ARMIS®

Crit.IX

WHITE PAPER

Uncovering Critical Vulnerabilities in Honeywell Experion® Platforms for Distributed Control Systems

by Dov Milgrom, Barak Hadad and Tom Gol

Table of Contents

Introduction

In recent years, there has been a noticeable rise in attacks directed towards Operational Technology (OT) targets. This trend highlights the growing risks encountered by critical infrastructure systems, as both cyber criminals and nation state actors have come to recognize the immense value in targeting such systems. Security experts at the Armis Research Labs, discovered 9 vulnerabilities in Honeywell's Experion® platforms for distributed control systems (DCS) which are widely used in a variety of industries, from agriculture and water management to pharmaceuticals and nuclear plants. Seven of the vulnerabilities are critical and allow remote code execution. Exploiting these vulnerabilities can lead to anything from DoS and ransom demand to destruction of equipment and danger to its surroundings. The vulnerabilities were found in Honeywell's proprietary CDA protocol and the vendor's implementation of the CDA Data Client Named Access protocol. Since the discovery of the vulnerabilities Armis and Honeywell have been working together to carefully uncover their impact radius and provide ongoing mitigations.

Who We Are

Our security experts at **Armis Research Labs** are committed to being at the forefront of technology and security. By employing advanced research techniques like protocol analysis and code inspection, we diligently examine diverse systems, from legacy industrial machinery to cutting-edge cloud computing infrastructure to proactively identify and mitigate potential security risks. Whether it's a connected home appliance, an unassuming IoT device or a critical control system, our mission is to leave no stone unturned in our pursuit of a safer digital landscape.

Our previous research includes:

TLStorm Learn More	Three critical vulnerabilities discovered in APC Smart-UPS devices which can allow attackers to remotely manipulate the power of millions of enterprise devices.
PwnedPiper Learn More	Nine vulnerabilities in Swisslog PneumaticTubes System (PTS) - a critical infrastructure used in hospitals.
Modipwn Learn More	Authentication bypass leads to remote-code-execution in Schneider Electric Modicon PLCs.
NAT Slipstreaming 2.0 Learn More	Authentication bypass leads to remote-code-execution in Schneider Electric Modicon PLCs.
EtherOops Learn More	Exploit utilizing packet-in-packet attacks on ethernet cables to bypass firewalls & NATs.

CDPwn Learn More	Five critical vulnerabilities in various implementations of the Cisco Discovery Protocol.
URGENT/11 Learn More	11 Zero-Day vulnerabilities impactingVxWorks, the most widely used Real-Time Operating System (RTOS).
BLEEDINGBIT Learn More	Two chip-level vulnerabilities in Texas Instruments BLE chips, embedded in Enterprise-grade Access Points.
BlueBorne Learn More	An attack vector targeting devices via RCE vulnerabilities in Bluetooth stacks used by over 5.3 Billion devices.

What We Found in Crit.IX

Researchers at Armis Research Labs revealed weak points in the CDA protocol – a proprietary protocol designed by Honeywell and used to communicate between Honeywell Experion® server and C300 controller. This protocol lacks encryption and proper authentication mechanisms, which allows anyone with an access to the network to impersonate both controller and the server. In addition, there are design flaws in the CDA protocol which make it hard to control the boundaries of the data and lead to buffer overflows.

Honeywell also implements a CDA Data Client Named Access protocol on the Experion Server, which is used to communicate between Honeywell Experion® server and Experion® applications allowing for tag name access by these applications. Honeywell's implementation of this protocol was found to contain 4 vulnerabilities, most of which allow remote code execution (RCE) on the Experion® Server.

Distributed Control Systems (DCS)

A Distributed Control System (DCS) is an automated control system that streamlines the functionalities of different devices used throughout a workspace. It is a digital automated industrial control system that uses geographically distributed control loops throughout a factory, machine, or control area. Unlike a centralized control system that operates all machines, a DCS enables each part of a machine to have its own dedicated controller that runs the operation. DCSes are made up of control elements distributed throughout a plant or factory, including computers, sensors, and controllers, each serving a specific purpose such as data collection, data storage, or process control. The core capability of a DCS is to orchestrate plant-wide operations, supervising and coordinating each of the many controllers that are deployed across a plant. The DCS concept increases reliability and reduces installation costs by localizing control functions near the process plant, with remote monitoring and supervision.

DCSes are used in various industries such as chemical plants, oil and gas, food processing, nuclear power plants, water management systems, automobile industries and more.

