Robin Hood vs Cisco ASA Anyconnect

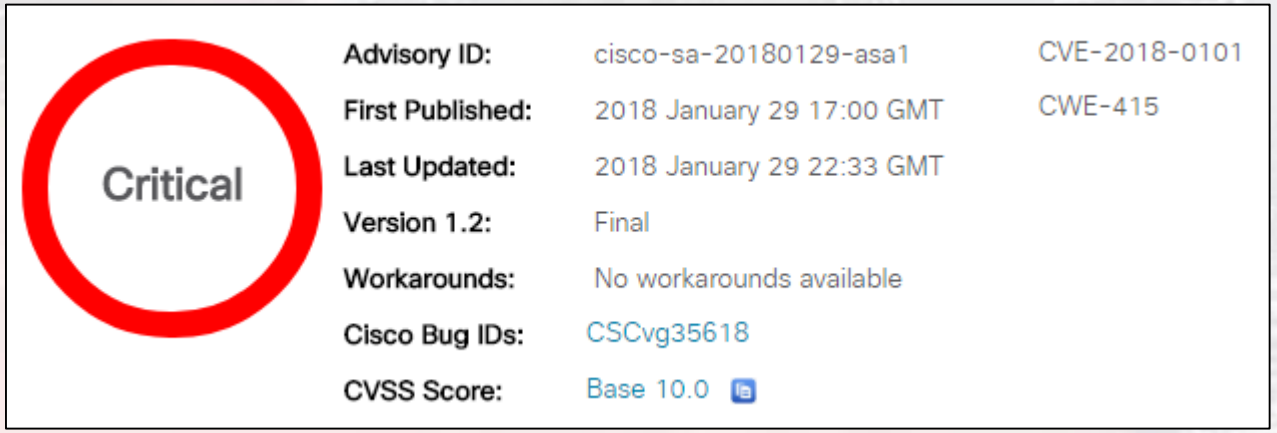
Recon Brussels – February 2018

Speaker

- Cedric Halbronn (@saidelike)
- Previously worked at Sogeti ESEC Lab
- Currently in Exploit Development Group (EDG) at NCC Group
  - Vulnerability research
  - Reverse engineering
  - Exploit development

Agenda

- Find a pre-auth 0-day in a Cisco ASA firewall
- Prove Remote Code Execution
- How to protect against 0-day?



https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180129-asa1

Cisco ASA firewalls

- Entry point to most enterprises
- ASA != IOS
  - ASA = Linux + a single “lina” binary / x86 or x86_64
    - dlmalloc or ptmalloc heap allocator [1]
  - IOS = proprietary operating system / PowerPC

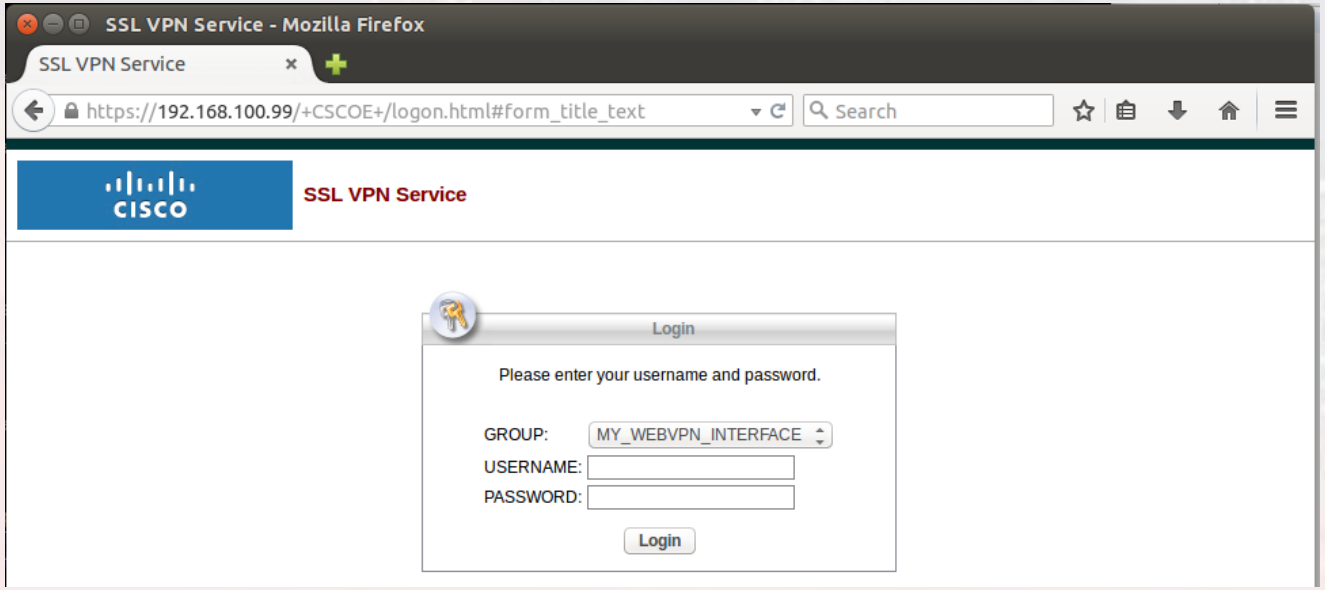
[1] https://github.com/nccgroup/asafw/blob/master/README.md#mitigation-summary

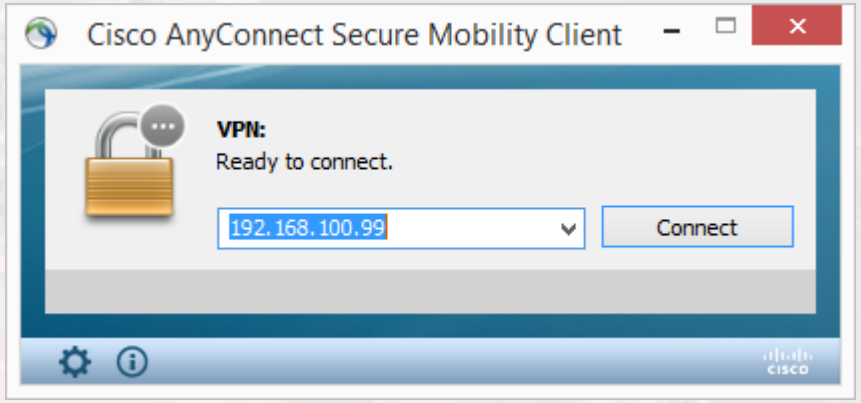
SSL VPN

- WebVPN: client-less (browser)
- AnyConnect: client on Windows, OS X, Linux, Android, iPhone OS

SSL

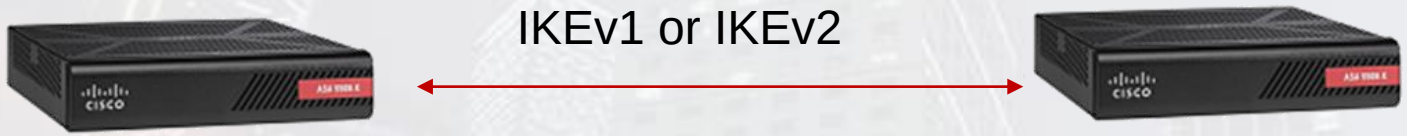
SSL





IKE VPN

- A.k.a. IPSec
- Typically static point-to-point VPNs



- Also supported by native Windows client or even AnyConnect?

Source: https://www.cisco.com/c/en/us/support/docs/security-vpn/webvpn-ssl-vpn/119208-config-asa-00.html#anc17

Previous work

- 2014
  - Various WebVPN ASA version leaks (Alec Stuart)
- 2016
  - CVE-2016-1287: heap overflow in IKE Cisco fragmentation (Exodus Intel)
  - CVE-2016-6366: SNMP OID stack overflow (Shadow Brokers)
- 2017
  - Cisco ASA series on NCC blog in 8-parts (so far ☺)

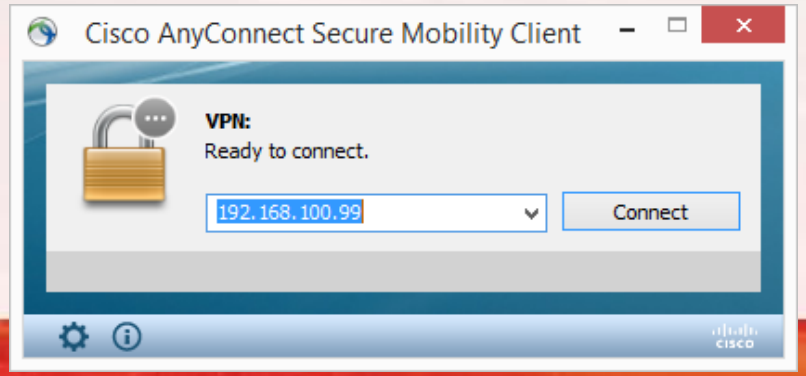
https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/september/cisco-asa-series-part-one-intro-to-the-cisco-asa/

https://github.com/nccgroup/asatools

Vulnerability & feng-shui overview

IKEv1

WebVPN/AnyConnect
SSL



The bigger the worse?

- What license to buy?

50 IKE sessions

250 IKE sessions

750 IKE sessions

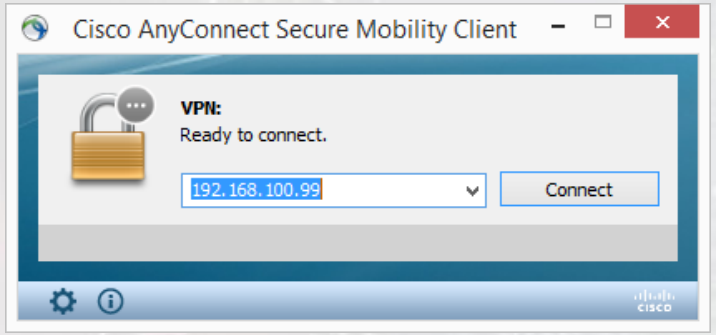
5000 IKE sessions

- An IKE session limits the quantity of data sent as IKE fragments to 0x8000 bytes
- More sessions ➔ more feng shui
- Exploit is more reliable against expensive Cisco hardware and license
- Possible to rob from the rich and give to the poor
- So I named my ~~vulnerability~~exploit: Robin Hood

Source: https://www.cisco.com/c/en/us/td/docs/security/asa/asa97/configuration/vpn/asa-97-vpn-config/vpn-ike.html#ID-2441-00000058

Finding a bug

Sniffing SSL AnyConnect



Burp (or similar)

- First message sent by AnyConnect client

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<config-auth client="vpn" type="init">
<version who="vpn">4.1.06020</version>
<device-id>win</device-id>
<group-select>EURO_RA</group-select>
<group-access>https://192.168.100.96</group-access>
</config-auth>
```

Supported XML tags

Reverse engineering

```
<?xml version="1.0" encoding="UTF-8"?>
<config-auth client="vpn" type="init">
<version who="vpn">4.1.06020</version>
<device-id>win</device-id>
<group-select>EURO_RA</group-select>
<group-access>https://192.168.100.96</group-access>
</config-auth>
```

