
看雪·第六届安全开发者峰会
漫谈AOSP蓝牙漏洞挖掘技术
韩子诺 OPPO安珀实验室
2022 SDC

关于我

- 韩子诺（ele7enxxh），OPPO安珀实验室高级安全专家，主要研究领域为Android安全，IoT安全和车联网安全。在二进制漏洞挖掘与利用方面拥有6年以上相关经验；
- 曾在Ruxcon，Zer0Con和Pacsec等多个国际知名会议进行技术演讲；
- 截至目前，累计独立获得Android数百个CVE编号，Google Bug Hunters排行榜第8，Hackerone高通致谢榜2022年度第1；
- 联系我：ele7enxxh（weibo | weixin | github）；

议程

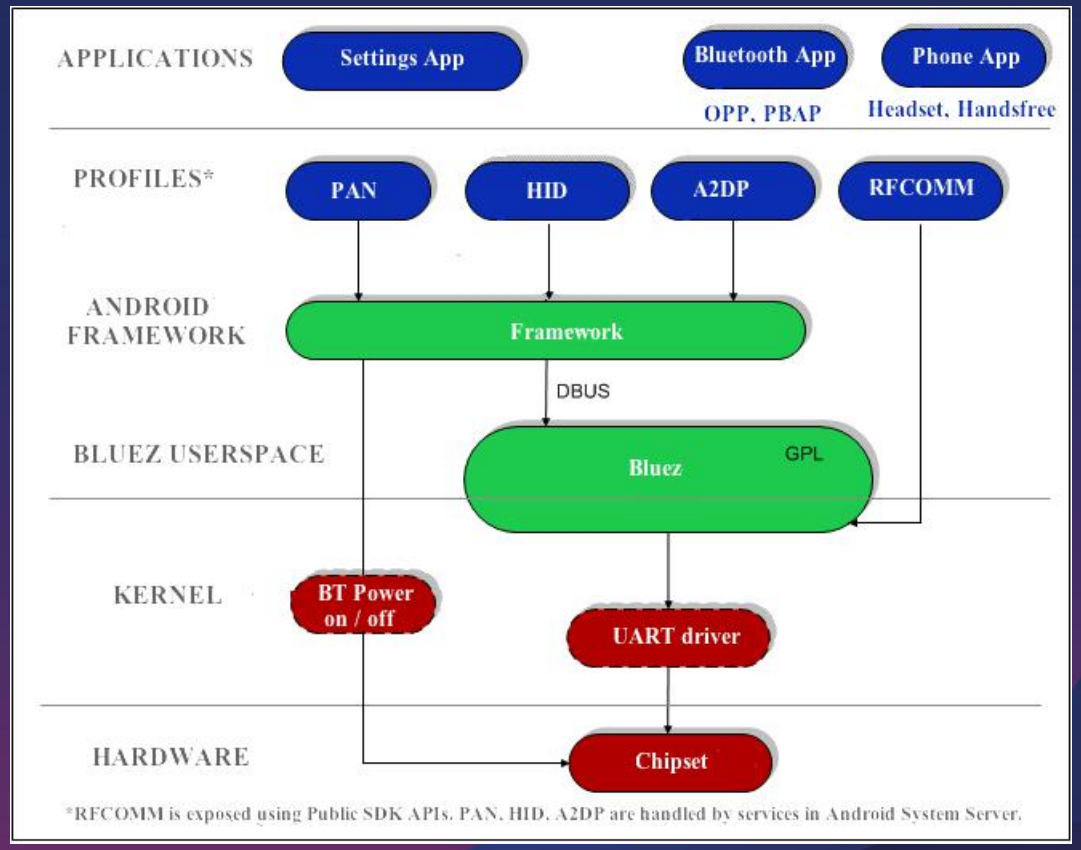
- 背景
  - Android蓝牙协议栈发展历史
  - 历史漏洞分析
- 攻击面
- 挖掘方法
  - 源码审计和模糊测试
- 展望
  - 潜在的脆弱点
  - 更高效的挖掘思路

