

## Talos Vulnerability Report

TALOS-2020-0994

### Videolabs libmicrodns 0.1.0 resource record recursive label uncompression denial-of-service vulnerability

MARCH 23, 2020

#### CVE NUMBER

CVE-2020-6071

#### Summary

An exploitable denial-of-service vulnerability exists in the resource record-parsing functionality of Videolabs libmicrodns 0.1.0. When parsing compressed labels in mDNS messages, the compression pointer is followed without checking for recursion, leading to 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-674: Uncontrolled Recursion

#### Details

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

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

```
static int
mdns_rcv(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 = rcv(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-byte 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) {
        *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) { // [5]
            const uint8_t *p;
            char *buf;
            size_t m;

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

            p = root + len;
            m = ptr - p + *n; // [6]
            rr_decode(p, &m, root, &buf); // [7]
            ...
        }
    }
}

```

The function `rr_decode` expects four 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

At [4], the function loops for each character in the label and, if a pointer is found [5], the pointed label location and its maximum size is computed at [6], and the `rr_decode` function is called recursively [7].

Since there are no recursion checks, an attacker could send a message with a label which uses a pointer to point to itself, triggering an infinite recursion and exhausting the program's stack, leading to a denial of service.

#### 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

NEXT REPORT

TALOS-2019-0966

TALOS-2020-0995

