black hat EUROPE 2020

DECEMBER 9–10 BRIEFINGS

FORESCOUT
Active Defense for the Enterprise of Things™

How Embedded TCP/IP Stacks Breed Critical Vulnerabilities

Daniel dos Santos, Stanislav Dashevskyi, Jos Wetzels, Amine Amri

Who we are

- Daniel dos Santos, Research Manager
- Stanislav Dashevskyi, Security Researcher
- Jos Wetzels, Security Researcher
- Amine Amri, Security Researcher

“At Forescout Research Labs we analyze the security implications of hyper connectivity and IT-OT convergence.”

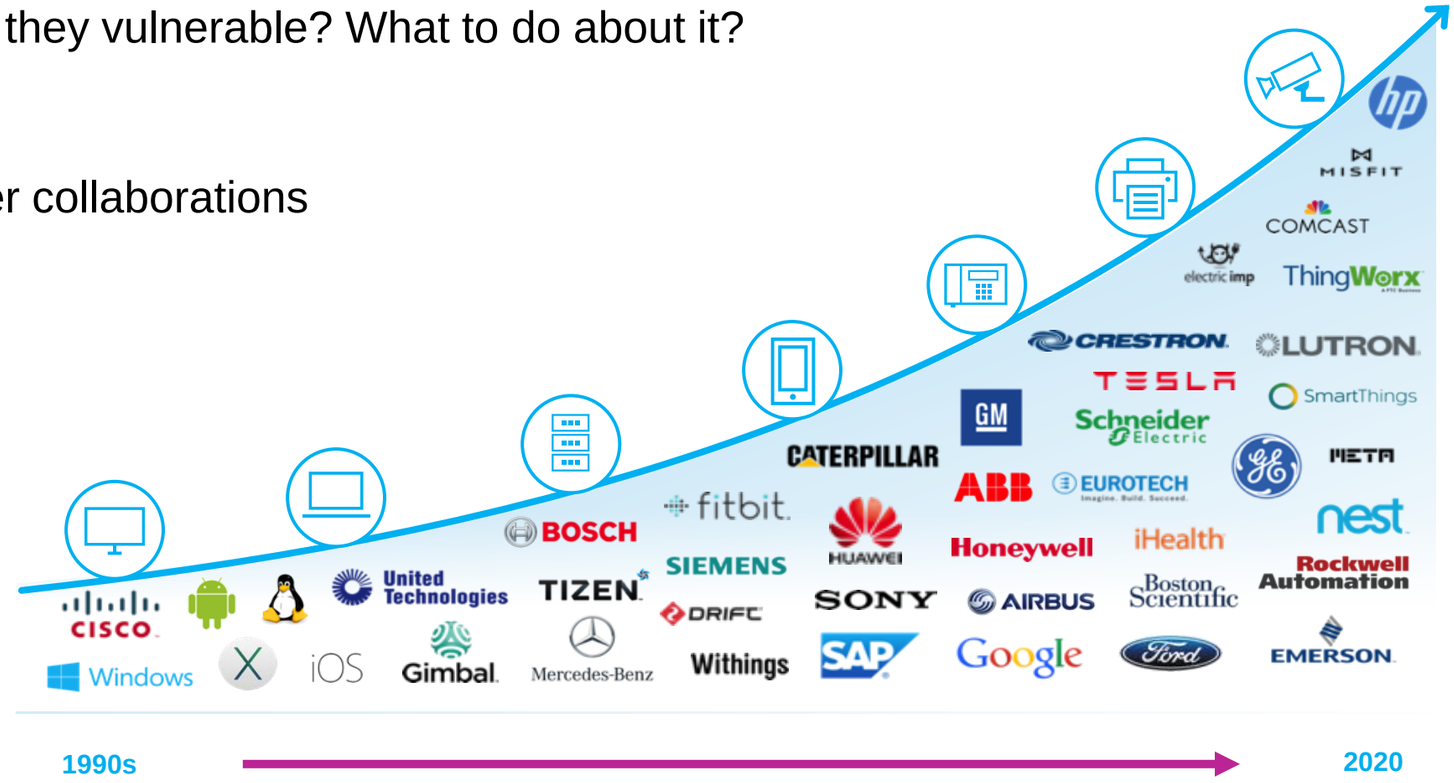
Outline

- Project Memoria
- AMNESIA:33 - vulnerabilities on open-source TCP/IP stacks
- Analysis of TCP/IP stack vulnerabilities
  - Affected components
  - Vulnerability types & Anti-patterns
  - Exploitability & Impact
- The consequences of TCP/IP stack vulnerabilities
- Conclusion & what comes next

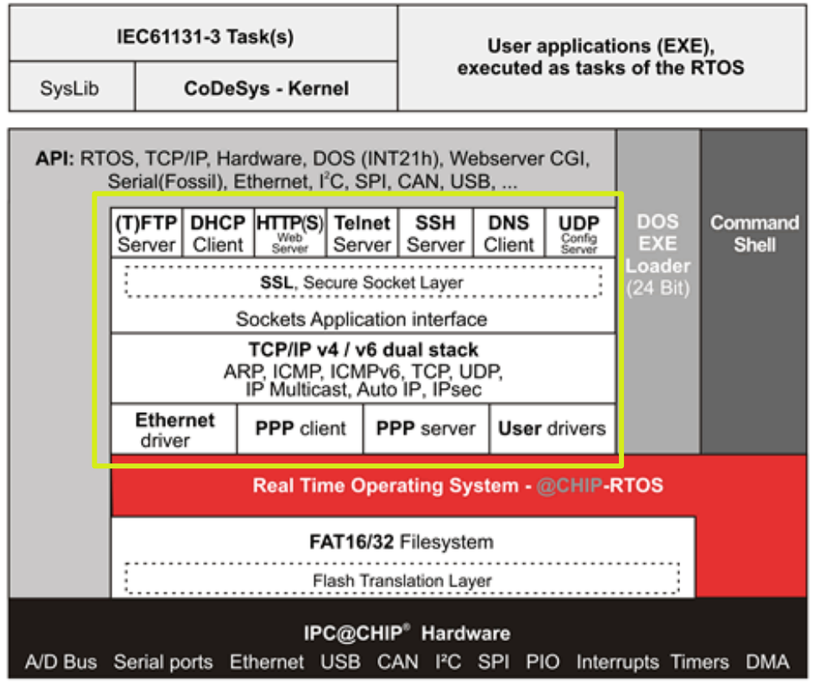
Project Memoria

Project Memoria

- Goal: large study of embedded TCP/IP stack security
  - Why are they vulnerable? How are they vulnerable? What to do about it?
  - Quantitative and qualitative
  - Forescout Research Labs and other collaborations



Embedded Systems Supply chain



http://smartbox.jinr.ru/doc/chip-rtos/software.htm

Components → **Devices** → **Connectivity** → **Applications / Services** → **System Integration** → **Distributer / Reseller** → **End User**

- MCU
- SoC
- Modem

- IP Cam
- Router
- ECU

- Cellular
- LPWAN
- Wi-Fi

- Library
- Daemon
- Platform

Why target protocol stacks?

- Wide deployment
- Vulnerabilities **trickle down** the supply-chain to many vendors

- Absence of *Software Bill of Materials (SBOM)*
- Vulnerabilities fixed in core stack **might never make it** to all OEM firmware

- Good chance of finding exploitable bugs

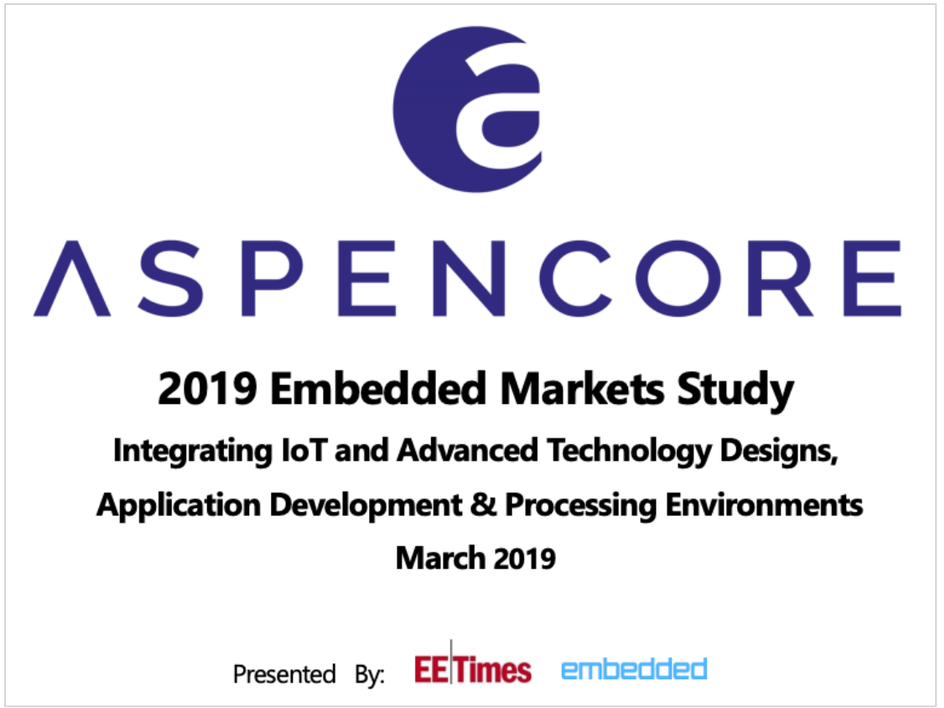
- Externally exposed, often run as privileged, low-level component

