Hacking the Supply Chain

The Ripple20 Vulnerabilities Haunt Hundreds of Millions of Critical Devices

Black Hat USA 2020

Who are we?

JSOF is a software security consultancy

- **Shlomi Oberman**, co-founder, JSOF
- **Moshe Kol**, Security researcher, JSOF; Finder of Ripple20
- **Ariel Schön**, Security researcher, JSOF

Agenda

- Ripple20
- CVE-2020-11901
- Exploiting CVE-2020-11901

Ripple20

- Series of 19 zero-day vulnerabilities in **Treck TCP/IP***
- Amplified by the supply chain
- 100’s of millions of devices
- Medical, ICS, Home, Enterprise, Transportation, Utilities

Ripple20

CVE-2020-11896	CVE-2020-11901	CVE-2020-11906	CVE-2020-11911
CVE-2020-11897	CVE-2020-11902	CVE-2020-11907	CVE-2020-11912
CVE-2020-11898	CVE-2020-11903	CVE-2020-11908	CVE-2020-11913
CVE-2020-11899	CVE-2020-11904	CVE-2020-11909	CVE-2020-11914
CVE-2020-11900	CVE-2020-11905	CVE-2020-11910	

- 4 critical remote code execution vulnerabilities

Ripple20

CVE-2020-11896	CVE-2020-11901	CVE-2020-11906	CVE-2020-11911
CVE-2020-11897	CVE-2020-11902	CVE-2020-11907	CVE-2020-11912
CVE-2020-11898	CVE-2020-11903	CVE-2020-11908	CVE-2020-11913
CVE-2020-11899	CVE-2020-11904	CVE-2020-11909	CVE-2020-11914
CVE-2020-11900	CVE-2020-11905	CVE-2020-11910	

- 8 medium-high severity vulnerabilities

100’s of Millions of Devices Affected

And many more...

