black hat EUROPE 2022

DECEMBER 7-8, 2022

BRIEFINGS

Deep into Android Bluetooth Bug Hunting: New Attack Surfaces and Weak Code Patterns

Zinuo Han

About me

- Senior security experts at OPPO Amber Security Lab
  - Fuzzing and vulnerability research on Android, IoT and Vehicles
  - Development of security defense tools
- Research findings
  - Hundreds of CVEs, Google Bug Hunter No. 7, Qualcomm HackerOne 2022 No. 1
- Previously talked at
  - Ruxcon, Zer0Con, Pacsec, etc
- Contact me
  - ele7enxxh(twitter | gmail)

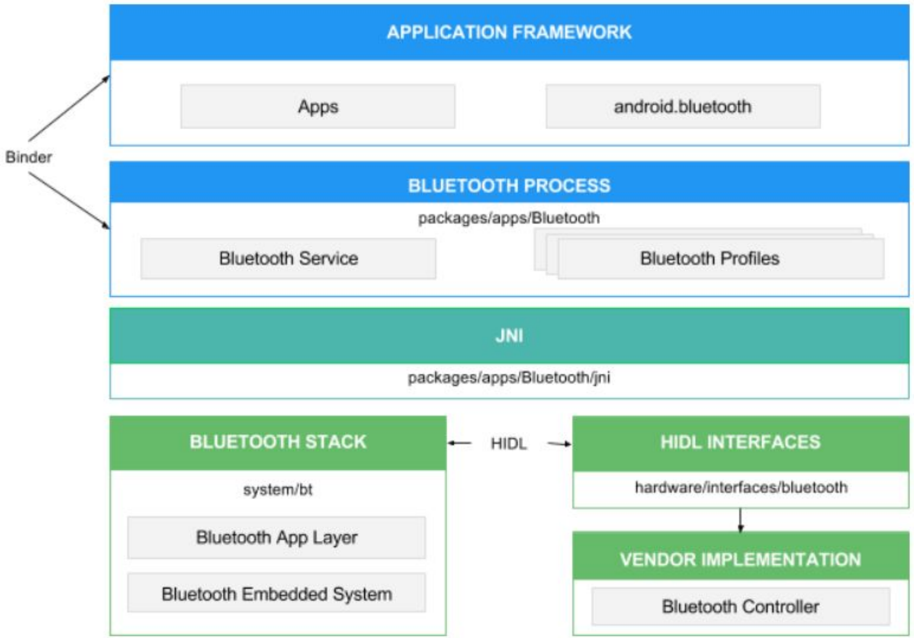
Agenda

- Introduction to Android Bluetooth
- Attack surfaces analysis
- Finding vulnerabilities
- Summary

Android default Bluetooth stack implementation

- Bluez(2.2 - 4.2), Bluedroid(4.2 - 6.0), Fluoride(6.0 - 13), Gabeldorsche(13 -)
  - Architecture changing constantly
- Different scenarios
  - Mobile phone, Android TV, Android Car, etc
- Support multiple protocols and profiles
  - SDP, BNEP, GATT, SMP, AVRCP, AVCTP, HID, RFCOMM, etc
  - Headset, OPP, PBAP, LEAudio, etc

Android Bluetooth architecture



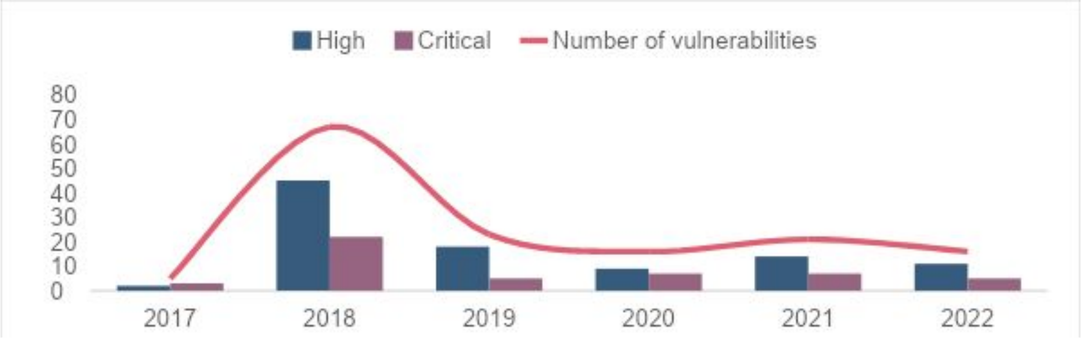
- Application framework
  - User Bluetooth Applications, calls the Bluetooth process through the Binder IPC mechanism
- Bluetooth system service
  - Implements the Bluetooth services and profiles at the Android framework layer, calls into the native Bluetooth stack via JNI
- JNI
  - Implements the Bluetooth services and profiles at the JNI layer, calls into the Bluetooth stack via callbacks
- Bluetooth stack
  - Implements the generic Bluetooth core stack
- Vendor
  - Implements Hardware independent code, interact with the Bluetooth stack using the Hardware Interface Design Language (HIDL)

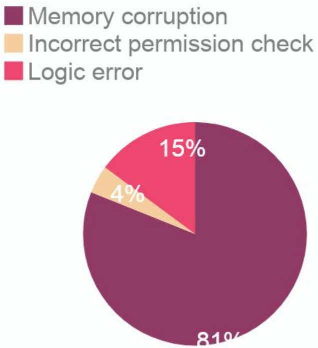
Previous research

- BlueBorne - a series of Bluetooth vulnerabilities endangering multiple operating systems including Android, IOS, Windows and Linux, leading security researchers to pay attention to Bluetooth security
- BadBluetooth - published by Fenghao Xu of the Chinese University of Hong Kong on NDSS, introduced the logic vulnerability in Bluetooth pairing
- BlueFrag - an attacker can use an out of bounds write vulnerability in ACL packet processing to remotely execute code

