Robin Hood vs Cisco ASA Anyconnect

Recon Brussels – February 2018
Speaker

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

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

Happy CVE-2018-0101 everyone!

8:12 AM - 1 Jan 2018

[Link to Cisco Security Advisory]

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

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

SSL VPN

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

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

- Also supported by native Windows client or even AnyConnect?

Previous work

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


https://github.com/nccgroup/asatools
Vulnerability & feng-shui overview

IKEv1

WebVPN/AnyConnect
SSL

CRASH!

Cisco AnyConnect Secure Mobility Client

VPN:
Ready to connect.
192.168.100.99
Connect
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

Finding a bug
Sniffing SSL AnyConnect

- 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

- Initial sample contains all supported tags
  - Input mutation fuzzing
Fuzzing architecture

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

Mutated XML packet (radamsa)

Ping (still alive?)

NO → save packet

https://github.com/aoh/radamsa
The wall is on fire…

- Want to start fuzzing before going on leave…
- ASA firewall keeps crashing
Understanding the bug
Triage

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

- Connect GDB
- Fire testcase
- Save crash info
Replay with gdb script

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

continue

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

• All the same crash
• Both ASAv 64-bit / ASA 32-bit
The smaller the better

- Fits in a tweet

```xml
<?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>
```

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*
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
<?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) {
        strcat(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 aggregateAuthFreeParserDataCutMem(...) {
    ...
    if (xml_tags[HOST_SCAN_REPLY].alloc)
        free(xml_tags[HOST_SCAN_REPLY].alloc); // [5]
    ...
}
Data handler

- First packet with <host-scan-reply> tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with <host-scan-reply> tag
  - No reallocation, copy data, free it
- Tags’ data copied and appended in the same chunk
  ➔ double-free vulnerability on 0x2040-byte chunk
Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags’ data copied and appended in the same chunk
  ➔ double-free vulnerability on 0x2040-byte chunk
Data handler

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

⇒ double-free vulnerability on 0x2040-byte chunk
Data handler

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

→ double-free vulnerability on 0x2040-byte chunk
Data handler

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

\[\Rightarrow\text{double-free vulnerability on 0x2040-byte chunk}\]
Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags’ data copied and appended in the same chunk
  - double-free vulnerability on 0x2040-byte chunk
Data handler

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

➔ double-free vulnerability on 0x2040-byte chunk
Data handler

• First packet with \textlangle host-scan-reply\rangle tag
  • Allocate heap buffer for data, copy data, free it (but dangling pointer)

• Second packet with \textlangle host-scan-reply\rangle tag
  • No reallocation, copy data, free it

• Tags’ data copied and appended in the same chunk

$\Rightarrow$ double-free vulnerability on 0x2040-byte chunk
Data handler

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

→ double-free vulnerability on 0x2040-byte chunk
Data handler

• First packet with `<host-scan-reply>` tag
  • Allocate heap buffer for data, copy data, free it (but dangling pointer)
• Second packet with `<host-scan-reply>` tag
  • No reallocation, copy data, free it
• Tags’ data copied and appended in the same chunk
  ➔ double-free vulnerability on 0x2040-byte chunk
Data handler

- First packet with `<host-scan-reply>` tag
  - Allocate heap buffer for data, copy data, free it (but dangling pointer)
- Second packet with `<host-scan-reply>` tag
  - No reallocation, copy data, free it
- Tags’ data copied and appended in the same chunk
  ➞ double-free vulnerability on 0x2040-byte chunk
assert() due to invalid metadata

- Inline metadata/header for heap chunks

```plaintext
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
```

- Hence why our fuzzer caught it!
Exploiting the bug like RobinHood
Objective: mirror write

- Allocated chunks hold pointers to alloc lists

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>prev_foot</td>
<td>0x8180d4d0</td>
</tr>
<tr>
<td>head</td>
<td>0x1d0 (CINUSE</td>
</tr>
<tr>
<td>mh_magic</td>
<td>0xa11c0123</td>
</tr>
<tr>
<td>mh_len</td>
<td>0x1a4</td>
</tr>
<tr>
<td>mh_refcount</td>
<td>0x0</td>
</tr>
<tr>
<td>mh_unused</td>
<td>0x0</td>
</tr>
<tr>
<td>mh_fd_link</td>
<td>0xacb85b30</td>
</tr>
<tr>
<td>mh bk_link</td>
<td>0xa8800604</td>
</tr>
<tr>
<td>allocator_pc</td>
<td>0x86816b3</td>
</tr>
<tr>
<td>free_pc</td>
<td>0x868161d</td>
</tr>
</tbody>
</table>

- Target mempool alloc lists to get a mirror write
  - No safe unlinking on Cisco specific metadata on all ASA versions
  - Even if dlmalloc or ptmalloc had safe unlinking for free chunks

- Mirror write: unlinking an element from a doubly-linked list will trigger two write operations
  - One operation is the useful one, the other is a side effect
  - Constraint: both need to be writable addresses

Was already abused in 2016 by Exodus Intel
Exploit strategy

Use double free
→ Create confusion state on the heap
→ Overflow some memory
    → Overwrite linked list pointers
→ Trigger mirror write primitives
    → Overwrite a function pointer
→ Get RCE
Use what you got