```
; struct tag_desc xml_tags[27]
tag_desc <0, 0, 0, 0, 0, offset qword_555559E52ED0, 0, 0, 0, 0, 0, 0, \
                         ; DATA XREF: aggregateAuthStartHandler+3F↑o
                         ; aggregateAuthStartHandler+72↑o ...
         0>
tag_desc <offset aConfigAuth, 1, offset dword_555559E52F68, 3, 0, \ ; "config-auth"
         offset dword_555559E52F20, 0, 0, \
         offset tag_handler_config_auth, 0, 0, 0, 0>
tag_desc <offset aVersion, 2, offset dword_555559E52F60, 4, 0, 0, 0, \ ; "version"
         0, offset tag_handler_version, 0, 0, 0, 0>
tag_desc <offset aAutoUpdateDevi+0Ch, 3, offset qword_555559E52EC0, 0,\ ; "device-id"
         0, 0, 0, 0, offset tag_handler_device_id, 0, 0, 0, 0>
tag_desc <offset aPhoneId, 4, 0, 0, 0, 0, 0, 0, \ ; "phone-id"
         offset tag_handler_phone_id, 0, 0, 0, 0>
tag_desc <offset aGroupSelect, 5, 0, 0, 0, 0, 0, 0, \ ; "group-select"
         offset tag_handler_group_select, 0, 0, 0, 0>
tag_desc <offset aSessionToken, 6, 0, 0, 0, 0, 0, 0, \ ; "session-token"
         offset tag_handler_session_token, 0, 0, 0, 0>
tag_desc <offset aSessionId, 7, 0, 0, 0, 0, 0, 0, \ ; "session-id"
         offset tag_handler_session_id, 0, 0, 0, 0>
tag_desc <offset aOpaque, 8, offset dword_555559E52F58, 8, 0, \ ; "opaque"
         offset qword_555559E52EE0, 0, 0, offset tag_handler_opaque, \
         0, 0, 0, 0>
tag_desc <offset aAuth, 9, 0, 0, 0, offset qword_555559E52F00, 0, 0, \ ; "auth"
         offset tag_handler_auth, 0, 0, 0, 0>
tag_desc <offset aUsername_0, 0Ah, 0, 0, 0, 0, 0, 0, \ ; "username"
         offset tag_handler_username, 0, 0, 0, 0>
```

- Initial sample contains all supported tags

  → Input mutation fuzzing

Fuzzing architecture

- Spray/pray/prey ☺

Mutated XML packet (radamsa)

Ping (still alive?)

NO → save packet

https://github.com/aoh/radamsa

- Speed: 1 test / few seconds…
- No gdb attached, is that not slow enough?

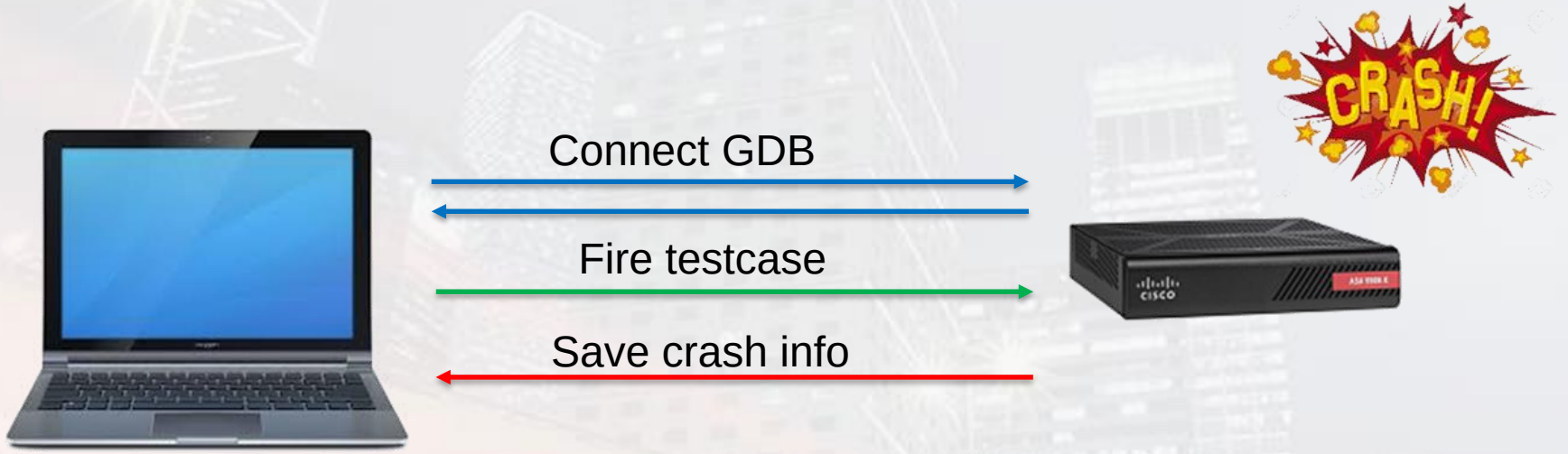
The wall is on fire…

- Want to start fuzzing before going on leave…
- ASA firewall keeps crashing

Understanding the bug

Triage

- asadbg-assisted
  - https://github.com/nccgroup/asadbg

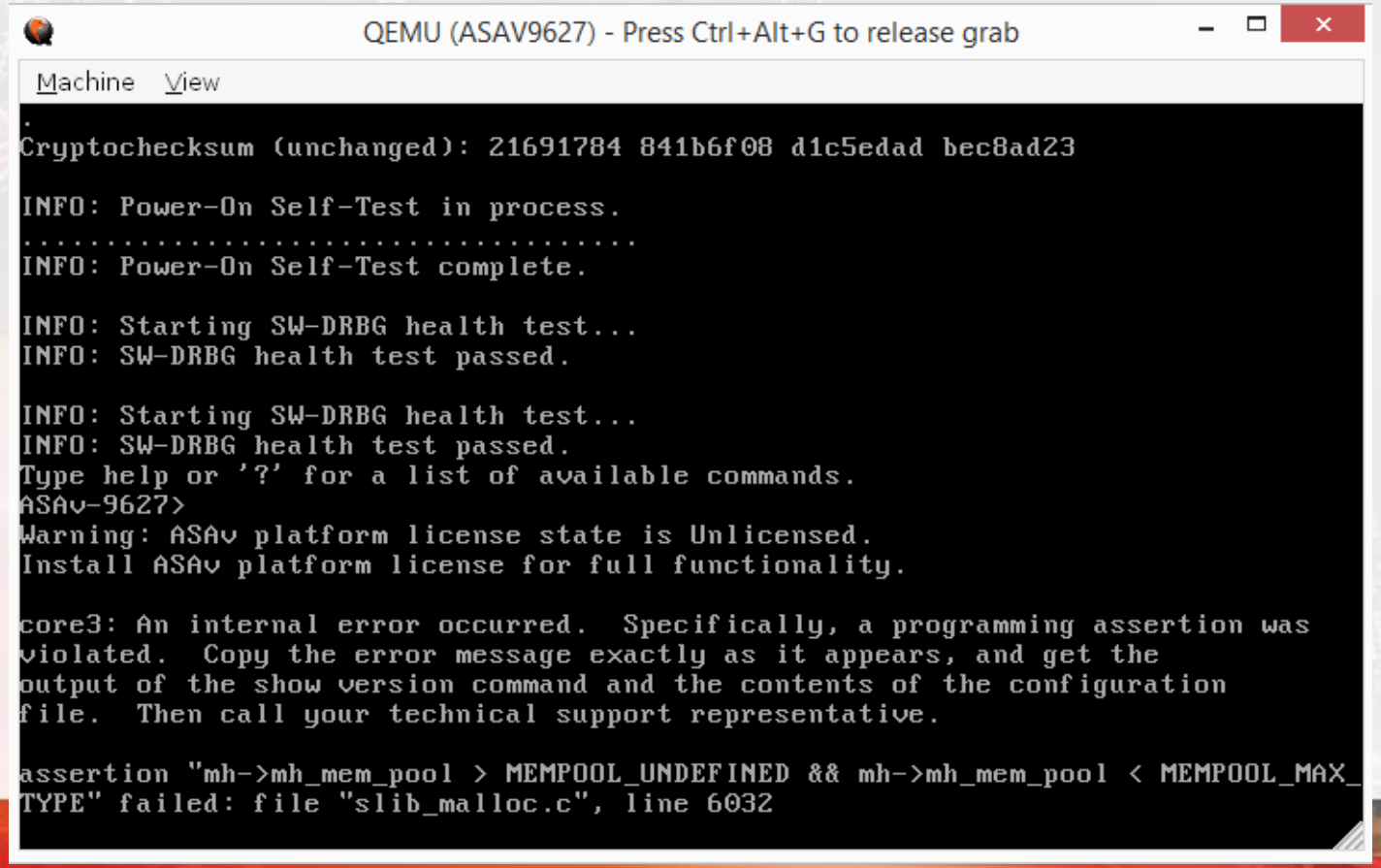


Replay with gdb script

```
# will be called next time it stops. Should be when it crashes
# so we log stuff
define hook-stop
    set logging file %CRASH_LOG_FILE%
    set logging on
    set logging redirect on
    set logging overwrite on
    sync
    bbt
    i r
    set logging off
    set logging redirect off
end

continue

# below will be executed after it breaks because of a crash
# and this allows us to exit gdb
detach
quit
```

One crash to rule them all

- All the same crash
- Both ASAv 64-bit / ASA 32-bit



The smaller the better

- Fits in a tweet

```
<?xml version="1.0" encoding="UTF-8"?>
<config-auth client="a" type="a" aggregate-auth-version="a">
    <host-scan-reply>A</host-scan-reply>
</config-auth>
```

AnyConnect Host Scan: https://www.cisco.com/c/en/us/td/docs/security/asa/asa84/configuration/guide/asa_84_cli_config/vpn_hostscan.html

Back to the trace

- What is it?
  - Crash in free()
  - Invalid heap metadata?
  - Heap overflow?
  - UAF?
  - Double free?
  - Other?

- Interesting functions
  - *auth_process_client*
  - *FreeParser*

```
(gdb) bt
#0  0x00007ffff7496afd in pause ()
#1  0x0000555557a3af65 in int3 ()
#2  0x00005555587444fa in __lina_assert ()
#3  0x0000555556307e0e in ?? ()
#4  0x00005555587758a9 in mem_get_pool_type ()
#5  0x0000555557b16225 in resMgrFree ()
#6  0x0000555557a47970 in free ()
#7  0x0000555556634a003 in aggregateAuthFreeParserDataOutMem ()
#8  0x00005555583e8c2d in lua_aggregate_auth_process_client_request ()
#9  0x0000555557eefb9b in luaD_precall ()
#10 0x0000555557eff9b8 in luaV_execute ()
#11 0x0000555557ef0630 in luaD_call ()
#12 0x0000555557eef63a in luaD_rawrunprotected ()
#13 0x0000555557ef0a83 in luaD_pcall ()
#14 0x0000555557ee9546 in lua_pcall ()
#15 0x0000555557f03f81 in lua_dofile ()
#16 0x0000555558223beb in aware_run_lua_script_ns ()
#17 0x0000555557dca59d in ak47_new_stack_call ()
#18 0x0000555558228c48 in aware_serve_request ()
#19 0x00005555822b5f4 in ?? ()
#20 0x000055555822bc0f in run_aware_fiber ()
#21 0x0000555557da2c75 in _fiber_jumpstart ()
#22 0x0000555557da2d98 in _fiber_setup_for_jumpstart ()
```

2 days reversing later…

- aggregateAuthParseBuf
  - Receive the XML / initialize the libexpat parser
- Cisco-specific callbacks registered
  - aggregateAuthStartHandler: called when XML tag opened
  - aggregateAuthDataHandler: called when XML data parsed
  - aggregateAuthEndHandler: called when XML tag closed

```
<?xml version="1.0" encoding="UTF-8"?>
<config-auth client="a" type="a" aggregate-auth-version="a">
    <host-scan-reply>A</host-scan-reply>
</config-auth>
```

```
void aggregateAuthDataHandler(struct userData *userData, const XML_Char *data, int len)
{
    // initialize pData to heap or global address
    if (userData->tag_idx; == HOST_SCAN_REPLY) {
        pData = xml_tags[HOST_SCAN_REPLY].alloc;  // [1]
        remaining_len = 8191;
    } else {
        remaining_len = 511;
        pData = &xml_tags[tag_idx].data;
    }
    ...
    // current buffer holds anything?
    if (!pData || pData[0] == '\0') {           // [2]
        prev_len_data = 0;
    } else {
        prev_len_data = strlen(pData);
        remaining_len -= prev_len_data;
    }
    ...
    // if there was data in the buffer already, assume it was allocated
    // just append data at the end and exit! It does not reallocate anything!
    if (prev_len_data) {
        strncat(pData, data, len);
        return;                                  // [3]
    }

    // if no data was in the buffer already
    if (userData->tag_idx == HOST_SCAN_REPLY) {
        pData = (char *)malloc(0x2000);          // [4]
        xml_tags[HOST_SCAN_REPLY].alloc = pData;
    } else {
        pData = xml_tags[userData->tag_idx].data;
    }
    ...
}
```

```
void aggregateAuthFreeParserDataOutMem(...)
{
        ...
    if (xml_tags[HOST_SCAN_REPLY].alloc)
        free(xml_tags[HOST_SCAN_REPLY].alloc);  // [5]
        ...
}
```

Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

Data handler

XML 1

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

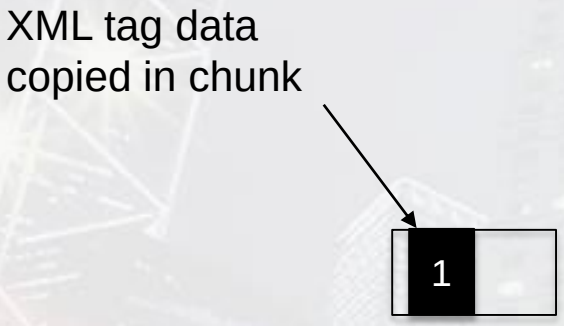
➔ double-free vulnerability on 0x2040-byte chunk

Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

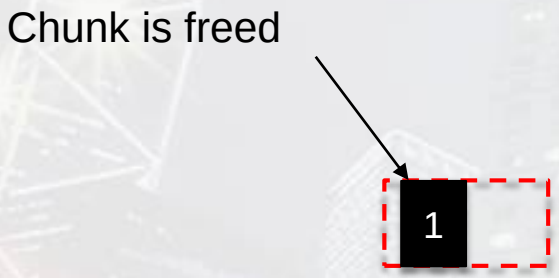
Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

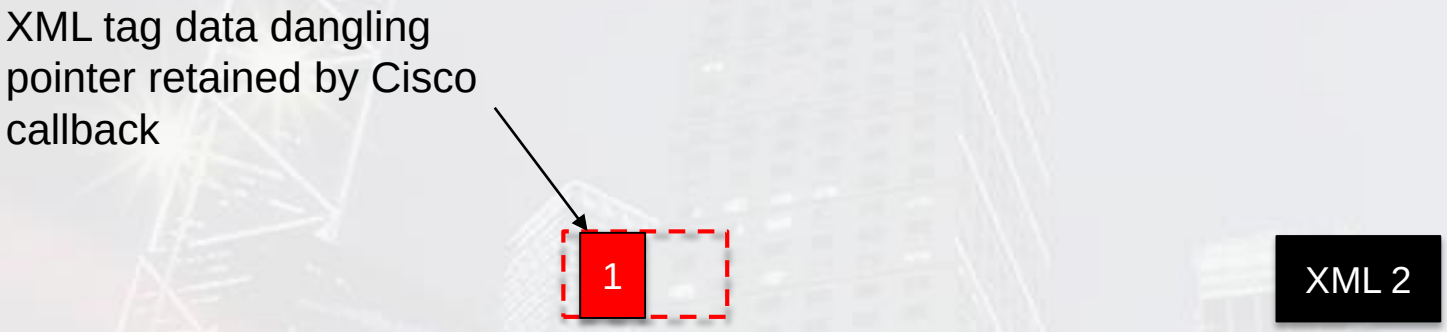
➔ double-free vulnerability on 0x2040-byte chunk

Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

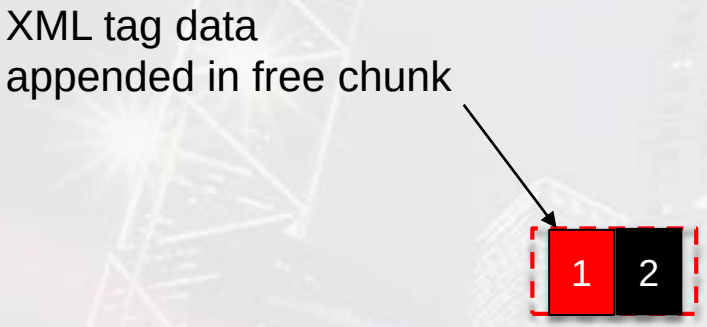
Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

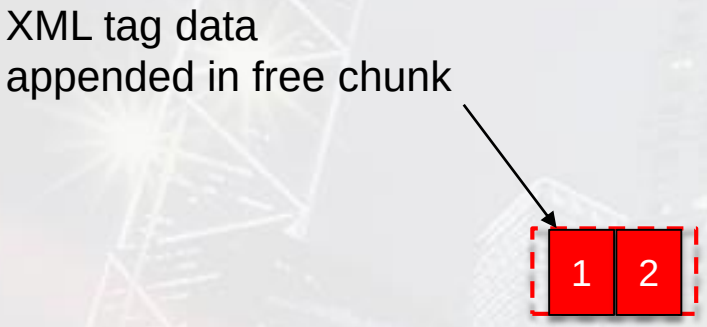
Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

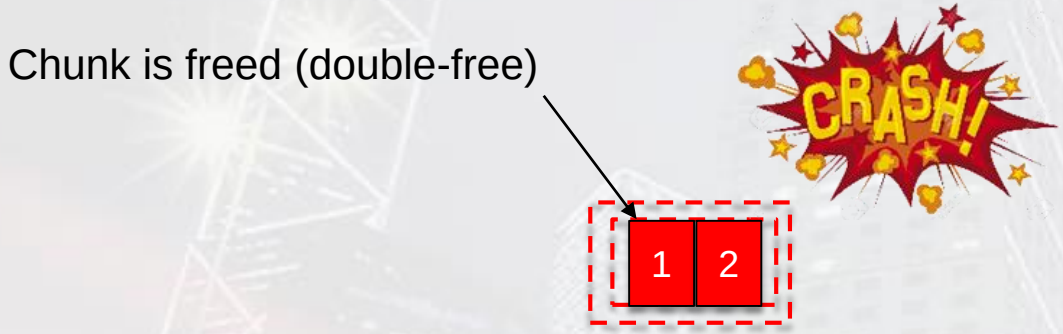
Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

Data handler



- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags' data copied and appended in the same chunk

➔ double-free vulnerability on 0x2040-byte chunk

assert() due to invalid metadata

- Inline metadata/header for heap chunks


```
prev_foot     = 0x8180d4d0
head          = 0x1d0 (CINUSE|PINUSE)
mh_magic      = 0xa11c0123
mh_len        = 0x1a4
mh_refcount   = 0x0
mh_unused     = 0x0
mh_fd_link    = 0xacb85b30
mh_bk_link    = 0xa8800604
allocator_pc  = 0x86816b3
free_pc       = 0x868161d
```

Allocated chunk header

Same offset

```
prev_foot     = 0x8180d4d0
head          = 0x30 (PINUSE)
fd            = 0xac825ab8
bk            = 0xa880005c
mh_refcount   = 0xf3ee0123
mh_unused     = 0x0
mh_fd_link    = 0x0
mh_bk_link    = 0x0
allocator_pc  = 0x0
free_pc       = 0x0
```

Free chunk header

- Hence why our fuzzer caught it!

