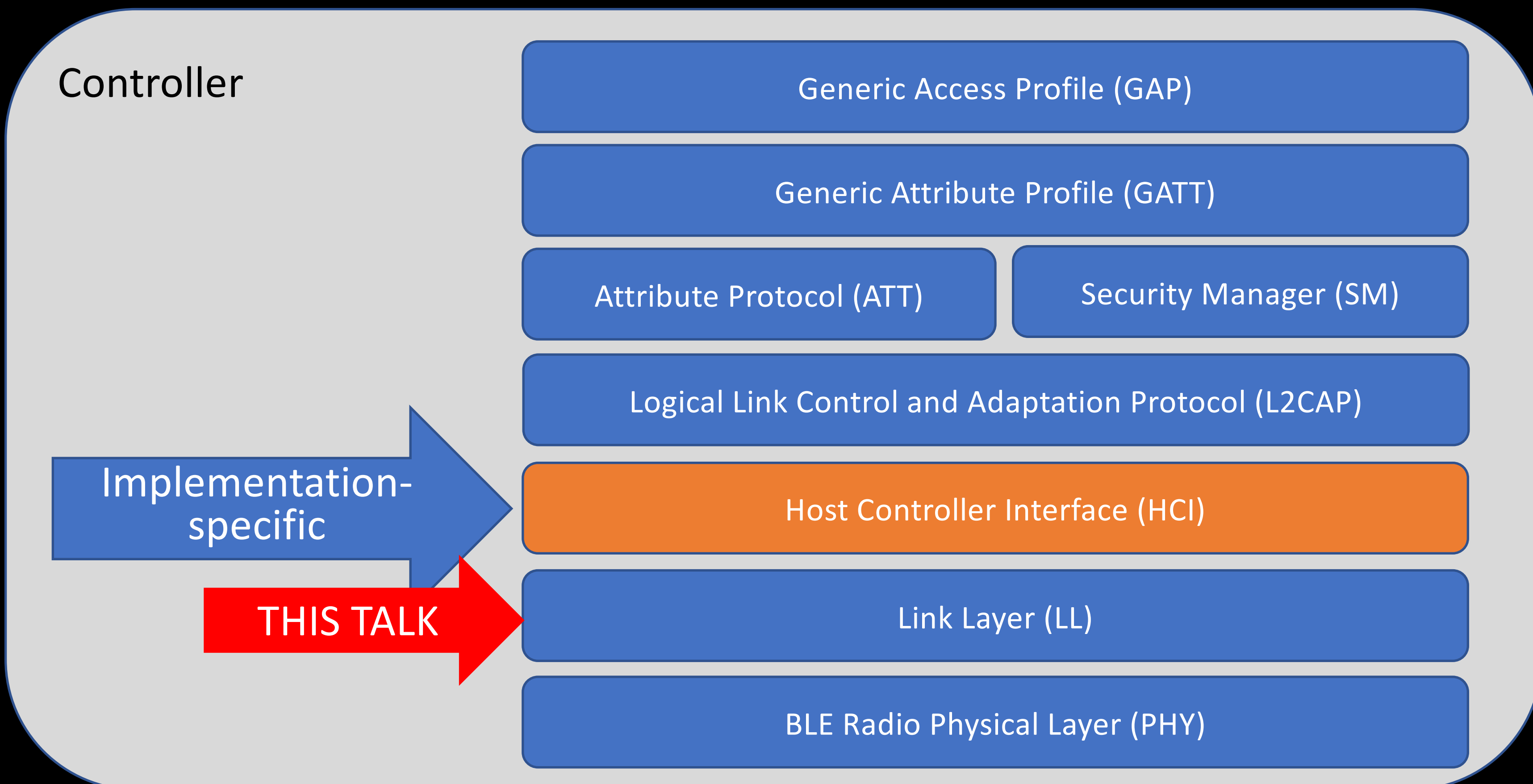
Learning mode

- Surveyed existing Bluetooth (BT) security research
- Read the complex, more than 3000 pages, Bluetooth specification
  - Not back to back!
  - Focus on common developer's mistake: e.g. length, nested fields
- Looked for if there is any open source implementation below HCI
  - BT classic: could not find any
  - Bluetooth Low Energy (BLE) : Zephyr and Apache Mynewt NimBLE
- Started with BT classic, then moved onto BLE

BLE stack in *dual* chip configuration



BLE stack in *single* chip configuration



Bluetooth (classic and low energy) vulnerability CVE ID counts ***when I started***

Host

132

Controller

0

Bluetooth (classic and low energy) vulnerability CVE ID counts ***now***

Host

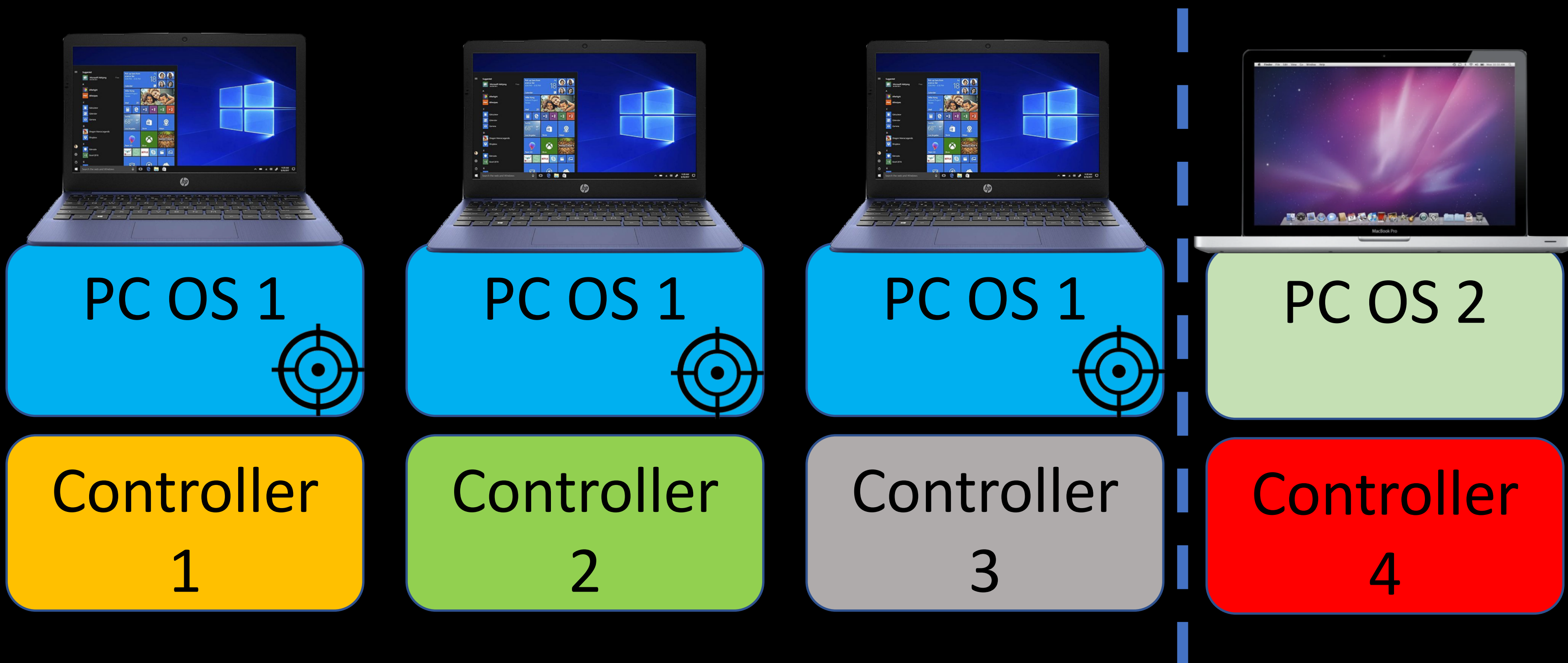
244

Controller

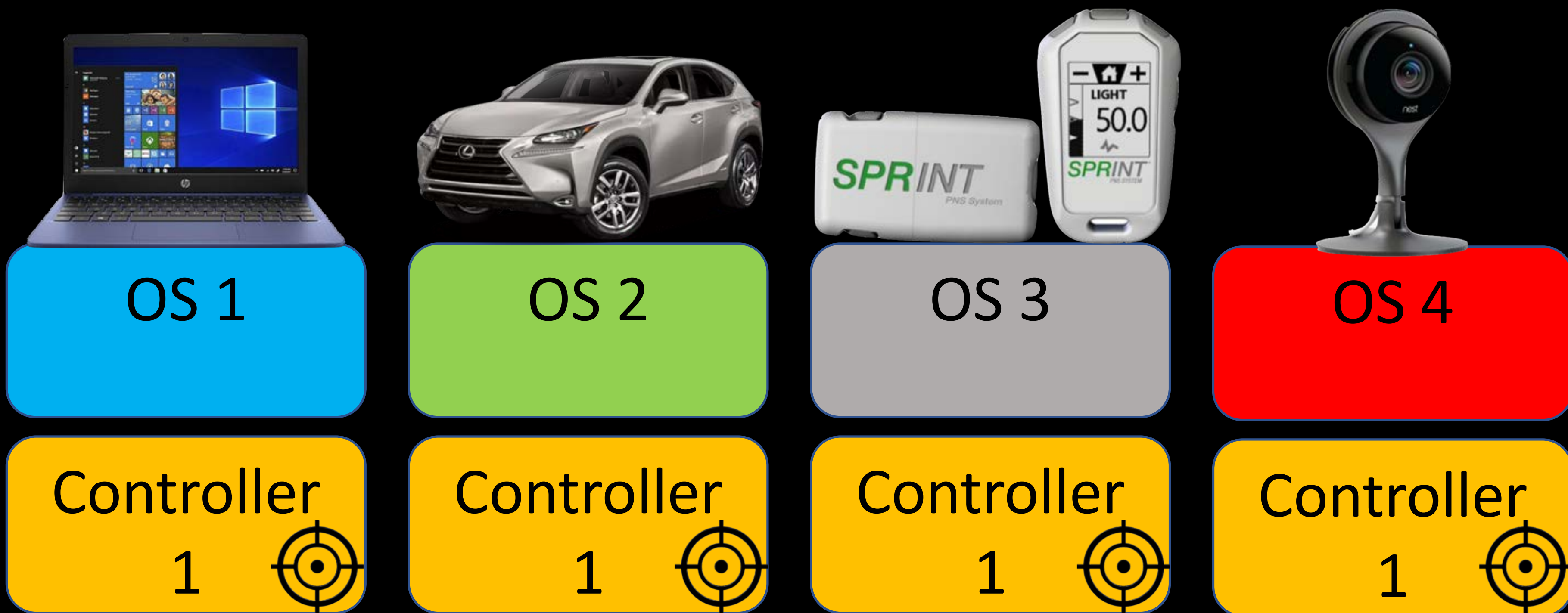
14

(2/3 BLE RCEs are this talk!)

Why target below the HCI layer?



Why target below the HCI layer?

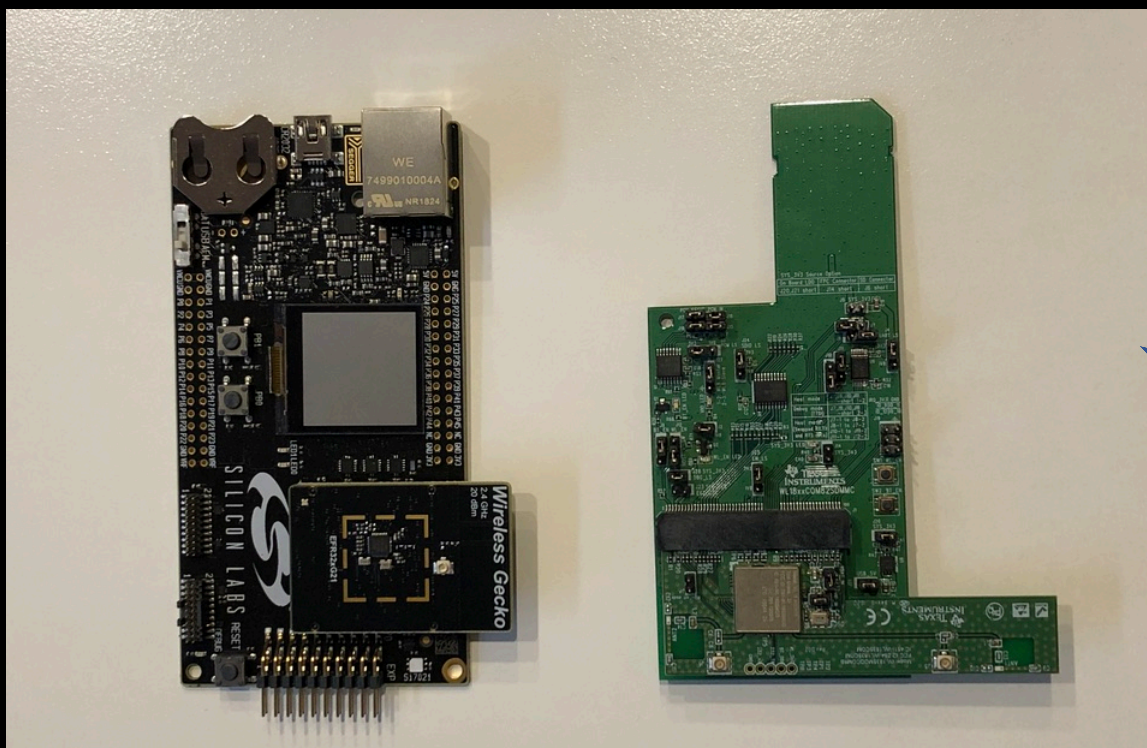


New BLE low layer vulnerabilities!

