Paint it Blue: Attacking the Bluetooth Stack

HEXACON

10-11 October 2025

Speakers

SYNACKTIV

Mehdi Talbi, PhD.
Security Researcher at Synacktiv

Etienne Helluy-Lafont, PhD.
Security Researcher at Synacktiv

Introduction

Introduction

SYNACKTIV

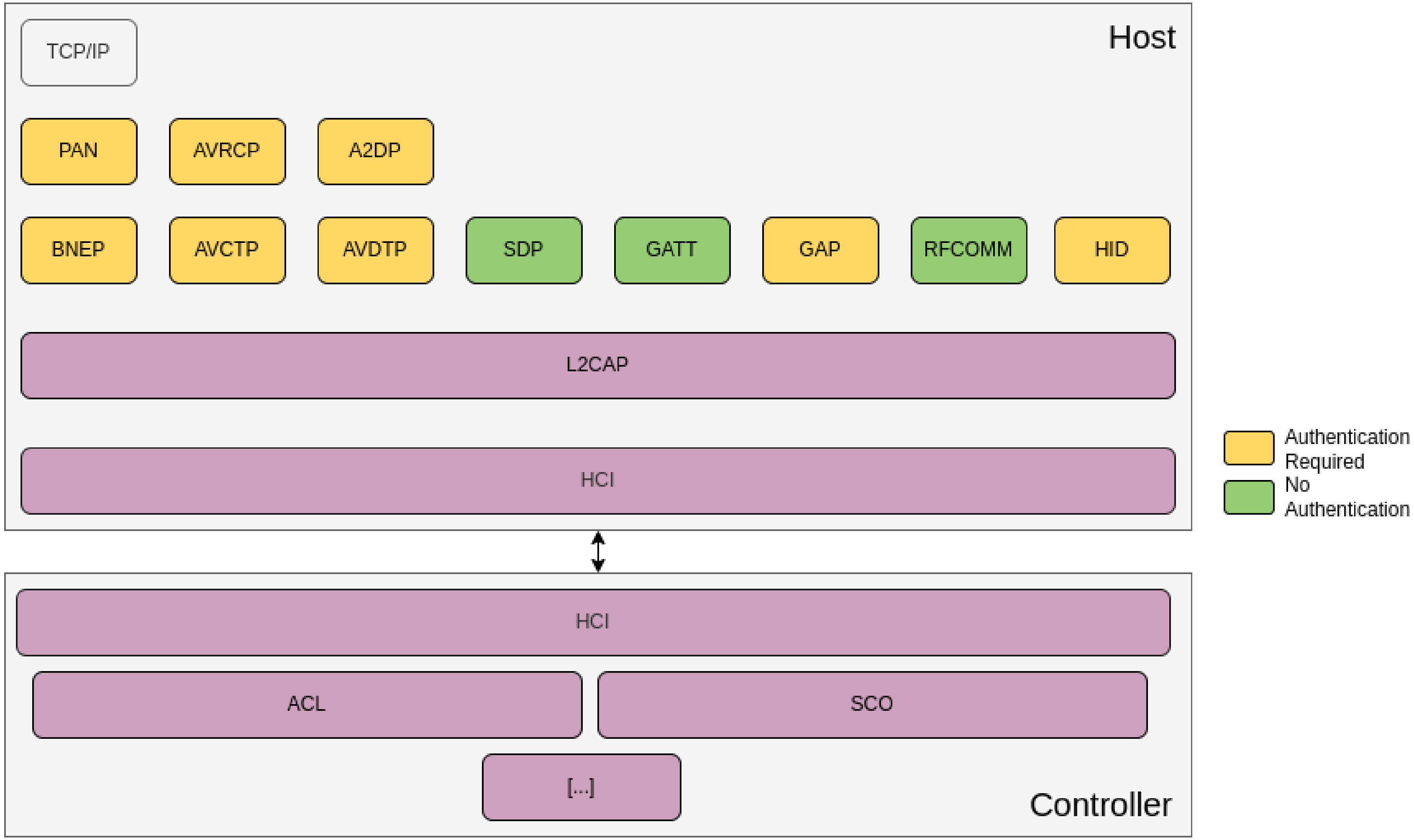
- Bluetooth is still an attacker's target of choice
  - Supported by every single mobile phones nowadays
  - Always-on on many devices
  - Proximity 0-click attack surface
- It has room for interesting vulnerabilities
  - e.g. Google's red team presentation at OffensiveCon'25
  - A collection of memory corruption in Android's BT stack
- Lets see how a full exploit can be developed

Outline

- Quick overview of the Bluetooth Stack
- CVE-2023-40129
- Exploitation primitives
- Code execution on Jemalloc devices
- Code execution on Scudo devices
- Conclusion

The Bluetooth Stack

SYNACKTIV



The Blueblue Framework

SYNACKTIV

- Python framework built on top of BlueBorne's code
- Built on top of the HCI layer
- Simple implementations for L2CAP, ERTM channels, etc.

```
acl = ACLConnection(src_bdaddr, dst_bdaddr, auth_mode = 'justworks')
gatt = acl.l2cap_connect(psm=PSM_ATT, mtu=672)
gatt.send_frag(p8(GATT_READ)+p16(1234))
print(gatt.recv())
```

- Very convenient to try ideas on a Bluetooth stack

Authentication in Bluetooth

- Many Bluetooth services require authentication
  - GAP, BNEP, AVCTP, etc.
- Usually done by pairing, with pin verification
- Several methods available, with various security level
  - MITM resistant or no, …
- Android adds fine-grained ACL for paired devices
  - Access to contacts, SMS, etc.

Authentication in Bluetooth

SYNACKTIV

L2CAP Authentication in Floride

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

- Most channels require authentication + encryption

```
if (!L2CA_Register2(BT_PSM_BNEP, bnep_cb.reg_info, false /* enable_snoop */,
                    nullptr, BNEP_MTU_SIZE, BNEP_MTU_SIZE,
                    BTA_SEC_AUTHENTICATE | BTA_SEC_ENCRYPT)) {
    BNEP_TRACE_ERROR("BNEP - Registration failed");
    return BNEP_SECURITY_FAIL;
}
```

Authentication in Bluetooth

Just Works, Still Works

- But.. some devices have no input/output capabilities
  - No display or keyboard to verify a PIN
- There is an authentication method for this
  - It "Just Works"
  - Allows authenticating to Fluoride without user interaction
- Comes with some shortcomings
  - Breaks existing pairing with same Bluetooth Address (BDADDR)
  - Does not provide full access (not MITM resistant), ...