```
; __int64 __fastcall mem_get_pool_type(void *mem)
                public mem_get_pool_type
mem_get_pool_type proc near
; __unwind {
                push    rbp
                mov     rbp, rsp
                push    r13
                push    r12
                xor     r12d, r12d
                push    rbx
                mov     rbx, rdi ; mem = pointer to data returned to user
                ;       (after ptmalloc and mp headers)
                sub     rsp, 8
                test    rdi, rdi
                jz      short loc_555558775403
```

```
                call    ms_overhead
                sub     rbx, rax ; rax = 0
                mov     r12, rbx
```

```
loc_555558775403:       ; mh_refcount = 1 generally
                movzx   eax, word ptr [r12-26h]
                lea     r13, [r12-30h] ; mp_header*
                mov     ecx, [r12-2Ch] ; mh_len
                lea     edx, [rax-1] ; mh_refcount--
                cmp     dx, 11 ; error if mh_refcount more than 11
                ja      loc_5555587758A4
```

```
cmp     dword ptr [r12-30h], 0A11C0123h
jz      loc_555558775748
```

```
loc_5555587758A4:
                call    assert_mh_mem_pool
; } // starts at 5555587753E0
mem_get_pool_type endp
```

Exploiting the bug like RobinHood

Objective: mirror write

- Allocated chunks hold pointers to alloc lists