背景

Android蓝牙默认协议栈

2.2 - BlueZ

- Linux默认协议栈

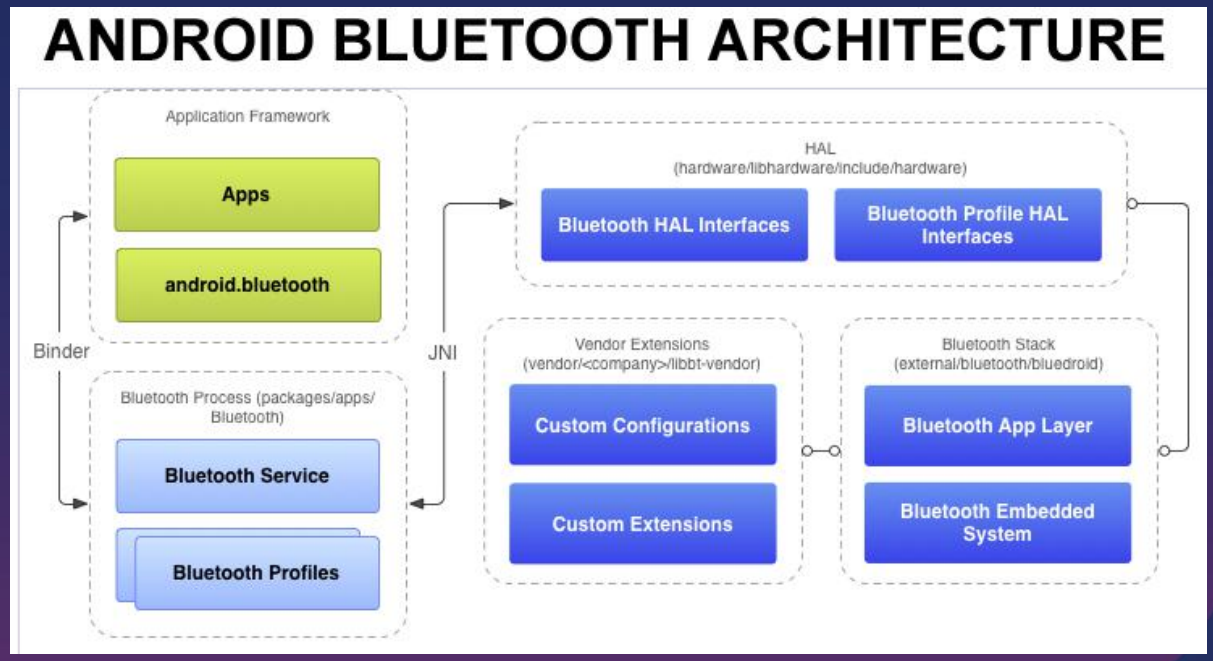


Android蓝牙默认协议栈

4.2 - Bluedroid

2.2 - BlueZ

- 由博通和Google共同开发
- external/bluetooth/bluedroid

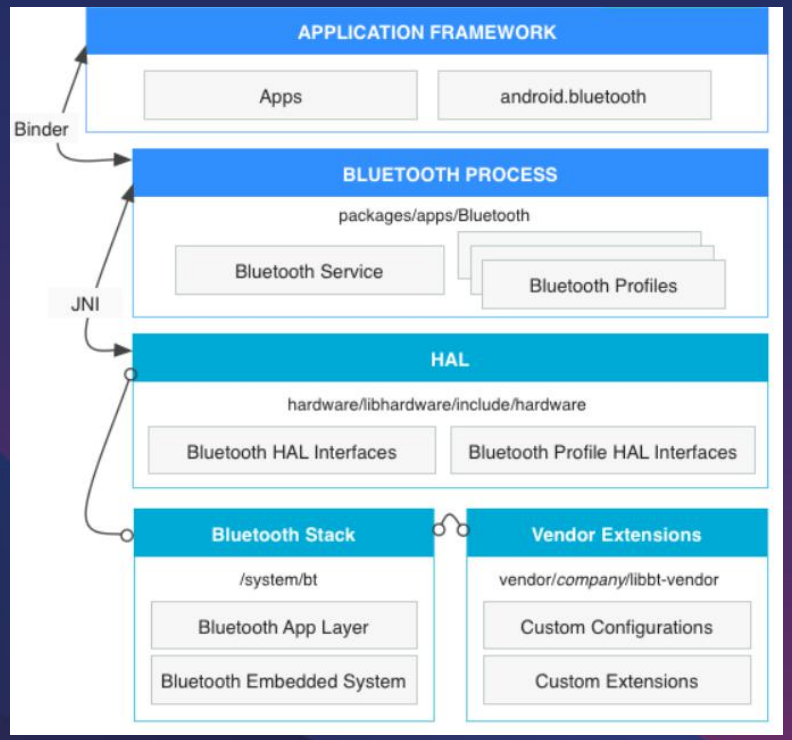


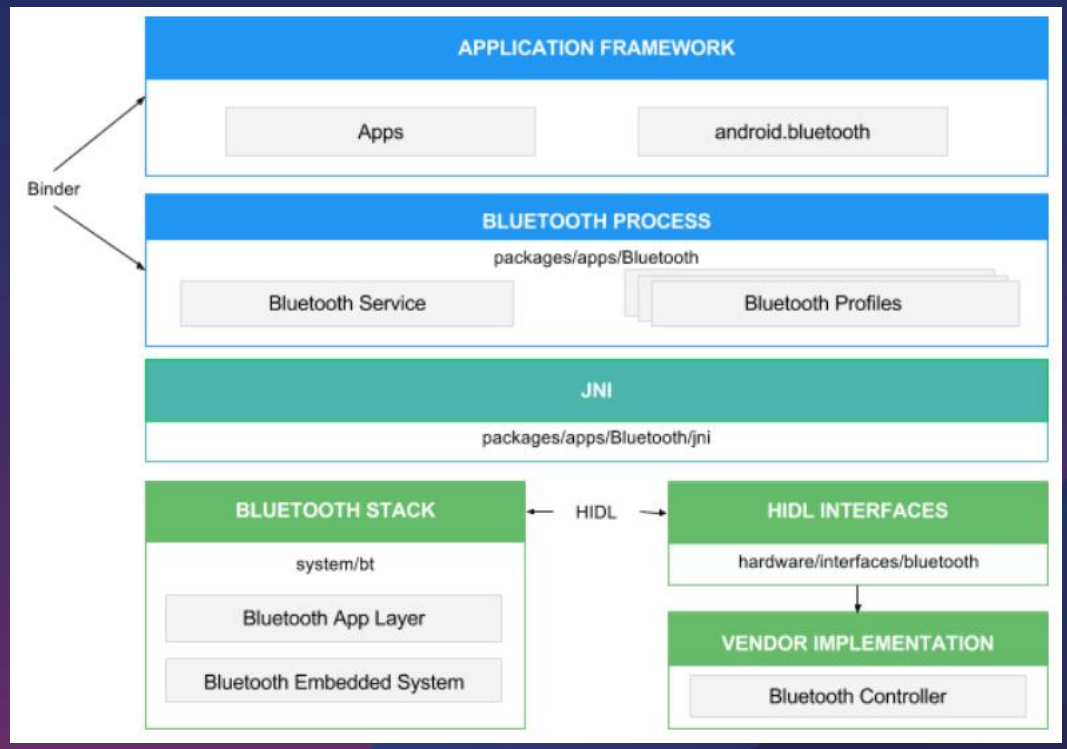
Android蓝牙默认协议栈

4.2 - Bluedroid

2.2 - BlueZ