100’s of Millions of Devices Affected

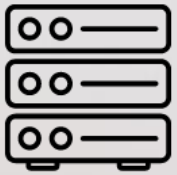
Medical

Printers

Utilities

Transportation

Networking



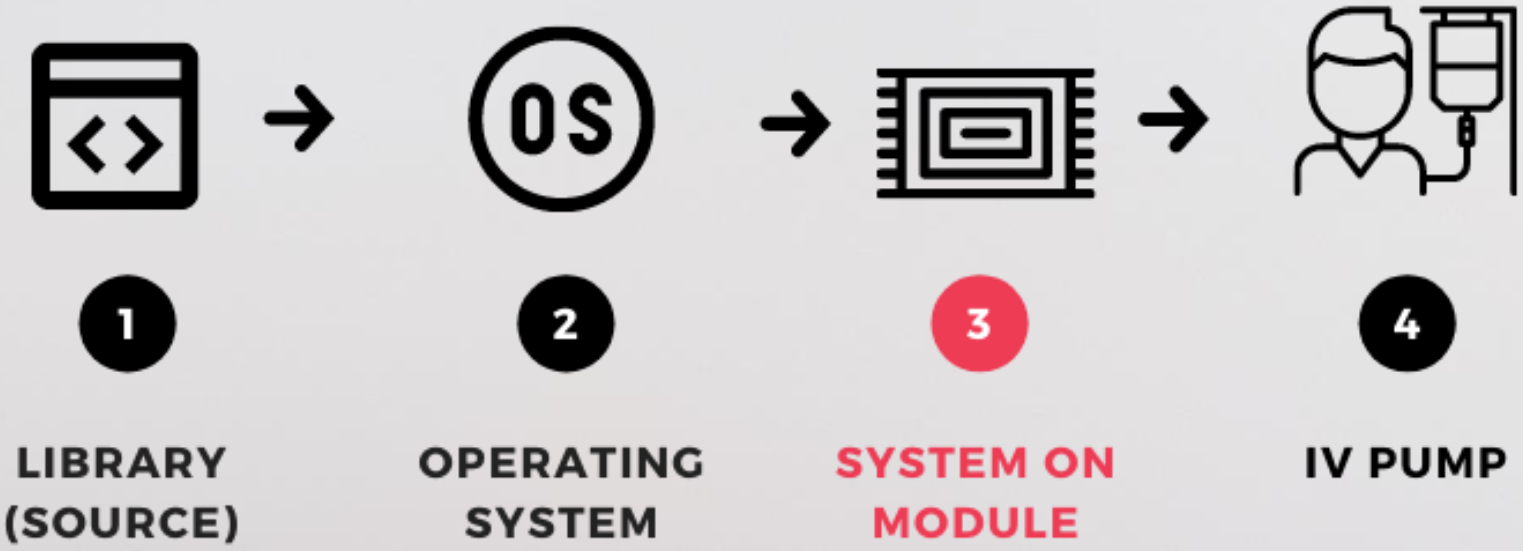
Datacenter

Smart Buildings

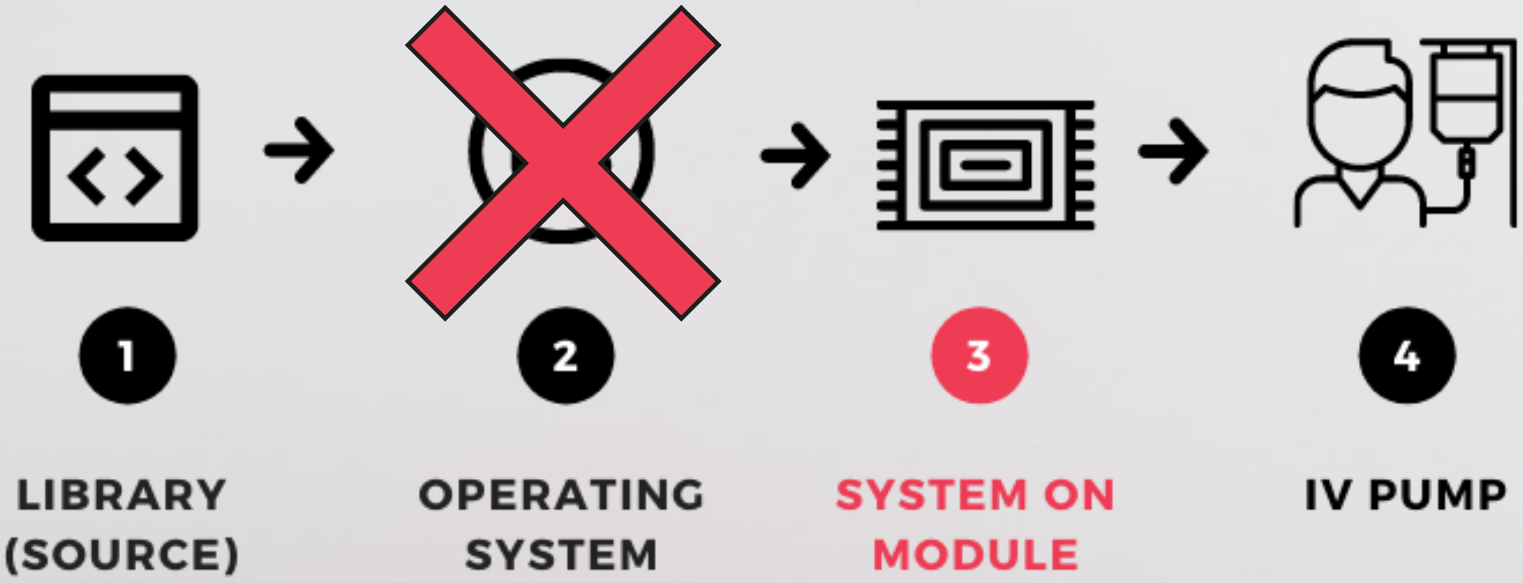
Industrial

- Assumption: Every mid-large US organization has one

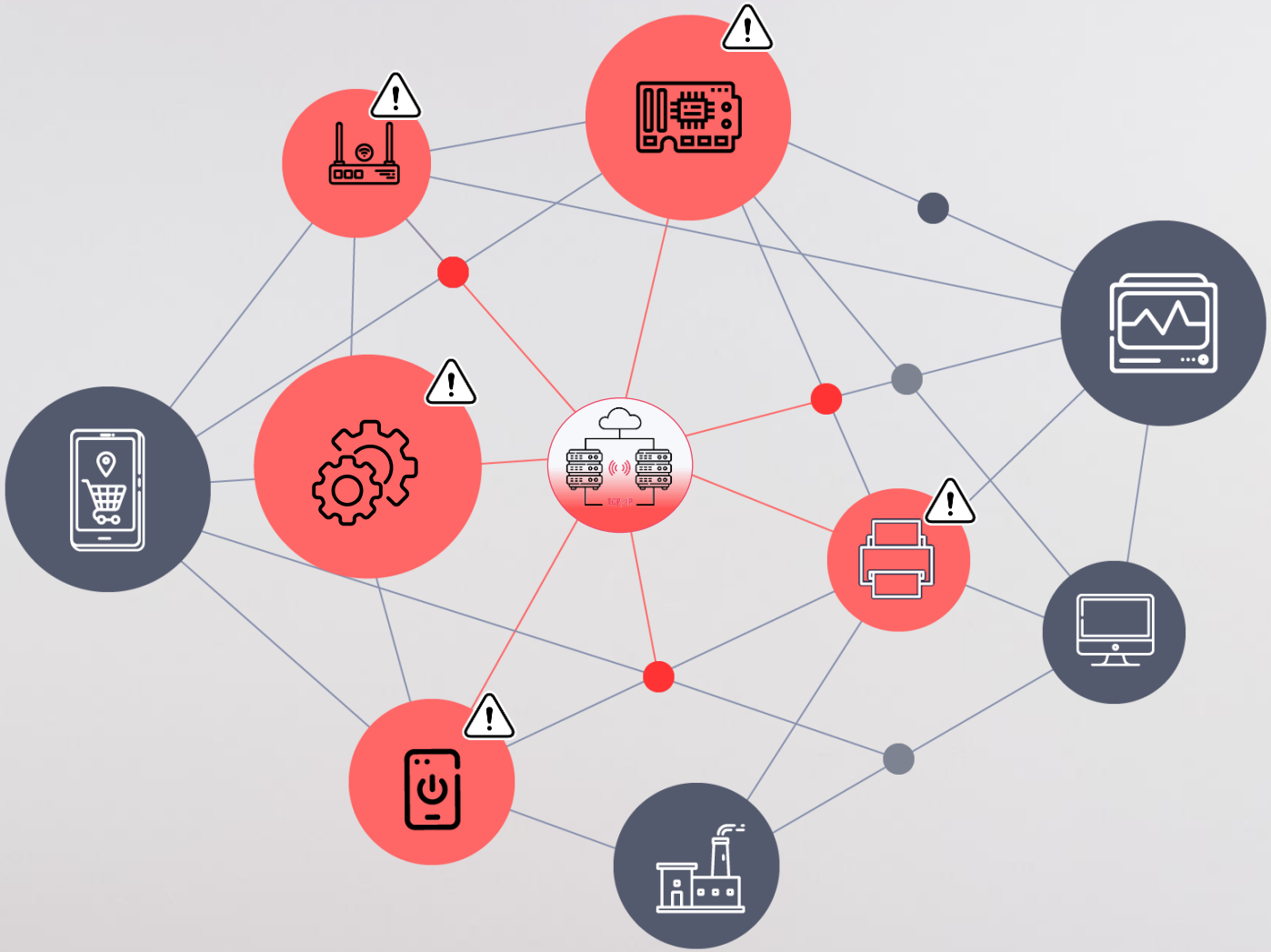
Supply chain



Supply chain



Icon from www.flaticon.com



Vulnerabilities

Ripple20

Why Treck TCP/IP?

- Supply chain - mostly unexplored
- 1 vulnerability == multiple products
- Large IoT impact
- Zombie vulnerabilities
- Good attack surface

Treck TCP/IP

- Treck is a small American company
- Treck TCP/IP is a proprietary TCP/IP stack; Available >20 years
- Embedded devices and RTOS
- Very configurable. Each Treck instance is different.
- Strategically located at the start of a long supply-chain

Ripple20 Research

- Reverse engineering of 6 different devices with multiple versions
- Every device has a different configuration
- Ongoing research Sep’19 - Jun’20 (9 months)
- Some strange architectures and firmwares involved

2 whitepapers released

About CVE-2020-11901

- Critical vulnerabilities in Treck's DNS Resolver component.
- Once successfully exploited, allows for remote code execution.
- Can traverse NAT boundaries.

