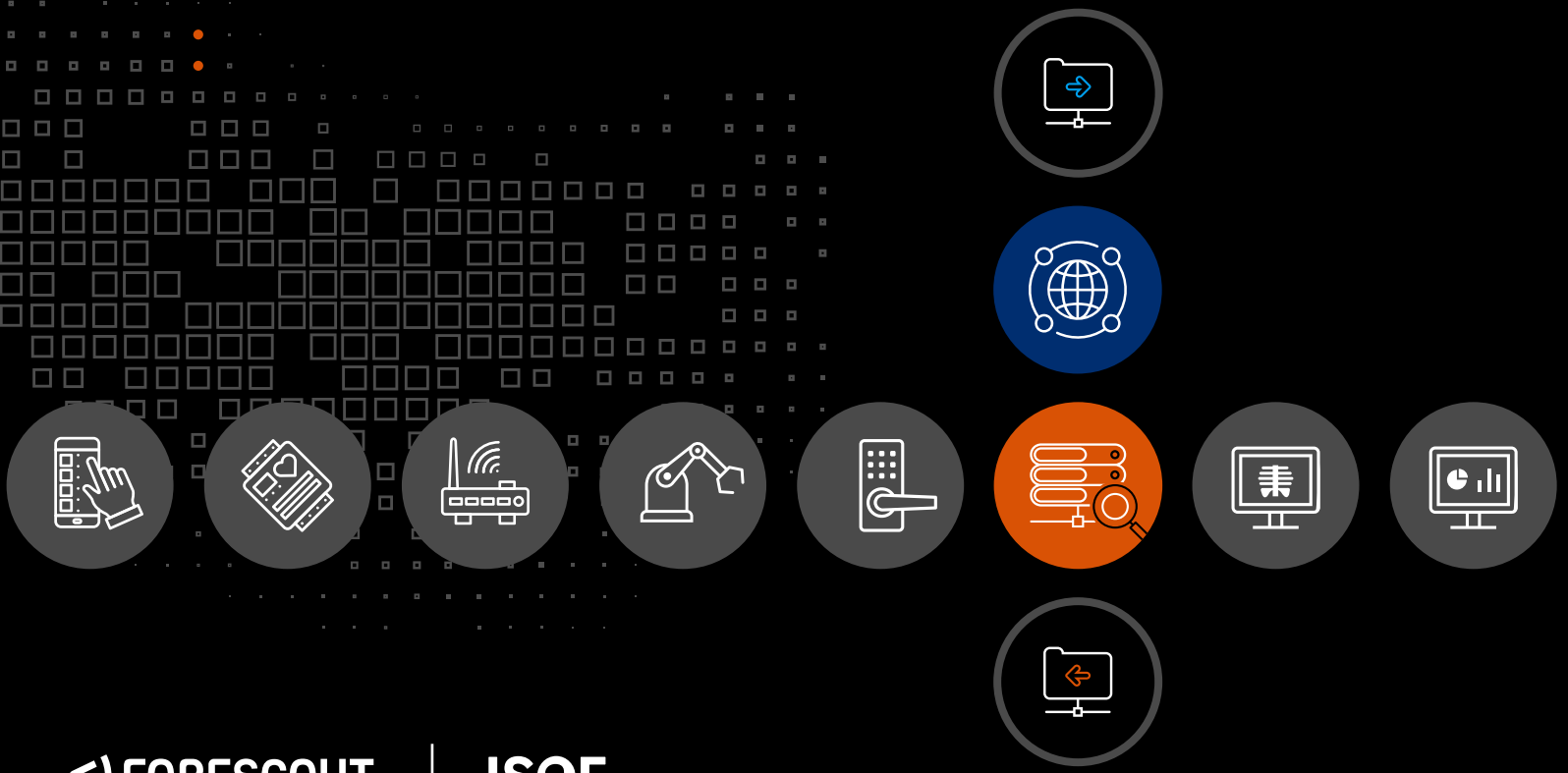
FORESCOUT
Active Defense for the Enterprise of Things™



FORESCOUT | JSOF

NAME:WRECK

Breaking and fixing DNS implementations

Published by Forescout Research Labs & JSOF

Forescout Research Labs
Daniel dos Santos
Stanislav Dashevskyi
Amine Amri
Jos Wetzels

JSOF
Shlomi Oberman
Moshe Kol

Contents

1. Executive Summary 3
2. Main Findings 5
2.1. Analyzed stacks and new vulnerabilities 6
3. Exploitation 11
3.1. About DNS message parsing 12
3.2. Technical details 12
4. An attack scenario leveraging NAME:WRECK 16
5. The impact of NAME:WRECK 17
6. It's You Again: Recurrent Anti-Patterns 21
6.1. AP#1 – Lack of TXID validation, insufficiently random TXID and source UDP port 21
6.2. AP#2 – Lack of domain name character validation 21
6.3. AP#3 – Lack of label and name lengths validation 22
6.4. AP#4 – Lack of NULL-termination validation 22
6.5. AP#5 – Lack of the record count fields validation 22
6.6. AP#6 – Lack of domain name compression pointer and offset validation 22
7. Mitigation Recommendations 24
8. Conclusions and Final Remarks 26

1. Executive summary

- In the third study of Project Memoria – NAME:WRECK – Forescout Research Labs and JSOF Research Labs joined forces to understand underlying problems related to Domain Name System (DNS) implementations, to disclose a set of 9 vulnerabilities affecting 4 popular TCP/IP stacks and to propose solutions for the community.
- The new vulnerabilities appear in well-known IT software (FreeBSD) and in popular IoT/OT firmware, such as Siemens' Nucleus NET. FreeBSD is widely known to be used for high-performance servers in millions of IT networks, including major websites such as Netflix and Yahoo. FreeBSD is also the basis for other well-known open-source projects. Nucleus NET has been used for decades in several critical OT and IoT devices.
- Not all devices running Nucleus RTOS or FreeBSD are vulnerable to NAME:WRECK. However, if we conservatively assume that 1% of the more than 10 billion deployments are vulnerable, **we can estimate that at least 100 million devices are impacted by NAME:WRECK.**
- The new vulnerabilities **allow for either Denial of Service or Remote Code Execution.** The widespread deployment and often external exposure of vulnerable DNS clients leads to a dramatically increased attack surface.
- NAME:WRECK illustrates the security cost of RFC complexity. We analyzed the implementation of DNS message compression in 7 new TCP/IP stacks and **found that 50% of them are vulnerable.** This is in addition to similar vulnerabilities in previous research (one in Ripple20 and two in AMNESIA:33) and other major vulnerabilities affecting DNS servers (SIGRed, DNSpooq, and several others disclosed over the years).
- General recommended mitigations for NAME:WRECK include **limiting the network exposure of critical vulnerable devices** via network segmentation, relying on internal DNS servers and patching devices whenever vendors release advisories.
- Of particular interest is that **to exploit NAME:WRECK vulnerabilities, an attacker should adopt a similar procedure for any TCP/IP stack.** This means that the same detection technique used to identify exploitation of NAME:WRECK also will work to detect exploitation on other TCP/IP stacks and products that we could not yet analyze.
- As part of the NAME:WRECK disclosure, **Forescout Research Labs shares with the cybersecurity community the following artifacts:**
  - This report, in which we **discuss six DNS anti-patterns** (implementation problems common in different TCP/IP stacks) and **provide researchers and developers around the world with tools and knowledge** enabling them to tackle the issue in other stacks.
  - An updated open-source script to identify possible vulnerable devices on a network.
  - A library of open-source Joern queries to be used by researchers and software developers to (partially) automate the finding of DNS-related vulnerabilities.
  - Samples of malicious traffic captures (available upon request) to be used by researchers and security analysists to test their intrusion detection systems.
  - A draft of an informational RFC discussing the identified anti-patterns to guide developers in avoiding making the same mistakes while writing future DNS implementations.
- This research is further proof that DNS protocol complexity leads to several vulnerable implementations and that **the community should act to fix a problem that we believe is more widespread of what we currently know.**

INFORMATIONAL

A Recap on TCP/IP stacks and Project Memoria

A TCP/IP stack is a piece of software that implements basic network communication for all IP-connected devices, including Internet of Things (IoT), operational technology (OT) and information technology (IT). Not only are TCP/IP stacks widespread; they also are notoriously vulnerable due to (i) codebases created decades ago and (ii) an attractive attack surface, including protocols that cross network perimeters and lots of unauthenticated functionality.

Noticing the impact of these foundational components, Forescout Research Labs has launched Project Memoria with the goal of collaborating with industry peers and research institutes to provide the cybersecurity community with the largest study on the security of TCP/IP stacks.

The latest examples of TCP/IP stack vulnerabilities include:

- Ripple20, a set of 19 vulnerabilities on the Treck TCP/IP stack released by JSOF in June 2020. Forescout Research Labs worked in close collaboration with JSOF to identify vendors and devices potentially affected by Ripple20.
- AMNESIA:33, a set of 33 vulnerabilities affecting 4 open-source TCP/IP stacks released in December 2020 by Forescout Research Labs.
- NUMBER:JACK, a set of 9 vulnerabilities affecting the Initial Sequence Number (ISN) implementation in 9 TCP/IP stacks disclosed in February 2021 by Forescout Research Labs.
- NAME:WRECK, discussed in this report, a set of 9 vulnerabilities affecting DNS clients of 4 TCP/IP stacks disclosed by Forescout Research Labs and JSOF. The vulnerabilities included in NAME:WRECK range in potential impact from Denial of Service to Remote Code Execution.

