CVE-2020-0022漏洞简析

李敬
Li Jing
lixeon.com
202028018670021
lixeon.lij@gmail.com
20210409
辛丑二月廿八

Credit with Chang Yue & Seebug

目录
CONTENTS

一、漏洞介绍

二、漏洞原理

三、漏洞利用

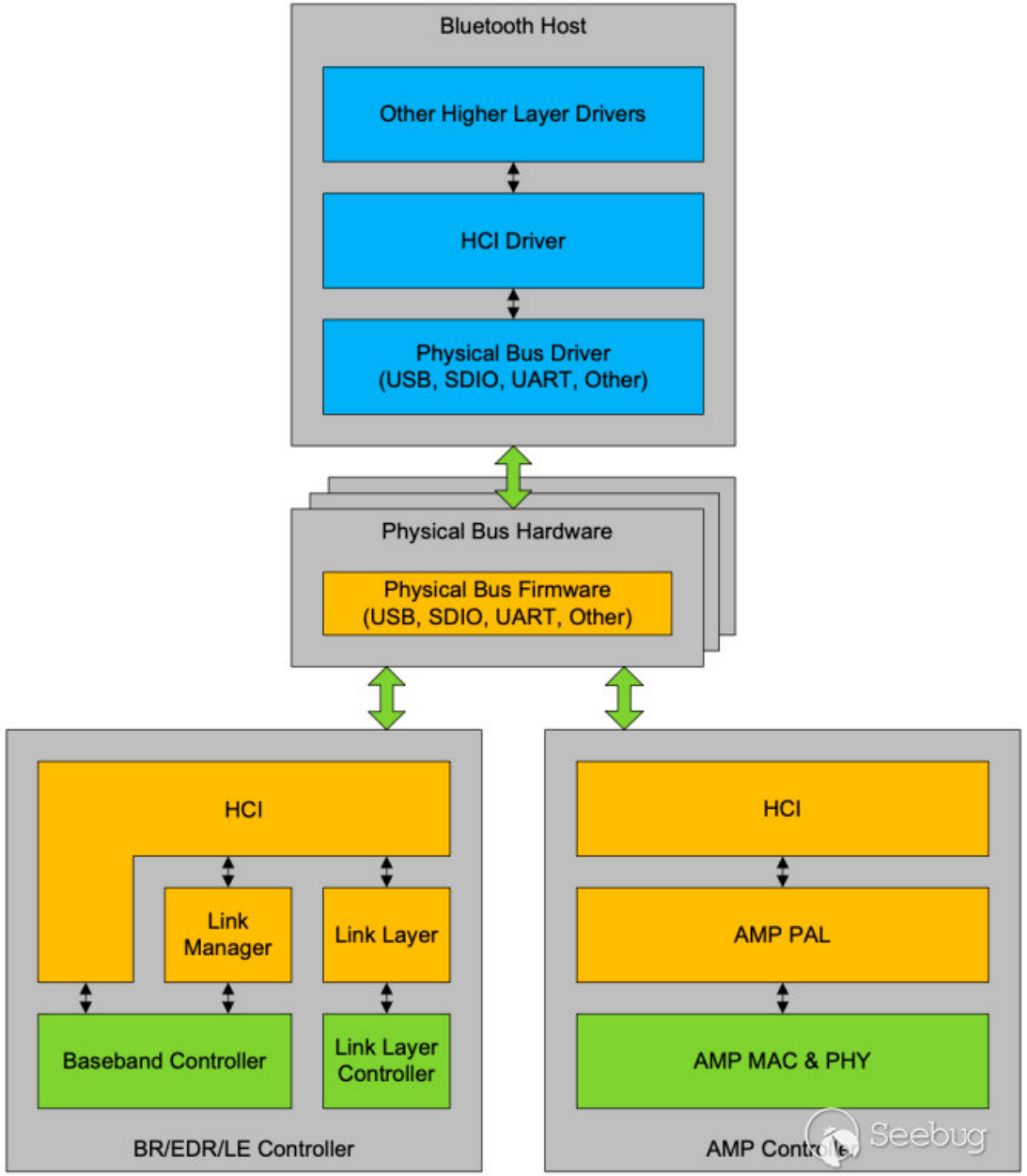
四、漏洞修复

Background

2020年2月，Android安全公告中披露并修复了一个严重漏洞，漏洞编号为CVE-2020-0022，又称BlueFrag，可影响Android蓝牙子系统。该漏洞是一个远程代码执行漏洞，出现在Bluedroid蓝牙协议栈的HCI层，当无线模块处于活动状态时，攻击者可以利用蓝牙守护程序提升权限进而在设备上执行代码。该漏洞影响Android Oreo（8.0和8.1）、Pie（9），但无法在Android 10上进行利用，仅能触发DoS攻击。

1.漏洞介绍

- CVE-2020-0022漏洞
- 又称BlueFrag
- 影响Android蓝牙子系统
- 利用缓冲区溢出实现远程代码执行
- 出现在蓝牙协议栈的HCI层
- 影响Android10以前版本