DCS Architecture

In a distributed control system (DCS) architecture, the system is typically divided into several levels, each serving specific functions and consisting of various components. These levels and components work together to enable effective control and monitoring of industrial processes. The levels of DCS architecture can vary depending on the specific implementation, but the following levels are commonly found:

- **Field Level:**
  At the lowest level of the hierarchy, the field level comprises physical field devices such as sensors, actuators, and control valves. These devices interact directly with the industrial processes, collecting data and converting it into electrical or digital signals. The field level includes devices like temperature sensors, pressure transmitters, flow meters, and motorized valves.
- **Control Level:**
  The control level is where the controllers and I/O modules are located. It receives data from the field devices and processes it using control algorithms. The controllers execute control strategies and generate output signals to regulate the industrial processes. The I/O modules serve as the interface between the controllers and field devices, converting signals from the field devices into a format that can be processed by the controllers. The control level includes devices like DCS controllers, I/O modules, and control valves.
- **Supervisory Level:**
  The supervisory level provides a higher level of control and monitoring. It includes operator stations, servers and engineering workstations. Operators at this level monitor the overall system, view real-time data, and make informed decisions based on the system's performance. Servers are responsible for moving data between the operator stations and the controllers. Engineering workstations are used for system configuration, programming, and maintenance.
- **Advance Control Level:**
  Refers to a higher level of control and optimization strategies implemented on top of the basic control functions. It involves advanced algorithms and techniques to improve the overall performance, efficiency, and stability of industrial processes. Manufacturing execution systems (MES) normally operate on this level.
- **Enterprise Level:**
  The enterprise level is the highest level in the DCS architecture and integrates the DCS with broader business processes. It typically includes business management systems such as enterprise resource planning (ERP).

Distributed Control Systems (DCS) architecture stands out in Industrial Control Systems (ICS) with its centralized supervisory control, dedicated controllers, distributed processing, scalability, and integration with enterprise systems. These features enable comprehensive control, precise monitoring, efficient data processing and flexible expansion.

A Distributed Control System (DCS) Example

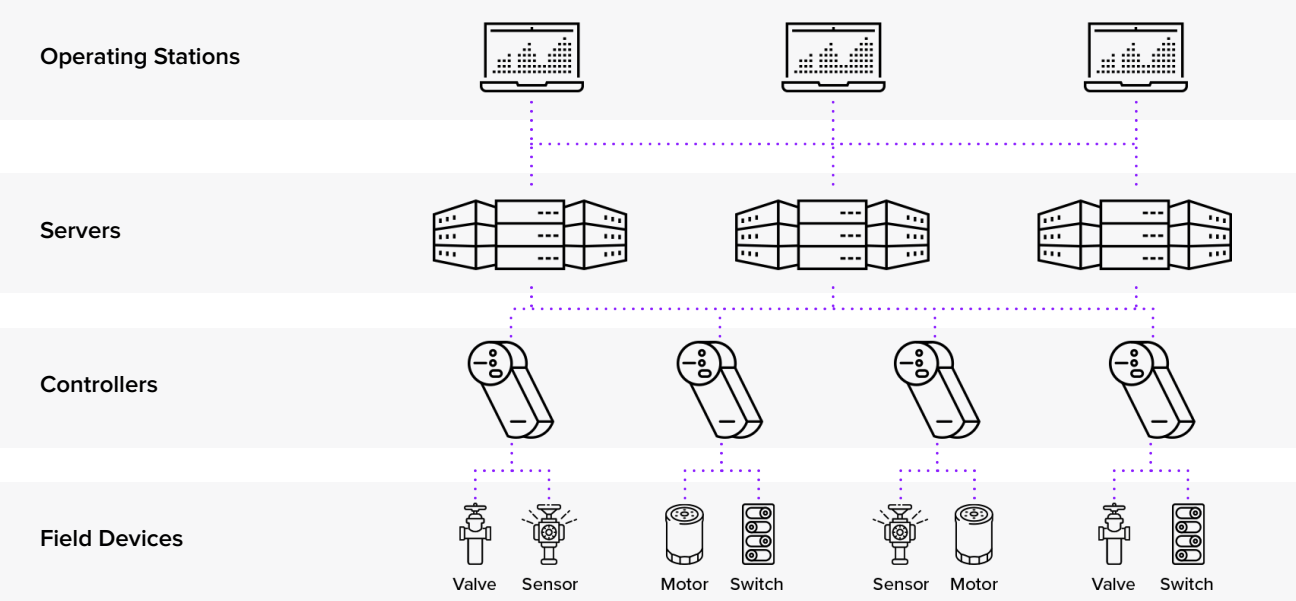

Figure 1. Distributed control system (DCS) architecture.

Honeywell Experion® Research

At Armis Research Labs, we study a great variety of devices in order to understand their protocols and security posture. Once we have a good understanding of a protocol, we can monitor the relevant device's usage for abnormalities and detect attackers as well as malfunctions.

One of the systems we analyzed is the Honeywell Experion® platform which includes the Experion® server and the C300 DCS controller.