A note on the title of this report

"NAME:WRECK" refers to how the parsing of domain names can break – "wreck" – DNS implementations in TCP/IP stacks, leading to denial of service or remote code execution. However, this research focuses not only on the "breaking" part, but also on "fixing" these types of issues by finding and patching similar vulnerabilities in other stacks as well as avoiding the identified anti-patterns in future implementations.

2. Main Findings

Domain names are character strings that identify assets on the internet. The Domain Name System (DNS), informally known as the "phonebook of the internet," is a decentralized system and protocol created by Paul Mockapetris in 1983 that allows a requesting device to resolve desired domain names (such as "google.com") to specific IP addresses (such as "172.217.6.78") by querying a hierarchy of servers (such as Google's Public DNS).

Recently, there have been major vulnerabilities on DNS implementations that raised attention to this protocol as an important attack vector, such as SIGRed (CVE-2020-1350) allowing attackers to take over machines running the Windows DNS server and SAD DNS (CVE-2020-25705), which revived the DNS cache poisoning attack that can redirect millions of devices to attacker-controlled domains. The most recent example of major vulnerability on a DNS implementation is DNSpooq, a set of 7 critical CVEs affecting the DNS forwarder dnsmasq, which is used by major networking vendors to cache the results of DNS requests.

This kind of research shows that DNS is a complex protocol that tends to yield vulnerable implementations, and these vulnerabilities can often be leveraged by external attackers to take control of millions of devices simultaneously.

One particularly interesting class of vulnerabilities in DNS implementations is related to a protocol feature called "message compression." Since DNS response packets often include the same domain name or a part of it several times, RFC 1035 ("Domain Names – Implementation and Specification") specifies a compression mechanism to reduce the size of DNS messages in its section 4.1.4 ("Message compression"). This type of encoding is used not only in DNS resolvers but also in multicast DNS (mDNS), DHCP clients as specified in RFC 3397 ("Dynamic Host Configuration Protocol (DHCP) Domain Search Option") and IPv6 router advertisements as specified in RFC8106 ("IPv6 Router Advertisement Options for DNS Configuration"). Also, while some protocols do not officially support compression, many implementations still do support it because of code reuse or a specific understanding of the specifications.

DNS compression is neither the most efficient compression method nor the easiest to implement. As evidenced by the historical vulnerabilities shown in Table 1, this compression mechanism has been problematic to implement for 20 years on a diverse range of products, such as DNS servers, enterprise devices (e.g., the Cisco IP phone) and, more recently, the TCP/IP stacks Treck, uIP and PicoTCP.

Table 1 – 20 years of vulnerabilities related to DNS message compression

#	Vulnerability	Affected Products	Year
1	CVE-2000-0333	tcpdump, ethereal	2000
2	CVE-2002-0163	Squid	2002
3	CVE-2004-0445	Symantec DNS client	2004
4	CVE-2005-0036	Cisco IP Phone+	2005
5	CVE-2006-6870	Avahi	2006
6	CVE-2011-0520	MaraDNS	2011
7	CVE-2017-2909	Mongoose	2017
8	CVE-2018-20994	TrustDNS	2018
9	CVE-2020-6071	VLC	2020
10	CVE-2020-6072	VLC	2020
11	CVE-2020-12663	Unbound	2020
12	CVE-2020-11901	Treck TCP/IP stack (Ripple20)	2020
13	CVE-2020-24335	uIP TCP/IP stack (AMNESIA:33)	2020
14	CVE-2020-24339	PicoTCP TCP/IP stack (AMNESIA:33)	2020

2.1. Analyzed stacks and new vulnerabilities

While working on Ripple20 and AMNESIA:33, we had already found and disclosed three vulnerabilities related to message compression (see Table 1). During that research, we hypothesized that this type of vulnerability could represent a general problem that is common to other stacks as well. For this reason, we decided to focus on other TCP/IP stacks. Including some of our previous work and this research, we evaluated 15 stacks for message compression vulnerabilities: 1 stack under Ripple20 (Treck), 7 stacks under AMNESIA:33 (uIP, PicoTCP, FNET, Nut/Net, lwIP, cycloneTCP and uC/TCP-IP) and 7 new stacks under NAME:WRECK (FreeBSD's DHCP, IPnet, NetX, Nucleus NET, FreeRTOS+TCP, OpenThread and Zephyr). **We found 7 of the analyzed stacks to be vulnerable.**

Table 2 lists all the stacks analyzed for message compression vulnerabilities under Ripple20, AMNESIA:33 and NAME:WRECK, as well as whether or not they are vulnerable. **As shown in the table, Treck TCP/IP, uIP, PicoTCP, FreeBSD, IPNet, NetX and Nucleus NET are vulnerable** to the DNS compression bug. FNET, cycloneTCP, uC/TCP-IP, FreeRTOS+TCP, Zephyr and OpenThread were found to implement message compression securely, hence, not to be vulnerable. Nut/Net and lwIP did not support message compression, which makes them not vulnerable by design.

Table 3 focuses on the new vulnerabilities found under NAME:WRECK. As shown in Table 3, these vulnerabilities can be exploited by attackers to achieve Remote Code Execution (RCE) via out-of-bounds write or Denial of Service (DoS) via out-of-bounds read.

Table 2 – Overview of the TCP/IP stacks scrutinized for DNS message compression vulnerabilities. Rows are colored according to whether the stack is vulnerable: 🟡 yellow for known vulnerable from previous research, 🔴 red for found vulnerable in NAME:WRECK and 🟢 green for not vulnerable.

Stack	Description	Versions Analyzed	Vulnerable	Research
Treck TCP/IP	TCP/IP stack actively developed by Treck Inc. since 1997 for real-time embedded devices. The stack is also known as Elmic KASAGO in Asia.	6.0.1.66	Vulnerable	Ripple20
uIP (microIP)	Released in 2001 as an open-source project and extended by Cisco in 2008 with IPv6. Its development has been halted as a standalone project, but it continues as part of the Contiki OS, which in turn has a new version called Contiki-NG.	uIP 1.0 Contiki 3.0 Contiki-NG 4.5	Vulnerable	AMNESIA:33
PicoTCP	Developed by Altran Intelligent Systems and made open source in 2013. The stack continues to be developed as picoTCP-NG, which is no longer supported by Altran.	picoTCP 1.7.0 picoTCP-NG 2.0.0	Vulnerable	AMNESIA:33
FreeBSD	Open-source Unix-like operating system with its own TCP/IP stack, developed since 1993. Currently the most popular OS in the BSD family. DHCP stack analyzed.	12.1	Vulnerable	NAME:WRECK
IPnet	Embedded TCP/IP stack developed originally by Interpeak and acquired by Wind River in 2006. It is used commonly by the VxWorks RTOS and was previously used with other OSes, such as OSE and INTEGRITY.	VxWorks 6.6	Vulnerable	NAME:WRECK
NetX	Developed by Express Logic as part of the ThreadX RTOS since 1997 and purchased by Microsoft in 2019. It is currently an open-source project maintained by Microsoft and has been renamed Azure RTOS NetX.	6.0.1	Vulnerable	NAME:WRECK
Nucleus NET	TCP/IP stack of the Nucleus RTOS, maintained by Siemens EDA. Developed since 1993, originally by Accelerated Technology.	4.3	Vulnerable	NAME:WRECK
FNET	Developed originally at Freescale in 2003 and made public in 2009. It is currently maintained by Andrey Butok.	4.6.3	Not Vulnerable	AMNESIA:33
Nut/Net	TCP/IP stack used by NutOS, which has been developed by the Ethernut project since 2002.	5.1	Not Vulnerable	AMNESIA:33
lwIP	Developed in 2000 by Adam Dunkels and now maintained by a large group of developers. lwIP has become very popular as part of FreeRTOS or as a standalone stack.	2.1.2	Not Vulnerable	AMNESIA:33

Stack	Description	Versions Analyzed	Vulnerable	Research
cycloneTCP	Developed by Oryx Embedded and distributed in source code form since 2013.	1.9.6	Not Vulnerable	AMNESIA:33
uC/TCP-IP	Developed originally by Micrium in 2002 and open sourced in February 2020. uC/OS, which typically relies on the stack, is very popular in mission-critical devices..	3.06.00	Not Vulnerable	AMNESIA:33
FreeRTOS +TCP	Open-source stack developed as part of the widely used FreeRTOS project.	2.2.2	Not Vulnerable	NAME:WRECK
OpenThread	Open-source implementation of the Thread networking technology developed by Google originally for Nest products.	20191113	Not Vulnerable	NAME:WRECK
Zephyr	Modern RTOS with its own TCP/IP stack (originally based on uIP). Developed by Wind River in 2015 and open sourced in 2016 as a project of the Linux Foundation.	2.3.0	Not Vulnerable	NAME:WRECK

Table 3 – New vulnerabilities in NAME:WRECK. Rows are colored according to the CVSS score: yellow for medium or high and red for critical.

