

Independent Submission
Internet-Draft
Intended status: Informational
Expires: November 18, 2022

S. Dashevskiy
D. dos Santos
J. Wetzels
A. Amri
Forescout Technologies
May 18, 2022

**Common implementation anti-patterns related
to Domain Name System (DNS) resource record (RR) processing
draft-dashevskiy-dnsrr-antipatterns-06**

Abstract

This memo describes common vulnerabilities related to Domain Name System (DNS) response record (RR) processing as seen in several DNS client implementations. These vulnerabilities may lead to successful Denial-of-Service and Remote Code Execution attacks against the affected software. Where applicable, violations of [RFC 1035](#) are mentioned.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 18, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction
2. Compression Pointer and Offset Validation
3. Label and Name Length Validation
4. Null-terminator Placement Validation
5. Response Data Length Validation
6. Record Count Validation
7. Security Considerations
8. IANA Considerations
9. References
 - 9.1. Normative References
 - 9.2. Informative References
- Acknowledgements
- Authors' Addresses

1. Introduction

Recently, there have been major vulnerabilities on DNS implementations that raised attention to this protocol as an important attack vector, such as [\[SIGRED\]](#), [\[SADDNS\]](#), and [\[DNSPOOO\]](#) - a set of 7 critical issues affecting the DNS forwarder "dnsmasq".

The authors of this memo have analyzed the DNS client implementations of several major TCP/IP protocol stacks and found a set of vulnerabilities that share common implementation flaws (anti-patterns). These flaws are related to processing DNS RRs (discussed in [\[RFC1035\]](#)) and may lead to critical security vulnerabilities.

While implementation flaws may differ from one software project to another, these anti-patterns are highly likely to span across multiple implementations. In fact, one of the first CVEs related to one of the anti-patterns [\[CVE-2000-0333\]](#) dates back to the year 2000. The observations are not limited to DNS client implementations. Any software that processes DNS RRs may be affected, such as firewalls, intrusion detection systems, or general purpose DNS packet dissectors (e.g., [\[CVE-2017-9345\]](#) in Wireshark). Similar issues may also occur in DNS-over-HTTPS [\[RFC8484\]](#) and DNS-over-TLS [\[RFC7858\]](#) implementations. However, any implementation that deals with the DNS wire format is subject to the considerations discussed in this draft.

[\[COMP-DRAFT\]](#) and [\[RFC5625\]](#) briefly mention some of these anti-patterns, but the main purpose of this memo is to provide technical details behind these anti-patterns, so that the common mistakes can be eradicated.

We provide general recommendations on mitigating the anti-patterns. We also suggest that all implementations should drop malicious/malformed DNS replies and log them (optionally).

2. Compression Pointer and Offset Validation

[RFC1035] defines the DNS message compression scheme that can be used to reduce the size of messages. When it is used, an entire domain name or several name labels are replaced with a (compression) pointer to a prior occurrence of the same name.

The compression pointer is a combination of two octets: the two most significant bits are set to 1, and the remaining 14 bits are the OFFSET field. This field specifies the offset from the beginning of the DNS header, at which another domain name or label is located:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 1|                               OFFSET                        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

The message compression scheme explicitly allows a domain name to be represented as: (1) a sequence of unpacked labels ending with a zero octet; (2) a pointer; (3) a sequence of labels ending with a pointer.

However, [\[RFC1035\]](#) does not explicitly state that blindly following compression pointers of any kind can be harmful [\[COMP-DRAFT\]](#), as we could not have had any assumptions about various implementations that would follow.

Yet, any DNS packet parser that attempts to decompress domain names without validating the value of OFFSET is likely susceptible to memory corruption bugs and buffer overruns. These bugs allow for easy Denial-of-Service attacks, and may result in successful Remote Code Execution attacks.

