Talos Vulnerability Report

TALOS-2021-1354

Garrett Metal Detectors iC Module CMA run_server_6877 authentication bypass vulnerability

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-21902

Summary

An authentication bypass vulnerability exists in the CMA run_server_6877 functionality of Garrett Metal Detectors iC Module CMA Version 5.0. A properly-timed network connection can lead to authentication bypass via session hijacking. An attacker can send a sequence of requests 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

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

CWE

CWE-303 - Incorrect Implementation of Authentication Algorithm

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

The software responsible for the CLI server maintains two separate lists for authenticated and unauthenticated connections. When an unauthenticated connection provides a valid password the software will copy certain fields from the unauthenticated structure to an authenticated structure. There is a logical flaw in the spawning of authenticated threads that allows an attacker with no knowledge of the device’s password to exploit a race condition and supplant themselves as the socket associated with the recently authenticated connection.

Included below is a partial decompilation of the described flow.

```
activity = select(max_sd + 1, &readfds, 0, 0, 0);
ready_6877 = 0;
if ( activity >= 0 || *_errno_location() == 4 )
{
  if ( (readfds.__fds_bits[tcpFd / 32] & (1 << (tcpFd % 32))) != 0 )
  {
    new_socket = accept(tcpFd, (struct sockaddr *)&from, &len);
    if ( new_socket >= 0 )
    {
      if ( new_socket > max_sd )
        max_sd = new_socket;
      socket_timeout.tv_sec = 20;
      socket_timeout.tv_usec = 0;
      setsockopt(new_socket, 1, 20, &socket_timeout, 8u);
      setsockopt(new_socket, 1, 21, &socket_timeout, 8u);
      set_telnet_options(new_socket);
      v1 = strlen((const char *)version_string);
      send(new_socket, version_string, v1, 0x4000);
      send(new_socket, "\r\n", 2u, 0x4000);
      send(new_socket, passwordPrompt, 0xDu, 0x4000);
      printf("%i <-- %s\n", ++clientNum, (const char *)passwordPrompt);
      printf("%i <-- %s\n", clientNum, (const char *)passwordPrompt);
      memset(&new_client, 0, sizeof(new_client));
      new_client.sd = new_socket;                                                   [1] Assign the current
socket to new_client.sd, and add to unauth_client array
      new_client.client_num = clientNum;
      memcpy(&v39.buf[13], &new_client.buf[13], 0x6Fu);
      *(_QWORD *)&v39.sd = *(_QWORD *)&new_client.sd;
      *(_QWORD *)&v39.buf[5] = *(_QWORD *)&new_client.buf[5];
      add_client(v39);
    }
    else
    {
      perror("accept");
    }
  }
}

...

for ( i = 0; i <= 49; ++i )                                                     [2] Iterate over all active,
unauthenticated clients
{
  client = &unauthenticated_clients[i];
  if ( client->sd )
  {
    sd = client->sd;
    if ( (readfds.__fds_bits[sd >> 5] & (1 << (sd % 32))) != 0 )
    {
      if ( !recv(sd, buffer, 0x200u, MSG_PEEK | MSG_DONTWAIT) )
      {
        printf("\x1B[33mClient %i disconnected\n\x1B[0m", clientNum);
        shutdown(sd, 2);
        close(sd);
        reset_unauth_client(client);
      }
      v2 = &client->buf[strlen((const char *)client->buf)];
      v3 = strlen((const char *)client->buf);
      valread = read(sd, v2, 124 - v3);
      if ( valread <= 0 )
      {
        printf("socket %i on 6877 has disconnected!\n", sd);
        shutdown(sd, 2);
        close(sd);
        reset_unauth_client(client);
      }
      else if ( strchr((const char *)client->buf, '\n') || strchr((const char *)client->buf, '\r') || client[1].sd )
      {
        v4 = strcspn((const char *)client->buf, "\r\n");
        client->buf[v4] = 0;
        if ( checkPassword(client->buf) == 1 )                                    [3] Verify password
supplied by active socket
        {
          increment_in_ptr();
          thread_args[thread_in_ptr].sd = new_socket;                              [4] Set authenticated
client struct's socket descriptor to `new_socket` (from [1]), NOT client.sd (from [2])
          memcpy(&thread_args[thread_in_ptr].sockaddr, &from, sizeof(thread_args[thread_in_ptr].sockaddr));
          thread_args[thread_in_ptr].clientNumber = thread_in_ptr;
          ...
          pthread_attr_init(&attr);
          pthread_attr_setdetachstate(&attr, 1);
          pthread_create(&thread, &attr, (void *(*)(void *))handle_new_connection, &thread_args[thread_in_ptr]);    [5] Spawn authenticated
client handler thread using most recent `new_client`
          thread_register[thread_in_ptr] = thread;
          pthread_attr_destroy(&attr);
          reset_unauth_client(client);
        }
        ...
```

If an attacker spawns a connection to TCP port 6877 of a vulnerable device between the time a remote user (A) connects to the device and (B) supplies a valid password, it is the attacker's socket that will be copied at position `[4]` and not the remote user's, resulting in a hijack of the authenticated session and access to the CLI without prior knowledge of the password. During testing, there was approximately 0.5 seconds between the time the remote CMA software established the connection and sent the user's password.

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.

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.