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

- Simple IKE reassembly

Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

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

```
SeqNo=1
sess1 sess1 sess2 sess1 sess1
```
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

- Session 1: fill holes
- Session 2: only two fragments
  - Frag 1: future hole
  - Frag 2: trigger reassembly, hence creating hole
Primitive 1 - Hole creation with IKEv1

• Session 1: fill holes
• Session 2: only two fragments
  • Frag 1: future hole
  • Frag 2: trigger reassembly, hence creating hole
Primitive 2 - Overflow with IKEv1

Note: for the sake of simplicity, we do not show sequence numbers anymore
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

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

sess1  sess1  sess1

LastFrag=1
Primitive 2 - Overflow with IKEv1

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

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

 sess1  sess1  sess1

LastFrag=1
Primitive 2 - Overflow with IKEv1

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

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

```
 sess1   sess1   sess1
  LastFrag=1

 Reassembled packet
```
Primitive 2 - Overflow with IKEv1

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

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

```
sess1  sess1  sess1  \rightarrow  Reassembled packet
   \__________  
         sess1  
             LastFrag=1
```
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

1. Reduce the accumulated length (CVE-2016-1287)
Primitive 2 - Overflow with IKEv1

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

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

   | sess1 | sess1 | sess1 | Reassembled packet | Heap overflow |
   | sess1 | sess1 | sess1 |
   |       | LastFrag=1 |

2. Increase fragment length (overflow primitive)
Note: for the sake of simplicity, we do not show sequence numbers anymore

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

2. Increase fragment length (overflow primitive)
Primitive 2 - Overflow with IKEv1

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

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

   sess1  sess1  sess1  \rightarrow  Reassembled packet
   \hspace{1cm} \text{LastFrag}=1

2. Increase fragment length (overflow primitive)

   sess1  sess1
Primitive 2 - Overflow with IKEv1

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

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

2. Increase fragment length (overflow primitive)
Primitive 2 - Overflow with IKEv1

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

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

   ![Diagram of sess1 sess1 sess1 → Reassembled packet with LastFrag=1]

2. Increase fragment length (overflow primitive)

   ![Diagram of sess1 sess1 sess1 → Heap overflow]
Primitive 2 - Overflow with IKEv1

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

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

   - LastFrag=1

2. Increase fragment length (overflow primitive)

   - LastFrag=1
Primitive 2 - Overflow with IKEv1

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

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

  (sess1, sess1, sess1) → Reassembled packet
   LastFrag=1

2. Increase fragment length (overflow primitive)

   sess1, sess1, sess1) → Reassembled packet
   LastFrag=1

Heap overflow
Primitive 2 - Overflow with IKEv1

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

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

2. Increase fragment length (overflow primitive)
Primitive 2 - Overflow with IKEv1

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

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

2. Increase fragment length (overflow primitive)
Primitive 2 - Overflow with IKEv1

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

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

2. Increase fragment length (overflow primitive)
Primitive 2 - Overflow with IKEv1

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

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

2. Increase fragment length (overflow primitive)
Limited overflow (18-byte on 32-bit)

```c
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 assembling 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);
        ...
    }
    return TRUE;
}
```
Primitive 3 – Repeatable free with XML

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

Interest confusion state
Primitive 3 – Repeatable free with XML

• XML data allocated for first packet, then freed
• Allocate IKEv1 fragment in same hole
• Free IKEv1 fragment using the double free primitive
• Allocate another IKEv1 fragment in same hole
→ Interesting confusion state
Primitive 3 – Repeatable free with XML

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

Interesting confusion state

XML tag data dangling pointer

feng feng

feng feng
Primitive 3 – Repeatable free with XML

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

→ Interesting confusion state
**Primitive 3 – Repeatable free with XML**

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

→ Interesting confusion state
Primitive 3 – Repeatable free with XML

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

→ Interesting confusion state
Primitive 3 – Repeatable free with XML

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

→ Interesting confusion state
Primitive 3 – Repeatable free with XML

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

Interesting confusion state
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
  - Blue: separators

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
  - Blue: separators

0x2040 feng 0x2040 feng

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
• Blue: separators
• Green: hole creation

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation

```
0x2040 0x2040 0x2040 0x2040 0x2040
feng  feng  creat1  feng  creat2
```

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

- 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 $\rightarrow$ corrupt linked list pointers
- Trigger mirror writes $\rightarrow$ corrupt a function pointer

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
Exploit in a (coco)nut shell

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

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

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes

XML tag data
dangling pointer

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data dangling pointer

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

![Diagram showing memory allocation and exploit process]
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

session frag

Adjacent on the heap
Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data dangling pointer

```plaintext
feng 0x2040
feng 0x2040
feng 0x2040
creat2 0x2040
creat2 0x2040
target1 0x2040
target2 0x2040
```

Adjacent on the heap Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

session frag replacement frag

Adjacent on the heap
Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

Adjacent on the heap
Somewhere else on the heap

"confused"
replacement frag
session frag
repl
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer
“confused”
session frag
replacement frag

