Talos Vulnerability Report

TALOS-2021-1394

Sealevel Systems, Inc. SeaConnect 370W OTA update task out-of-bounds write vulnerability

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21967

Summary

An out-of-bounds 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 denial of service. 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

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

CWE

CWE-120 - Buffer Copy without Checking Size of Input (‘Classic Buffer Overflow’)

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 When an outdated firmware is detected, a message is published to that device’s command channel detailing the FTP(S) URL containing the new image and the destination filename of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 a stack-based buffer overflow in the OTA update task, due to the use of the unsafe function `strcpy` from a not null-terminated string.

The function responsible for parsing this OTA message is `ParseToDownloadMessage`:

```
undefined4 ParseToDownloadMessage(OTAUpdateStruct *otastruct_obj,char *payload)

{
undefined *puVar1;
undefined *puVar2;
size_t sVar3;
void *parsed_payload;
int jObj;
char *temp_array;
dword in_r2;
dword in_r3;
undefined jParser [4];
dword local_2c;

sVar3 = strlen(payload);
if (sVar3 >> 8 == 0) {
    sVar3 = strlen(payload);
}
else {
    sVar3 = 0x100;
}
parsed_payload = (void *)malloc(sVar3);
puVar1 = read_volatile_4(PTR_s_ParseToDownloadMessage_20010394);
if (parsed_payload == (void *)0x0) {
    sVar3 = strlen(payload);
    if (sVar3 >> 8 == 0) {
    sVar3 = strlen(payload);
    }
    else {
    sVar3 = 0x100;
    }
    Report(aErrorSeaconnec_1,(dword)puVar1,sVar3,in_r3);
}
else {
    unescape(parsed_payload,payload);
    unquote(parsed_payload);
    jObj = json_parser_init(jParser,parsed_payload);
    if (jObj == -1) {
    Report(aErrorSeaconnec_0,(dword)puVar1,in_r2,in_r3);
    }
    else {
        puVar2 = read_volatile_4(p_Report);
        json_parser_dump(jParser,puVar2);
        if (0 < (int)local_2c) {
            temp_array = (char *)malloc(0x100);
            if (temp_array == (char *)0x0) {
                Report(aErrorSeaconnec,(dword)puVar1,0x100,in_r3);
            }
            else {
                json_object_get_string(jParser,jObj,aUri,temp_array);
                strncpy((char *)otastruct_obj,temp_array,0xff);
                json_object_get_string(jParser,jObj,aDest,temp_array);
                strncpy((char *)otastruct_obj->dest,temp_array,0xff);                                   [1]
                json_object_get_string(jParser,jObj,aCrc,temp_array);
                sscanf(temp_array,aX,&otastruct_obj->crc);
                free(temp_array);
    [...] }
```

This function takes as argument a `OTAUpdateStruct` struct pointer that will be filled with the info contained in `payload`. The `OTAUpdateStruct` struct is defined as follow:

```
struct OTAUpdateStruct{
        char        uri[0x100];
        char        dest[0x40];
        uint32_t    crc;
    };
```

The `payload` variable is a string that will be parsed as a json to fill the `otastruct_obj` variable. The json should contain, among the keys, the `dest` one. The value of the `dest` json key is used at `[1]` to populate the `otastruct_obj`'s `dest` field. After the underlying `OTAUpdateStruct`, pointed by `otastruct_obj`, is populated, the `copy_update_structure_and_signal` function is called:

```
void copy_update_structure_and_signal(OTAUpdateStruct *OTA_struct)

{
undefined *temp_ptr;

temp_ptr = read_volatile_4(PTR_OTAUpdateStruct_2000d7a0);
sl_Memcpy(temp_ptr,OTA_struct,0x144);                                                                   [2]
temp_ptr = read_volatile_4(pg_startDownloadEvent);
probably_queue_signal((char)temp_ptr);
return;
}
```

The function performs two actions: 1) at `[2]` copy the object populated in `ParseToDownloadMessage` in a location used by the OTA update task. 2) signal to the OTA update task that a payload is ready to be parsed.