- Neither pairing nor authentication is required, just need proximity
- Texas Instruments CC256x and WL18xx dual-mode Bluetooth controller devices
  - Demo → RCE #1 (CVE-2019-15948)
  - Potential RCE (CVE-2019-15948)
- Silicon Labs BLE EFR32 SoC's and associated modules
  - Demo → RCE #2 (CVE-2020-15531)
  - DoS (CVE-2020-15532)

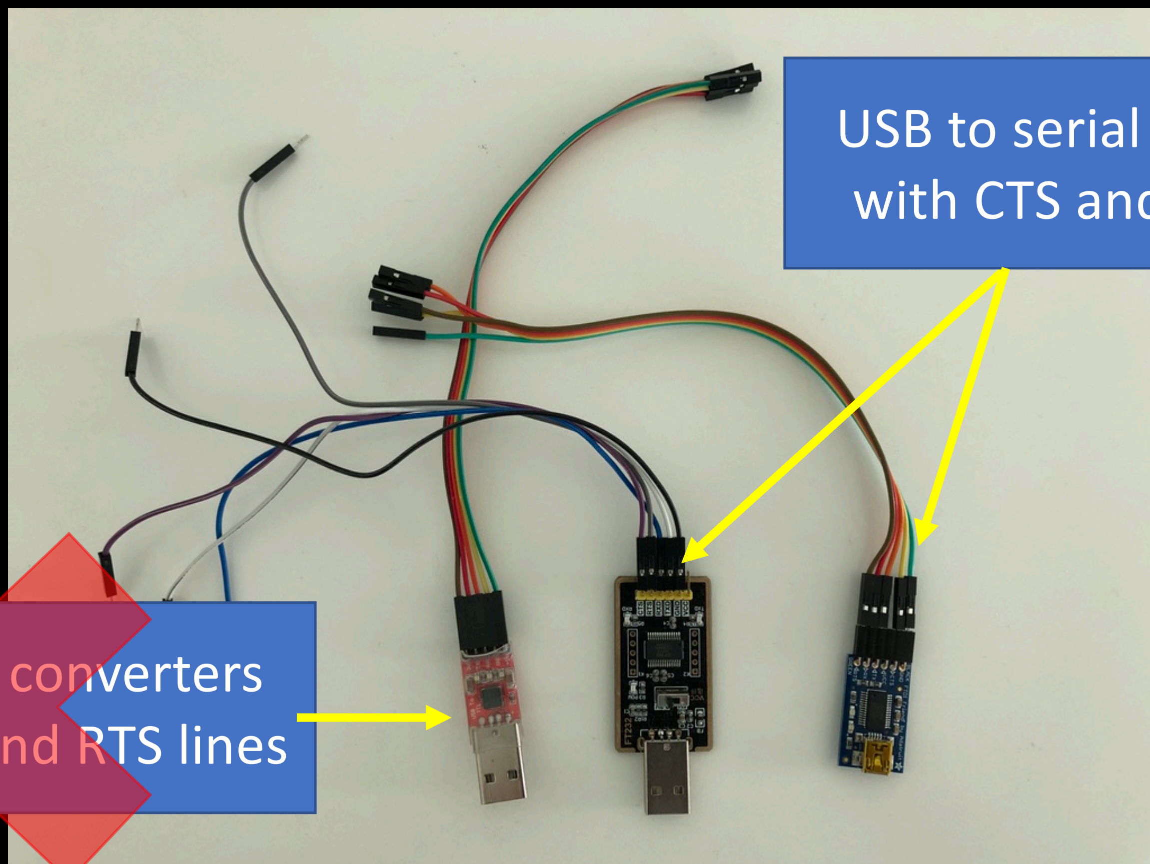
Lab Setup

Lab setup: targets



My lab has *way* more development boards but these are the ones I will talk about today ☺

Lab setup: for basic HW debug 1



USB to serial converters with CTS and RTS lines

USB to serial converters without CTS and RTS lines

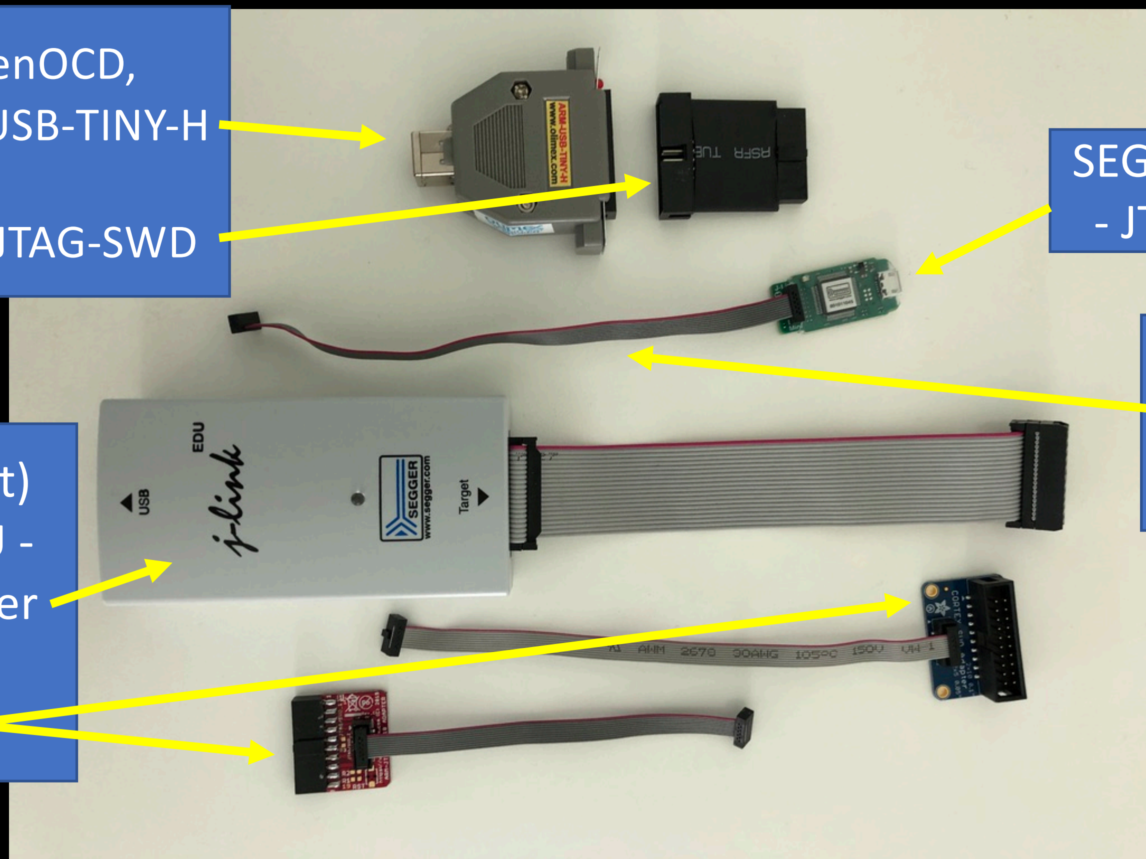
Lab setup: for basic HW debug 2

To use OpenOCD,
Olimex ARM-USB-TINY-H
+
Olimex ARM-JTAG-SWD

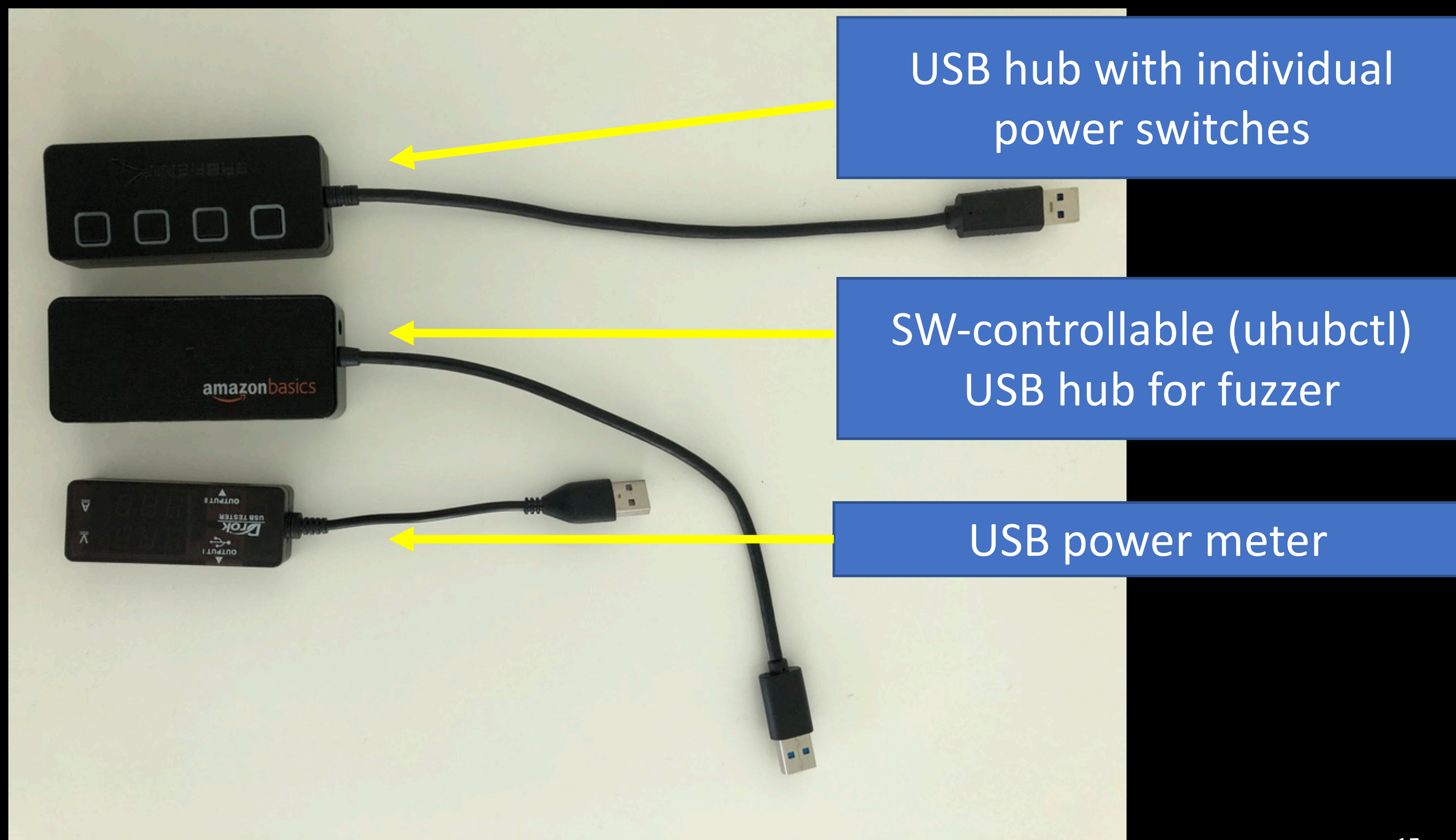
SEGGER J-Link EDU Mini
- JTAG/SWD debugger

10-pin 2x5 socket-socket 1.27mm IDC (SWD) cable

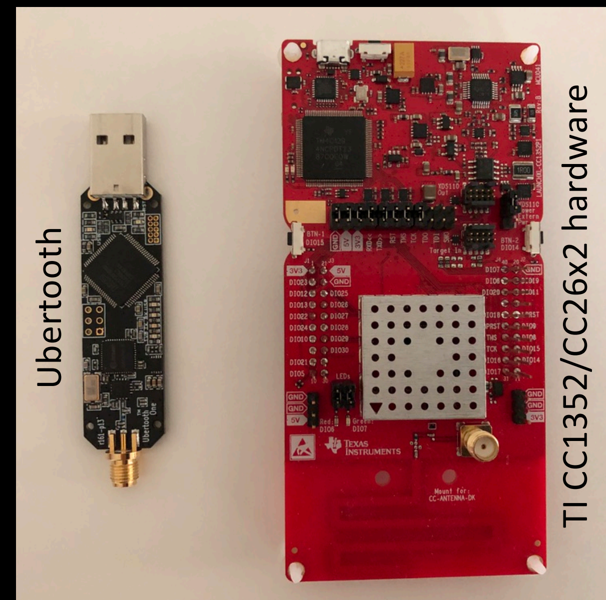
(used this the most)
SEGGER J-Link EDU -
JTAG/SWD debugger
+
SWD adapter



Lab setup: for fuzzer and convenience



Lab setup: sniffers



- Ubertooth
  - Great Scott Gadgets hardware
  - Pretty console display
  - (SW) does not support extended advertisement packets
  - http://ubertooth.sourceforge.net/
- Sniffle
  - TI CC1352/CC26x2 hardware
  - Supports BT 5 packet formats / PHY modes
  - Was very useful to build/debug a BLE fuzzer
  - Less pretty console display for a demo
  - https://www.nccgroup.com/us/our-research/sniffle-a-sniffer-for-bluetooth-5/

