Sep 21, 2023

Pentest Report - [redacted]

[redacted] Dongle

Block Harbor.
Cybersecurity

[redacted]

Contents

Provider Contact

Block Harbor Cybersecurity

Tristan Buffington, Lead Automotive Penetration Tester

tristan.buffington@blockharbor.io

Prepared on

September 21, 2023

Scope Summary

[redacted] Dongle

Prepared For

[redacted]

Proposal Reference

[redacted]

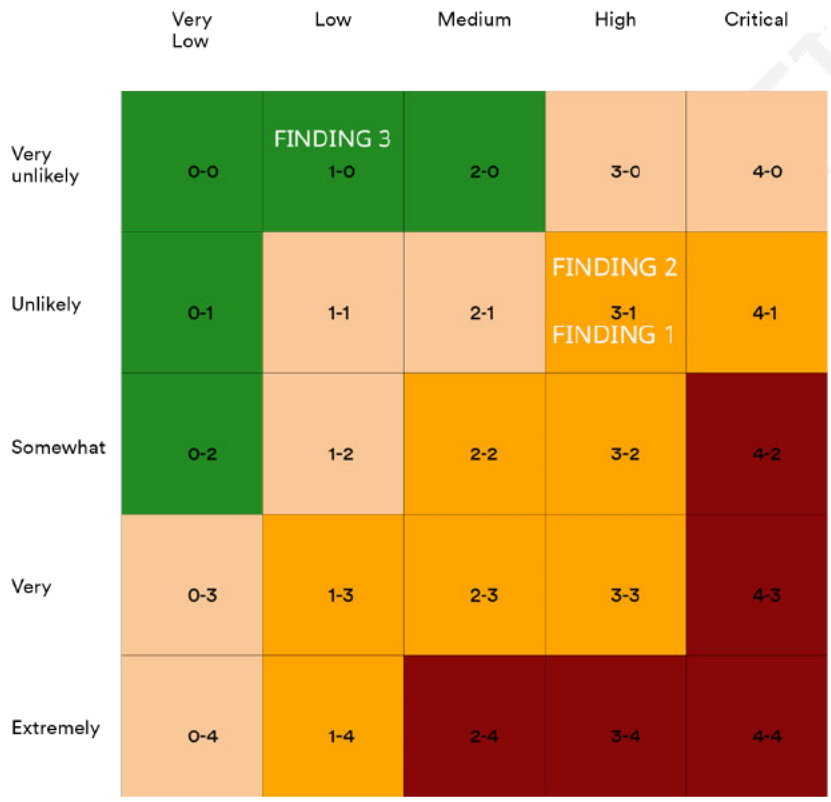
Timeline (est.)

5 Weeks

1 Executive Summary

Block Harbor performed a penetration test against the [redacted] dongle manufactured by [redacted]. Three security issues were found after code review, fuzzing, and device analysis. However, these required user interaction, as to interface with the file uploading service on the device, it is required to press the button on the case. Couple with that with weak debug protections allowing SWD, the overall risk rating is a medium due to the impact a device compromise could have on an individuals wallet, or vehicle.

1.1 Risk Rating Chart



1.2 Service Scope

The purpose of this testing was to evaluate and calculate risk associated with the target system by approaching the system from an attacker's perspective. Because systems often are complex with data flowing between different teams and organizations, the scope is clearly defined in a pre-engagement meeting to set boundaries for testing. The scope included:

- [redacted] Dongle

While the testing approach was guided by threat modeling and attack surface enumeration as a part of Block Harbor Cybersecurity's methodology, the assessment focused our attention and goals on the following components of the target:

- Firmware analysis
- Source code analysis
- Hardware debug interfaces
- SWD
- CAN
- Bluetooth
- LORA
- NFC
- Software Flashing and Update Process

1.3 Summary of Strengths

Number	1
Strength	Not vulnerable to tested Bluetooth CVE's
Description	After running off the shelf Bluetooth tests against the [redacted] dongle, it was observed that the device was not vulnerable to any of the tested CVE's described in the Bluetooth section.

Number	2
Strength	Not exploitable without user interaction.
Description	After analysis of the code, and crash found through fuzzing, it was observed that in order to exploit the issue, it would be necessary for a local user to press a physical button on the device, in order to send an exploit.

1.4 Summary of Weaknesses

Number	1
CVSSv3.1 Severity	5.6 - MEDIUM
Weakness	Weak debug protections
CVSS Reference	link
Description	A physically connected user is able to run arbitrary code and inspect the state of the machine, via the unprotected SWD interface.

Number	2
CVSSv3.1 Severity	5.5 - MEDIUM
Weakness	Stack Buffer Overflow
CVSS Reference	link
Description	A stack based buffer overflow was found in the code responsible for parsing DBC files. This could allow a local user to gain remote code execution on their device.

Number	3
CVSSv3.1 Severity	2.1 - LOW
Weakness	Denial of service
CVSS Reference	link
Description	A uniquely crafted DBC file could result in a denial of service to the device.

1.5 Strategic Recommendations

The following items are the strategic recommendations from Block Harbor.

- Locking down of the SWD interface.
- Enabling stack cookies by adding a new compile time flag (see Fixes in the Fuzzing section)

2 Security Assessment

The intention of a security assessment is to analyze hardware, software, or information. The security assessment reveals the findings similar to those than an attacker could achieve.

2.1 Organization

Block Harbor is a team of experts that united to provide security solutions where the cyber and physical worlds intersect. Block Harbor specializes in automotive & mobility ecosystems, cloud environments, industrial control systems, embedded & internet of things devices. As a leader in the Automotive Cybersecurity space, we have been trusted to provide top-quality assessments for automotive OEMs and suppliers.