Pseudocode that illustrates a typical example of a broken domain name parsing implementation is shown below (Snippet 1):

```
1:decompress_domain_name(*name, *dns_payload) {
2:
3:  name_buffer[255];
4:  copy_offset = 0;
5:
6:  label_len_octet = name;
7:  dest_octet = name_buffer;
8:
9:  while (*label_len_octet != 0x00) {
10:
11:    if (is_compression_pointer(*label_len_octet)) {
12:      ptr_offset = get_offset(label_len_octet,
                             label_len_octet+1);
13:      label_len_octet = dns_payload + ptr_offset + 1;
14:    }
15:
16:    else {
17:      length = *label_len_octet;
```

```

18:         copy(dest_octet + copy_offset,
                label_len_octet+1, *length);
19:
20:         copy_offset += length;
21:         label_len_octet += length + 1;
22:     }
23:
24: }
25:}

```

Snippet 1 - A broken implementation of a function that is used for decompressing DNS domain names (pseudocode)

Such implementations typically have a dedicated function for decompressing domain names (for example, see [[CVE-2020-24338](#)] and [[CVE-2020-27738](#)]). Among other parameters, these functions may accept a pointer to the beginning of the first name label within a RR ("name") and a pointer to the beginning of the DNS payload to be used as a starting point for the compression pointer ("dns_payload"). The destination buffer for the domain name ("name_buffer") is typically limited to 255 bytes as per [[RFC1035](#)] and can be allocated either in the stack or in the heap memory region.

The code of the function at Snippet 1 reads the domain name label-by-label from a RR until it reaches the NUL octet ("0x00") that signifies the end of a domain name. If the current label length octet ("label_len_octet") is a compression pointer, the code extracts the value of the compression offset and uses it to "jump" to another label length octet. If the current label length octet is not a compression pointer, the label bytes will be copied into the name buffer, and the number of bytes copied will correspond to the value of the current label length octet. After the copy operation, the code will move on to the next label length octet.

The first issue with this implementation is due to unchecked compression offset values. The second issue is due to the absence of checks that ensure that a pointer will eventually arrive at an decompressed domain label. We describe these issues in more detail below.

[[RFC1035](#)] states that "... [compression pointer is] a pointer to a prior occurrence of the same name". Also, according to [[RFC1035](#)], the maximum size of DNS packets that can be sent over the UDP protocol is limited to 512 octets.

The pseudocode at Snippet 1 violates these constraints, as it will accept a compression pointer that forces the code to read out of the bounds of a DNS packet. For instance, the compression pointer of "0xffff" will produce the offset of 16383 octets, which is most definitely pointing to a label length octet somewhere past the original DNS packet. Supplying such offset values will most likely

cause memory corruption issues and may lead to Denial-of-Service conditions (e.g., a Null pointer dereference after "label_len_octet" is set to an invalid address in memory). As an additional example, see [CVE-2020-25767], [CVE-2020-24339], and [CVE-2020-24335].

The pseudocode at Snippet 1 allows for jumping from a compression pointer to another compression pointer and it does not restrict the number of such jumps. That is, if a label length octet which is currently being parsed is a compression pointer, the code will perform a jump to another label, and if that other label is a compression pointer as well, the code will perform another jump, and so forth until it reaches an decompressed label. This may lead to unforeseen side-effects that result in security issues.

Consider the excerpt from a DNS packet illustrated below:

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+0x00 | ID | FLAGS | QCOUNT | ANCOUNT | NSCOUNT | ARCOUNT |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
->+0x0c | 0xc0|0xc0| TYPE | CLASS | 0x04| t | e | s | t | 0x03|
| +-----+---+-----+-----+-----+-----+-----+-----+-----+-----+
| +0x18 | c | o | m | 0x00| TYPE | CLASS | ..... |
| +-----+---+-----+-----+-----+-----+-----+-----+-----+-----+
|
-----

```

The packet begins with a DNS header at the offset +0x00, and its DNS payload contains several RRs. The first RR begins at the offset of 12 octets (+0xc0) and its first label length octet is set to the value "0xc0", which indicates that it is a compression pointer. The compression pointer offset is computed from the two octets "0xc00c" and it is equal to 12. Since the broken implementation at Snippet 1 follows this offset value blindly, the pointer will jump back to the first octet of the first RR (+0xc0) over and over again. The code at Snippet 1 will enter an infinite loop state, since it will never leave the "TRUE" branch of the "while" loop.

Apart from achieving infinite loops, the implementation flaws at Snippet 1 make it possible to achieve various pointer loops that have other effects. For instance, consider the DNS packet excerpt shown below:

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+0x00 | ID | FLAGS | QCOUNT | ANCOUNT | NSCOUNT | ARCOUNT |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
->+0x0c | 0x04| t | e | s | t | 0xc0|0xc0| ..... |
| +-----+---+-----+-----+-----+-----+---+-----+-----+-----+
|
-----