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
- dangling pointer

confused

session frag

replacement frag

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
- dangling pointer
- "confused"
- session frag
- replacement frag

- Adjacent on the heap
- Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

"confused"
session frag
replacement frag

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

“confused”
session frag
replacement frag

Adjacent on the heap
Somewhere else on the heap
Exploit in a (coco)nut shell

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

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

XML tag data
dangling pointer

“confused”
session frag replacement frag

Adjacent on the heap
Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
dangling pointer

---

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
dangling pointer

adjacent on the heap

somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
dangling pointer

"confused" session frag replacement frag

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
dangling pointer

Adjacent on the heap

Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
Dangling pointer
Session frag
Replacement frag

Adjacent on the heap
Somewhere else on the heap
Exploit in a (coco)nut shell

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

Robin Hood uses IKEv1 sessions
- Blue: separators
- Green: hole creation
- Orange: targets for mirror writes
- Red: confused session reassembled
- Brown: replacement frag
- Purple: reassembled packet
- Grey: overlapping packet

XML tag data
dangling pointer

```
0x2040 0x2040 0x2040 0x2040
feng feng feng O
```

adjacent on the heap

```
0x6120
```

somewhere else on the heap

```
0x2040 0x2040
H H
```
Key facts

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

```
(gdb) dlchunk 0xad854108 -c 2 -p 0x44
0xad854108 M sz:0x02030 fl:CP alloc_pc:ike_receiver_process_data+0x3ed 0x6262 bb
0xad854108 P 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
```

- We leave a small free chunk behind sess
- Confusion state: IKEv1 frags with different length in same chunk
Key facts (2)

- Overlapping chunk’s size dictates max number of mirror writes
  - Overlapping chunk's size dictates the maximum number of mirror writes.
  
- For a given session, total accumulated length needs < 0x8000
  - For a given session, total accumulated length needs to be less than 0x8000.
  - XML buffer used by double free primitive is 0x2040 chunk.
  - With 0x2040 chunks, it means maximum 2 mirror writes (see above).
  
- Solution is to change the granularity and use 0x810 chunks
  - Change the granularity and use 0x810 chunks to solve the problem.
Other approaches

1. Having one frag / the reassembled packet in the same chunk
   • But when reassembly fails, results in another double-free 😊

2. XML data is appended with `strncat()`
   • Overwrite first fragment to change its length?
   • Need a `strncat()`-friendly character
   • Can’t use very large length due to reassembly incomplete check
   • But still need to allocate something else anyway to avoid double-free
   • Took 2 weeks to build an exploit
     • Prior to that, took months to write `asatools`
Conclusions
Lessons learnt

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

Next steps

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

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

<table>
<thead>
<tr>
<th>Version</th>
<th>End of Life</th>
<th>Still Patched</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.x</td>
<td>&lt; 1/2/2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.x</td>
<td>1/5/2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.0</td>
<td>&lt; 1/2/2017</td>
<td>9.0.4.40</td>
<td></td>
</tr>
<tr>
<td>9.1</td>
<td>9.1.6.11</td>
<td>9.1.7.9</td>
<td>9.1.7.20</td>
</tr>
<tr>
<td>9.2</td>
<td>9.2.4.5</td>
<td>9.2.4.14</td>
<td>9.2.4.25</td>
</tr>
<tr>
<td>9.3</td>
<td>9.3.3.7</td>
<td>9.3.3.10</td>
<td>9.4.4.14</td>
</tr>
<tr>
<td>9.4</td>
<td>9.4.2.4</td>
<td>9.4.3.8</td>
<td></td>
</tr>
<tr>
<td>9.5</td>
<td>9.5.2.2</td>
<td>9.5.3</td>
<td></td>
</tr>
<tr>
<td>9.6</td>
<td>9.6.2</td>
<td>9.6.3.20</td>
<td></td>
</tr>
<tr>
<td>9.7</td>
<td></td>
<td>9.7.1.16</td>
<td></td>
</tr>
<tr>
<td>9.8</td>
<td></td>
<td>9.8.2.14</td>
<td></td>
</tr>
<tr>
<td>9.9</td>
<td></td>
<td>9.9.1</td>
<td></td>
</tr>
</tbody>
</table>

- **Only ASLR if >= 9.5.1**
- **Only NX if >= 9.3.3.9 or >= 9.4.3**
- **ASLR & NX if >= 9.5.3**

**CVE-2016-1237**
- IKE heap overflow patch
  - 10/2/2016

**CVE-2016-6366**
- SNMP stack overflow patch
  - 17/8/2016

**CVE-2018-0101**
- AnyConnect/WebVPN double-free patch
  - 31/1/2018

**Hypothetical 0-day vulnerability**
- 4/12/2017

**Timeline:**
- 2016
  - 10/2/2016
  - 17/8/2016
- 2017
  - 31/1/2018
- 2018
Questions

• Special thanks to
  • My colleague Aaron Adams for the help on exploiting this 😊
  • Terri Grant from Cisco PSIRT for handling this

• Contact
  • @saidelike
  • cedric(dot)halbronn(at)nccgroup(dot)trust