6.0 - Fluoride





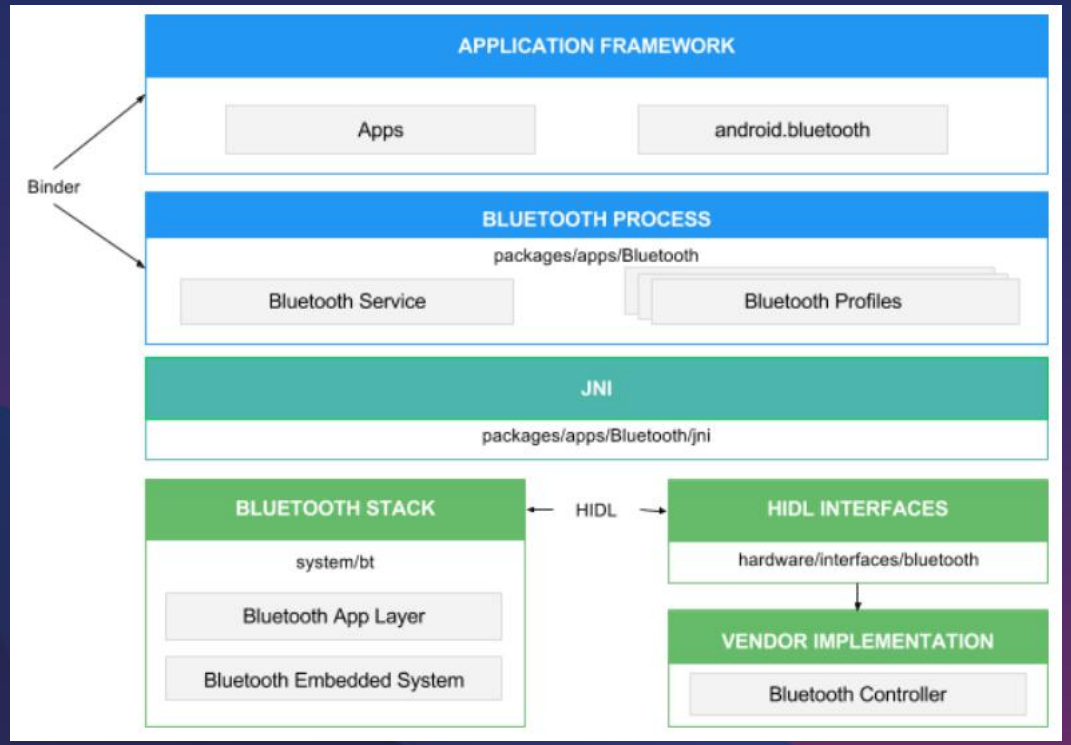
Android蓝牙默认协议栈

4.2 - Bluedroid

13 - Gabeldorsche

2.2 - BlueZ

6.0 - Fluoride

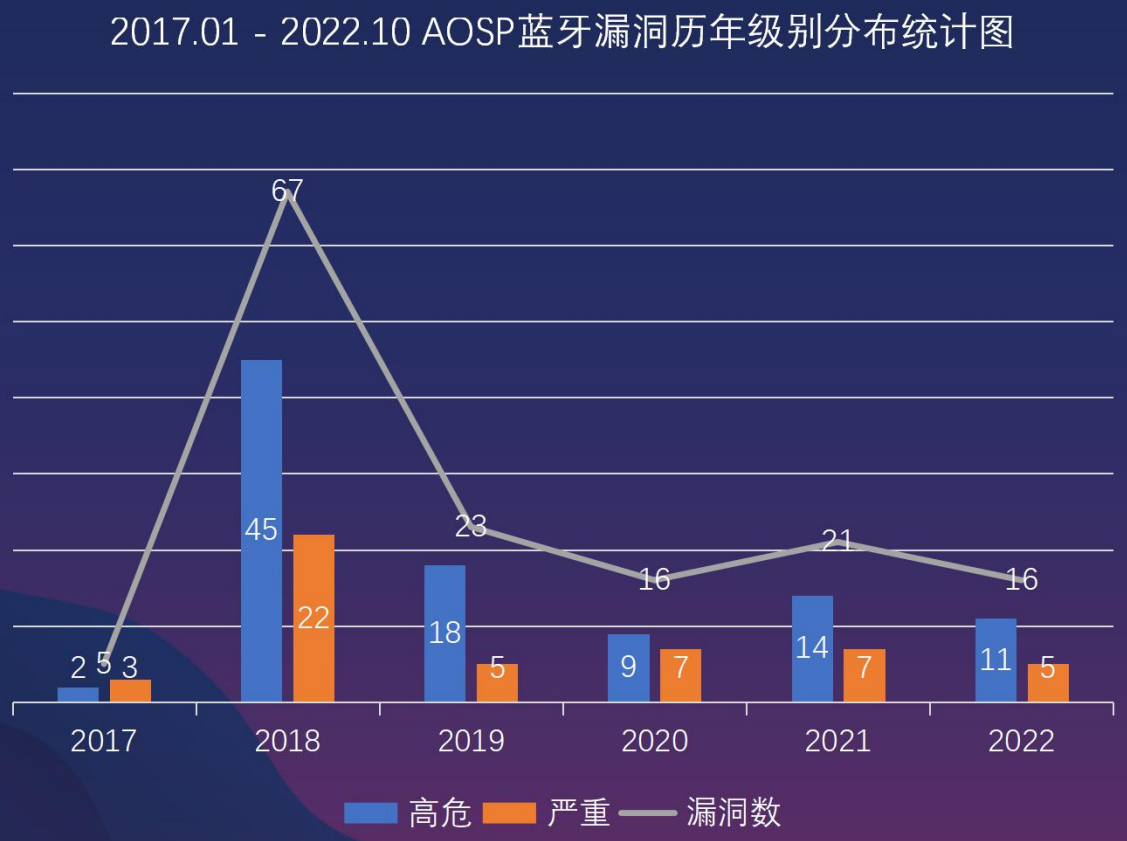


- /system/bt - /packages/modules/Bluetooth/system
- 新的AVRCP协议实现
- 重写了BLE扫描，BLE广播，ACL连接管理，控制器信息管理，HCI层，HAL接口层等模块
- 部分模块开始使用rust语言