Historical vulnerability analysis

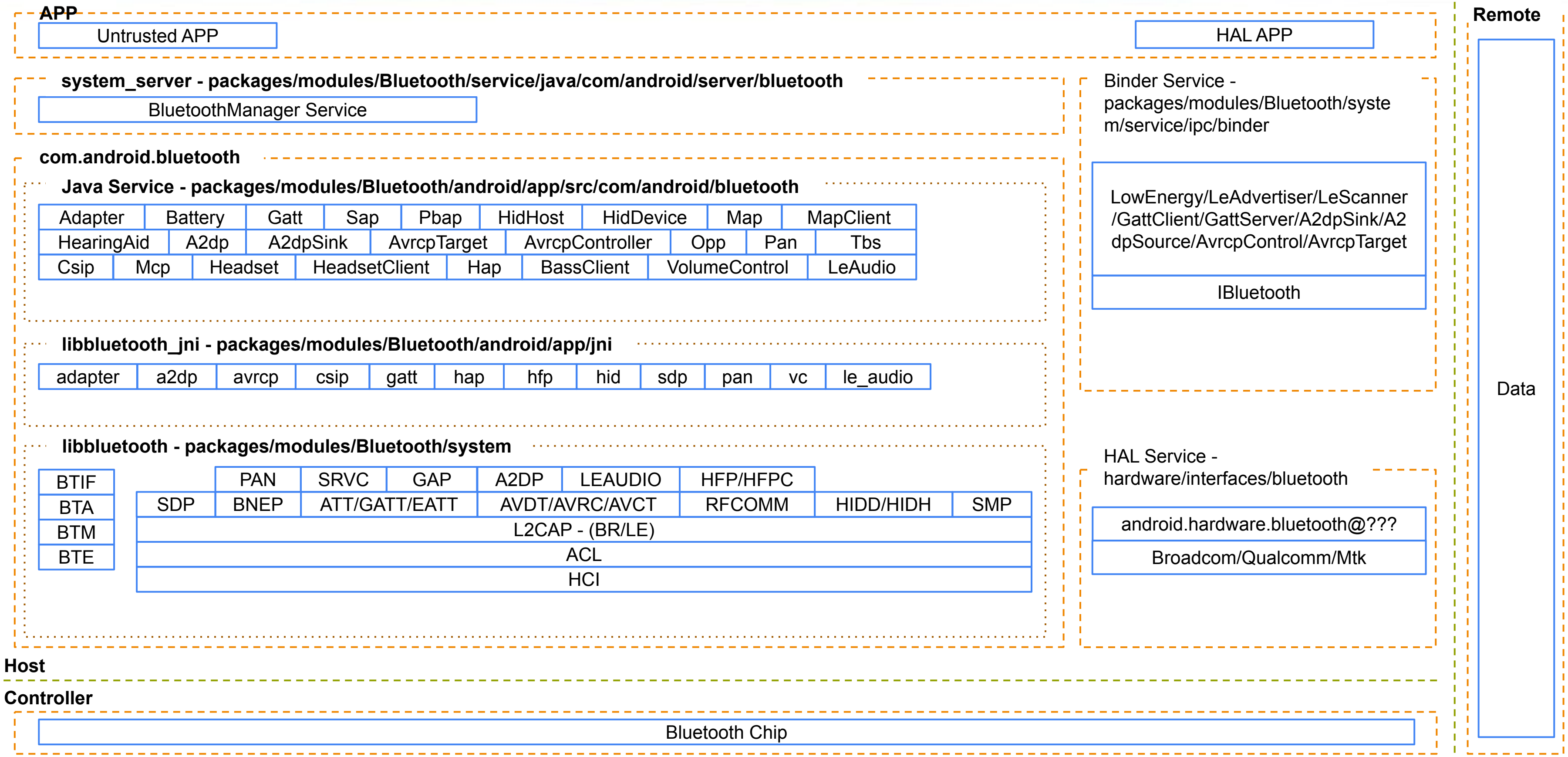
- From January 2017 to October 2022, at least 148 vulnerabilities have been disclosed, including 99 high vulnerabilities and 49 critical vulnerabilities
- Most are memory corruption types