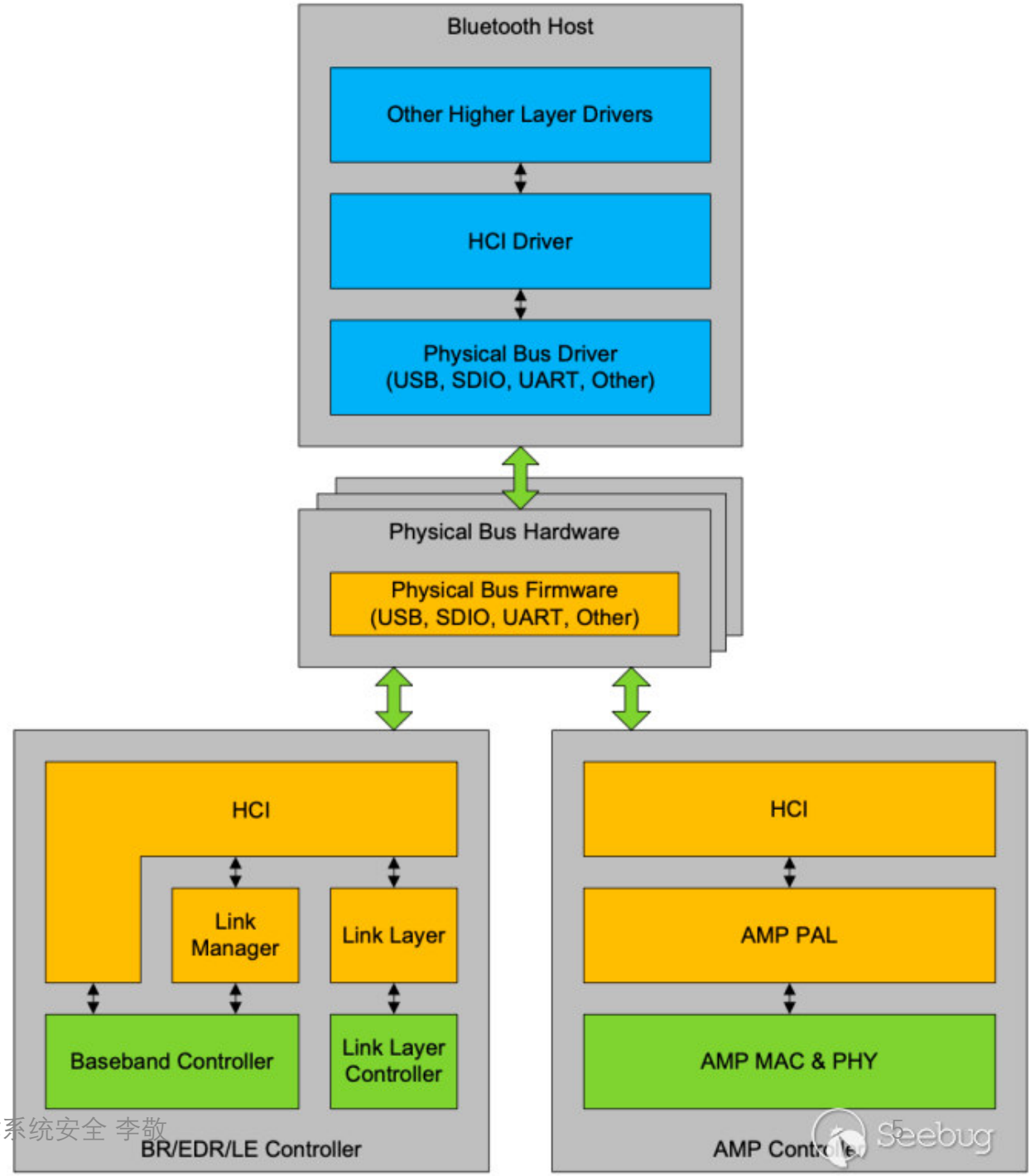


HCI 层位于蓝牙协议栈高层协议和低层协议之间，提供了对基带控制器和链路管理器的命令以及访问蓝牙硬件的统一接口方法，其接口适用于BR/EDR控制器、BR/EDR/LE控制器、LE控制器、AMP控制器，与底层的结构关系如图

Basic - HCI
主机控制接口协议

HCI 层位于蓝牙协议栈高层协议和低层协议之间，提供了对基带控制器和链路管理器的命令以及访问蓝牙硬件的统一接口方法，其接口适用于BR/EDR控制器、BR/EDR/LE控制器、LE控制器、AMP控制器，与底层的结构关系如图

主机系统上的HCI驱动程序和控制器中的HCI层之间会存在中间层， 这些中间层即是主机控制器传输层，这些传输层是透明的，只需完成传输数据的任务，不必清楚数据的具体格式。