- Patching issues + Long lifespans + Broad trickle-down = **High vulnerability lifespan = High attacker ROI**

AMNESIA:33

AMNESIA:33 – a Project Memoria Study

- Lots of (publicly) unexamined popular stacks out there
  - NetX (Microsoft), NicheStack (HCC), Nucleus NET (Siemens), ...
- Why open source stacks?
  - **88%** of embedded projects reuse source code
  - **58%** use an open source RTOS (open source TCP/IP stacks often come bundled with them)
  - Use of comercial OSes declining and open-source increasing
  - **4%** of design time spent on security and privacy assessment



https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf

Research Methodology

Prior work

Year	TCP/IP Stack	Reference	# Vulns
2013	Microchip TCP/IP	https://www.securityfocus.com/bid/59603/info	1
2014	uIP, lwIP	https://www.kb.cert.org/vuls/id/210620	1
2017	RTCS TCP/IP	https://www.us-cert.gov/ics/advisories/ICSMA-17-250-02A	2
2017	picoTCP	https://nvd.nist.gov/vuln/detail/CVE-2017-1000210	1
2018	FreeRTOS+TCP	https://blog.zimperium.com/freertos-tcpip-stack-vulnerabilities-details	10
2019	Nucleus NET	https://us-cert.cisa.gov/ics/advisories/icsa-19-318-01	1
2019	Interpeak IPnet	https://us-cert.cisa.gov/ics/advisories/icsa-19-274-01	11
2020	InterNiche NicheStack	https://us-cert.cisa.gov/ics/advisories/icsa-20-105-08	1
2020	Treck TCP/IP	https://us-cert.cisa.gov/ics/advisories/icsa-20-168-01	19

Not many vulnerabilities, even on very popular stacks used for decades => Lack of public analysis

Analysis of prior work – anti-patterns*

- Shotgun & permissive parsing
  - Absence of bounds checks
  - Integer wraparounds
  - State confusion
- Unclear and conflicting RFCs
  - E.g., TCP Urgent pointer

```
ucProtocol = pxIPPacket->xIPHeader.ucProtocol;
/* Check if the IP headers are acceptable and if it has our destination. */
eReturn = prvAllowIPPacket( pxIPPacket, pxNetworkBuffer, uxHeaderLength );

if( eReturn == eProcessBuffer )
{
    if( uxHeaderLength > ipSIZE_OF_IPv4_HEADER )
    {
        /* All structs of headers expect a IP header size of 20 bytes
         * IP header options were included, we'll ignore them and cut them out
         * Note: IP options are mostly use in Multi-cast protocols */
        const size_t optlen = ( ( size_t ) uxHeaderLength ) - ipSIZE_OF_IPv4_HEADER;
        /* From: the previous start of UDP/ICMP/TCP data */
        uint8_t *pucSource = ( ( uint8_t * ) pxIPHeader ) + uxHeaderLength;
        /* To: the usual start of UDP/ICMP/TCP data at offset 20 from IP header */
        uint8_t *pucTarget = ( ( uint8_t * ) pxIPHeader ) + ipSIZE_OF_IPv4_HEADER;
        /* How many: total length minus the options and the lower headers */
        const size_t  xMoveLen = pxNetworkBuffer->xDataLength - optlen - ipSIZE_OF_IPv4_HEADER - ipSIZE_OF_ETH_HEADER;

        memmove( pucTarget, pucSource, xMoveLen );
        pxNetworkBuffer->xDataLength -= optlen;
    }
```

CVE-2018-16601 – no bounds checks on FreeRTOS+TCP

```
struct rarp_packet *__cdecl ipnet_eth_rarp_input(void *interface, void *a2)
{
  packet = (struct rarp_packet *)(*(_DWORD *)(a2 + 0x3C) + *(_DWORD *)(a2 + 0x18));
  if ( packet->hardware_type == 0x100               // Ethernet
       && packet->protocol_type == 8                // IPv4
       && packet->hardware_protocol_sizes == 0x406  // Hardware Size: 6, Protocol Size: 4
       && packet->opcode == 0x400                   // Reverse Reply
       && !(packet->sender_mac[0] & 1))
  {
    ipnet_timeout_cancel(*(_DWORD *)(*(_DWORD *)(a1 + 0x5C) + 0x10));
    packet = (struct rarp_packet *)ipnet_ip4_add_addr(interface, packet->target_ip, 0, 1, 1);
  }
  return packet;
}
```

CVE-2019-12262 – state confusion (unsolicited reply) on IPnet

* “The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them” – F. Momot et al.

Target selection

- Selection criteria:
  - Use in popular RTOSes (e.g., FreeRTOS, uC/OS)
  - More than 1k instances found online

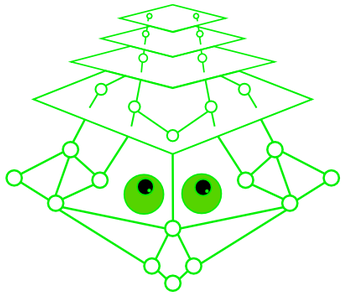
#	Stack	Description	Examples of OSes using it
1	lwIP	• Developed in 2000 by Adam Dunkels • Very popular as part of FreeRTOS or as a standalone stack (used by, e.g., Tesla and Philips)	LiteOS, RT-Thread, FreeRTOS
2	uIP	• Designed to be lighter than lwIP, released in 2001 (also by Adam Dunkels), popular base for forks • Development halted as a standalone project, continues as part of the Contiki OS	Contiki, Contiki-NG RT-Thread, FreeRTOS
3	Nut/Net	• IP stack used by NutOS, developed by the Ethernut project since 2002 • The project is currently under active development	NutOS
4	FNET	• Developed at Freescale (now NXP) in 2003 • Made public in 2009 and maintained by Andrey Butok	-
5	picoTCP	• Developed by Altran Intelligent Systems and made open source in 2013 • Continues to be developed as picoTCP-NG (no longer supported by Altran)	seL4, TRENTOS
6	CycloneTCP	• Developed by Oryx Embedded and distributed in source code form since 2013	-
7	uC/TCP-IP	• Developed by Micrium in 2002 and open sourced in February 2020 • uC/OS, which typically relies on the stack, is very popular	uC/OS-II, uC/OS-III, Cesium

Static and Dynamic Analysis

- Fuzzing = libFuzzer
  - White-box (targets are open-source) and coverage-guided
  - Complications for embedded targets
  - Challenging to achieve full coverage (e.g., stateful fuzzing, checksums) when used out-of-the box

https://llvm.org/docs/LibFuzzer.html

- Variant hunting = Joern – *thanks to Gabriele Acerbi*
  - Encode corpus of anti-patterns as code property graph queries in Scala
  - Search for anti-patterns in the new targets
  - Depends on queries, may leave out important code regions



https://joern.io/

- Manual analysis
  - Deep analysis of other points of interest left out before (e.g., memory operations, pointer arithmetic)

Negative (positive?) results

- 3 stacks with no vulnerabilities found
  - lwIP, CycloneTCP, uC/TCP-IP
- Selected anti-patterns not trivially present
  - Generally, consistent bounds checking and safe parsing
- Actively developed and supported
  - Projects have been contacted for other issues discovered, and they were very responsive