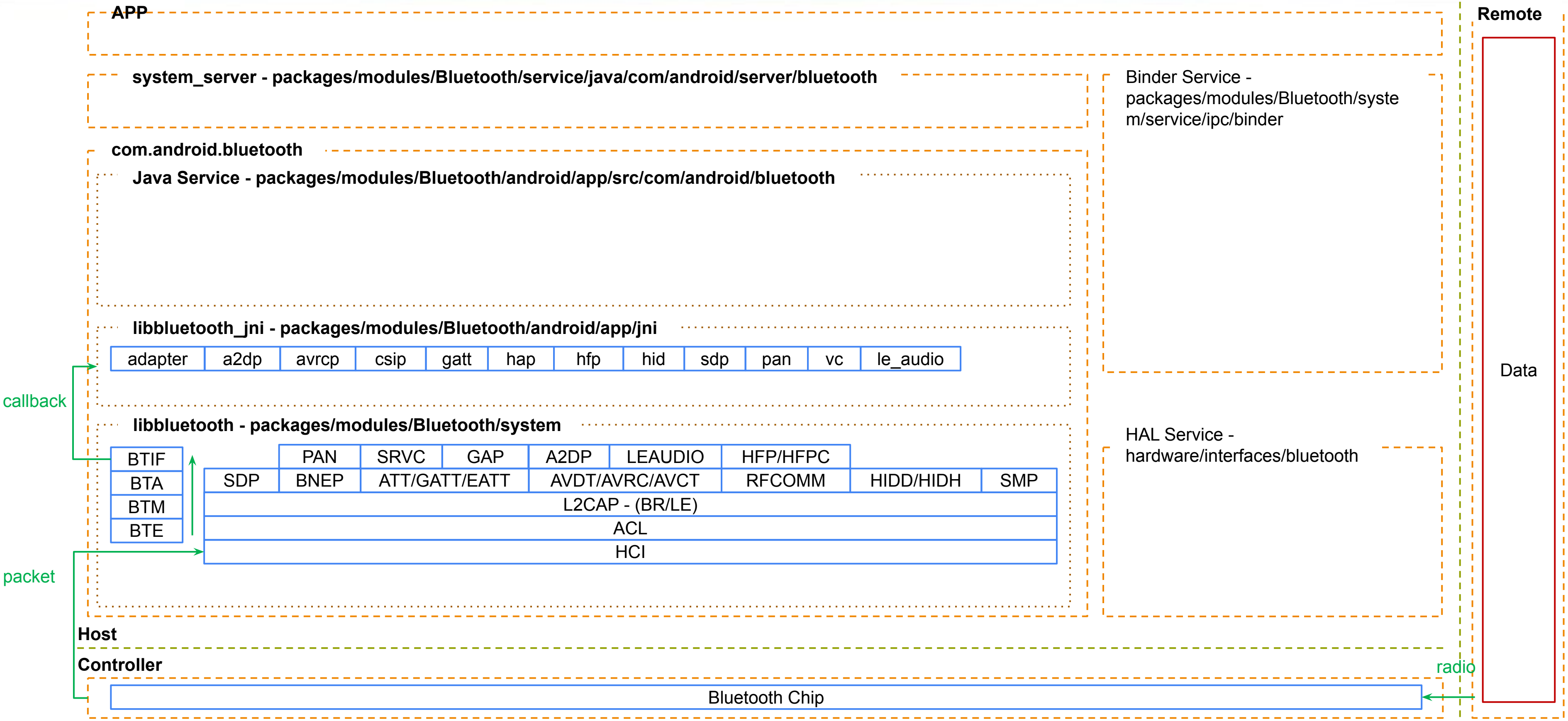


Agenda

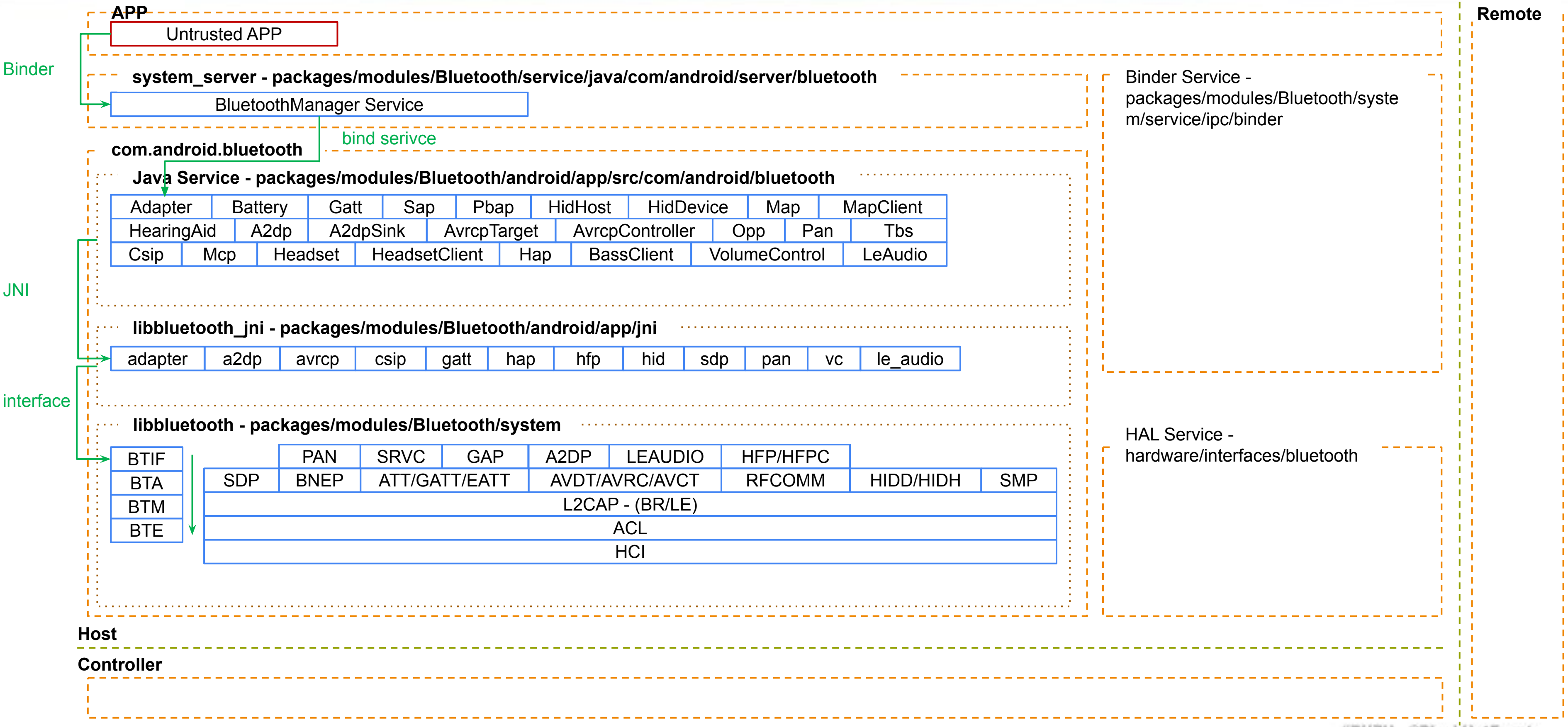
- Introduction to Android Bluetooth
- Attack surfaces analysis
- Finding vulnerabilities
- Summary