Given a timeline of 5 weeks, Block Harbor Cybersecurity (BH) assessed security implementations, attempted to exploit the system, and organized findings into a report to yield insight from an attacker's perspective.

2.2 Methodology

Using tools at our disposal, Block Harbor Cybersecurity's methodology was derived from extensive threat modeling. This yielded a tailored approach with similar motivations to that of the adversaries that a system will face in a deployment. From a high-level, the security assessment will follow:

1) Information Gathering
2) Attack surface enumeration & passive reconnaissance
3) Security defense/protection check & vulnerability detection
4) Active scanning & vulnerability research
5) Deep testing & attempt to exploit
6) Assessment reporting

Further, this methodology allows us to perform consistent security assessments when the target system is seeking to implement a vehicle cybersecurity engineering methodology such as ISO/SAE 21434.

3 Test Narrative

3.1 Information Gathering

Block Harbor began the project by performing reconnaissance against the provided [redacted] device. This process involves extensive research, gathering as much information as possible before beginning active testing. Block Harbor performed standard reconnaissance against the source code provided and investigated the exposed SWD debug interface.

3.2 Attack Surface and Passive Reconnaissance

Block Harbor proceeded to build a threat model identifying attack vectors into the system and understanding the exposed services that may be of interest to an attacker as well as investigating the debug interfaces present on the hardware. Block Harbor discovered the points of interest to an attacker to be the CAN interface, the code responsible for parsing uploaded DBC files, Bluetooth, and SWD.

3.3 Security Defense Checks and Vulnerability Detection

Block Harbor enumerated the defensive posture of the [redacted] dongle and found the security to be well rounded overall with a minimal attack surface that can be explored in the provided threat model diagram in section 8.

3.4 Active Scanning and Vulnerability Research

Block Harbor then proceeded to map out the code in the device that would receive data from outside sources and read the code of various sub modules within the device. Block Harbor focused on areas of interest but noted that there was limited attack surface for what would be present to an attacker.

3.5 Deep testing and Exploitation

After discussion with the [redacted] team it was understood that no POC was necessary in order to demonstrate the impact of the found issues.

4 Software Flashing and Update Process

4.1 BLE OTA

Utilizing `mcumgr` that was installed via `go install github.com/apache/mynewt-mcumgr-cli/mcumgr@latest` Block Harbor was able to upload a new firmware revision without any user interaction on the hardware revision that was shipped to us.

```
sudo ~/go/bin/mcumgr --conntype ble -i 0 --connstring ctlr_name=hci1,
    peer_name=[redacted]4601590002' image upload ~/Documents/[redacted]
    [redacted].signed.bin
```

After, Block Harbor was able to switch to the newly uploaded image with:

```
sudo ~/go/bin/mcumgr --conntype ble -i 0 --connstring ctlr_name=hci1,
    peer_name='m[redacted]-24601590002' image test
    dedd63335a8f9249f7353638344bbf3390eadaa1765310a621589b2bfab0af2b
```

After, it was observed that this had deleted the first image slot and was now in the newly uploaded image.

```
sudo ~/go/bin/mcumgr --conntype ble -i 1 --connstring ctlr_name=hci1,
    peer_name=[redacted]-000000000000' image list
Images:
 image=0 slot=0
    version: 0.0.24
    bootable: true
    flags: active confirmed
    hash: [redacted]
Split status: N/A (0)
```

Finally, it was noted that it was impossible to now upload a new image with the same command.

4.2 SWD Flash

Using `pyocd` Block Harbor confirmed it is possible to flash different revisions of software with the following command:

```
python3 -m pyocd flash --frequency=4000000 -t nrf52840 ~/Downloads/zephyr.
    signed.hex
```

This was intended behavior. Attempting to flash unsigned code was unsuccessful:

```
python3 -m pyocd flash --frequency=4000000 -t nrf52840 ~/Downloads/zephyr.
    hex
```

5 Hardware Recommendations

5.1 Compliments to the Chef

Block Harbor does not have any hardware recommendations based on our understanding of the current hardware design.

6 SWD

6.1 Narrative

Block Harbor was able to connect to the SWD interface of the chip, by soldering to the correct pins and utilizing a STLINK as a programmer: the setup of which can be seen below:

Figure: Micro soldering

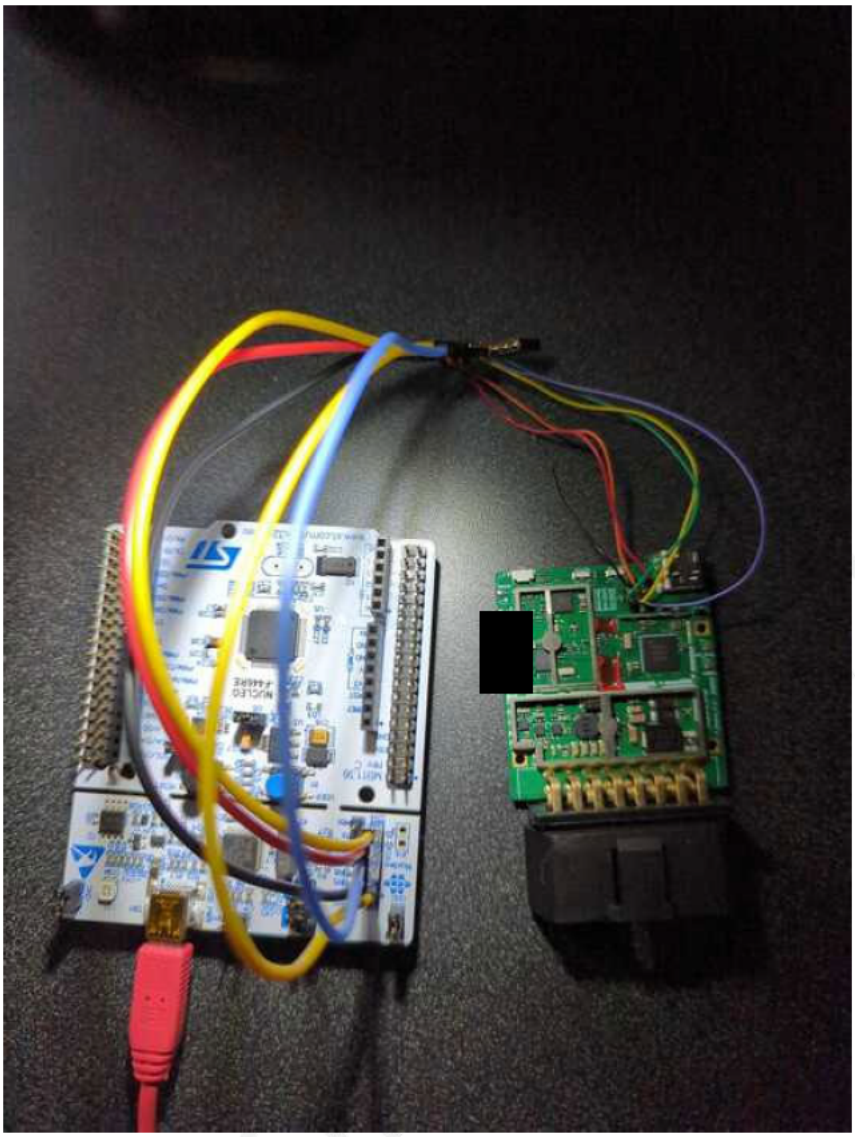


Figure: STLINK connection

This allowed full control of the device. This is currently intended behavior according to the [redacted] team. This was necessary as the provided cable to connect to SWD was non functional with our programmer.

6.2 Enumeration

Below is information gathered from the device.

```
> nrf5 info

[factory information control block]
```

```
code page size: 4096B
code memory size: 1024kB
code region 0 size: 0kB
pre-programmed code: not present
number of ram blocks: 4294967295
ram block 0 size: 4294967295B
ram block 1 size: 0B
ram block 2 size: 0B
ram block 3 size: 0B
config id: ffff01eb
device id: 0x382dc5e97cd390e0
encryption root: [redacted]
identity root: 0xc8f4e596fb38d77e504f1558221e85a8
device address type: 0xffffffff
device address: 0x9cbb603202cce260
override enable: ffffffff
NRF_1MBIT values: ffffffff ffffffff ffffffff ffffffff ffffffff
BLE_1MBIT values: ffffffff ffffffff ffffffff ffffffff ffffffff

[user information control block]

code region 0 size: 0kB
read back protection configuration: ffff
reset value for XTALFREQ: ff
firmware id: 0xffff
```

An example of modifying the memory of the application can be seen below. First, a gdbserver connection was started:

```
python3 -m pyocd gdbserver
```

Then the server was attached to and memory was shown to be written. This will allow for any code to be executed, bypassing the current secure boot protections:

```
gef:  gef-remote localhost 3333
gef:  hexdump byte 0x0

0x00000000     c0 6d 00 20 21 26 00 00 df 88 00 00 f5 25 00 00    .m.
   !&.......%..
0x00000010     f5 25 00 00 f5 25 00 00 f5 25 00 00 00 00 00 00
   .%...%...%......
0x00000020     00 00 00 00 00 00 00 00 00 00 00 00 0d 21 00 00
   .............!..
0x00000030     f5 25 00 00 00 00 00 00 b9 20 00 00 f5 25 00 00
   .%....... ...%..
gef:  set *0x0=0x0f

gef:  hexdump byte 0x0
```

```
0x00000000     0f 00 00 00 21 26 00 00 df 88 00 00 f5 25 00 00
    ....!&.......%..
0x00000010     f5 25 00 00 f5 25 00 00 f5 25 00 00 00 00 00 00
    .%...%...%......
0x00000020     00 00 00 00 00 00 00 00 00 00 00 00 0d 21 00 00
    .............!..
0x00000030     f5 25 00 00 00 00 00 00 b9 20 00 00 f5 25 00 00
    .%....... ...%..
```

It is understood that [redacted] will disable the SWD interface to protect against this kind of attack.

```
> flash banks
#0 : nrf52.flash (nrf5) at 0x00000000, size 0x00100000, buswidth 1,
    chipwidth 1
#1 : nrf52.uicr (nrf5) at 0x10001000, size 0x00001000, buswidth 1,
    chipwidth 1
```

7 Fuzzing

7.1 Narrative

Below are the results from Block Harbor's fuzzing campaign, using the test DBC file provided in the [redacted] project, as well as various DBC files from the opendbc. This was achieved by ripping out the DBC parsing code present in the Macaroon source tree, adding some defines from the Zephyr SDK, and then writing a small wrapper around the code. This was then instrumented with AFL++ and fuzzing was started.

```
int main(int argc, char** argv) {
        FILE* fp;
        uint8_t buf[1024];
        app_dbc_data_t _dbc_data = {0};

        fp = fopen(argv[1], "r");
        size_t rc = fread(buf, 1, sizeof(buf), fp);
        app_dbc_parse(buf, rc, &_dbc_data);
}
```

```
afl-clang-fast ./test.c ./app-dbc-utils.c
afl-fuzz -i ./dbcs/ -o ~/fuzz2-results/ -- ./a.out @@
```