Honeywell implements several protocols for various purposes. The interesting ones for this research are:

- **Generic Communication Layer (GCL) name service** - A unicast service which is responsible for notifying the network about the existence of the controller and a few of its basic properties. The GCL name service is also responsible for telling devices that want to connect to the controller which port to use for each protocol/service.
- **Control Data Access (CDA) protocol** - Used by the management server to read and write controller properties.

The CDA server service is responsible for communicating with controllers using the CDA protocol. Its purpose is to link C300 controllers with the Experion® Server and console stations.

So to recap - we have two interfaces of the control data access server that we want to check, the CDA Data Client Named Access server side, and the CDA client side. Since the behavior of the client and the server in CDA protocol is not symmetric, we had to examine both the server and the controller.

Analyzing C300 DCS Controller

The C300 controller has a PowerPC processor and runs the INTEGRITY operating system. After detecting the relevant process that implements the CDA protocol, we reviewed it and found something interesting in the deserialization process.

The controller processor is Big Endian and it usually communicates with the x86 processor which is Little Endian. This is why the protocol is designed to let either the client or the server fix the endianness of the data. The connection request of this protocol has a flag that determines who fixes the endianness, the client or the server. Honeywell implemented a library called "ntoh" which both the client and server use to process and fix data endianness.

The "ntoh" library takes every message and checks its type. According to the type, the library parses the structure of the message and swaps out the bytes of every field individually. This library doesn't make use of the content of the message itself, but still has to process every field that is related to type or size just to fix the endianness. For this reason, this library is very useful during the analysis process. It has all the information about the structure of the messages in one place, and even if we don't know the meaning of each field, we have enough info to successfully parse any packet.

While examining this library, we discovered that there are almost no boundary checks on the client input. This leads to some memory vulnerabilities that can allow remote code execution. Since this library is used the same way in the client controller and the management windows server, those vulnerabilities affect both the C300 controller and the Experion® server.

After finding the vulnerabilities in the "ntoh" library, we started mapping all the network facing attack surfaces.

Analyzing CDA Service

The C300 controllers communicate to the CDA service on the Experion® server using CDA Data Client Protocol and doesn't require any authentication. The purpose of this protocol is to exchange data between Experion Server / Console Station and Experion Controllers. It has multiple methods to read, write, and change properties in a controller.

When the server gets a read request on an item, it will send a request via the CDA protocol to the controller asking for that property, and then send the response back to the CDA Data Client Named Access client.

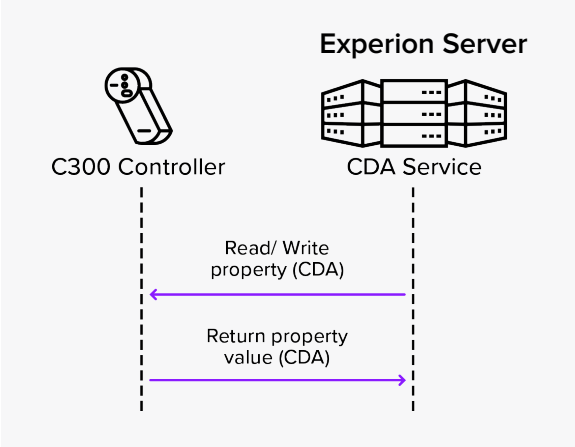

Figure 2. Experion server communicates with the controller via CDA

Aside from the methods that allow a read and write properties, the CDA Data Client Named Access Interface also has a few methods for controlling the components in the system:

- Add more components to the server database
- Get the list of all components
- Get the IP for each component
- Load more types of components to the server
- Debug command for dumping the group and handles for this client

A short examination made clear that this interface has many weak points. Most functions don't perform any security checks, and the ones that do, have poor implementation. These security issues are severe and can lead to denial of service, info leak, and even remote code execution.

Honeywell Protocols

Honeywell uses a set of protocols, each with its own purpose:

- **GCL** - Generic Communication Layer. A wrapping layer for Honeywell protocols.
- **CDA** - Control Data Access. Used to communicate between the C300 controller and the CDA service on the Experion® Server.

GCL Protocol

All of the Honeywell protocols have a GCL (Generic Communication Layer) header prefix. The GCL header contains metadata about the service and the client. As part of the connection handshake, the client sends a "verify connection" request to the server containing the name and identifier of the service.

Verify Connection Request

The "verify connection" request contains an LE or BE (Little Endian or Big Endian) flag which determines who needs to fix the endianness of the data, the client or the server. It also contains the name of the service we connected to. This is just for verification, not for choosing a service. Examples for services:

- **PsCdaCeeNtComm** - Control Data Access service
- **PsCdaSrvNamAcc** - CDA Data Client Protocol
- **EpicMo** - Controller Management and firmware upgrade service

CDA Protocol

CDA messages have two layers:

1. **Message header** - Containing type, size of the message and number of requests in the second layer.
2. **Request list** - A list of requests, each one representing an action on a single property, its size, type, and in the case of a write request, also the content.

