Talos Vulnerability Report

TALOS-2021-1353

Garrett Metal Detectors iC Module CMA check_udp_crc memcpy stack-based buffer overflow vulnerability

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-21901

Summary

A stack-based buffer overflow vulnerability exists in the CMA check_udp_crc function of Garrett Metal Detectors' iC Module CMA Version 5.0. A specially-crafted packet can lead to a stack-based buffer overflow during a call to `memcpy`. An attacker can send a malicious packet to trigger this vulnerability.

Tested Versions

Garrett Metal Detectors iC Module CMA Version 5.0

Product URLs

https://garrett.com/security/walk-through/accessories

CVSSv3 Score

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

CWE

CWE-120 - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Details

The Garrett iC Module provides network connectivity to either the Garrett PD 6500i or Garrett MZ 6100 models of walk-through metal detectors. This module enables a remote user to monitor statistics such as alarm and visitor counts in real time as well as make configuration changes to metal detectors.

The Garrett iC Module exposes a discovery service on UDP port 6877. The "CMA Connect" software, used to interact with the iC modules from a remote system, can broadcast a particularly formatted UDP packet onto the network and iC modules that receive this packet will reply with various descriptors such as MAC address, serial number, and location. A function call to memcpy within the CRC validation logic of these UDP packets is vulnerable to a stack-based buffer overflow.

This buffer overflow occurs due to a mismatch in maximum buffer sizes between the function responsible for handling incoming UDP packets, `udp_thread`, and the function responsible for validating the message's checksum, `check_udp_crc`. When a UDP message is received inside of `udp_thread`, a `msghdr` struct is populated with an `iovec` struct whose `message` attribute points to a 512-byte long character array called `msgbuf`.

```
.text:0001D730                SUB     R3, R11, #-msgbuf                       [1] uint8_t msgbuf[512]
.text:0001D734                MOV     R0, R3          ; s
.text:0001D738                MOV     R1, #0          ; c
.text:0001D73C                MOV     R2, #0x200      ; n
.text:0001D740                BL      memset                                  [2] memset(msgbuf, 0, 512)
.text:0001D744                SUB     R3, R11, #-msgbuf
.text:0001D748                STR     R3, [R11,#iov]                          [3] iov[0].iov_base = msgbuf
.text:0001D74C                MOV     R3, #0x200
.text:0001D750                STR     R3, [R11,#iov.iov_len]                  [4] iov[0].iov_len = 512
.text:0001D754                SUB     R3, R11, #-(src_addr.__ss_padding+4)
.text:0001D758                SUB     R3, R3, #0xC
.text:0001D75C                STR     R3, [R11,#message]
.text:0001D760                MOV     R3, #0x80
.text:0001D764                STR     R3, [R11,#message.msg_namelen]
.text:0001D768                SUB     R3, R11, #-iov
.text:0001D76C                STR     R3, [R11,#message.msg_iov]              [5] message.msg_iov = iov
.text:0001D770                MOV     R3, #1
.text:0001D774                STR     R3, [R11,#message.msg_iovlen]           [6] message.msg_iovlen = 1
.text:0001D778                SUB     R3, R11, #-(cmbuf+0xC)
.text:0001D77C                SUB     R3, R3, #0xC
.text:0001D780                STR     R3, [R11,#message.msg_control]
.text:0001D784                MOV     R3, #0x100
.text:0001D788                STR     R3, [R11,#message.msg_controllen]
.text:0001D78C                SUB     R3, R11, #-message
.text:0001D790                LDR     R0, [R11,#udpFd] ; fd
.text:0001D794                MOV     R1, R3          ; message
.text:0001D798                MOV     R2, #0          ; flags
.text:0001D79C                BL      recvmsg                                 [7] recvmsg(udpFd, &message, 0);
```