Eventually the OTA task will call the `SeaConnectOTADownload_file` function:

```
bool SeaConnectOTADownload_file
            (OTAUpdateStruct *OTA_struct,undefined4 param_2,undefined4 param_3,dword param_4)

{
[...]
dword dVar4;
OTAUpdateStruct_without_crc OTAUpdate_struct_without_crc;

puVar1 = read_volatile_4(DEFAULT_OTAStruct_WITHOUT_CRC);
dVar4 = 0x140;
sl_Memcpy(&OTAUpdate_struct_without_crc,puVar1,0x140);
strcpy((char *)&OTAUpdate_struct_without_crc,(char *)OTA_struct);
strcpy((char *)OTAUpdate_struct_without_crc.dest,(char *)OTA_struct->dest);                             [3]
[...]
}
```

The variable `OTA_struct` is a pointer to the data copied at `2`.

The `OTAUpdateStruct`'s dest field is only 0x40 bytes, but at `[1]` up to 0xff are copied from the json. Because the `OTAUpdateStruct` struct used at `[1]` is just before the one used at `[2]`, this oveflow will not cause a security issue by itself. But the overflow allows the `OTAUpdateStruct`'s `dest` string to not have a null terminator. In `SeaConnectOTADownload_file` the `OTAUpdate_struct_without_crc` struct used is similar to the `OTAUpdateStruct` one but without the `crc` field. The `OTA_struct`'s `uri` and `dest` fields are copied to the correspondent fields in `OTAUpdate_struct_without_crc`, using `strcpy`. Because the `OTA_struct`'s `dest` field is not null-terminated, the `strcpy` will copy the `OTA_struct`'s `crc` field and everything following it up to encouter a null terminator, resulting in a stack-based buffer overflow.

Here is the beginning of the `SeaConnectOTADownload_file` function in assembly:

```
2000cf1e 10 b5          push                        { r4, lr }                                          [4]
2000cf20 04 46          mov                         r4, r0
2000cf22 4f f4 a0 72    mov.w                       r2, #0x140
2000cf26 ad f5 a0 7d    sub.w                       sp, sp, #0x140                                      [5]
2000cf2a 68 46          mov                         r0, sp
2000cf2c 14 f0 e2 ff    bl                          sl_Memcpy
2000cf30 68 46          mov                         r0, sp
2000cf32 21 46          mov                         r1, r4
2000cf34 18 f0 42 fe    bl                          strcpy
2000cf38 40 a8          add                         r0, sp,#0x100
2000cf3a 04 f5 80 71    add.w                       r1, r4, #0x100
2000cf3e 18 f0 3d fe    bl                          strcpy                                              [6]
```

At `[4]` this function pushes `r4` and `lr` into the top of the stack, then at `[5]` the function reserves 0x140 bytes of space, 0x100 for the `uri` and 0x40 for the `dest`. At `[6]` the `strcpy` is copying the `OTA_struct`'s `dest` field into a 0x40 sized buffer. Based on the MQTT message this could cause a buffer overflow overwriting the `r4` register with the `CRC` and `lr` with what follows in memory.

For example with a MQTT message like this:

```
{
    "name": "u-download",
    "payload": "{
                \"uri\":\"A\",
                \"dest\":\"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\",
                \"crc\":\"41414141\"
                }"
}
```

After the instruction at `[4]` the top of the stack would look likes:

```
0x200372d8:     (r4) 0x20031928      (lr) 0x2000d3a7
```

And at the end of the function:

```
0x200372d8:     (r4) 0x41414141      (lr) 0x2000d300
```

The `r4` register was overwritten with the `crc` field, and the `lr`'s first byte was overwritten with the null terminator found after the `crc`. This would result in a crash of the device.

Timeline

2021-10-21 - Initial vendor contact
2021-10-26 - Vendor disclosure
2022-02-01 - Public Release

CREDIT

Discovered by Francesco Benvenuto and Matt Wiseman of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2021-1392

NEXT REPORT

TALOS-2021-1395