CVE ID	Stack	Description	Affected feature	Potential Impact	CVSSv3.1 Score
2020-7461	FreeBSD	The vulnerability exists due to a boundary error when parsing option 119 data in DHCP packets in dhclient(8). A remote attacker on the local network can send specially crafted data to the DHCP client, trigger heap-based buffer overflow and execute arbitrary code on the target system.	Message compression	RCE	7.7
2016-20009	IPnet	The DNS client has a stack-based overflow on the message decompression function leading to a potential RCE. We found this independently but it turned out to be a bug collision with an issue previously reported by Exodus Intelligence, fixed by Wind River in 2016 and that never got assigned a CVE. We discussed the matter with Wind River and the CERT CC in November 2020, who agreed that CVE IDs with an end-of-life tag should be issued. After months without further action from Wind River, we asked the original finders of the vulnerability to request the IDs in January 2021.	Message compression	RCE	9.8
2020-15795	Nucleus NET	The DNS domain name label parsing functionality does not properly validate the names in DNS responses. The parsing of malformed responses could result in a write past the end of an allocated structure. An attacker with a privileged position in the network could leverage this vulnerability to execute code in the context of the current process or cause a denial-of-service condition.	Domain name label parsing	RCE	8.1
2020-27009	Nucleus NET	The DNS domain name record decompression functionality does not properly validate the pointer offset values. The parsing of malformed responses could result in a write past the end of an allocated structure. An attacker with a privileged position in the network could leverage this vulnerability to execute code in the context of the current process or cause a denial-of-service condition.	Message compression	RCE	8.1
2020-27736	Nucleus NET	The DNS domain name label parsing functionality does not properly validate the name in DNS responses. The parsing of malformed responses could result in a read past the end of an allocated structure. An attacker with a privileged position in the network could leverage this vulnerability to cause a denial-of-service condition.	Domain name label parsing	DoS	6.5

CVE ID	Stack	Description	Affected feature	Potential Impact	CVSSv3.1 Score
2020-27737	Nucleus NET	The DNS response parsing functionality does not properly validate various length and counts of the records. The parsing of malformed responses could result in a read past the end of an allocated structure. An attacker with a privileged position in the network could leverage this vulnerability to cause a denial-of-service condition.	Domain name label parsing	DoS	6.5
2020-27738	Nucleus NET	The DNS domain name record decompression functionality does not properly validate the pointer offset values. The parsing of malformed responses could result in a read access past the end of an allocated structure. An attacker with a privileged position in the network could leverage this vulnerability cause a denial-of-service condition.	Message compression	DoS	6.5
2021-25677	Nucleus NET	The DNS client does not properly randomize DNS transaction ID (TXID) and UDP port numbers, allowing attackers to perform DNS cache poisoning/spoofing attacks.	Transaction ID	DNS cache poisoning	5.3
*	NetX	In the DNS resolver component, functions `_nx_dns_name_string_unencode` and `_nx_dns_resource_name_real_size_calculate` do not check that the compression pointer does not equal the same offset currently being parsed, which could lead to an infinite loop. In the function `_nx_dns_resource_name_real_size_calculate` the pointer can also point forward and there is no out-of-bounds check on the packet buffer. Microsoft has classified these issues as leading to DoS. We believe they could lead to a difficult to exploit RCE.	Message compression	DoS	6.5

*We are waiting for a CVE ID to be assigned to this issue.

INFORMATIONAL

A note on the Nucleus NET vulnerabilities

Notice that CVE-2020-15795, CVE-2020-27736, CVE-2020-27737 and CVE-2021-25677 on Nucleus NET are not related to message compression. These vulnerabilities were found as by-products of the analysis of the implementation of message compression and, as discussed in Section 3, they can be used in conjunction with CVE-2020-27009 or CVE-2020-27738 to amplify their effects. This is representative of the facts that compression vulnerabilities are often found with other DNS-related issues in TCP/IP stacks and that typically a combination of vulnerabilities can be exploited together to achieve an RCE. For example, we previously used a combination of CVE-2020-25107 (lack of `NULL`-termination validations) and CVE-2020-25108 (lack of length validation) to create a proof-of-concept for Remote Code Execution in the Nut/Net stack (see our AMNESIA:33 report for more details).

A note on the Nordic nRF5 Software Development Kit

We also analyzed the DNS implementation of the Nordic nRF5 SDK v15.2.0 (file `dns6.c` in `amazon-freertos/vendors/nordic/nRF5_SDK_15.2.0/components/iot/ipv6_stack/dns6`). We found two out-of-bounds reads, potentially leading to denial-of-service, in the DNSv6 resolver component within functions `skip_compressed_hostname` and `server_response`. These issues were reported to Nordic, acknowledged and patched, but never issued CVE IDs because the vendor considered that this is experimental code that should not be used in production devices. **We believe this is dangerous since, in many cases, reference code included with SDKs ends up forming the basis for products developed with that SDK.**

3. Exploitation

In this section, we discuss how an attacker could get remote control of a device by leveraging three vulnerabilities in NAME:WRECK to inject malicious code on a target.

With the first vulnerability, CVE-2020-27009, the attacker can craft a DNS response packet with a combination of invalid compression pointer offsets that allows them to write arbitrary data into sensitive parts of a device's memory, where they will then inject the code. The second vulnerability, CVE-2020-15795, allows the attacker to craft meaningful code to be injected by abusing very large domain name records in the malicious packet. Finally, to deliver the malicious packet to the target, the attacker can bypass DNS query-response matching using CVE-2021-25667.

In Section 4, we discuss how this exploitation fits in a realistic attack scenario.

TECHNICAL DIVE IN

3.1. About DNS message parsing

Before discussing the technical details of the exploitation, we briefly present the format of domain names transmitted in network packets. This is mostly specified in RFC 1035.

A domain name is encoded as a sequence of labels terminated by the `NULL` byte (`0x00`). Each label is preceded by a byte specifying its length (with a maximum length of 63 bytes). For example, the domain name "google.com" is encoded as follows: it starts with the byte `0x06` that indicates the length of the first label, followed by the bytes that correspond to the first label itself (`0x67 0x6f 0x6f 0x67 0x6c 0x65` == "google"), continues with the length of the second label (`0x03`), the bytes that correspond to the second label (`0x63 0x6f 0x6d` == "com") and ends with the `NULL` terminator byte (`0x00`). **Since DNS response packets often include the same domain name several times, RFC 1035 specifies a compression mechanism to reduce the size of DNS messages by replacing a sequence of labels with a pointer to a previous occurrence of the same sequence.** The pointer is encoded in two bytes, the first of them begins with two high bits `11` and the other 14 bits specify an offset from the start of the DNS header. Continuing the example above, and supposing there is a label "google.com" at offset 0x10 of a DNS response packet, the domain "www.google.com" could be encoded as `0x03 0x77 0x77 0x77 0xC0 0x10` (length 3, then "www", then `0b1100000000010000`, which is the two first bits `0b11` and the offset `0x10`). A parser in a DNS server or client would then have to read this packet and when encountering the bits `0b11`, shall follow the pointer to offset `0x10` to be able to expand the data into the desired result ("www.google.com").

3.2. Technical details

Figure 1 shows the `DNS_Unpack_Domain_Name()` function from Nucleus NET. Despite its small size, this function has several vulnerabilities that may lead to a successful Remote Code Execution attack: CVE-2020-27736, CVE-2020-27738, CVE-2020-15795 and CVE-2020-27009.

```
1  INT DNS_Unpack_Domain_Name(CHAR *dst, CHAR *src, CHAR *buf_begin) {
2      INT16           size;
3      INT             i, retval = 0;
4      CHAR            *savesrc;
5
6      savesrc = src;
7
8      while (*src) {
9          size = *src;
10
11         while ((size & 0xC0) == 0xC0) {
12             if (!retval) {
13                 retval = src - savesrc + 2;
14             }
15
16             src++;
17             src = &buf_begin[(size & 0x3f) * 256 + *src];
18             size = *src;
19         }
20
21         src++;
22
23         for (i = 0; i < (size & 0x3f); i++)  {
24             *dst++ = *src++;
25         }
26
27         *dst++ = '.';
28     }
29     *(--dst) = 0;
30     src++;
31
32     if (!retval)   {
33         retval = src - savesrc;
34     }
35
36     return (retval);
37 }
```

Figure 1 – The `DNS_Unpack_Domain_Name()` function in Nucleus NET

The function is called whenever a domain name must be retrieved from a DNS answer record. The first parameter of the function (`dst`) is a pointer to a buffer into which the parsed domain name will be copied. The second parameter (`src`) initially points to the first byte of a domain name. The third parameter (`buf_begin`) is a pointer to the first byte of the DNS header.