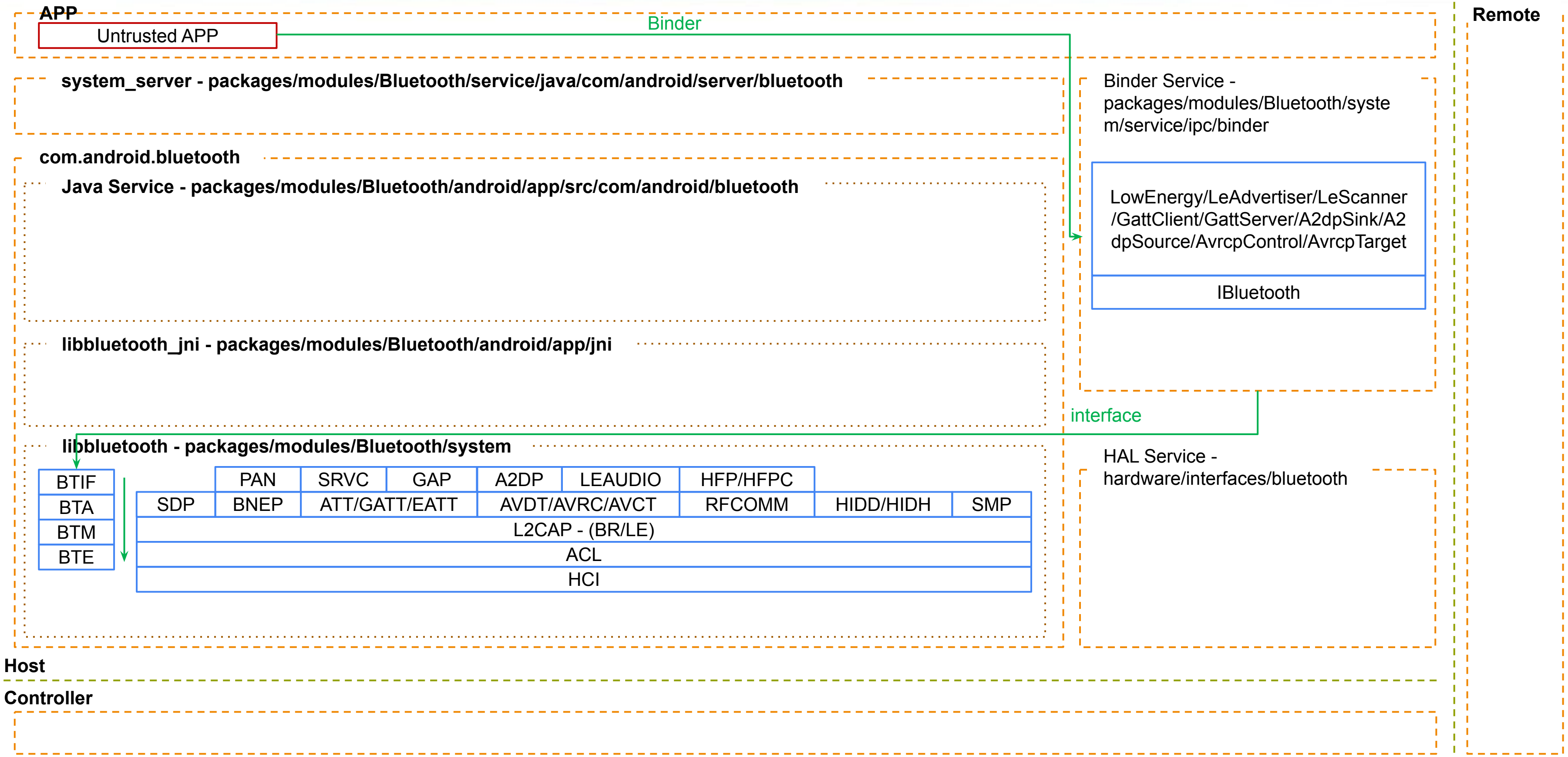

APP
Untrusted APP
HAL APP
Remote
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
BluetoothManager Service
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
LowEnergy/LeAdvertiser/LeScanner/GattClient/GattServer/A2dpSink/A2dpSource/AvrcpControl/AvrcpTarget
IBluetooth
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
Adapter
Battery
Gatt
Sap
Pbap
HidHost
HidDevice
Map
MapClient
HearingAid
A2dp
A2dpSink
AvrcpTarget
AvrcpController
Opp
Pan
Tbs
Csip
Mcp
Headset
HeadsetClient
Hap
BassClient
VolumeControl
LeAudio
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
adapter
a2dp
avrcp
csip
gatt
hap
hfp
hid
sdp
pan
vc
le_audio
libbluetooth - packages/modules/Bluetooth/system
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
HAL Service - hardware/interfaces/bluetooth
android.hardware.bluetooth@???
Broadcom/Qualcomm/Mtk
Data
Host
Controller
Bluetooth Chip

Attack scenario 1 - malicious Bluetooth radio


APP
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
adapter
a2dp
avrcp
csip
gatt
hap
hfp
hid
sdp
pan
vc
le_audio
callback
libbluetooth - packages/modules/Bluetooth/system
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
packet
HAL Service - hardware/interfaces/bluetooth
Host
Controller
Bluetooth Chip
radio
Remote
Data

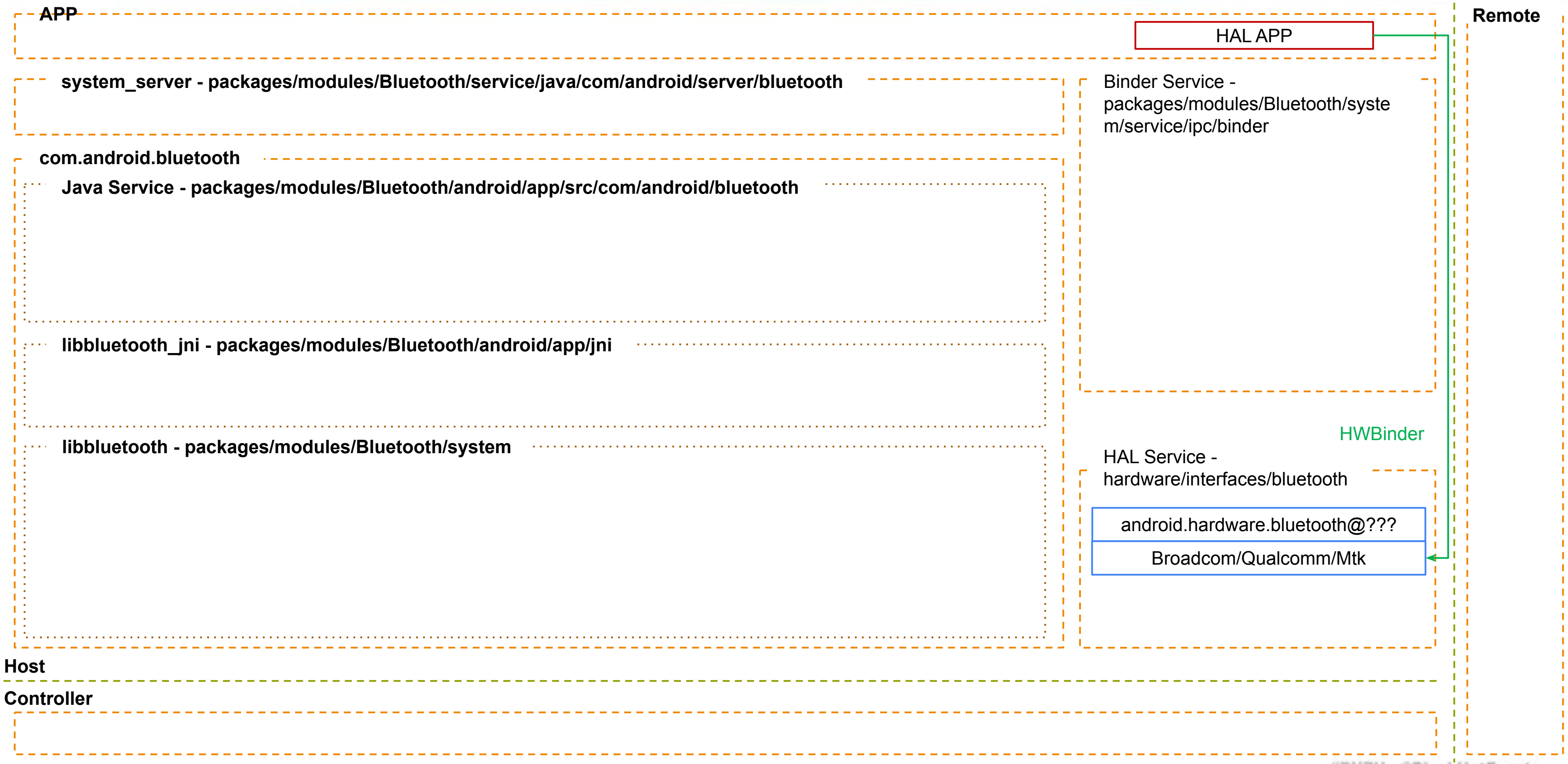
Attack scenario 2 - malicious untrusted APP


APP
Untrusted APP
Binder
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
BluetoothManager Service
bind serivce
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
Adapter | Battery | Gatt | Sap | Pbap | HidHost | HidDevice | Map | MapClient
HearingAid | A2dp | A2dpSink | AvrcpTarget | AvrcpController | Opp | Pan | Tbs
Csip | Mcp | Headset | HeadsetClient | Hap | BassClient | VolumeControl | LeAudio
JNI
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
adapter | a2dp | avrcp | csip | gatt | hap | hfp | hid | sdp | pan | vc | le_audio
interface
libbluetooth - packages/modules/Bluetooth/system
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
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
HAL Service - hardware/interfaces/bluetooth
Host
Controller
Remote