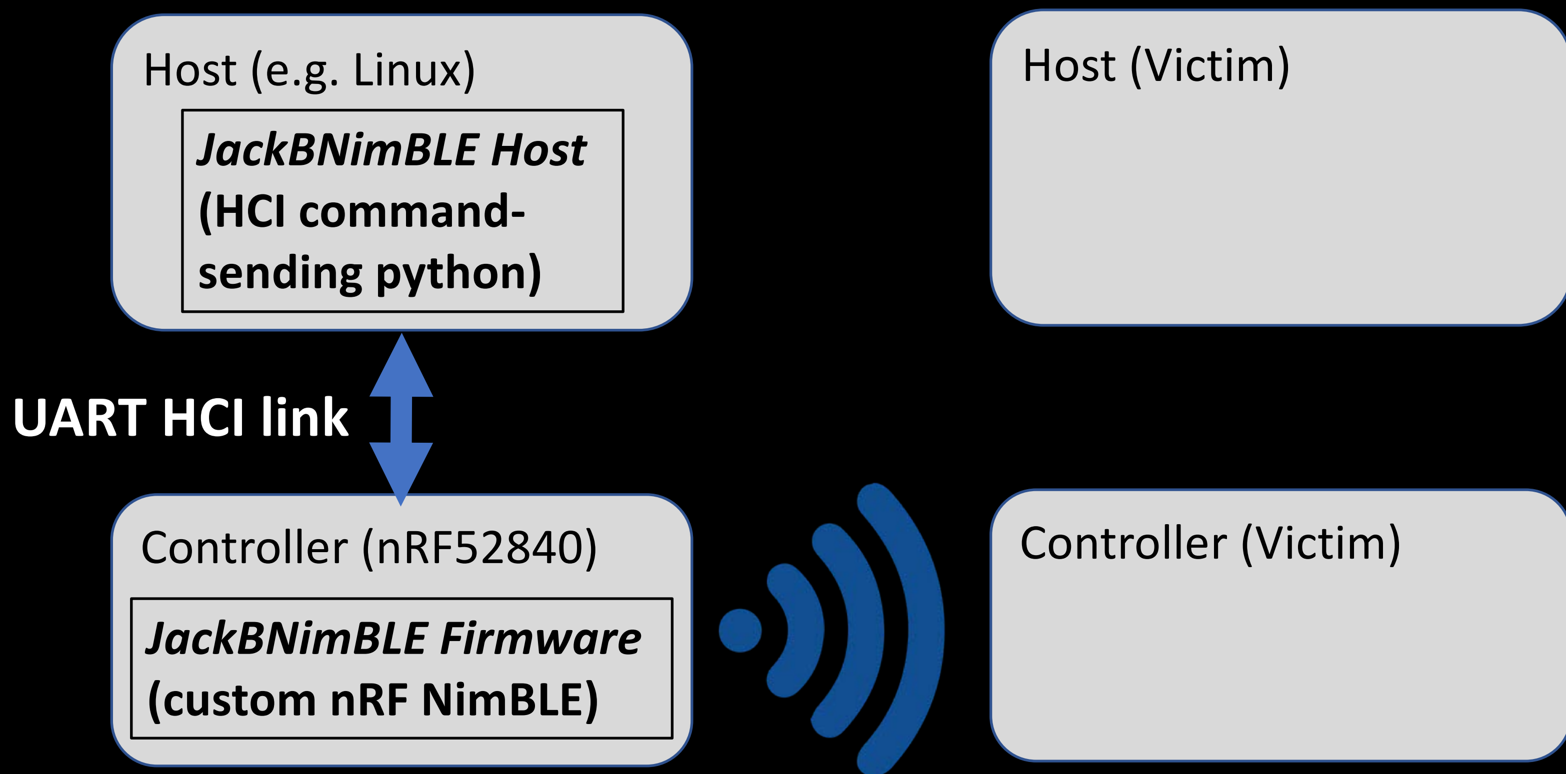
Note: There are many other sniffers, check if your project goal aligns with a sniffer's features

Lab setup: packet sending HW

- Started with Nordic Semiconductor nRF52832 dev board
  - Selected this first because open source BLE implementations had more documentation with this board (obviously B/C it's older dev board!)
  - USB to serial converter is necessary
- Ended up with nRF52840 dev board
  - UART interface through a virtual COM port
  - No USB to serial converter is needed

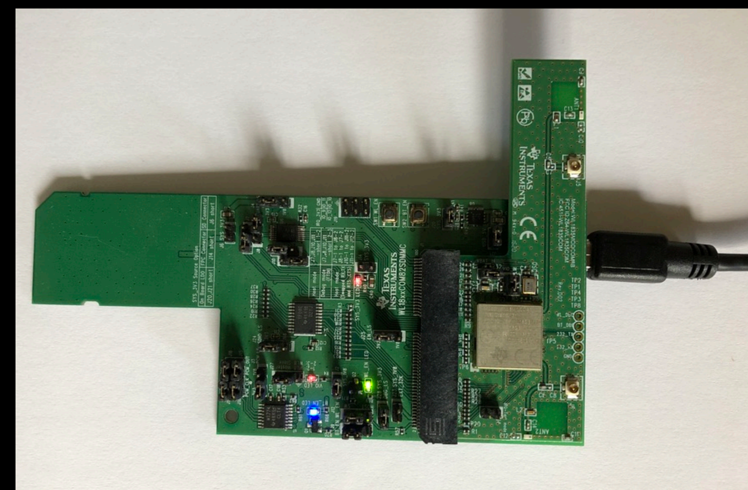
Lab setup: JackBNimBLE, packet sending SW

- Send arbitrary BLE Link Layer packets
- Extracted from my home-made fuzzer
- Controller SW: made modification to Apache Mynewt NimBLE (https://mynewt.apache.org/)
- Host SW: python scripts via HCI interface
- Current version can be used to share PoC
- Easy to extend, e.g. fuzzer
- https://github.com/darkmentorllc/jackbnimble


Host (e.g. Linux)
JackBNimBLE Host (HCI command-sending python)
UART HCI link
Controller (nRF52840)
JackBNimBLE Firmware (custom nRF NimBLE)
Host (Victim)
Controller (Victim)

Lab Setup Complete! Let’s attack!

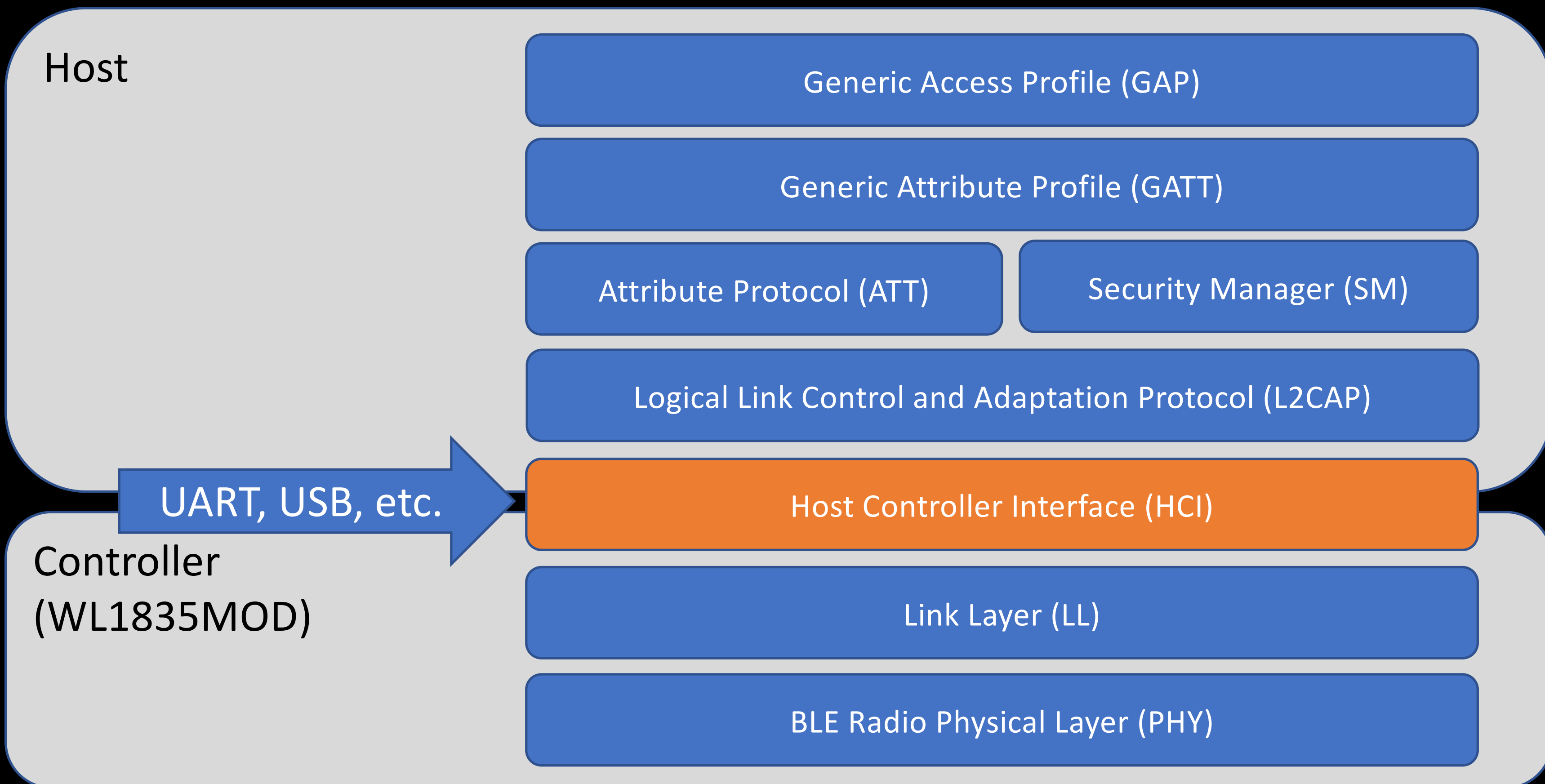
Target #1: Texas Instruments WL1835MOD



- Bluetooth v4.2
- Dual mode (BT classic and BLE)
- No JTAG or SWD readily available
- BLE Link Layer is in ROM
  - Host applies a patch
- No firmware image readily available
- WiLink™ Wireless Tools for WL18XX modules
  - HCITester: .bts binary patch -> human-readable format
  - Logger: UART binary debug messages-> human-readable format

Target #1

BLE stack in *dual chip* configuration

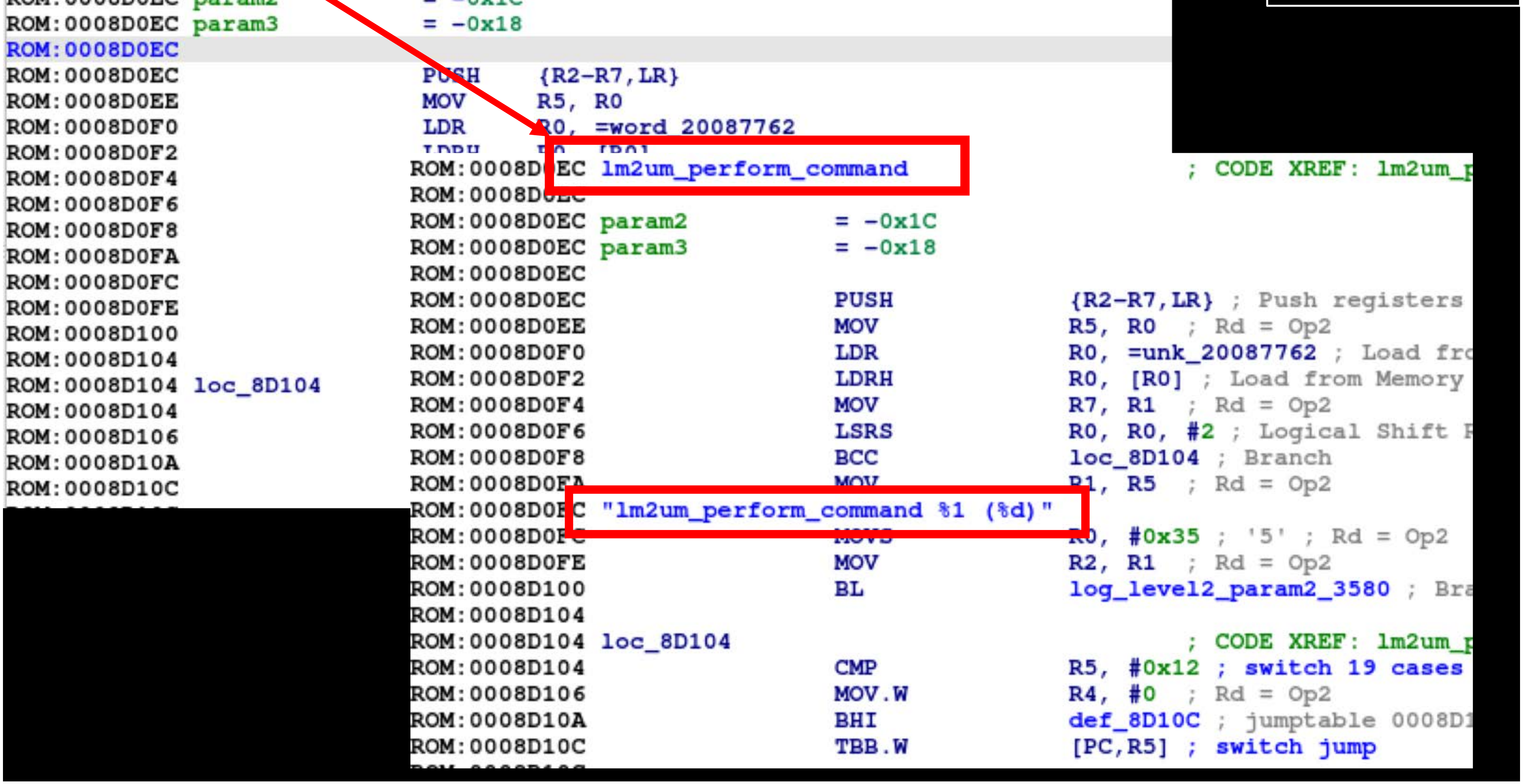


Target #1

Static analysis

- Memory dumping via Vendor Specific “HCI_VS_Read_Memory” command
- Used IDA Pro to analyze the dumped memory
- Identified log print functions whose arguments are a log string identifier(s) and the log string’s optional parameters like a format string
- Made an IDA Python script to add log strings where a log function call exists
  - Identified some function names
  - Inferred a lot of functions’ context