The Bug

The Bug

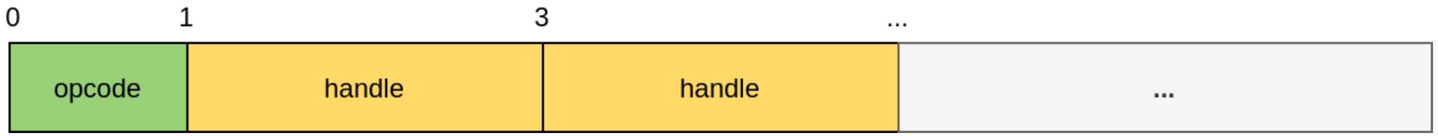
SYNACKTIV

⚠ CVE-2023-40129

- Heap overflow in the GATT server
- Reachable without authentication or user interaction
- Integer underflow leading to a 64 KB memcpy heap/heap

The Bug

- Fluoride implements a GATT client/server
  - Allows setting or getting data attributes
- The vulnerability affects `GATT_RSP_READ_MULTI_VAR`
- Command to request multiple attributes at once
  - Request: list of attribute's handles



  - Reply: length/value of returned attributes

The Bug

- Replying to GATT read multi requests

```
static void build_read_multi_rsp(tGATT_SR_CMD* p_cmd, uint16_t mtu) {
  uint16_t ii, total_len, len;
  uint8_t* p;
  bool is_overflow = false;

  len = sizeof(BT_HDR) + L2CAP_MIN_OFFSET + mtu;
  BT_HDR* p_buf = (BT_HDR*)osi_calloc(len); // [0]
  p_buf->offset = L2CAP_MIN_OFFSET;
  p = (uint8_t*)(p_buf + 1) + p_buf->offset;
```

- Declares length variables as short unsigned int

0. Allocate a buffer large enough to hold MTU
   - There is a vulnerability here too (CVE-2023-35673) but that's another story

The Bug

- Appending a value to the reply buffer

```
        total_len = (p_buf->len + p_rsp->attr_value.len);                    // [1]
        if (p_cmd->multi_req.variable_len) {
          total_len += 2;                                                    // [2]
        }
        if (total_len > mtu) {
          /* just send the partial response for the overflow case */
          len = p_rsp->attr_value.len - (total_len - mtu);                   // [3]
            [...]
        memcpy(p, p_rsp->attr_value.value, len);                             // [4]
```

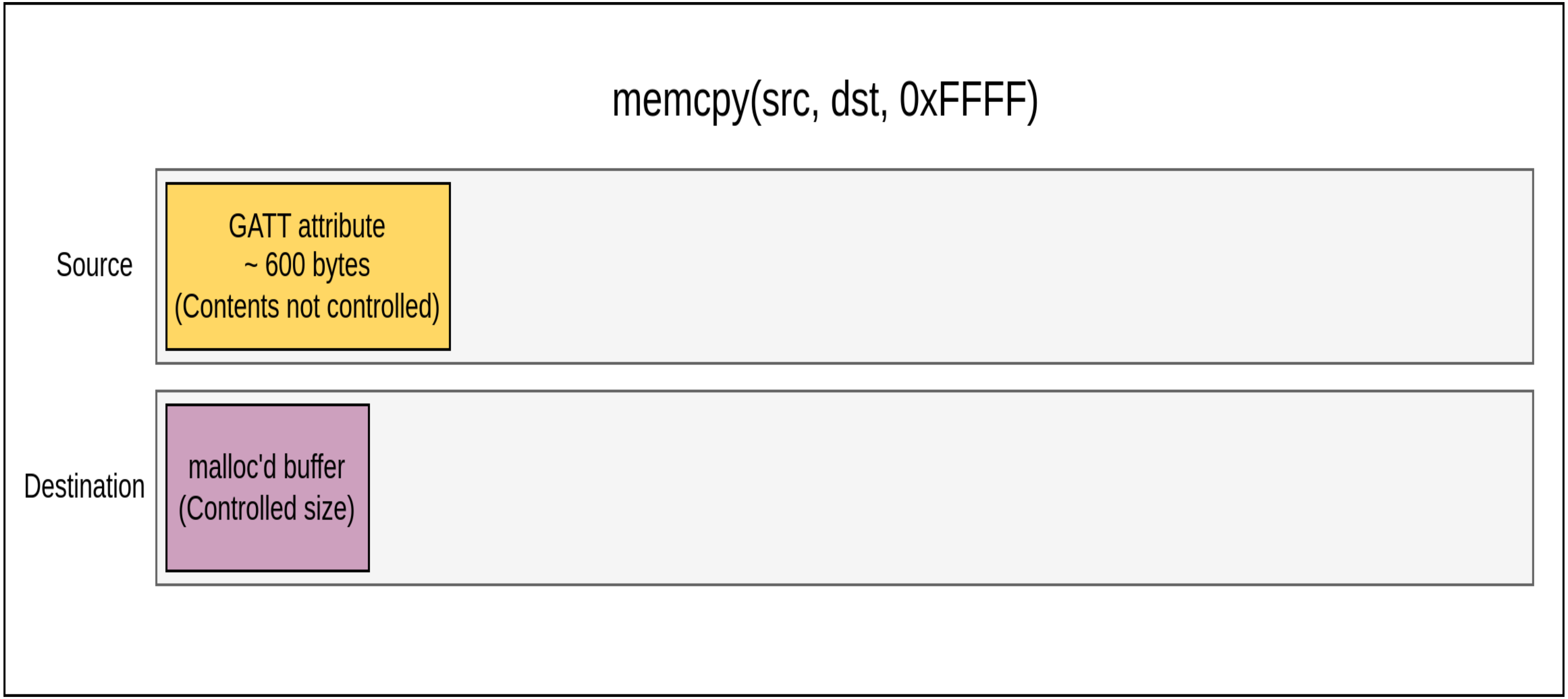
1. Compute the space required to append the attribute (**total_len**)
2. Add 2 to encode the attribute's length
3. Compute the length to append, but forgets to account for the 2-bytes from [2]
   - Can set **len** to **-1** or **-2** (as an unsigned short integer)
4. **Huge memcpy** of ~ **64 KB** (0xfffe-0xffff)

The Bug

SYNACKTIV

- Causes a massive heap overflow of ~ 64 KB
- **Source**: heap buffer of ~ **600 bytes** of GATT attribute (**not controlled** by the attacker)
  - Can be partially controlled post-pairing by setting GATT attributes
- **Destination**: Heap buffer with **controllable size**, depending on the MTU configuration
- A bit messy, but good enough for RCE !