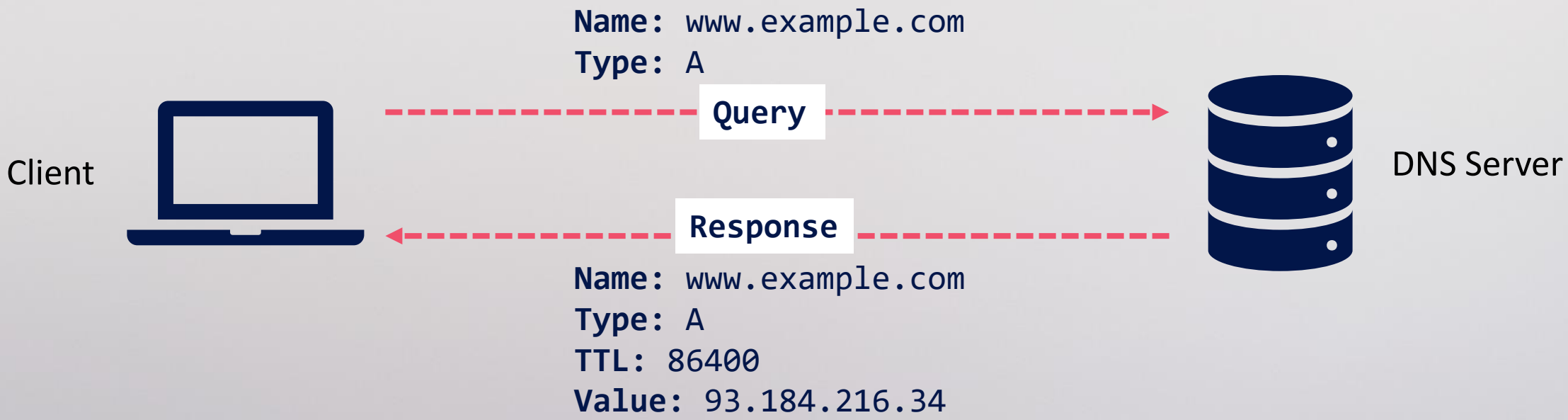
- 4 vulnerabilities and 1 artifacts.
- Vary over time and vendor.

CVE-2020-11901

AKA “the DNS bugs”

DNS Primer: The Basics

- The DNS protocol maps between **domain names** and **IP addresses**.
- Client **resolves** a name by issuing a query to a DNS server.
- The DNS server **looks up** the name and returns a response.



DNS Primer: Record Types

- DNS servers can return multiple answers in the same DNS response.
- An answer is specified as a **resource record**:

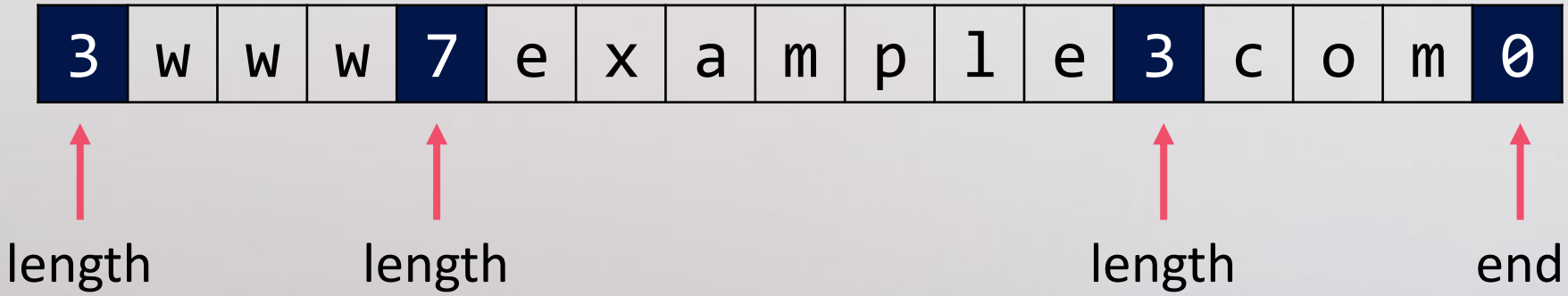
NAME (var)	**TYPE** (2 bytes)	**CLASS** (2 bytes)	**TTL** (4 bytes)	**RDLENGTH** (2 bytes)	**RDATA** (var)

- Questions and answers have a **type**. Common types include:

Type	Description
A	IPv4 address for the queried domain.
CNAME	Alias (canonical name).
MX	Domain name of a mail server for the queried domain.

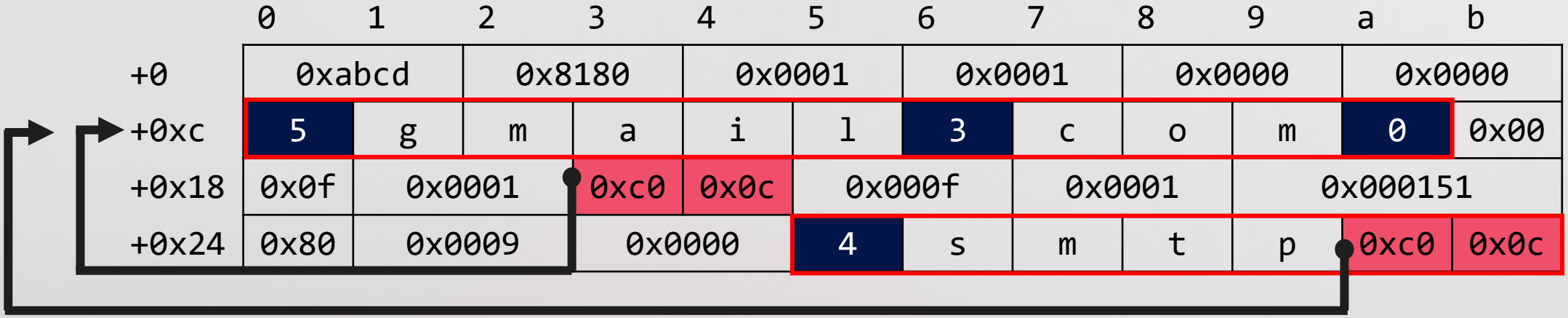
Domain Names Encoding

- Domain names are encoded as a **sequence of labels**.
- Each label is preceded by a **length byte**.
- Maximum label length is 63.