Positive & Negative results

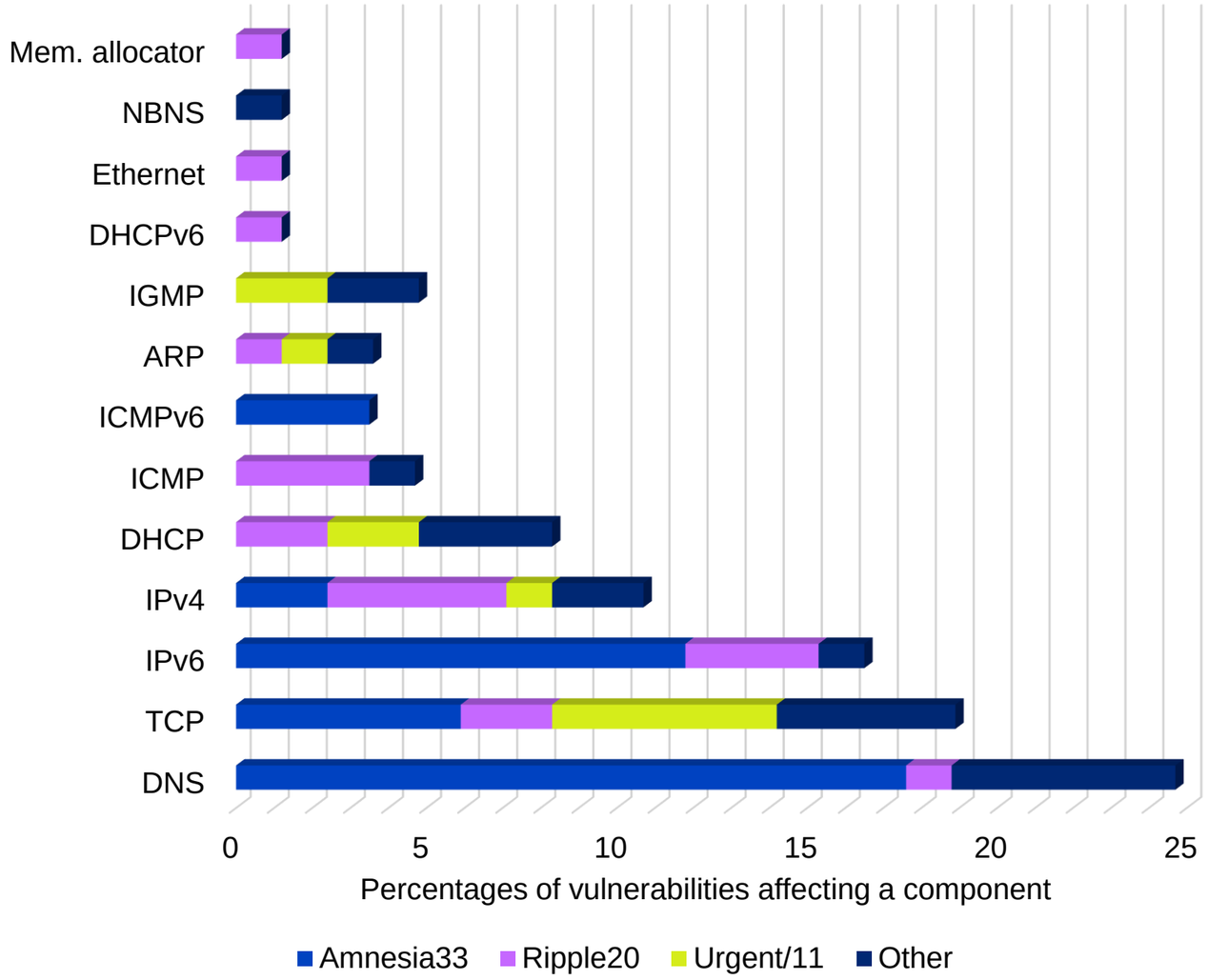
- 4 vulnerable stacks
- 7 affected components
  - DNS, IPv6/v4, TCP, ICMP, LLMNR, mDNS
- 4 potential impacts
  - Denial of Service
    - Crash
    - Infinite loops
  - DNS cache poisoning
  - Information leak
  - Remote code execution

Stack	Versions analyzed	Total vulnerabilities	Dynamic Analysis	Static Analysis
uIP	uIP 1.0, Contiki 3.0, Contiki-NG 4.5	13	6	7
picoTCP	picoTCP 1.7.0, picoTCP-NG 2.0.0	10	5	5
FNET	4.6.3	5	0	5
Nut/Net	5.1	5	0	5

Detailed Analysis

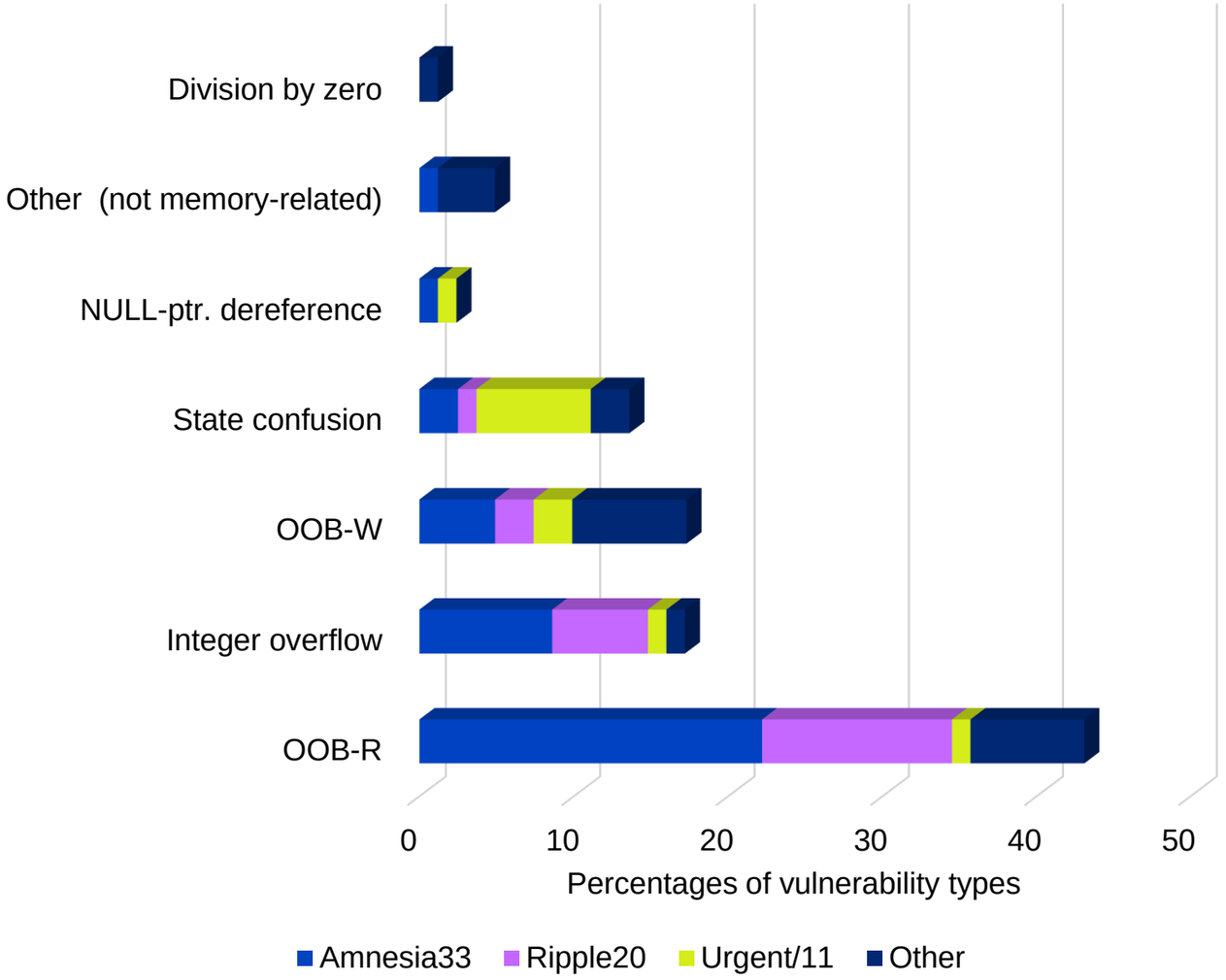
AMNESIA:33 and prior work

Affected Components



- TCP/IP vulnerabilities are independent of user application
- Protocol complexity remains good predictor of vulnerabilities

