Talos Vulnerability Report

TALOS-2021-1390

Sealevel Systems, Inc. SeaConnect 370W OTA Update "u-download" heap-based buffer overflow vulnerability

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21962

Summary

A heap-based buffer overflow vulnerability exists in the OTA Update u-download functionality of Sealevel Systems, Inc. SeaConnect 370W v1.3.34. A series of specially-crafted MQTT payloads can lead to remote code execution. An attacker must perform a man-in-the-middle attack in order 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

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

CWE

CWE-122 - Heap-based 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 When an outdated firmware is detected, a message is published to that device’s `command` channel detailing the FTP(S) url containing the new image. The parsing of this message is vulnerable to a heap overflow, which can be leveraged to create a write-what-where primitive, allowing a remote attacker to overwrite arbitrary instructions and cause their execution.

The message published to the `command` channel to initiate a firmware update is a JSON blob which the system expects will be structured as follows:

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

Note the `payload` field of the JSON blob is expected to be a string containing a quote-escaped JSON blob. When an MQTT message is received with a `name` field containing `u-download`, the `payload` field of that message will be passed as an argument to a function named `ParseToDownloadMessage`, located at raw offset 0xFFFC, which is mapped to RAM for execution at 0x2000FFFC. This function is responsible for parsing the payload and populating a structure that will be used by a separate task to begin downloading and applying the described firmware update. The relevant portion of the decompilation for this function is included below. This portion of the function is responsible for unescaping the quotes contained in the `payload` string so that it may be parsed as a JSON object.

```
int __fastcall ParseToDownloadMessage(OTAUpdateStruct *a1, char *payload)
{
  int len; // r0
  char *reformattedPayload; // r0 MAPDST
  const char *v6; // r8
  int v8; // r2
  int result; // r0
  int v10; // r5
  char *v11; // r7
  jsonParser jParser; // [sp+0h] [bp-30h] BYREF

  len = MIN(strlen(payload), 256);
  reformattedPayload = malloc(len);             [1] Allocate either the length of the payload, or 256, whichever is smaller
  if ( !reformattedPayload )
  {
    Report("ERROR: SeaConnectOtaUpdate %s Allocation of reformattedPayload (%d bytes) failed\r\n", "ParseToDownloadMessage", len);
    return 0;
  }
  unescape(reformattedPayload, payload);        [2] Unescape the quotes into the recently allocated buffer, ignoring the actual length of
the payload
  ...
}
```

As noted above, at `[1]` a buffer is allocated that is the minimum of 256 or the length of the payload. Then, this newly allocated buffer and the original `payload` are passed as arguments to `unescape`. The `unescape` function is very straightforward - it acts like a call to `strcpy` that discards `\` characters. An annotated decompilation is included below for reference.

```
void unescape(char *dest, char *src)
{
  int src_idx;
  int dest_idx;
  char curr_char;

  for ( src_idx = 0; src_idx < strlen(src); src_idx++ )
  {
    curr_char = src[src_idx];
    if ( curr_char == '\\' )
    {
      continue;
    } else {
      dest[dst_idx] = curr_char;
      dst_idx++;
    }
  }
  dest[dst_idx] = '\0';
}
```

For the purposes of this vulnerability, viewing `unescape` as an alias for `strcpy` makes it clear that the call to `unescape` at `[2]` can result in a buffer overflow. Supplying a payload larger than 256 bytes will cause an overflow on the heap. A particularly-formatted overflow can corrupt heap metadata in such a way as to allow an attacker to control the pointer that will be returned by subsequent calls to `malloc`. In practice, controlling the next allocated pointer is enough to gain remote code execution.

Crash Information

```
[Hard fault handler]
R0       = 0x00000000
R1       = 0x00000000
R2       = 0x00000000
R3       = 0x200372dd
R12      = 0x20032bd4
LR [R14] = 0x2000cf51  subroutine call return address
PC [R15] = 0x2000d300  Program Counter
PSR      = 0x60000000
BFAR     = 0xe000ed38
CFSR     = 0x00020000
HFSR     = 0x40000000
DFSR     = 0x00000000
AFSR     = 0x00000000
```

Traversing the FreeRTOS heap freelist after corruption, we observe that one of the `BlockLink_t`s has been modified to point to our target address.

```
xStart => {
  pxNextFreeBlock = 0x2003a418,
  xBlockSize = 48
}
0x2003a418 => {
  pxNextFreeBlock = 0x41414141,
  xBlockSize = 23392
}
0x41414141 => {
  pxNextFreeBlock = 0x0,
  xBlockSize = 0
}
```

Timeline

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

CREDIT

Discovered by Francesco Benvenuto and Matt Wiseman of Cisco Talos.