```
prev_foot    = 0x8180d4d0
head         = 0x1d0 (CINUSE|PINUSE)
mh_magic     = 0xa11c0123
mh_len       = 0x1a4
mh_refcount  = 0x0
mh_unused    = 0x0
mh_fd_link   = 0xacb85b30
mh_bk_link   = 0xa8800604
allocator_pc = 0x86816b3
free_pc      = 0x868161d
```

- Target mempool alloc lists to get a mirror write
  - No safe unlinking on Cisco specific metadata on all ASA versions
  - Even if dlmalloc or ptmalloc had safe unlinking for free chunks
- Mirror write: unlinking an element from a doubly-linked list will trigger two write operations
  - One operation is the useful one, the other is a side effect
  - Constraint: both need to be writable addresses
- Was already abused in 2016 by Exodus Intel

Exploit strategy

Use double free

→ Create confusion state on the heap

→ Overflow some memory

→ Overwrite linked list pointers

→ Trigger mirror write primitives

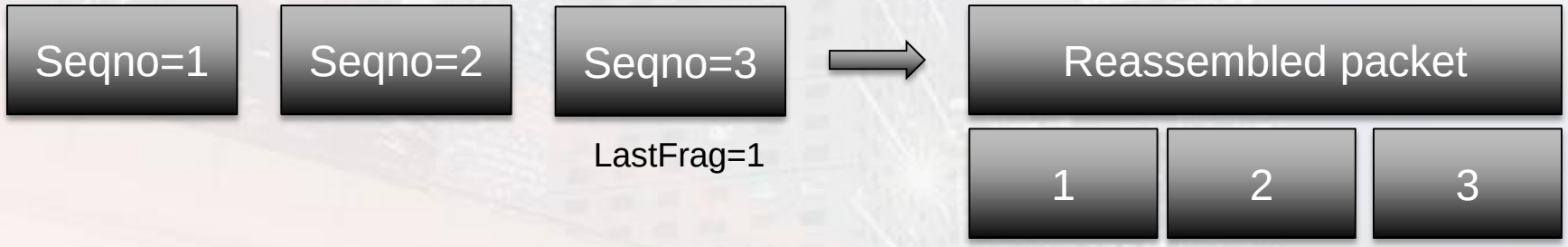
→ Overwrite a function pointer

→ Get RCE

Use what you got

- Leverage what you learnt from CVE-2016-1287 (IKE heap overflow)
  - IKEv1 feng shui is quite reliable
  - IKE fragmentation can be used to overflow memory

- Simple IKE reassembly



https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/november/cisco-asa-series-part-eight-exploiting-the-cve-2016-1287-heap-overflow-over-ikev1/

Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole

Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole

sess1

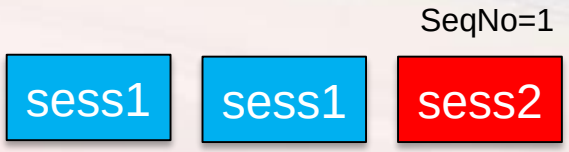
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole

sess1 sess1

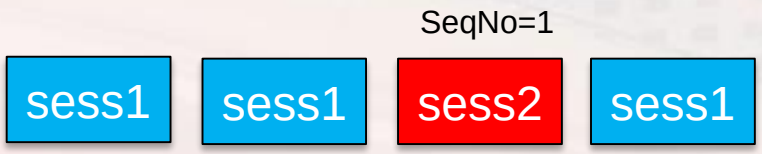
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



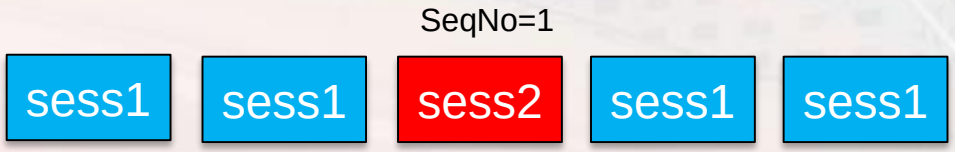
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



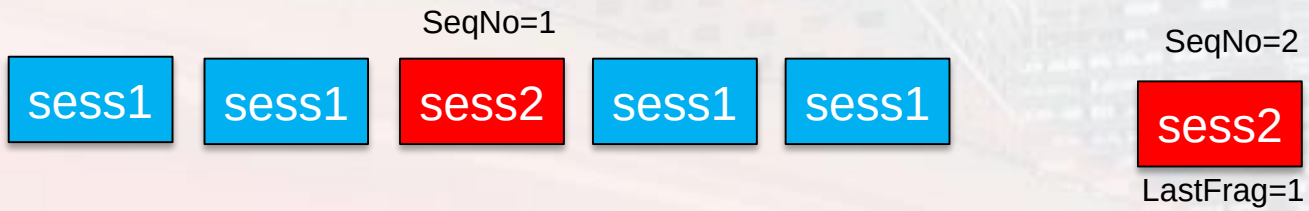
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



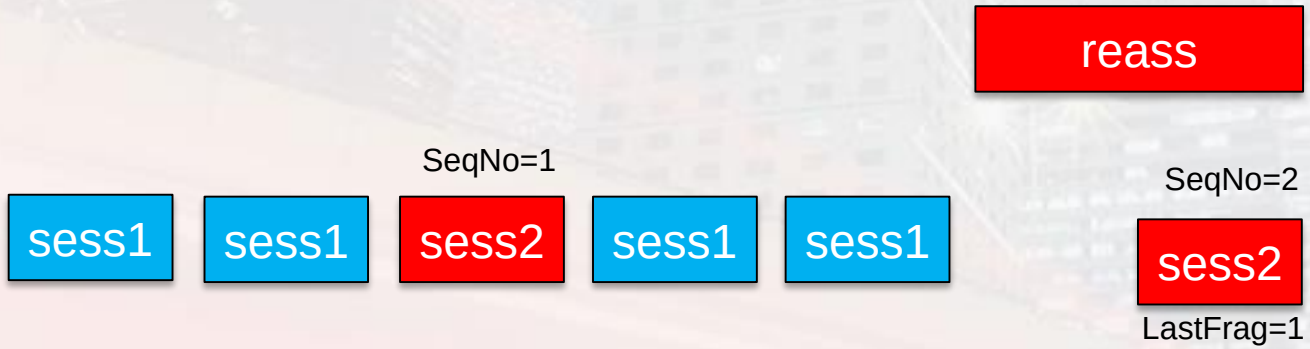
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



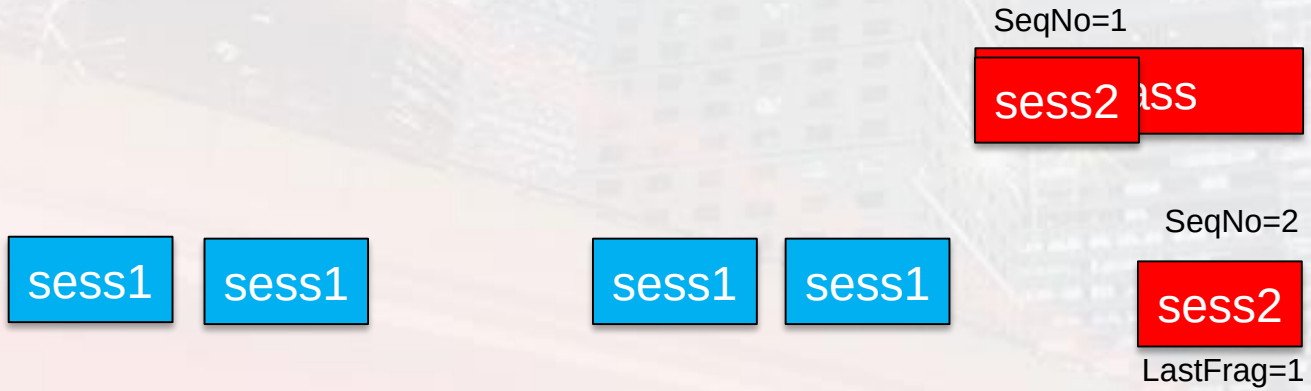
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



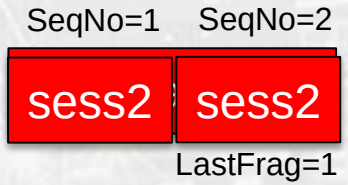
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole



Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

Primitive 2 - Overflow with IKEv1

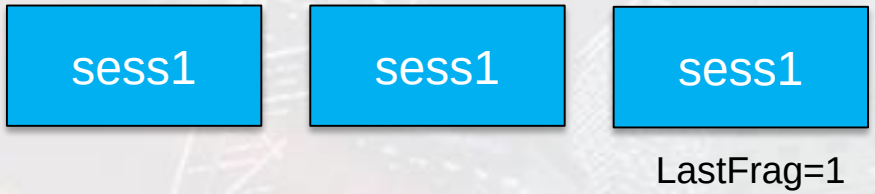
Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

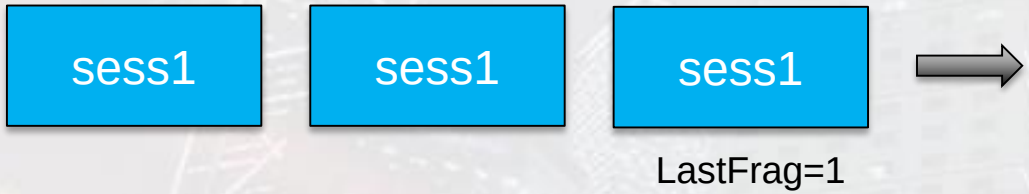
1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

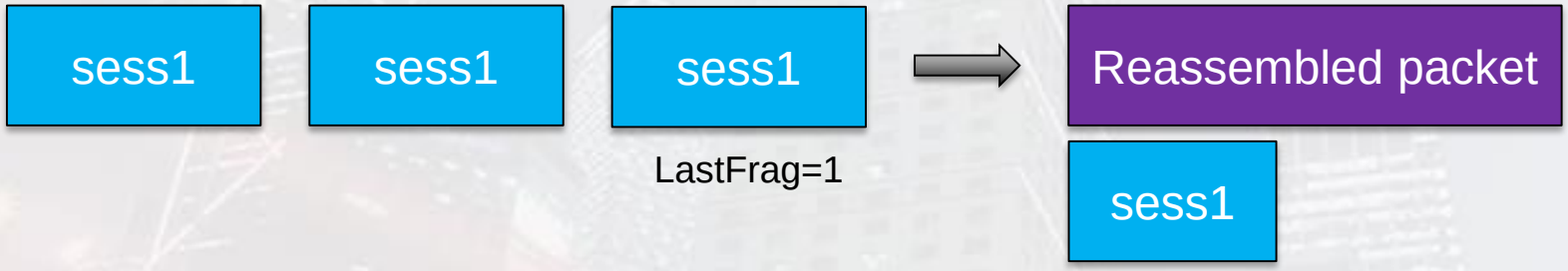
Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

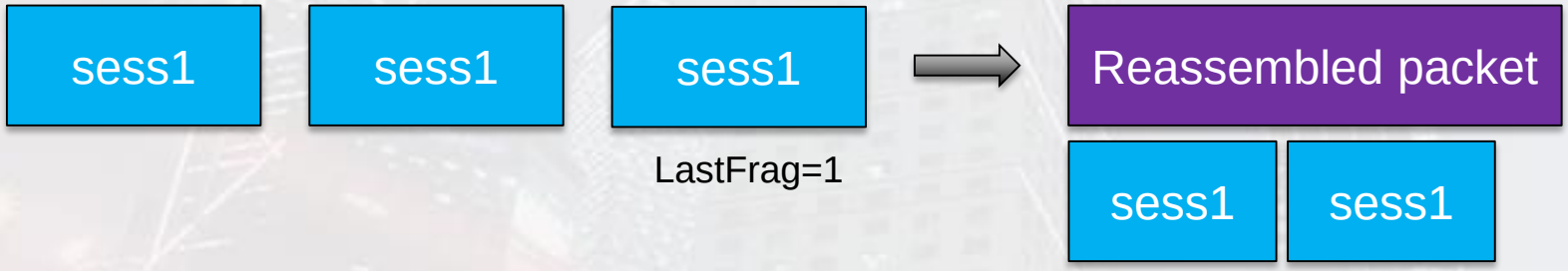
1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

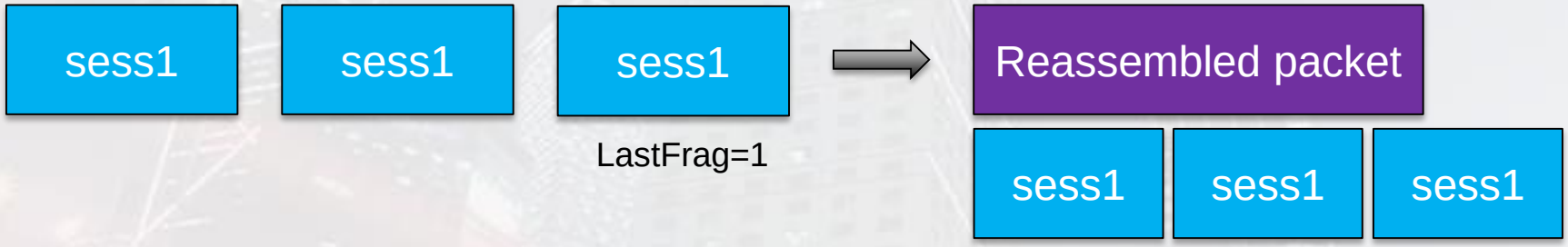
1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

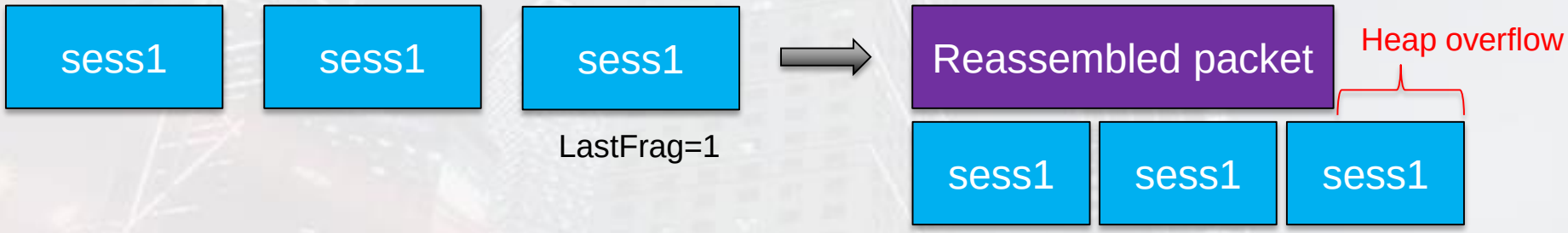
1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

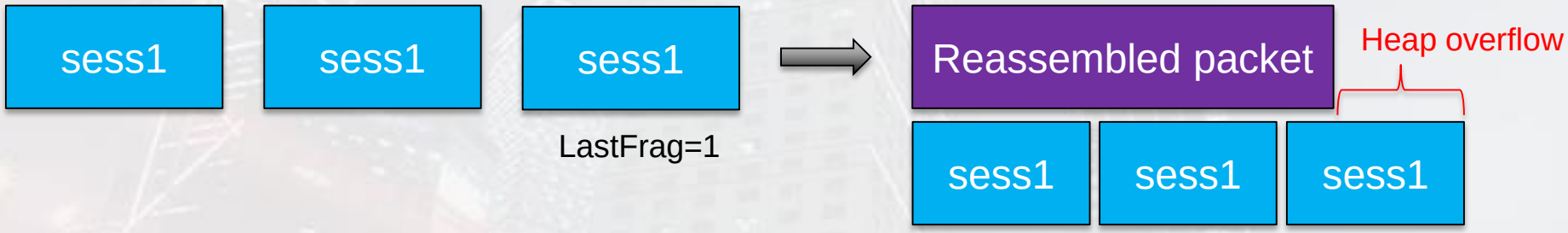
1. Reduce the accumulated length (CVE-2016-1287)



Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

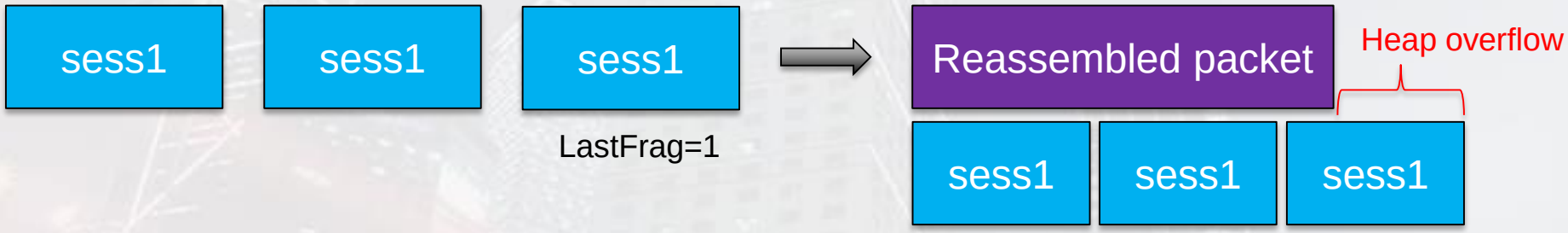


2. Increase fragment length (overflow primitive)

Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)

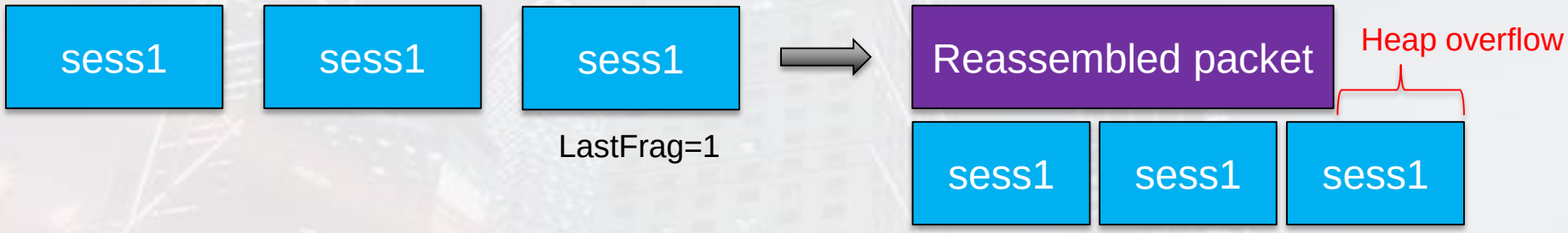


2. Increase fragment length (overflow primitive)

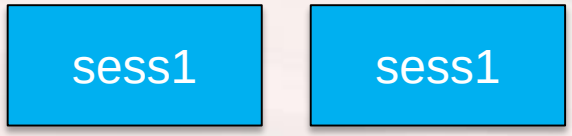
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



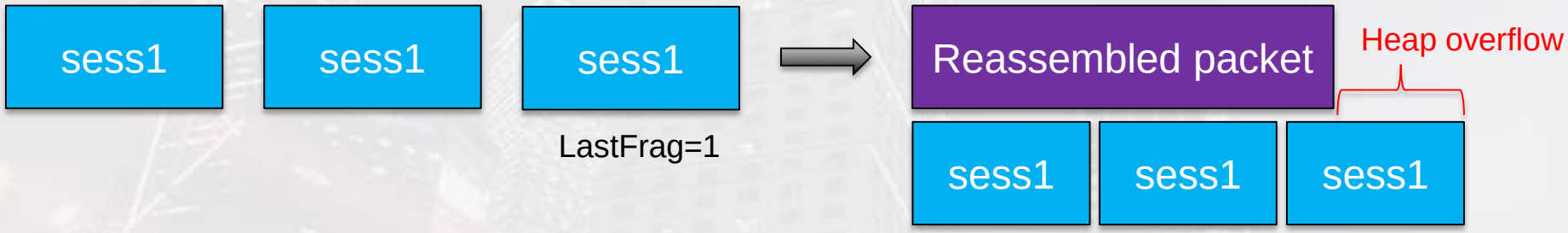
2. Increase fragment length (overflow primitive)



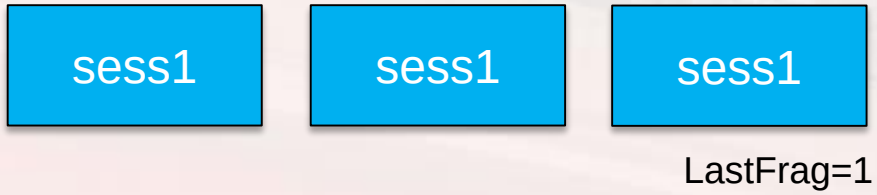
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



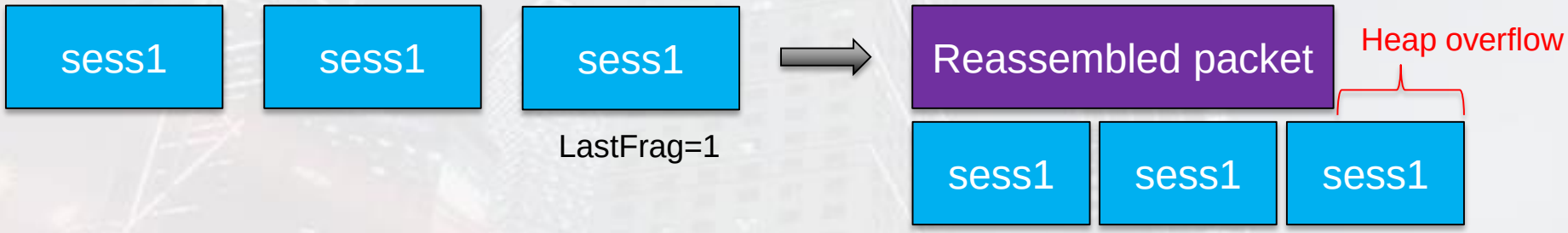
2. Increase fragment length (overflow primitive)



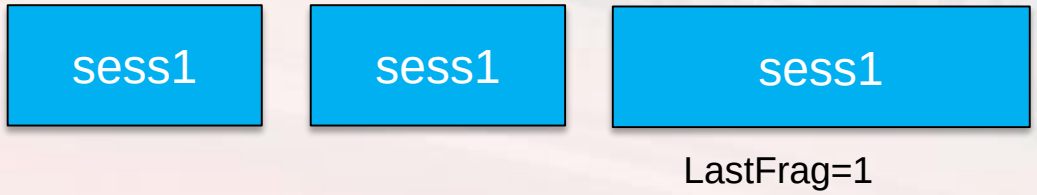
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



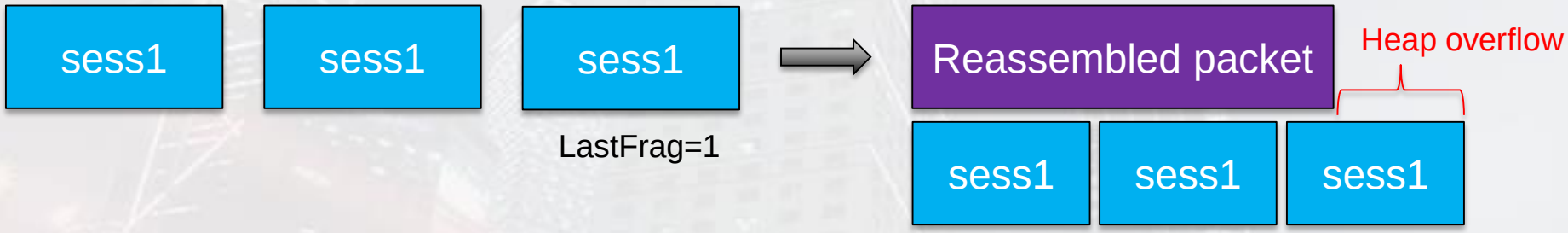
2. Increase fragment length (overflow primitive)



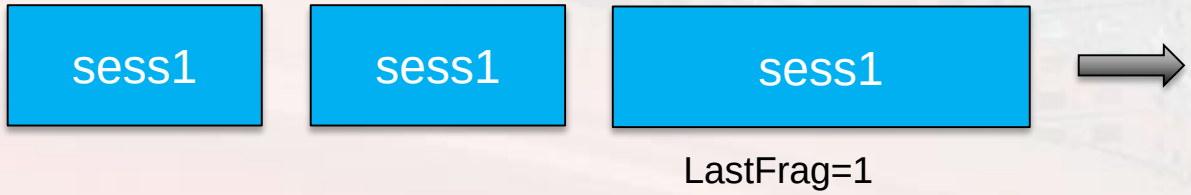
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



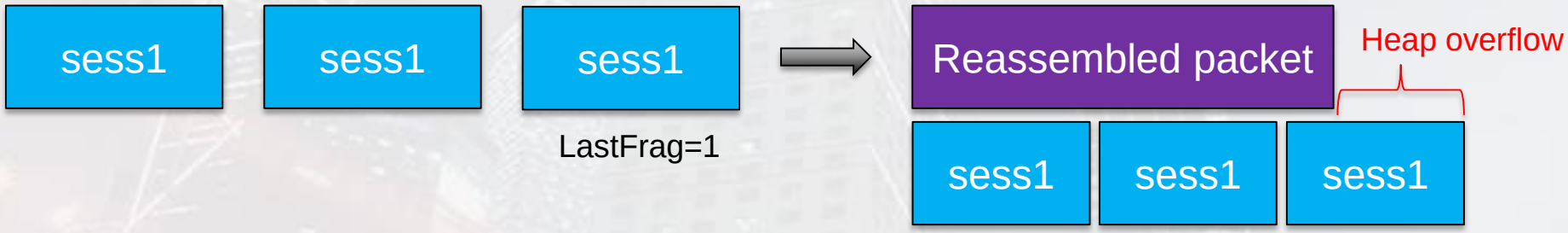
2. Increase fragment length (overflow primitive)



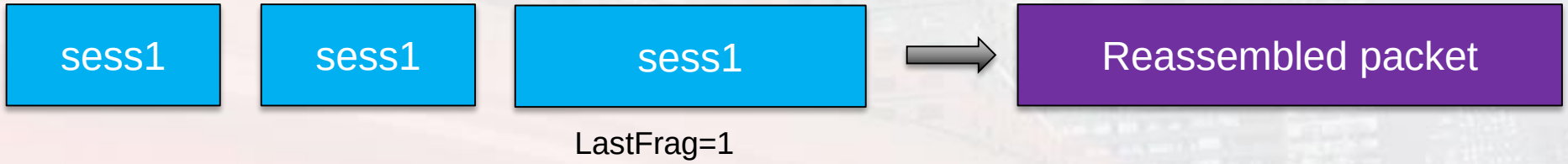
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



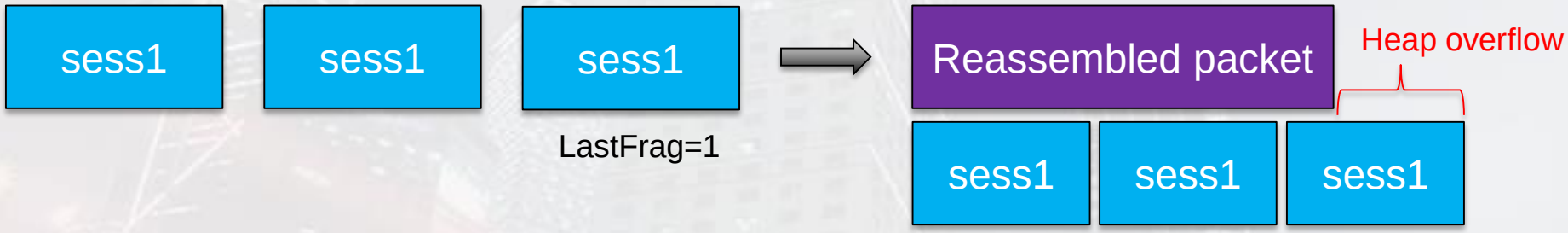
2. Increase fragment length (overflow primitive)



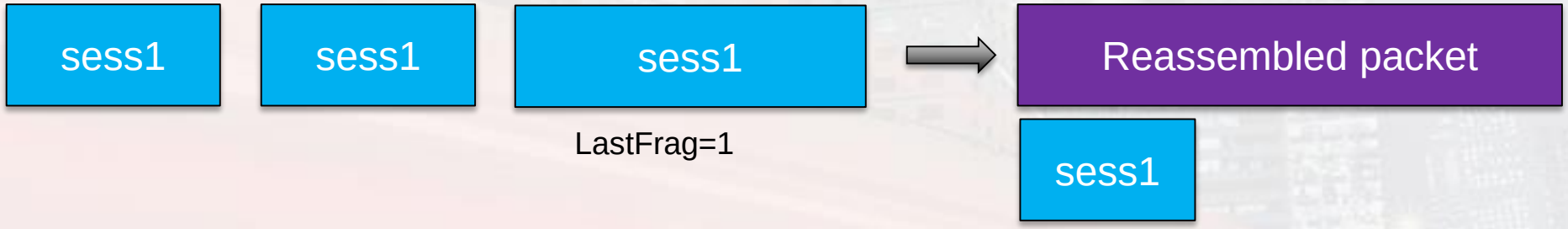
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



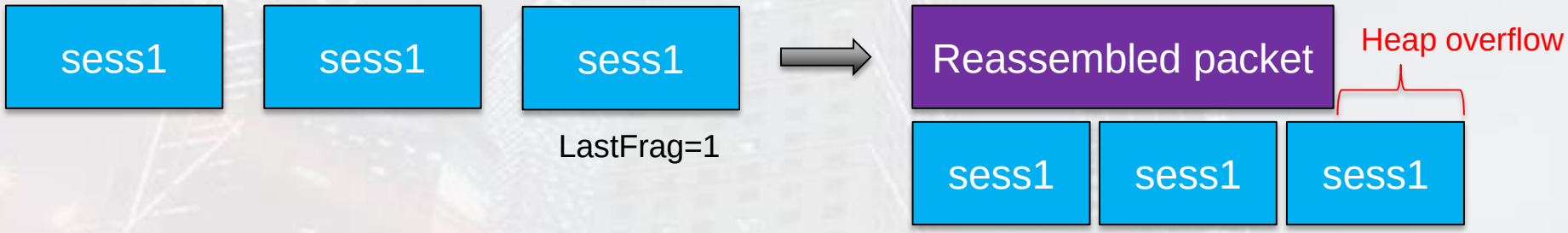
2. Increase fragment length (overflow primitive)



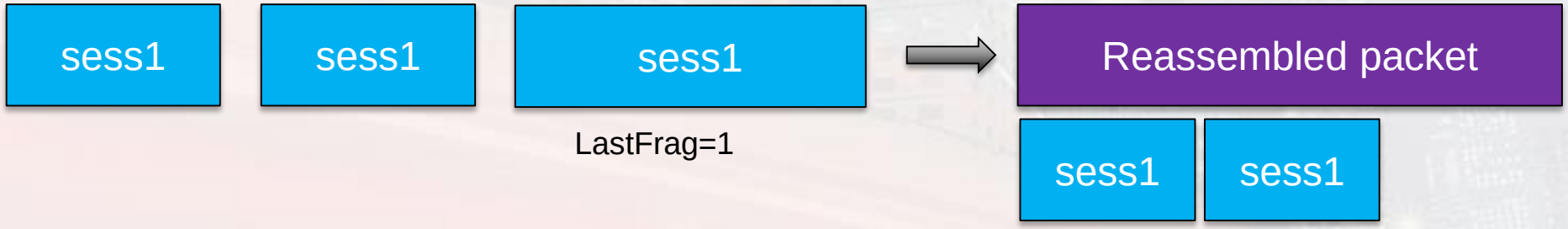
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



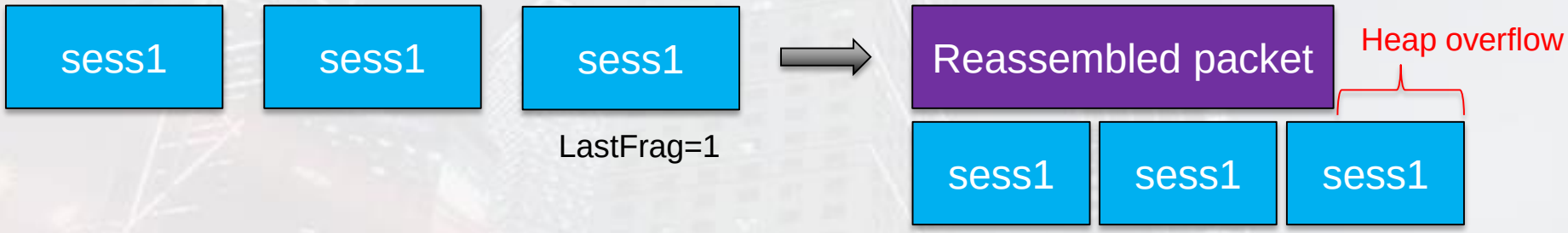
2. Increase fragment length (overflow primitive)



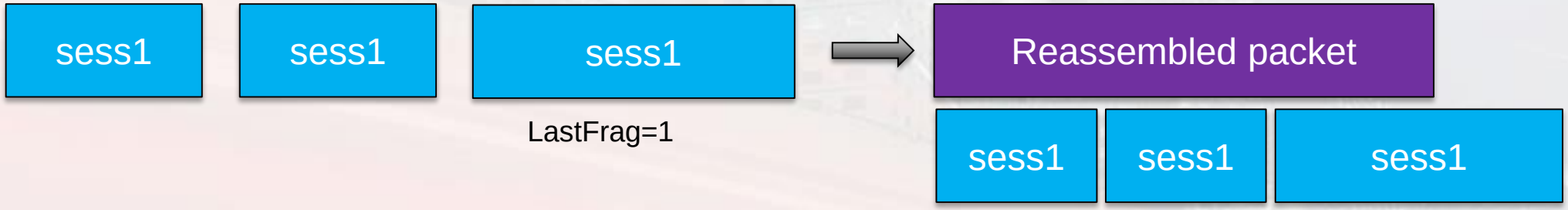
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



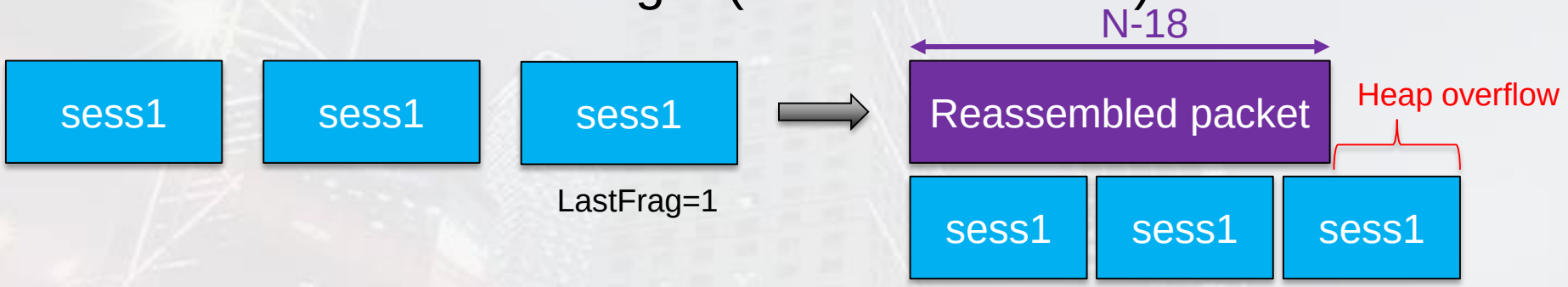
2. Increase fragment length (overflow primitive)



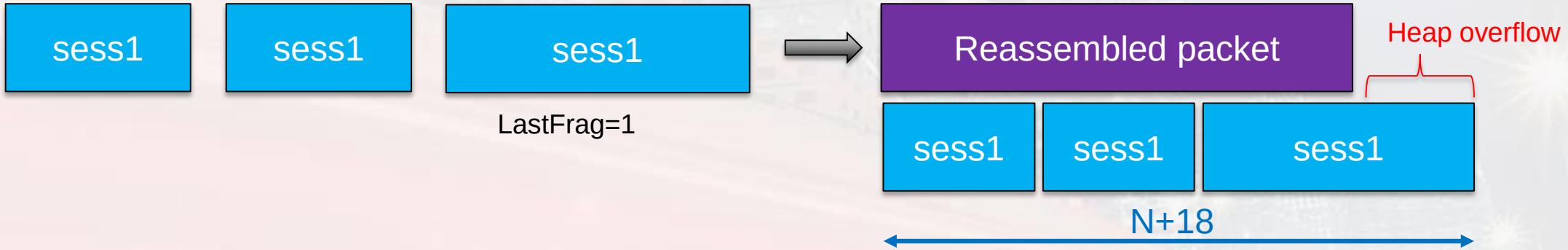
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore

1. Reduce the accumulated length (CVE-2016-1287)



2. Increase fragment length (overflow primitive)



Limited overflow (18-byte on 32-bit)