The Bug



memcpy(src, dst, 0xFFFF)
Source
GATT attribute
~ 600 bytes
(Contents not controlled)
Destination
malloc'd buffer
(Controlled size)

Exploitation Primitives

Exploitation Primitives

Persistent Data Allocation

Heap spraying in Fluoride

- We need to control the heap layout
  - Put some controlled data in the source buffer
  - Shape the destination heap
- But there are virtually no persistent allocation in Fluoride
- Packet buffers are typically freed upon transmission to the controller

Solution

- Force packet allocations to become persistent

Exploitation Primitives

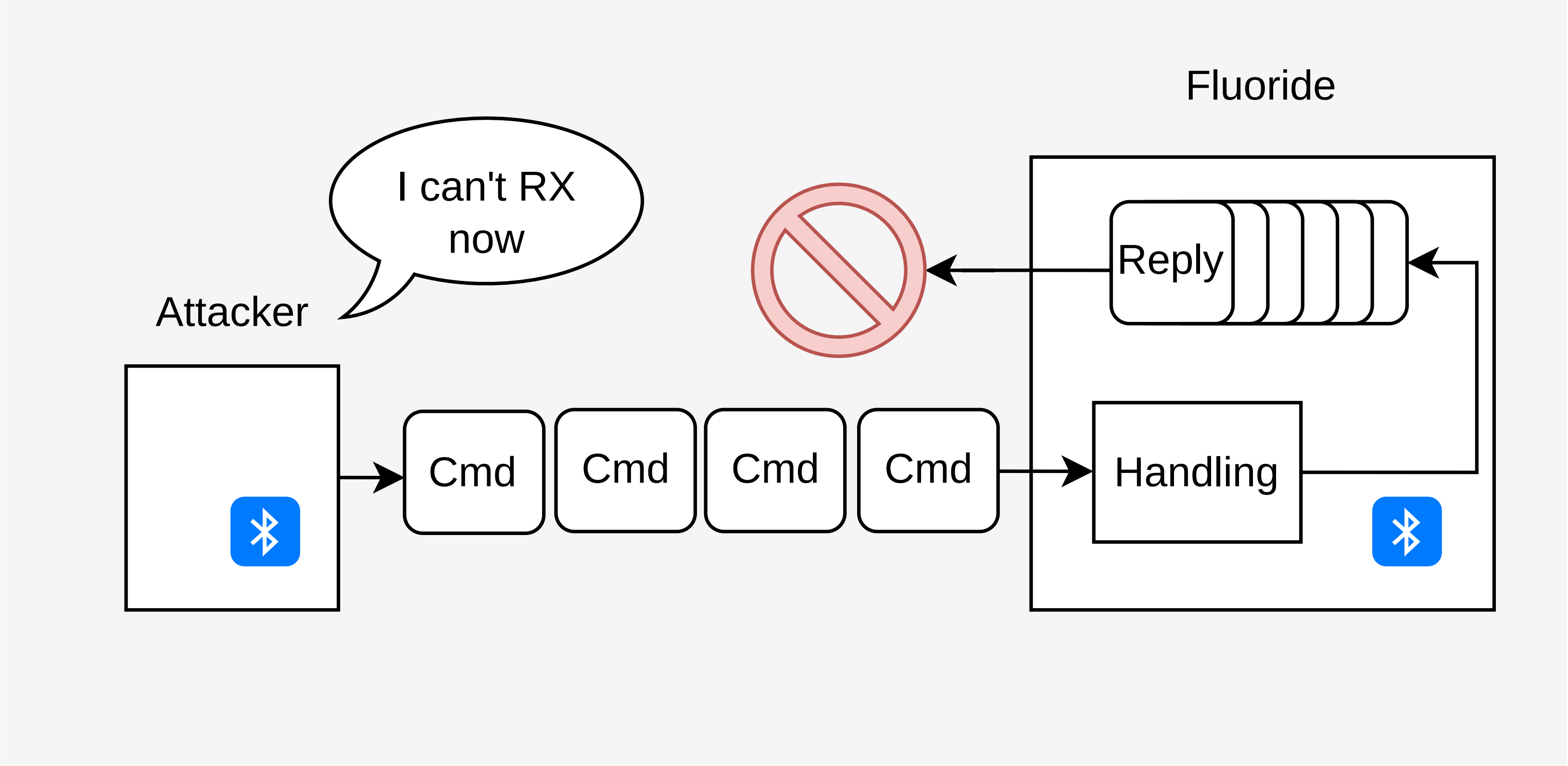
Persistent Data Allocation

ACL Congestion

- Control-flow feature offered by the Bluetooth specification
  - To avoid Bluetooth Controller's memory exhaustion
- Easy to toggle on Cypress Bluetooth Controllers
  - A vendor-specific HCI command allows us to simulate ACL congestion
- A peer under congestion can still send messages to the remote peer

Exploitation Primitives

Persistent Data Allocation



Exploitation Primitives

Persistent Data Allocation

ACL Congestion

- Fluoride gracefuly handles ACL congestion
- Messages are processed and responses are inserted into a queue
- Quota limits message queuing during congestion
  - But there is no quota on the signaling channel
- All pending messages are freed when the connection is closed

Exploitation Primitives

Controlled Data Allocation

- Invalid L2CAP config requests
  - Rejected options are sent back to the peer (`CONFIG REJ` messages)
  - Allocations of **controlled size** and **data**

```
void l2cu_send_peer_config_rej(tL2C_CCB* p_ccb, uint8_t* p_data,
                               uint16_t data_len, uint16_t rej_len) {
    uint16_t len, cfg_len, buf_space, len1;
    uint8_t *p, *p_hci_len, *p_data_end;
    uint8_t cfg_code;

    /* ... */

    len = BT_HDR_SIZE + HCI_DATA_PREAMBLE_SIZE + L2CAP_PKT_OVERHEAD +
          L2CAP_CMD_OVERHEAD + L2CAP_CONFIG_RSP_LEN;

    BT_HDR* p_buf = (BT_HDR*)osi_malloc(len + rej_len);

    /* ... */
}
```