7.2 Final AFL status

Block Harbor ran AFL++ on the target for 12 days. After 1 week of no new finds, Block Harbor stopped the fuzzing campaign. The reasoning behind this is after 1 week of no new results it is extremely unlikely to observe any new paths.

```
    American fuzzy lop ++4.09a {default} (./a.out) [fast]
        run time : 12 days, 0 hrs, 0 min, 29 sec
   last new find : 9 days, 17 hrs, 48 min, 16 sec
last saved crash : 7 days, 7 hrs, 53 min, 31 sec
 last saved hang : none seen yet
```

7.3 Fuzzing Triage

7.3.1 Stack based buffer overflow

After sufficient test cases were acquired, Block Harbor set to triage the crash. This was accomplished with the following compiler flags for our mock program.

```
gcc test.c app-dbc-utils.c -fsanatize=address
```

This allows Block Harbor to easily debug memory corruption issues. In this case, Block Harbor can follow along and see that the out of bounds write occurs in `get_next_non_empty_line`

```
~/fuzzing/a.out /home/ubuntu/fuzz2-results/default/crashes/id:000038,
    siubuntu@decomp:~/triage$ ~/fuzzing/a.out /home/ubuntu/fuzz2-results/
    default/crashes/id:000038,sig:06,src:00
0137,time:610480,execs:1221863,op:havoc,rep:25
=================================================================
==541172==ERROR: AddressSanitizer: stack-buffer-overflow on address 0
    x7ffe9d2e5360 at pc 0x55e7a32955b5 bp 0x7ffe9d2e5070 sp 0x7ffe9d2e5060
WRITE of size 1 at 0x7ffe9d2e5360 thread T0
    #0 0x55e7a32955b4 in get_next_non_empty_line (/home/ubuntu/fuzzing/a.
       out+0x25b4)
    #1 0x55e7a329726f in app_dbc_parse (/home/ubuntu/fuzzing/a.out+0x426f)
    #2 0x55e7a3297c20 in main (/home/ubuntu/fuzzing/a.out+0x4c20)
    #3 0x7f250dac7082 in __libc_start_main ../csu/libc-start.c:308
    #4 0x55e7a32953cd in _start (/home/ubuntu/fuzzing/a.out+0x23cd)

Address 0x7ffe9d2e5360 is located in stack of thread T0 at offset 576 in
    frame
    #0 0x55e7a32970dd in app_dbc_parse (/home/ubuntu/fuzzing/a.out+0x40dd)

  This frame has 2 object(s):
    [48, 52) 'parser_state' (line 443)
    [64, 576) 'line_buffer' (line 439) <== Memory access at offset 576
       overflows this variable
HINT: this may be a false positive if your program uses some custom stack
    unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/ubuntu/fuzzing/a.
    out+0x25b4) in get_next_non_empty_line
Shadow bytes around the buggy address:
  0x100053a54a10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54a20: 00 00 00 00 f1 f1 f1 f1 f1 f1 04 f2 00 00 00 00
  0x100053a54a30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54a40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54a50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100053a54a60: 00 00 00 00 00 00 00 00 00 00 00 00[f3]f3 f3 f3
  0x100053a54a70: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54a80: 00 00 f1 f1 f1 f1 f1 f1 00 00 00 00 00 00 00 00
  0x100053a54a90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54aa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100053a54ab0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
```

```
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==541172==ABORTING
```

It was observed that if the `src_buf_len` is greater than 512, we will achieve a stack based buffer overflow in the `get_next_non_empty_line` function as there is no check that this write is happening outside of the dst buffer from the following lines:

```
for (i = 0; i < src_buf_len, i < dst_buf_len; i++) {
    ---snip---
    *(line_buf + line_len) = c;
    line_len++;
}
```

See `id:000015,sig:06,src:000135,time:600084,execs:1201109,op:havoc,rep:36` for an example of a crash due to this out of bounds write.

7.3.2 Misc issues

Minor issues, such as one byte out of bounds reads were discovered, however these are not determined to be exploitable issues. One which results in a denial of service when attempting to parse the `signals.dbc` file. See `id:000005,sig:11,src:000067,time:8499,execs:18729,op:havoc,rep:2` for an example of a null ptr dereference.

All the associated files can be found attached to the report in the `testcases.zip` file.