历史漏洞

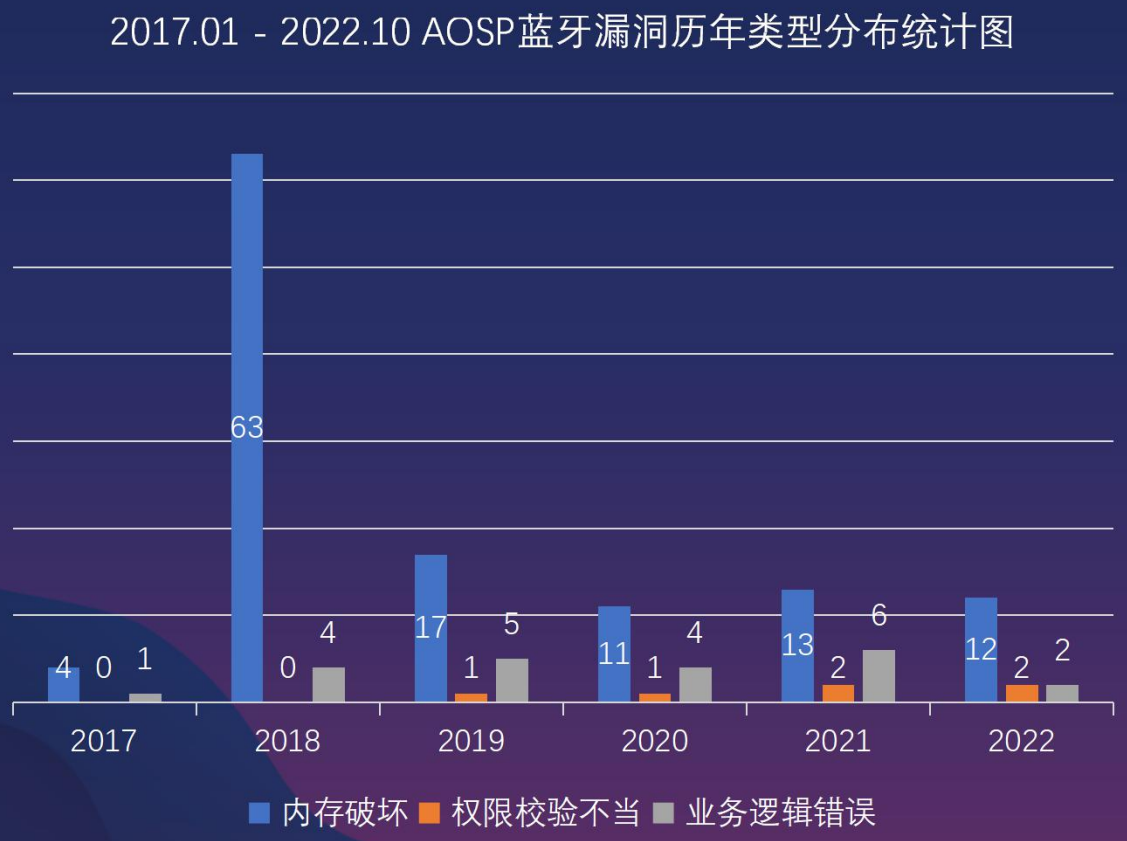
- BlueBorne - 2017年公开的一组蓝牙HOST层协议内存破坏漏洞，影响多个平台和系统，造成了很广的影响，引导了众多安全研究者关注蓝牙安全
- BadBluetooth - 2019年由香港中文大学Fenghao Xu发表于2019年安全顶会NDSS上的一篇论文，主要介绍了蓝牙配对时的逻辑缺陷导致的绕过风险
- BlueFrag - 2020年Android安全公告披露的一个严重漏洞，攻击者利用ACL分包处理时的一个越界写漏洞可以远程代码执行

历史漏洞分析



- 从2017年1月到2022年10月，累计至少披露148个漏洞，其中高危99个，严重49个（未统计中危）
- 2018年发现漏洞67个，几乎占比近6年总数的一半（主要是受到BlueBorne的影响）

历史漏洞分析

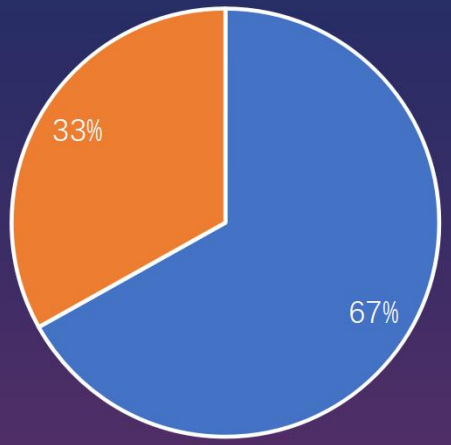


- 漏洞的类型具有较强的时间相关性：业务逻辑错误漏洞 -> 内存破坏漏洞 -> 权限校验不当漏洞 -> 内存破坏漏洞
- 漏洞的类型随着代码的健壮性和白帽子的关注点而变化

