Talos Vulnerability Report

TALOS-2021-1355

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

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-21903

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 strcpy. 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. iC modules that receive this packet will reply with various descriptors such as MAC address, serial number, and location. A function call to strcpy within the CRC validation logic of these UDP packets is vulnerable to a stack-based buffer overflow.

The UDP packet is composed of a set of text fields, delimited by semi-colons and terminated with a CRC value. The CRC is represented as the ASCII-encoded decimal representation of the CRC. For example, if the calculated CRC of the payload is `0xA7CF` then the UDP packet will be terminated with the string "42959".

The following chunk of code is responsible for identifying the offset of the CRC field and copying it into a fixed-size buffer.

```
.text:0001D178                SUB     R3, R11, #-msg_buf
.text:0001D17C                MOV     R0, R3          ; s
.text:0001D180                MOV     R1, #0x3B ; ';' ; c
.text:0001D184                BL      strrchr
.text:0001D188                STR     R0, [R11,#end_of_data]                      [1] v2 = strrchr(msg, ';')  // Find the last
.text:0001D18C                LDR     R3, [R11,#end_of_data]
.text:0001D190                ADD     R2, R3, #1
.text:0001D194                STR     R2, [R11,#end_of_data]                      [2] end_of_data = v2 + 1
.text:0001D198                CMP     R3, #0                                                      [3] if ( !v2 ) { return 1 }
.text:0001D19C                BEQ     loc_1D23C
.text:0001D1A0                SUB     R3, R11, #-input_crc_str                    [4] char input_crc_str[8]
.text:0001D1A4                MOV     R0, R3          ; s
.text:0001D1A8                MOV     R1, #0          ; c
.text:0001D1AC                MOV     R2, #8          ; n
.text:0001D1B0                BL      memset                                                      [4] memset(input_crc_str, 0,
8)
.text:0001D1B4                SUB     R3, R11, #-input_crc_str
.text:0001D1B8                MOV     R0, R3          ; dest
.text:0001D1BC                LDR     R1, [R11,#end_of_data] ; src
.text:0001D1C0                BL      strcpy                                                      [5] strcpy(input_crc_str,
end_of_data)
```

The function identifies the CRC field by searching for the last semi-colon in the packet and then moving the pointer forward one byte. This is referred to in the code as `end_of_data` but may also be thought of as `start_of_crc`. The software allocates an 8-byte character array, called `input_crc_str`, on the stack and then will `strcpy` the CRC into `input_crc_str`. The call to `strcpy` is unbounded and no validation occurs to ensure the data following the last semi-colon is appropriately sized.

Therefore, supplying a UDP packet whose CRC field is sufficiently longer than expected will cause a straightforward 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  : 0x0
$r2  : 0x6b251d4c
$r3  : 0x1
$r4  : 0x0
$r5  : 0xb636b6b0  →  "eth0"
$r6  : 0x0
$r7  : 0x152
$r8  : 0xbefffbe0  →  0xb6ff86d0  →  0xb6ff8db8  →  0x00000001
$r9  : 0xb6ff86d0  →  0xb6ff8db8  →  0x00000001
$r10 : 0xb636c460  →  0x00000001
$r11 : 0x4d4d4d4d ("MMMM"?)
$r12 : 0xa
$sp  : 0xb636b6a0  →  0xb636b700  →  0x00000000
$lr  : 0x0001d22c  →  <check_udp_crc+248> mov r3,  #1
$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.