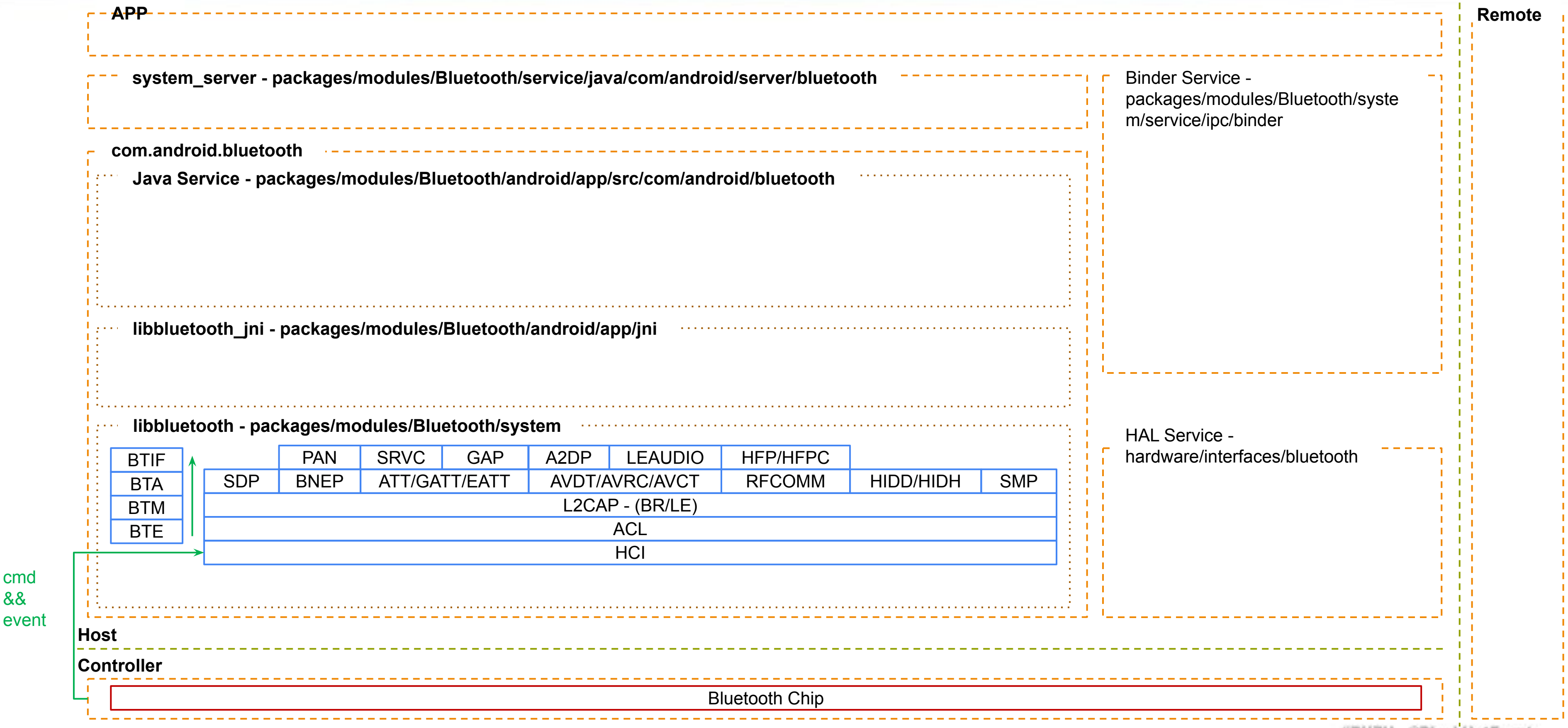


APP
Untrusted APP
Binder
Remote
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
LowEnergy/LeAdvertiser/LeScanner/GattClient/GattServer/A2dpSink/A2dpSource/AvrcpControl/AvrcpTarget
IBluetooth
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
interface
libbluetooth - packages/modules/Bluetooth/system
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
HAL Service - hardware/interfaces/bluetooth
Host
Controller