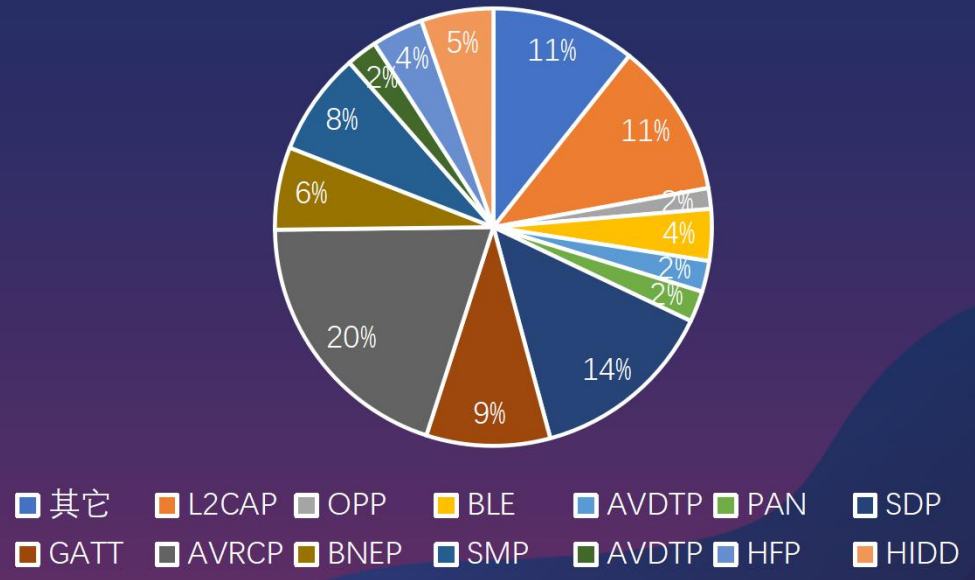
历史漏洞分析

- 由于蓝牙存在天然的远程攻击面，因此漏洞级别定级偏高
- 白帽子在AVRCP，SDP，L2CAP，GATT等模块中发现了大量的漏洞

2017.01 - 2022.10 AOSP蓝牙漏洞级别饼图



2017.01 - 2022.10 AOSP蓝牙漏洞模块饼图



攻击面


APP
Normal APP
Binder IPC
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
BluetoothManagerService
权限校验不当，业务逻辑错误（CVE-2020-27054，CVE-2020-0183等）
Bind Service
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
Adapter
battery
HearingAid
Gatt
Sap
Pbap
A2dp
HidDevice
HidHost
LeAudio
Map
Headset
Opp
Pan
AvrcpTarget
A2dpSink
BassClient
Mcp
HeadsetClient
Tbs
AvrcpController
MapClient
VolumeControl
Csip
Hap
权限校验不当，业务逻辑错误（CVE-2021-0328，CVE-2021-0588，CVE-2021-0604，CVE-2022-20126等）
JNI
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
adapter
a2dp
avrcp
csip
gatt
hap
hfp
hid
sdp
vc
le_audio
pan
内存破坏（CVE-2018-9414，CVE-2022-XXXX等）
Interface && Callback
libbluetooth - packages/modules/Bluetooth/system
JNI/Main Threads
BTIF
BTA
BTM
BTE
PAN
SRVC
GAP
A2DP
LEAUDIO
HFP/HFPC
SDP
BNEP
ATT/GATT/EATT
AVDT/AVRC/AVCT
RFCOMM
HIDD/HIDH
SMP
L2CAP - (BR/LE)
ACL
HCI
业务逻辑错误，C++特性使用不当，线程不安全，内存破坏（CVE-2021-0522，CVE-2021-39708，CVE-2022-20345等）
cmd && event && packet
Host
Controler
Bluetooth Chip
Radio
Remote
Data