Exploitation Primitives

Heap shaping primitives

SYNACKTIV

More shaping primitives

- Allocations that can be allocated / freed on demand
- Useful objects to build read / write primitives

Enhanced Retransmission Mode (ERTM)

- Reliable transport over L2CAP: Sequence numbering, ack, retransmission
- Two ways to force persistent allocations:
  - Start transmission with `seq_tx = 1`
    - → Since `seq_tx = 0` is missing, the peer holds all subsequent messages in memory
    - Controlled size + Controlled data
  - Do not acknowledge incoming messages

Exploitation Primitives

Persistent Data Allocation

Enhanced Retransmission Mode (ERTM) - Limitations

⚠ Quota

- ERTM messages limited by a quota
- UP to 10 messages per L2CAP channel

⚠ Authenticated channels

- ERTM is supported by a subset of L2CAP channels (GAP, AVCTP)
- Authentication is required on all ERTM-enabled channels

Exploitation Primitives

Read and Write Primitives

Bluetooth packets in Fluoride

- Simple data structure
  - **len** : Length of data
  - **offset** : Position of the data
- No pointer → Easy to forge

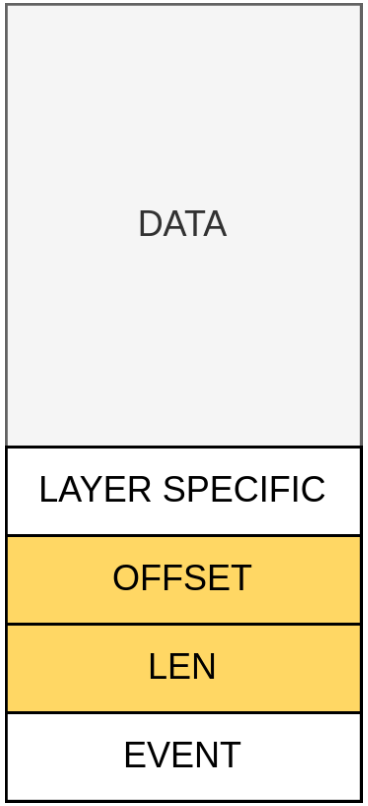
```
typedef struct {
  uint16_t event;
  uint16_t len;
  uint16_t offset;
  uint16_t layer_specific;
  uint8_t data[];
} BT_HDR;
```

Exploitation Primitives

Read and Write Primitives

Relative Read Primitive

1. Force Fluoride to send an ERTM fragment
2. Corrupt the pending fragment
    - → Alter `len` and `offset` fields
3. Request its retransmission
    - → Leak up to 64 KB of heap data



Exploitation Primitives

Read and Write Primitives

Forcing an ERTM transmission

- AVCTP browsing channel is a good candidate
  - Supports ERTM mode
- `GET_FOLDER_ITEMS` request:
  - Request metadata of a music playlist (song name, artist name, etc.)
  - Select metadata's attributes → Force a response message of a controlled size (same `bin` as vulnerable object)

Exploitation Primitives

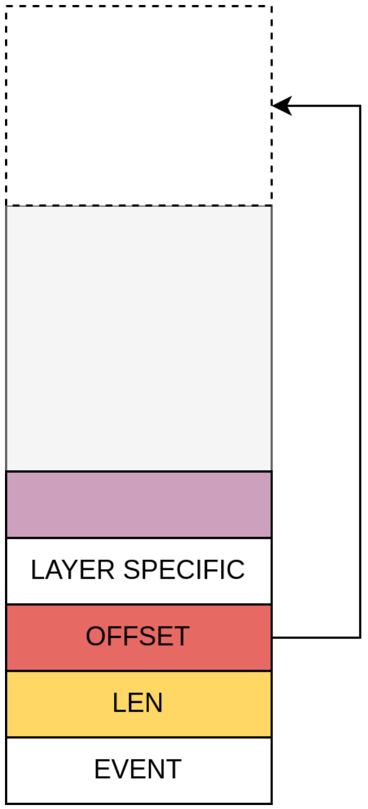
Read and Write Primitives

Relative Write Primitive

1. Send an ERTM fragment
2. Corrupt it to control **offset** and **len**
3. Send next fragment
   - → Subsequent fragments are copied using `len` and `offset`'s `BT_HDR` fields:

```
memcpy(((uint8_t*)(p_fcrb->p_rx_sdu + 1)) +
         p_fcrb->p_rx_sdu->offset +
         p_fcrb->p_rx_sdu->len,
         p, p_buf->len);

p_fcrb->p_rx_sdu->len += p_buf->len;
```



Exploitation Primitives

Code Execution

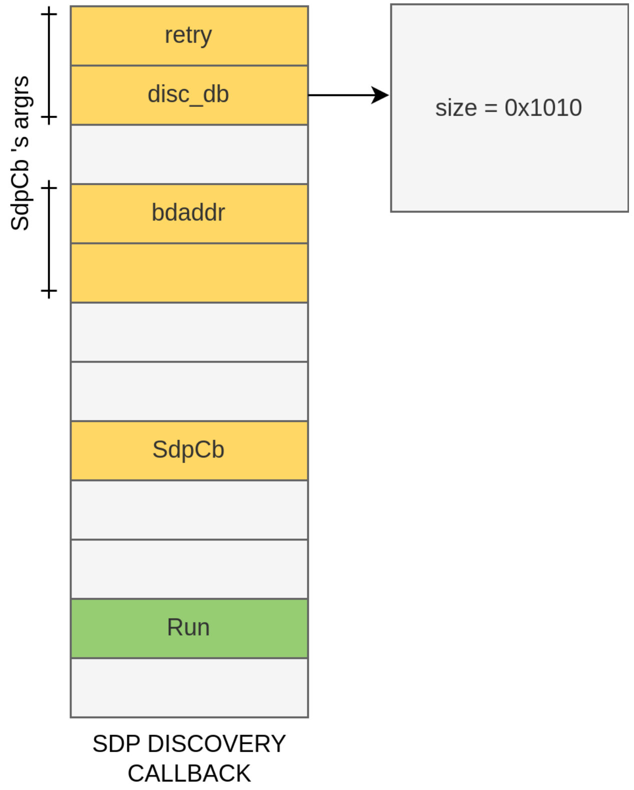
SYNACKTIV

Target Object

- Fluoride stack uses plenty of `callback` objects (from `libchrome`)
- Multiple function pointers

Target Callback

- The SDP discovery callback is a good candidate
- (Most of) Callback's arguments embedded in the object
- Callback **allocated** while establishing an AVRCP conn.
- Callback **triggered** when closing the related SDP conn.



