Talos Vulnerability Report

TALOS-2020-1000

Videolabs libmicrodns 0.1.0 message-parsing bounds denial-of-service vulnerability

MARCH 23, 2020

CVE NUMBER

CVE-2020-6077

Summary

An exploitable denial-of-service vulnerability exists in the message-parsing functionality of Videolabs libmicrodns 0.1.0. When parsing mDNS messages, the implementation does not properly keep track of the available data in the message, possibly leading to an out-of-bounds read that would result in a denial of service. An attacker can send an mDNS message to trigger this vulnerability.

Tested Versions

Videolabs libmicrodns 0.1.0

Product URLs

https://github.com/videolabs/libmicrodns

CVSSv3 Score

7.5 - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

CWE

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

Details

The libmicrodns library is an mDNS resolver that aims to be simple and compatible cross-platform.

The function `mdns_recv` reads and parses an mDNS message:

```
static int
mdns_recv(const struct mdns_conn* conn, struct mdns_hdr *hdr, struct rr_entry **entries)
{
        uint8_t buf[MDNS_PKT_MAXSZ];
        size_t num_entry, n;
        ssize_t length;
        struct rr_entry *entry;

        *entries = NULL;
        if ((length = recv(conn->sock, (char *) buf, sizeof(buf), 0)) < 0)       // [1]
                return (MDNS_NETERR);

        const uint8_t *ptr = mdns_read_header(buf, length, hdr);                 // [2]
        n = length;

        num_entry = hdr->num_qn + hdr->num_ans_rr + hdr->num_add_rr;
        for (size_t i = 0; i < num_entry; ++i) {
                entry = calloc(1, sizeof(struct rr_entry));
                if (!entry)
                        goto err;
                ptr = rr_read(ptr, &n, buf, entry, i >= hdr->num_qn);            // [3]
                if (!ptr) {
                        free(entry);
                        errno = ENOSPC;
                        goto err;
                }
                entry->next = *entries;
                *entries = entry;
        }
        ...
}
```

At [1], a message is read from the network. The 12-bytes mDNS header is then parsed at [2]. Based on the header info, the loop parses each resource record (“RR”) using the function `rr_read` [3], which in turn calls `rr_read_RR` and then `rr_decode`.

```
#define advance(x) ptr += x; *n -= x

/*
 * Decodes a DN compressed format (RFC 1035)
 * e.g "\x03foo\x03bar\x00" gives "foo.bar"
 */
static const uint8_t *
rr_decode(const uint8_t *ptr, size_t *n, const uint8_t *root, char **ss)
{
        char *s;

        s = *ss = malloc(MDNS_DN_MAXSZ);
        if (!s)
                return (NULL);

        if (*ptr == 0) {                                                    // [9]
                *s = '\0';
                advance(1);
                return (ptr);
        }
        while (*ptr) {                                                      // [4]
                size_t free_space;
                uint16_t len;

                free_space = *ss + MDNS_DN_MAXSZ - s;
                len = *ptr;
                advance(1);

                /* resolve the offset of the pointer (RFC 1035-4.1.4) */
                if ((len & 0xC0) == 0xC0) {
                        const uint8_t *p;
                        char *buf;
                        size_t m;

                        if (*n < sizeof(len))                               // [5]
                                goto err;
                        len &= ~0xC0;
                        len = (len << 8) | *ptr;
                        advance(1);

                        p = root + len;                                     // [8]
                        m = ptr - p + *n;                                   // [6]
                        rr_decode(p, &m, root, &buf);
                        if (free_space <= strlen(buf)) {
                                free(buf);
                                goto err;
                        }
                        (void) strcpy(s, buf);
                        free(buf);
                        return (ptr);
                }
                if (*n <= len || free_space <= len)                         // [7]
                        goto err;
                strncpy(s, (const char *) ptr, len);
                advance(len);
                s += len;
                *s++ = (*ptr) ? '.' : '\0';                                 // [10]
        }
        advance(1);
        return (ptr);
err:
        free(*ss);
        return (NULL);
}
```

The function `rr_decode` expects 4 parameters:

- `ptr`: the pointer to the start of the label to parse
- `n`: the number of remaining bytes in the message, starting from ptr
- `root`: the pointer to the start of the mDNS message
- `ss`: buffer used to build the domain name

The task of this function is to parse a domain name in a given resource record, according to RFC 1035.
To walk through the message, the `advance` macro is used to move `ptr` forward and to decrement `*n` (the number of bytes left in the message) accordingly.
Also note how the code relies on the value of `*n` to make decisions [5] [6] [7] about the loop [4] termination.

In the code above, the comparison at [5] is performed incorrectly. Because of the `sizeof`, the code checks if `*n` is either `0` or `1`. This means that the pointer `p` at [8] can point from `root` up to `root + 16383`. The function would then call recursively, reading out of bounds and possibly crashing at [9] or [10] when dereferencing `ptr`.

Additionally, before `rr_decode` is called, the function `mdns_read_header` [2] parses the mDNS header:

```
static const uint8_t *
mdns_read_header(const uint8_t *ptr, size_t n, struct mdns_hdr *hdr)
{
        if (n <= sizeof(struct mdns_hdr)) {
                errno = ENOSPC;
                return NULL;
        }
        ptr = read_u16(ptr, &n, &hdr->id);
        ptr = read_u16(ptr, &n, &hdr->flags);
        ptr = read_u16(ptr, &n, &hdr->num_qn);
        ptr = read_u16(ptr, &n, &hdr->num_ans_rr);
        ptr = read_u16(ptr, &n, &hdr->num_auth_rr);
        ptr = read_u16(ptr, &n, &hdr->num_add_rr);
        return ptr;
}
```

The function `read_u16` takes care of reading a 2-bytes unsigned integer, and decrementing `n` accordingly [11]:

```
static inline const uint8_t *read_u16(const uint8_t *p, size_t *s, uint16_t *v)
{
        *v = 0;
        *v |= *p++ << 8;
        *v |= *p++ << 0;
        *s -= 2;           // [11]
        return (p);
}
```

However, as we can see from the definition of `mdns_read_header` and the way it's called [12], `n` is passed to `mdns_read_header` by value rather than by reference, losing any modification performed by `read_u16`.

```
static int
mdns_recv(const struct mdns_conn* conn, struct mdns_hdr *hdr, struct rr_entry **entries)
{
        uint8_t buf[MDNS_PKT_MAXSZ];
        size_t num_entry, n;
        ssize_t length;
        struct rr_entry *entry;

        *entries = NULL;
        if ((length = recv(conn->sock, (char *) buf, sizeof(buf), 0)) < 0)
                return (MDNS_NETERR);

        const uint8_t *ptr = mdns_read_header(buf, length, hdr);                 // [12]
        n = length;
```

This makes the `*n` value to be 0xC bytes (the header size) bigger than it should. While, in absence of other bugs, this last issue alone may not cause a direct impact on the service, it may be used together with the first bug presented in this advisory in order to evade detection.

Finally, while this bug alone would result in a simple denial-of-service, because of other bugs in the code, it may be used to trigger the same double-free that was reported in TALOS-2020-0995.

Timeline

2020-01-30 - Vendor Disclosure
2020-03-20 - Vendor Patched
2020-03-23 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1018

NEXT REPORT TALOS-2020-1001