```

With such a domain name, the implementation at Snippet 1 will first

copy the domain label at the offset "0xc0" ("test"), then it will fetch the next label length octet, which is a compression pointer ("0xc0"). The compression pointer offset is computed from the two octets "0xc00c" and is equal to 12 octets. The code will jump back at the offset "0xc0" where the first label "test" is located. The code will again copy the "test" label, and jump back to it, following the compression pointer, over and over again.

Snippet 1 does not contain any logic that restricts multiple jumps from the same compression pointer and does not ensure that no more than 255 octets are copied into the name buffer ("name_buffer"). In fact, the code will continue to write the label "test" into it, overwriting the name buffer and the stack of the heap metadata. In fact, attackers would have a significant degree of freedom in constructing shell-code, since they can create arbitrary copy chains with various combinations of labels and compression pointers.

Therefore, blindly following compression pointers may not only lead to Denial-of-Service as pointed by [\[COMP-DRAFT\]](#), but also to successful Remote Code Execution attacks, as there may be other implementation issues present within the corresponding code.

Some implementations may not follow [\[RFC1035\]](#), which states: "the first two bits [of a compression pointer octet] are ones; this allows a pointer to be distinguished from a label, the label must begin with two zero bits because labels are restricted to 63 octets or less (the 10 and 01 combinations are reserved for future use)". Snippets 2 and 3 show pseudocode that implements two functions that check whether a given octet is a compression pointer: correct and incorrect implementations respectively.

```
1: unsigned char is_compression_pointer(*octet) {
2:     if ((*octet & 0xc0) == 0xc0)
3:         return true;
4:     } else {
5:         return false;
6:     }
7: }
```

Snippet 2 - Correct compression pointer check

```
1: unsigned char is_compression_pointer(*octet) {
2:     if (*octet & 0xc0) {
3:         return true;
4:     } else {
5:         return false;
6:     }
7: }
```

Snippet 3 - Broken compression pointer check

The correct implementation (Snippet 2) ensures that the two most significant bits of an octet are both set, while the broken

implementation (Snippet 3) would consider an octet with only one of the two bits set as a compression pointer. This is likely an implementation mistake rather than an intended violation of [RFC1035], because there are no benefits in supporting such compression pointer values. The implementations related to [CVE-2020-24338] and [CVE-2020-24335] had a broken compression pointer check illustrated on Snippet 3.

While incorrect implementations alone do not lead to vulnerabilities, they may have unforeseen side-effects when combined with other vulnerabilities. For instance, the first octet of the value "0x4130" may be incorrectly interpreted as a label length by a broken implementation. Such label length (65) is invalid, and is larger than 63 (as per [RFC1035]), and a packet that has this value should be discarded. However, the function shown on Snippet 3 will consider "0x41" to be a valid compression pointer, and the packet may pass the validation steps.

This might give an additional leverage for attackers in constructing payloads and circumventing the existing DNS packet validation mechanisms.

The first occurrence of a compression pointer in a RR (an octet with the 2 highest bits set to 1) must resolve to an octet within a DNS record with the value that is greater than 0 (i.e., it must not be a Null-terminator) and less than 64. The offset at which this octet is located must be smaller than the offset at which the compression pointer is located - once an implementation makes sure of that, compression pointer loops can never occur.