7.4 Fixes

Various changes, such as checking the length of the destination buffer, and switching to `strncpy` will alleviate most of these issues. This should not be viewed as a comprehensive list of changes to make, but a starting point for further analysis. Note that line numbers in the diff file may not directly apply to the [redacted] source code.

```
diff --git a/.app-dbc.h.swp b/.app-dbc.h.swp
deleted file mode 100644
index b807734..0000000
Binary files a/.app-dbc.h.swp and /dev/null differ
diff --git a/app-dbc.h b/app-dbc.h
index 64cf2cb..40f8c62 100644
--- a/app-dbc.h
+++ b/app-dbc.h
@@ -102,6 +102,6 @@ int app_dbc_parse(const uint8_t *buf, size_t len,
    app_dbc_data_t *dbc_data);
 int app_dbc_decode_frame(const struct can_frame *frame, const
     app_dbc_data_t *dbc_data, app_dbc_data_value_t *out_values, size_t
     out_values_size);

 // for internal use, exposed for unit testing only
-size_t get_next_non_empty_line(const uint8_t *src_buf, size_t src_buf_len
    , uint8_t *line_buf);
+size_t get_next_non_empty_line(const uint8_t *src_buf, size_t src_buf_len
    , uint8_t *line_buf, size_t dst_buf_len);

 #endif //__APP_DBC_H
diff --git a/test.c b/test.c
index 33ad3e6..216d6af 100644
--- a/test.c
+++ b/test.c
@@ -17,13 +17,15 @@ enum line_parse_state_t {

 #define APP_DBC_CAN_EXTENDED_ID_MASK    (0x1FFFFFFFU)

-size_t get_next_non_empty_line(const uint8_t *src_buf, size_t src_buf_len
    , uint8_t *line_buf)
+size_t get_next_non_empty_line(const uint8_t *src_buf, size_t src_buf_len
    , uint8_t *line_buf, size_t dst_buf_len)
 {
     int i;
     size_t line_len = 0;
     enum line_parse_state_t parse_state = LINE_PARSE_LINE_START;

     for (i = 0; i < src_buf_len; i++) {
+       if (i >= dst_buf_len - 1)
+               break;
         char c = *(src_buf + i);

         if (parse_state == LINE_PARSE_LINE_START) {
@@ -99,7 +101,7 @@ int parse_msg_name(char **line_state, app_dbc_message_t
    *message)
     }
     //LOG_DBG("Line state: %s", *line_state);

-    strcpy(message->name, token);
```

```
+    strncpy(message->name, token,sizeof(message->name));
     //LOG_DBG("Name: %s", message->name);
     return APP_DBC_ERR_OK;
 }
@@ -170,7 +172,7 @@ int parse_sig_name(char **line_state,
    app_dbc_message_t *message, app_dbc_signal
     }
     //LOG_DBG("Line state: %s", *line_state);

-    strcpy(signal->name, token);
+    strncpy(signal->name, token, sizeof(signal->name));
     //LOG_DBG("Name: %s", signal->name);
     return APP_DBC_ERR_OK;
 }
@@ -365,7 +367,7 @@ int parse_sig_unit(char **line_state,
    app_dbc_message_t *message, app_dbc_signal
         }
         //LOG_DBG("Line state: %s", *line_state);

-        strcpy(signal->unit, token);
+        strncpy(signal->unit, token, sizeof(signal->unit));
     }

     //LOG_DBG("Unit: %s", signal->unit);
@@ -447,7 +449,7 @@ int app_dbc_parse(const uint8_t *buf, size_t len,
    app_dbc_data_t *dbc_data)
     dbc_data->messages_size = 0;

     while (len_left_to_parse > 0) {
-size_t parsed_bytes = get_next_non_empty_line(buf + buffer_index,
    len_left_to_parse, line_buffer);
+size_t parsed_bytes = get_next_non_empty_line(buf + buffer_index,
    len_left_to_parse, line_buffer, sizeof(line_buffer));
         len_left_to_parse -= parsed_bytes;
         buffer_index += parsed_bytes;
```

To mitigate the impact of any potential memory corruption, Block Harbor recommends the usage of stack cookies or canaries in order to protect the integrity of the stack. This is achieved by use of the `CONFIG_STACK_CANARIES` compile time flag.

8 Threat Model

8.1 Diagram

Block Harbor constructed a thread model to present at a high level where data flows in the system, and to highlight potential avenues of interest. This resulted in most of Block Harbor's efforts being spent investigating the physical debug ports, wireless interface, and areas of code that would parse user supplied input.