TECHNICAL DIVE IN

The code parses a domain name in a `while` loop (line 8) moving the `src` pointer within the packet so that it points to a byte currently being parsed. The `while` loop stops when the `src` pointer encounters the `NULL` byte, which means the end of the domain name. Before entering the parsing loop, the code saves the original `src` pointer into a different variable called `savesrc`. This pointer is used for calculating the length of an uncompressed portion of the domain name.

Within the loop, the code fetches the first byte of the domain name on its first iteration. This byte must be the size of the first label of the name, which will be stored in the `size` variable (line 9). Next, the most significant two bits of this length byte are checked to determine whether it is a normal length byte or a compression pointer (line 11). If at this point `size` is a normal label length, `src` is moved one byte forward (line 21) and the number of bytes equal to `size` are copied into the buffer using the pointer `dst` (lines 23-27). Note that the code truncates the value of `size` to 63 bytes as per RFC1035 (line 23). The variable `retval` (line 33) will hold the total length of the retrieved domain name.

Let us consider the compression pointer check at line 11 again. If at this point `size` holds a compression pointer, the code will add 2 bytes to the resulting name length `retval` if it is the first compression pointer encountered (lines 12-13). Then, the compression offset will be calculated and `src` will be moved from the first byte of the DNS payload (`buf_begin`) according to that offset (lines 16-17); the `size` variable will then hold the label length byte of a domain name to which `src` now points (line 18). Then, the code should process this domain label as shown before (lines 21-27). The assumption here may be that a byte pointed at a compression offset will be the length of an uncompressed label. However, if it is another compression pointer, the `while` loop at line 11 will perform another iteration and `src` will jump to another location specified by the new compression offset.

This directly violates RFC1035 because "… [compression pointer is] a pointer to a prior occurrence of the same name". **The actual problem with this code is that the compression offset value is not validated and, therefore, is under complete control of the attackers. We have reported this vulnerability as CVE-2020-27009**. This issue has several consequences:

- If we choose a compression offset such that `src` jumps back to the same compression pointer, the while loop on lines 11-18 will never terminate and the TCP/IP stack will reach a Denial-of-Service condition. Consider the example shown in Figure 8. In this case, we would have to set the compression pointer and the next byte to `0xc01e` so that the offset in this case will be 30, and this is exactly the offset at which this compression pointer is located.
- If we choose a large enough value such as `0xffff` (the offset will be **16383**), `src` will jump forward within the packet instead of pointing "to a prior occurrence of the same name" as per RFC1035. The code at lines 23-24 will eventually read out of bounds of the packet. The immediate impact may be a Denial-of-Service condition and/or an information leak.
- By carefully choosing a combination of invalid compression offset placed in a DNS packet, attackers can perform controlled out-of-bounds writes into the destination buffer `dst`, potentially achieving Remote Code Execution. The exploitation nuances depend on the implementation specifics of a TCP/IP stack (e.g., how domain name buffers are allocated, what other issues present, among others).

TECHNICAL DIVE IN

Below, we discuss how the third point can be achieved by leveraging other vulnerabilities present in Nucleus NET.

Figure 1 shows the code containing CVE-2020-15795: **There are no checks that ensure that a domain name extracted from a DNS record is within the 255 bytes limit** (as required by RFC1035).

Figure 2 shows an excerpt from the `DNS_Extract_Data()` function that processes DNS response packets and that eventually calls the `DNS_Unpack_Domain_Name()` function shown on Figure 1 (lines 27 and 41). The buffer `name` into which a domain name is copied is allocated in the heap (line 19) with the `NU_Allocate_Memory()`[1] function call. The size of the `name` buffer is limited to 255 bytes with the constant `DNS_MAX_NAME_SIZE` (as per RFC1035).

```
1  STATUS DNS_Extract_Data (DNS_PKT_HEADER *pkt, CHAR *data, UNSIGNED *ttl, INT type) {
2      DNS_RR          *rr_ptr;
3      INT             name_size, n_answers, rcode;
4      UINT16          length;
5      CHAR            *p_ptr, *name;
6      CHAR            answer_received = 0;
7
8      n_answers = GET16(pkt, DNS_ANCOUNT_OFFSET);
9      // [...]
10     /* If there is at least one answer and this is a response, process it. */
11     if ((n_answers > 0) && (GET16(pkt, DNS_FLAGS_OFFSET) & DNS_QR))
12     {
13         /* Point to where the question starts. */
14         p_ptr = (CHAR *)(pkt + 1);
15         /* Allocate a block of memory to put the name in. */
16         if (NU_Allocate_Memory (&System_Memory, (VOID **)&name,
17                                 DNS_MAX_NAME_SIZE,
18                                 NU_NO_SUSPEND) != NU_SUCCESS) {
19             return (NU_NO_MEMORY);
20         }
21
22         /* Extract the name. */
23         name_size = DNS_Unpack_Domain_Name (name, p_ptr, (CHAR *)pkt);
24
25         /*  Move the pointer past the name QTYPE and QCLASS to point at the
26             answer section of the response. */
27         p_ptr += name_size + 4;
28         /*
29         *  At this point, there may be several answers.  We will take the first
30         *  one which has an IP number.  There may be other types of answers that
31         *  we want to support later.
32         */
33         while ((n_answers--) > 0)
34         {
35             /* Extract the name from the answer. */
36             name_size = DNS_Unpack_Domain_Name (name, p_ptr, (CHAR *)pkt);
37
38             /* Move the pointer past the name. */
39             p_ptr += name_size;
40
41            /* Point to the resource record. */
42            rr_ptr = (DNS_RR *)p_ptr;
43
44            // [...]
45
46            /* Copy the length of this answer. */
47            length = GET16(rr_ptr, DNS_RDLENGTH_OFFSET);
48            // [...]
49         }
50        // [...]
51     }
52     // [...]
53 }
```

Figure 2 – An excerpt from the `DNS_Extract_Data()` function in Nucleus NET

[1] The way the memory allocated may be platform-specific, which may influence the exploitation nuances.

TECHNICAL DIVE IN

The domain label length in the `DNS_Unpack_Domain_Name()` function (Figure 1) is limited to 63 bytes, respecting RFC1035. The expression (`size & 0x3f`) on line 23 ensures that the code copies at most 63 bytes inside `name` at a time. However, there is no actual check that limits the number of bytes copied into name. Moreover, the code of `DNS_Unpack_Domain_Name()` relies on the presence of a `NULL` terminator in a domain name to stop copying more bytes (Figure 1, line 8). This is a mistake because the `NULL` byte can be placed at any offset within the name (or not placed at all). We reported this issue under CVE-2020-15795.

These two issues may lead to controlled out-of-bounds writes causing either a Denial-of-Service condition through dereferencing or writing to unmapped or protected memory or leading to Remote Code Execution. Specifically, out-of-bounds writes can occur during either of the two calls of `DNS_Unpack_Domain_Name()` shown on Figure 2 (lines 27 or 41), corrupting the metadata of heap memory chunks adjacent to the memory chunk allocated for `name`. This is a classic heap overflow, which is very similar to CVE-2020-25111, whose exploitation is described in the AMNESIA:33 report.

The easiest way to construct a payload that will overflow `name` and overwrite heap metadata is to chain multiple domain labels of size 63 together so that the total sum of the bytes copied inside `name` will be larger than 255. However, this is not very practical since it means that the DNS packet may be overly large to hold such a large sequence of bytes (and may be flagged by an IDS). This is where the compression pointer vulnerability comes in handy, allowing attackers to wreak havoc even with tiny sequences of bytes.

To illustrate this, we construct such a domain name that consists of only four bytes: a single label of length 1 and a compression pointer that points back on the length byte of this label (e.g., `0x01 0x41 0xc0 0xe1`). These four bytes will cause `DNS_Unpack_Domain_Name()` to overwrite all 255 bytes of `name` with a sequence of **"A"** and **"."** characters and to write this sequence past `name`, corrupting the memory. Eventually, the code may crash (if there is a memory protection mechanism in place) at line 24 of `DNS_Unpack_Domain_Name()` when the `dst` pointer attempts to dereference memory at an invalid address.

This is a simple example of how to abuse vulnerable DNS decompression functions. Details about creating more complex payloads are available in the CVE-2020-11901 whitepaper released by JSOF.

Finally, to deliver the malicious DNS response packet, attackers must be able to pass the DNS query-response matching. CVE-2021-25667 simplifies this bit: Nucleus NET stack did not generate sufficiently random values of TXID (in fact this value was not even used for query-response matching) and the source UDP port number for DNS query packets, allowing to perform low-effort DNS spoofing attacks and to force the stack into accepting malicious DNS responses.

4. An attack scenario leveraging NAME:WRECK

Domain name parsing vulnerabilities may expose both internet-connected and internal devices to attacks since they affect exposed DNS clients, as well as local DHCP clients (although DHCP broadcasts can also be transported across networks via relay agents, as described in RFC1542).

An attack scenario leveraging NAME:WRECK vulnerabilities on internal and external targets is shown in Figure 3. The scenario we describe is based on real-world attacks that have happened in the past and that have seen IoT devices being used both as entry points and for data exfiltration. These past attacks include the data exfiltration at NASA JPL using a Raspberry Pi, the Las Vegas casino hack exploiting an internet-connected thermometer and the oil and gas company that had internet-connected exercise bicycles sending corporate data to the internet.