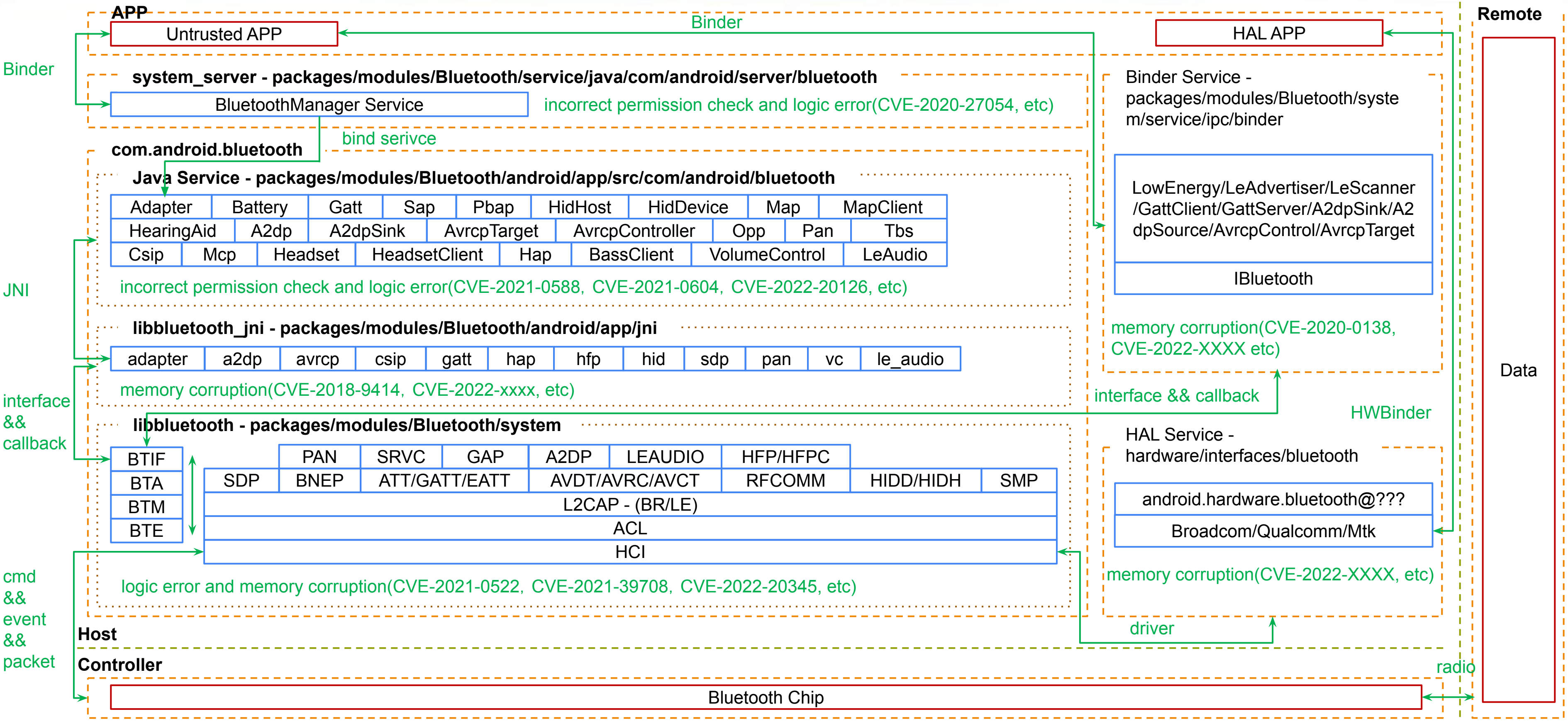
Attack scenario 3 - malicious HAL APP


APP
HAL APP
Remote
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
libbluetooth - packages/modules/Bluetooth/system
HWBinder
HAL Service - hardware/interfaces/bluetooth
android.hardware.bluetooth@???
Broadcom/Qualcomm/Mtk
Host
Controller

Attack scenario 4 - malicious Bluetooth chip


APP
Remote
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
libbluetooth - packages/modules/Bluetooth/system
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
HAL Service - hardware/interfaces/bluetooth
cmd
&&
event
Host
Controller
Bluetooth Chip

Full attack scenarios


APP
Untrusted APP
Binder
HAL APP
Remote
Binder
system_server - packages/modules/Bluetooth/service/java/com/android/server/bluetooth
BluetoothManager Service
incorrect permission check and logic error(CVE-2020-27054, etc)
Binder Service - packages/modules/Bluetooth/system/service/ipc/binder
bind serivce
com.android.bluetooth
Java Service - packages/modules/Bluetooth/android/app/src/com/android/bluetooth
Adapter
Battery
Gatt
Sap
Pbap
HidHost
HidDevice
Map
MapClient
HearingAid
A2dp
A2dpSink
AvrcpTarget
AvrcpController
Opp
Pan
Tbs
Csip
Mcp
Headset
HeadsetClient
Hap
BassClient
VolumeControl
LeAudio
incorrect permission check and logic error(CVE-2021-0588, CVE-2021-0604, CVE-2022-20126, etc)
LowEnergy/LeAdvertiser/LeScanner/GattClient/GattServer/A2dpSink/A2dpSource/AvrcpControl/AvrcpTarget
IBluetooth
JNI
libbluetooth_jni - packages/modules/Bluetooth/android/app/jni
adapter
a2dp
avrcp
csip
gatt
hap
hfp
hid
sdp
pan
vc
le_audio
memory corruption(CVE-2018-9414, CVE-2022-xxxx, etc)
memory corruption(CVE-2020-0138, CVE-2022-XXXX etc)
Data
interface && callback
interface && callback
HWBinder
libbluetooth - packages/modules/Bluetooth/system
HAL Service - hardware/interfaces/bluetooth
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
android.hardware.bluetooth@???
Broadcom/Qualcomm/Mtk
memory corruption(CVE-2022-XXXX, etc)
logic error and memory corruption(CVE-2021-0522, CVE-2021-39708, CVE-2022-20345, etc)
cmd && event && packet
driver
Host
Controller
radio
Bluetooth Chip

Agenda

- Introduction to Android Bluetooth
- Attack surfaces analysis
- Finding vulnerabilities
- Summary

How to discover the bugs

- Code audit
  - Goal - focus on full life cycle of data event/commands/packets in different scenarios, including receiving, subcontracting, packaging, parsing, handling, circulation, storage, recycling, etc
  - Disadvantage - take significant investment
  - Solution - combine with CodeQL
- Fuzz
  - Goal - focus on some complex code fragments, avoid missing some ignored issues
  - Disadvantage - remote fuzzing can handle various complex state machines, but inefficient and costly
  - Solution - local fuzzing with simulated state machine

Basic knowledge

- BT_HDR
  - define the header of each buffer used in the Bluetooth stack

```
typedef struct {
    uint16_t event;
    uint16_t len; // packet length
    uint16_t offset;
    uint16_t layer_specific;
    uint8_t data[]; // packet data
} BT_HDR;

BT_HDR* p_msg;
uint8_t* p_req = (uint8_t*) (p_msg + 1) + p_msg->offset;
uint8_t* p_req_end = p_req + p_msg->len;
```

- *_STREAM_TO*
  - Used to read values from input packet data

```
#define BE_STREAM_TO_UINT8(u8, p) \
  {                               \
    (u8) = (uint8_t)(*(p));       \
    (p) += 1;                     \
  }
#define BE_STREAM_TO_UINT16(u16, p)
#define BE_STREAM_TO_UINT24(u32, p)
#define BE_STREAM_TO_UINT32(u32, p)
#define BE_STREAM_TO_UINT64(u64, p)
#define BE_STREAM_TO_ARRAY(p, a, len)
```

Weak code patterns - integer overflow in stream deserialization

- A simple example

```
int xxx_data_ind(uint8_t* data, size_t len) {
    uint16_t num;
    BE_STREAM_TO_UINT16(num, data); // source - num is read via BE_STREAM_TO_UINT16
    uint16_t size = num * sizeof(int); // sink - num is used in arithmetic operation, and result maybe larger
than the maximum value of the storage variable
    int* dst = (int*) malloc(size); // allocated memory size is smaller than expected
    for (uint16_t i = 0; i < num; ++i) BE_STREAM_TO_UINT32(dst[i], data); // out of bounds write
}
```

- Code is possible vulnerable if tainted data flows from theses macros source to a sink in the arithmetic expression

Weak code patterns - integer overflow in stream deserialization

- Step 1 - find all calls of BE_STREAM_TO* or STREAM_TO*

```
// CodeQL code
class BTStreamMacroInvocationExpr extends AssignExpr {
    BTStreamMacroInvocationExpr() {
        // BE_STREAM_TO_UINT8 or STREAM_TO_UINT8
        this.getRValue()
            .(PointerDereferenceExpr)
            .getOperand()
            .(VariableAccess)
            .getTarget()
            .getName()

.regexpMatch("p|pp|p_data|p_req|p_reply|p_buf|p_stream|value|stream|data")
        or
        // BE_STREAM_TO_UINT16 or STREAM_TO_UINT16
    }
}
```

```
// c++ code
BE_STREAM_TO_UINT8(id, p);
BE_STREAM_TO_UINT32(u32, value);
STREAM_TO_UINT16(num, p_req);
STREAM_TO_UINT24(u24, data);
```

Weak code patterns - integer overflow in stream deserialization

- Step 2 - check the source corresponds to this expression

```
// CodeQL code
override predicate isSource(DataFlow::Node source) {
    exists(AssignExpr ae |
        ae instanceof BTStreamMacroInvocationExpr and
        source.asExpr() = ae
    )
}
```