In small DNS implementations (e.g., embedded TCP/IP stacks) the support for nested compression pointers (pointers that point to a compressed name) should be discouraged: there is very little to be gained in terms of performance versus the high possibility of introducing errors, such as the ones discussed above.

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. A compression pointer must not be "followed" more than once. We have seen several implementations using a check that ensures that a compression pointer is not followed more than several times. A better alternative may be to ensure that the target of a compression pointer is always located before the location of the pointer in the packet.

3. Label and Name Length Validation

[RFC1035] restricts the length of name labels to 63 octets, and lengths of domain names to 255 octets (i.e., label octets and label length octets). Some implementations do not explicitly enforce these

restrictions.

Consider the function "copy_domain_name()" shown on Snippet 4 below. The function is a variant of the "decompress_domain_name()" function (Snippet 1), with the difference that it does not support compressed labels, and copies only decompressed labels into the name buffer.

```
1: copy_domain_name(*name, *dns_payload) {
2:
3:   name_buffer[255];
4:   copy_offset = 0;
5:
6:   label_len_octet = name;
7:   dest_octet = name_buffer;
8:
9:   while (*label_len_octet != 0x00) {
10:
11:     if (is_compression_pointer(*label_len_octet)) {
12:       length = 2;
13:       label_len_octet += length + 1;
14:     }
15:
16:     else {
17:       length = *label_len_octet;
18:       copy(dest_octet + copy_offset,
            label_len_octet+1, *length);
19:
20:       copy_offset += length;
21:       label_len_octet += length + 1;
22:     }
23:
24:   }
25: }
```

Snippet 4 - A broken implementation of a function
that is used for copying non-compressed domain names

This implementation does not explicitly check for the value of the label length octet: this value can be up to 255 octets, and a single label can fill the name buffer. Depending on the memory layout of the target, how the name buffer is allocated, and the size of the malformed packet, it is possible to trigger various memory corruption issues.

Both Snippets 1 and 4 restrict the size of the name buffer to 255 octets, however there are no restrictions on the actual number of octets that will be copied into this buffer. In this particular case, a subsequent copy operation (if another label is present in the packet) will write past the name buffer, allowing to overwrite heap or stack metadata in a controlled manner.

Similar examples of vulnerable implementations can be found in the

code relevant to [\[CVE-2020-25110\]](#), [\[CVE-2020-15795\]](#), and [\[CVE-2020-27009\]](#).

As a general recommendation, a domain label length octet must have the value of more than 0 and less than 64 ([\[RFC1035\]](#)). 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 number of domain label characters must correspond to the value of the domain label octet. To avoid possible errors when interpreting the characters of a domain label, developers may consider recommendations for the preferred domain name syntax outlined in [\[RFC1035\]](#).

The domain name length must not be more than 255 octets, including the size of decompressed domain names. The NUL octet ("0x00") must be present at the end of the domain name, and within the maximum name length (255 octets).

4. Null-terminator Placement Validation

A domain name must end with a NUL ("0x00") octet, as per [\[RFC1035\]](#). The implementations shown at Snippets 1 and 4 assume that this is the case for the RRs that they process, however names that do not have a NUL octet placed at the proper position within a RR are not discarded.

This issue is closely related to the absence of label and name length checks. For example, the logic behind Snippets 1 and 4 will continue to copy octets into the name buffer, until a NUL octet is encountered. This octet can be placed at an arbitrary position within a RR, or not placed at all.

Consider a pseudocode function shown on Snippet 5. The function returns the length of a domain name ("name") in octets to be used elsewhere (e.g., to allocate a name buffer of a certain size): for compressed domain names the function returns 2, for decompressed names it returns their true length using the "strlen(3)" function.

```
1: get_name_length(*name) {
2:
3:     if (is_compression_pointer(name))
4:         return 2;
5:
6:     name_len = strlen(name) + 1;
7:     return name_len;
8: }
```

Snippet 5 - A broken implementation of a function that returns the length of a domain name

"strlen(3)" is a standard C library function that returns the length of a given sequence of characters terminated by the NUL ("0x00") octet. Since this function also expects names to be explicitly Null-terminated, the return value "strlen(3)" may be also controlled by attackers. Through the value of "name_len" attackers may control the allocation of internal buffers, or specify the number by octets copied into these buffers, or other operations depending on the implementation specifics.