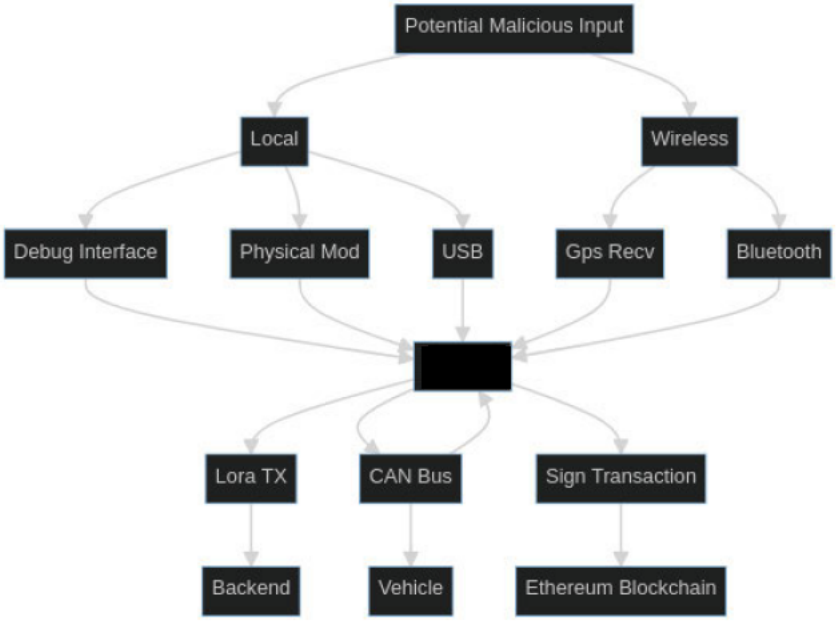


Figure: Threat model

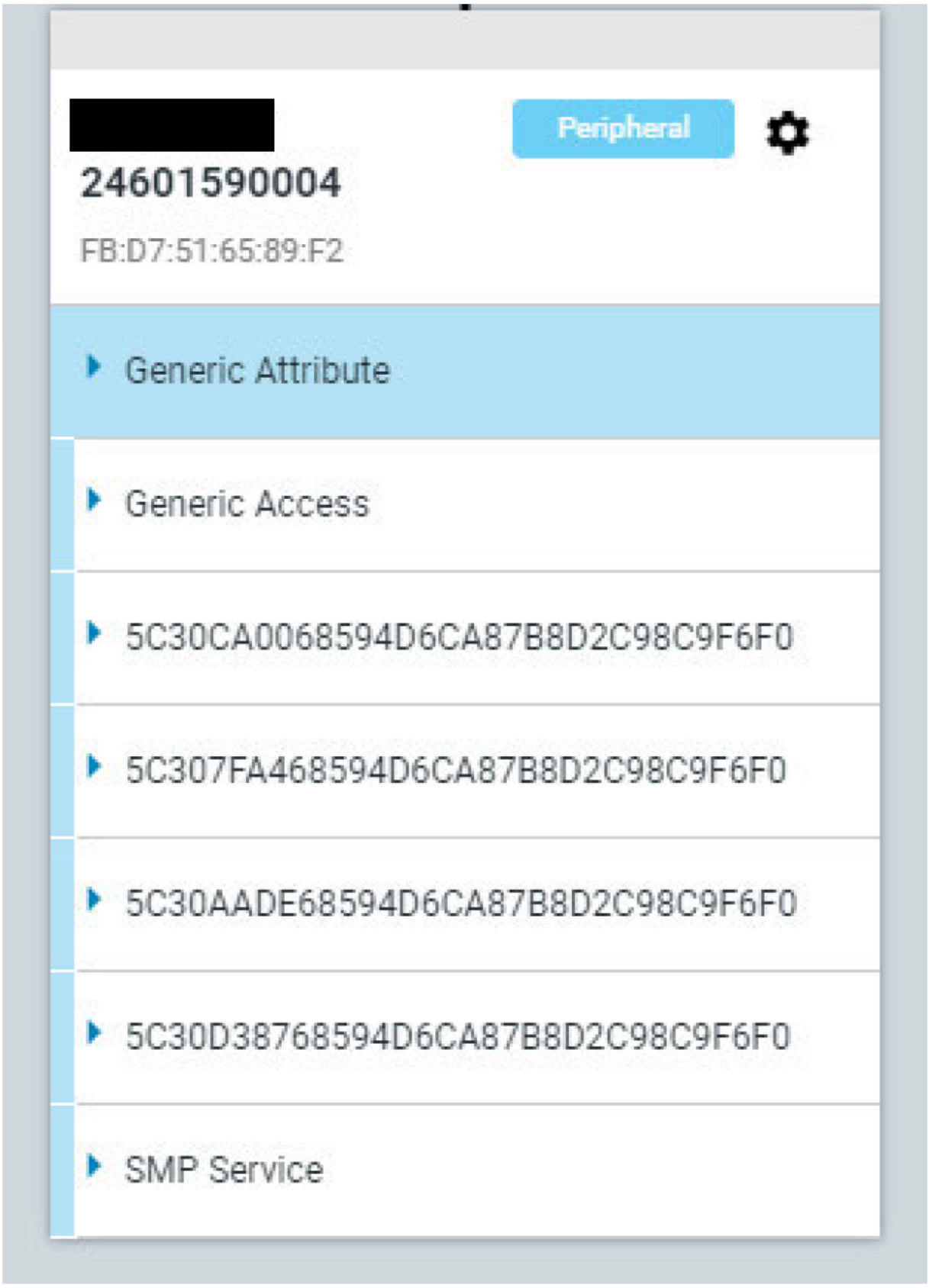
9 BLE Protocol

9.1 Narrative

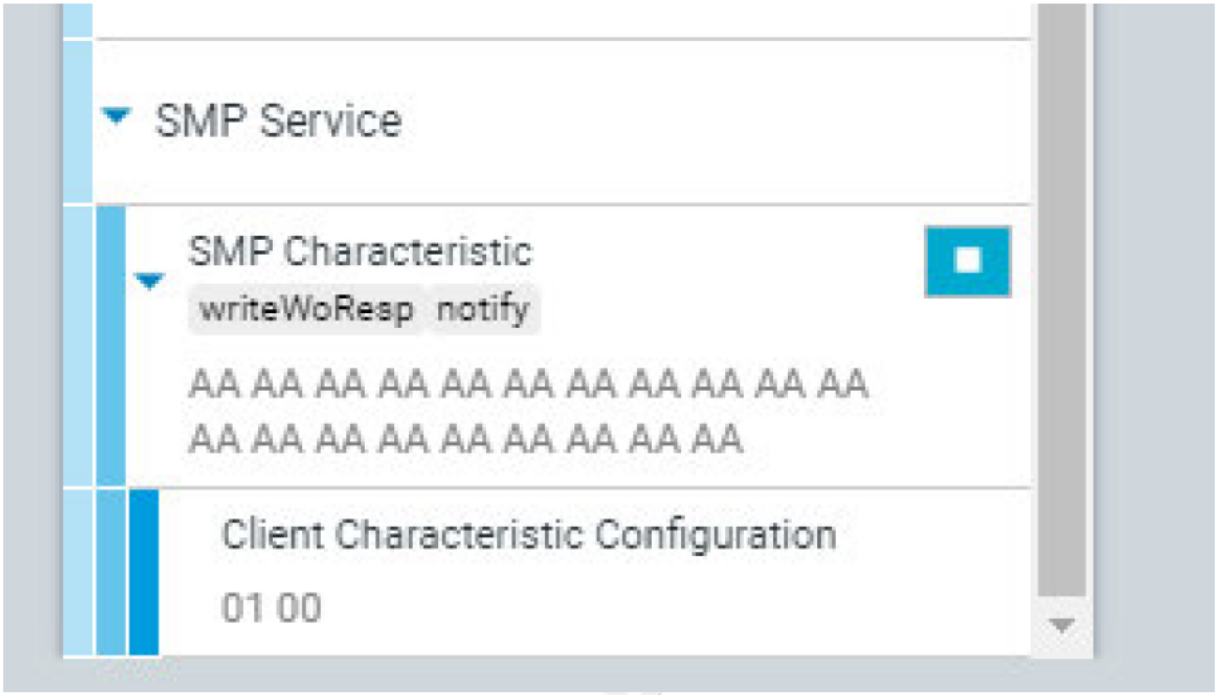
A thorough test was performed on the Bluetooth protocol utilizing an esp32 board, an nrf52840 board, the nrfconnect application, and publicly available Bluetooth exploits. It was noted that the on the software revision tested, the [redacted] device connects without pairing but the pairing authentication fails. The findings are mentioned below.