Basic - 两个蓝牙设备点对点HCI层的交互过程

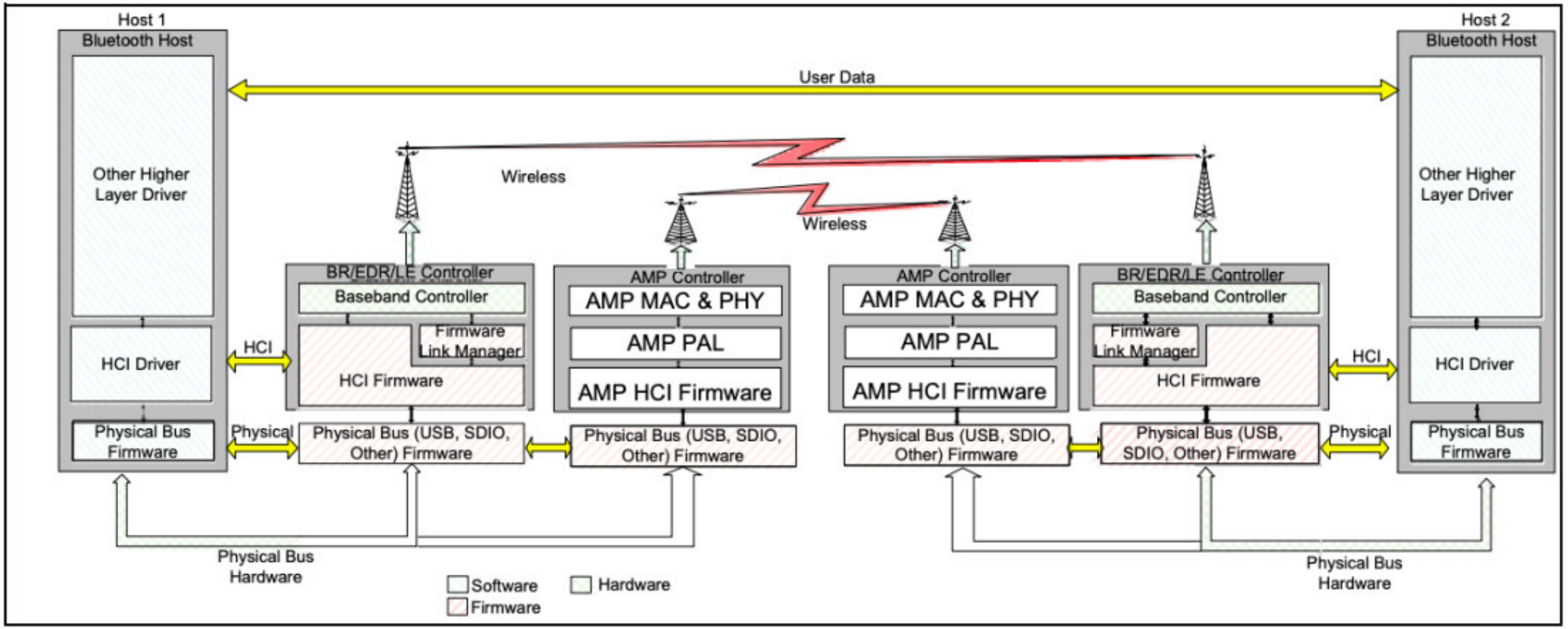


Figure 1.2: End to End Overview of Lower Software Layers to Transfer Data

HCI包通信

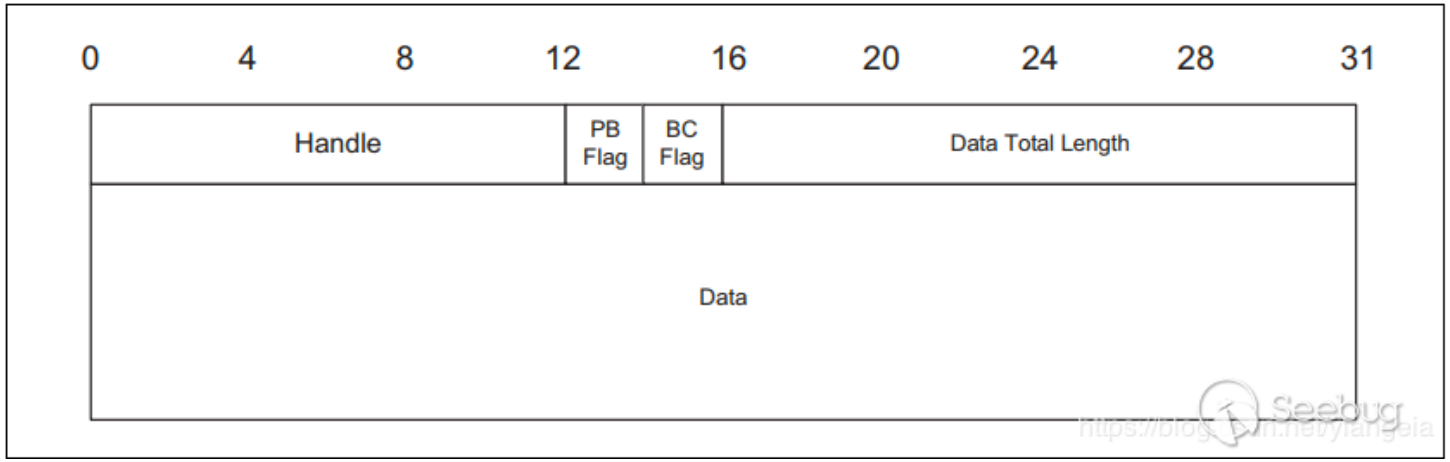
HCI通过包的方式来传送数据、命令和事件的，所有在主机和主机控制器之间的通信都以包的形式进行。包括每个命令的返回参数都通过特定的事件包来传输。
HCI有数据、命令和事件三种类型的包。

命令包COMMAND（0x01）只能从主机发往主机控制器

其中数据包是双向的，分为两类：ACL（0x02）、SCO（0x03）

而事件包EVENT（0x04）始终是主机控制器发向主机的。主机发出的大多数命令包都会触发主机控制器产生相应的事件包作为响应，在传输过程中会有一个句柄，用于识别主机之间的逻辑通道和控制器，共有三种类型的句柄：连接句柄、逻辑链路句柄和物理链路句柄。

Handle：Connection_Handle用于在主控制器上传输数据包或段
PB Flag：包边界和适应范围
BC Flag：广播标志
Data Total Length：以八位位组为单位的数据长度，包含高层协议data