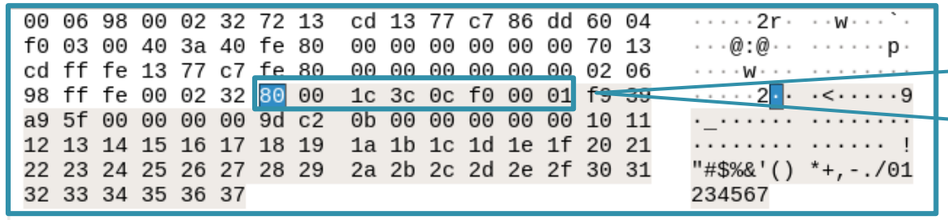
Vulnerability types



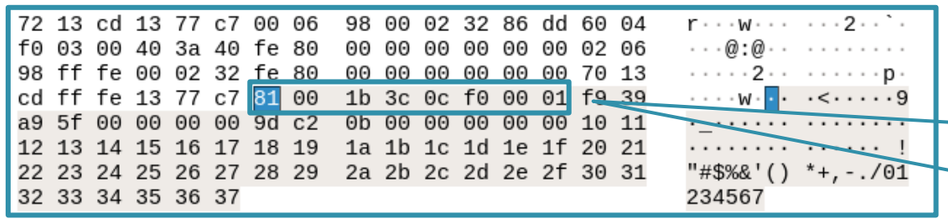
- OOB-R/W = unvalidated attacker-controlled length and offset fields
- Integer Overflow = unvalidated length and offset fields incorporated into assignments & comparison arithmetic
- State Confusion = unsolicited responses, permissive state machines, ambiguous RFCs

Example 1 – OOB-W and overflow

ICMPv6 echo request packet



ICMPv6 echo reply packet

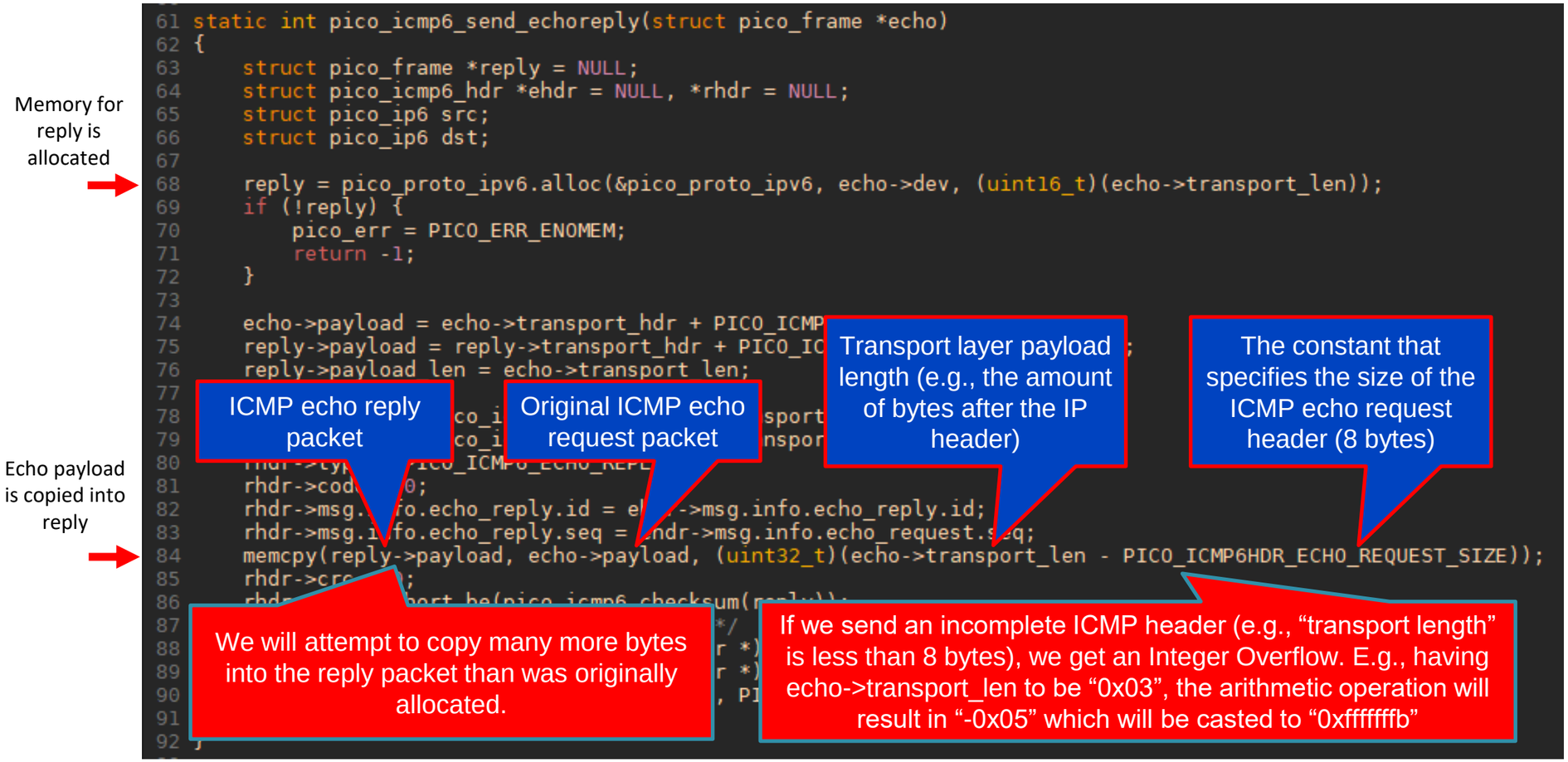


0x80 - type
0x00 - code
0x1c3c - checksum
0x0cf0 - identifier
0x0001 - sequence number

0x81 - type
0x00 - code
0x1b3c - checksum
0x0cf0 - identifier
0x0001 - sequence number

Some TCP/IP stacks will interpret any sequence of bytes that starts with 0x80 (ICMPv6) and/or 0x08 (ICMPv4) as an echo request

Example 1 – OOB-W and overflow




```
61 static int pico_icmp6_send_echoreply(struct pico_frame *echo)
62 {
63     struct pico_frame *reply = NULL;
64     struct pico_icmp6_hdr *ehdr = NULL, *rhdr = NULL;
65     struct pico_ip6 src;
66     struct pico_ip6 dst;
67
68     reply = pico_proto_ipv6.alloc(&pico_proto_ipv6, echo->dev, (uint16_t)(echo->transport_len));
69     if (!reply) {
70         pico_err = PICO_ERR_ENOMEM;
71         return -1;
72     }
73
74     echo->payload = echo->transport_hdr + PICO_ICMP
75     reply->payload = reply->transport_hdr + PICO_IC
76     reply->payload_len = echo->transport_len;
77
78                 co_i                sport
79                 co_i                nspor
80
81     rhdr->cod      0;
82     rhdr->msg.     fo.echo_reply.id = e     ->msg.info.echo_reply.id;
83     rhdr->msg.i    fo.echo_reply.seq =     ndr->msg.info.echo_request.     q;
84     memcpy(reply->payload, echo->payload, (uint32_t)(echo->transport_len - PICO_ICMP6HDR_ECHO_REQUEST_SIZE));
85     rhdr->cr       ;
86
87                                          */
88                                          r *)
89                                          r *)
90                                          , PI
91
92 }
```

CVE-2020-17443 on PicoTCP

New anti-patterns in AMNESIA:33

- IPv6 extension headers and options
  - The lengths of extension headers and/or options are not validated

- DNS domain name length
  - Domain name labels are parsed based on unvalidated length fields

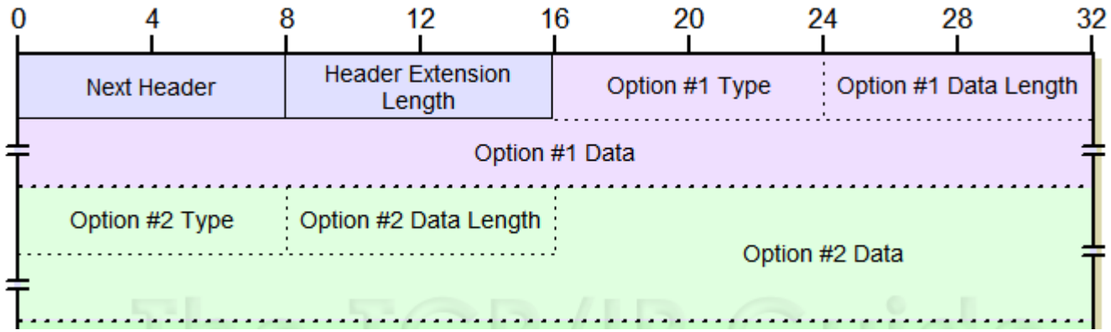
- DNS domain name NULL termination
  - The parsing code expects but doesn’t enforce that domain names are NULL-terminated