This protocol is used to get and set parameters of the controller. In this protocol, the GCL header is followed by a message header which specifies the header size, message type, etc.

Request Structure

Each request comprises the following:

- **ID** - Unique only for this message, used to link request and response.
- **Size** - Total size of request header together with the data.
- **Req or Resp** - Indicates if this is a request or response.
- **Type - Data type** (int, string, double etc.). This field also determines the actual data size for basic types. Inconsistencies between the size and the type can lead to unwanted results.
- **Device Address** - Address of the device we query.
- **Parameter ID** - The ID of the parameter the request is for.
- **Data** will follow the request header in case of a write request.

Analyzing CDA Data Client Named Access Service

CDA Data Client Named Access protocol allows Experion applications needing named access to read, write, delete or subscribe to changes of an item.

First the client creates a new group identified by a unique group ID. Each group contains an array of items, each of them as specific type, size and a unique identifier. For each group, the user can decide whether they want to use a name to identify the items. This will be done by loading the data about this handle from an internal DB. The user can also describe the handle by itself. This is why for each method (READ, WRITE etc.) there are two options. For example, ReadNames and ReadHandles.

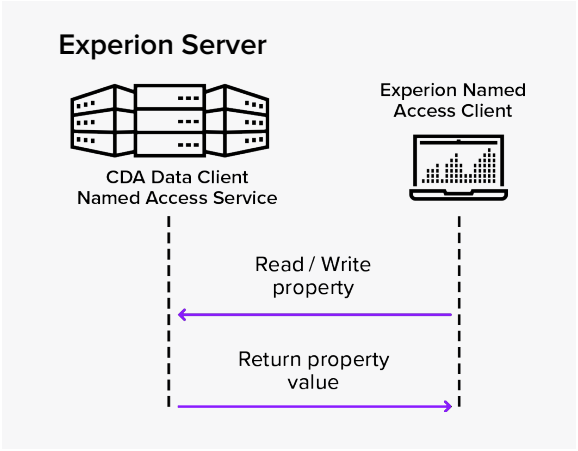

Figure 3. CDA Data Client Named Access Clients communicate with Experion server

In the CDA Data Client Named Access protocol the basic data object is the item. Each data object has a handle describing it. The handle contains the ID of this property, the type and size of the data, the type of component in which this property is stored and the address of that component. For example:

Image Version of C300 controller handle:

- ID - 0x13d
- Type - String
- Size - up to 0x20
- Component type - PCM
- Component address - 0x3e8

When a CDA Data Client Named Access client connects to the server and wants to read or update an item's value, it defines a new group of all the relevant items and reads or updates the entire group together. To specify an item to the server, the client has two options:

- Sending the name of the item and the server will retrieve the item's handle from an internal DB.
- Sending the handle itself with the proper metadata of the item.

Example of a group:

Group id: 0x000A0001

Items:

- Image Version, ID - 0x13d, Type - String
- Initialization Complete, ID - 0x13e, Type - Boolean

Each CDA Data Client Named Access protocol message starts with a header of 40 bytes, describing the command type, group ID and more.

CDA Data Client Named Access Message Structure

Each CDA Data Client Named Access message type has a different structure. For the purpose of this white paper we'll go over the structure of two commands: **WRITE_HANDLES and Get_db_path_for_driver**.

WRITE_HANDLES Command

This method is used to write a property into a device by specifying its handle explicitly, not by its name. This message contains two lists:

- **Handle list** - Each handle describes an item
- **Data element list** - Each data element to write to an item

Data Element Structure

The Data Element descriptor is 28 bytes. In the case of a basic data type like an integer or byte, the data itself is a part of the descriptor. If the data is longer than 8 bytes, after the descriptor there will be another uint specifying the size of the data and then the data itself.

Type	Size	Data (for basic types up to 8 bytes)	Length (for strings and structs)	Data (for strings and structs)

As we can see here, the type of data is specified twice in the message, and its size is specified three times. This kind of redundancy leads to a type confusion in the end.

Get_db_path_for_driver Command

This method generates a connection string from parameters received from the client. The connection string can be used later to add a device to a system.

Network Type	Path Size	Path	Name

Path size - Number of elements in path, each element is 4 bytes long.