The absence of explicit checks for the NUL octet placement may also facilitate controlled memory reads and writes. An example of vulnerable implementations can be found in the code relevant to [\[CVE-2020-25107\]](#), [\[CVE-2020-17440\]](#), [\[CVE-2020-24383\]](#), and [\[CVE-2020-27736\]](#).

As a general recommendation for mitigating such issues, developers should never trust user data to be Null-terminated. For example, to fix/mitigate the issue in the code Snippet 5, developers should use the function "strnlen(3)" that reads at most X characters(the second argument of the function), and ensure that X is not larger than the buffer allocated for the name.

5. Response Data Length Validation

As stated in [\[RFC1035\]](#), every RR contains a variable length string of octets that contains the retrieved resource data (RDATA) (e.g., an IP address that corresponds to a domain name in question). The length of the RDATA field is regulated by the resource data length field (RDLENGTH), that is also present in an RR.

Implementations that process RRs may not check for the validity of the RDLENGTH field value, when retrieving RDATA. Failing to do so may lead to out-of-bound read issues (similarly to the label and name length validation issues discussed in [Section 3](#)), whose impact may vary significantly depending on the implementation specifics. We have observed instances of Denial-of-Service conditions and information leaks.

Therefore, 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 RR.

Examples of vulnerable implementations can be found in the code relevant to [\[CVE-2020-25108\]](#), [\[CVE-2020-24336\]](#), and [\[CVE-2020-27009\]](#).

6. Record Count Validation

According to [\[RFC1035\]](#), the DNS header contains four two-octet fields that specify the amount of question records (QDCOUNT), answer records (ANCOUNT), authority records (NSCOUNT), and additional

records (ARCOUNT).

```
1: process_dns_records(dns_header, ...) {
    // ...
2:     num_answers = dns_header->ancount
3:     data_ptr = dns_header->data
4:
5:     while (num_answers > 0) {
6:         name_length = get_name_length(data_ptr);
7:         data_ptr += name_length + 1;
8:
9:         answer = (struct dns_answer_record *)data_ptr;
10:
11:         // process the answer record
12:
13:         --num_answers;
14:     }
    // ...
15: }
```

Snippet 6 - A broken implementation of a RR processing function

Snippet 6 illustrates a recurring implementation anti-pattern for a function that processes DNS RRs. The function "process_dns_records()" extracts the value of ANCOUNT ("num_answers") and the pointer to the DNS data payload ("data_ptr"). The function processes answer records in a loop decrementing the "num_answers" value after processing each record, until the value of "num_answers" becomes zero. For simplicity, we assume that there is only one domain name per answer. Inside the loop, the code calculates the domain name length "name_length", and adjusts the data payload pointer "data_ptr" by the offset that corresponds to "name_length + 1", so that the pointer lands on the first answer record. Next, the answer record is retrieved and processed, and the "num_answers" value is decremented.

If the ANCOUNT number retrieved from the header ("dns_header->ancount") is not checked against the amount of data available in the packet and it is, e.g., larger than the number of answer records available, the data pointer "data_ptr" will read out of the bounds of the packet. This may result in Denial-of-Service conditions.

In this section, we used an example of processing answer records. However, the same logic is often reused for implementing the processing of other types of records: e.g., the number of Question (QCOUNT), Authority (NSCOUNT), and Additional (ARCOUNT) records. The number of these records specified must correspond to the actual data present within the packet. Therefore, all record count fields must be checked before fully parsing the contents of a packet. Specifically, [Section 6.3](#) of [RFC5625] recommends that such malformed DNS packets should be dropped, and (optionally) logged.