Weak code patterns - integer overflow in stream deserialization

- Step 3 - check the sink corresponds to an arithmetic expression, and the maximum value of the result is greater than the maximum value of the lvalue

```
// CodeQL code
override predicate isSink(DataFlow::Node sink) {
    exists(AssignAddExpr aae |
        aae.getRValue() = sink.asExpr() and
        aae.getLValue().getType().getSize() <=
        aae.getRValue().getExplicitlyConverted().getType().getSize()
    ) or
    exists(AddExpr ae |
    ) or
    exists(MulExpr me |
    ) or
    // ...
}
```

```
// c++ code
uint16_t len, size;
BE_STREAM_TO_UINT16(len, p);
// possible integer overflow
size += len;
```

```
// c++ code
uint16_t len, size;
BE_STREAM_TO_UINT8(len, p);
// ingore this
size += len;
```

Weak code patterns - integer overflow in stream deserialization

- Step 4 - run query,analysis results and report issue
  - 100+ results in 5 seconds
  - Most are false positives

```
// calculation logic of the right value is complex, the ql script can't properly handle
min_len += MIN(p_result->get_caps.count, AVRC_CAP_MAX_NUM_COMP_ID) * 3;
```

  - After analysis, discovered 4 security bugs

Weak code patterns - integer overflow in stream deserialization

- CVE-2022-20445
  - Triple integer overflow causes heap buffer overflow read bug

```
static void process_service_search_rsp(tCONN_CB* p_ccb, uint8_t* p_reply, uint8_t* p_reply_end) {
    uint16_t cur_handles, orig;

    // cur_handles can take on any value from 0 to UINT16_MAX
    BE_STREAM_TO_UINT16(cur_handles, p_reply);
    orig = p_ccb->num_handles;
    // p_ccb->num_handles is uint16_t type, thus if cur_handles is sufficiently large, can result in an integer
overflow causing p_ccb->num_handles to be less than orig
    p_ccb->num_handles += cur_handles;
    // double integer overflow in "p_ccb->num_handles - orig" and "p_reply + ((p_ccb->num_handles - orig) * 4) +
1", causing this check to be bypassed
    if (p_reply + ((p_ccb->num_handles - orig) * 4) + 1 > p_reply_end) return;
    for (uint16_t xx = orig; xx < p_ccb->num_handles; xx++) BE_STREAM_TO_UINT32(p_ccb->handles[xx], p_reply);
}
```

Weak code patterns - integer overflow in stream deserialization

- CVE-2022-20410
  - Bypass length check due to integer overflow, leading to heap buffer overflow read

```
uint16_t min_len = 0;
min_len += 8;
// "p_attrs[i].name.str_len" can take any value form 0 to UINT16_MAX
BE_STREAM_TO_UINT16(attr_entry->name.str_len, p);
// possible integer overflow, "min_len" may less than expected
min_len += attr_entry->name.str_len;
// bypass this length check
if (pkt_len < min_len) goto browse_length_error;
attr_entry->name.p_str = (uint8_t*) osi_malloc(attr_entry->name.str_len * sizeof(uint8_t));
// oob read due to "attr_entry->name.str_len" could be larger than real size of input data
BE_STREAM_TO_ARRAY(p, attr_entry->name.p_str, attr_entry->name.str_len);
```

Weak code patterns - use after free due to ignoring error status

- A simple example

```
int some_function(BT_HDR* p_msg) {
    if (/*...*/) {
        osi_free(p_msg); // free p_msg
        return -1;
    }
    return 0;
}
BT_HDR* p_msg = (BT_HDR*) osi_malloc(sizeof(BT_HDR) + 4096);
some_function(p_msg); // missing check the return value
uint8_t* p = (uint8_t*) (p_msg + 1) + p_msg->len; // uaf
```

- Code is possible vulnerable if there is reachability flow between deallocation function call source and dereference sink

Weak code patterns - use after free due to ignoring error status

- Step 1 - define osi_free function

```
// CodeQL code
private class SpecialDeallocationFunction extends DeallocationFunction {
    int freedArg;

    SpecialDeallocationFunction() {
        hasGlobalOrStdOrBslName([
            // --- AOSP Bluetooth library allocation
            "osi_free", "osi_free_and_reset"
        ]) and freedArg = 0
    }

    override int getFreedArg() { result = freedArg }
}
```

```
// C++ code
void osi_free(void* ptr);
```

Weak code patterns - use after free due to ignoring error status

- Step 2 - find all calls that direct or possible call to deallocation function

```
// CodeQL code
predicate deallocCallOrIndirect(FunctionCall fc, VariableAccess va) {
    // direct free call
    fc.(DeallocationExpr).getFreedExpr() = va or
    // direct free call with assigned value
    fc.(DeallocationExpr).getFreedExpr()
    .(VariableAccess).getTarget().getInitializer().getExpr() = va or
    // // indirect free call
    exists(FunctionCall midcall, Function mid, int arg |
        fc.(Call).getArgument(arg) = va and
        mayCallFunction(fc, mid) and
        midcall.getEnclosingFunction() = mid and
        deallocCallOrIndirect(midcall, mid.getParameter(arg).(Variable).getAnAccess())
    )
}
```

```
// C++ code
// direct free call
osi_free(p_msg);

// direct free call with assigned value
void* p_buf = p_msg;
osi_free(p_buf);

// indirect free call
do_some_work(p_buf);
void do_some_work(BT_HDR* p_buf) {
    osi_free(p_buf);
}
```

Weak code patterns - use after free due to ignoring error status

- Step 3 - check an expression that may dereference variable

```
// CodeQL code
predicate isDerefByCallExpr(Call c, int i, VariableAccess va,
StackVariable v) {
    v.getAnAccess() = va and
    va = c.getAnArgumentSubExpr(i) and
    not c.passesByReference(i, va) and
    (c.getTarget().hasEntryPoint() implies
    isDerefExpr(_, c.getTarget().getParameter(i)))
}

predicate isDerefExpr(Expr e, StackVariable v) {
    v.getAnAccess() = e and dereferenced(e) or
    isDerefByCallExpr(_, _, e, v)
}
```

```
// C++ code
// example 1
uint16_t len = p_msg->len;

// example 2
p_msg->offset += 4;

// example 3
uint8_t* p_start =
    (uint8_t*) (p_msg + 1) + p_msg->offset;
```

Weak code patterns - use after free due to ignoring error status

- Step 4 - do reachability analysis

```
// CodeQL code
class UseAfterFreeReachability extends StackVariableReachability {
    UseAfterFreeReachability() {
        this = "UseAfterFreeDueToIgnoringErrorStatusInAospBluetooth"
    }
    override predicate isSource(ControlFlowNode node, StackVariable v) {
        deallocCallOrIndirect(node, v.getAnAccess())
    }
    override predicate isSink(ControlFlowNode node, StackVariable v) {
        isDerefExpr(node, v) and not deallocCallOrIndirect(_, node)
    }
    override predicate isBarrier(ControlFlowNode node, StackVariable v) {
        definitionBarrier(v, node) or deallocCallOrIndirect(node, v.getAnAccess())
    }
}
```

