Talos Vulnerability Report

TALOS-2021-1234

EIP Stack Group OpENer Ethernet/IP UDP handler information disclosure vulnerability

JUNE 16, 2021

CVE NUMBER

CVE-2021-21777

Summary

An information disclosure vulnerability exists in the Ethernet/IP UDP handler functionality of EIP Stack Group OpENer 2.3 and development commit 8c73bf3. A specially crafted network request can lead to an out-of-bounds read.

Tested Versions

EIP Stack Group OpENer 2.3
EIP Stack Group OpENer development commit 8c73bf3

Product URLs

https://github.com/EIPStackGroup/OpENer

CVSSv3 Score

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

CWE

CWE-125 - Out-of-bounds Read

Details

OpENer is an EtherNet/IP stack for I/O adapter devices. It supports multiple I/O and explicit connections and includes objects and services for making EtherNet/IP-compliant products as defined in the ODVA specification.

An Ethernet/IP protocol header always consists of 24 bytes. The first 2 bytes are for the command code, followed by another two bytes for the length of the packet.
Due to an integer conversion bug, a request with a maliciously high length field, about `0x8000`, will be treated as a signed negative short integer. After processing a request packet, the code will use the provided length value to try and jump the specified amount of bytes forward, past the packet, to parse a potential follow-up request. By abusing the length field, this jump can also be directed up to roughly 32000 bytes in front of the buffer. Whatever is found there in memory will be treated as a command packet, the bytes parsed and acted upon. Should the first 2 bytes found correspond to a valid opcode, and the follow-up 2 bytes in the size location there be bigger than 32K, the next jump will be even further up to lower memory addresses, and so on.
There are several opcodes which will return part of the request packet. By sending a request to jump into existing memory in front of a malicious packet, if one of several possible matching opcodes is found there, a return packet will be sent to the attacker based on what is found in memory at this address, including bytes from the parsed data. This way arbitrary memory may be returned to an attacker, resulting in an information leak. This attack only works using UDP when communication with the Opener server.

The problematic function is `CheckAndHandleUdpUnicastSocket` in `generic_networkhandler.c`:

```
void CheckAndHandleUdpUnicastSocket(void) {
  /* see if this is an unsolicited inbound UDP message */
  if(true == CheckSocketSet(g_network_status.udp_unicast_listener)) {

    struct sockaddr_in from_address = { 0 };
    socklen_t from_address_length = sizeof(from_address);

    OPENER_TRACE_STATE(
        "networkhandler: unsolicited UDP message on EIP unicast socket\n");

    /* Handle UDP broadcast messages */
    CipOctet incoming_message[PC_OPENER_ETHERNET_BUFFER_SIZE] = { 0 };
    int received_size = recvfrom(g_network_status.udp_unicast_listener, NWBUF_CAST incoming_message, sizeof(incoming_message), 0,
      (struct sockaddr*) &from_address, &from_address_length);

    if(received_size <= 0) { /* got error */
        ...
    }

    OPENER_TRACE_INFO("Data received on UDP unicast:\n");

    EipUint8 *receive_buffer = &incoming_message[0];
    int remaining_bytes = 0;
    ENIPMessage outgoing_message;
    InitializeENIPMessage(&outgoing_message);
    do {
      // [1]
      EipStatus need_to_send = HandleReceivedExplictUdpData(g_network_status.udp_unicast_listener,
                                                            &from_address,
                                                            receive_buffer,
                                                            received_size,
                                                            &remaining_bytes,
                                                            true,
                                                            &outgoing_message);

      receive_buffer += received_size - remaining_bytes;
      received_size = remaining_bytes;

      if(need_to_send > 0) {
        OPENER_TRACE_INFO("UDP unicast reply sent:\n");

        /* if the active socket matches a registered UDP callback, handle a UDP packet */
        if(sendto(g_network_status.udp_unicast_listener, (char*) outgoing_message.message_buffer, outgoing_message.used_message_length, 0,
          (struct sockaddr*) &from_address, sizeof(from_address)) != outgoing_message.used_message_length) {
          OPENER_TRACE_INFO(
              "networkhandler: UDP unicast response was not fully sent\n");
        }
      }
    } while(remaining_bytes > 0);
  }
}
```

At [1], this function passes the UDP data to `HandleReceivedExplictUdpData`, passing `received_size` as a signed integer.

```
EipStatus HandleReceivedExplictUdpData(const int socket,
                                       const struct sockaddr_in *from_address,
                                       const EipUint8 *buffer,
                                       const size_t buffer_length,
                                       int *number_of_remaining_bytes,
                                       bool unicast,
                                       ENIPMessage *const outgoing_message) {
  EipStatus return_value = kEipStatusOk;
  EncapsulationData encapsulation_data = { 0 };
  /* eat the encapsulation header*/
  /* the structure contains a pointer to the encapsulated data*/
  /* returns how many bytes are left after the encapsulated data*/
  *number_of_remaining_bytes = CreateEncapsulationStructure(buffer,
                                                            buffer_length,         // [2]
                                                            &encapsulation_data);
  ...
```

In turn, `HandleReceivedExplictUdpData` calls the function `EipInt16 CreateEncapsulationStructure` [2], where the length is parsed as a signed variable, and a final length calculation uses this signed number for the return value [3].

```
EipInt16 CreateEncapsulationStructure(const EipUint8 *receive_buffer,
                                      int receive_buffer_length,
                                      EncapsulationData *const encapsulation_data)
{
  encapsulation_data->communication_buffer_start = (EipUint8 *) receive_buffer;
  encapsulation_data->command_code = GetUintFromMessage(&receive_buffer);
  encapsulation_data->data_length = GetUintFromMessage(&receive_buffer);
  encapsulation_data->session_handle = GetUdintFromMessage(&receive_buffer);
  encapsulation_data->status = GetUdintFromMessage(&receive_buffer);

  memcpy(encapsulation_data->sender_context, receive_buffer,
         kSenderContextSize);
  receive_buffer += kSenderContextSize;
  encapsulation_data->options = GetUdintFromMessage(&receive_buffer);
  encapsulation_data->current_communication_buffer_position =
    (EipUint8 *) receive_buffer;
  return (receive_buffer_length - ENCAPSULATION_HEADER_LENGTH -
          encapsulation_data->data_length);                                  // [3]
}
```

This issue can be also abused for a denial-of-service. If the length field in a request makes the parser jump back to the start of the packet, the same packet will be processed over and over again, resulting in 100% CPU usage. This can be triggered with a single 24 bytes UDP packet

Moreover, this issue can also be abused for a distributed denial-of-service. By combining the above DOS request with a spoofed source IP address for the request, an opcode that results in a reply to the specified source address will result in a continuous stream of replies to a victim, which cannot be stopped remotely. This can be triggered by a single UDP packet.

While the server is bound to a specific interface during launch, a listening socket is opened by default on all available interfaces, for a “broadcast mode”. The attack works not just on the specified, but all available interfaces that were bound during startup.

Timeline

2021-01-23 - Vendor Disclosure
2021-04-20 - Disclosure deadline extended
2021-06-08 - Talos confirmed patch/fix

2021-06-16 - Public Release

CREDIT

Discovered by Martin Zeiser of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1229

NEXT REPORT TALOS-2021-1277