Example 2 – IPv6 extension header parsing

- **Destination Options** header is used for specifying packet delivery parameters for a destination **(0x60)**

- Consists of the **Next Header, Header Extension Length** fields, and a variable length **Options** field (provides some specific options for packet delivery or padding)

- The **Options** field has the following structure: **type**, **length**, and variable length **data**



http://www.tcpipguide.com/

Example 2 – IPv6 extension header parsing

```
1  static int pico_ipv6_process_destopt(struct pico_ipv6_exthdr *destopt, struct pico_frame *f, uint32_t opt_ptr)
2  {
3      uint8_t *option = NULL;
4      uint8_t len = 0, optlen = 0;
5      opt_ptr += (uint32_t)(2u);
6      option = ((uint8_t *)&destopt->ext.destopt) + sizeof(struct destopt_s);
7      len = (uint8_t)(((destopt->ext.destopt.len + 1) << 3) - 2);
8      while (len) {
9          optlen = (uint8_t)(*(option + 1) + 2);
10         switch (*option)
11         {
12         case PICO_IPV6_EXTHDR_OPT_PAD1:
13             break;
14
15         case PICO_IPV6_EXTHDR_OPT_PADN:
16             break;
17
18         case PICO_IPV6_EXTHDR_OPT_SRCADDR:
19             break;
20
21         default:
22             // THE CODE HERE IS OMITTED FOR BREVITY
23             break;
24         }
25         opt_ptr += optlen;
26         option += optlen;
27         len = (uint8_t)(len - optlen);
28     }
29     return 0;
30 }
```

Pointer to the first byte of an option (**type**) within the ext. header

Total length of the ext. header

A loop that is parsing the option data based on **type** and **length**

Here the parsing loop jumps to the next option (e.g., next **type** byte)

CVE-2020-17445 on PicoTCP

Example 2 – IPv6 extension header parsing

```
1  static int pico_ipv6_process_destopt(struct pico_ipv6_exthdr *destopt, struct pico_frame *f, uint32_t opt_ptr)
2  {
3      uint8_t *option = NULL;
4      uint8_t len = 0, optlen = 0;
5      opt_ptr += (uint32_t)(2u);
6      option = ((uint8_t *)&destopt->ext.destopt) + sizeof(struct destopt_s);
7      len = (uint8_t)(((destopt->ext.destopt.len + 1) << 3) - 2);
8      while (len) {
9          optlen = (uint8_t)(*(option + 1) + 2);
10         switch (*option)
11         {
12         case PICO_IPV6_EXTHDR_OPT_PAD1:
13             break;
14
15         case PICO_IPV6_EXTHDR_OPT_PADN:
16             break;
17
18         case PICO_IPV6_EXTHDR_OPT_SRCADDR:
19             break;
20
21         default:
22             // THE CODE HERE IS OMITTED FOR BREVITY
23             break;
24         }
25         opt_ptr += optlen;
26         option += optlen;
27         len = (uint8_t)(len - optlen);
28     }
29     return 0;
30 }
```

Attackers explicitly control **optlen**

Attackers implicitly control **option** pointer

Attackers implicitly control **len**

CVE-2020-17445 on PicoTCP

Example 2 – IPv6 extension header parsing

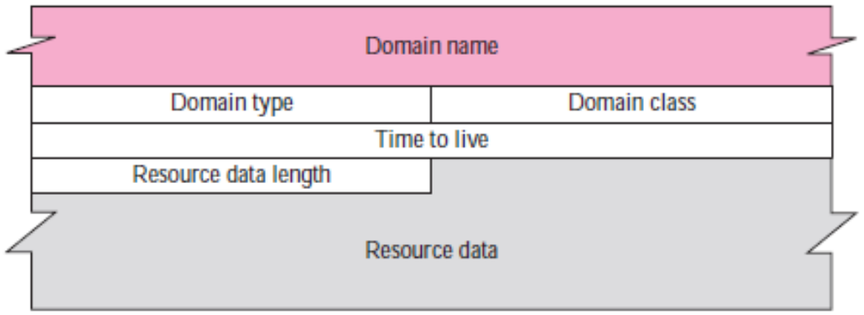
```c
static int pico_ipv6_process_destopt(struct pico_ipv6_exthdr *destopt, struct pico_frame *f, uint32_t opt_ptr)
{
    uint8_t *option = NULL;
    uint8_t len = 0, optlen = 0;
    opt_ptr += (uint32_t)(2u);
    option = ((uint8_t *)&destopt->ext.destopt) + sizeof(struct destopt_s);
    len = (uint8_t)(((destopt->ext.destopt.len + 1) << 3) - 2);
    while (len) {
        optlen = (uint8_t)(*(option + 1) + 2);
        switch (*option)
        {
        case PICO_IPV6_EXTHDR_OPT_PAD1:
            break;

        case PICO_IPV6_EXTHDR_OPT_PADN:
            break;

        case PICO_IPV6_EXTHDR_OPT_SRCADDR:
            break;

        default:
            // THE CODE HERE IS OMITTED FOR BREVITY
            break;
        }
        opt_ptr += optlen;
        option += optlen;
        len = (uint8_t)(len - optlen);
    }
    return 0;
}
```

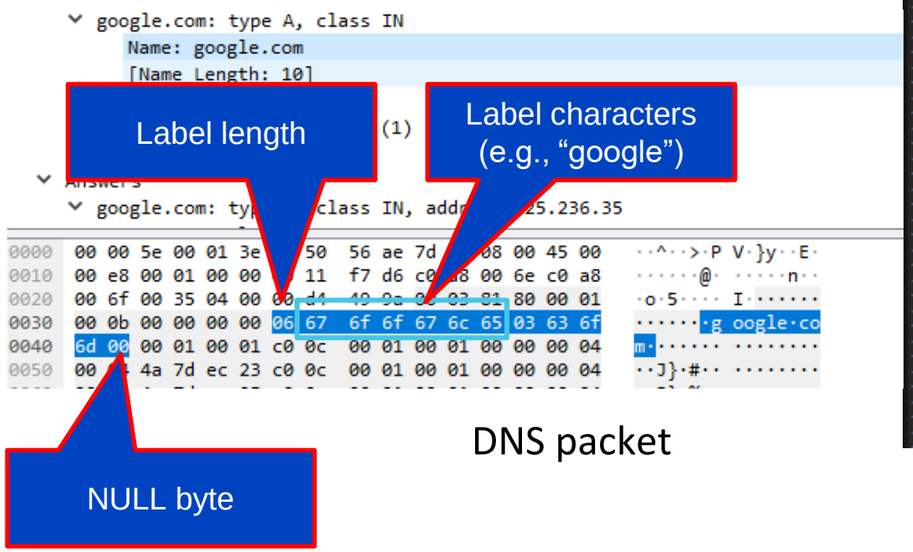
With this degree of control, attackers may achieve Out-of-bounds Read by shifting **option** into unmapped memory.

It is also possible to achieve “infinite loop”: **optlen** can be simply set to 0 via Integer Overflow during assignment

CVE-2020-17445 on PicoTCP

Example 3 – DNS domain name issues





DNS packet

```
168 static uint16_t ScanName(uint8_t * cp, uint8_t ** npp)
169 {
170     uint8_t len;
171     uint16_t rc;
172     uint8_t *np;
173
174     if (*npp) {
175         free(*npp);
176         *npp = 0;
177     }
178
179     if ((*cp & 0xC0) == 0xC0)
180         return 2;
181
182     rc = strlen((char *) cp) + 1;
183     np = *npp = malloc(rc);
184     len = *cp++;
185     while (len) {
186         while (len--)
187             *np++ = *cp++;
188         if ((len = *cp++) != 0)
189             *np++ = '.';
190     }
191     *np = 0;
192
193     return rc;
194 }
```