Code Execution on Jemalloc Devices

Code Execution on Jemalloc Devices

Heap Shaping

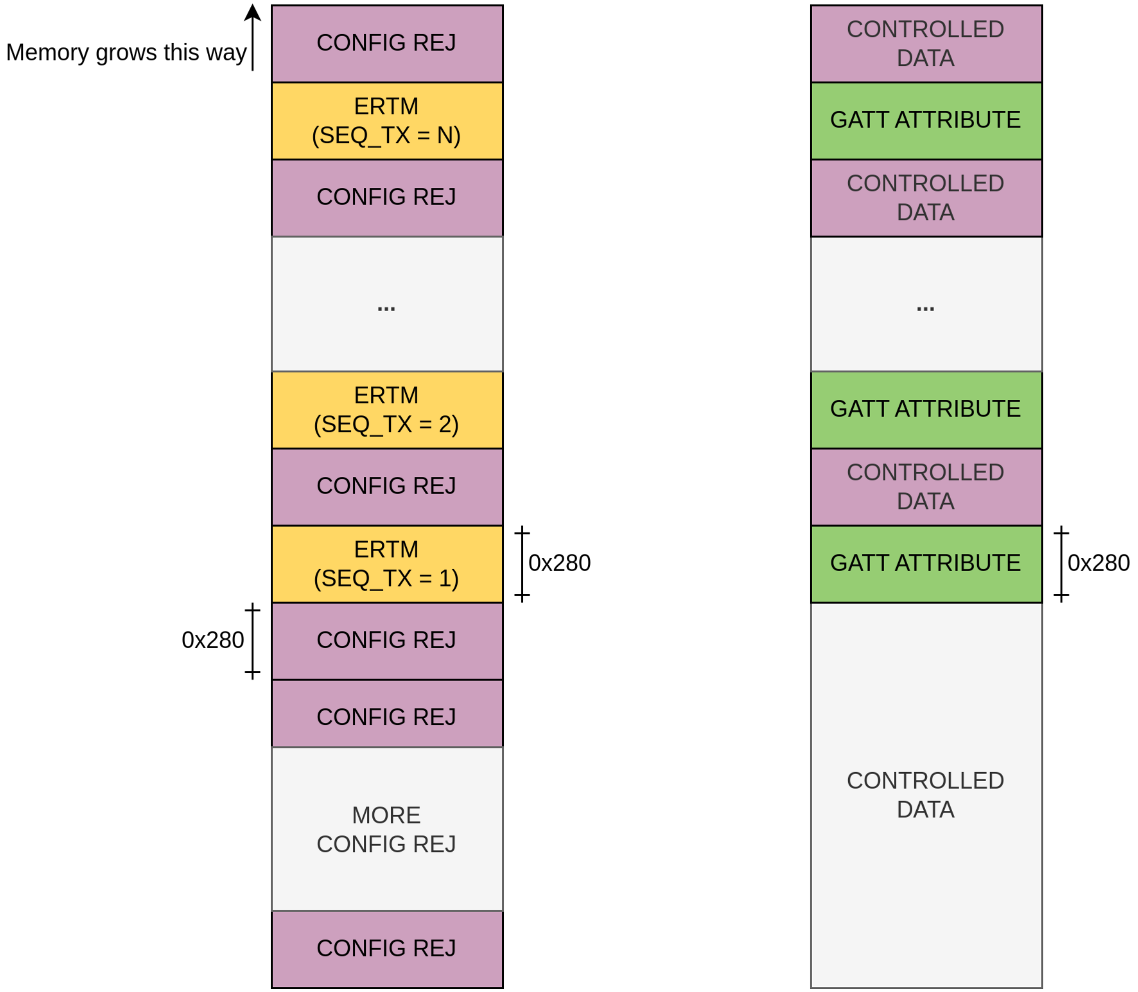
SYNACKTIV

Heap Shaping Strategy

1. Enable ACL congestion.
2. Spray multiple `CONFIG REJ` messages
3. Interleave ERTM message allocations during the spray
   - ERTM allocations are used to create "holes" in the heap
4. Disable ACL congestion
   - `CONFIG REJ` allocations are freed
5. Free the ERTM allocations
   - ERTM allocations are reused by the GATT-related allocations