9.2 Enumeration

Utilizing nrfconnect tool, Block Harbor was able to connect to the [redacted] dongle and list out the services offered by the BLE protocol.


[redacted]
Peripheral
24601590004
FB:D7:51:65:89:F2
Generic Attribute
Generic Access
5C30CA0068594D6CA87B8D2C98C9F6F0
5C307FA468594D6CA87B8D2C98C9F6F0
5C30AADE68594D6CA87B8D2C98C9F6F0
5C30D38768594D6CA87B8D2C98C9F6F0
SMP Service

Testing SMP service functionality by fuzzing it with random bytes of data gave no results. It was noted that the SMP service only accepts 20 characters of data.



Using `bluetoothctl` Block Harbor was able to connect to the [redacted] dongle through BLE protocol without pairing or bonding.

```
[bluetooth]# connect FB:D7:51:65:89:F2
Attempting to connect to FB:D7:51:65:89:F2
[CHG] Device FB:D7:51:65:89:F2 Connected: yes
Connection successful
```

After successfully connecting to the dongle Block Harbor could enumerate the services that are present in the BLE protocol and the information of the services are in the figure below. However, none of the services were exploitable during testing.

```
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0001
        00001801-0000-1000-8000-00805f9b34fb
        Generic Attribute Profile
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0001/char0002
        00002a05-0000-1000-8000-00805f9b34fb
        Service Changed
[NEW] Descriptor (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0001/char0002/desc0004
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0001/char0005
        00002b29-0000-1000-8000-00805f9b34fb
        Client Supported Features
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0001/char0007
        00002b2a-0000-1000-8000-00805f9b34fb
        Database Hash
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0010
        5c30ca00-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0010/char0011
        5c30ca01-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0010/char0013
        5c30ca02-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Descriptor (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0010/char0013/desc0015
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0016
        5c307fa4-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0016/char0017
        5c305a11-6859-4d6c-a87b-8d2c98c9f6f0
```

```
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0016/char001d
        5c305a19-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0016/char001f
        5c305a18-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0021
        5c30aade-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0021/char0022
        5c301dd2-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0021/char0024
        5c30e60f-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0026
        5c30d387-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0026/char0027
        5c300acc-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service0026/char0029
        5c300add-6859-4d6c-a87b-8d2c98c9f6f0
        Vendor specific
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service002b
        8d53dc1d-1db7-4cd3-868b-8a527460aa84
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service002b/char002c
        da2e7828-fbce-4e01-ae9e-261174997c48
        Vendor specific
[NEW] Descriptor (Handle 0x0000)
        /org/bluez/hci0/dev_FB_D7_51_65_89_F2/service002b/char002c/desc002e
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
```

Block Harbor was able to enumerate the general information about the device using bluetoothctl after

connecting with the [redacted]

```
[macaron-24601590004]# info FB:D7:51:65:89:F2
Device FB:D7:51:65:89:F2 (random)
        Name:           ¼601590004
        Alias:          ?4601590004
        Paired: no
        Trusted: yes
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Vendor specific           (5c307fa4-6859-4d6c-a87b-8d2c98c9f6f0)
        UUID: Vendor specific           (5c30aade-6859-4d6c-a87b-8d2c98c9f6f0)
        UUID: Vendor specific           (5c30ca00-6859-4d6c-a87b-8d2c98c9f6f0)
        UUID: Vendor specific           (5c30d387-6859-4d6c-a87b-8d2c98c9f6f0)
        UUID: Vendor specific           (8d53dc1d-1db7-4cd3-868b-8a527460aa84)
[macaron-24601590004]#
```

9.3 Tested Bluetooth Exploits

9.3.1 Braktooth

Various exploits within the Braktooth test suite were tested against the [redacted] dongle and the results confirm that the [redacted] dongle is not vulnerable against these attacks.