After a successful call to `recvmsg`, the `udp_thread` function will null-terminate the `msgbuf` buffer at the first instance of `\r\n`. The `msgbuf` buffer is then passed to `check_udp_crc` to confirm the payload's CRC is valid.

```
.text:0001D7C8                SUB     R3, R11, #-msgbuf
.text:0001D7CC                MOV     R0, R3          ; s
.text:0001D7D0                LDR     R1, =asc_2FCD8  ; "\r\n"
.text:0001D7D4                BL      strcspn                                 [8] $r0 = strcspn(msgbuf, "\r\n")
.text:0001D7D8                MOV     R2, R0
.text:0001D7DC                LDR     R3, =0xFFFFFC8C
.text:0001D7E0                SUB     R0, R11, #-var_C
.text:0001D7E4                ADD     R2, R0, R2
.text:0001D7E8                ADD     R3, R2, R3
.text:0001D7EC                MOV     R2, #0
.text:0001D7F0                STRB    R2, [R3]                                [9] msgbuf[$r0] = '\0'
.text:0001D7F4                SUB     R3, R11, #-msgbuf
.text:0001D7F8                MOV     R0, R3          ; msg
.text:0001D7FC                BL      check_udp_crc                           [10] check_udp_crc(msgbuf)
```

While the maximum length of the UDP payload in `udp_thread` is 512 bytes, the `check_udp_crc` function only allocates 256 bytes for its internal copy of the payload. A `memcpy` is executed which will copy `strlen(msg)` bytes from `msgbuf` into `msg`, resulting in a very straightforward buffer overflow.

```
.text:0001D134                PUSH    {R11,LR}
.text:0001D138                ADD     R11, SP, #4
.text:0001D13C                SUB     SP, SP, #0x128
.text:0001D140                STR     R0, [R11,#msg]
.text:0001D144                SUB     R3, R11, #-msg_buf
.text:0001D148                MOV     R0, R3          ; s
.text:0001D14C                MOV     R1, #0          ; c
.text:0001D150                MOV     R2, #0x100      ; n
.text:0001D154                BL      memset                                  [11] memset(msg, 0, 256)
.text:0001D158                LDR     R0, [R11,#msg]  ; s
.text:0001D15C                BL      strlen
.text:0001D160                MOV     R3, R0                                  [12] msglen = strlen(msg)
.text:0001D164                SUB     R2, R11, #-msg_buf
.text:0001D168                MOV     R0, R2          ; dest
.text:0001D16C                LDR     R1, [R11,#msg]  ; src
.text:0001D170                MOV     R2, R3          ; n
.text:0001D174                BL      memcpy                                  [13] memcpy(msg, msgbuf, msglen)
```

As shown above, a maliciously crafted UDP packet with a significantly long payload will result in a buffer overflow. This overflow directly leads to attacker control of the program counter, which may be seen in the debugger output below.

Crash Information

```
Thread 2 "cma" received signal SIGSEGV, Segmentation fault.
0x4d4d4d4c in ?? ()
──────────────────────────────────────────────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x1
$r1  : 0x3b
$r2  : 0x1
$r3  : 0x1
$r4  : 0x0
$r5  : 0xb636b6b0  →  "eth0"
$r6  : 0x0
$r7  : 0x152
$r8  : 0xbefffbe0  →  0x00000000
$r9  : 0xb6ff86d0  →  0xb6ff8db8  →  0x00000001
$r10 : 0xb636c460  →  0x00000001
$r11 : 0x4d4d4d4d ("MMMM"?)
$r12 : 0xb636b6a8  →  0x00000000
$sp  : 0xb636b6a0  →  0xb636b7d0  →  0x00000018
$lr  : 0xb6bf7898  →  <strrchr+40> cmp r0,  #0
$pc  : 0x4d4d4d4c ("LMMM"?)
$cpsr: [negative ZERO CARRY overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────── code:arm:THUMB ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x4d4d4d4c
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
```

Timeline

2021-08-17 - Vendor Disclosure

2021-11-10 - Talos granted disclosure extension

2021-12-13 - Vendor patched

2021-12-15 - Talos tested patch

2021-12-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.