ACL数据包

PB_Flag

设置为 00'b 的时候，代表 Host -> Contoller 的 L2CAP 的首包。

设置为 01'b 的时候，代表 Host -> Contoller 或者 Contoller -> Host 的 L2CAP 的续包（中间的）。

设置为 10'b 的时候，代表 Contoller -> Host 的 L2CAP 的首包。

Packet_Boundary_Flag: *Size: 2 Bits*

Value	Parameter Description		ACL-U	AMP-U	LE-U
00	First non-automatically-flushable packet of Higher Layer Message (start of a non-automatically-flushable L2CAP PDU) from Host to Controller. Shall not be used on a BR/EDR Controller from Controller to Host.	Host to Controller	Allowed	Not allowed	Allowed
		Controller to Host	Not allowed (except during loopback)	Not allowed	Not allowed
01	Continuing fragment of Higher Layer Message	Host to Controller	Allowed	Not allowed	Allowed
		Controller to Host	Allowed	Not allowed	Allowed
10	First automatically flushable packet of Higher Layer Message (start of an automatically-flushable L2CAP PDU).	Host to Controller	Allowed	Not allowed	Not Allowed
		Controller to Host	Allowed	Not allowed	Allowed
11	A complete L2CAP PDU. Automatically flushable.	Host to Controller	Allowed	Allowed	Not Allowed
		Controller to Host	Not Allowed	Allowed	Not Allowed