Examples of vulnerable implementations can be found in the code relevant to [[CVE-2020-25109](#)], [[CVE-2020-24340](#)], [[CVE-2020-24334](#)], and [[CVE-2020-27737](#)].

7. Security Considerations

Security issues are discussed throughout this memo. The document discusses implementation flaws (anti-patterns) that affect the functionality of processing DNS RRs. The presence of such anti-patterns leads to bugs causing buffer overflows, read-out-of-bounds, and infinite loop issues. These issues have the following security impact: Information Leak, Denial-of-Service, and Remote Code Execution.

The document lists general recommendation for the developers of DNS record parsing functionality that allow to prevent such implementation flaws, e.g., by rigorously checking the data received over the wire before processing it.

8. IANA Considerations

This document introduces no new IANA considerations. Please see [[RFC6895](#)] for a complete review of the IANA considerations introduced by DNS.

9. References

9.1 Normative References

[RFC1035]

Mockapetris, P., "Domain names - implementation and specification", [RFC 1035](#), November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.

[RFC5625]

Bellis, R., "DNS Proxy Implementation Guidelines", [RFC 5625](#), August 2009, <<https://www.rfc-editor.org/info/rfc5625>>.

9.2 Informative References

[SIGRED]

Common Vulnerabilities and Exposures, "CVE-2020-1350: A remote code execution vulnerability in Windows Domain Name System servers", July 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-1350>>.

[SADDNS]

Man, K., Qian, Z., Wang, Z., Zheng, X., Huang, Y., Duan, H., "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels", November 2020, Proc. of ACM CCS'20, <<https://dl.acm.org/doi/pdf/10.1145/3372297.3417280>>.

[DNSPOOQ]

Kol, M., Oberman, S., "DNSpooq: Cache Poisoning and RCE in popular DNS Forwarder dnsmasq", January 2021, technical report, <<https://www.jsf-tech.com/wp-content/uploads/2021/01/DNSpooq-Technical-WP.pdf>>.

[CVE-2000-0333]

Common Vulnerabilities and Exposures, "CVE-2000-0333: A denial-of-service vulnerability in tcpdump, Ethereal, and other sniffer packages via malformed DNS packets", 2000, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0333>>.

[CVE-2020-24338]

Common Vulnerabilities and Exposures, "CVE-2020-24338: A denial-of-service and remote code execution vulnerability in the DNS domain name record decompression functionality of picoTCP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24338>>

[CVE-2020-27738]

Common Vulnerabilities and Exposures, "CVE-2020-27738: A denial-of-service and remote code execution vulnerability DNS domain name record decompression functionality of Nucleus NET", April 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27738>>.

[CVE-2020-25767]

Common Vulnerabilities and Exposures, "CVE-2020-25767: An out-of-bounds read and denial-of-service vulnerability in the DNS name parsing routine of HCC Embedded NicheStack", August 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25767>>.

[CVE-2020-24339]

Common Vulnerabilities and Exposures, "CVE-2020-24339: An out-of-bounds read and denial-of-service vulnerability in the DNS domain name record decompression functionality of picoTCP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24339>>.

[CVE-2020-24335]

Common Vulnerabilities and Exposures, "CVE-2020-24335: A memory corruption vulnerability in domain name parsing routines of uIP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24335>>.

[CVE-2020-25110]

Common Vulnerabilities and Exposures, "CVE-2020-25110: A denial-of-service and remote code execution vulnerability in the DNS implementation of Ethernut Nut/OS", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25110>>.

[CVE-2020-15795]

Common Vulnerabilities and Exposures, "CVE-2020-15795: A denial-of-service and remote code execution vulnerability DNS domain name label parsing functionality of Nucleus NET", April 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15795>>.

[CVE-2020-27009]

Common Vulnerabilities and Exposures, "CVE-2020-27009: A denial-of-service and remote code execution vulnerability DNS domain name record decompression functionality of Nucleus NET", April 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27009>>.

