```
Date: 1/8/23, 10:05 PM
Process: bluetoothd
Bundle id: (null)
Device: iPhone 11, iOS 13.6

Exception type: EXC_SOFTWARE (SIGABRT)
Exception subtype: EXC_UNIX_ABORT
Exception codes: 0x0000000000010002, 0x0000000000000000
Culprit: Unknown

Triggered by thread: 4
Thread name: Dispatch queue: AACPService
Call stack:
0   libsystem_kernel.dylib              0x0000000194a34df0 0x194a0e000 + 159216        // __pthread_kill
1   libsystem_pthread.dylib             0x0000000194954948 0x194952000 + 10568         // pthread_kill
2   libsystem_c.dylib                   0x00000001948e3c24 0x19486e000 + 482340        // __abort
3   libsystem_c.dylib                   0x00000001948c3d98 0x19486e000 + 351640        // a641
4   bluetoothd                          0x000000010100697c 0x100dc0000 + 2386300       // func_1002464dc
5   bluetoothd                          0x0000000101004bd8 0x100dc0000 + 2378712       // func_1002448ec
6   bluetoothd                          0x0000000101002da8 0x100dc0000 + 2370984       // func_100242c88
7   bluetoothd                          0x0000000100f59d14 0x100dc0000 + 1678612       // func_100199bf4
8   bluetoothd                          0x000000010107de80 0x100dc0000 + 2875008       // func_1002bde5c
9   libdispatch.dylib                   0x00000001948ef5ac 0x1948ec000 + 13740         // _dispatch_client_callout
10  libdispatch.dylib                   0x00000001948f5a64 0x1948ec000 + 39524         // _dispatch_lane_serial_drain
11  libdispatch.dylib                   0x00000001948f64cc 0x1948ec000 + 42188         // _dispatch_lane_invoke
12  libdispatch.dylib                   0x00000001948ffa5c 0x1948ec000 + 80476         // _dispatch_workloop_worker_thread
13  libsystem_pthread.dylib             0x0000000194955718 0x194952000 + 14104         // _pthread_wqthread
14  libsystem_pthread.dylib             0x000000019495b9c8 0x194952000 + 39368         // start_wqthread
```

Bluetoothd crashed due to __stack_chk_fail triggered.
Clearly there is a stack overflow vulnerability in the function.

```
0000010100[illegible] 61 1B 00 D0 21 40 2D 91                ADRL    X1, stru_101374B50 ; block
0000010100[illegible] AC 8F 09 94                            BL      _dispatch_once
0000010100[illegible] ED FE FF 17                            B       loc_101006528
0000010100[illegible]                        ; ---------------------------------------------------------------------------
0000010100[illegible]
0000010100[illegible]                        loc_101006978                           ; CODE XREF: sub_1010064DC+1C4↑j
0000010100[illegible] 66 8E 09 94                            BL      ___stack_chk_fail
0000010100[illegible]                        ; End of function sub_1010064DC
0000010100[illegible]
0000010100[illegible]                        ; ---------------------------------------------------------------------------
0000010100697C 9C FC FF FF                   jpt_101006614   DCD loc_101006618 - 0x10100697C
0000010100697C                                                                       ; DATA XREF: sub_1010064DC+128↑o
0000010100697C                                                                       ; jump table for switch statement
0000010100[illegible] 28 FE FF FF                            DCD loc_1010067A4 - 0x10100697C ; jumptable 0000000101006614 case 1
0000010100[illegible] 44 FD FF FF                            DCD loc_1010066C0 - 0x10100697C ; jumptable 0000000101006614 case 2
0000010100[illegible] 9C FD FF FF                            DCD loc_101006718 - 0x10100697C ; jumptable 0000000101006614 case 3
0000010100698C F0 FD FF FF                                   DCD loc_10100676C - 0x10100697C ; jumptable 0000000101006614 case 4
0000010100[illegible] 6C FE FF FF                            DCD loc_1010067E8 - 0x10100697C ; jumptable 0000000101006614 case 5
0000010100[illegible] C0 FE FF FF                            DCD loc_10100683C - 0x10100697C ; jumptable 0000000101006614 case 6
0000010100[illegible]
0000010100[illegible]                        ; =============== S U B R O U T I N E =======================================
```

```
00000101006618                                          ; DATA XREF: __text:jpt_101006614↓o
00000101006618 48 4E 80 52      MOV     W8, #0x272 ; jumptable 0000000101006614 case 0
0000010100661C 08 01 13 CB      SUB     X8, X8, X19
00000101006620 7F C6 09 71      CMP     W19, #0x271
00000101006624 E1 83 88 9A      CSEL    X1, XZR, X8, HI ; size_t
00000101006628 F7 83 00 91      ADD     X23, SP, #0x2E0+__dst
0000010100662C E0 02 13 8B      ADD     X0, X23, X19 ; void *
00000101006630 BC 8F 09 94      BL      _bzero
00000101006634 E0 83 00 91      ADD     X0, SP, #0x2E0+__dst ; __dst
00000101006638 E1 03 16 AA      MOV     X1, X22 ; __src
0000010100663C E2 03 13 AA      MOV     X2, X19 ; __n
00000101006640 AC 91 09 94      BL      _memcpy
00000101006644 E0 23 00 91      ADD     X0, SP, #0x2E0+var_2D8
00000101006648 A1 00 80 52      MOV     W1, #5
0000010100664C E2 03 15 AA      MOV     X2, X21
00000101006650 CF FB 01 94      BL      sub_10108558C
00000101006654 F7 0F 00 F9      STR     X23, [SP,#0x2E0+var_2C8]
```