Code Execution on Jemalloc Devices

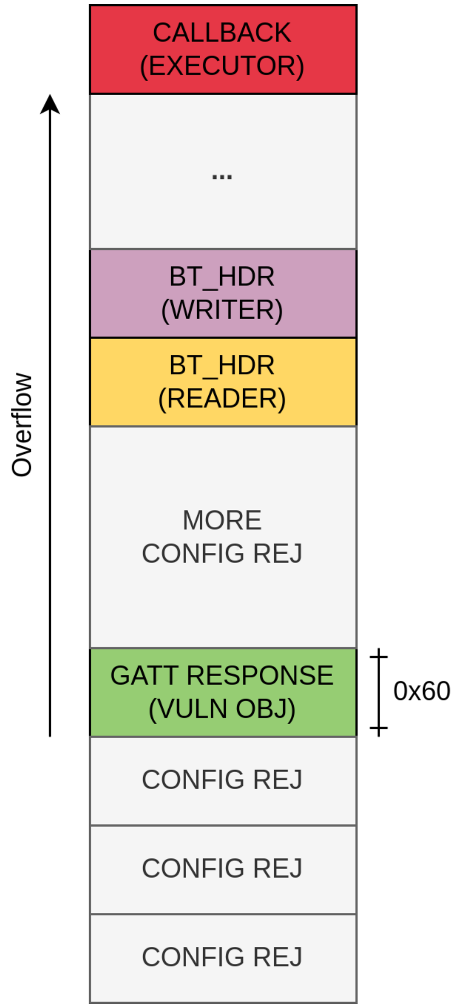
Heap Shaping - Source



Code Execution on Jemalloc Devices

Heap Shaping - Dest

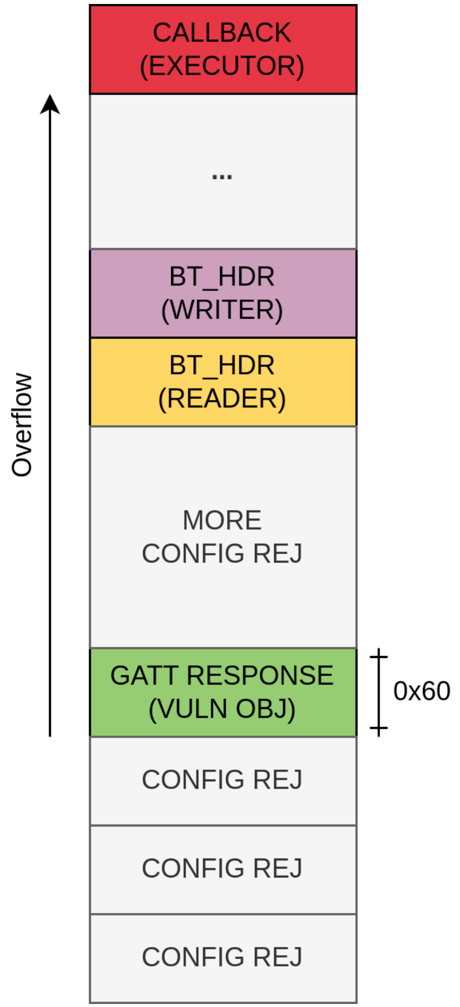
- Spray Multiple `CONFIG REJ` messages
- Create placeholders for `READER` and `WRITER` objects
  - With ERTM messages sent on a GAP channel
- Create placeholder for vulnerable object
  - With ERTM message sent on a second GAP channel
- Close first GAP channel
- Allocate `WRITER` and `READER` objects
- Close second GAP channel
- Trigger Overflow
- Allocate callback object



Code Execution on Jemalloc Devices

Exploitation Scenario

1. Shape the heap (src & dst)
2. Trigger overflow and corrupt `READER` & `WRITER` objects
3. Allocate the SDP Discovery Callback (`EXECUTOR` object)
4. Request the retransmission of the altered packet
5. Leak the content of the callback
6. Rewrite the content of the callback
7. Trigger the callback



Code Execution on Jemalloc Devices

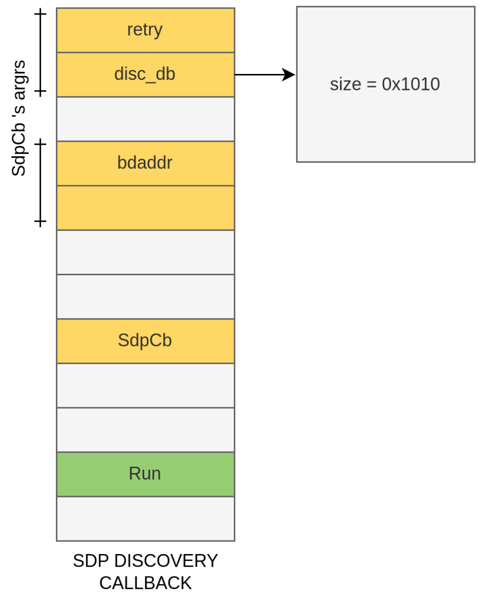
Code Execution

Arguments Control

- 3rd argument of *SdpCb* callback **not controlled**
- → call an intermediate function: **gadget** function

```
uint64_t gadget(gadget_t *obj)
{
    int64_t v1;
    uint8_t *v2;
    uint64_t *v3;

    v1 = obj->field_28;
    v2 = obj->field_20;
    v3 = (obj->field_30 + (v1 >> 1));
    if ( (v1 & 1) != 0 )
        v2 = *&v2[*v3];
    return (v2)(v3, obj->field_38, obj->field_40, obj->field_44, obj->field_4c);
}
```



Code Execution on Jemalloc Devices

Code Execution

Multiple Function Calls

- Call to **mprotect** + jump to **shellcode**

```
void list_clear(list_t* list) {
    CHECK(list != NULL);
    for (list_node_t* node = list->head; node;)
        node = list_free_node_(list, node);
    list->head = NULL;
    list->tail = NULL;
    list->length = 0;
}

static list_node_t* list_free_node_(list_t* list, list_node_t* node) {
    CHECK(list != NULL);
    CHECK(node != NULL);

    list_node_t* next = node->next;

    if (list->free_cb) list->free_cb(node->data);
    list->allocator->free(node);
    --list->length;

    return next;
}
```

- Inject a *fake* `list` object
  - → Require controlled **data** at a known **address**

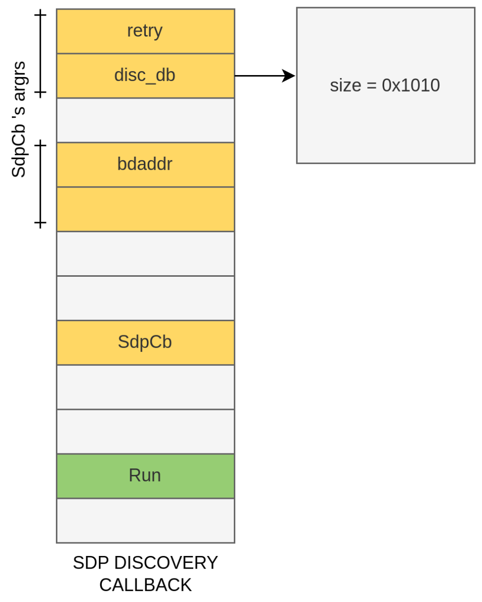
Code Execution on Jemalloc Devices

Code Execution

Controlled Data at Known Address

- Leak the heap pointer of the SDP discovery callback
  - The callback has a reference to a 0x1010 bytes object