挖掘方法

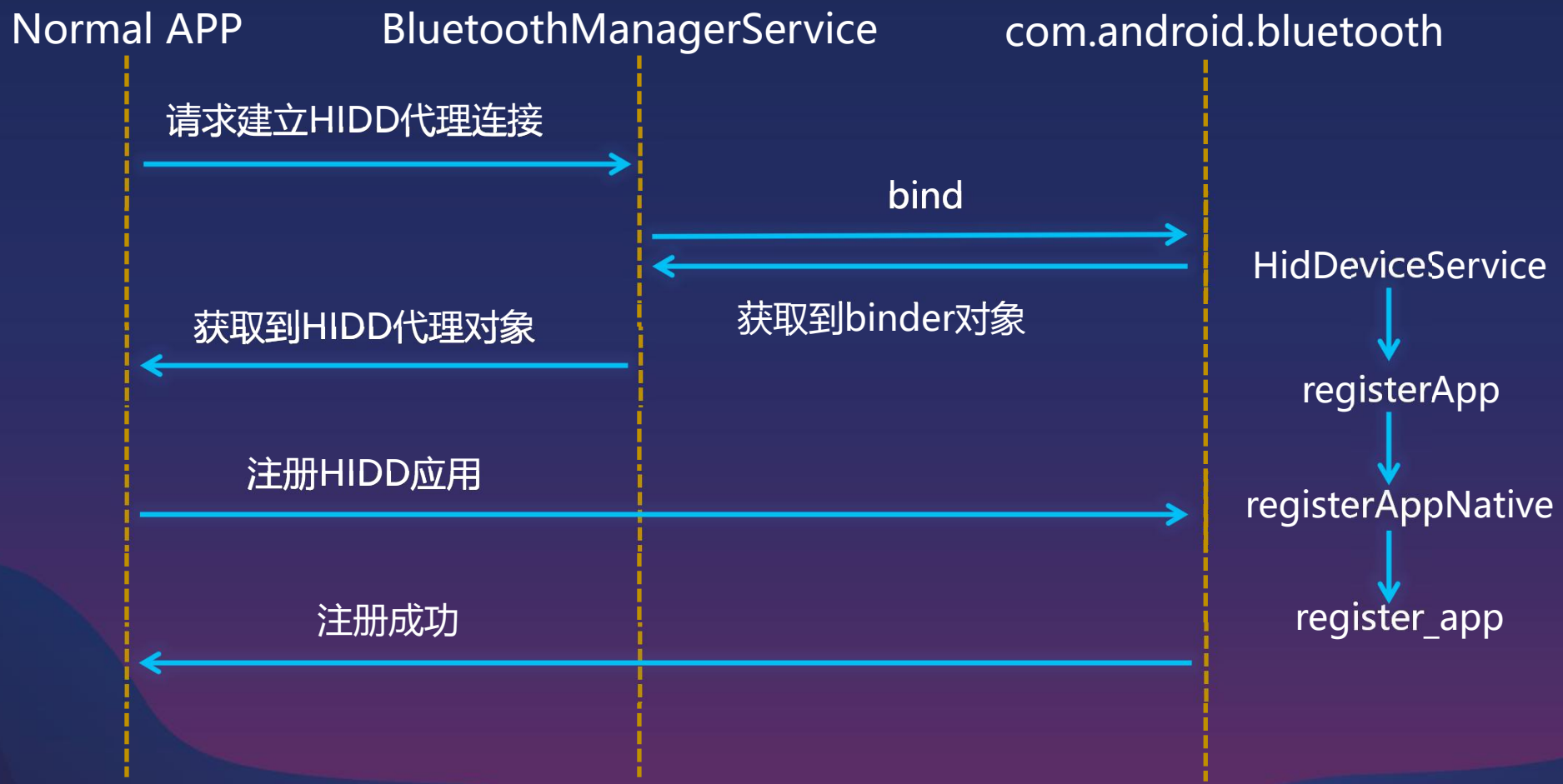
源码审计

- 优势
  - 无需运行：传统的蓝牙远程Fuzz效率较低，且复杂度高（需要考虑不同协议栈的状态机）
  - 不依赖硬件：某些蓝牙漏洞需要特定的硬件才能触发
  - 方便且高效：https://cs.android.com/
- 劣势
  - 需要比较了解蓝牙协议和实现架构
  - 费眼，费脑

源码审计 - HID简介

- HID(Human Input Device)：人体输入设备配置文件，定义了蓝牙键盘和鼠标的功能
- Android在9.0之后开放了HID相关的API，APP主要通过BluetoothHidDevice类使用相关能力
- HID设备主要分为Host和Device，Android目前只默认启用HID Device
- 在framework，libbluetooth_jni，libbluetooth等多个模块中均有大量的HID实现代码

源码审计 - HIDD工作流程之应用注册



源码审计 - CVE-2018-9544/9545

本地应用通过Binder IPC可以调用registerApp接口，且参数全部可控

sdp参数包含多个不定长的数据，如果传入一个很长的值呢?

```
1 synchronized boolean registerApp(BluetoothHidDeviceAppSdpSettings sdp,
2             BluetoothHidDeviceAppQosSettings inQos,
  BluetoothHidDeviceAppQosSettings outQos,
3             IBluetoothHidDeviceCallback callback) {
4     return mHidDeviceNativeInterface.registerApp(sdp.getName(),
  sdp.getDescription(),
5         sdp.getProvider(), sdp.getSubclass(), sdp.getDescriptors(), inQos,
  outQos);
6 }
```

```
1 public final class BluetoothHidDeviceAppSdpSettings implements Parcelable
  {
2     private final String mName;
3     private final String mDescription;
4     private final String mProvider;
5     private final byte mSubclass;
6     private final byte[] mDescriptors;
7 }
```

源码审计 - CVE-2018-9544/9545

通过JNI接口调用registerAppNative

通过hidd的BTIF接口调用register_app

```java
public boolean registerApp(String name, String description, String provider,
        byte subclass, byte[] descriptors, int[] inQos, int[] outQos) {
    return registerAppNative(name, description, provider, subclass, descriptors, inQos, outQos);
}
```

```
static jboolean registerAppNative(JNIEnv* env, jobject thiz, jstring name,
    jstring description, jstring provider, jbyte subclass, jbyteArray descriptors,
    jintArray p_in_qos, jintArray p_out_qos) {
    bthd_app_param_t app_param;
    bthd_qos_param_t in_qos;
    bthd_qos_param_t out_qos;

    jsize size = env->GetArrayLength(descriptors);
    uint8_t* data = (uint8_t*)malloc(size);

    env->GetByteArrayRegion(descriptors, 0, size, (jbyte*)data);
    app_param.name = env->GetStringUTFChars(name, NULL);
    app_param.description = env->GetStringUTFChars(description, NULL);
    app_param.provider = env->GetStringUTFChars(provider, NULL);
    app_param.subclass = subclass;
    app_param.desc_list = data;
    app_param.desc_list_len = size;

    bt_status_t ret = sHiddIf->register_app(&app_param, &in_qos, &out_qos);
}
```

源码审计 - CVE-2018-9544/9545

bthdInterface的register_app接口

memcpy越界读

```
static const bthd_interface_t bthdInterface = {
    sizeof(bthdInterface),
    init,
    cleanup,
    register_app,
    unregister_app,
    connect,
    disconnect,
    send_report,
    report_error,
    virtual_cable_unplug,
};
```

```
static bt_status_t register_app(bthd_app_param_t* p_app_param,
    bthd_qos_param_t* p_in_qos, bthd_qos_param_t* p_out_qos) {
  app_info.p_name = (char*)osi_malloc(BTIF_HD_APP_NAME_LEN);
  memcpy(app_info.p_name, p_app_param->name, BTIF_HD_APP_NAME_LEN);
  app_info.p_description = (char*)osi_malloc(BTIF_HD_APP_DESCRIPTION_LEN);
  memcpy(app_info.p_description, p_app_param->description,
BTIF_HD_APP_DESCRIPTION_LEN);
  app_info.p_provider = (char*)osi_malloc(BTIF_HD_APP_PROVIDER_LEN);
  memcpy(app_info.p_provider, p_app_param->provider,
BTIF_HD_APP_PROVIDER_LEN);
  app_info.subclass = p_app_param->subclass;
  app_info.descriptor.dl_len = p_app_param->desc_list_len;
  app_info.descriptor.dsc_list =
(uint8_t*)osi_malloc(app_info.descriptor.dl_len);
  memcpy(app_info.descriptor.dsc_list, p_app_param->desc_list,
p_app_param->desc_list_len);
}
```