Target #1


ROM:0008D0EC sub_8D0EC ; CODE XREF: sub_8D1D4+18↓p
ROM:0008D0EC
ROM:0008D0EC param2 = -0x1C
ROM:0008D0EC param3 = -0x18
ROM:0008D0EC
ROM:0008D0EC PUSH {R2-R7,LR}
ROM:0008D0EE MOV R5, R0
ROM:0008D0F0 LDR R0, =word_20087762
ROM:0008D0F2
ROM:0008D0F4
ROM:0008D0F6
ROM:0008D0F8
ROM:0008D0FA
ROM:0008D0FC
ROM:0008D0FE
ROM:0008D100
ROM:0008D104
ROM:0008D104 loc_8D104
ROM:0008D104
ROM:0008D106
ROM:0008D10A
ROM:0008D10C
ROM:0008D0EC lm2um_perform_command ; CODE XREF: lm2um_p
ROM:0008D0EC param2 = -0x1C
ROM:0008D0EC param3 = -0x18
ROM:0008D0EC
ROM:0008D0EC PUSH {R2-R7,LR} ; Push registers
ROM:0008D0EE MOV R5, R0 ; Rd = Op2
ROM:0008D0F0 LDR R0, =unk_20087762 ; Load fro
ROM:0008D0F2 LDRH R0, [R0] ; Load from Memory
ROM:0008D0F4 MOV R7, R1 ; Rd = Op2
ROM:0008D0F6 LSRS R0, R0, #2 ; Logical Shift R
ROM:0008D0F8 BCC loc_8D104 ; Branch
R1, R5 ; Rd = Op2
"lm2um_perform_command %1 (%d)"
#0x35 ; '5' ; Rd = Op2
ROM:0008D0FE MOV R2, R1 ; Rd = Op2
ROM:0008D100 BL log_level2_param2_3580 ; Bra
ROM:0008D104
ROM:0008D104 loc_8D104 ; CODE XREF: lm2um_p
ROM:0008D104 CMP R5, #0x12 ; switch 19 cases
ROM:0008D106 MOV.W R4, #0 ; Rd = Op2
ROM:0008D10A BHI def_8D10C ; jumptable 0008D1
ROM:0008D10C TBB.W [PC,R5] ; switch jump

Target #1

Dynamic analysis

- Used a home-made fuzzer
- RE’ed the hard fault handler and enabled more logs to see register, stack, and heap memory states
- Patched binary for debugging via hooking
  - Don’t know how to do JTAG wiring
  - Cortex-M3 Flash Patch and Breakpoint Unit (FPB)
  - Used HCI_VS_Write_Memory to have an alternate code for reading memory and/or register values
  - Used log() to send values to UART

Target #1

log_without_patch.lgr - Logger 5.0 - Not Connected

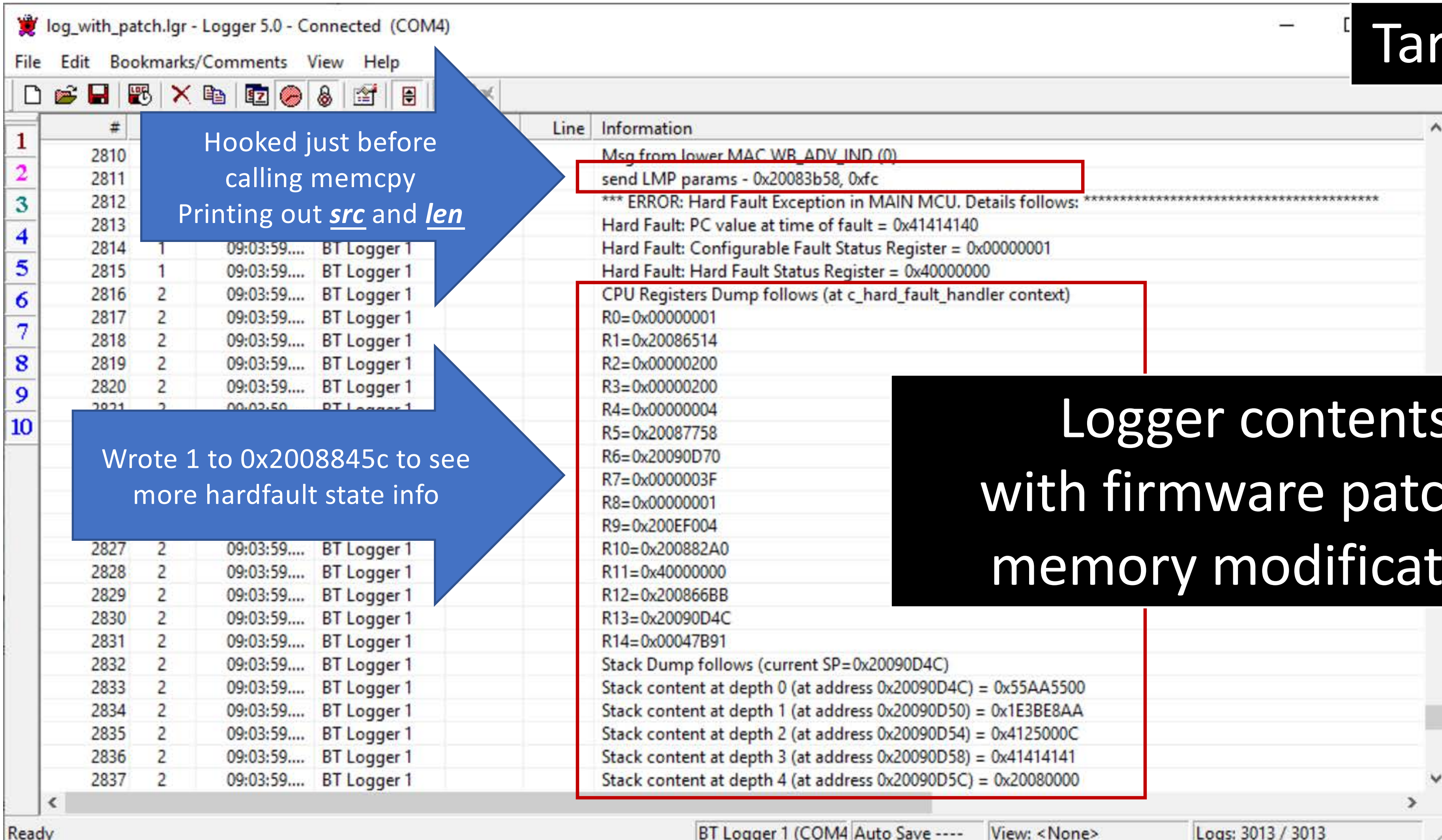
File Edit Bookmarks/Comments View Help

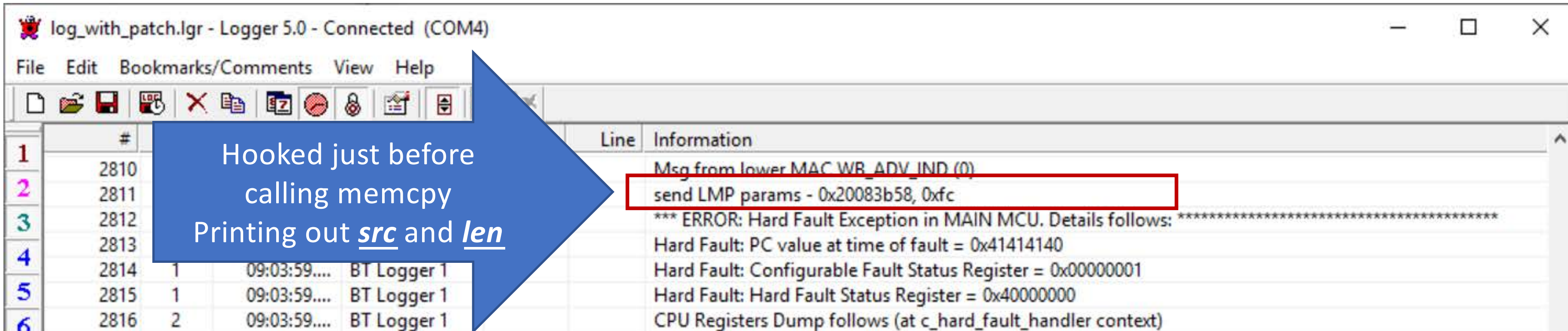
#	Level	Time	Port	File N...	Line	Information
1152	6	15:25:59....	BT Logger 1			Msg from lower MAC WB_ADV_IND (0)
1153	6	15:25:59....	BT Logger 1			SCANNER RCV PKT: type 0,clk 40471, pt 291
1154	6	15:25:59....	BT Logger 1			SCANNER RCV PKT: type 2,clk 40475, pt 499
1155	6	15:25:59....	BT Logger 1			Msg from lower MAC WB_NON_CONN_ADV_IND
1156	6	15:25:59....	BT Logger 1			SCAN, got invalid packet. type=14, length=33, wb_ac_corr_ind=0xf1
1157	6	15:25:59....	BT Logger 1			SCANNER RCV PKT: type 0,clk 40487, pt 738
1158	6	15:25:59....	BT Logger 1			SCANNER RCV PKT: type 0,clk 40511, pt 36
1159	6	15:25:59....	BT Logger 1			SCANNER RCV PKT: type 0,clk 40533, pt 1187
1160	6	15:25:59....	BT Logger 1			Msg from lower MAC WB_ADV_IND (0)
1161	1	15:25:59....	BT Logger 1			*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **
1162	1	15:25:59....	BT Logger 1			Hard Fault: PC value at time of fault = 0x41414140
1163	1	15:25:59....	BT Logger 1			Hard Fault: Configurable Fault Status Register = 0x00000001
1164	1	15:25:59....	BT Logger 1			Hard Fault: Hard Fault Status Register = 0x40000000
1165	1	15:25:59....	BT Logger 1			*** Hard Fault Information end. Trying recovery at address [PC + 2] ************************************
1166	1	15:25:59....	BT Logger 1			*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **
1167	1	15:25:59....	BT Logger 1			Hard Fault: PC value at time of fault = 0x41414142
1168	1	15:25:59....	BT Logger 1			Hard Fault: Configurable Fault Status Register = 0x00000001
1169	1	15:25:59....	BT Logger 1			Hard Fault: Hard Fault Status Register = 0x40000000
1170	1	15:25:59....	BT Logger 1			*** Hard Fault Information end. Trying recovery at address [PC + 2] ************************************
1171	1	15:25:59....	BT Logger 1			*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **
1172	1	15:25:59....	BT Logger 1			Hard Fault: PC value at time of fault = 0x41414144
1173	1	15:25:59....	BT Logger 1			Hard Fault: Configurable Fault Status Register = 0x00000001
1174	1	15:25:59....	BT Logger 1			Hard Fault: Hard Fault Status Register = 0x40000000
1175	1	15:25:59....	BT Logger 1			*** Hard Fault Information end. Trying recovery at address [PC + 2] ************************************
1176	1	15:25:59....	BT Logger 1			*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **
1177	1	15:25:59....	BT Logger 1			Hard Fault: PC value at time of fault = 0x41414146
1178	1	15:25:59....	BT Logger 1			Hard Fault: Configurable Fault Status Register = 0x00000001
1179	1	15:25:59....	BT Logger 1			Hard Fault: Hard Fault Status Register = 0x40000000
1180	1	15:25:59....	BT Logger 1			*** Hard Fault Information end. Trying recovery at address [PC + 2] ************************************
1181	1	15:25:59	BT Logger 1			*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **

Ready | Auto Save ---- | View: <None> | Logs: 78749 / 78749

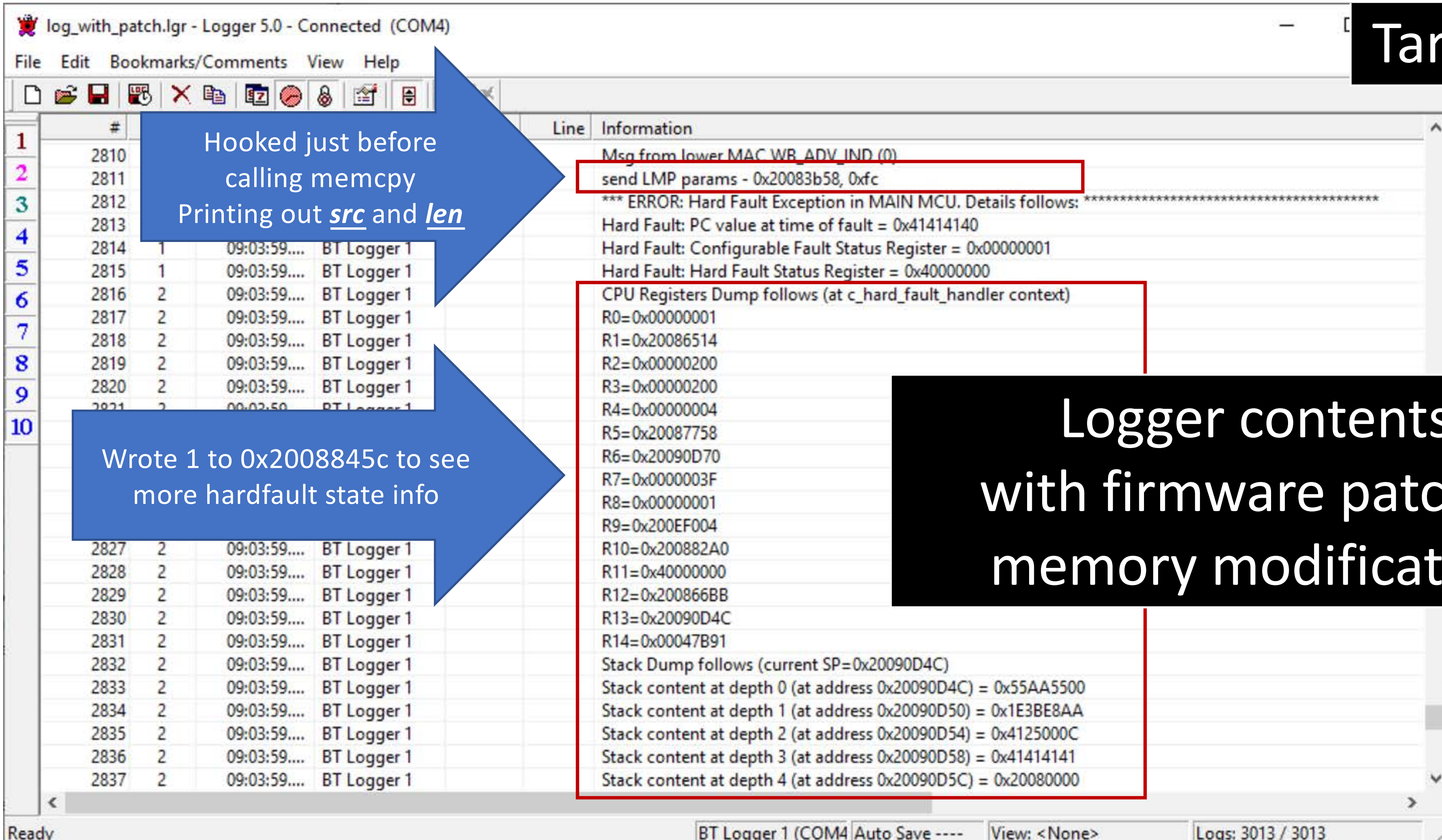
Target #1

Logger contents with firmware patch & memory modification




log_with_patch.lgr - Logger 5.0 - Connected (COM4)
File Edit Bookmarks/Comments View Help
Hooked just before calling memcpy
Printing out src and len
#
Line
Information
2810
Msg from lower MAC WB_ADV_IND (0)
2811
send LMP params - 0x20083b58, 0xfc
2812
*** ERROR: Hard Fault Exception in MAIN MCU. Details follows: **
2813
Hard Fault: PC value at time of fault = 0x41414140
2814 1 09:03:59.... BT Logger 1
Hard Fault: Configurable Fault Status Register = 0x00000001
2815 1 09:03:59.... BT Logger 1
Hard Fault: Hard Fault Status Register = 0x40000000
2816 2 09:03:59.... BT Logger 1
CPU Registers Dump follows (at c_hard_fault_handler context)

