Talos Vulnerability Report

TALOS-2021-1395

Sealevel Systems, Inc. SeaConnect 370W OTA update task file overwrite vulnerability

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21968

Summary

A file write vulnerability exists in the OTA update task functionality of Sealevel Systems, Inc. SeaConnect 370W v1.3.34. A specially-crafted MQTT payload can lead to arbitrary file overwrite. An attacker can perform a man-in-the-middle attack to trigger this vulnerability.

Tested Versions

Sealevel Systems, Inc. SeaConnect 370W v1.3.34

Product URLs

SeaConnect 370W - https://www.sealevel.com/product/370w-a-wifi-to-form-c-relays-digital-inputs-a-d-inputs-and-1-wire-bus-seaconnect-multifunction-io-edge-module-powered-by-seacloud/

CVSSv3 Score

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

CWE

CWE-20 - Improper Input Validation

Details

The SeaConnect 370W is a Wi-Fi connected IIoT device offering programmable cloud access and control of digital and analog I/O and a 1-wire bus.

This device offers remote control via several means including MQTT, Modbus TCP and a manufacturer-specific protocol named “SeaMAX API”.

The device is built on top of the TI CC3200 MCU with built-in Wi-Fi capabilities.

The SeaConnect 370W implements an over the air firmware update mechanism which is controlled remotely from the “Sealevel SeaCloud” via an MQTTS connection. When a device comes online, it connects to the SeaCloud MQTTS broker and transmits its current firmware version. When an outdated firmware is detected, a message is published to that device’s command channel detailing the FTP(S) url containing the new image and the destination filename of the new image.

A specially-crafted MQTT message can lead to overwrite arbitrary “files” due to improper input validation.

The OTA update task is responsible for updating the firmware, if a new one is available. After a certain MQTT message is sent, the downloading process begins. The message that starts the downloading process is in the following form:

```
{
    "name": "u-download",
    "payload": "{
                \"uri\": \"<uri to firmware image>\",
                \"dest\": \"<local filename to write>\",
                \"crc\": \"<crc>\"
                }"
}
```

The `uri` is an FTP(S) URI. The downloading process revolves around FTP(S). There are several steps in the download process. The principal ones are `DoConnectedTick` and `DoDownloadingTick`.

Here is the relevant part of the `DoConnectedTick` function:

```
undefined4 DoConnectedTick(void)

{
[...]

ftp_command_ptr_ptr = read_volatile_4(FTP_COMMAND_PTR_PTR);
ftp_struct = read_volatile_4(FTP_struct);
pdVar3 = (dword *)read_volatile_4(remoteFileSize_ptr);
ftp_command_ptr = *ftp_command_ptr_ptr;
file_type = L'I';
is_error = get_file_size(ftp_struct->filepath,pdVar3,L'I',ftp_command_ptr);
puVar2 = read_volatile_4(PTR_s_DoConnectedTick_2000f4cc);
if (is_error == 0) {
    Report(aESFtpsizeFaile,(dword)puVar2,file_type,(dword)ftp_command_ptr);
    return 1;
}
Report(aSSizeOfSIsD,(dword)puVar2,(dword)ftp_struct->filepath,*pdVar3);
dest_file = read_volatile_4(PTR_OTAUpdateStruct_2003241c.dest[0]_2000f4d0);
puVar1 = (undefined4 *)read_volatile_4(FILE_HANDLER_FTP_DEST);
attempt_number = 0;
do {
    Report(aSOpeningLocalF,(dword)puVar2,(dword)dest_file,attempt_number + 1);
    dVar5 = 0;
    open_ret = sl_extlib_FlcOpenFile(dest_file,0x2ac00,0,puVar1,1);                    [1]
    [...]
}
```

This function will get the size of the file specified in the `uri` and call the `sl_extlib_FlcOpenFile` function at `[1]`. This function will try to open or create the “file” specified at `dest` in the MQTT message. Once the “file” is created or opened a file handler is saved for later use.

Here is the relevant part of the `DoDownloadingTick` function:

```
undefined4 DoDownloadingTick(void)

{
  dword *pdVar1;
  FTP_command **ppFVar2;
  undefined *func_ptr;
  dword *pdVar3;
  undefined new_status;
  int get_file_successed;
  char *pcVar4;
  dword dVar5;
  FTP_command *big_struct_seems;
  undefined *filepath;

  ppFVar2 = (FTP_command **)read_volatile_4(PTR_PTR_2000f574);
  FUN_2000be7a(1,1,*ppFVar2);
  filepath = read_volatile_4(PTR_FTP_Struct_20032bd4.filepath[0]_2000f898);
  func_ptr = read_volatile_4(FileWriteCB);
  big_struct_seems = *ppFVar2;
  dVar5 = L'I';
  get_file_successed = FtpGet(func_ptr,(char)filepath,'I',big_struct_seems);
  [...]
}
```

This function will download the file from the FTP(S) server and write it into a “file” using the file handler obtained during the `DoConnectedTick` step. After the downloading and writing process, different checks are perfomed, like: size matching between the one obtained in `DoConnectedTick` and the one obtained during the `DoDownloadingTick` phase, and matching between the `crc` provided in the MQTT message and the one calculated on the downloaded file. These checks are performed after the content of the “file” is already written.

The OTA update task does not enforce any validation check on the `dest` field of the `MQTT` message. So any “file” could potentially be created or opened. Because at `[1]` the size specified, the second argument, is so big, it seems that the only “files” that are possible to create are the ones related with the `mcuimg`. But, if the “file” already exist, the Texas Instrument library will not consider the size into the opening operation. This would allow an attacker who perfomed a man-in-the-middle attack to overwrite arbitrary “files”.

For example if the MQTT message is:

```
{
    "name": "u-download",
    "payload": "{
                \"uri\":\"ftp://<username>:<password>@<ip>:<port>/<filepath>\",
                \"dest\":\"/usr/settings.bin\",
                \"crc\":\"41414141\"
                }"
}
```

The consequence would be to overwrite the `/usr/settings.bin` “file”. If crafted in such a way that the total size is less than 0x15 and the first byte is `\x0A` this would result in setting the HTTP's basic auth password to a known value. This would result in giving the attacker possibilities to perform arbitrary HTTP requests.

Timeline

2021-10-26 - Vendor disclosure
2022-02-01 - Public Release

CREDIT

Discovered by Francesco Benvenuto and Matt Wiseman of Cisco Talos.