源码审计 - CVE-2018-9544/9545

description为一个固定大小的数组

BTIF层通过多个函数调用到BTA层，导致memcpy越界写

```
typedef struct {
    BT_HDR_RIGID hdr;
    char name[BTA_HD_APP_NAME_LEN];
    char description[BTA_HD_APP_DESCRIPTION_LEN];
    char provider[BTA_HD_APP_PROVIDER_LEN];
    uint8_t subclass;
    uint16_t d_len;
    uint8_t d_data[BTA_HD_APP_DESCRIPTOR_LEN];

    tBTA_HD_QOS_INFO in_qos;
    tBTA_HD_QOS_INFO out_qos;
} tBTA_HD_REGISTER_APP;
```

```
extern void BTA_HdRegisterApp(tBTA_HD_APP_INFO* p_app_info,
tBTA_HD_QOS_INFO* p_in_qos,
    tBTA_HD_QOS_INFO* p_out_qos) {
    tBTA_HD_REGISTER_APP* p_buf =
(tBTA_HD_REGISTER_APP*)osi_malloc(sizeof(tBTA_HD_REGISTER_APP));
    p_buf->d_len = p_app_info->descriptor.dl_len;
    memcpy(p_buf->d_data, p_app_info->descriptor.dsc_list, p_app_info-
>descriptor.dl_len);
}
```

模糊测试

- 优势
  - 无需深入了解蓝牙协议
  - 结合ASAN和MSAN可以更容易发现不明显的漏洞
  - 节省人力
- 劣势
  - 没有现成的适合模糊测试的暴露接口
  - 部分配置文件/协议栈包含复杂的状态机

模糊测试

- 寻找入口
  - L2CA_Register()，L2CA_Register2()，L2CA_RegisterLECoc()

```
1 uint16_t L2CA_Register(uint16_t psm, const tL2CAP_APPL_INFO& p_cb_info,
2     bool enable_snoop, tL2CAP_ERTM_INFO* p_ertm_info, uint16_t my_mtu,
  uint16_t required_remote_mtu,
3     uint16_t sec_level);
```

模糊测试

- 寻找入口
  - L2CA_Register()，L2CA_Register2()，L2CA_RegisterLECoc()

```
typedef struct {
    tL2CA_CONNECT_IND_CB* pL2CA_ConnectInd_Cb;
    tL2CA_CONNECT_CFM_CB* pL2CA_ConnectCfm_Cb;
    tL2CA_CONFIG_IND_CB* pL2CA_ConfigInd_Cb;
    tL2CA_CONFIG_CFM_CB* pL2CA_ConfigCfm_Cb;
    tL2CA_DISCONNECT_IND_CB* pL2CA_DisconnectInd_Cb;
    tL2CA_DISCONNECT_CFM_CB* pL2CA_DisconnectCfm_Cb;
    tL2CA_DATA_IND_CB* pL2CA_DataInd_Cb;
    tL2CA_CONGESTION_STATUS_CB* pL2CA_CongestionStatus_Cb;
    tL2CA_TX_COMPLETE_CB* pL2CA_TxComplete_Cb;
    tL2CA_ERROR_CB* pL2CA_Error_Cb;
    tL2CA_CREDIT_BASED_CONNECT_IND_CB* pL2CA_CreditBasedConnectInd_Cb;
    tL2CA_CREDIT_BASED_CONNECT_CFM_CB* pL2CA_CreditBasedConnectCfm_Cb;
    tL2CA_CREDIT_BASED_RECONFIG_COMPLETED_CB*
        pL2CA_CreditBasedReconfigCompleted_Cb;
    tL2CA_CREDIT_BASED_COLLISION_IND_CB*
pL2CA_CreditBasedCollisionInd_Cb;
} tL2CAP_APPL_INFO;
```

模糊测试

- 寻找入口
  - pL2CA_DataInd_Cb
  - avct | avdt | bnep | eatt | gap | gatt | hidd | hidh | rfcomm | sdp

```
packages/modules/Bluetooth/system/stack/avct/avct_l2c_br.cc (1 occurrence)
58: avct_l2c_br_data_ind_cback,
packages/modules/Bluetooth/system/stack/avct/avct_l2c.cc (1 occurrence)
57: avct_l2c_data_ind_cback,
packages/modules/Bluetooth/system/stack/avdt/avdt_l2c.cc (1 occurrence)
59: avdt_l2c_data_ind_cback,
packages/modules/Bluetooth/system/stack/bnep/bnep_main.cc (1 occurrence)
85: bnep_cb.reg_info.pL2CA_DataInd_Cb = bnep_data_ind;
packages/modules/Bluetooth/system/stack/eatt/eatt.cc (1 occurrence)
47: reg_info_.pL2CA_DataInd_Cb = eatt_data_ind;
packages/modules/Bluetooth/system/stack/gap/gap_conn.cc (1 occurrence)
123: conn.reg_info.pL2CA_DataInd_Cb = gap_data_ind;
packages/modules/Bluetooth/system/stack/gatt/gatt_main.cc (1 occurrence)
80: gatt_l2cif_data_ind_cback,
packages/modules/Bluetooth/system/stack/hid/hidd_conn.cc (1 occurrence)
57: hidd_l2cif_data_ind,
packages/modules/Bluetooth/system/stack/hid/hidh_conn.cc (1 occurrence)
76: .pL2CA_DataInd_Cb = hidh_l2cif_data_ind,
packages/modules/Bluetooth/system/stack/include/l2c_api.h (1 occurrence)
324: typedef struct {
packages/modules/Bluetooth/system/stack/rfcomm/rfc_l2cap_if.cc (1 occurrence)
71: p_l2c->pL2CA_DataInd_Cb = RFCOMM_BufDataInd;
packages/modules/Bluetooth/system/stack/sdp/sdp_main.cc (1 occurrence)
91: sdp_cb.reg_info.pL2CA_DataInd_Cb = sdp_data_ind;
```

