Talos Vulnerability Report

TALOS-2021-1359

Garrett Metal Detectors iC Module CMA CLI del[env] command directory traversal vulnerabilities

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-21908,CVE-2021-21909

Summary

Directory traversal vulnerabilities exist in the CMA CLI del and delenv commands of Garrett Metal Detectors' iC Module CMA Version 5.0. Specially-crafted command line arguments can lead to arbitrary file deletion. An attacker can provide malicious inputs 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

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

CWE

CWE-22 - Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

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 configuration.

This service exposes a command that allows an authenticated user to delete historical application logs and event counter files from long term storage. The command, `del [file]`, expects as an argument one of the file names returned by the `ls` command. Log files are stored as `*.log` and are kept in `/mnt/flash/logs`. Count files are stored as `*.cnt` and are kept in `/mnt/flash/alarms`. If the provided `[file]` argument does not end in either `[*.cnt|*.log]` then the `del` function operates as an alias for the `delenv` command, which allows an authenticated user to delete application "environment variables", which are stored as files in the `/ltrx_user/env/` directory.

For reference, an approximate decompilation of the `handle_delete` and `delEnv` handler functions are included below. For brevity, functionality that was not relevant to these vulnerabilities (such as logging, error handling, and remote client interaction) has been excluded.

```
void handle_delete(uint8_t argc, char** argv, client_6877 *client)
{
  char filename[36];

  if ( argc == 2 )
  {
    if ( endsWith(argv[1], ".log") )
    {
      strcpy(filename, "/mnt/flash/logs/");
      strncat(filename, argv[1], 20u);
      remove(filename);
    }
    else if ( endsWith(argv[1], ".cnt") )
    {
      strcpy(filename, "/mnt/flash/alarms/");
      strncat(filename, argv[1], 18u);
      remove(filename);
    }
    else
    {
      delEnv(argv[1]);  // `del` serves as an alias for `delenv` in this case
    }
  }
}

void delEnv(char* key)
{
  // Please note that the function that handles receiving CLI data from remote clients
  // limits the value in `key` to less than 256 bytes
  char filename[128];
  // Therefore, this `log_buf` stack variable protects the stack frame from corruption
  // when attempting to overflow `filename`
  char log_buf[256];

  strcpy(filename, "/ltrx_user/env/");
  strcat(filename, key);
  if ( file_exists(filename) )
    remove(filename);
}
```

CVE-2021-21908 - delenv file delete

As shown in the above decompilation, the `handle_delete` function does not attempt to sanitize or otherwise validate the contents of the `[file]` parameter (passed to the function as `argv[1]`), allowing an authenticated attacker to supply directory traversal primitives and delete semi-arbitrary files. In the case that the attacker supplies a file path that does not end in `[*.cnt|*.log]`, or the attacker calls `delenv [key]` directly, the `delEnv` function is invoked. This function concatenates the base directory "`/ltrx_user/env`" with the user supplied `[key]` or `[file]` parameter, then deletes the resulting filepath.

The ability to delete arbitrary files from a file system can have a significant impact on the stability of any system, but on this system in particular the ability to delete arbitrary files also opens the system up to a persistent authentication bypass. Briefly, deleting `/ltrx_user/secret` causes the authentication logic to fall back to the default password. Were an attacker to successfully abuse the race condition described in TALOS-2021-XXXXX to gain authenticated access to the CLI, they could abuse this vulnerability to persist that access across connections by resetting and subsequently changing the password.

Exploit Proof of Concept

```
delenv ../../../ltrx_user/secret
```

CVE-2021-21909 - del .cnt|.log file delete

Similarly, were an attacker to call `delenv [file]` with a `[file]` argument ending in either `[*.cnt|*.log]` then the attacker supplied argument is first checked to determine which conditional branch is taken, and depending on the file extension supplied the parameter is `strncat` to one of `/mnt/flash/[alarms|logs]/` without any validation or sanitization. However, compared to the previous vulnerability, the files that may be targeted with this traversal are very restricted given that the `filename` array is limited to 36 bytes, 16 to 18 of which are taken up by the directory paths, `/mnt/flash/[logs|alarms]/`, and 9 more bytes are taken for the traversal primitive (`../../../`) needed to reach the root directory. The attacker ultimately has a maximum of 9 or 11 bytes to work with, depending on which file extension is used.

Note that the calls to `strncat` will only copy the first 18 or 20 bytes of the provided `[file]` argument, no matter the original length, which allows an attacker to craft a path that will pass the `endsWith` call checking for the appropriate file extension. This results in a `filename` that has been truncated prior to either file extension, allowing the attacker to delete arbitrary file types.

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.