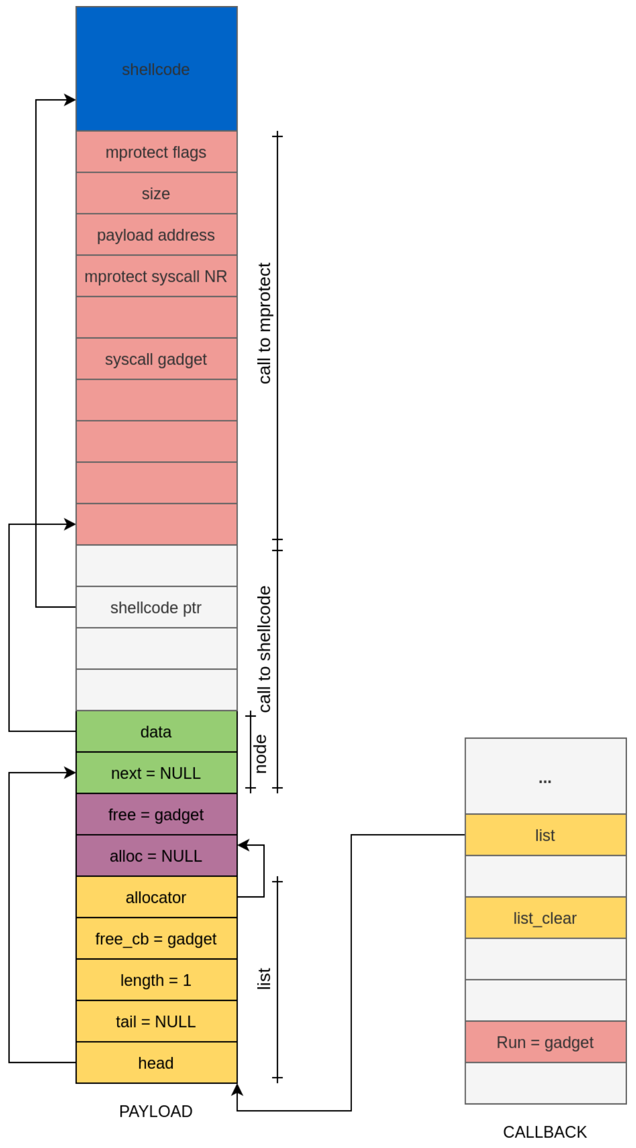
- Spray objects of the same size (with controlled data)
  - → Initiate spray right after the callback allocation



Code Execution on Jemalloc Devices

SYNACKTIV

Code Execution



- gadget → call list_clear(list)

1. list->free_cb(node->data) → gadget → syscall(NR_mprotect)
2. list->allocator->free(node) → gadget → shellcode

Demo

Code Execution on Scudo Devices

Code Execution on Scudo Devices

Scudo Allocator

Overview

- Hardened security allocator
- **Primary allocator**: serves small allocations (< 0x10000 bytes)

Building blocks

- Scudo organizes memory into **regions**
- A region is dedicated to allocations of a specific size class (class id)
- Each region is sandwiched between two guard pages
- A region is divided into memory **blocks**
- A block consists of:
  - 16 bytes of metadata
  - Memory **chunk**: actual memory returned to the program when calling **malloc**

Code Execution on Scudo Devices

Scudo Allocator

SYNACKTIV

Memory allocation

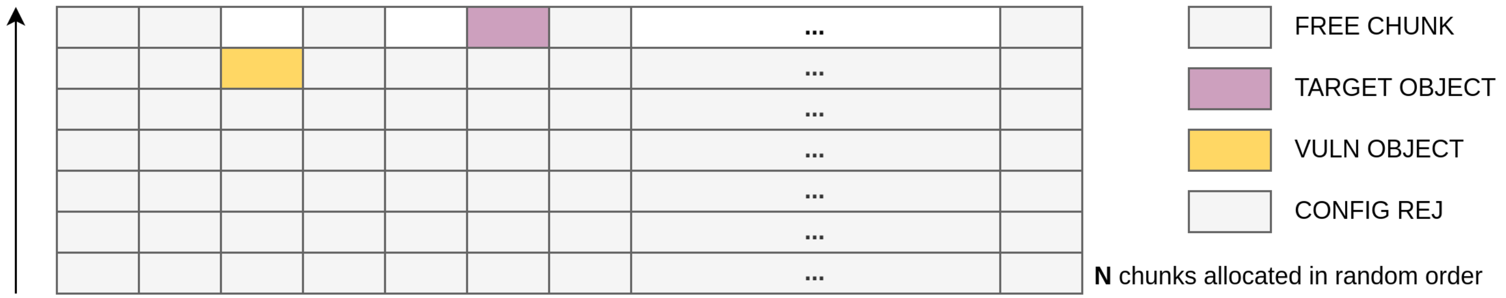
- Pick a chunk from the thread-local cache
- Refill cache if no available chunks from the global freelist
  - Pull a `TransferBatch` (group of pre-allocated chunks)
- Populate the freelist with a group of `TransferBatches` :
  - Allocate memory from region
  - Split memory into individual blocks
  - **Shuffle** memory blocks
  - Group memory blocks into TransferBatches

Code Execution on Scudo Devices

Scudo Allocator - Mitigations

Memory Blocks Shuffling

- Applied per batch of memory blocks rather than the entire region
- Number of randomized blocks depends on the class size
  - **N = 52** (4 * 13) for allocations smaller than 0x350 bytes
- How to make the target object reachable from the vulnerable object during the overflow?
  - → Insert **N** intermediate objects between the vulnerable object and the target object



Code Execution on Scudo Devices

Scudo Allocator - Mitigations

SYNACKTIV

Checksum verification

- Memory chunks prefixed by metadata including a checksum
- Checksum verified when a chunk is freed
- Program aborts if the checksum is corrupted
- → Overflow on freed chunks or on persistent allocations

Code Execution on Scudo Devices

Exploitation Strategy

The Need of a New Exploitation Scenario

⚠ **Memory shuffling issue**

- No relative write primitive
  - Expects the callback at a **fixed** offset

Solution

- Trigger overflow twice!!
  1. Overwrite a `READER` object → Memory Leak
  2. Overwrite a callback object (`EXECUTOR`) → Code Execution
- ... And **survive** to a 64 KB overflow in between

Code Execution on Scudo Devices

Heap Shaping

ERTM	REJ	ERTM	REJ	ERTM	REJ	ERTM	...	REJ
REJ	REJ	REJ	REJ	REJ	REJ	REJ	MORE REJ	REJ
REJ	REJ	REJ	REJ	ERTM	REJ	REJ	MORE REJ	REJ