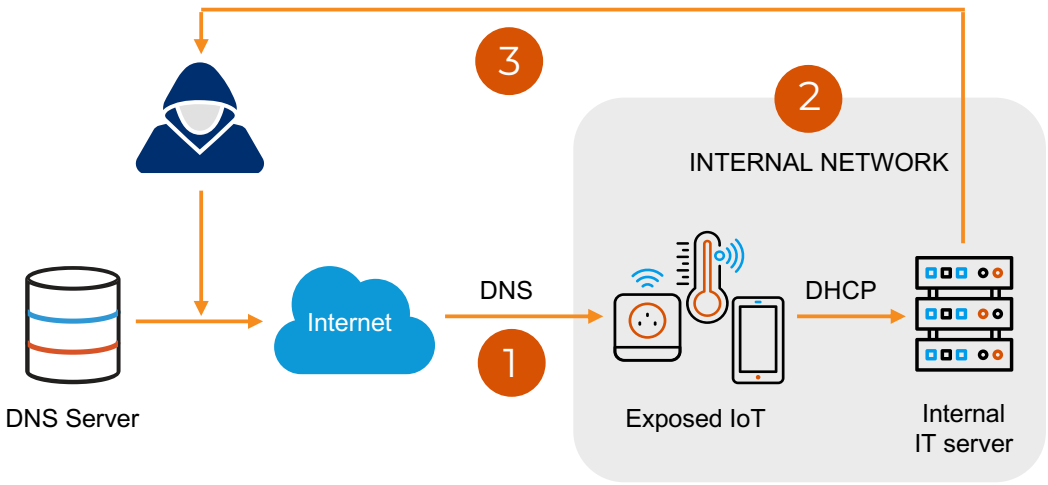


Figure 3 - Attack scenario

In our scenario, the attacker obtains Initial Access into an organization's network (step 1 in the figure) by compromising a device issuing DNS requests to a server on the internet. To obtain initial access, the attacker can exploit one of the RCEs affecting Nucleus NET. The compromise can happen, for instance, by weaponizing the exploitation discussed briefly in Section 3.

The caveat about DNS-based vulnerabilities is that they require the attacker to reply to a legitimate DNS request with a malicious packet. That can be achieved via a man-in-the-middle somewhere in the path between the request and the reply or by exploiting the queried DNS servers. Servers or forwarders vulnerable to DNSpooq and similar vulnerabilities on the way between the target device and a more authoritative DNS server, for instance, could be exploited to reply with malicious messages carrying a weaponized payload.

After the initial access, the attacker can use the compromised entry point to set up an internal DHCP server and do a Lateral Movement (step 2) by executing malicious code on vulnerable internal FreeBSD servers broadcasting DHCP requests.

Finally, the attacker can use those internal compromised servers to Persist on the target network or to Exfiltrate data (step 3) via the internet-exposed IoT device.

Another exploitation scenario involves setting up a DNS server in the internal network and exploiting devices broadcasting mDNS requests. We did not discuss specific mDNS vulnerabilities in this report, but there are two prominent DoS examples in AMNESIA:33 – CVE-2020-17469 affecting FNET and CVE-2020-24340 affecting picoTCP. These can be used to Impact internal IoT/OT devices by taking them offline, thus stopping critical operations.

One important note is that although this type of vulnerability appears both in typical IT software (FreeBSD) and typical OT/IoT embedded software (Nucleus NET and NetX), exploitation in each case is very different. Embedded OSes usually have no support for modern exploit mitigations, such as non-executable data memory (also known as ESP, DEP, NX and W^X), address space layout randomization (ASLR) and stack canaries for protection against memory corruption exploitation. A modern OS such as FreeBSD, on the other hand, implements not only those standard mitigations, but also advanced concepts such as capabilities and sandboxing. The affected FreeBSD DHCP client (dhclient) is one of the applications that runs under a sandbox in the OS.

5. The impact of NAME:WRECK

As discussed at length in the Ripple20 and AMNESIA:33 works, understanding the full impact of vulnerabilities on TCP/IP stacks is difficult because identifying affected vendors and devices using specific IP stack components is very challenging given the absence of a software bill of materials.

This research uncovered vulnerabilities on very popular stacks, and below we discuss some of their uses to give an idea of where they can be found and how many devices are affected. We focus our analysis on three stacks: Nucleus NET, NetX and FreeBSD. **Nucleus NET and NetX have been used for decades in several critical OT and embedded devices**. FreeBSD's network stack is the one that stands out because, although it is used in some embedded devices, it has its origins in general-purpose computing and is still today popular in several IT servers and network appliances.

The Nucleus NET TCP/IP stack is affected by two vulnerabilities that could lead to RCE, CVE-2020-15795 and CVE-2020-27009. According to the website of Nucleus RTOS (which runs the Nucleus TCP/IP stack), it is deployed in more than 3 billion devices. A quick look at Siemens' page listing customer success stories reveals its use in scenarios such as healthcare (ZOLL defibrillators and ZONARE ultrasound machines), IT (BDT AG storage systems) and critical systems (Garmin avionics navigation). But we believe that most of those 3 billion are actually device components such as MediaTek IoT chipsets and baseband processors used in smartphones and other wireless devices (which is similar to the distribution seen below for ThreadX).

FreeBSD (also vulnerable to NAME:WRECK) is widely known to be used for high-performance servers in millions of IT networks, including major websites such as Netflix and Yahoo. FreeBSD is also the basis for other well-known open-source projects, such as the m0n0wall and pfSense firewalls, as well as several commercial network appliances, such as Check Point IPSO and McAfee SecurOS.

Another vulnerable stack is NetX, usually run by the ThreadX RTOS. According to their website, the stack might be used in HTC wearable fitness products, Welch Allyn wearable patient monitors, Broadcom SoCs, Autotalks automotive solutions and the NASA Mars Reconnaissance Orbiter. Several HP printer models, old versions of Intel's Management Engine, popular WiFi SoCs and cellular basebands also run ThreadX. ThreadX was known to have 6.2 billion deployments in 2017 with the following distribution of product categories:

Table 4 – Deployments of ThreadX across products categories [Source: BusinessWire, 2017]

Product category	ThreadX deployments
Mobile Phones	3.4 billion
Consumer Electronics	1.4 billion
Office/Business Automation	923 million
Retail Automation	195 million
Industrial Automation & Energy/Power	161 million
Communication & Networking	86 million
Aerospace & Defense	19 million
Other & General Purpose	12 million
Automotive & Transportation	12 million
Medical Devices	12 million

HIGHLIGHTS

A note on actual affected devices

Note that not all devices running ThreadX, Nucleus RTOS or FreeBSD are vulnerable to NAME:WRECK. Baxter infusion pumps, for instance, use Digi's Net+OS, which is based on ThreadX but runs the Treck TCP/IP stack. In addition, not all devices using a vulnerable stack enable a DNS (or DHCP in the case of FreeBSD) client, and not all versions of the clients are vulnerable. However, if we conservatively assume that 1% of the more than 10 billion deployments discussed above are vulnerable, **we can estimate that at least 100 million devices are impacted by NAME:WRECK.**

To have a better picture of what these impacted devices are and how they are deployed, we looked at two data sources [2]:

- **Online devices.** We queried Shodan for devices having banners indicating the use of the OSes associated with the stacks, for instance the "FreeBSD" string on SSH, HTTP, NTP and other servers.
- **Forescout Device Cloud.** Device Cloud is a closed repository of information coming from devices monitored by Forescout appliances. We queried it for information such as OS classification and application banners, similar to what was done for Ripple20 and AMNESIA:33.

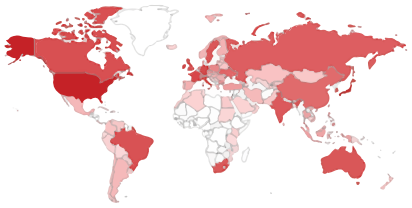
[2] All numbers in this section are up to date as of January 27, 2021.

Shodan searches (shown in Figures 4, 5, 6 and 7) reveal that there are more than 1 million internet-connected instances of FreeBSD, more than 2,500 running Nucleus RTOS and more than 600 running ThreadX. The big difference can be partially explained by the fact that devices running FreeBSD are often supposed to be internet-exposed (such as web servers and firewalls), whereas those running embedded RTOSes are not supposed to be remotely accessible. Interestingly, a search for "Nucleus/4.3" returns more than 700,000 instances, but they seem to be mostly honeypots containing several disconnected application banners.