Pointer to the label length byte

Buffer into which the domain name will be copied

Do not process compressed names

Get the total domain name length using **strlen()**

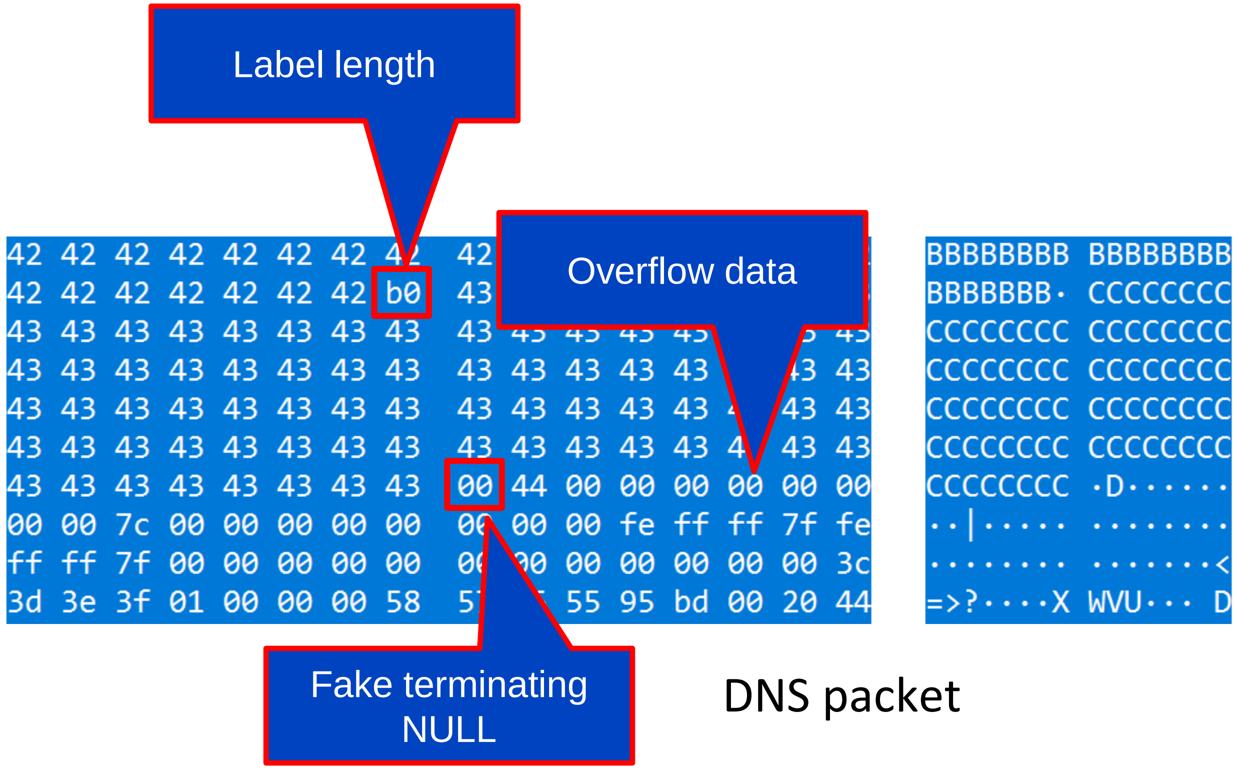
Allocate *npp based on the total domain name length

Iterate over individual labels and copy their contents into *npp

Return the total length of domain name

CVE-2020-25111 on Nut/Net

Example 3 – DNS domain name issues



DNS packet

```c
static uint16_t ScanName(uint8_t * cp, uint8_t ** npp)
{
    uint8_t len;
    uint16_t rc;
    uint8_t *np;

    if (*npp) {
        free(*npp);
        *npp = 0;
    }

    if ((*cp & 0xC0) == 0xC0)
        return 2;

    rc = strlen((char *) cp) + 1;
    np = *npp = malloc(rc);
    len = *cp++;
    while (len) {
        while (len--)
            *np++ = *cp++;
        if ((len = *cp++) != 0)
            *np++ = '.';
    }
    *np = 0;

    return rc;
}
```

Attackers explicitly control the allocation of ***npp**

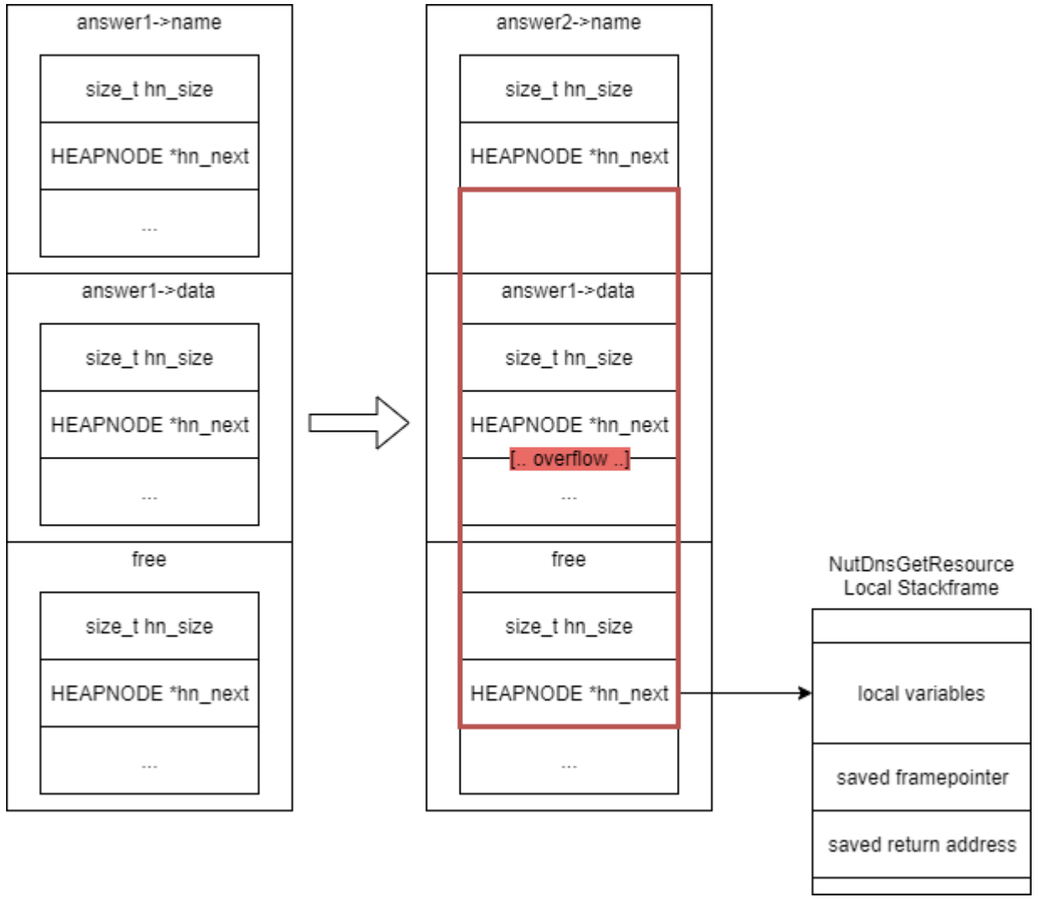
Attackers explicitly control the number of bytes to be written into ***npp (len)**

CVE-2020-25111 on Nut/Net

Example 3 – CVE-2020-2511 Exploitation*

- NutOS has a simple heap allocator
  - Singly-linked free list
  - Deterministic best-fit algorithm
  - Forward coalescing
  - Guards optional and static (*0xdeadbeef*)

- Heap shaping
  - Use resource record field sizes
  - Target node gets allocated in proper position before a free node