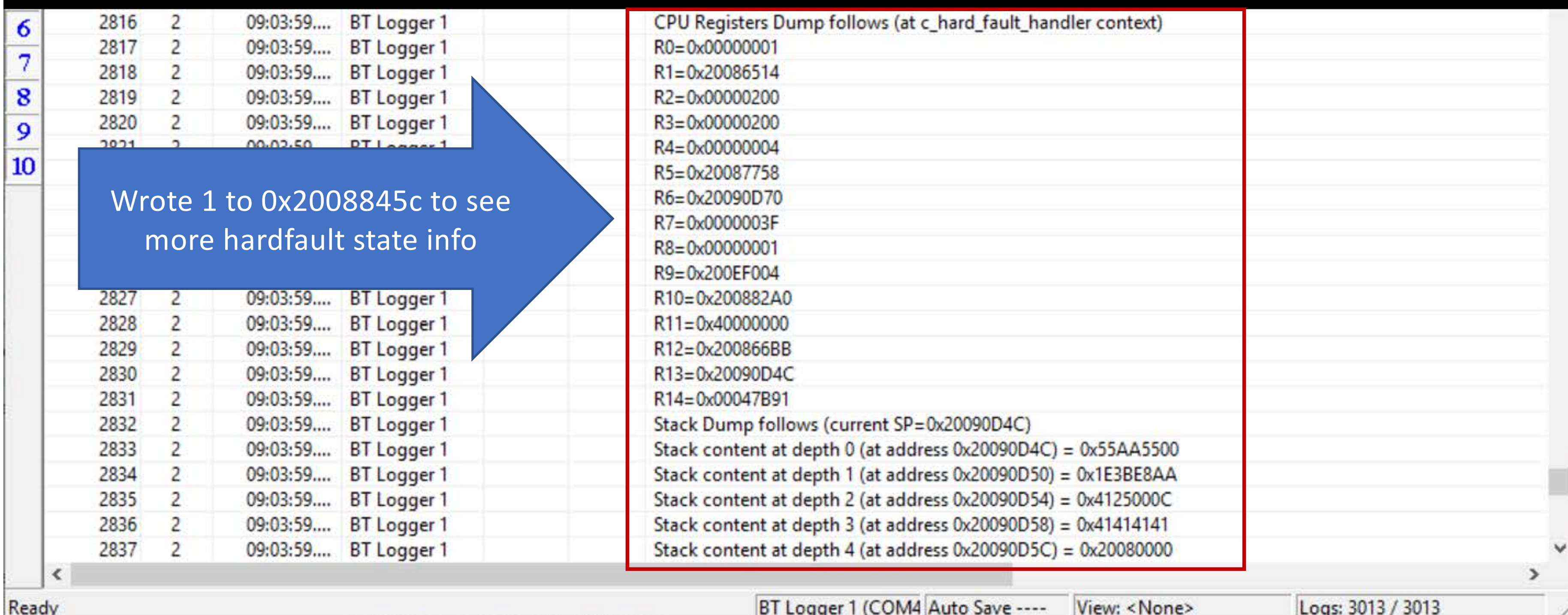
Target #1



Hooked just before calling memcpy
Printing out ***src*** and ***len***

Wrote 1 to 0x2008845c to see more hardfault state info

Logger contents with firmware patch & memory modification


Wrote 1 to 0x2008845c to see more hardfault state info
CPU Registers Dump follows (at c_hard_fault_handler context)
R0=0x00000001
R1=0x20086514
R2=0x00000200
R3=0x00000200
R4=0x00000004
R5=0x20087758
R6=0x20090D70
R7=0x0000003F
R8=0x00000001
R9=0x200EF004
R10=0x200882A0
R11=0x40000000
R12=0x200866BB
R13=0x20090D4C
R14=0x00047B91
Stack Dump follows (current SP=0x20090D4C)
Stack content at depth 0 (at address 0x20090D4C) = 0x55AA5500
Stack content at depth 1 (at address 0x20090D50) = 0x1E3BE8AA
Stack content at depth 2 (at address 0x20090D54) = 0x4125000C
Stack content at depth 3 (at address 0x20090D58) = 0x41414141
Stack content at depth 4 (at address 0x20090D5C) = 0x20080000
Ready
BT Logger 1 (COM4
Auto Save ----
View: <None>
Logs: 3013 / 3013

Target #1

Remote code execution bugs

- Static reverse engineering revealed integer underflows could cause stack buffer overflows
- Fuzzing with advertisement packets confirmed with a crash
- Wait… Yes, the “same” problem as BleedingBit but in a different code base (BleedingBit is heap overflow, mine is stack overflow)
- Reported on 5/22/2019, fixed on 11/12/2019

Target #1

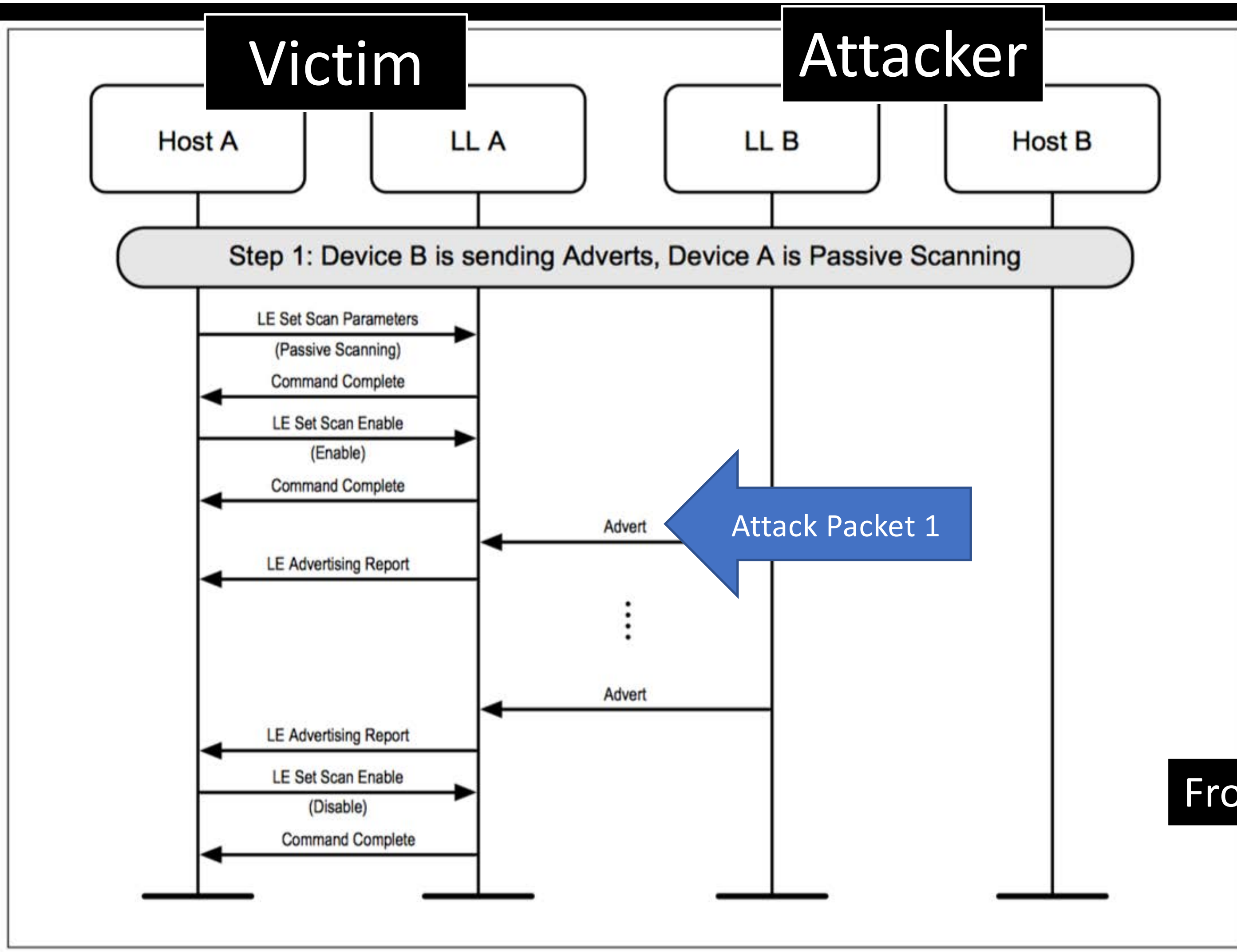


Figure 4.1: Passive Scanning

From Spec v4.2

Target #1

Stack buffer overflow 1
CVE-2019-15948

```
ROM:0005B3A0    PUSH      {R4-R7,LR}       ; LR is stored on stack
ROM:0005B3A2    SUB.W     SP, SP, #0x2C    ; stack buffer
...                                        ; R6 is PDU length
ROM:0005B3CE    SUBS      R6, R6, #6       ; integer underflow
ROM:0005B3D0    UXTB      R2, R6           ; unsigned byte extension
ROM:0005B3D2    ADD.W     R1, R5, #8       ; src, heap buffer address
ROM:0005B3D6    ADD.W     R0, SP, #9       ; dst, stack buffer address
ROM:0005B3DA    STRB.W    R2, [SP,#8]
ROM:0005B3DE    BL        memcpy
```

```
void *memcpy(void *dest, const void *src, size_t n);
```

R0 R1 R2

Target #1

Attack packet example 1

From Spec v4.2

LSB	MSB
Header (16 bits)	Payload (as per the Length field in the Header)

Figure 2.2: Advertising channel PDU

LSB					MSB
PDU Type (4 bits)	RFU (2 bits)	TxAdd (1 bit)	RxAdd (1 bit)	Length (6 bits)	RFU (2 bits)

Figure 2.3: Advertising channel PDU Header

Example: ADV_IND PDU Type

Header		Payload	
0x0**0**	**0x02**	0x41	0x41

Target #1

From Spec v4.2

PDU Type $b_3b_2b_1b_0$	Packet Name
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111-1111	Reserved

Table 2.1: Advertising channel PDU Header's PDU Type field encoding

Payload		
InitA (6 octets)	AdvA (6 octets)	LLData (22 octets)

Figure 2.10: CONNECT_REQ PDU payload

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.4: ADV_IND PDU Payload

Payload	
AdvA (6 octets)	InitA (6 octets)

Figure 2.5: ADV_DIRECT_IND PDU Payload

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_NONCONN_IND PDU Payload

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.7: ADV_SCAN_IND PDU Payload

Payload	
ScanA (6 octets)	AdvA (6 octets)

Figure 2.8: SCAN_REQ PDU Payload

Payload	
AdvA (6 octets)	ScanRspData (0-31 octets)