模糊测试 - SDP协议栈

- 测试用例
  - sdp_data_ind函数是sdp协议数据处理入口函数
  - 服务端响应数据处理 - sdp_disc_server_rsp
  - 客户端请求数据处理 - sdp_server_handle_client_req

```
1 static void sdp_data_ind(uint16_t l2cap_cid, BT_HDR* p_msg) {
2     tCONN_CB* p_ccb;
3
4     /* Find CCB based on CID */
5     p_ccb = sdpu_find_ccb_by_cid(l2cap_cid);
6     if (p_ccb != NULL) {
7     if (p_ccb->con_state == SDP_STATE_CONNECTED) {
8         if (p_ccb->con_flags & SDP_FLAGS_IS_ORIG)
9         sdp_disc_server_rsp(p_ccb, p_msg);
10        else
11        sdp_server_handle_client_req(p_ccb, p_msg);
12    }
13 }
```

模糊测试 - SDP协议栈

- 测试用例
  - p_ccb为当前SDP连接的控制块
  - p_msg为当前需要处理的数据，其中len代表数据长度，data为可控数据

```
1 void sdp_disc_server_rsp(tCONN_CB* p_ccb, BT_HDR* p_msg);
2 void sdp_server_handle_client_req(tCONN_CB* p_ccb, BT_HDR* p_msg);
```

```
1 typedef struct {
2     uint16_t event;
3     uint16_t len;
4     uint16_t offset;
5     uint16_t layer_specific;
6     uint8_t data[];
7 } BT_HDR;
```

需要手动创建p_ccb，并生成sdp db数据，根据上下文设置状态机值和pdu值

模糊测试 - SDP协议栈

- 测试用例

```
1  extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size)
2  {
3      FuzzedDataProvider fdp(data, size);
4
5      sdp_init();
6
7      init_ccb();
8
9      init_sdp_db(fdp);
10
11     size_t count = fdp.ConsumeIntegralInRange<int>(1, 20);
12     while (count-- && sdp_db_vect.size() && fdp.remaining_bytes()) {
13         if (fdp.ConsumeBool()) {
14             sdp_disc_server_rsp_fuzzer(fdp);
15         } else {
16             sdp_server_handle_client_req_fuzzer(fdp);
17         }
18     }
19
20     SDP_DeleteRecord(0);
21     sdp_db_vect.clear();
22
23     sdpu_release_ccb(p_ccb);
24
25     sdp_free();
26
27     return 0;
28 }
```

- 调用sdp_init完成sdp协议栈初始化工作
- 初始化ccb，模拟蓝牙连接状态
- 生成sdp数据库
- 模拟蓝牙连续发包，持续对两个接口进行测试
- 清理现场

模糊测试 - SDP协议栈

- 测试用例

```
static void sdp_disc_server_rsp_fuzzer(FuzzedDataProvider &fdp)
{
    p_ccb->con_flags |= SDP_FLAGS_IS_ORIG;
    p_ccb->p_db = sdp_db_vect.at(fdp.ConsumeIntegralInRange<size_t>(0,
sdp_db_vect.size() - 1)).get();

    uint8_t rsp_pdu;
    switch (fdp.ConsumeIntegralInRange<int>(0, 2)) {
        case 0:
            rsp_pdu = SDP_PDU_SERVICE_SEARCH_RSP;
            p_ccb->disc_state = SDP_DISC_WAIT_HANDLES;
            break;
        case 1:
            rsp_pdu = SDP_PDU_SERVICE_ATTR_RSP;
            p_ccb->disc_state = SDP_DISC_WAIT_ATTR;
            break;
        case 2:
            rsp_pdu = SDP_PDU_SERVICE_SEARCH_ATTR_RSP;
            p_ccb->disc_state = SDP_DISC_WAIT_SEARCH_ATTR;
            break;
    }
    size_t size = fdp.remaining_bytes();
    BT_HDR *p_msg = (BT_HDR *) osi_malloc(sizeof(BT_HDR) + sizeof(uint8_t)
+ size);
    p_msg->offset = 0;
    p_msg->len = sizeof(uint8_t) + size;
    uint8_t *p = (uint8_t *) (p_msg + 1) + p_msg->offset;
    UINT8_TO_BE_STREAM(p, rsp_pdu);
    fdp.ConsumeData(p, size);
    sdp_disc_server_rsp(p_ccb, p_msg);
    osi_free(p_msg);

    return;
}
```

- 从上一步生成的数据库中选择一个
- 生成不同的状态机，并构造数据进行测试

总结

- 源码审计
  - 使用一个好的平台：https://cs.android.com/
  - 从攻击面展开，跟随数据的全生命周期进行代码审计
- 模糊测试
  - 在写测试用例的时候，不能只是简单的调用入口函数，而需要根据协议栈的具体情况，考虑到所有的状态机，并且完成初始化工作，否则无法测试到代码深处 - 模拟真实场景下的蓝牙交互过程
- 从今年的结果上看，两种方法的产出几乎一样：CVE-2021-39774，CVE-2021-39708，CVE-2021-39809，CVE-2022-20140，CVE-2022-20221，CVE-2022-20222，CVE-2022-20224，CVE-2022-20229，CVE-2022-20273，CVE-2022-20283，CVE-2022-20273，CVE-2022-20283，CVE-2022-20362，CVE-2022-20410

展望

展望

- 关于攻击面
  - 常见协议栈上的漏洞会越来越少，需要更多关注新特性（如Android13上的le audio）
  - 传统的数据解析导致的越界读写漏洞几乎已经被修复干净，数据在蓝牙进程的全生命周期的处理也许有更多可能性
  - HAL服务
  - 架构设计导致的安全风险 - 并发线程，内存管理，代码错误
- 关于挖掘方法
  - 攻击面代码审计 + 核心代码模糊测试
  - 相似漏洞挖掘方法 - CodeQL

感谢
Q & A