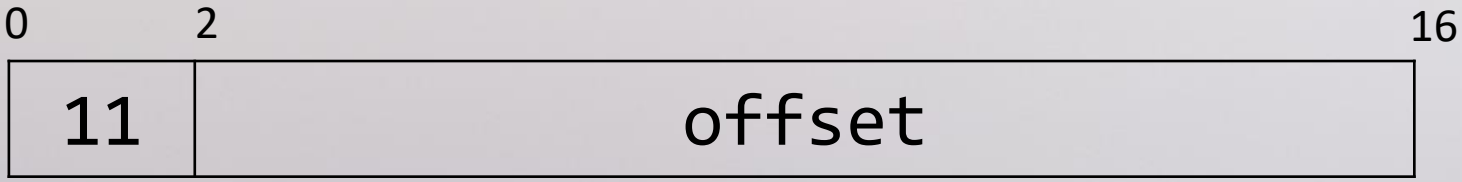


DNS Message Compression

- Compression is achieved by replacing a sequence of labels with a **pointer to prior occurrence** of the same sequence.



- Compression pointer is encoded in **two bytes**, the first begins with 11.



DNS Parsing Logic: Type MX

```
if (cacheEntryQueryType == DNS_TYPE_MX && rrtype == DNS_TYPE_MX) {
    addr_info = tfDnsAllocAddrInfo();
    if (addr_info != NULL) {
        /* copy preference value of MX record */
        memcpy(&addr_info->ai_mxpref, resourceRecordAfterNamePtr + 10, 2);
        /* compute the length of the MX hostname */
        labelLength = tfDnsExpLabelLength(resourceRecordAfterNamePtr + 0xc, pktDataPtr);
        addr_info->ai_mxhostname = NULL;
        if (labelLength != 0) {
            /* allocate buffer for the expanded name */
            asciiPtr = tfGetRawBuffer(labelLength);
            addr_info->ai_mxhostname = asciiPtr;
            if (asciiPtr != NULL) {
                /* copy MX hostname to `asciiPtr` as ASCII */
                tfDnsLabelToAscii(resourceRecordAfterNamePtr + 0xc, asciiPtr, pktDataPtr);
                /* ... */
            }
/* ... */
```

1

2

3

*Pseudo-code

DNS Label Length Calculation

```
tt16Bit tfDnsExpLabelLength(tt8BitPtr labelPtr, tt8BitPtr pktDataPtr){
    tt8Bit currLabelLength;
    tt16Bit i = 0, totalLength = 0;

    while (labelPtr[i] != 0) {
        currLabelLength = labelPtr[i];
        if ((currLabelLength & 0xc0) == 0) {
            totalLength += currLabelLength + 1;
            i += currLabelLength + 1;
        } else {
            newLabelPtr = pktDataPtr + (((currLabelLength & 0x3f) << 8) | labelPtr[i+1]);
            if (newLabelPtr >= labelPtr) {
                return 0;
            }
            labelPtr = newLabelPtr;
            i = 0;
        }
    }
    return totalLength;
}
```

Reads the current label length

Handles the common case: no compression

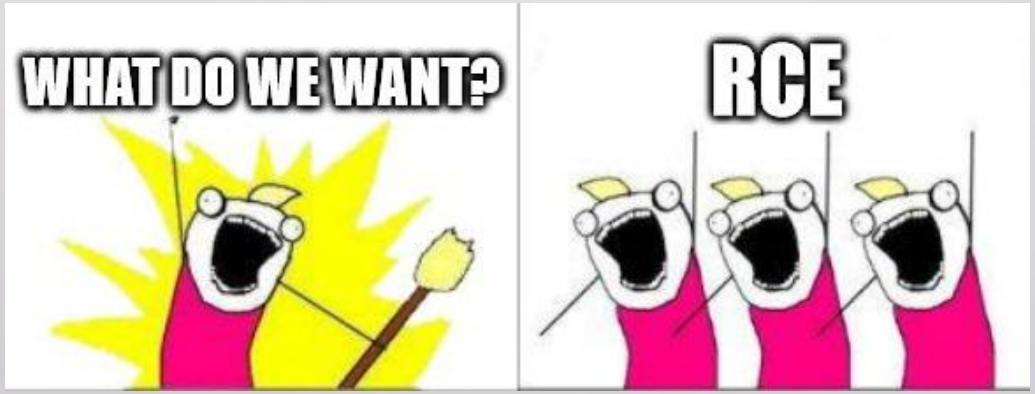
Reads the compression offset

Only allows jumping **backwards**

*Pseudo-code

Vulnerability #1: Read Out-Of-Bounds

- `tfDnsExpLabelLength` might read data out of the packet buffer while iterating over the length bytes (stops at a zero length byte).
- Could result in **denial-of-service** (e.g., read from unmapped page).
- **Information leakage**:
  - `tfDnsLabelToAscii` has no bounds check either.
  - Data from the heap could be interpreted as an MX hostname.
  - Data is leaked when the client tries to resolve the MX hostname.
- Affects Treck version 4.7+, fixed later.
- Sweet! but we want RCE…