Figure 2.9: SCAN_RSP PDU payload

Target #1

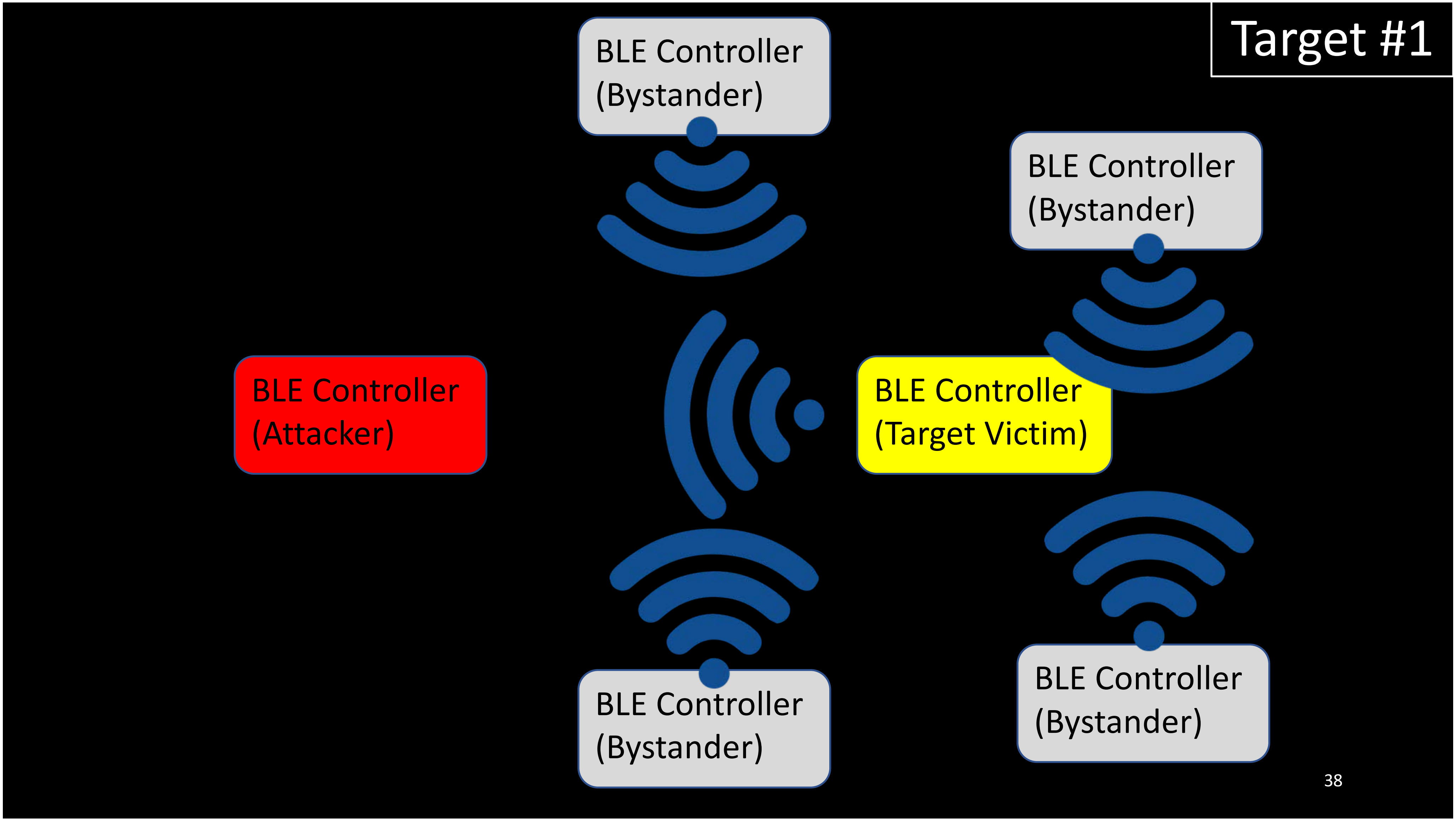
One little problem…

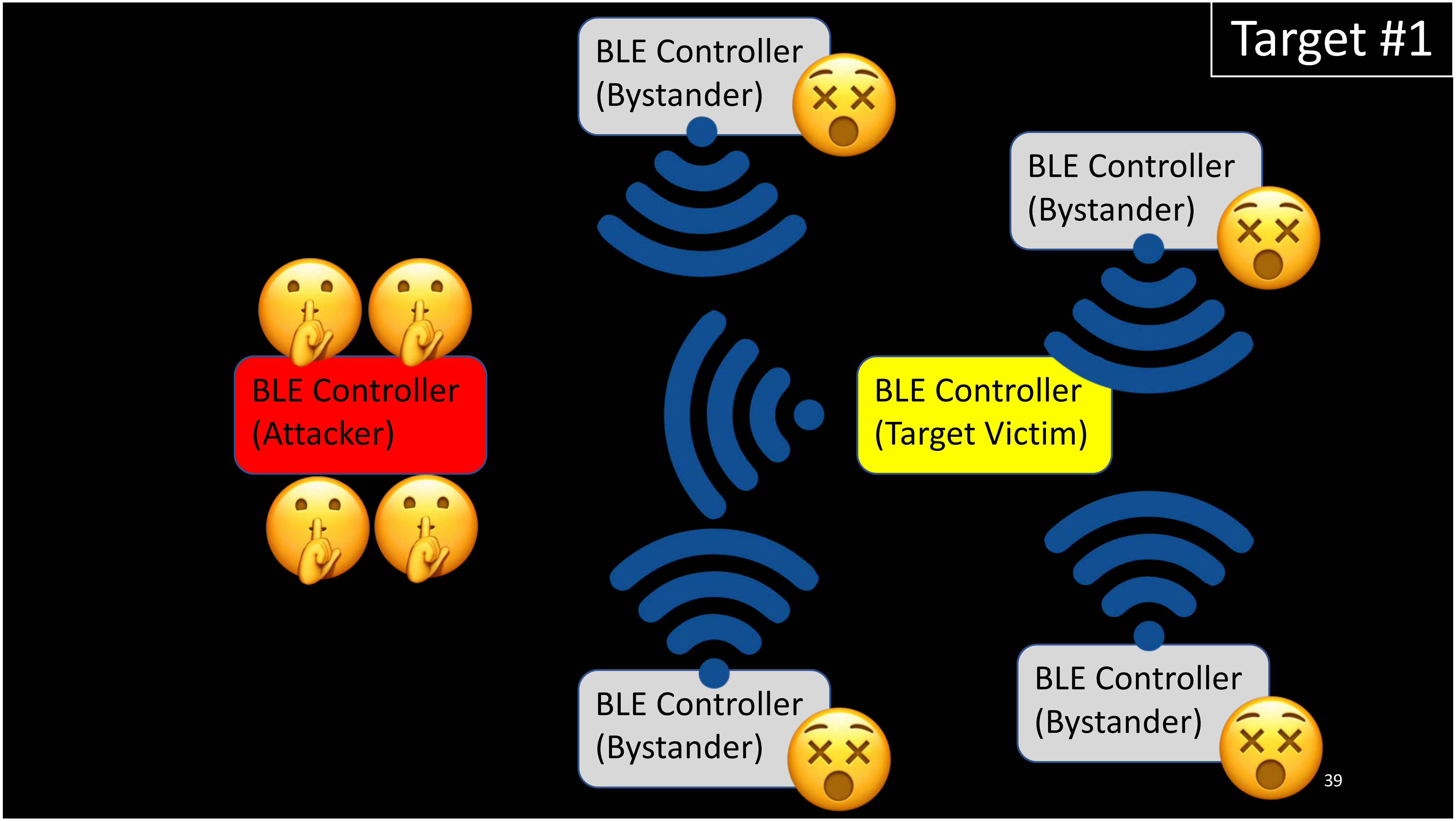
- Background BLE traffic affects heap contents, which affects exploit reliability

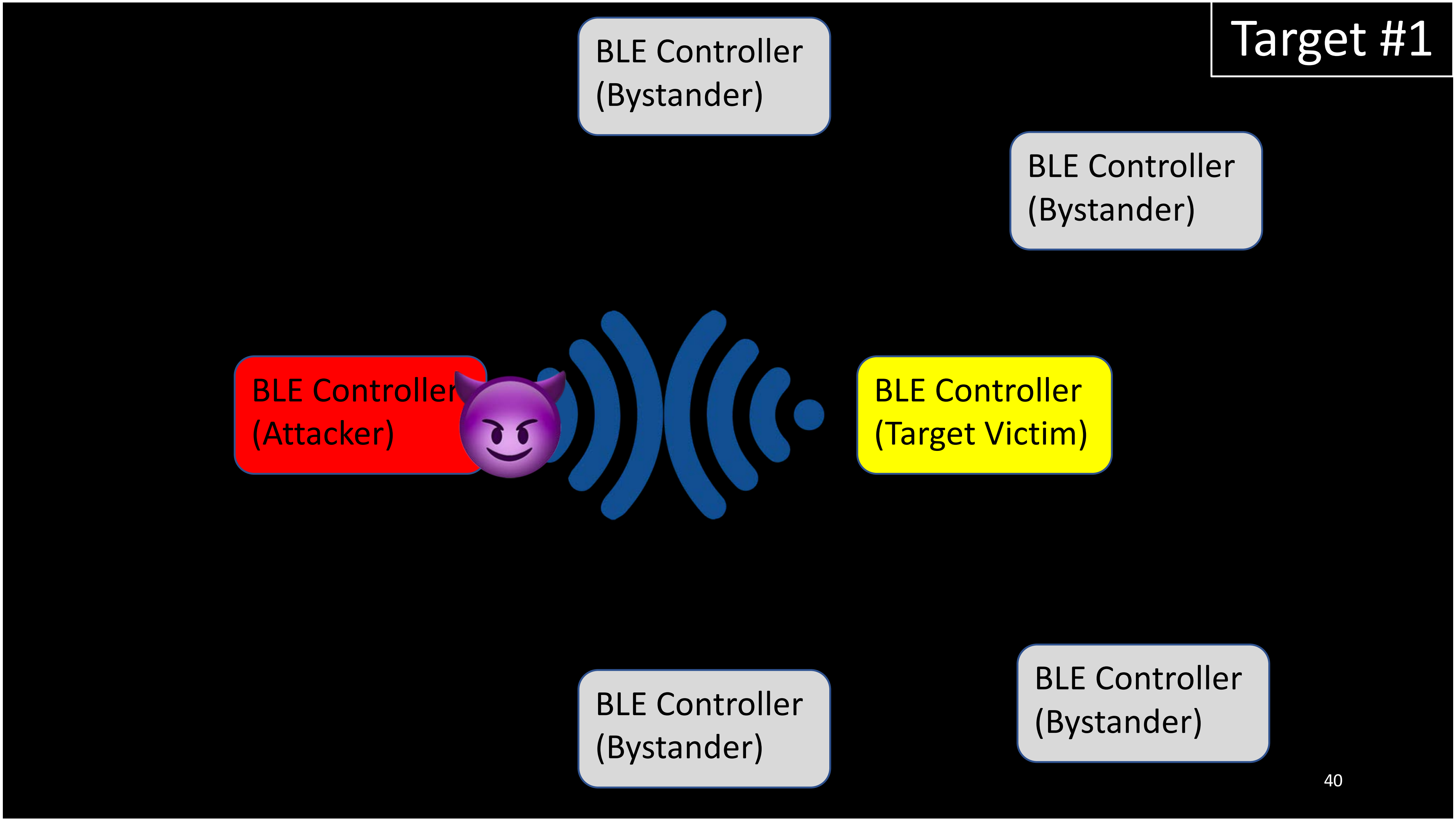
Target #1

“Quiet Place” attack

- Lots of DoS attacks
  - One (two?) of mine
  - Sweyntooth collection
  - Multiple SEEMOO’s findings
  - Any failed RCE attacks -> DoS ☺
- An attacker can selectively DoS nearby devices to quiet them down, to make it more reliable to exploit a target

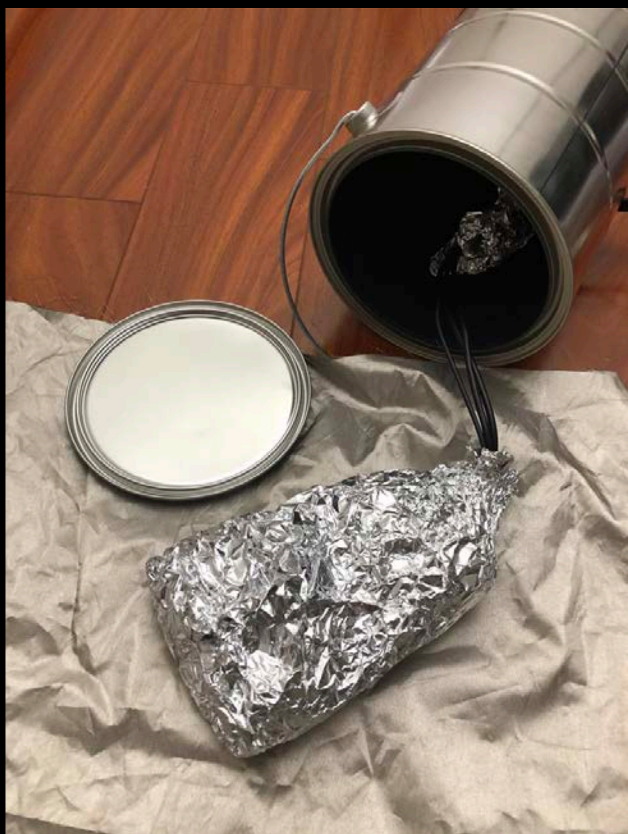

Target #1
BLE Controller (Bystander)
BLE Controller (Bystander)
BLE Controller (Attacker)
BLE Controller (Target Victim)
BLE Controller (Bystander)
BLE Controller (Bystander)

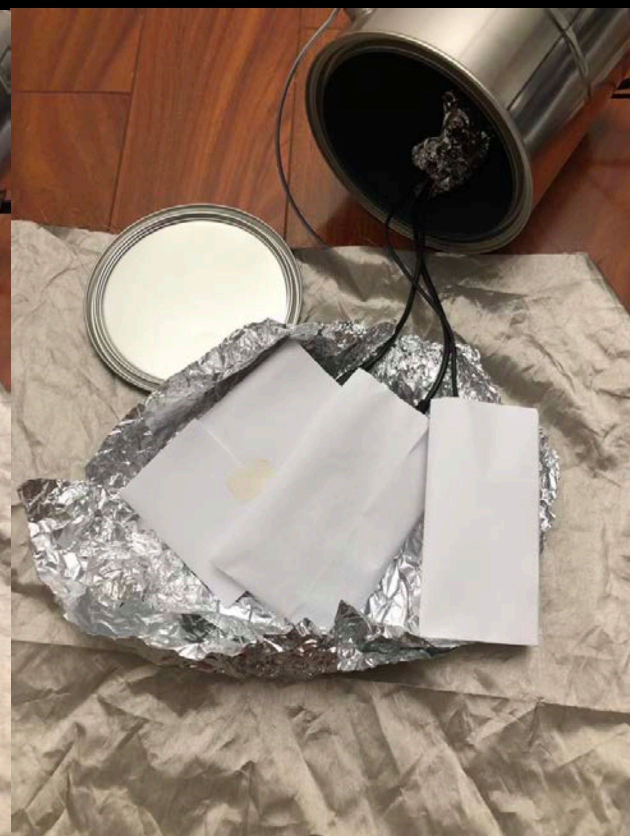

Target #1
BLE Controller
(Bystander)
BLE Controller
(Bystander)
BLE Controller
(Attacker)
BLE Controller
(Target Victim)
BLE Controller
(Bystander)
BLE Controller
(Bystander)