SOURCE - BEFORE OVERFLOW

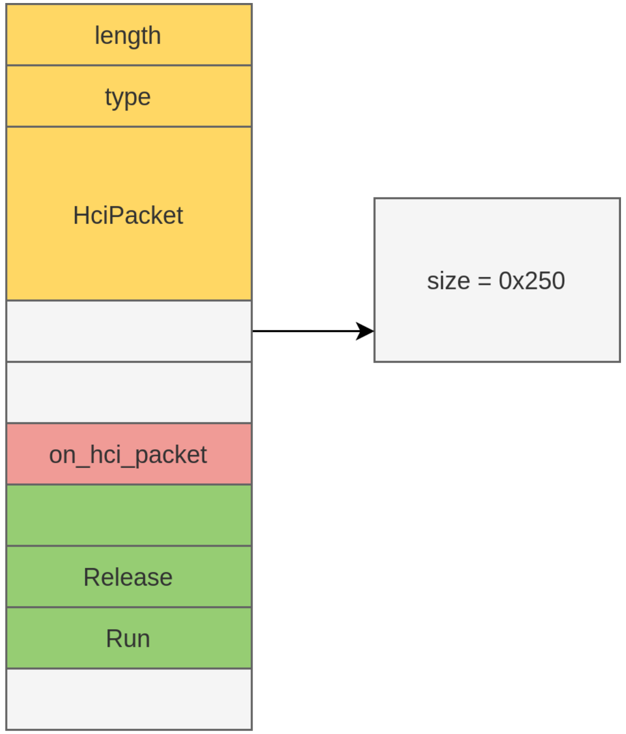
GATT	REJ	GATT	REJ	GATT	REJ	GATT	...	REJ
REJ	REJ	REJ	REJ	REJ	REJ	REJ	MORE REJ	REJ
REJ	REJ	REJ	REJ	GATT	REJ	REJ	MORE REJ	REJ

SOURCE - AFTER OVERFLOW

Code Execution on Scudo Devices

Memory Leak

- The SDP Discovery Callback is rarely present in the leaked heap data
- **However**, a second callback object was consistently observed in the leaked data
- The `Capture Callback` :
  - Log HCI packets
  - Heap reference
  - Multiple function pointers



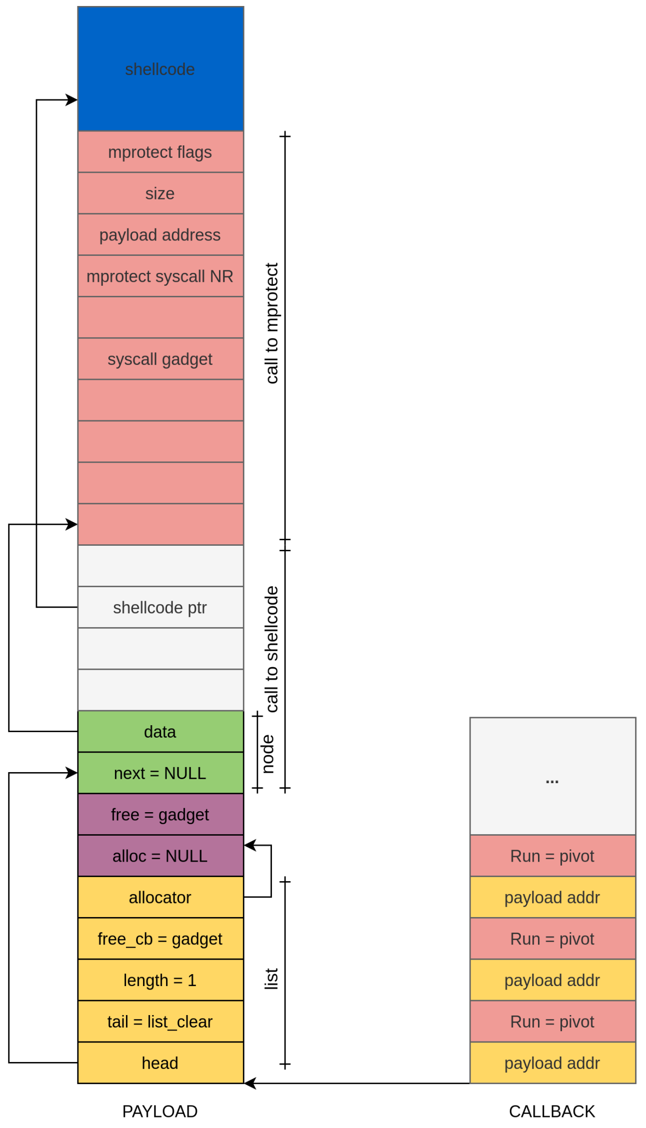
CAPTURE CALLBACK

Code Execution on Scudo Devices

Code Execution

- Corrupt the SDP Discovery Callback
- Memory chunk shuffling makes it hard to rewrite reliably all the fields of the callback object (alignement issue)
- Use a pivot gadget → Require overwriting only 2 specific fields:

```
LDR   X0, [X0]
MOV   W8, W1
MOV   W1, W2
MOV   W2, W8
LDR   X3, [X0,#8]
BR    X3
```



Code Execution on Scudo Devices

Post Exploitation

SYNACKTIV

Shellcode

- Control channel implemented over Bluetooth
  - Can receive & send Bluetooth frames
- Expose a simple command handler
  - Run shell command, upload file, etc.
- Register a signal handler to catch SIGSEGV signals
  - Keep Bluetooth process in a state of clinical death

Conclusion

Conclusion

Conclusion

SYNACKTIV

CVE-2023-40129

- Critical vulnerability in the Bluetooth stack
- No user interaction
- No authentication
- Non-trivial to exploit

2 Exploits

- Remote code execution on Android devices
- Successfully tested on Xiaomi 12T (**Jemalloc**) & Samsung A54 (**Scudo**)

Reliability

- Bluetooth process crashes and silently reboots in case of a failed attempt
- Retry !! (in a loop)
- Estimated Time of Shell (ETS): ~2mn (Jemalloc), ~5mn (Scudo)

Conclusion

SYNACKTIV

The Gabledorsche Stack (GD)

- Introduced in Android 12, default stack in Android 13
- Bluetooth stack rewrite in Rust (work in progress)
- Exploit still functional with GD enabled
  - Only low-level layers have been rewritten as of late 2023

References

References

References

- BlueBorne
  - Ben Seri, Gregory Vishnepolsky (Armis Labs)
- 0-click RCE on the IVI component: Pwn2Own Automotive Edition
  - Mikhail Evdokimov (PCAutomotive) - Hexacon'24
- Fighting Cavities: Securing Android Bluetooth by Red Teaming
  - Jeong Wook Oh, Rishika Hooda and Xuan Xing (Google) - OffensiveCon'25

Acknowledgement

- Kevin Denis aka 0xmitsurugi

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com