```
int IKE_GetAssembledPkt(struct ikev1_sa*ikev1_sa)
{
    ...
    // allocate reassembled packet
    int alloc_size = ikev1_sa->frag_queue1->assembled_len + sizeof(struct pkt_buffer);
    struct pkt_buffer* pkt_buffer = malloc(alloc_size);
    pkt_buffer->total_size = ikev1_sa->frag_queue1->assembled_len;

    ...
    // loop on all fragments
    while (TRUE)
    {
        ...
        // update the reassembled packet length
        int curr_frag_len = entry1_found->pkt_info->packet_ike->payload_length - 8;
        curr_reass_len += curr_frag_len;

        // This check is incomplete.
        // Does not take into account sizeof(struct pkt_buffer) added to alloc_size
        if (alloc_size < curr_reass_len) {
            es_PostEvent("Error assmbling fragments! Fragment data longer than packet.");
            ...
            return NULL;
        }
        // Process copying one fragment
        memcpy(&(pkt_buffer->data + curr_reass_len),
                    entry1_found->pkt_info->packet_ike->data,
                    curr_frag_len);
        ...
```

Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 3 – Repeatable free with XML

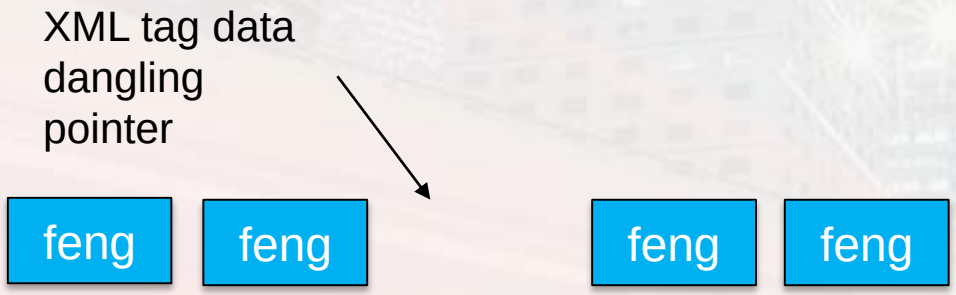
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

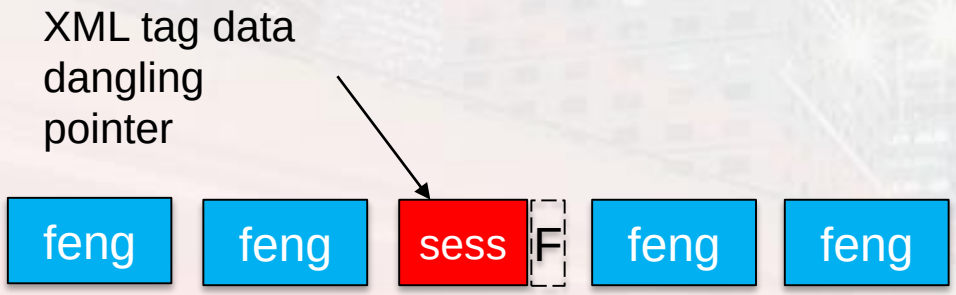
➔ Interesting confusion state



Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

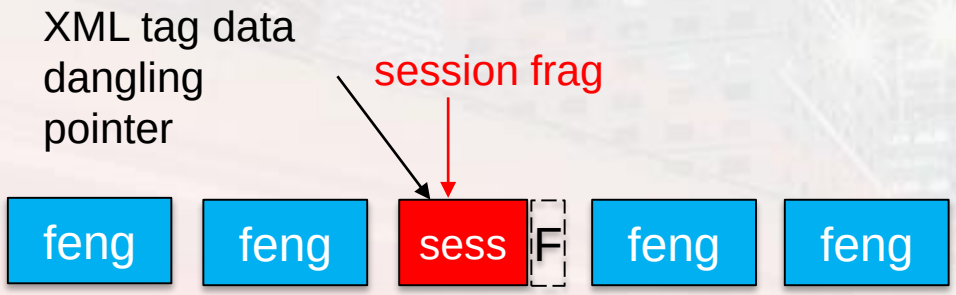
➔ Interesting confusion state



Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state



Primitive 3 – Repeatable free with XML

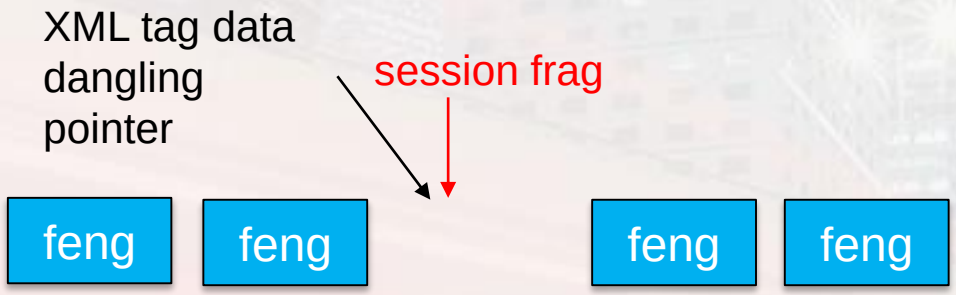
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

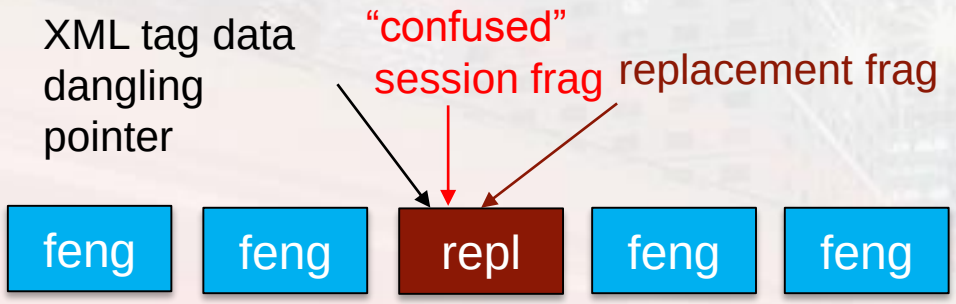
➔ Interesting confusion state



Primitive 3 – Repeatable free with XML

- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the double free primitive
- Allocate another IKEv1 fragment in same hole

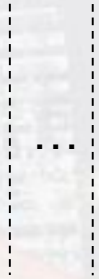
➔ Interesting confusion state



Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

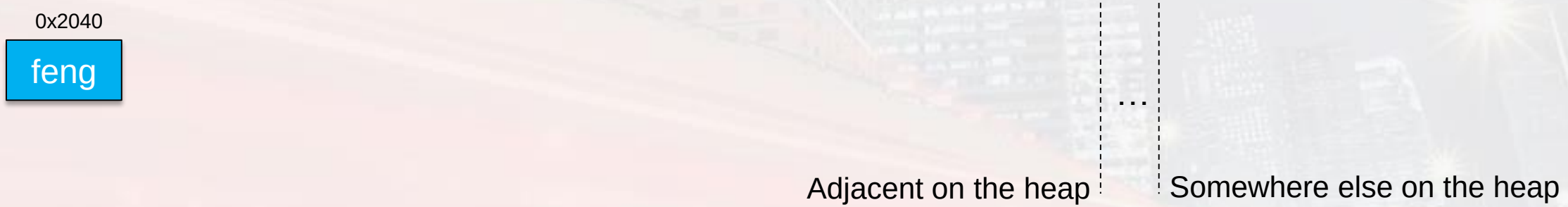


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators

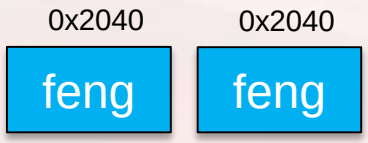


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators

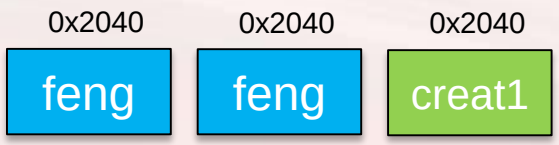


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation

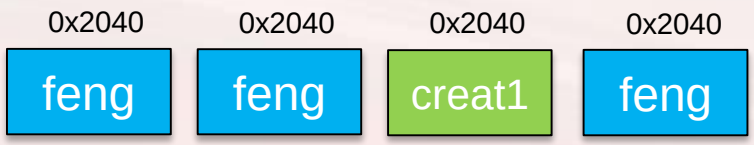


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



…

Adjacent on the heap

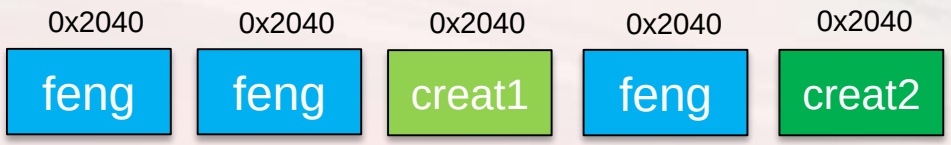
Somewhere else on the heap

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



…

Adjacent on the heap

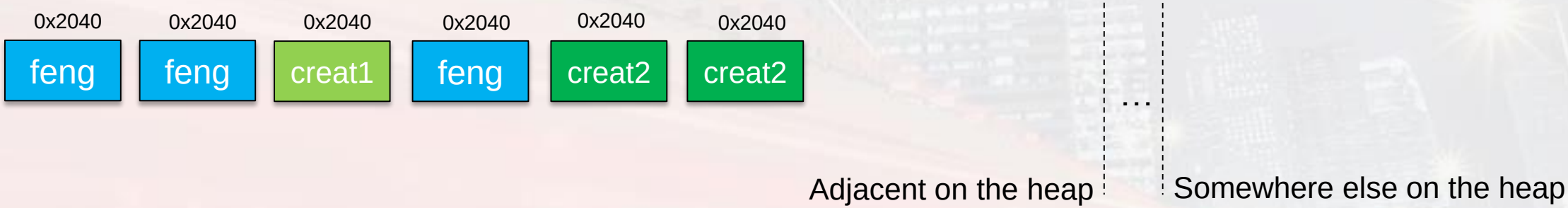
Somewhere else on the heap

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