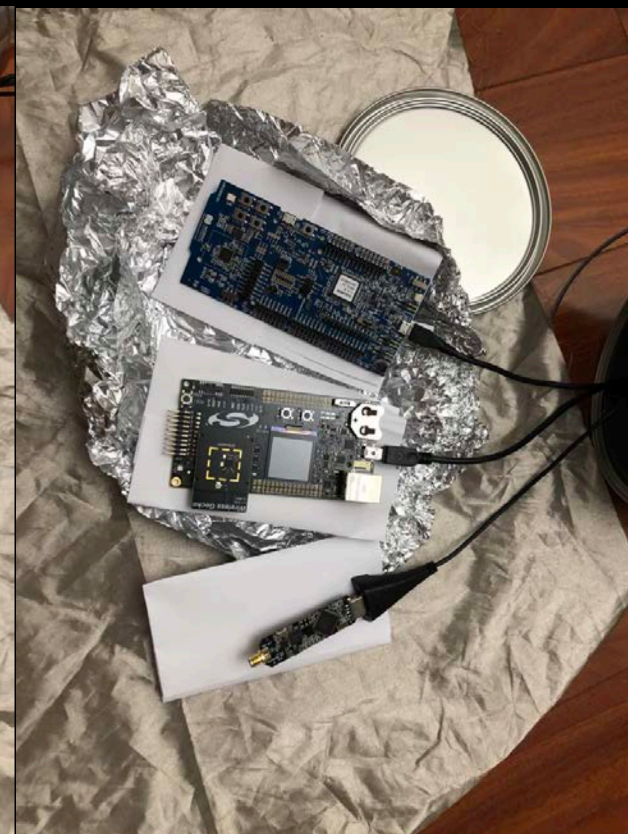

Target #1
BLE Controller
(Bystander)
BLE Controller
(Bystander)
BLE Controller
(Attacker)
BLE Controller
(Target Victim)
BLE Controller
(Bystander)
BLE Controller
(Bystander)

I has a bucket!

I has a bucket!



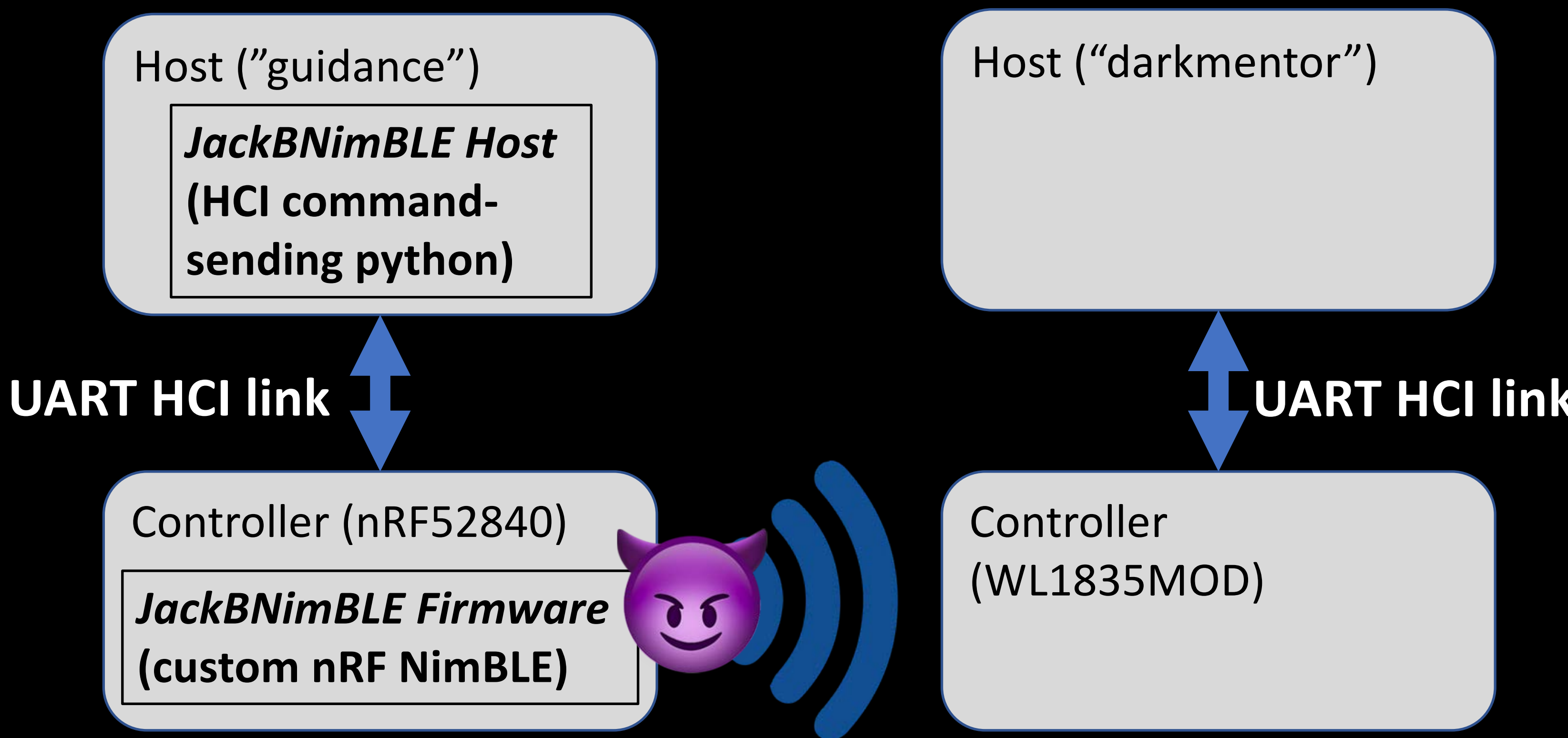




Target #1

RCE demo

Target #1


Host (”guidance”)
JackBNimBLE Host
(HCI command-sending python)
UART HCI link
Controller (nRF52840)
JackBNimBLE Firmware
(custom nRF NimBLE)
Host (“darkmentor”)
UART HCI link
Controller
(WL1835MOD)

Target #1

Stack buffer overflow 2
CVE-2019-15948

```
ROM:0005B348    PUSH      {R4,R5,LR}       ; LR is stored on stack
ROM:0005B34A    SUB.W     SP, SP, #0x2C    ; stack buffer
…                                          ; R0 is PDU length
ROM:0005B36E    ADD.W     R1, R4, #8       ; src, heap buffer address
ROM:0005B372    SUBS      R0, R0, #6       ; integer underflow
ROM:0005B374    UXTB      R2, R0           ; unsigned byte extension
ROM:0005B376    ADD.W     R0, SP, #9       ; dst, stack buffer address
ROM:0005B37A    STRB.W    R2, [SP,#8]
ROM:0005B37E    BL        memcpy
```

Target #1

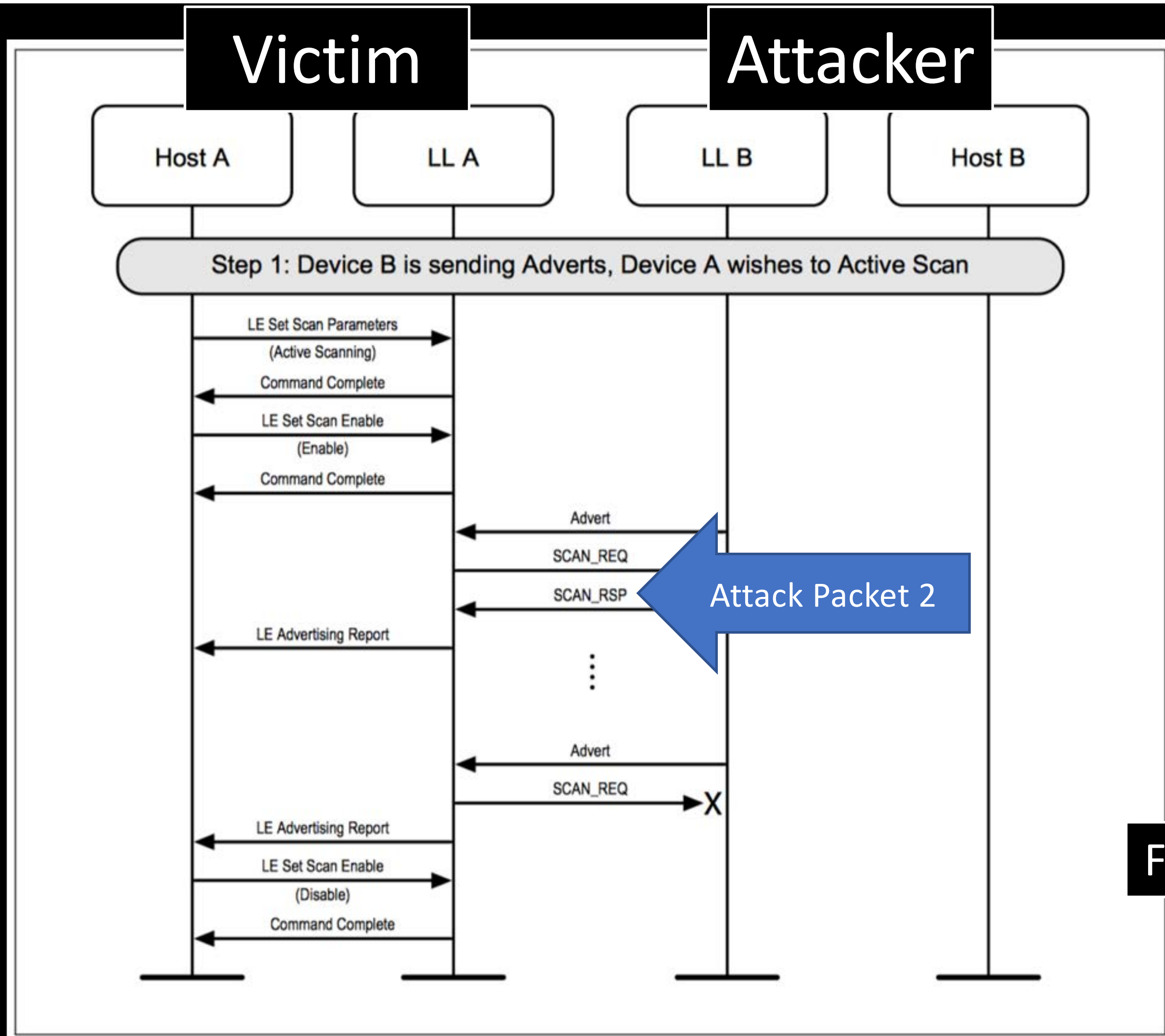


Figure 4.2: Active Scanning

From Spec v4.2

Target #1

Attack packet example 2

LSB		MSB
Header (16 bits)	Payload (as per the Length field in the Header)	

Figure 2.2: Advertising channel PDU

From Spec v4.2

LSB					MSB
PDU Type (4 bits)	RFU (2 bits)	TxAdd (1 bit)	RxAdd (1 bit)	Length (6 bits)	RFU (2 bits)

Figure 2.3: Advertising channel PDU Header

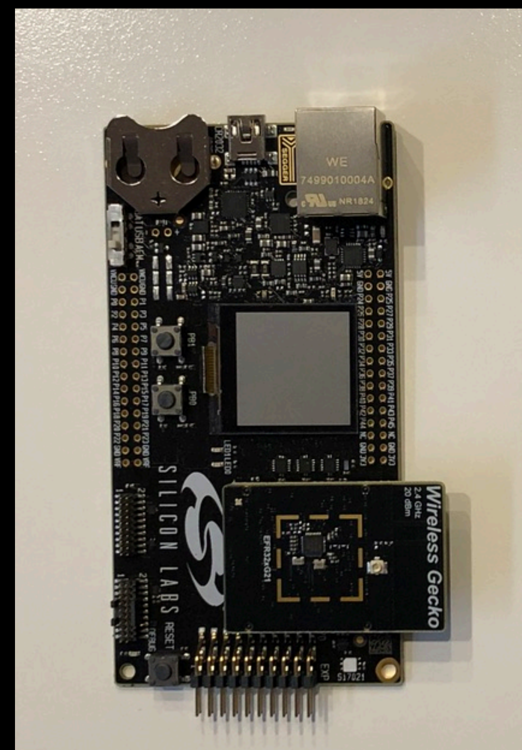
Example: SCAN_RSP PDU Type

Header		Payload	
0x0**4**	**0x02**	0x41	0x41

Next!