Using IDA Pro's tracing features, I located the following code.

The size of __dts buffer is 0x278 bytes.
I noticed that bluetoothd crashes with v35 == 0x27A, causing the lowest two bytes of the canary to be overwritten.

```
112  v44 = a4 + 4;
113  switch ( (int)v36 )
114  {
115    case 0:
116      if ( (unsigned int)v35 <= 0x271 )
117        v45 = 626 - v35;
118      else
119        v45 = 0LL;
120      bzero((char *)&__dst + v35, v45);
121      memcpy(&__dst, v44, v35);
122      v46 = sub_10108558C(v59, 5LL, v40);
123      p_dst = &__dst;
124      if ( v41 )
125        v68 |= v41;
126      goto LABEL_54;
127    case 1:
128      if ( (unsigned int)v35 <= 0x1D5 )
129        v52 = 470 - v35;
```

```
00000101006618                                          ; DATA XREF: __text:jpt_101006614↓o
00000101006618 48 4E 80 52      MOV     W8, #0x272 ; jumptable 0000000101006614 case 0
0000010100661C 08 01 13 CB      SUB     X8, X8, X19
00000101006620 7F C6 09 71      CMP     W19, #0x271
00000101006624 E1 83 88 9A      CSEL    X1, XZR, X8, HI ; size_t
00000101006628 F7 83 00 91      ADD     X23, SP, #0x2E0+__dst
0000010100662C E0 02 13 8B      ADD     X0, X23, X19 ; void *
00000101006630 BC 8F 09 94      BL      _bzero
00000101006634 E0 83 00 91      ADD     X0, SP, #0x2E0+__dst ; __dst
00000101006638 E1 03 16 AA      MOV     X1, X22 ; __src
0000010100663C E2 03 13 AA      MOV     X2, X19 ; __n
00000101006640 AC 91 09 94      BL      _memcpy
00000101006644 E0 23 00 91      ADD     X0, SP, #0x2E0+var_2D8
00000101006648 A1 00 80 52      MOV     W1, #5
0000010100664C E2 03 15 AA      MOV     X2, X21
00000101006650 CF FB 01 94      BL      sub_10108558C
00000101006654 F7 0F 00 F9      STR     X23, [SP,#0x2E0+var_2C8]
```

The easiest way to fix that is to fix the third argument of memcpy to 0x271 to ensure that no stack overflow occurs

```
for(uint64_t i=0x200000;i<0x300000;i+=4){
    uint64_t target=_dyld_get_image_vmaddr_slide(0)+0x100000000+i;
    if(*(uint32_t*)target==0x52804e48 && *(uint32_t*)(target+8)==0x7109c67f){ //find it
        NSLog(@"find it!!!");
        NSLog(@"%llx",target);
        MSHookMemory((void*)(target+0x24), patch, sizeof(patch)); //patch MOV X2,X 19 -> MOV X2,0x271
        break;
    }
}
```