Vulnerable Products

An examination performed by Honeywell of the Crit.IX vulnerabilities and compared against their various DCS platforms revealed the following products are affected by the vulnerabilities:

CVE ID	Severity	CVSS	Product	Platform	Versions
CVE-2023-23585 CVE-2023-22435 CVE-2023-24474 CVE-2023-25078	Critical	9.8	Experion Server Experion Station	EPKS	501.1-520.2
			Engineering Station Direct Station	LX	510.5-520.2
CVE-2023-25178	High	7.5	Engineering Station Direct Station	PlantCruise	510.5-520.2
CVE-2023-24480 CVE-2023-25948 CVE-2023-25770	Critical	9.8	C300	EPKS	501.1-520.2
				LX	510.5-520.2
CVE-2023-26597	High	7.5		PlantCruise	510.5-520.2

Discovered Vulnerabilities

Experion® CDA Server Vulnerabilities

Endian Fixing - CVE-2023-24474

The discovered vulnerability is a buffer overflow that allows remote code execution on the Experion® server. CDA protocol allows the flexibility to decide which side will fix the endianness of the message, the client or the server. This option is decided by a flag on the connection request that was sent in the beginning of the session. Honeywell implemented a library that handles the LE to BE conversion.

The “ntoh” library first converts the header of the message and each request header to the correct order, and then fixes the data field depending on the data size and type. To simplify the implementation, for all types for which the conversion is simply a reverse of the bytes (short, int, int64 and double) the library calls a reverse memcpy and uses the size parameter in the request header to know how many bytes we need to reverse. Lack of checking of the size parameter allows an attacker to reverse an array of bytes up to 65535 bytes, and actually let the attacker write up to 0x2000 of data (the maximum size of the CDA message buffer). This leads to remote code execution since the message buffer is part of the static memory which contains both object vtables and global pointers.

CDA Data Client Named Access Vulnerabilities

CDA Data Client Named Access protocol allows Experion applications needing named access to read, write, delete or subscribe to changes of an item. First the client creates a new group identified by a unique group ID. Each group contains an array of items, each of them has specific type, size and a unique identifier.

For each group the user can decide if they want to use a name to identify the item. This will be done by loading the data about this handle from an internal DB. The user can also describe the handle explicitly. This is why for each method (READ, WRITE, etc) there are two options. For example, ReadNames and ReadHandles.

List of CDA Data Client Named Access service methods exposed to the client:

- CANCEL_REQUEST
- READ_NAMES
- READ_HANLES
- WRITE_NAMES
- WRITE_HANDLES
- SUBSCRIBE_NAMES
- SUBSCRIBE_HANDLES
- SUBSCRIBE_ADD_NAMES
- SUBSCRIBE_ADD_HANDLES
- SUBSCRIBE_DELETE
- SUBSCRIBE_TRIGGER
- MESSAGE
- REGISTER_NAME
- DEBUG_CMD
- Get_db_path_for_driver
- Get_driver_list
- Set_Comm_Path_For_Pcm
- Get_Ip_Address
- LOG_INFORMATION

Each message has a header of 40 bytes that contains the size of messages, number of items, group ID, command and more. After the header there is a serialized array of items.

If the client uses names as a descriptor for the item, the item will be 32 bits that describe the size of the name string and a unicode string. If the client uses handles, it will be a header of 24 bytes describing the Item size and type.

On Write request, the message also consists of the data itself. The data also has a header of 28 bytes, describing the data size and type. This redundancy of header for describing the data type and size is unnecessary and can cause type confusion which eventually allows an attacker to get a read primitive in the process memory.

Data Confusion in WRITE_HANDLES - CVE-2023-25178

On **WRITE_HANDLES** request, we first send a list of handles which describe the data type and size. Then, we send the actual data with a header that contains the size and type of the data.

The header of the data is copied as is into an array of objects called _data_element. This object uses its buffer in two ways: if the size of the data is 8 bytes or less, it will be copied into the buffer. Otherwise, the first 4 bytes will be used as a pointer to a new allocated buffer. Because the object is copied as is, an attacker can write any value to the pointer field and set the type to int32_t which is only 4 bytes and doesn't require an external buffer. Later, the program takes the data and sends it to the PCM mentioned in the CDA Data Client Named Access request. Instead of relying on the _data_element header, it relies on the handle the attacker provided earlier for the type of the data. As a result, the attacker can set the type to String and then the program will treat the first 4 bytes of the data as a pointer and send the data it points to the device until it hits a null termination. Listening to the connection between the PCM and the server will allow us to catch that data.

```
char *__cdecl copy_list_of_data_elements_into_Carray(clsPsCommBuf *msg, OPC_data_element *OPC_data_element_array, unsigned int num_of_elem
{
  char *buffer; // eax
  unsigned int i; // [esp+0h] [ebp-10h]
  char *_buffer; // [esp+Ch] [ebp-4h]

  buffer = get_buffer(msg);
  _buffer = buffer;
  for ( i = 0; i < num_of_elemnts; ++i )        // num_of_elemnts not checked against msg bounds
  {
    reset(&OPC_data_element_array[i]);
    memcpy(&OPC_data_element_array[i], &_buffer[*offset], sizeof(OPC_data_element));// the data is copied into the array
                                                // straight from the msg buffer without any checks.
    *offset += 0x1C;
    *offset += copy_len_of_bytes(&OPC_data_element_array[i], &_buffer[*offset]);//
                                                // copy bytes eve if there is null inside it
                                                // len is the first 4 bytes, and not checked.
    buffer = (char *)(i + 1);
  }
  return buffer;
}
```