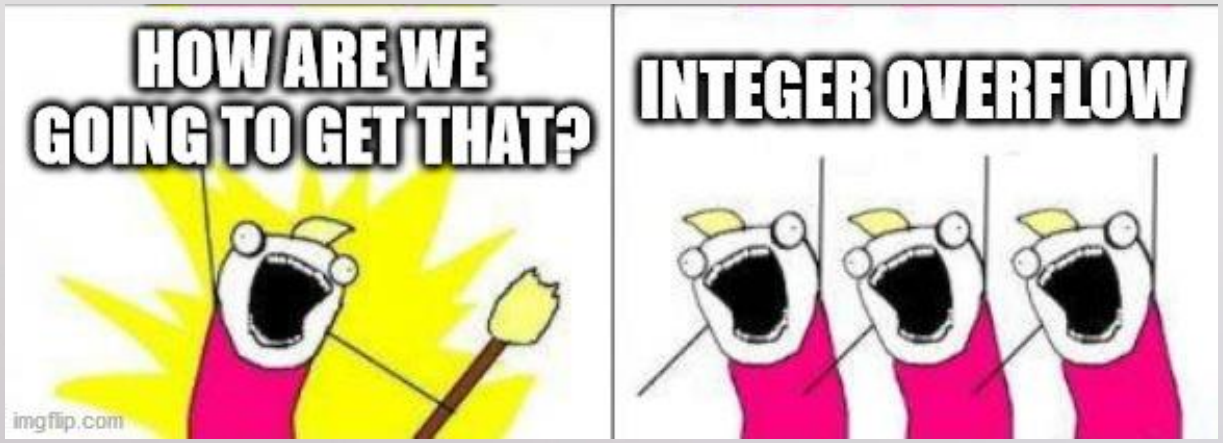
More Issues with `tfDnsExpLabelLength`

- Maximum domain name of 255 characters is not enforced.
- Does not validate the characters of the domain name: should be alphanumeric and ‘-’ only.
- **`totalLength` variable is stored as an `unsigned short` (`tt16Bit`).**

```
tt16Bit tfDnsExpLabelLength(tt8BitPtr labelPtr, tt8BitPtr pktDataPtr){
    tt8Bit currLabelLength;
    tt16Bit i = 0, totalLength = 0;
    /* ... */
    return totalLength;
}
```

More Issues with `tfDnsExpLabelLength`

- Maximum domain name of 255 characters is not enforced.
- Does not validate the characters of the domain name: should be alphanumeric and ‘-’ only.
- **`totalLength` variable is stored as an `unsigned short` (`tt16Bit`).**



Vulnerability #2: Integer Overflow

- We need to construct a name whose length is larger than 65536.
- **Can we overflow the `totalLength` variable within a DNS response packet?**
- **Yes!** We use the DNS compression feature to achieve this.
- Idea: nested compression pointers.
- Two challenges:
  - Maximum size of the DNS response packet allowed is 1460 bytes.
  - We can only jump **backwards** from our current label pointer.

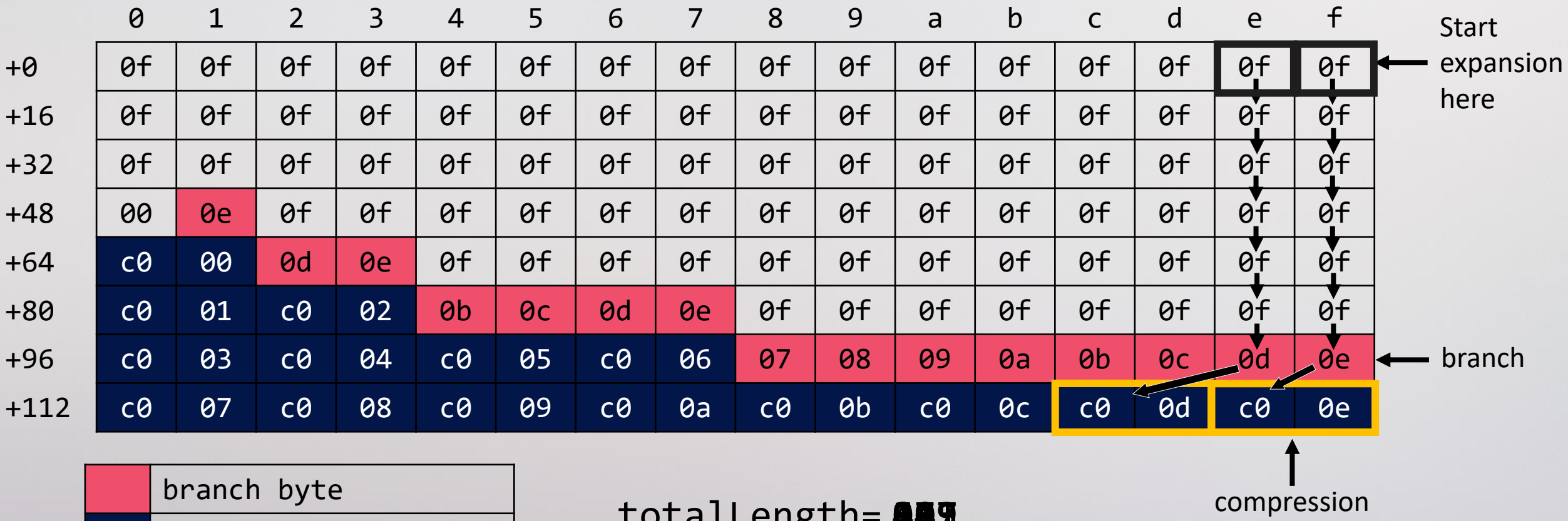
Vulnerability #2: Integer Overflow

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
+0	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f
+16	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f
+32	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f
+48	00	0e	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f
+64	c0	00	0d	0e	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f	0f
+80	c0	01	c0	02	0b	0c	0d	0e	0f	0f	0f	0f	0f	0f	0f	0f
+96	c0	03	c0	04	c0	05	c0	06	07	08	09	0a	0b	0c	0d	0e
+112	c0	07	c0	08	c0	09	c0	0a	c0	0b	c0	0c	c0	0d	c0	0e