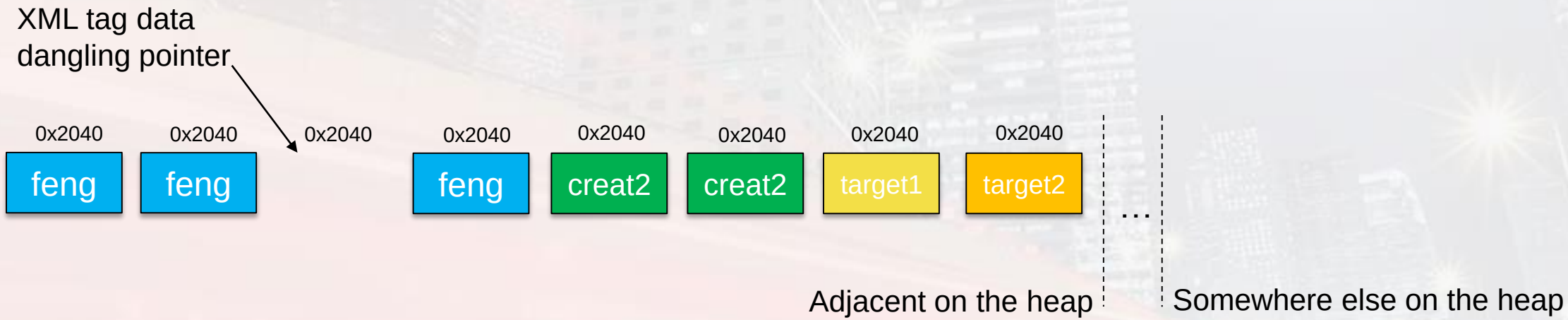
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

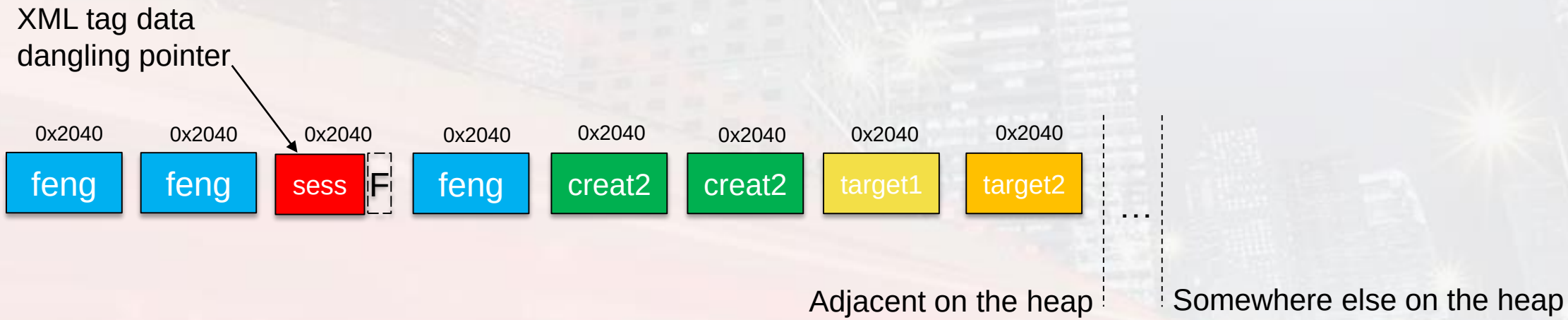


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled

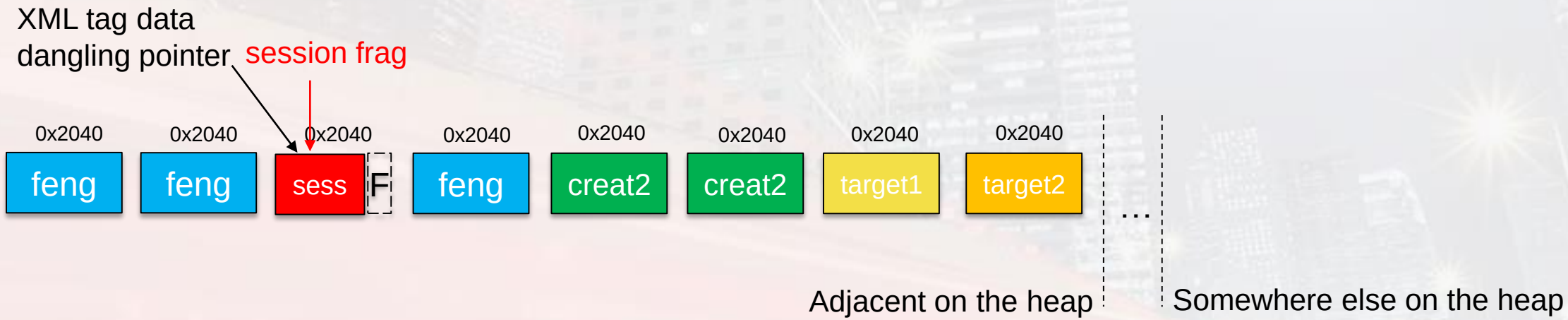


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled

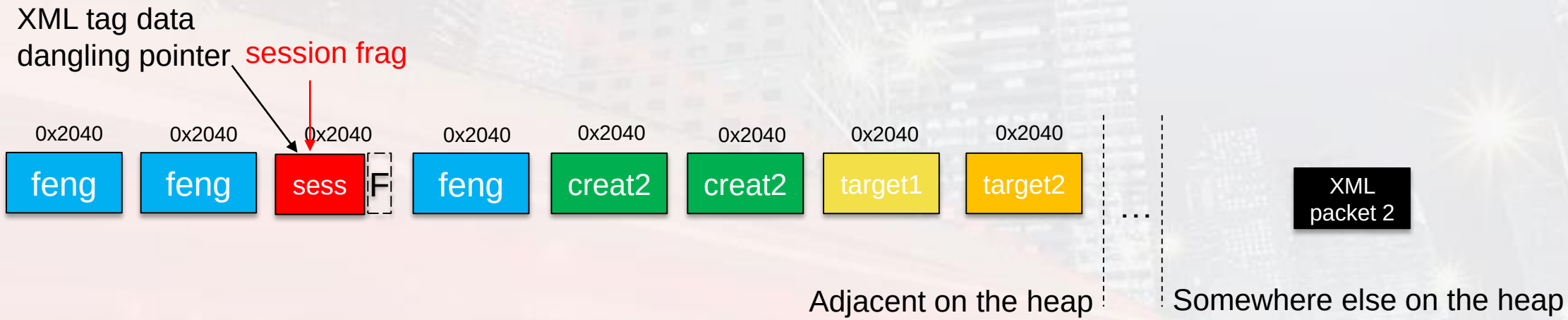


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled

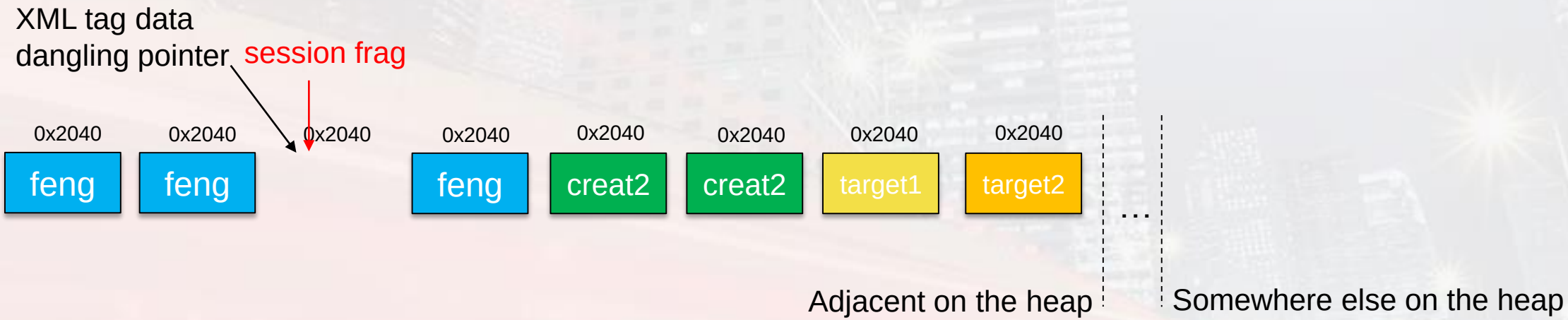


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled

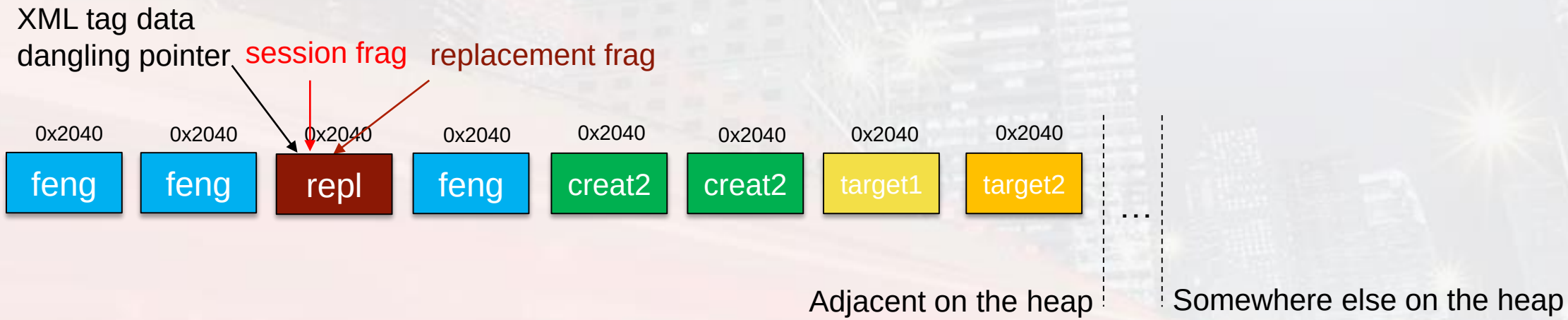


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag

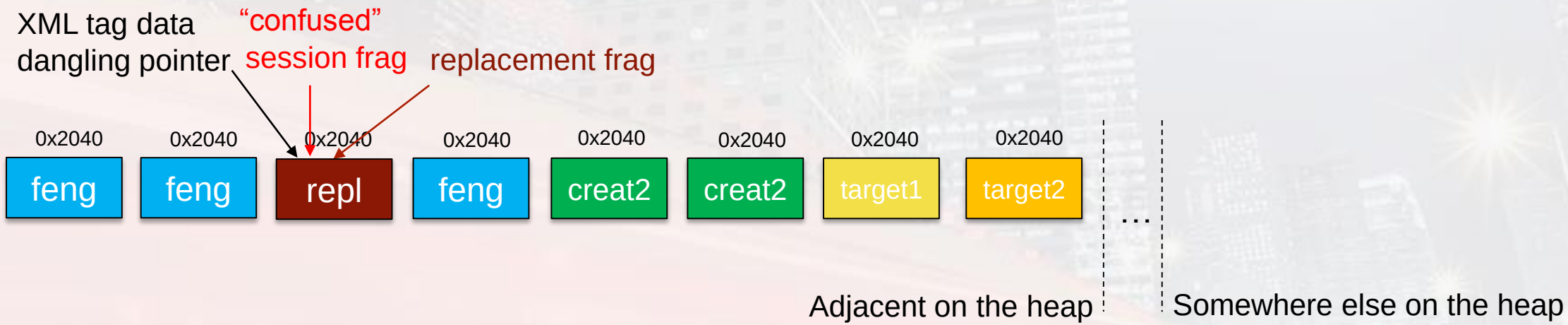


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag

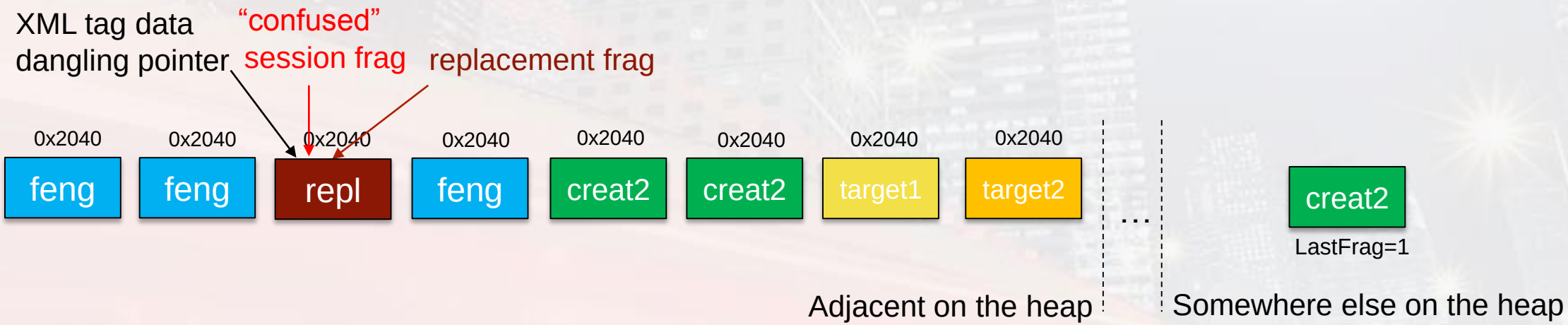


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag



Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

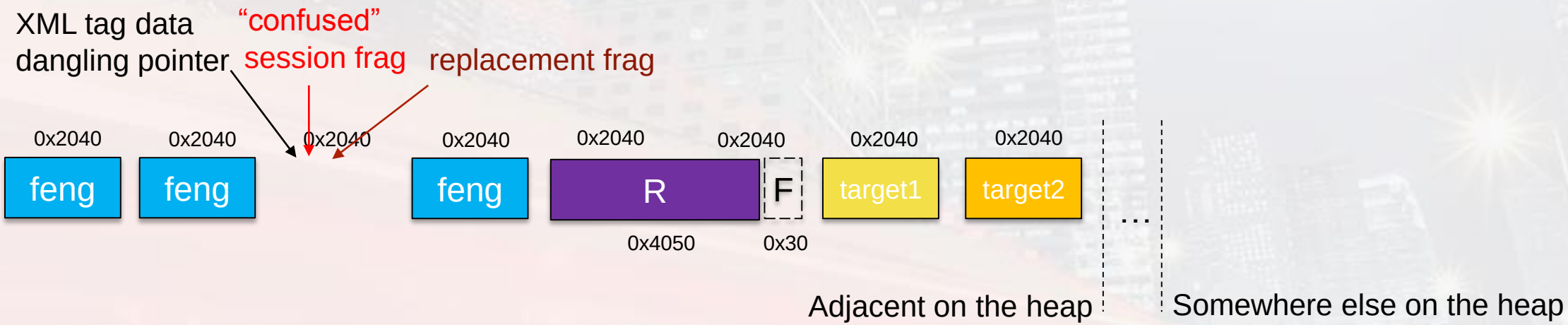
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag

Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet

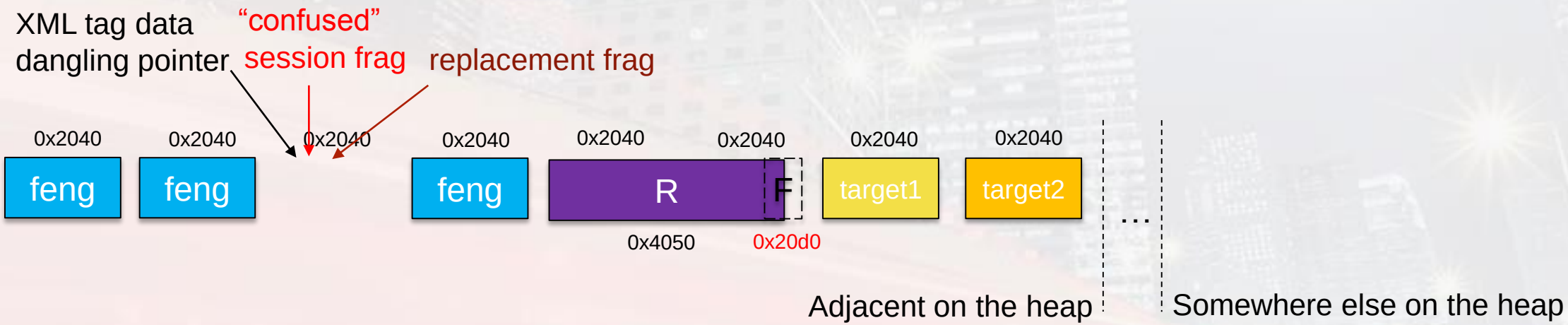


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet

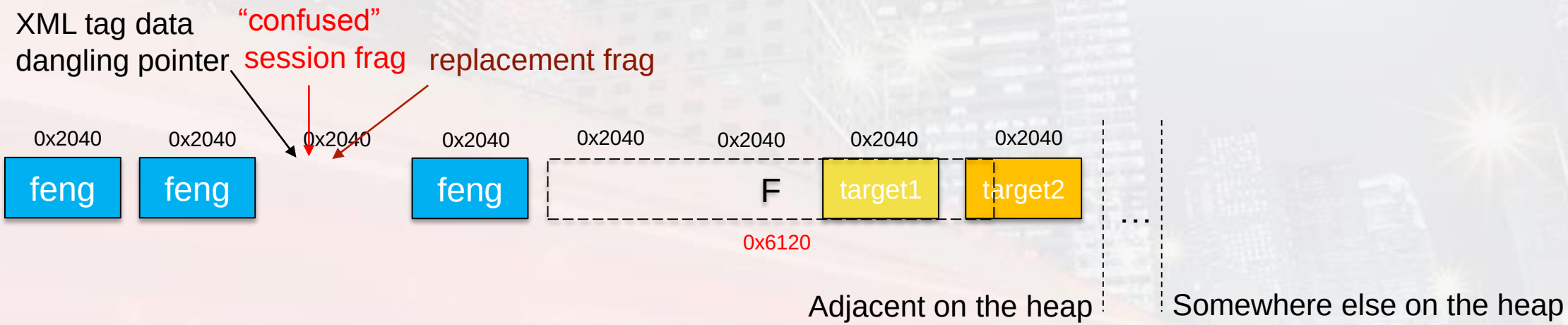


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet

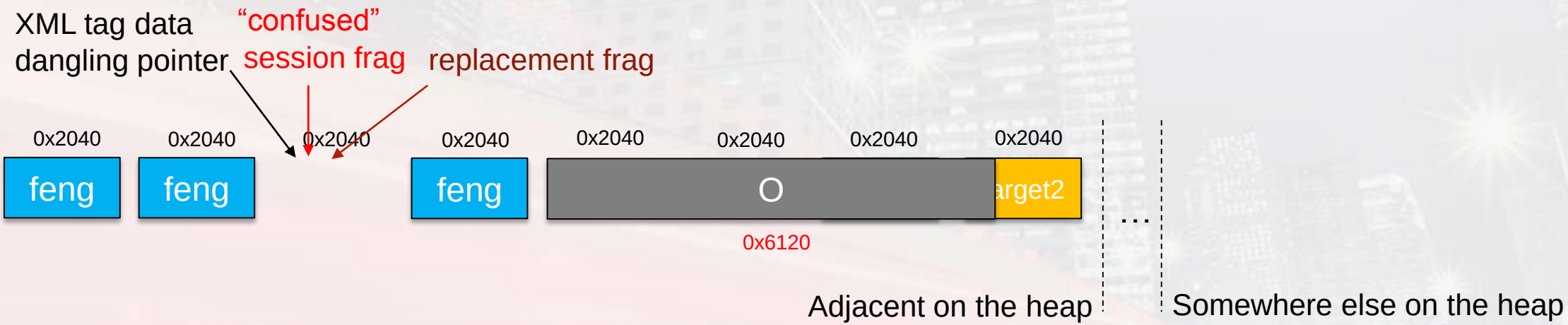


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

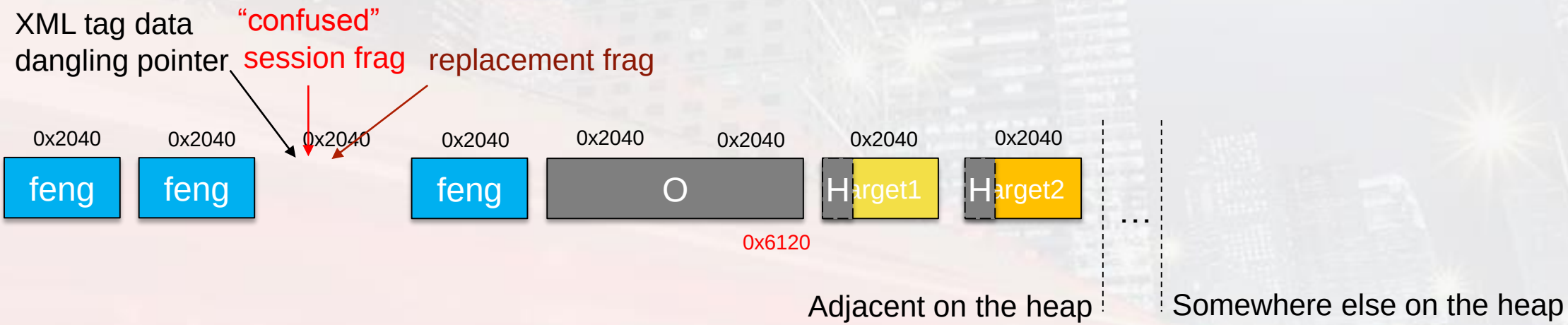


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

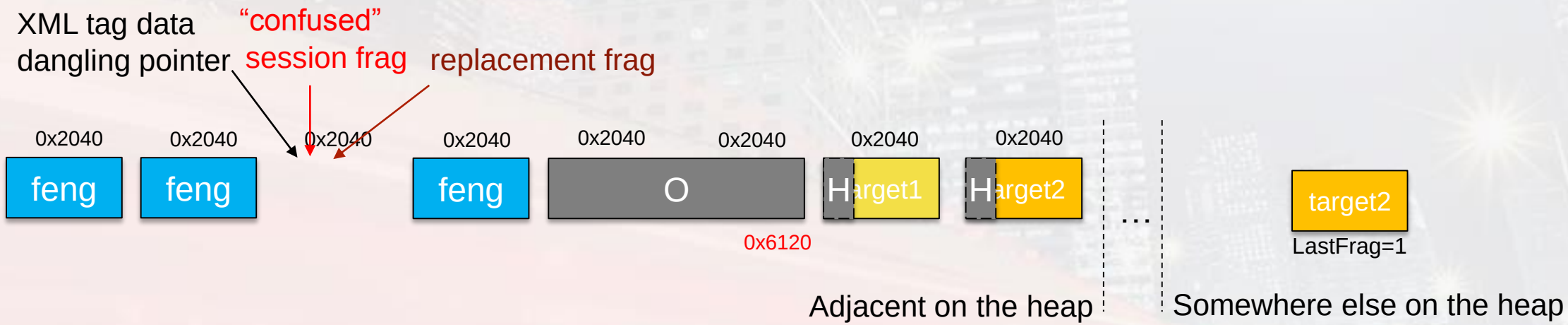


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

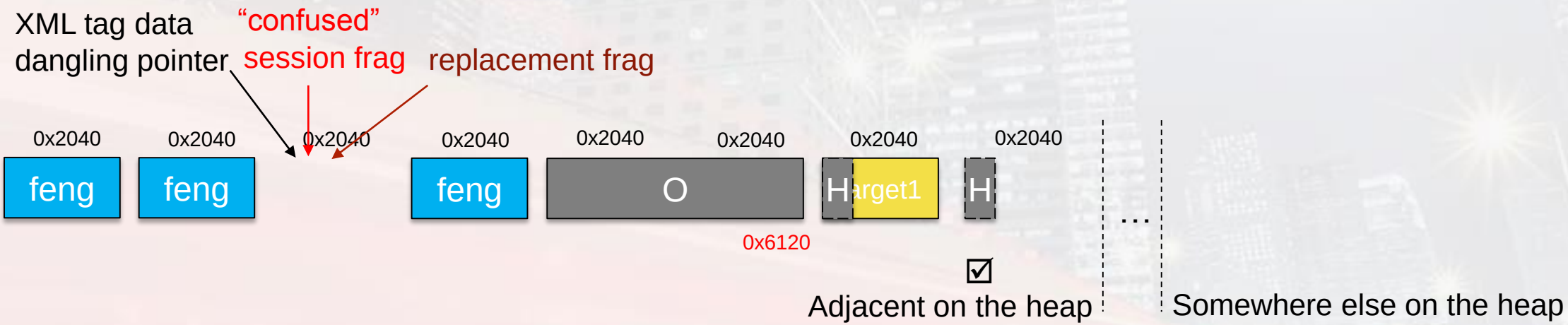


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

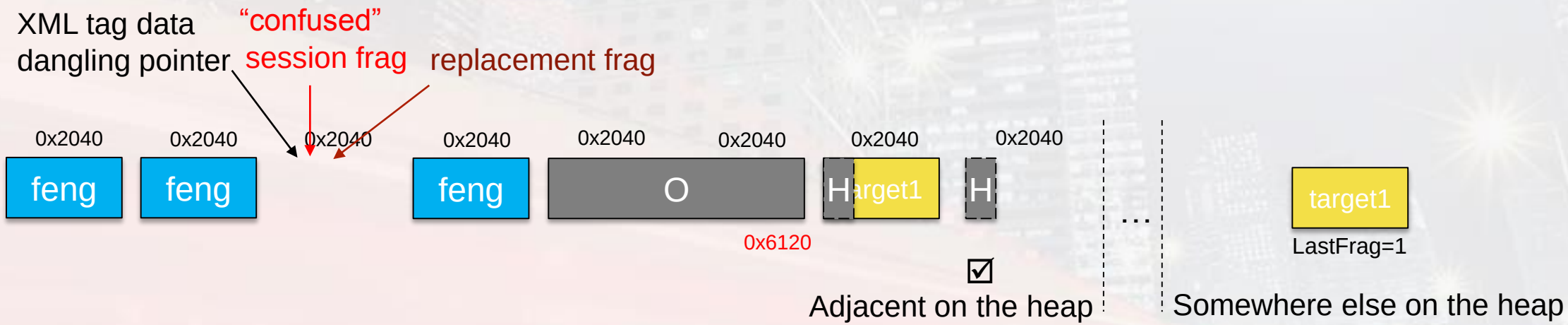


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

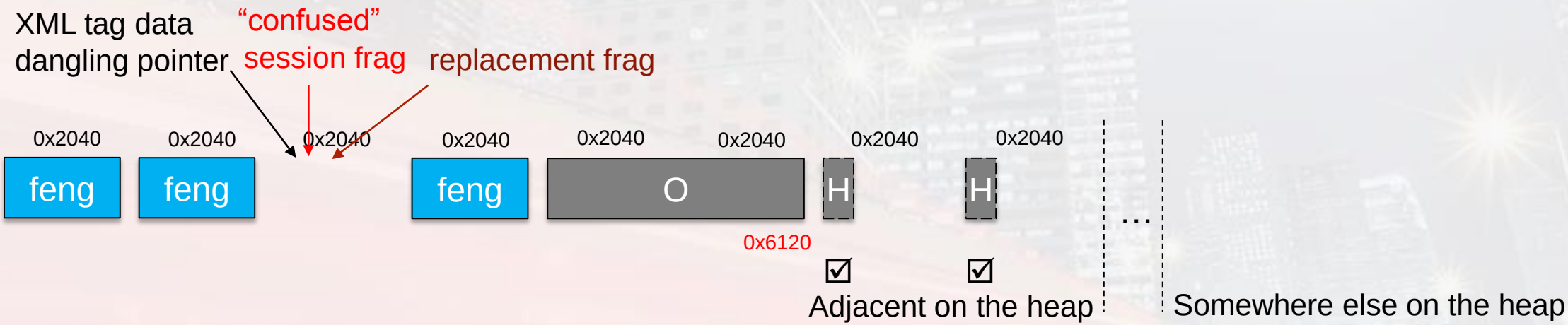


Exploit in a (coco)nut shell

- Hole creation primitive with IKEv1
- Allocate XML data in hole / freed at the end
- Allocate fragment in same hole
- Repeatable free primitive with XML
- Allocate fragment with larger size in same hole
- Trigger reassembly → corrupt linked list pointers
- Trigger mirror writes → corrupt a function pointer

Robin Hood uses IKEv1 sessions

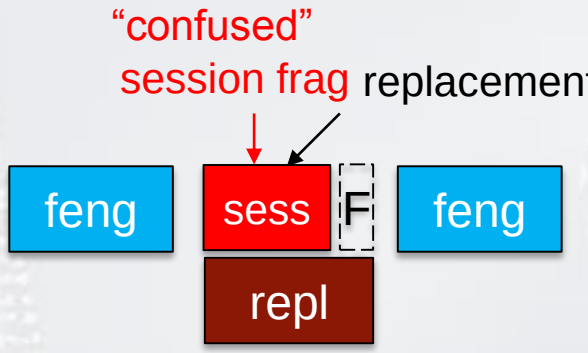
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet



Key facts

- We need sess/repl frags in same hole with len(repl) > len(sess)

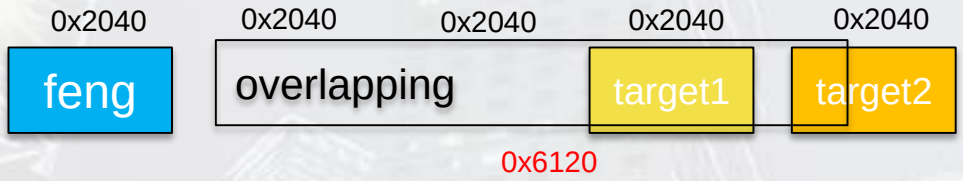
```
(gdb) dlchunk 0xad854108 -c 2 -p 0x44
0xad854108 M sz:0x02030 fl:CP alloc_pc:ike_receiver_process_data+0x3ed 0x6262 bb
0xad856138 F sz:0x00010 fl:-P 0x0000 hex(07c8)
(gdb) python print(frag_payload(0xad854108+0x28+0x1c))
struct frag_payload @ 0xad85414c {
 next_payload     = 0x0
 critical_bit     = 0x0
 payload_length   = 0x1fe6
 id               = 0x10
 seqno            = 0x2
 last_frag        = 0x1