Figure 4. Copying the _data_element as it is from the request buffer to the _Data_Element array.

```
UINT __thiscall copy_len_of_bytes(OPC_data_element *this, void *Src)
{
  OPC_data_element *v3; // [esp+4h] [ebp-B4h]
  WCHAR v4[82]; // [esp+8h] [ebp-B0h]
  DWORD v5; // [esp+B0h] [ebp-8h]
  UINT len; // [esp+B4h] [ebp-4h]

  v3 = this;
  switch ( this->type )
  {
    case 2:
    case 6:
    case 7:
    case 8:
    case 9:
    case 0x14:
    case 0x15:
    case 0x1A:
    case 0x1D:
    case 0x1E:
    case 0x1F:
    case 0x22:
    case 0x23:
    case 0x24:
      memcpy(&len, Src, sizeof(len));           // read primitive because buffer is controlable
      v3->buffer = SysAllocStringByteLen((LPCSTR)Src + 4, len);
      if ( !v3->buffer )
      {
        reset(v3);
        v5 = GetLastError();
        wsprintfW(v4, L"Error return code from GlobalAlloc = %li.", v5);
      }
      len += 4;
      break;
    default:
      len = 0;
      break;
  }
  return len;
}
```

Figure 5. Use type 1 so this->buffer will not be overwritten.

```
v6 = (OPCItem *)get_ptr_data((void **)&v38);
item_handle_type = get_item_handle(v6)->type;// use item handle type instead of OPC_data_element type
v7 = (OPCItem *)get_ptr_data((void **)&v38);
v8 = get_item_handle(v7);
LOWORD(data_element_size) = get_item_size_from_item_type(&v35[v37 - 1], v8);
data_element = alloc_data_element(data_element_size, item_handle_type);// by default alloc_data_element
                                        // allocate 0x208 bytes for its buffer
if ( data_element )
{
  v9 = (OPCItem *)get_ptr_data((void **)&v38);
  item_handle = get_item_handle(v9);
  if ( !copy_from_OPC_to_data_element(&v35[v37 - 1], data_element, item_handle) )
  {
    if ( data_element::get_field_4(data_element) == 0xDB90 )
      sloggerclient 4(
```

Figure 6. Data element is allocated according to the item_handle and not to _data_element.

```
  case 2:
  case 0x22:
  case 0x23:
    v6 = 2 * wcslen((const wchar_t *)this->buffer) + 2;
    if ( v6 <= get_avail_size(data_element) )
    {
      v7 = wcslen((const wchar_t *)this->buffer);
      copy_wstring_into_data_element(data_element, (unsigned __int16 *)this->buffer, 2 * v7);
    }
    else
    {
      data_element::set_field_4(data_element, 0xDB90);
      v24 = 0;
    }
    break;
```

Figure 7. If type is 2 (String) the program will copy the data from this->buffer which we control.

Get_db_path_for_driver Stack Overflow - CVE-2023-22435

This method is designed to receive a few parameters and use it to build the DB path string. One of the parameters is the PathParm which is an array of integers. The client sends the number of integers in the PathParm and then the list of integers. The server allocates an array of integers on the stack in size of 0x410 bytes which is 0x104 integers. When the server copies the integer into the array, it checks that the size of the array is not more than 0x104 integers but since it is a signed check, if the client sends a negative integer, it will pass the check. This only allows the client to corrupt the stack with a large number of bytes. However, since the client sends the number of integers, not the number of bytes, the server needs to multiply the number by 4 or in other words - shift it two bits to the left. However, the check is done on the original number, not on the result of the multiplication. This is a classic “time of check time of use” scenario and it allows an attacker to send a number that only its MSB bit is on, that number will be considered as a negative number at the time of the check, but after shift by 2 it will become a positive number because of integer overflow. For example, 0x80000100 is equal to 0x00000400 after multiplication by 4. This allows the attacker to overflow the stack by any number of bytes and fill it with any type of data.

```
int __cdecl FormPathString_parse_input(int a1, int *network_type, int a3, int *path_size, char *path, int *a6, int *offset)
{
  void *v7; // eax
  int result; // eax
  size_t Size; // [esp+4h] [ebp-Ch]
  int v10; // [esp+8h] [ebp-8h]
  char *v11; // [esp+Ch] [ebp-4h]

  v11 = get_buffer((clsPsCommBuf *)a1);
  *network_type = *(_DWORD *)&v11[*offset];
  *offset += 4;
  sloggerclient_4(215, 40, off_4DCEE8[0], "497", "Network type %d ", *network_type);
  *path_size = *(_DWORD *)&v11[*offset];
  *offset += 4;
  Size = 4 * *path_size;                        // path_size is checked instead of size
  sloggerclient_4(215, 40, off_4DCEE8[0], "504", " Path size  %d", *path_size);
  if ( *path_size > 0x104 )                     // signed check
                                                // can overflow the integer
                                                //
  {
    sloggerclient_4(215, 10, off_4DCEE8[0], "515", "Invalid input: Path size is bigger than expected %d ", *path_size);
    sloggerclient_5(215, 10, off_4DCEE8[0], "516", v11, Size, "Print the PathParm buffer");
  }
  else
  {
    memcpy(path, &v11[*offset], Size);          // can put any number aligned to 4 and overflow the stack
                                                // path is allocated on stack with size of 0x40c
    sloggerclient_5(215, 40, off_4DCEE8[0], "511", path, Size, "Print the PathParm buffer");
  }
  *offset += Size;
  if ( *path_size > 0x104 )
    *path_size = 0;
  *a6 = *( DWORD *)&v11[*offset];
```