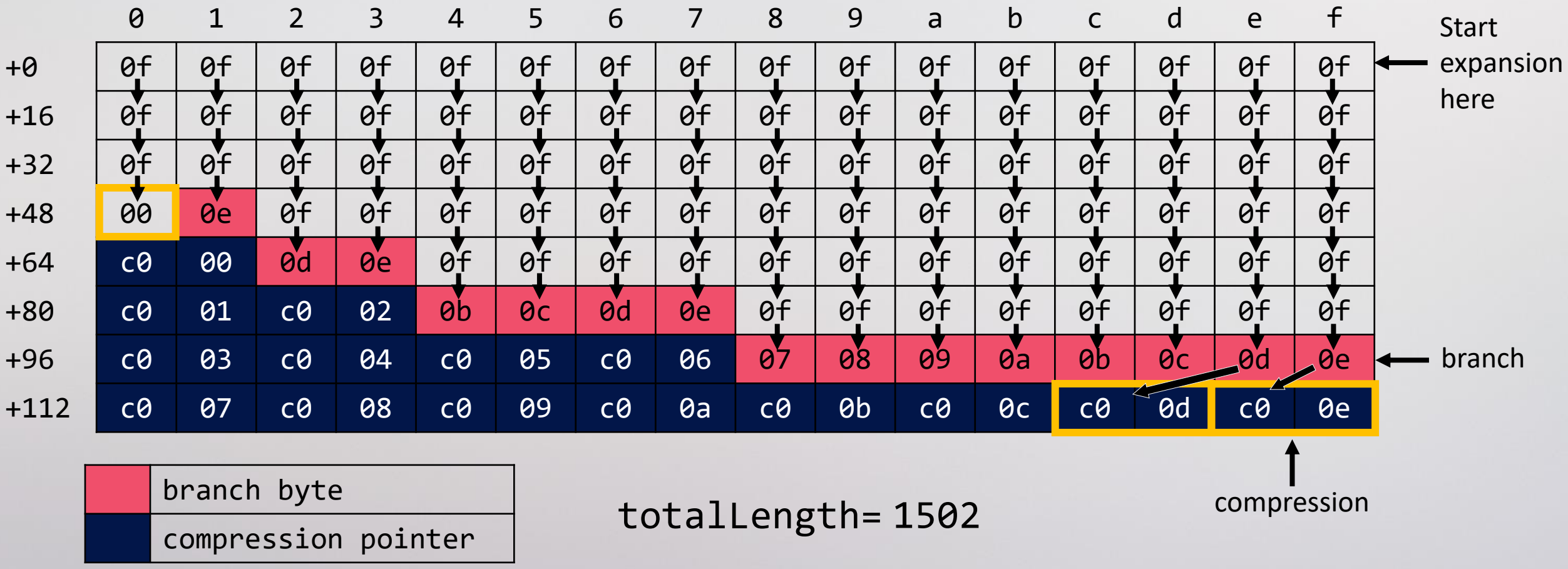
	branch byte
	compression pointer

totalLength= 0

Vulnerability #2: Integer Overflow

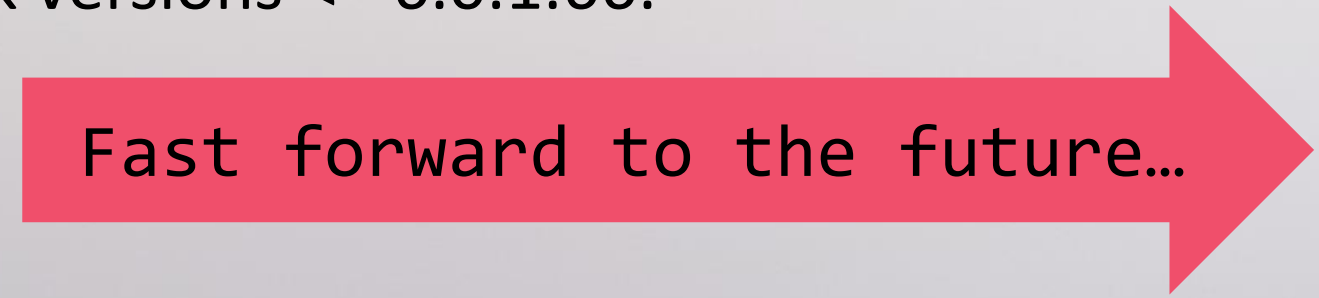


Vulnerability #2: Integer Overflow



Vulnerability #2: Integer Overflow

- To maximize the `totalLength`, we used the maximum label length 63 (`0x3f`) instead of `0x0f` shown in the example.
- Using this construction, we reached a name of length ~72700 bytes, overflowing the `totalLength` variable.
- We have an RCE candidate ☺
- Can be triggered in response to every query type supported - using CNAME records.
- Affects Treck versions <= 6.0.1.66.



Bad Fix

Bad Fix for the Read Out-Of-Bounds Vulnerability

Fixing the Read Out-Of-Bounds

```
if (RDLENGTH <= remaining_size) {
    labelEndPtr = resourceRecordAfterNamePtr + 10 + RDLENGTH;
    if (cacheEntryQueryType == DNS_TYPE_MX && rrtype == DNS_TYPE_MX) {
        addr_info = tfDnsAllocAddrInfo();
        if (addr_info != NULL && RDLENGTH >= 2) {
            /* copy preference value of MX record */
            memcpy(&addr_info->ai_mxpref, resourceRecordAfterNamePtr + 10, 2);
            /* compute the length of the MX hostname */
            labelLength = tfDnsExpLabelLength(resourceRecordAfterNamePtr+0xc,dnsHeaderPtr,labelEndPtr);
            addr_info->ai_mxhostname = NULL;
            if (labelLength != 0) {
                /* allocate buffer for the expanded name */
                asciiPtr = tfGetRawBuffer(labelLength);
                addr_info->ai_mxhostname = asciiPtr;
                if (asciiPtr != NULL) {
                    /* copy MX hostname to `asciiPtr` as ASCII */
                    tfDnsLabelToAscii(resourceRecordAfterNamePtr + 0xc, asciiPtr, dnsHeaderPtr, 1, 0);
                    /* ... */
                }
    /* ... */
```

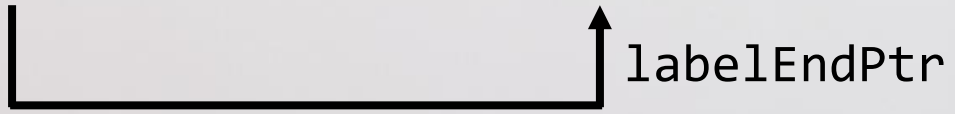
When `tfDnsExpLabelLength` reaches `labelEndPtr`, it stops processing (w/o error) and returns the current `totalLength`.

*Pseudo-code

Vulnerability #3: Bad RDLENGTH

- `labelEndPtr` is calculated based on the RDLENGTH field of the current resource record.
- RDLENGTH is attacker-controlled! Oops…

NAME	TYPE	CLASS	TTL	RDLENGTH	RDATA
example.com	MX	IN	86400	~~20~~ 7	0 0 4 s m t p 7 e x a m p l e 3 c o m 0



- `tfDnsExpLabelLength` returns 5;
- `tfDnsLabelToAscii` will copy the entire MX hostname.