9.3.2 Random Flooding (Not Vulnerable)

The [redacted] device is not vulnerable to the random byte flooding attack performed against the BLE services. The device did not seem to crash during the test.

```
❯ sudo bin/bt_exploiter --host-port=/dev/ttyUSB1 --target=FB:D7:51:65:89:F2 --exploit=au_rand_flooding
Logical Cores: 8
No SMT support
Assigned CPUSET:
CPU 0 Allowed
CPU 1 Allowed
CPU 2 Allowed
CPU 3 Allowed
CPU 4 Allowed
CPU 5 Allowed
CPU 6 Allowed
CPU 7 Allowed
sched_setscheduler: Current process set to realtime (RR Scheduler)
Thread priority is 99
/proc/sys/kernel/sched_rt_runtime_us = -1
Enabling Core dump: ulimit -c unlimited
Loading Model...
Model Loaded. Total States:169  Total Transistions:1299
Loop detection ENABLED
[Modules] Loading C++ Modules...
```

```
[BT Program] Starting program bin/sdp_rfcomm_query -u /dev/pts/6 -a FB:D7:51:65:89:F2 --iocap 3 --authreq 3 --bounding 1
Packet Log: logs/Bluetooth/hci_dump.pklg
H4 device: .
address=FB:D7:51:65:89:F2
iocap=3
authreq=3
bouding=1
Local version information:
- HCI Version    0x0008
- HCI Revision   0x030e
- LMP Version    0x0008
- LMP Subversion 0x030e
- Manufacturer 0x0060
Unknown manufacturer / manufacturer not supported yet.
Local name:
BTstack up and running on 46:D5:F1:2D:3A:39.
Starting RFCOMM Query
[Baseband] TX --> FHS
SDP query failed 0x04, retrying...
[Baseband] TX --> FHS
SDP query failed 0x04, retrying...
[Baseband] TX --> FHS
```

9.3.3 KNOB (Not Vulnerable)

This specific BLE attack attempts to exploit the key negotiation functionality in the BLE protocol, Block Harbor was able to confirm that the [redacted] dongle is not vulnerable against this attack.

```
Host BDAddress randomized to 82:31:2b:17:b9:70
[!] Global timeout started with 45 seconds
Packet Log: logs/Bluetooth/hci_dump.pklg
H4 device: /dev/pts/6
address=FB:D7:51:65:89:F2
iocap=3
authreq=3
bouding=1
Local version information:
- HCI Version    0x0008
- HCI Revision   0x030e
- LMP Version    0x0008
- LMP Subversion 0x030e
- Manufacturer 0x0060
Unknown manufacturer / manufacturer not supported yet.
Local name:
BTstack up and running on 82:31:2B:17:B9:70.
Starting RFCOMM Query
[Baseband] TX --> FHS
SDP query failed 0x04, retrying...
[Baseband] TX --> FHS
SDP query failed 0x04, retrying...
[Baseband] TX --> FHS
[Timeout] No Response received for 45 seconds
[Timeout] Target is not responding, check if target is still alive...
```

9.4 Sweyntooth Attack Suite

9.4.1 Summary

Various exploits within the Sweyntooth test suite were tested against the [redacted] dongle and the results confirm that the [redacted] dongle is not vulnerable against these attacks and has not crashed.

9.4.2 Attack script - Microchip_non_compliant connection (Not Vulnerable)

[redacted] did not crash during this test and Block Harbor confirmed to be not exploitable against this attack.

```
 130 ❱ sudo ./docker.sh run extras/Microchip_and_others_non_compliant_connection.py /dev/ttyACM0 FB:D7:5
1:65:89:F2
Serial port: /dev/ttyACM0
Advertiser Address: FB:D7:51:65:89:F2
TX ---> BTLE_ADV / BTLE_SCAN_REQ
Waiting advertisements from fb:d7:51:65:89:f2
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
No advertisement from FB:D7:51:65:89:F2 received
The device may have crashed!!!
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
```

9.4.3 Attack script - anomaly_unexptected_encryption_start (Not Vulnerable)

[redacted] dongle did not crash during this test and Block Harbor confirmed to be not exploitable against this attack.

```
 2 ❱ sudo ./docker.sh run extras/anomaly_unexpected_encryption_start.py /dev/ttyACM0 FB:D7:51:65:89:F2
Using IOCap: 0x3, Auth REQ: 0x1
Serial port: /dev/ttyACM0
Advertiser Address: FB:D7:51:65:89:F2
TX ---> BTLE_ADV / BTLE_SCAN_REQ
Waiting advertisements from FB:D7:51:65:89:F2
[!] Timeout
TX ---> BTLE_ADV / BTLE_SCAN_REQ
[!] Timeout
TX ---> BTLE_ADV / BTLE_SCAN_REQ
[!] Timeout
TX ---> BTLE_ADV / BTLE_SCAN_REQ
[!] Timeout
TX ---> BTLE_ADV / BTLE_SCAN_REQ
```

9.4.4 Attack script - connection_req_crash_truncated (Not vulnerable)

[redacted] dongle did not crash during this test and Block Harbor confirmed to be not exploitable against this attack.

```
 130 ❯ sudo ./docker.sh run extras/CC2540_connection_req_crash_truncated.py /dev/ttyACM0 FB:D7:51:65:89:
F2
Serial port: /dev/ttyACM0
Advertiser Address: FB:D7:51:65:89:F2
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
Waiting advertisements from fb:d7:51:65:89:f2
No advertisement from FB:D7:51:65:89:F2 received
The device may have crashed!!!
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
TX ---> BTLE_ADV / BTLE_SCAN_REQ
```

10 Appendix

10.1 Acronyms

- SWD: Serial Wire Debug
- OSINT: Open Source Intelligence
- CVE: Common Vulnerabilities and Exposures
- PCB: Printed Circuit Board
- CAN: Controller Area Network

Business Details

WeWork
Merchants Row
19 Clifford Street
Detroit, MI 48226
9 AM - 5 PM
313.246.1860

750 Letica Drive
Rochester, MI 48307
9 AM - 5 PM
313.246.1860

Contact

313.246.1860
contactus@blockharbor.io

Learn More

blockharbor.io