[CVE-2020-25107]

Common Vulnerabilities and Exposures, "CVE-2020-25107: A denial-of-service and remote code execution vulnerability in the DNS implementation of Ethernut Nut/OS", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25107>>.

[CVE-2020-17440]

Common Vulnerabilities and Exposures, "CVE-2020-17440: A denial-of-service vulnerability in the DNS name parsing implementation of uIP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-17440>>.

[CVE-2020-24383]

Common Vulnerabilities and Exposures, "CVE-2020-24383: An information leak and denial-of-service vulnerability while parsing mDNS resource records in FNET", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24383>>.

[CVE-2020-27736]

Common Vulnerabilities and Exposures, "CVE-2020-27736: An information leak and denial-of-service vulnerability in the DNS name parsing functionality of Nucleus NET", April 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27736>>.

[CVE-2020-25108]

Common Vulnerabilities and Exposures, "CVE-2020-25108: A denial-of-service and remote code execution vulnerability in the DNS implementation of Ethernut Nut/OS", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25108>>.

[CVE-2020-24336]

Common Vulnerabilities and Exposures, "CVE-2020-24336: A buffer overflow vulnerability in the DNS implementation of Contiki and Contiki-NG", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24336>>.

[CVE-2020-25109]

Common Vulnerabilities and Exposures, "CVE-2020-25109: A denial-of-service and remote code execution vulnerability in the DNS implementation of Ethernut Nut/OS", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25109>>.

[CVE-2020-24340]

Common Vulnerabilities and Exposures, "CVE-2020-24340: An out-of-bounds read and denial-of-service vulnerability in the DNS response parsing functionality of picoTCP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24340>>.

[CVE-2020-24334]

Common Vulnerabilities and Exposures, "CVE-2020-24334: An out-of-bounds read and denial-of-service vulnerability in the DNS response parsing functionality of uIP", December 2020, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24334>>.

[CVE-2020-27737]

Common Vulnerabilities and Exposures, "CVE-2020-27737: An information leak and denial-of-service vulnerability in the DNS response parsing functionality of Nucleus NET", April 2021, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27737>>.

[CVE-2017-9345]

Common Vulnerabilities and Exposures, "CVE-2017-9345: An infinite loop in the DNS dissector of Wireshark", 2017, <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9345>>.

[COMP-DRAFT]

Koch, P., "A New Scheme for the Compression of Domain Names", Internet-Draft, [draft-ietf-dnsind-local-](#)

[compression-05](#), June 1999, Work in progress,
<<https://tools.ietf.org/html/draft-ietf-dnsind-local-compression-05>>.

[RFC6895]

Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", [RFC 6895](#), April 2013,
<<https://www.rfc-editor.org/info/rfc6982>>.

[RFC8484]

Hoffman, P., McManus, P., "DNS Queries over HTTPS (DoH)", [RFC 8484](#), October 2018,
<<https://www.rfc-editor.org/info/rfc8484>>.

[RFC7858]

Hu, Z. et al, "Specification for DNS over Transport Layer Security (TLS)", [RFC 7858](#), May 2016,
<<https://www.rfc-editor.org/info/rfc7858>>.

Acknowledgements

We would like to thank Shlomi Oberman, who has greatly contributed to the research that led to the creation of this document.

Authors' Addresses

Stanislav Dashevskiy
Forescout Technologies
John F. Kennedylaan, 2
Eindhoven, 5612AB
The Netherlands

Email: stanislav.dashevskiy@forescout.com

Daniel dos Santos
Forescout Technologies
John F. Kennedylaan, 2
Eindhoven, 5612AB
The Netherlands

Email: daniel.dossantos@forescout.com

Jos Wetzels
Forescout Technologies
John F. Kennedylaan, 2
Eindhoven, 5612AB
The Netherlands

Email: jos.wetzels@forescout.com

Amine Amri

Forescout Technologies
John F. Kennedylaan, 2
Eindhoven, 5612AB
The Netherlands

Email: amine.amri@forescout.com