Artifact: Memory Leak

```
if (RDLENGTH <= remaining_size) {
    labelEndPtr = resourceRecordAfterNamePtr + 10 + RDLENGTH;
    if (cacheEntryQueryType == DNS_TYPE_MX && rrtype == DNS_TYPE_MX) {
        addr_info = tfDnsAllocAddrInfo();
        if (addr_info != NULL && RDLENGTH >= 2) {
            /* copy preference value of MX record */
            memcpy(&addr_info->ai_mxpref, resourceRecordAfterNamePtr + 10, 2);
            /* compute the length of the MX hostname */
            labelLength = tfDnsExpLabelLength(resourceRecordAfterNamePtr+0xc,dnsHeaderPtr,labelEndPtr);
            addr_info->ai_mxhostname = NULL;
            if (labelLength != 0) {
                /* allocate buffer for the expanded name */
                asciiPtr = tfGetRawBuffer(labelLength);
                addr_info->ai_mxhostname = asciiPtr;
                if (asciiPtr != NULL) {
                    /* copy MX hostname to `asciiPtr` as ASCII */
                    tfDnsLabelToAscii(resourceRecordAfterNamePtr + 0xc, asciiPtr, dnsHeaderPtr, 1, 0);
                    /* ... */
                }
    /* ... */
```

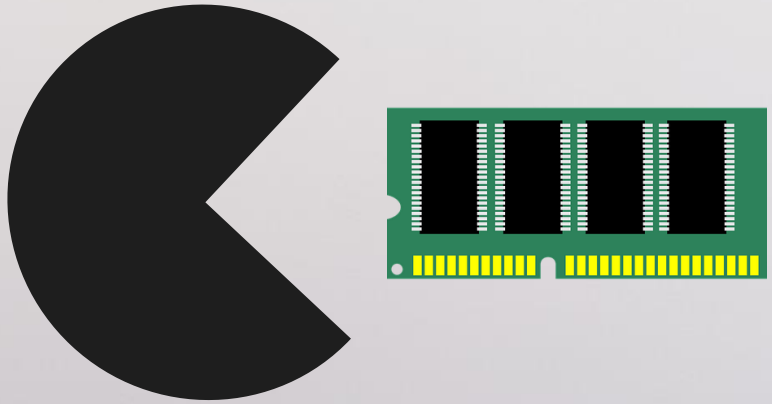
addrinfo structure is allocated

addr_info is not freed on error flows

*Pseudo-code

Artifact: Memory Leak

- An `addrinfo` structure can be leaked during MX parsing logic.
- Size of the leak `0x3c`.
- Comes in handy when exploiting heap vulnerabilities.



CVE-2020-11901: Summary

Treck Version	Vuln #1: Read OOB	Vuln #2: Integer Overflow	Vuln #3: Bad RDLENGTH	Artifact: Memory Leak
Old	✓	✓	✗	✓
New	✗	✓	✓	✓

✓ Affected

✗ Not affected

A device can be affected by one or more vulnerabilities depending on the exact version.

Exploitation

Exploiting CVE-2020-11901 on Schneider Electric UPS Device

Target Device

- Schneider Electric APC UPS network card

- Turbo186 (x86-based)
  - 16-bit Real Mode
  - No ASLR or DEP
  - Weird segmentation (shift 8 instead of 4)

- No debugging capabilities
  - Only limited crashdumps

Target Device

- Schneider Electric APC UPS network card
- Turbo186 (x86-based)
  - 16-bit Real Mode
  - No ASLR or DEP
  - Weird segmentation (shift 8 instead of 4)
- No debugging capabilities
  - Only limited crashdumps

```
Current stack at _SS:_SP 06d1: 057e

46f29a0000007c01 a0001285b81f0f04
0000000008850f04 aa053c0ee3c8b81f
0f0401003e014c00 c400a000d2057c0c
3e01b81f0f04ee75 0f04450001003e01
003833313900530a 530af80000000000
1800180000003f03 0000d100ffff0000
3900c205d1065004 2bc4000052042bc4
66c40f040f049a00 00007c010e068604
0e06920466c40800 00008c0059010f04
36068b088c008304 2bc46b002d018c00
00000000302e5151 0000000000000000
00000000a0750f04 6a0666018c005411
2d015f0000001b01 0b0378017c010200
f800ce0000000000 000000005f000000
6100000000000000 54112d017a064710
1b017c019a00ce00 000000000a079822
```

```
Register Set

AX = 0120
BX = 0120
CX = f000
DX = 07b6
SI = 017c
DI = 0000
BP = 05fe
CS = c046
DS = 001b
ES = 07b6
```

Vulnerability Recap

- **Primitive:** heap overflow via DNS response parsing
  - Only alpha-numeric characters are copied*
- We will exploit using “bad RDLENGTH” (#3)

Treck Version	Vuln #1: Read OOB	Vuln #2: Integer Overflow	Vuln #3: Bad RDLENGTH
Old	✓	✓	✗
New	✗	✓	✓

Exploitation Technique

- We can overflow through all DNS response types
- When the device boots*, 3 MX requests are transmitted
- **Interactivity** in exploits is advantageous
  - Allows easier shaping
- Crashing is favorable in order to reach **deterministic state**
  - No penalty* for crashing the network card

Overflow Target

- `tsDnsCacheEntry`
- Contains a list of `addrinfo` structs
  - `addrinfo` holds the contents of a DNS answer (name, IP address, …)
- Has many pointers and interesting fields
- Many references in DNS response parsing

tsDnsCacheEntry
`tsDnsCacheEntry *dnscNextEntryPtr`
`tsDnsCacheEntry *dnscPrevEntryPtr`
`addrinfo *dnscAddrInfoPtr`
`...`
`char *dnscRequestStr`
`int dnscErrorCode`
`...`
`short dnscFlags`
`...`

CNAME Processing

```
if (found_cname) {
  // Get the first addrinfo struct from `tsDnsCacheEntry`
  first_addr_info = t_dns_cache_entry->dnscAddrInfoPtr;
  if (first_addr_info) {
    // get CNAME name length from the packet
    length = tfDnsExpLabelLength(cname_rdata_ptr, packet_ptr, cname_rdata_end_ptr);
    if (length) {
      // allocate
      cname_label_buffer = tfGetRawBuffer(length);
      if (cname_label_buffer) {
        // copy to new buffer
        tfDnsLabelToAscii(cname_rdata_ptr, cname_label_buffer, packet_ptr, 1, 0);
        first_addr_info->ai_canonname = cname_label_buffer;
      }
    }
  }
}
```