Figure 8. Parsing of Get_db_path_for_driver command.

Set_Comm_Path_For_Pcm Buffer Overflows - CVE-2023-23585/25078

In this method the client is sending a string that represents many values about the PCM. The string has a key-value format as follows: “<Key>=<Value>:<Key_2>=<value_2>...”. The server parses the string and copies each value to a specific buffer. Since the client sends a Unicode buffer, the server has implemented a function called “UnicodeToMultiByte” to change the encoding of the buffer. This function takes two arguments, the Unicode buffer pointer and an out pointer for the multiByte string. To ensure that the out buffer is big enough for the multiByte string, the function calls IsBadWritePtr and sends as an argument the size of the multiByte string we need to copy. Now except for the fact that this function is obsolete and should not be used, it also only checks if the memory region pointed by the first argument has a write privilege. If we send a pointer to the stack for example, since the entire stack is writable for the program, the function will return true up to the size of the whole stack.

```
int __cdecl convert_from_utf16_to_multibyte(const char **a1, LPVOID lp)
{
  const WCHAR *v2; // eax
  const WCHAR *v3; // eax
  int ucb; // [esp+0h] [ebp-8h]
  unsigned __int8 v6; // [esp+7h] [ebp-1h]

  v6 = 0;
  v2 = (const WCHAR *)ATL::CSimpleStringT<wchar_t,1>::operator wchar_t const *(a1);
  ucb = WideCharToMultiByte(0, 0, v2, -1, 0, 0, 0, 0);
  if ( !lp || IsBadWritePtr(lp, ucb) )          // IsBadWritePtr not really check the size of lp, but only if the ptr has write.
                                                // if ptr is allocated on the stack, that means the all stack, which lead to a stack overflow
    return v6;
  v3 = (const WCHAR *)ATL::CSimpleStringT<wchar_t,1>::operator wchar_t const *(a1);
  WideCharToMultiByte(0, 0, v3, -1, (LPSTR)lp, ucb, 0, 0);
  v6 = 1;
  return v6;
}
```

Figure 9. The function converts from Unicode to multiByte using the obsolete function IsBadWritePtr.

From this point forward, the function “UnicodeToMultiByte” is unsafe and every and every place it has been used, is potentially exploitable by an attacker to get an RCE. set_Comm_Path_For_Pcm uses this function six times - three times on a buffer allocated on the stack and three times on the heap. That gives an attacker three opportunities for stack overflow and three opportunities for heap overflow.

Experion® C300 Controller Vulnerabilities

Endian Fixing - CVE-2023-25770

Since Honeywell uses the “ntoh” library in the C300 controller as well, the same vulnerability that has been found on the server also applies to the controller.

CDA Protocol - CVE-2023-24480/26597

When the controller parses the CDA protocol messages from the server, if it discovers an error for some reason - could be a boundary check, invalid size etc. the controller calls SendErrorResponse with an error code parameter. This function iterates on all the message parts, and builds an error response with the same number of requests and sets the error code on each of them. In the process, this function copies the request data into a buffer on the stack. The problem is that this message is called also if the size fields in the message are not valid. That means that the already known problematic input is used again when sending the error response. This allows an attacker to overflow the stack in two different scenarios and eventually lead to a RCE.

In addition, this function doesn't check how many parts are written into the response buffer, and since a response to every request is 8 bytes, sending enough requests will easily overflow the response buffer and crash the system.

Unsigned Firmware Design Flaw - CVE-2023-25948

The firmware of the controller is unsigned, meaning that an attacker can make a counterfeit firmware and install it over the network with no user interaction.