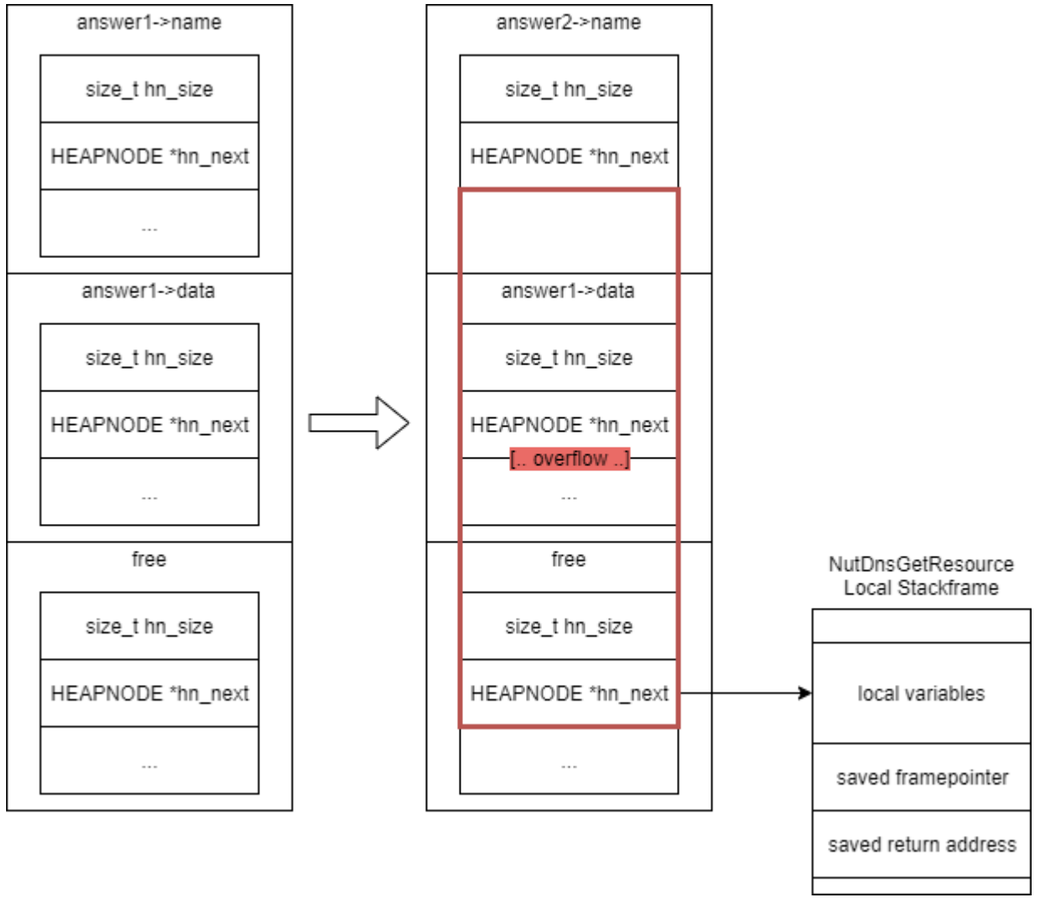


*Detailed writeup coming soon

Example 3 – CVE-2020-2511 Exploitation*

- Controlled allocation via corrupted free node
  - We control ***src***, ***dst*** & ***len*** now: write primitive
  - Overwrite local stackframe or parent (saved ***ret***)

- RCE
  - No exploit mitigations, no memory protections
  - Sleep ROP chain for cache coherency (ARM, etc.)
  - Redirect to shellcode



*Detailed writeup coming soon

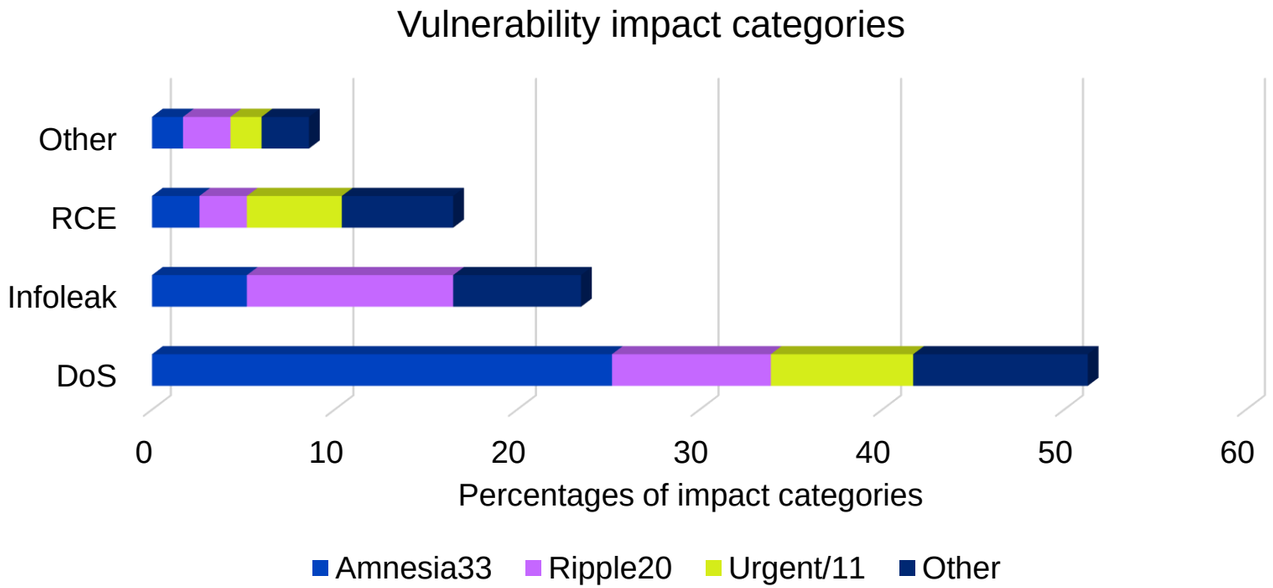
Example 3 – CVE-2020-2511 Exploitation*

- PoC developed against latest NutOS
  - We used Ethernut 5 development board as a target
  - The payload simply prints a string to UART, but it can be anything (manipulate IOs, extract key material, talk to other busses, e.g., I2C, CAN, …)

```
DRAM:  128 MiB
WARNING: Caches not enabled
Flash: 512 KiB
NAND:  1024 MiB
MMC:   mci: 0
DataFlash:AT45DB321
Nb pages:   8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0021000 to C0041FFF (RO) setup
Area 1: C0042000 to C00C5FFF (RO) uboot
Area 2: C00C6000 to C0359FFF (RO) kernel
Area 3: C035A000 to C03DDFFF (RO) nutos
Area 4: C03DE000 to C03FEFFF      env
SF: Detected AT45DB321D with page size 528 Bytes, total 16.5 MiB
In:    serial
Out:   serial
Err:   serial
Net:   macb0: PHY present at 0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0xc1e1)
macb0
Hit any key to stop autoboot:  0
## Starting application at 0x20000000 ...

[i] Initiating DNS Client ...
[i] Configuring Ethernet interface
[+] 192.168.0.94 ready


[!][!][!] PWNED PWNED PWNED [!][!][!]
```

*Detailed writeup coming soon

Exploitability & Impact



- Exploitability and concrete impact depend on:
  - Stack configuration – components/features enabled, buffer sizes, allocators…
  - Networking hardware & drivers – validation and filtering offloading
  - Target platform – hardware architecture and configuration (memory protection, organization & security features, fault & watchdog handling, etc.)

Exploitability & Impact: Examples

```
#if ! UIP_ARCH_CHKSUM
```

```
#if UIP_CONF_IPV6
  upper_layer_len = (((u16_t)(BUF->len[0]) << 8) + BUF->len[1]);
#else /* UIP_CONF_IPV6 */
  upper_layer_len = (((u16_t)(BUF->len[0]) << 8) + BUF->len[1]) - UIP_IPH_LEN;
#endif /* UIP_CONF_IPV6 */

  /* First sum pseudoheader. */

  /* IP protocol and length fields. This addition cannot carry. */
  sum = upper_layer_len + proto;
  /* Sum IP source and destination addresses. */
  sum = chksum(sum, (u8_t *)&BUF->srcipaddr[0], 2 * sizeof(uip_ipaddr_t));

  /* Sum TCP header and data. */
  sum = chksum(sum, &uip_buf[UIP_IPH_LEN + UIP_LLH_LEN],
               upper_layer_len);
```

CVE-2020-13987: uiP OOB-R

```
/* Temporary variables. */
uint8_t uip_acc32[4];
static uint8_t c, opt;
```

```
  /* All other options have a length field, so that we easily
     can skip past them. */
  if(uip_buf[UIP_TCPIP_HLEN + UIP_LLH_LEN + 1 + c] == 0) {
    /* If the length field is zero, the options are malformed
       and we don't process them further. */
    break;
  }
  c += uip_buf[UIP_TCPIP_HLEN + UIP_LLH_LEN + 1 + c];
```

CVE-2020-13988: uiP infinite loop

- Checksum must be not offloaded to platform/driver
- Memory protection needs to be enabled
- Fault handler determines the nature of DoS (reset, hang, etc.)