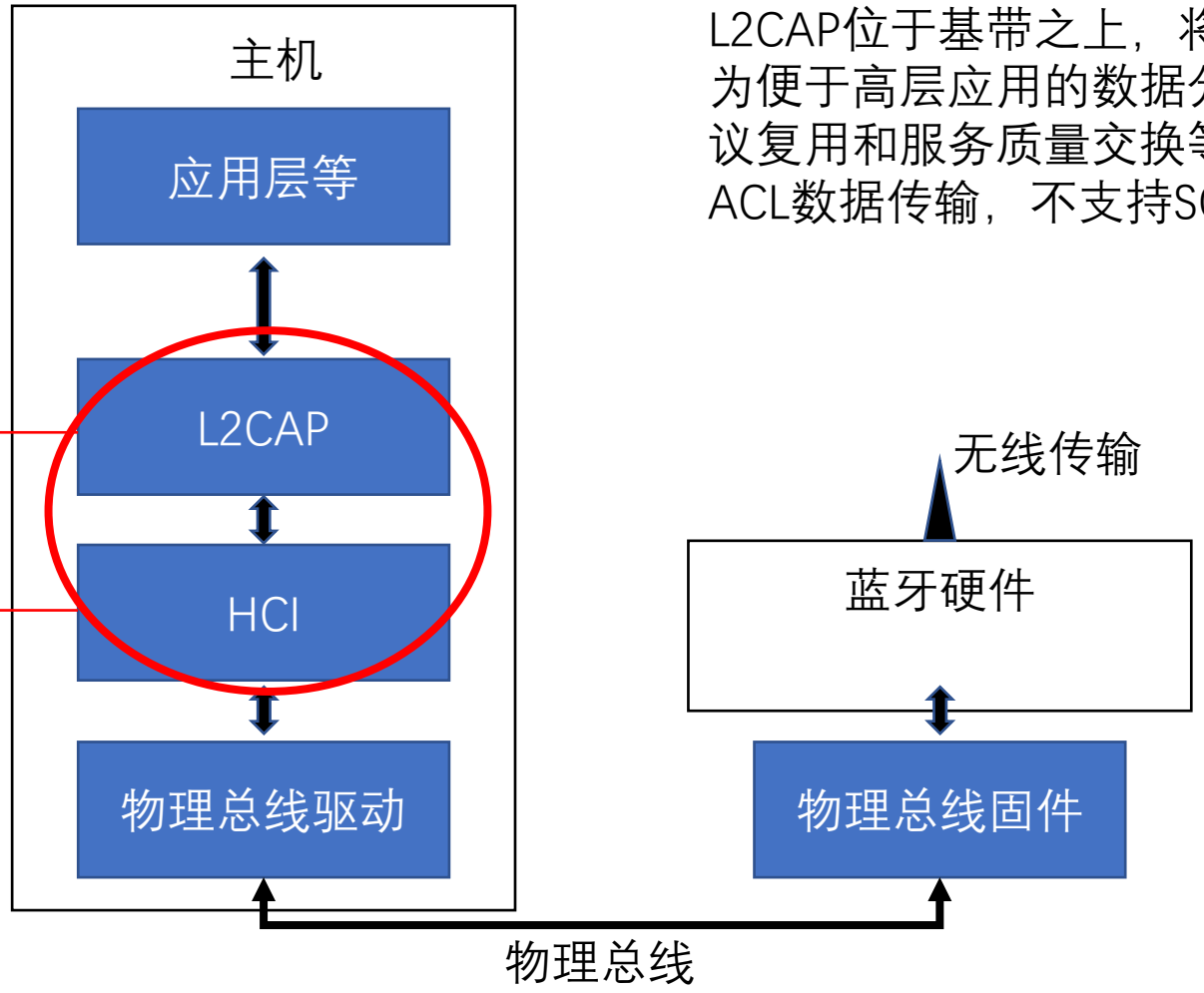
2.蓝牙通信架构

L2CAP：逻辑链路控制与适配协议

L2CAP位于基带之上，将基带的数据分组转换为便于高层应用的数据分组格式，并提供协议复用和服务质量交换等功能。L2CAP只支持ACL数据传输，不支持SCO数据。

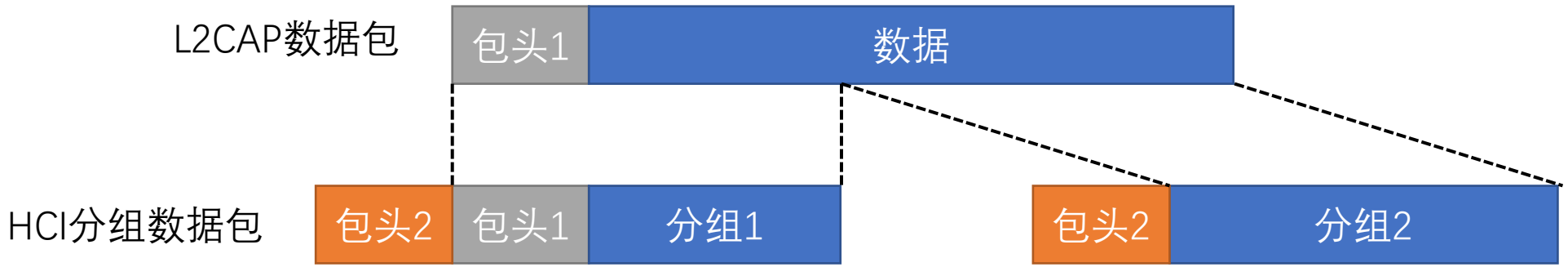
1. 信道复用
2. ACL信道中数据分组和重组

主机通讯端口



3.分段Fragmentation与重组Reassembly

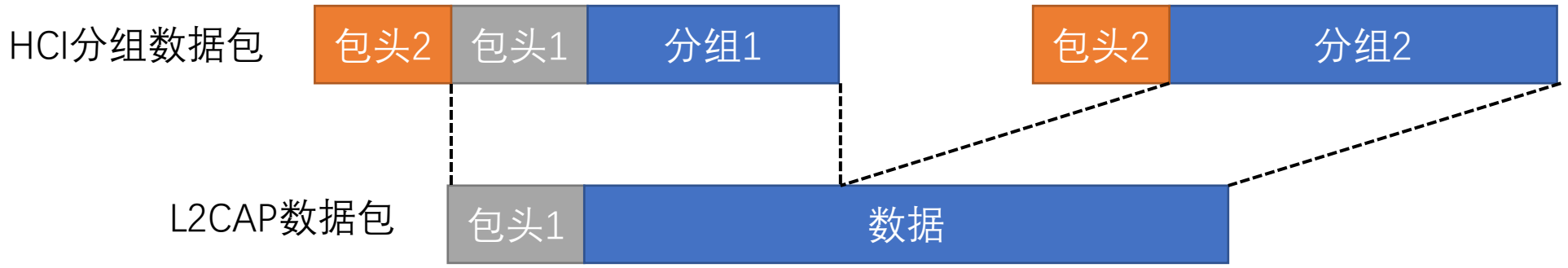
分段是将PDU分解成较小的部分，以便从L2CAP传递到较低层。
重组是根据从下层传递来的片段重组PDU的过程。
分段和重组可以应用于任何L2CAP PDU。