TOTAL RESULTS

1,052,162

TOP COUNTRIES



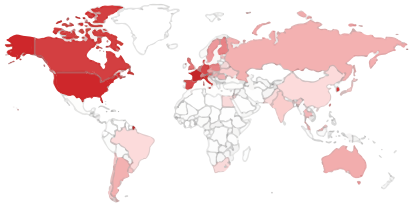
United States	201,818
Japan	103,653
France	57,304
Germany	55,354
Canada	47,603

Figure 4 – Exposed devices running FreeBSD

TOTAL RESULTS

1,345

TOP COUNTRIES



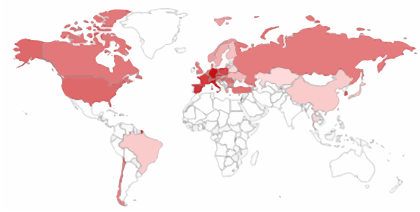
France	264
United States	172
Italy	134
Canada	113
Germany	102

Figure 5 – Exposed devices running Nucleus RTOS ("220 Nucleus FTP")

TOTAL RESULTS

1,453

TOP COUNTRIES



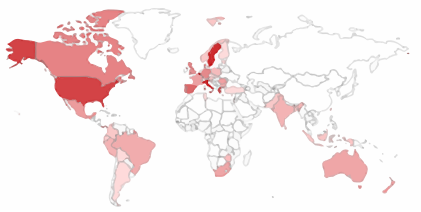
Germany	261
Italy	143
Spain	127
France	113
Czechia	102

Figure 6 – Exposed devices running Nucleus RTOS ("Operating System: Nucleus PLUS")

TOTAL RESULTS

621

TOP COUNTRIES

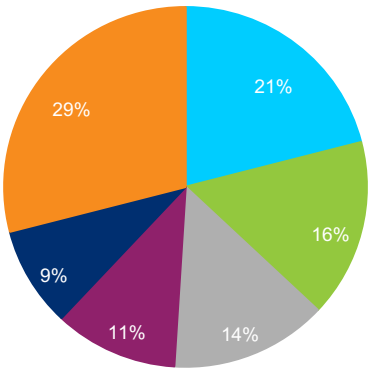


Belgium	140
Sweden	85
Italy	72
United States	59
Greece	54

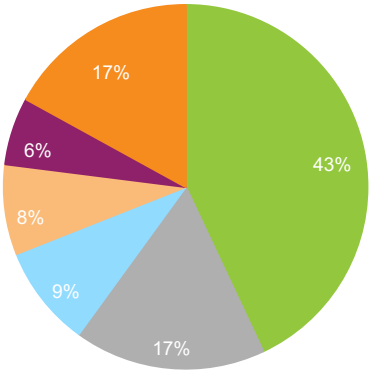
Figure 7 – Exposed devices running ThreadX RTOS

To further identify and classify the impacted devices, we analyzed the Forescout Device Cloud, which contains information about more than 13 million devices in customer networks. Figure 8 shows a breakdown of the number of devices running those OSes per industry vertical. More than 230,000 devices were found running FreeBSD. Similarly, more than 4,000 devices were found running Nucleus RTOS and more than 2500 running ThreadX RTOS.

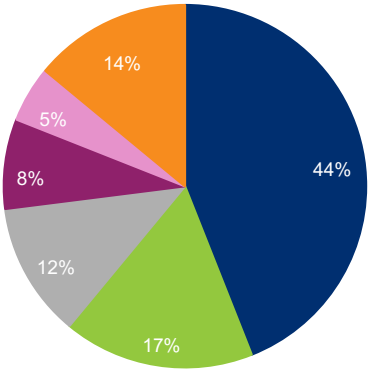
Figure 8 - Devices running the affected IP stack/OSes in Forescout Device Cloud: Top verticals



Vertical	FreeBSD devices
Entertainment	50,012
Healthcare	37,358
Government	32,047
Manufacturing	26,975
Retail	20,254
Others	69,055



Vertical	Nucleus RTOS devices
Healthcare	1,726
Government	670
Financial Services	364
Technology	328
Manufacturing	232
Others	711



Vertical	ThreadX RTOS devices
Retail	1,152
Healthcare	461
Government	320
Manufacturing	210
Services	125
Others	357

6. It's You Again: Recurrent Anti-Patterns

In this Section, we discuss several recurring implementation issues within DNS message parsers, which we call *anti-patterns (AP)*. These anti-patterns were identified as the common causes of vulnerabilities present in NAME:WRECK and in our previous research. Often, these issues can be exploited individually, but their combined presence can give more freedom to the attacker when it comes to exploitation.

6.1. AP#1 – Lack of TXID validation, insufficiently random TXID and source UDP port

The DNS header begins with the DNS transaction ID (TXID) field (see RFC6895). The source UDP port of the outgoing DNS query is used in conjunction with TXID as a synchronization mechanism between DNS clients and servers to match outgoing DNS queries to incoming DNS responses. Similar to initial sequence numbers (ISN) for TCP connections, **both source UDP port and TXID must be difficult to predict, otherwise attackers can forge DNS replies that will be accepted by a vulnerable DNS client** (DNS spoofing). Even if only one of these two values can be easily predicted, it would significantly reduce the effort required for the attackers to "brute-force" the other one and to perform DNS spoofing.

We have observed cases in which TXID of incoming DNS replies is not validated (e.g., CVE-2020-17439 in uIP) and cases in which TXID of outgoing DNS requests will be always set to a constant value (e.g., CVE-2020-17470 in FNET). CVE-2021-25677 in Nucleus NET combines both cases: The TXID has a constant value which is not even used for query-reply matching, and the source UDP port value is predictable.

To remediate this anti-pattern, a vulnerable implementation should use a secure pseudo-random number generator algorithm for creating less predictable TXIDs and source UDP ports, use different TXID and UDP port values for each outgoing DNS query and implement proper DNS query-response matching logic.

6.2. AP#2 – Lack of domain name character validation

RFC1035 recommends that domain labels should only consist of the alphabetic characters from *"A"* to *"Z"* digits and the hyphen character. For example, example-demo112.com conforms to this recommendation, while ex@mple_demo.com does not. However, as stated in RFC2181, this is not a strong requirement (unlike maximum domain name and label lengths), so DNS implementations must not place any restrictions on the label characters that can be used.

This contradiction may stem from the confusion between *domain names*[3] and *hostnames*[4]. As stated in RFC2181, *"... [the fact that] any binary label can have a MX record does not imply that any binary name can be used as the host part of an e-mail address... "* Therefore, while certain DNS records may have any set of characters as a part of the domain name, internet hosts must conform to RFC1035. In practice, we rarely see these checks implemented, and **to keep things simple, any character will be accepted as a valid part of a domain name of any record type**. While this is not a vulnerability per se, the consequence is that attackers have more freedom in creating exploit payloads because any character will be accepted by a vulnerable DNS parser.

[3] Domain name – an identifier of a network/resource in a DNS database.

[4]Hostname – a specific kind of domain name which is used to identify internet hosts.

6.3. AP#3 – Lack of label and name lengths validation

RFC1035 restricts the length of individual domain name labels to 63 characters and the length of domain names to 255 characters. **Some implementations do not restrict domain labels and names to these lengths, allowing attackers to craft longer payloads for facilitating exploitation of other vulnerabilities in DNS parsers.** Another related issue occurs when these length values are copied directly from a network packet and not checked with respect to the data present. For example, even though the maximum domain label and name lengths were respected in Nucleus NET, there was no check whether the reported lengths were correct with respect to the actual number of bytes present in a domain name. This resulted in CVE-2020-27736 and CVE-2020-15795.

Often, these length values correspond to the size of some internal buffers that will hold domain names or labels, and they are allocated in heap or stack regions of the memory. The absence of bounds checks here allows the attackers to control the allocation of these buffers.

6.4. AP#4 – Lack of NULL-termination validation

RFC1035 states that domain names must end with a `NULL` byte (`0x00`) that signifies the end of a name. Some implementations may just assume that domain names in incoming DNS messages are terminated with `NULL`, but they make no checks for it. This issue is closely related to the absence of name and response data length checks. Attackers can control the placement of a `NULL` byte at a certain offset in a domain name, which in combination with lax domain name and label length checks may result in controlled memory reads and writes.

Even when the domain name boundary checks are implemented correctly, the absence of explicit checks for the `NULL` byte placement may lead to memory-related off-by-one errors, causing a Denial-of-Service condition. CVE-2020-27736 in Nucleus NET and CVE-2020-17440 in uIP (part of AMNESIA:33) are good examples of such a vulnerability.

6.5. AP#5 – Lack of the record count fields validation

RFC6895 provides the format for DNS query/response headers: Every header contains four 2-byte fields that specify the number of questions (`QCOUNT`), answers (`ANCOUNT`), authorities (`NSCOUNT`) and additional information records (`ARCOUNT`). After the DNS header, there must be present the data that can be parsed into individual records (e.g., answers, questions, among others).

We have observed a recurring implementation error in which record count fields (e.g., `QCOUNT` and `ANCOUNT`) are taken directly from the DNS packet, but there are no checks that validate whether the packet has enough data to hold the specified numbers of records. While RFC5625 mentions that DNS packets with incorrect QCOUNT/ANCOUNT/NSCOUNT/ARCOUNT values should be dropped, developers of TCP/IP stacks may fail to do so.