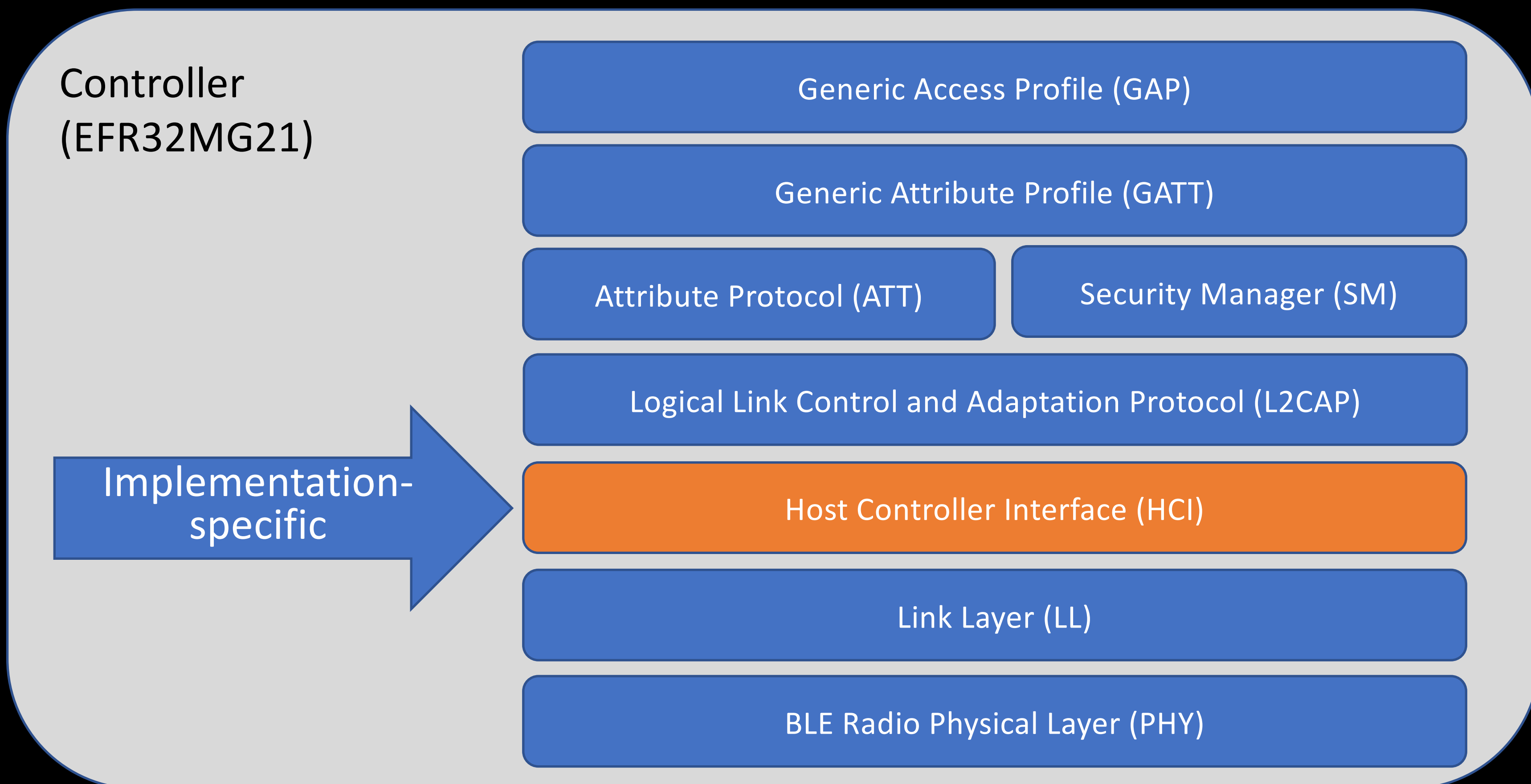
Target #2

- Silicon Labs EFR32MG21
- Supports BT 5 extended advertisements
- SWD debug interface is available
- Provides Simplicity Studio
  - BT stack comes as a library
  - Symbols are available, GOOD & … bad … no novel RE process to talk about ☺



Target #2

BLE stack in *single* chip configuration



Target #2

Fuzzing extended advertisements

- Fuzzer major update: had to move from Zephyr to NimBLE to start fuzzing extended advertisements
- Found DoS then fuzzed for a while but no crash
  - Ubertooth (SW) does not support the extended length advertisement packets
  - Sniffle does, thanks!
- NimBLE debugging? modified NimBLE scheduling code to send a large packet for longer time
- Soon after the NimBLE modification, CRASH!!

Target #2

Not every memory buffer overflow leads to RCE

Target #2

DoS: heap buffer overflow CVE-2020-15532

```
00021800        ldrb      r6,[r0,#0x6]      ; controlled by an attacker
…
0002180e        ldrb      r2,[r0,#0x7]      ; controlled by an attacker
00021810        sub       r2,r2,r6          ; integer underflow
…                                           ; but it's too large value
0002181a        add.w     r1,r6,#0xc
0002181e        add       r1,r0
00021820        sub       r0,r5,r6
00021822        add       r0,r1
00021824        bl        memmove           ; memory access violation
```

void *memmove(void *dest, const void *src, size_t n);

R0 R1 R2

Target #2

Difference from the target #1's RCE bug

```
ROM:0005B3A0    PUSH      {R4-R7,LR}      ; LR is stored on stack
ROM:0005B3A2    SUB.W     SP, SP, #0x2C   ; stack buffer
...                                       ; R6 is LL packet length
ROM:0005B3CE    SUBS      R6, R6, #6      ; integer underflow
ROM:0005B3D0    UXTB      R2, R6          ; unsigned byte extension
ROM:0005B3D2    ADD.W     R1, R5, #8      ; src, heap buffer address
ROM:0005B3D6    ADD.W     R0, SP, #9      ; dst, stack buffer address
ROM:0005B3DA    STRB.W    R2, [SP,#8]
ROM:0005B3DE    BL        memcpy
```

Target #2

RCE: heap buffer overflow CVE-2020-15531

- Neither pairing nor authentication is required
- Found a heap memory corruption via fuzzing, which leads to RCE, in extended advertisement packet parsing
- Packet data is chopped into a chained buffer, an entry holds max 0x45 bytes
- Length mis-calculation took place
- Manipulated the last byte of a memory chunk pointer
- With a heap spray, overwrote a function pointer
- Reported 2/21/2020, fixed 3/20/2020, Impressive!!

Target #2

Attack packet example

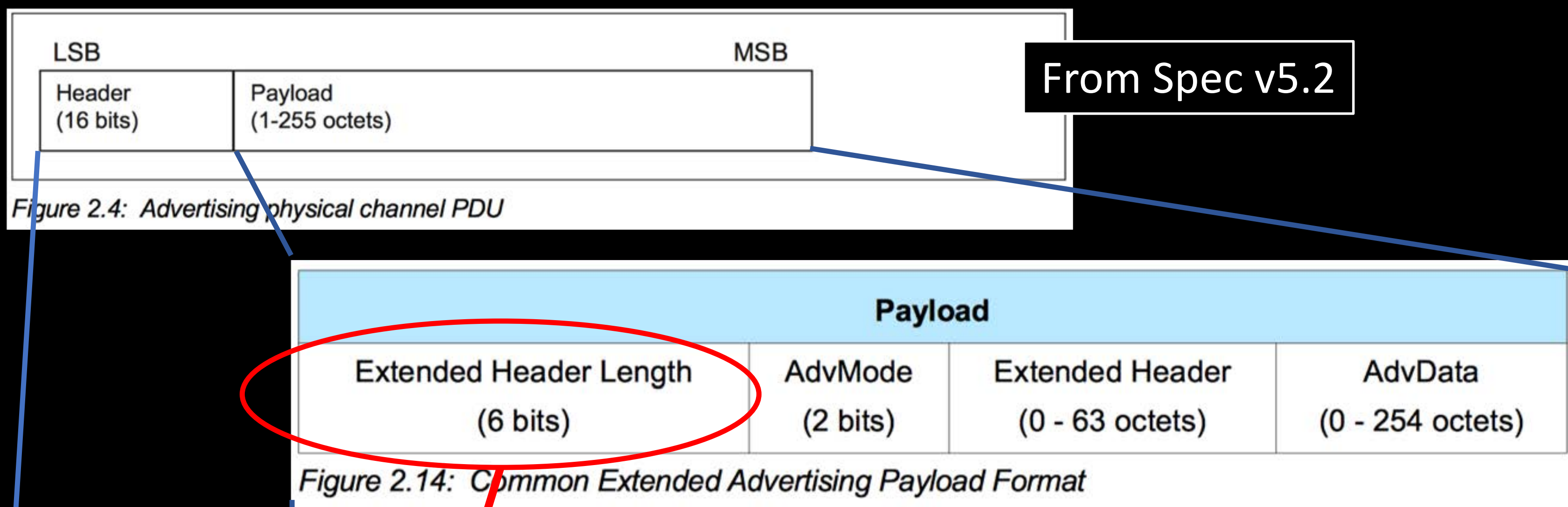


Figure 2.4: Advertising physical channel PDU

Figure 2.14: Common Extended Advertising Payload Format

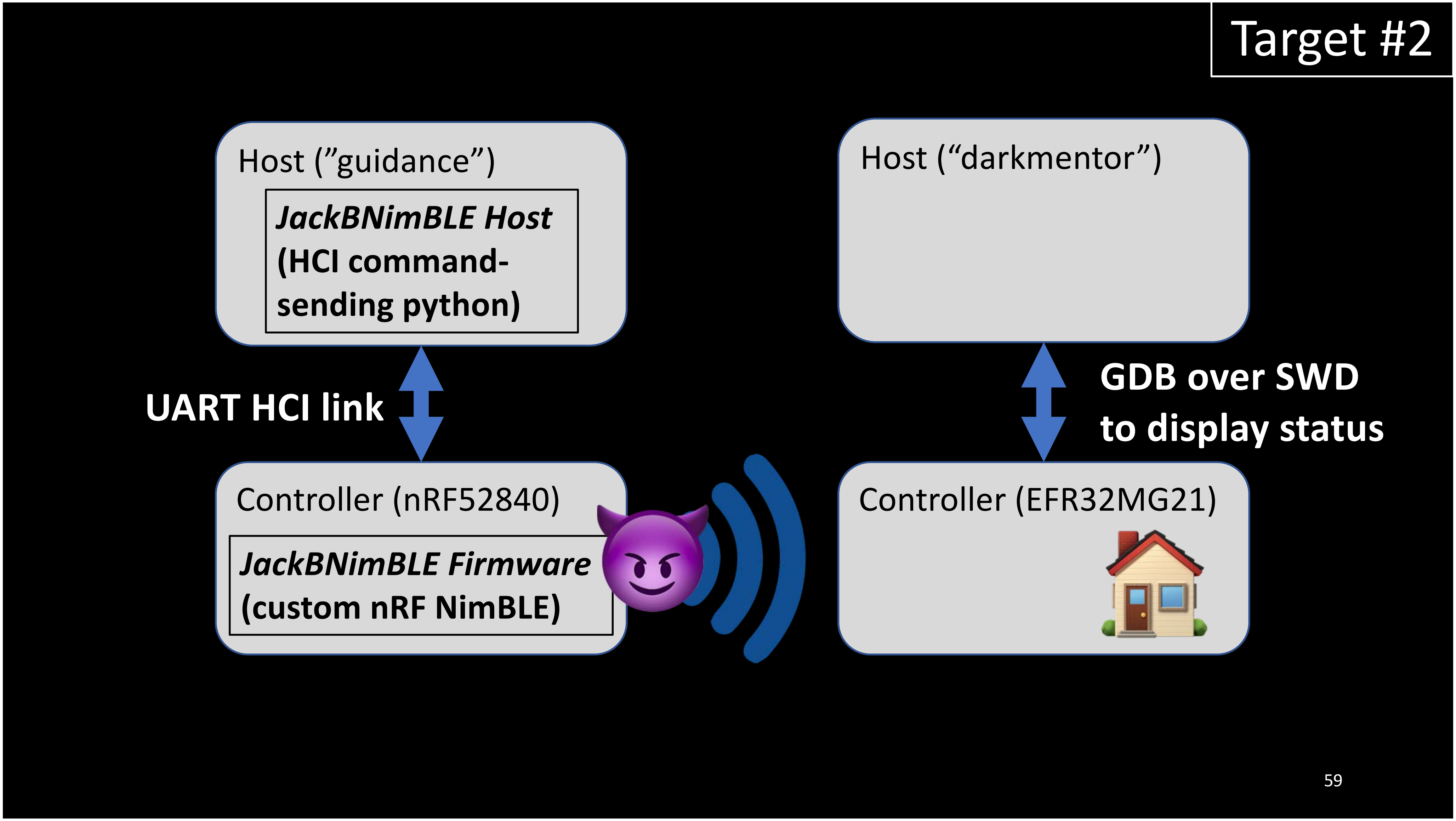
Example: ADV_EXT_IND Type, introduced on v5.0

Header		Payload								
0x0**7**	0xFF	**0x3C**	0x00	0x41	0x41	0x41	0x41	0x41	0x41	…

Target #2

RCE persistence demo

The successful attack is probabilistic


Target #2
Host (”guidance”)
JackBNimBLE Host
(HCI command-sending python)
UART HCI link
Controller (nRF52840)
JackBNimBLE Firmware
(custom nRF NimBLE)
Host (“darkmentor”)
GDB over SWD
to display status
Controller (EFR32MG21)

General BT security challenges:

BT security challenge 1:
Lack of all common exploit mitigations

- Stack Canaries
- Data Execution Prevention (DEP)
- Address Space Layout Randomization (ASLR)
- Return Oriented Programming (ROP) Prevention

…

BT security challenge 2:
SecureBoot

- Many chip vendors do not support secure boot or secure reset
- An exploit only has to work once for the attacker to have control forever
- Even if chip vendors support, it's up to the company who uses the chips in their end product to enable it
  - Silicon Labs' Gecko Bootloader does support secure boot
  - Hope that all Silabs' customers patched the vulnerability

BT security challenge 3:

Impact assessment

- How to assess the impact of a vulnerability
  - Difficult to identify which end products are vulnerable
  - Light bulbs vs. medical devices
- Customer information is often secret and it's up to the chip vendors to notify their customers
- Or even worse case: chip vendors -> packaging providers -> end product makers
- Some ways to find end products but it won't be the complete list
  - Googling with "site:fccid.io"
  - https://launchstudio.bluetooth.com/Listings/Search

For additional information

https://github.com/darkmentorllc

Thanks for valuable feedback!

Xeno Kovah

Rafal Wojtczuk

Marion Marschalek

Thank you…

Root

Lily

for watching!