3.分组与重组

- 重组

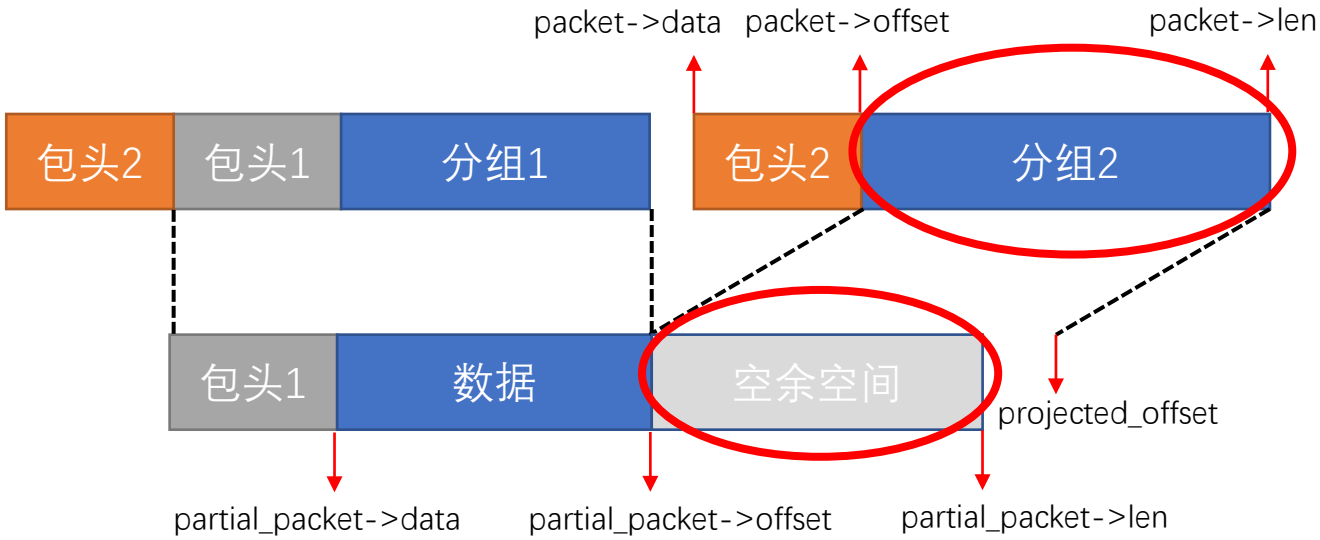
每个数据包的包长都存储在包头中



二、漏洞原理

1.重组函数

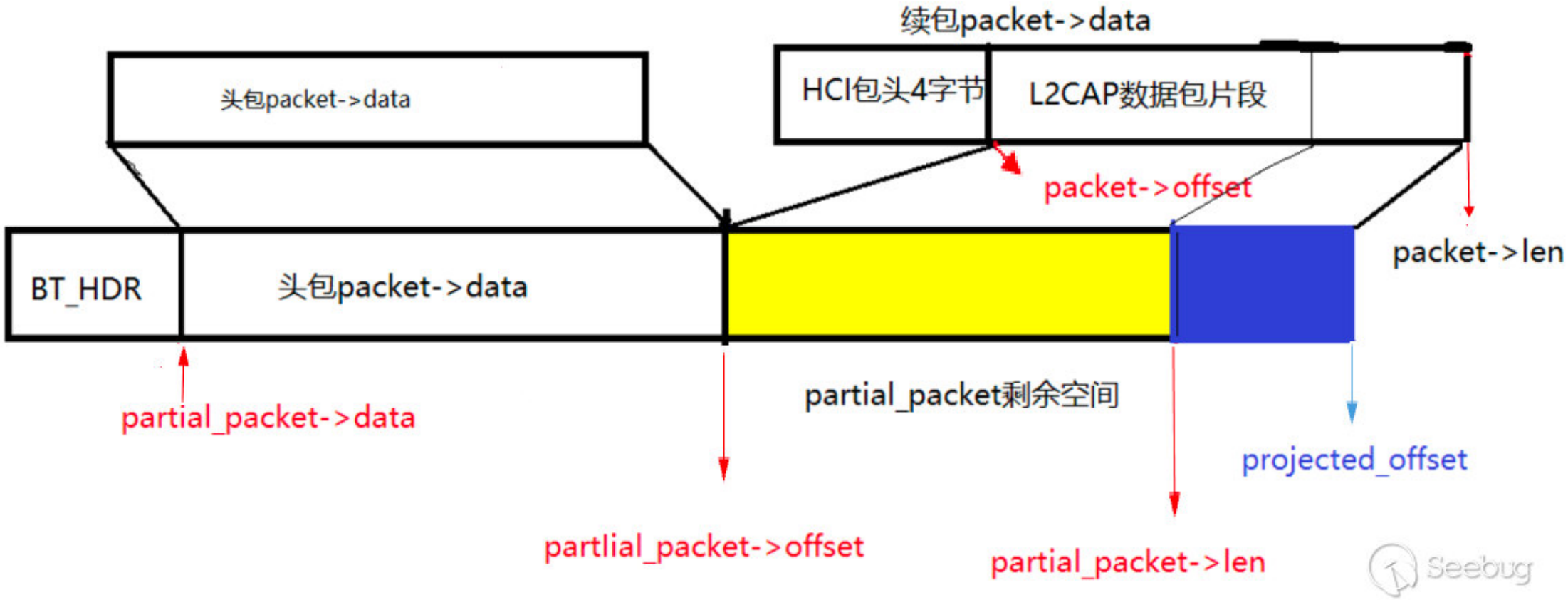
```
packet->offset = HCI_ACL_PREAMBLE_SIZE;
uint16_t projected_offset =
    partial_packet->offset + (packet->len - HCI_ACL_PREAMBLE_SIZE);
if (projected_offset >
    partial_packet->len) {  // len stores the expected length
  LOG_WARN(LOG_TAG,
           "%s got packet which would exceed expected length of %d. "
           "Truncating.",
           __func__, partial_packet->len);
  packet->len = partial_packet->len - partial_packet->offset;
  projected_offset = partial_packet->len;
}

memcpy(partial_packet->data + partial_packet->offset,
       packet->data + packet->offset, packet->len - packet->offset);
```



- > 0 数据丢失
- < 0 缓冲区溢出

只需构造使得Line 221处的packet->len小于packet->offset，即小于4。例如，令其为2，则memcpy的第三个参数为-2，强制转化成无符号数为0xfffffffffffffffe，使得产生overflow

数据包重组


续包packet->data
头包packet->data
HCI包头4字节
L2CAP数据包片段
packet->offset
packet->len
BT_HDR
头包packet->data
partial_packet->data
partial_packet剩余空间
projected_offset
partlial_packet->offset
partial_packet->len
Seebug