Mitigation

Honeywell has made available the following security patches and strongly advises all affected organizations to promptly apply them.

Honeywell customers can access and apply patches by logging into https://process.honeywell.com/ and searching through the Technical Publications section. For more information regarding Honeywell's Coordinated Vulnerability Disclosure Process visit https://www.honeywell.com/us/en/product-security.

It is essential to prioritize the implementation of these patches and firmware updates to address reported vulnerabilities effectively. Furthermore, organizations should persist in their commitment to robust security measures, including routine security assessments, penetration testing, and comprehensive security training for their development teams.

For Experion PKS, the earlier hardware versions of the C300 (C300 PCNT01, C300 PCNT02) do not have the hardware capabilities to support secure boot or IPSec in the device. There is a C300 security lock feature that can be used to require physical access for certain functions, including loading the firmware. Experion PKS has a newer hardware version of the C300 (C300 PCNT05) which supports secure boot and IPSec communications.

How Can Armis Help

The development and deployment of patches to resolve vulnerabilities present in controllers and engineering workstations in OT environments is essential to reduce the attack surface. Due to the business criticality and their impact in operational processes, the release and installation of patches for these assets requires a very thorough QA process and most likely a shutdown window, which can take a long period of time to coordinate and ultimately to complete. It is reasonable to assume that affected assets will remain vulnerable for a long period of time. During this time, mitigations can be implemented for detecting and preventing attacks on these critical infrastructure assets.

Armis customers can leverage the Asset Intelligence and Security Platform for protecting their network in the following ways:

1. **Achieve comprehensive Asset Visibility.** By obtaining an accurate inventory that encompasses every aspect, from hardware to firmware and software version, organizations can effectively identify vulnerable servers and controllers in their environment.
2. **By implementing a Vulnerability Management program that prioritizes according to risk**, organizations can effectively minimize their weak points and reduce the risk of exploits targeting devices without

available patches. Moreover, promptly applying security patches upon their release will significantly decrease the window of vulnerability for these devices.

3. **Since the discovered vulnerabilities require only network access to a vulnerable device, Network Segmentation will go a long way in preventing exploitation of these vulnerabilities.** By separating the network into distinct segments based on security levels or device types, organizations can limit the lateral movement of attackers, effectively containing potential threats and mitigating the impact on vulnerable devices. The segmentation effort in OT environments can be achieved using an industry reference model such as the Purdue Model, which represents a logical or functional view of OT environments and can be used to identify any deviations from OT assets expected behaviors, specially assets communicating to Level 0 and Level 1 from higher-level assets including assets in the IT networks. The segmentation can be achieved by understanding these asset behaviors in order to whitelist only the expected ones.
4. **Experience has shown that even well protected networks are susceptible to breaches. Thus, it becomes imperative to implement a robust Threat Detection system** capable of identifying exploit attempts spanning the entire network and encompassing all devices, including IT, OT, and IoT. Employing a blend of detection techniques, including signature-based analysis, anomaly detection, and indicators of compromise (IOCs), adds an extra layer of security, augmenting the overall defensive posture in the event of an attack.

Final Notes

Over the past few years, there has been a notable increase in the frequency and impact of cyber attacks targeting critical infrastructure. The attractiveness of these attacks stems from inadequately secured legacy systems, coupled with their potential to cause significant damage and financial losses. Furthermore, as these previously isolated networks become more interconnected, their attack surface expands. As a result, vulnerabilities such as Crit.IX are lucrative to both cyberwarfare and cybercrime actors.

Time and time again we see how legacy technologies, that weren't developed with security in mind, give rise to vulnerabilities as they adjust to modern times. In this example, protocols and code libraries that were made for serial connections, without encryption or authentication, are repurposed in a modern network environment and result in critical vulnerabilities with high impact and minimal prerequisites from an attacker.

Finally, Comprehensive vulnerability disclosures necessitate active cooperation from the vendors of the affected equipment. In this particular disclosure, we engaged in a collaborative effort with Honeywell to ensure a thorough understanding of the vulnerabilities, the scope of impacted devices, and effective mitigation strategies. This case study exemplifies the vital collaboration between security researchers and vendors, highlighting the significance of responsible disclosures in fostering a more secure digital ecosystem.

About Armis

Armis, the leading asset visibility and security company, provides the industry's first unified asset intelligence platform designed to address the new extended attack surface that connected assets create. Fortune 100 companies trust our real-time and continuous protection to see with full context all managed, unmanaged assets across IT, cloud, IoT devices, medical devices (IoMT), operational technology (OT), industrial control systems (ICS), and 5G. Armis provides passive cyber asset management, risk management, and automated enforcement. Armis is a privately held company and headquartered in California.

armis.com

info@armis.com

20221107-1