CVE-2020-27737 in Nucleus NET and several other vulnerabilities within AMNESIA:33 are examples of this issue. Here, by setting a bogus value to ANCOUNT and by providing no answer records within the packet (or fewer than set in ANCOUNT), attackers may cause a Denial-of-Service condition when the code attempts to read out of bounds of the packet when it tries to parse the answer records that do not exist. The code that implements the parsing of DNS records must first validate that the specified number of records exist within the packet, before attempting to parse it.

6.6. AP#6 – Lack of domain name compression pointer and offset validation

RFC1035 defines that whenever a domain label length byte has the value such that the two highest bits are set to 1, this byte is treated as a *compression pointer*. The *compression offset* is specified in the next 14 bits (beginning at the third most significant bit of the *compression pointer*). The value of the *compression offset* represents the offset in bytes starting from the beginning of the DNS header, at which a non-compressed domain name is located. Consider the example shown on Figure 9.

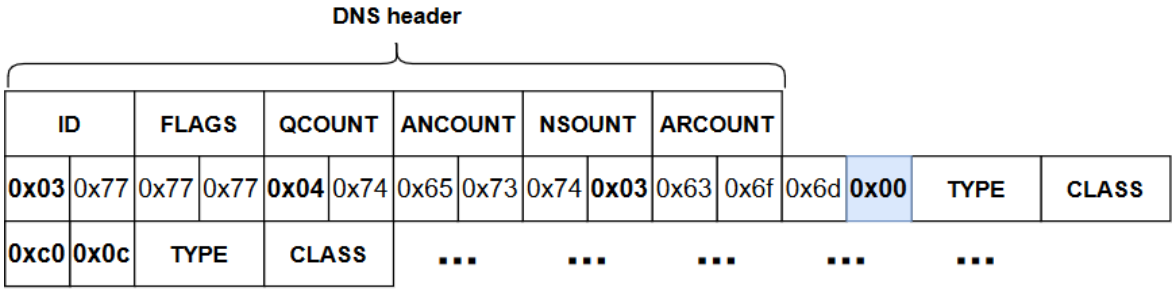


Figure 9 – DNS packet with a compressed domain name

Here, the first row represents the DNS header (the bytes from 1 to 12). Let us assume that this packet is a reply to a DNS query and its QCOUNT and ANCOUNT are both equal to 1. The next 18 bytes correspond to the question record: 14 bytes of the domain name that the DNS client requested, followed by 4 bytes of TYPE and CLASS fields. The last 4 bytes correspond to the answer record.

The domain name in the question record is www.test.com. The bytes that correspond to the domain label lengths and the `NULL` terminator are shown in bold. When we look at the bytes of the answer, we notice the byte `0xc0` – it is a *compression pointer* because the two highest bits of this value are set to 1 (the binary representation of `0xc0` is `0b11000000`). This means that the record does not contain a domain name, but it refers an uncompressed name that should be present in some previous record. We must now compute the *compression offset* which holds the position of this name in the DNS payload.

The *compression offset* is computed as follows (the process is illustrated on Figure 10): The two most significant bits of the *compression pointer* byte are set to 0, and it is shifted left by 8 bits so that the lower byte of the resulting 2-byte value becomes 0x00. Next, this 2-byte value is added to the second byte that comes after the pointer. The resulting 2-byte value will be the offset in the DNS payload at which the domain name starts. For the case illustrated on Figure 9 the *compression offset* is **12**, and the byte at that offset is `0x03` (the first byte of the domain name in the question record), and the domain name in the answer record is also "*www.test.com*".

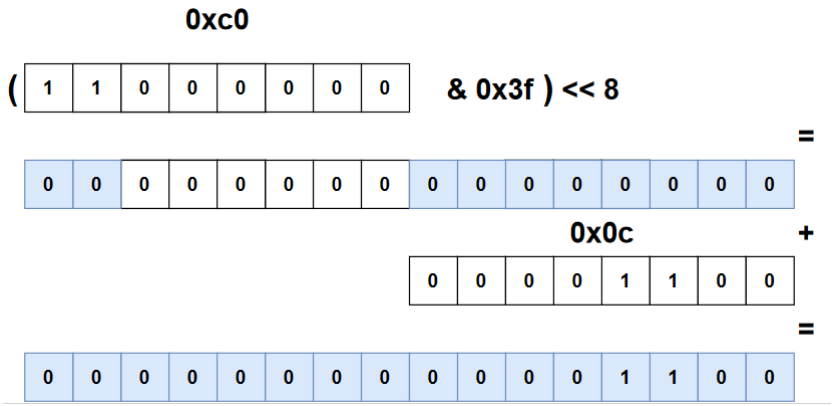


Figure 10 – Computing the compression offset

RFC1035 discusses domain name compression very briefly and does not warn about some non-obvious mistakes that the developers of TCP/IP stacks may make when implementing this logic. **The value of the compression pointer is often unchecked in the code of TCP/IP stacks and, since it is a 14-bit value, it can in theory point to 16383 (`0x3fff`) bytes past the beginning of the DNS header (which makes it quite unlikely that it points to a valid domain name in this case).** Simply put, if the packet is shorter than this value (consider our example on Figure 9), the code might read out of bounds of the packet. If the pointer points to itself (e.g., we set the two relevant bytes to `0xc01e` so that the compression points again to `0xc0`), we might cause the parsing code to enter an infinite loop.

In fact, the devil is in the details, as RFC1035 states that "… [*in compression scheme] an entire domain name or a list of labels at the end of a domain name is replaced with a* ***pointer to a prior occurrence of the same name.***" This means that there must be checks in place to ensure that a compression offset in an incoming packet points "backwards" within the packet and lands on a valid uncompressed domain name. When such checks do not exist, it is possible to craft offset values at which the offset will be pointing "forward," allowing the attackers to "hijack" the vulnerable DNS parser with carefully crafted compression pointers and offsets.

RFC5625 gives a hint about *"invalid compression pointers"* as *"… those that point outside of the current packet or that might cause a parsing loop"*, mentioning them as *"examples of malformed packets that MAY be dropped"*. As we have found during our research, such packets **MUST be dropped**, as parsing them may result in a variety of security issues, depending on how a particular DNS response parser in a TCP/IP stack is implemented.

Another typical mistake with compression pointers that we have seen in the past (e.g., uIP and PicoTCP in AMNESIA:33) is to check only that **either** of the two most significant bits of a label length is `0b1`. In this case, label lengths such as `0x80` and `0x40` will also be considered valid compression pointers. This violates RFC1035 (high bit combinations 10 and 01 are reserved for special use and must not be present in a domain name), and while this is not a vulnerability per se, it may be beneficial to attackers. For example, an intrusion detection system that has a rule for detecting invalid compression offsets may not flag specific malformed packets because **0x80** is not a compression pointer, but some vulnerable implementations treat this value as such.

7. Mitigation Recommendations

Complete protection against NAME:WRECK requires patching devices running the vulnerable versions of the IP stacks. FreeBSD, Nucleus NET and NetX have been patched recently, and device vendors using this software should provide their own updates to customers.

However, patching devices is not always possible, and the required effort changes drastically whether the device is a standard IT server or an IoT device, as we discuss briefly below.

- If security operators intend to patch vulnerable FreeBSD servers or network appliances, they 'just' need to (1) identify what operating system is running on their devices, (2) obtain the versions of currently installed packages (such as dhclient) and (3) update the vulnerable systems. These operations can even be automated and parallelized in case the servers support remote management via SSH, for instance. The official patch for the FreeBSD vulnerability makes it very clear that an administrator must simply run three commands to patch the system. Usually, these servers are even deployed with high availability and load balancing, which means they can be rebooted without major problems while other servers provide a similar service.
- If security operators intend to patch vulnerable IoT devices running vulnerable Nucleus NET- or NetX-based firmware, the situation becomes more complex. First, the user (and sometimes even the device vendor) is unsure of what TCP/IP stack runs on a device, which means that

identifying vulnerable devices and issuing patches takes longer. Second, even when security patches trickle down from the stack vendor to the firmware of the device, it is more difficult for the user to apply those patches because the devices are not centrally managed (so patches must be manually applied to each device), and sometimes they cannot be taken offline due to their mission-critical nature (such as medical devices or industrial control systems).

Even worse, we found that new firmware sometimes runs unsupported versions of an RTOS that may have known vulnerabilities. This is extremely concerning since assuming that a new firmware is not vulnerable might lead to serious blind spots in network risk assessment. An example of this is the case of CVE-2016-20009, which affects older versions of VxWorks that are not officially supported anymore, unless under a paid extended support program. Even without having a CVE assigned before NAME:WRECK, this issue was silently fixed (a practice that is unfortunately common among certain vendors) in at least some devices, such as Huawei firewalls. The currently supported versions of VxWorks are 6.9 and 7. However, versions of the RTOS as old as 5.x, which was released more than 20 years ago, still seem to be very popular. There is supporting evidence that old versions of the OS are still used. For instance, according to the results of our Shodan searches, there are more than 4,000 results for "vxworks5.4.2" and close to 1,500 for "vxworks5.5.1". Additionally, there are several devices with newly released firmware based on old VxWorks, such as Dell PowerConnect IT switches, Siemens SCALANCE ICS switches and Echelon i.LON 600 IP routers for building automation. There are also other vulnerabilities (unrelated to domain name parsing) affecting VxWorks back to 5.5 (e.g., CVE-2020-11440) that seem to be fixed only for versions 6.9 and 7. We have not checked that any of these firmware are vulnerable. In conversation with WindRiver, they informed us that older versions may also be patched, but this depends on their customer having special support agreements.