三、漏洞利用

1.漏洞利用思想

- 构造含有恶意代码的数据包
- 设定包头中的包长字段
- 利用缓冲区溢出，将函数返回地址替换为恶意代码的地址

CVE-2020-0022漏洞位于HCI层，漏洞补丁代码位于hci/src/packet_fragmenter.cc（以8.1.0_r33为例）中的reassemble_and_dispatch()函数中，该函数是用于数据包分片的重组。对于过长的ACL数据包需要进行包的重组，主要是根据ACL包中的PB Flag标志位进行重组，如果当前是起始部分并且是不完整的，则生成一个部分包（partial_packet）放到map里，等下次收到它的后续部分进行拼装，拼装完毕后就分发出去。详细分析reassemble_and_dispatch()函数如下：

```
120 static void reassemble_and_dispatch(UNUSED_ATTR BT_HDR* packet) {
121   if ((packet->event & MSG_EVT_MASK) == MSG_HC_TO_STACK_HCI_ACL) {
122     uint8_t* stream = packet->data;
123     uint16_t handle;
124     uint16_t l2cap_length;
125     uint16_t acl_length;
126
127     STREAM_TO_UINT16(handle, stream);
128     STREAM_TO_UINT16(acl_length, stream);
129     STREAM_TO_UINT16(l2cap_length, stream);
130
131     CHECK(acl_length == packet->len - HCI_ACL_PREAMBLE_SIZE);
132
133     uint8_t boundary_flag = GET_BOUNDARY_FLAG(handle);
134     handle = handle & HANDLE_MASK;
```

```
136     if (boundary_flag == START_PACKET_BOUNDARY) {
137       auto map_iter = partial_packets.find(handle);
138       if (map_iter != partial_packets.end()) {
139         LOG_WARN(LOG_TAG,
140                  "%s found unfinished packet for handle with start packet. "
141                  "Dropping old.",
142                  __func__);
143
144         BT_HDR* hdl = map_iter->second;
145         partial_packets.erase(map_iter);
146         buffer_allocator->free(hdl);
147       }
148
149       if (acl_length < L2CAP_HEADER_SIZE) {
150         LOG_WARN(LOG_TAG, "%s L2CAP packet too small (%d < %d). Dropping it.",
151                  __func__, packet->len, L2CAP_HEADER_SIZE);
152         buffer_allocator->free(packet);
153         return;
154       }
```

```
156       uint16_t full_length =
157           l2cap_length + L2CAP_HEADER_SIZE + HCI_ACL_PREAMBLE_SIZE;
158
159       // Check for buffer overflow and that the full packet size + BT_HDR size
160       // is less than the max buffer size
161       if (check_uint16_overflow(l2cap_length,
162                                 (L2CAP_HEADER_SIZE + HCI_ACL_PREAMBLE_SIZE)) ||
163           ((full_length + sizeof(BT_HDR)) > BT_DEFAULT_BUFFER_SIZE)) {
164         LOG_ERROR(LOG_TAG, "%s Dropping L2CAP packet with invalid length (%d).",
165                   __func__, l2cap_length);
166         buffer_allocator->free(packet);
167         return;
168       }
169
170       if (full_length <= packet->len) {
171         if (full_length < packet->len)
172           LOG_WARN(LOG_TAG,
173                    "%s found l2cap full length %d less than the hci length %d.",
174                    __func__, l2cap_length, packet->len);
175
176         callbacks->reassembled(packet);
177         return;
178       }
```

开始搞事情

```
180       BT_HDR* partial_packet =
181           (BT_HDR*)buffer_allocator->alloc(full_length + sizeof(BT_HDR));
182       partial_packet->event = packet->event;
183       partial_packet->len = full_length;
184       partial_packet->offset = packet->len;
185
186       memcpy(partial_packet->data, packet->data, packet->len);
187
188       // Update the ACL data size to indicate the full expected length
189       stream = partial_packet->data;
190       STREAM_SKIP_UINT16(stream);  // skip the handle
191       UINT16_TO_STREAM(stream, full_length - HCI_ACL_PREAMBLE_SIZE);
192
193       partial_packets[handle] = partial_packet;
194
195       // Free the old packet buffer, since we don't need it anymore
196       buffer_allocator->free(packet);
197     } else {
```

计算数据包分片长度和

```
197     } else {
198       auto map_iter = partial_packets.find(handle);
199       if (map_iter == partial_packets.end()) {
200         LOG_WARN(LOG_TAG,
201                  "%s got continuation for unknown packet. Dropping it.",
202                  __func__);
203         buffer_allocator->free(packet);
204         return;
205       }
206       BT_HDR* partial_packet = map_iter->second;
207
208       packet->offset = HCI_ACL_PREAMBLE_SIZE;
209       uint16_t projected_offset =
210           partial_packet->offset + (packet->len - HCI_ACL_PREAMBLE_SIZE);
```

关键代码 （hole here）
应该是瞌睡时写的（逃

```
211         if (projected_offset >
212             partial_packet->len) {  // len stores the expected length
213           LOG_WARN(LOG_TAG,
214                    "%s got packet which would exceed expected length of %d. "
215                    "Truncating.",
216                    __func__, partial_packet->len);
217           packet->len = partial_packet->len - partial_packet->offset;
218           projected_offset = partial_packet->len;
219         }
220
221         memcpy(partial_packet->data + partial_packet->offset,
222                packet->data + packet->offset, packet->len - packet->offset);
```