- Depending on task scheduler, entire system may hang
- If no watchdog is present, the system will hang indefinitely
- Otherwise, only network functionality may be affected, and the system may reset after some time

Consequences

Identifying affected systems

- 3 data sources
  - Device documentation
  - Online instances
  - Fingerprints on proprietary repository
- ~150 potentially affected vendors
  - Popular components and devices (e.g., WiFi SoCs, network switches) are shipped in the millions
  - Other components (e.g., RTUs) are much harder to estimate

EMBEDDED COMPONENTS
EXAMPLES: SOCS, CONNECTIVITY MODULES, OEM BOARDS

NETWORK & OFFICE
EXAMPLES: PRINTERS, ROUTERS, SERVERS

CONSUMER IOT
EXAMPLES: SMART PLUGS, SENSORS

OT
EXAMPLES: ACCESS CONTROL, IP CAMERAS, PROTOCOL GATEWAYS, HVAC

Many of these vulnerabilities affect components high up in the supply chain and trickle down to many different vendors and verticals

A supply chain example with AMNESIA:33

TCP/IP stack
(Vendor A)

Operating System
(Vendor B)

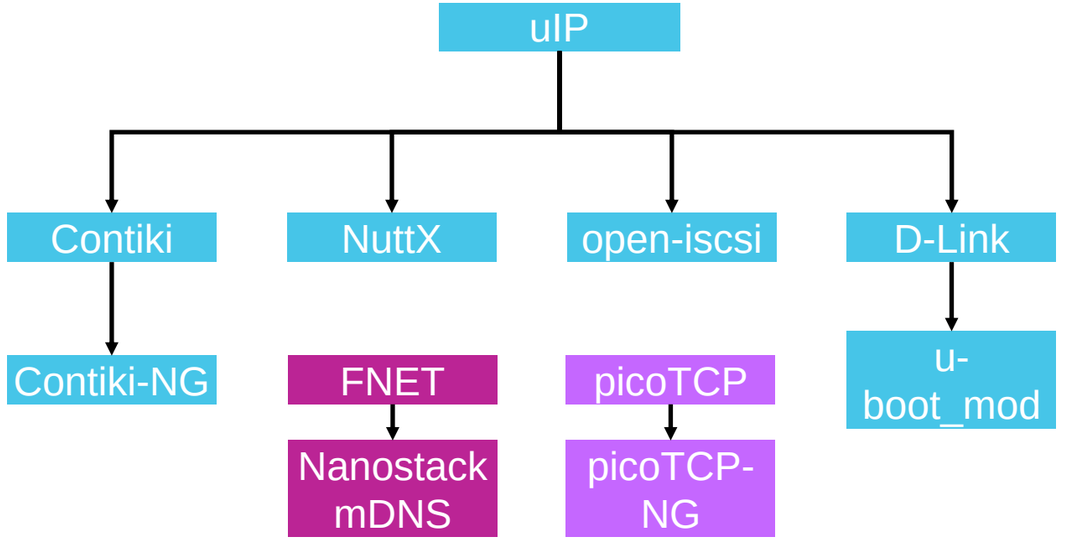
Network Management Card
(Vendor C)

UPS
(Vendor D)

Open-source challenges

Vulnerable code spreads widely

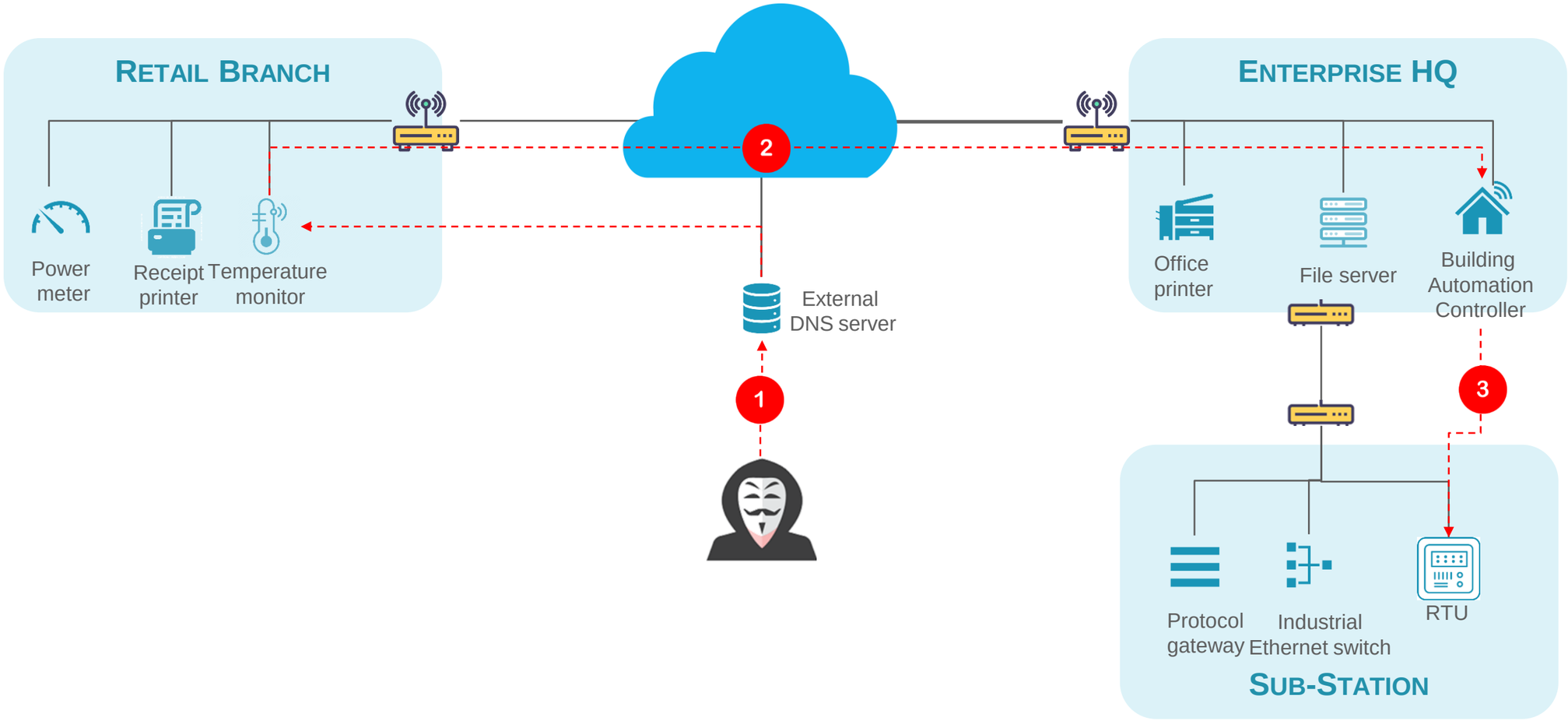
- Starts by forking and then evolves independently
- Sometimes only a single component is used
- Different versions of a stack might be forked into different projects



Who is responsible for fixes?

- We disclosed all vulnerabilities to CERT/CC and ICS-CERT, who got the help of GitHub's security team and other coordinating agencies (e.g., BSI, JPCERT)
- Despite effort, no official patches for the original uIP, Contiki, and PicoTCP, only their –NG versions
- Some vendors are implementing their own patches => more code fragmentation
- All of these will need to trickle down to OEMs, product vendors and end users

New attack scenarios



1 Initial Access 2 Lateral Movement 3 Impact

Key takeaways

- TCP/IP stacks are vulnerable across the board
  - Some resilient stacks, but rule of thumb is many bugs
- Vulnerabilities from well-known bad software development practices
- Bugs in several components but they tend to correlate with protocol complexity
- IoT risk management is difficult
  - Vulnerabilities affect hundreds of vendors and millions of devices
  - Exploitability & impact are highly device-specific
  - Complex supply chains make it challenging to determine which devices are affected and to eradicate the vulnerabilities

Mitigation

- It is difficult to identify assets on the network that are vulnerable
  - Requires specific knowledge of how a stack behaves (accurate fingerprinting)
- Patching may be impossible
  - Vendors may not know, not patch, or even be out of business
- Some specific recommendations:
  - Disable or block IPv6 when not needed
  - Rely on internal DNS servers for critical devices
  - Monitor the network for anomalous packets
  - Segment the network

What comes next...

- Vulnerabilities resulting from protocol complexity and external dependencies in multiple stacks
  - In one case, vulnerabilities in 6 stacks
  - In another, vulnerabilities in 8 stacks
  - Both cases are reminiscent of issues seen in the 90s & early 00s
- Vulnerabilities in popular closed-source stacks used in several critical devices
  - Already 8 CVEs in 2 stacks
  - Some popular stacks have *highly vulnerable, non robust and non RFC compliant code*
- Discussions about the current disclosure process and how to scale it to the IoT supply chain
- Recommendations about mitigation

Thank you!

Daniel.dosSantos@forescout.com

Stanislav.Dashevskyi@forescout.com

Jos.Wetzels@forescout.com

Amine.Amri@forescout.com

https://www.forescout.com/forescout-research-labs/