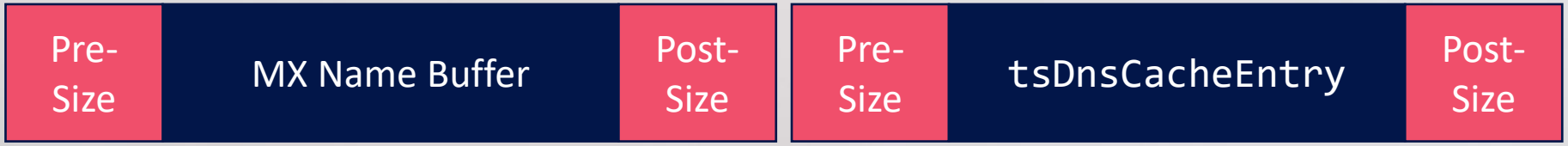

*Pseudo-code

Controlled Pointer Write

- We can write a 4-byte pointer
  - (Offset, Segment)
- To any alpha-numeric address
- Relatively strong exploitation primitive

Linear Overflow

- Overflow is from end of MX name buffer



Linear Overflow

- Overflow is from end of MX name buffer
- `tsDnsCacheEntry` allocated on DNS request creation
- Overflow is from MX name buffer, allocated on response
- `tsDnsCacheEntry` must be placed after MX name buffer

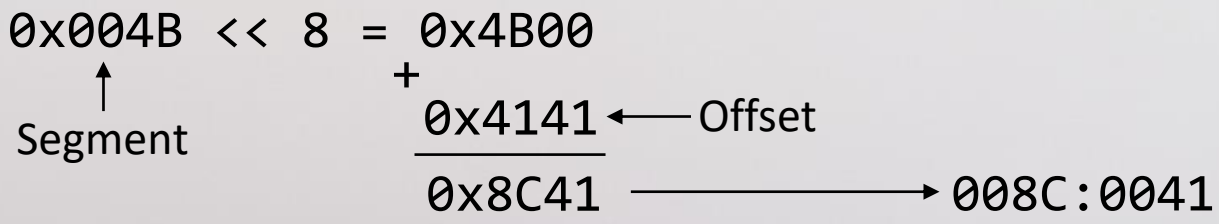
Heap Shaping

- A specific hole pattern would allow us to overflow `tsDnsCacheEntry`
  - Because of **tight-fit preference**

- Shaping using a memory leak artifact and name allocation

Pointer Write Limitations

- CNAME pointer written to address in `tsDnsCacheEntry`
- Overflow is only alpha-numeric, with trailing null-byte
  - Can be used as segment MSB
- Nothing placed in a strictly alpha-numeric address
- Combine two alpha-numeric bytes => Non-alpha-numeric segment



- This allows us to overwrite heap utility functions

Overwriting a Far Call

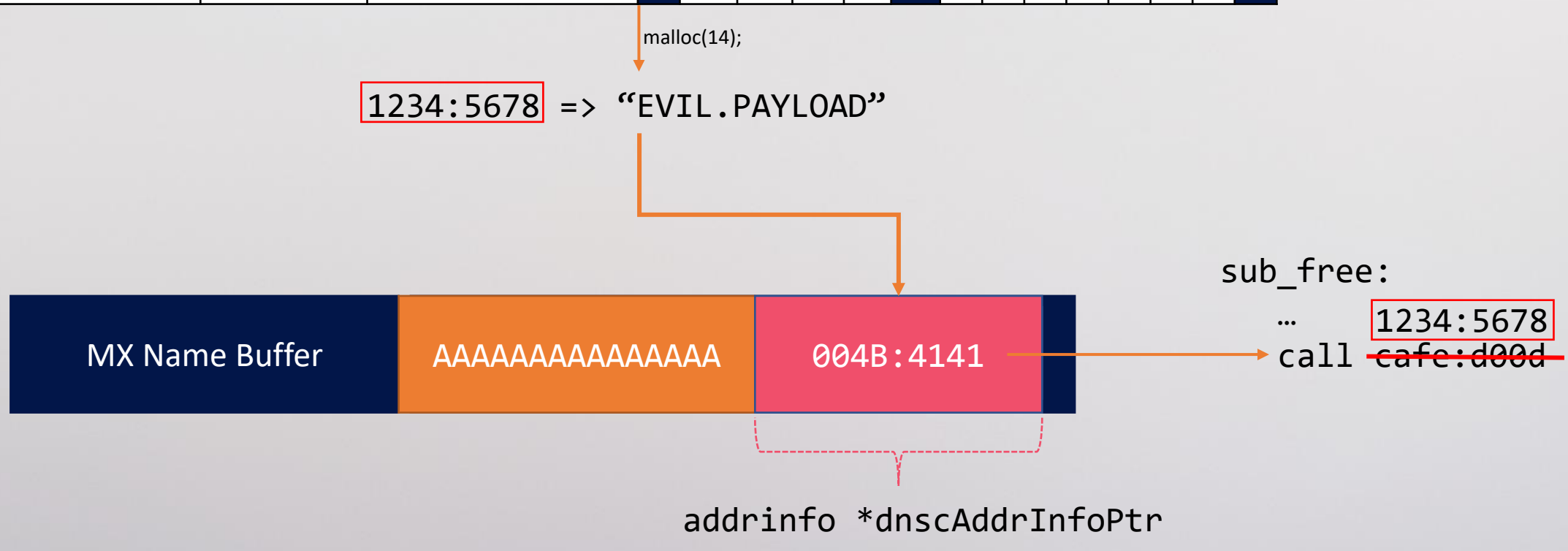
- Far calls in x86 are encoded with a pointer
- Patching a far call using our primitive results in the CNAME buffer being executed

- We patch a far call in free() error flow
  - Called when metadata corruption is detected

Recap

Recap

NAME	TYPE	RDLENGTH	RDATA
example	CNAME	14	4 E V I L 7 P A Y L O A D 0



Payload Trigger

- free() error flow will be triggered on overflown MX name free
- CNAME buffer contains crafted alpha-numeric shellcode
  - 2-stage decoder

Payload Trigger

- free() error flow will be triggered on overflown MX name free
- CNAME buffer contains crafted alpha-numeric shellcode
  - 2-stage decoder

- We have achieved arbitrary payload execution!

DEMO

Thanks for listening!

info@jsof-tech.com