2.漏洞利用展示

- 扫描蓝牙设备

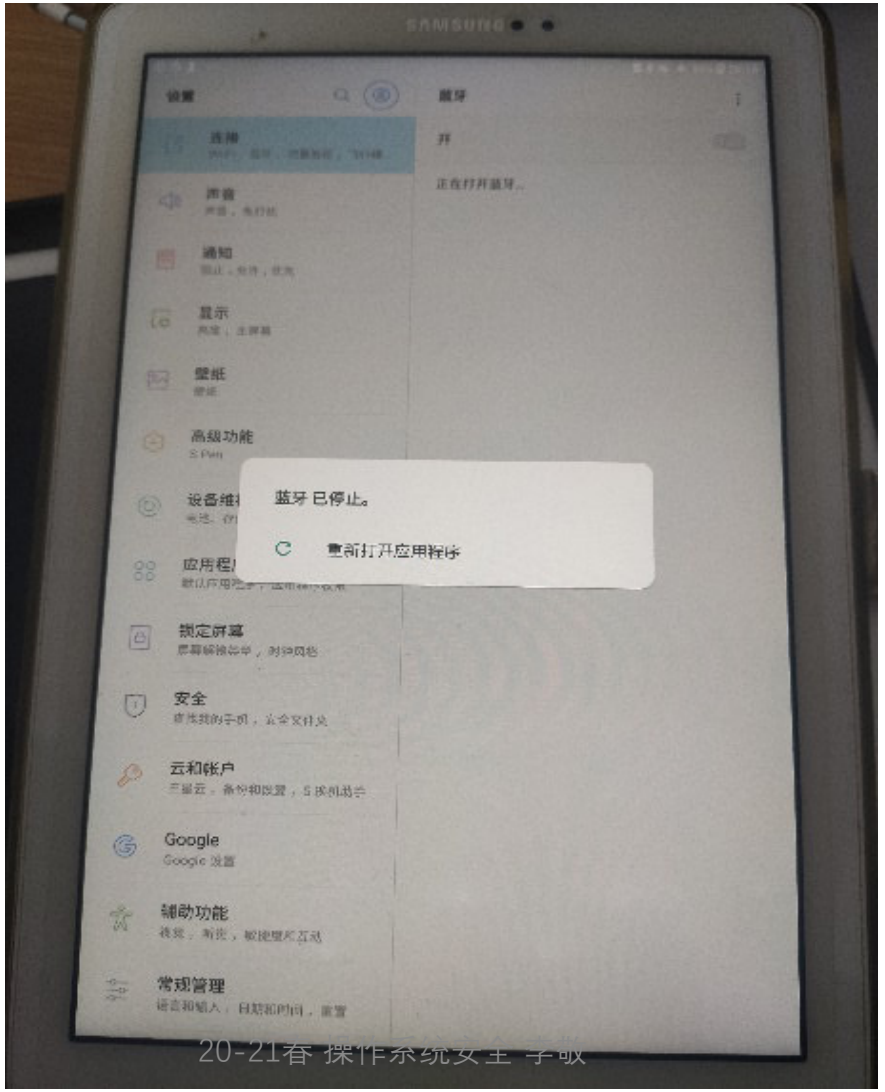
```
yc@ubuntu:~/cve-2020-0022$ hcitool scan
Scanning ...
        E4:46:DA:69:52:CB       米6
        84:98:66:DA:E8:AE       Galaxy Tab A (2016) with S Pen
        20:F4:78:01:51:EA       Redmi K20 Pro Premium Edition
```

- 发送恶意数据包

```
yc@ubuntu:~/cve-2020-0022$ sudo ./poc 84:98:66:DA:E8:AE

Creating HCI socket...

Creating l2cap socket...
crash 256
Creating an HCI BLE connection...

Prepare to send packet

Sent first packet

Closing HCI socket...

Closing l2cap socket...
```

2.漏洞利用展示

- 设备蓝牙被断开



四、漏洞修复

- 尽快更新最新Android安全补丁
- 仅在绝对必要时启用蓝牙
- 保持蓝牙设备不可发现

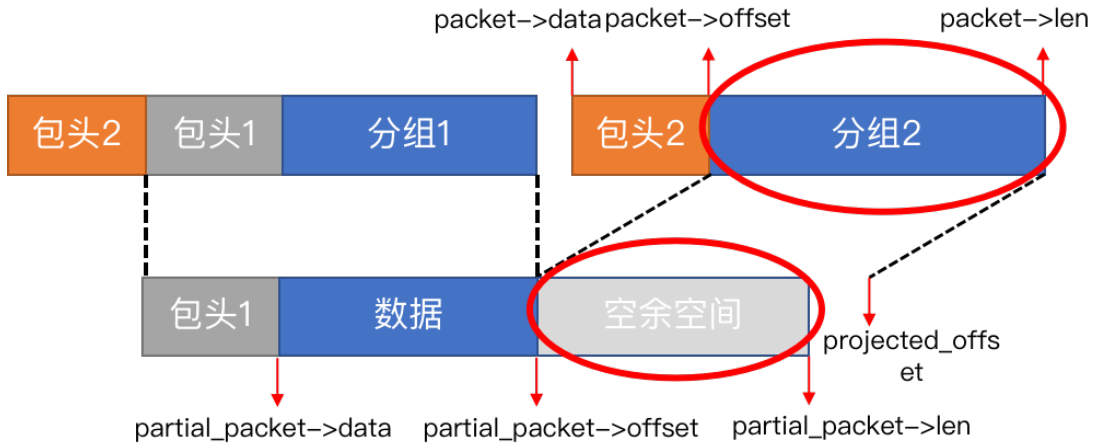
1.漏洞修复

```
@@ -221,7 +221,8 @@
                    "%s got packet which would exceed expected length of %d. "
                    "Truncating.",
                    __func__, partial_packet->len);
-         packet->len = partial_packet->len - partial_packet->offset;
+         packet->len =
+             (partial_packet->len - partial_packet->offset) + packet->offset;
          projected_offset = partial_packet->len;
        }
```

补丁代码中将packet->len加上了一个packet->offset，用于后面抵消减packet->offset的操作。

Reference

- https://insinuator.net/2020/02/critical-bluetooth-vulnerability-in-android-cve-2020-0022/
- https://akhozo.blogs.com/2020/02/critical-android-bluetooth-flaw-cve.html?spref=tw
- https://android.googlesource.com/platform/system/bt/+/3cb7149d8fed2d7d77ceaa95bf845224c4db3baf%5E%21/#F0](#F0)
- https://source.android.com/security/bulletin/2020-02-01.html
- http://androidxref.com/8.1.0_r33/xref/system/bt/hci/src/packet_fragmenter.cc
- Bluetooth_Core_v4.2蓝牙官方文档



编程打瞌睡
社区两行泪
拒绝**overflow**

THANKS

感谢批评指正

@lix3on

University of Chinese Academy of Sciences