(gdb) dlchunk 0xad854108 -c 1 -p 0x44
0xad854108 M sz:0x02040 fl:CP alloc_pc:ike_receiver_process_data+0x3ed 0x6666 ff
(gdb) python print(frag_payload(0xad854108+0x28+0x1c))
struct frag_payload @ 0xad85414c {
 next_payload     = 0x0
 critical_bit     = 0x0
 payload_length   = 0x1ff2
 id               = 0x20
 seqno            = 0x2
 last_frag        = 0x1
```



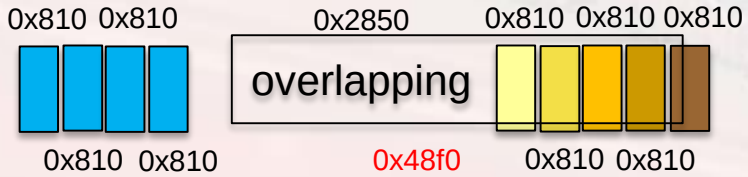
- We leave a small free chunk behind sess
- Confusion state: IKEv1 frags with different length in same chunk

Key facts (2)

- Overlapping chunk's size dictates max number of mirror writes



- For a given session, total accumulated length needs < 0x8000
  - XML buffer used by double free primitive is 0x2040 chunk
  - With 0x2040 chunks, it means maximum 2 mirror writes (see above)
- Solution is to change the granularity and use 0x810 chunks



Other approaches

1. Having one frag / the reassembled packet in the same chunk
   - But when reassembly fails, results in another double-free ☺
2. XML data is appended with `strncat()`
   - Overwrite first fragment to change its length?
   - Need a `strncat()`-friendly character
   - Can't use very large length due to reassembly incomplete check
   - But still need to allocate something else anyway to avoid double-free

- Took 2 weeks to build an exploit
  - Prior to that, took months to write asatools

Conclusions

Lessons learnt

- Fuzzing just the tags list is enough to find the bug
  - Radamsa was useless in our case
- Working exploit on 32-bit (no ASLR/DEP)
  - Note: some old 64-bit don't have ASLR either [1] 😐
- 7-year old bug? – AnyConnect Host Scan available since 2011
  - Cisco-specific handlers, not libexpat
- IKEv1 frag primitive to overflow memory / create mirror writes
  - Confusion state: one chunk used for two different IKEv1 packets
- IKEv1 feng shui useful for any heap-based bug

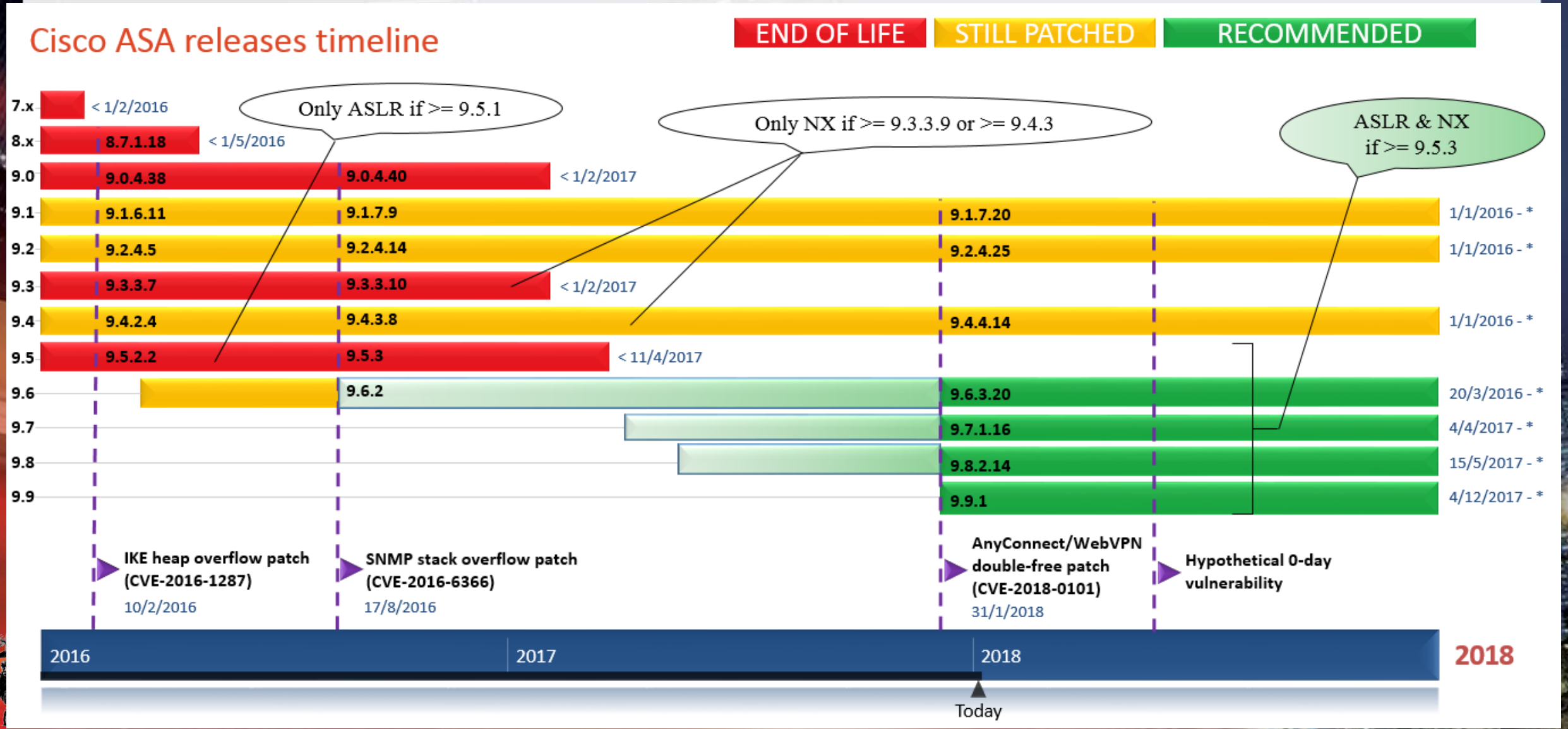
[1] https://github.com/nccgroup/asafw/blob/master/README.md#mitigation-summary

Next steps

- WebVPN/AnyConnect exploit only (not relying on IKEv1)?
- Turn a repeatable free into a memory revelation primitive?
  - Bypass ASLR on recent 64-bit?
  - Something like BENIGNCERTAIN on Cisco IOS [1]?
- XML grammar-based fuzzer to find new 0-day?
  - Support for tags, attributes, etc.

[1] https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20160916-ikev1

Protect against 0-day vulnerabilities?



Questions

- Special thanks to
  - My colleague Aaron Adams for the help on exploiting this ☺
  - Terri Grant from Cisco PSIRT for handling this
- Contact
  - @saidelike
  - cedric(dot)halbronn(at)nccgroup(dot)trust