Given the challenges described above, we also recommend the following mitigation strategy in case pathing is not possible.

- **Discover and inventory devices running the vulnerable stacks.** Forescout Research Labs has released an open-source script that uses active fingerprinting to detect devices running the affected stacks. The script is updated constantly with new signatures to follow the latest development of our research.
- **Enforce segmentation controls and proper network hygiene** to mitigate the risk from vulnerable devices. Restrict external communication paths and isolate or contain vulnerable devices in zones as a mitigating control if they cannot be patched or until they can be patched.
- **Monitor progressive patches released by affected device vendors** and devise a remediation plan for your vulnerable asset inventory balancing business risk and business continuity requirements.
- **Configure devices to rely on internal DNS servers** as much as possible and closely monitor external DNS traffic since exploitation require a malicious DNS server to reply with malicious packets.
- **Monitor all network traffic for malicious packets** that try to exploit known vulnerabilities or possible 0-days affecting DNS, mDNS and DHCP clients. Anomalous and malformed traffic should be blocked, or at least alert its presence to network operators.

Following the anti-patterns described in Section 6, below we discuss a general set of features of malicious packets that may indicate exploitation attempts for the vulnerabilities outlined in this report. TCP/IP stacks should properly detect and drop malformed packets within their code, but since this is not the case as exemplified by the multiple vulnerabilities in this report, we recommend implementing the rules outlined below in a network IDS.

- **Invalid compression pointer.** A compression pointer (a byte with the 2 highest bits set to 1) must resolve to a byte within a DNS record with the value that is greater than 0 (it must not be a `NULL` terminator) and is less than 64. The offset at which this byte is located must be smaller than the offset at which the compression

pointer is located. There is no valid reason for nesting compression pointers. The code that implements domain name parsing should check the offset not only with respect to the bounds of a packet, but also its position with respect to the compression pointer in question. The little payload of `0x01 0x41 0xc0 0xe1` that we demonstrated in Section 3 meets all these requirements but can still lead to writing out of bounds or infinite loops. Therefore, a compression pointer must not be "followed" more than once. This might be difficult to implement within the logic of TCP/IP stacks, as we have seen several implementations using a check that ensures that a compression pointer is not followed more than several times. While this is not a perfect solution, it may still be a practical one.

- **Invalid domain label, name and resource data lengths.** A domain name length byte must have the value of more than 0 and less than 64. If this is not the case, an invalid value has been provided within the packet, or a value at an invalid position might be interpreted as a domain name length due to other errors in the packet (e.g., misplaced `NULL` terminator or invalid compression pointer). The characters of the domain label allowed for internet hosts must strictly conform to RFC1035, and the number of domain label characters must correspond to the value of the domain label byte. The domain name length must not be more than 255 bytes, and the `NULL` terminator character must be present at the end of the domain name. The value of the data length byte in response DNS records (`RDLENGTH`) must reflect the number of bytes available in the field that describes the resource (`RDATA`). The format of `RDATA` must conform to the `TYPE` and `CLASS` fields of the resource record.
- **Invalid counts for Question/Answer/Authority/Additional records.** The values of the bytes within a DNS header that reflect the number of Question (`QCOUNT`), Answer (`ANCOUNT`), Authority (`NSCOUNT`) and Additional (`ARCOUNT`) must correspond to the actual data present within the packet.

8. Conclusions and Final Remarks

In previous research, we noticed that mis-implementations of RFCs (sometimes because of ambiguities, as in the case of the TCP Urgent pointer) are one of the most common causes of vulnerabilities (what we called an "anti-pattern") and that similar constraints tend to lead to similar vulnerabilities in TCP/IP stacks.

NAME:WRECK is a case where bad implementations of a specific part of an RFC can have disastrous consequences that spread across different parts of a TCP/IP stack and then different products using that stack.

It is noteworthy that when a stack has a vulnerable DNS client, there are often several vulnerabilities together, but the **message compression anti-pattern stands out because it commonly leads to potential RCEs, as it is often associated with pointer manipulation and memory operations**. It is also interesting that simply not implementing support for compression (as seen for instance in lwIP) is an effective mitigation against this type of vulnerability. Since the bandwidth saving associated to this type of compression is almost meaningless in a world of fast connectivity, we believe that support for DNS message compression currently introduces more problems than it solves.

While working on NAME:WRECK, we noticed that DNS client implementations may be tested less rigorously than server implementations for security issues. Because the clients regularly communicate with a limited set of servers (instead of a large set of clients), they may be more prone to security vulnerabilities being detected later in the development cycle and potentially remaining for longer in production software.

Besides disclosing vulnerabilities to vendors, thus helping to secure impacted products, **this research helps the cybersecurity community in many other ways**. Below, we provide a list of lessons learned during our research and some recommendations that we believe could prevent vulnerabilities like NAME:WRECK from resurging in other TCP/IP stacks.

- In Project Memoria, we learned that often the same mistake (anti-pattern) leads to similar vulnerabilities in different stacks. **We urge developers of TCP/IP stacks** that have not (yet) been analyzed to take the anti-patterns available in Section 6 (as well as the ones available in the AMNESIA:33 report), check their code for the presence of bugs and fix them.
- To help with the point above, we are releasing open-source code developed for the Joern static analysis tool. (The results of running this code are shown in Figure 11 for PicoTCP and Figure 12 for Nucleus NET). This code is a formalization of the anti-patterns we identified and **allows researchers and developers to automatically analyze other stacks for the presence of similar vulnerabilities.**

```
joern> cpg.runScript("/home/stanislav.dashevskyi/work/joern/static-analysis-queries/joern/vuln_taxonomy/main.sc")
!!! POTENTIAL DNS COMPRESSION POINTER CHECK ERROR !!!
>>> Check only ensures that either of the two most significant bits of compression pointer must be 1
        The full control expression:'if (*label & 0xC0)'
        File : /home/stanislav.dashevskyi/work/code-analysis/picotcp-ng/modules/pico_dns_common.c
        Function : pico_dns_decompress_name
        Line : 139
        Statement : *label & 0xC0

!!! POTENTIAL DNS COMPRESSION POINTER CHECK ERROR !!!
>>> Check only ensures that either of the two most significant bits of compression pointer must be 1
        The full control expression:'if ((0xC0 & *label))'
        File : /home/stanislav.dashevskyi/work/code-analysis/picotcp-ng/modules/pico_dns_common.c
        Function : pico_dns_namelen_comp
        Line : 91
        Statement : 0xC0 & *label

!!! POTENTIAL DNS COMPRESSION OFFSET OUT OF BOUND BUG !!!
>>> Doesn´t check if the dns compression offset is out of bound
        File : /home/stanislav.dashevskyi/work/code-analysis/picotcp-ng/modules/pico_dns_common.c
        Function : pico_dns_decompress_name
        Line : 141
        Statement : ptr = (uint16_t)((((uint16_t) *label) & 0x003F) << 8)
```

Figure 11 – Running the Joern script against vulnerable PicoTCP code

```
joern> cpg.runScript("/home/stanislav.dashevskyi/work/joern/static-analysis-queries/joern/vuln_taxonomy/main.sc")
!!! POTENTIAL DNS COMPRESSION OFFSET OUT OF BOUND BUG !!!
>>> Doesn´t check if the dns compression offset is out of bound
        File : /home/stanislav.dashevskyi/work/code-analysis/nucleus_net/Net/Src/DNS.C
        Function : DNS_Unpack_Domain_Name
        Line : 761
        Statement : src = &buf_begin[(size & 0x3f) * 256 + *src]
```

Figure 12 – Running the Joern script against vulnerable Nucleus NET code

- The discussion about exploit detection in Section 7 allows security engineers to develop detection signatures for DNS vulnerabilities, which can be used for known and new vulnerabilities. Alongside this report, we invite researchers, developers and vendors to reach out to us if they are interested in a set of small proof-of-concept crashing network packets for the identified anti-patterns. These packets can be used to automatically test detection rules.
- We realized that many of the vulnerabilities exist because RFC documents are either too complex, ambiguous or outdated. This is the case with RFC5625 stating that "… invalid compression pointers" such as "… those that point outside of the current packet or that might cause a parsing loop" are "… examples of malformed packets that MAY be dropped." As we have found during our research, such packets MUST be dropped, as parsing them may result in a variety of security issues. To help prevent such issues from reappearing in the future, we have submitted to the IETF an informational RFC draft where we list the anti-patterns of section 6 and how to avoid them while implementing a DNS client or server.

We welcome collaboration with vendors, researchers and the cybersecurity community as a whole under the scope of Project Memoria. There is much work left to be done to understand the real dangers behind the foundations of IT/OT/IoT connectivity, and the more parties we can get involved in finding vulnerabilities, fixing them and providing higher-level solutions, the faster we can transition to a more secure world.