Weak code patterns - use after free due to ignoring error status

- Step 5 - run query, analysis results and report issue
  - 30+ results in 5 seconds
  - Some are false positives due to complex condition judgment between edges
  - After analysis, discovered 3 security bugs

Weak code patterns - use after free due to ignoring error status

- CVE-2022-20447
  - Use after free due to the code ignores the possibility that the bnepu_check_send_packet function may release the p_buf object

```
// in special case, p_buf will be freed bu result is still BNEP_SUCCESS
result = BNEP_WriteBuf(pcb->handle, dst, p_buf, protocol, &src, ext);
if (result == BNEP_IGNORE_CMD) {
   PAN_TRACE_DEBUG("PAN ignored data buf write to PANU");
   pcb->write.errors++;
   return PAN_IGNORE_CMD;
} else if (result != BNEP_SUCCESS) {
   PAN_TRACE_ERROR("PAN failed to send data buf to the PANU");
   pcb->write.errors++;
   return (tPAN_RESULT)result;
}
pcb->write.octets += p_buf->len;
pcb->write.packets++;
```

```
tBNEP_RESULT BNEP_WriteBuf(uint16_t handle, const RawAddress& p_dest_addr,
BT_HDR* p_buf,
   uint16_t protocol, const RawAddress* p_src_addr, bool fw_ext_present) {
   // no return value
   bnepu_check_send_packet(p_bcb, p_buf);
   return (BNEP_SUCCESS);
}
```

```
void bnepu_check_send_packet(tBNEP_CONN* p_bcb, BT_HDR* p_buf) {
   if (p_bcb->con_flags & BNEP_FLAGS_L2CAP_CONGESTED) {
      if (fixed_queue_length(p_bcb->xmit_q) >= BNEP_MAX_XMITQ_DEPTH) {
         // in this case, p_buf will be freed
         osi_free(p_buf);
      }
   }
}
```

Weak code patterns - others

- Double free due to ignoring error status
- Out of bounds read due to missing length check in *STREAM_TO_*
- Out of bounds write due to ignoring check of return value
- Deadloop due to index integer underflow
- And more...

Fuzzing - find the target function

- L2CA_Register(), L2CA_Register2(), L2CA_RegisterLECoc()
  - Used to register all L2CAP based profile data callback functions

```
extern uint16_t L2CA_Register(uint16_t psm, const tL2CAP_APPL_INFO& p_cb_info,
                              bool enable_snoop, tL2CAP_ERTM_INFO* p_ertm_info,
                              uint16_t my_mtu, uint16_t required_remote_mtu,
                              uint16_t sec_level);
uint16_t L2CA_Register2(uint16_t psm, const tL2CAP_APPL_INFO& p_cb_info,
                        bool enable_snoop, tL2CAP_ERTM_INFO* p_ertm_info,
                        uint16_t my_mtu, uint16_t required_remote_mtu,
                        uint16_t sec_level);
```

Fuzzing - find the target function

- pL2CA_DataInd_Cb

```
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;
```

https://cs.android.com/

- 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)
  81: 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)
  326: 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;

Fuzzing - SDP

- sdp_data_ind
  - Handle SDP data received from L2CAP
- sdp_disc_server_rsp
  - Handle response data from server
- sdp_server_handle_client_req
  - Handle request data from client

```c
static void sdp_data_ind(uint16_t l2cap_cid, BT_HDR* p_msg) {
  tCONN_CB* p_ccb;

  /* Find CCB based on CID */
  p_ccb = sdpu_find_ccb_by_cid(l2cap_cid);
  if (p_ccb != NULL) {
    if (p_ccb->con_state == SDP_STATE_CONNECTED) {
      if (p_ccb->con_flags & SDP_FLAGS_IS_ORIG)
        sdp_disc_server_rsp(p_ccb, p_msg);
      else
        sdp_server_handle_client_req(p_ccb, p_msg);
    } else {
      SDP_TRACE_WARNING(
          "SDP - Ignored L2CAP data while in state: %d, CID: 0x%x",
          p_ccb->con_state, l2cap_cid);
    }
  } else {
    SDP_TRACE_WARNING("SDP - Rcvd L2CAP data, unknown CID: 0x%x", l2cap_cid);
  }

  osi_free(p_msg);
}
```

Fuzzing - SDP

- Write a test harness
  - Init sdp stack
  - Init sdp connection control block to simulate a real sdp connection
  - Generate some sdp databases
  - Generate some states
  - Generate sdp packet data
  - Continuously fuzz sdp server and client interfaces

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size)
{
   FuzzedDataProvider fdp(data, size);

   sdp_init();
   init_ccb();
   init_sdp_db(fdp);

   size_t count = fdp.ConsumeIntegralInRange<int>(1, 20);
   while (count-- && sdp_db_vect.size() && fdp.remaining_bytes()) {
       if (fdp.ConsumeBool()) {
           sdp_disc_server_rsp_fuzzer(fdp);
       } else {
           sdp_server_handle_client_req_fuzzer(fdp);
       }
   }

   SDP_DeleteRecord(0);
   sdp_db_vect.clear();
   sdpu_release_ccb(p_ccb);
   sdp_free();

   return 0;
}
```

Results

- Since 2022, total 33 vulnerabilities have been found
  - 3 are duplicate
  - 8 have been fixed
  - others are still in process
- Found with CodeQL
  - 15 vuls, including CVE-2022-20445, CVE-2022-20222, CVE-2022-20362, CVE-2022-20283, CVE-2022-20447, CVE-2022-20410
- Found with fuzzing
  - 9 vuls, including CVE-2022-20224, CVE-2022-20229, CVE-2022-20140

Agenda

- Introduction to Android Bluetooth
- Attack surfaces analysis
- Finding vulnerabilities
- Summary

Conclusion

- Some basic information about Android Bluetooth
- Historical vulnerability analysis
- Full attack surfaces and vectors
- How to find the bugs
  - Weak code patterns
  - Fuzzing tricks
- CVE cases

Future work

- Write more CodeQL scripts to catch more vulnerable patterns
  - Dangling pointers nullification in C++ smart pointers
  - Race condition in JNI/main/hci threads
  - Type confusion in message queue
- Focus on new protocols and profiles implemented on Android 13
  - LeAudio
- Other vendors Bluetooth stack implementation
  - Qualcomm
  - Mediatek

Reference

- https://source.android.com/docs/core/connect/bluetooth
- https://source.android.com/security/bulletin/
- https://codeql.github.com/
- https://bughunters.google.com/leaderboard
- https://hackerone.com/qualcomm/thanks/2022

Thanks