ICS Village CIP CTF

Table of Contents

CIP CTF

This section of the DC33 ICS Village CTF is to provide a guided intro into Common Industrial Protocol (CIP) and basic ways to exploit it.

Intro to CIP

CIP is a protocol used by many industrial devices. Everything from Programmable Logic Controllers (PLC) to Variable Frequency Drives (VFD) use CIP as a means of communication.

CIP itself is simply a universal protocol for transporting data and commands. It requires a network to transport the protocol. Several different network types exist, the most common are:

- DeviceNet: CIP over CAN BUS
- ControlNet: CIP over coax / fiber
- Ethernet/IP: CIP over Ethernet

Each network that transports CIP is designed for inter-compatability. This means that the same CIP message used on one network is also used on another network.

CIP Object Model

CIP is a protocol similar to a Remote Procedure Call (RPC) protocol. Every CIP packet is a command to run a Service within an Object.

Every CIP object has a Class which is a single static instance of the object, along with a variable number of Instances. Every class is defined by a unique Class ID, and every instance is defined by a unique Instance ID. The static instance of the class is always Instance ID 0.

CIP Objects also have Attributes. Each attribute is a variable which stores data. Some attributes are read-only, others have special properties such as persisting the value within them to EEPROM. Each Attribute is identified with an Attribute ID.

Finally, all CIP Objects have Services. Services are functions that can be called on the

object, or on individual attributes. Every service has a unique Service ID.

Class

> Every CIP object is a instance of a class. Each class is a C++ object within the code of the device. The CIP class defines the structure and functionality of the instances of that class.

Instance

> An Instance is a single instance of a class. Each instance has a unique Instance ID. The static instance of the class is always Instance ID 0. The static instance usually contains information about how many dynamic instances exist and has a completely unique structure compared to the dynamic instances.

Attribute

> An Attribute is a variable which stores data. Each attribute has a unique Attribute ID.

Service

> A Service is a function that can be called on the Class, Instance, or on individual Attributes. Every service has a unique Service ID.

An example of a CIP object is Class 0x01, the Identity Object (Identity Object (0x01)). It contains basic information about the device.

Common CIP Services

Many CIP objects utilize the same services. While objects do not need to implement services the same between each other, some service IDs are specified by the specification for how they work.

Here are examples of common services, which allow reading and writing data from attributes. Each service includes a usage example, shown in a pseudocode format representing a CIP message structure (e.g., as sent over EtherNet/IP). These examples assume you're targeting the Identity Object (Class 0x01, Instance 1).

Get_Attribute_Single (Service ID: 0x0E): Retrieves the value of a single specified attribute from an object instance or class. **Usage Example**: To read the Vendor ID (Attribute 1) from the Identity Object.

```
CIP Message:
- Service: 0x0E (Get_Attribute_Single)
- Path: Class 0x01, Instance 1, Attribute 1 (encoded as logical
segments: 0x20 0x01 0x24 0x01 0x30 0x01)
- Data: None (request has no additional data)

Response: Returns the Vendor ID value (e.g., 0x0001 for Rockwell
Automation).
```

Get_Attribute_All (Service ID: 0x01): Retrieves the values of all attributes from an object instance or class in one request. This is efficient for getting an overview.

Usage Example: To fetch all attributes from the Identity Object.

```
CIP Message:
- Service: 0x01 (Get_Attribute_All)
- Path: Class 0x01, Instance 1 (encoded as: 0x20 0x01 0x24 0x01)
- Data: None

Response: A sequence of all attribute values (e.g., Vendor ID, Device
Type, Product Code, etc., in order).
```

Get_Attribute_List (Service ID: 0x03): Retrieves the values of a list of specified attributes from an object. You provide a list of Attribute IDs.

Usage Example: To read Vendor ID (1) and Product Name (7) from the Identity Object.

```
CIP Message:
- Service: 0x03 (Get_Attribute_List)
- Path: Class 0x01, Instance 1 (encoded as: 0x20 0x01 0x24 0x01)
- Data: Number of attributes (2), followed by IDs (0x0001, 0x0007)

Response: List of attribute values (e.g., Vendor ID value, then Product
Name string).
```

Set_Attribute_Single (Service ID: 0x10): Sets (writes) the value of a single specified attribute, if it's writable.

Usage Example: To set a writable attribute like a custom parameter in a vendor-specific object (e.g., Attribute 10 in a hypothetical Parameter Object, Class 0x0F, Instance 1).

```
CIP Message:
- Service: 0x10 (Set_Attribute_Single)
- Path: Class 0x0F, Instance 1, Attribute 10 (encoded as: 0x20 0x0F 0x24
0x01 0x30 0x0A)
- Data: New value (e.g., 0x005A for integer 90)

Response: Success code (0x00) if set, or error if not writable.
```

Set_Attribute_List (Service ID: 0x04): Sets the values of a list of specified attributes in one request. Similar to Get_Attribute_List but for writing multiple values atomically.

Usage Example: To set two writable attributes (e.g., Attributes 10 and 11 in Parameter Object, Class 0x0F, Instance 1).

```
CIP Message:
- Service: 0x04 (Set_Attribute_List)
- Path: Class 0x0F, Instance 1 (encoded as: 0x20 0x0F 0x24 0x01)
- Data: Number of attributes (2), ID 10, value (0x000A, 0x005A), ID 11,
value (0x000B, 0x0014)

Response: Success for each, or partial errors.
```

These services are part of explicit messaging and form the basis for interacting with CIP devices, like querying status or configuring settings.

CIP Routine

CIP uses a very powerful and complex routing system.

This allows CIP messages to traverse multiple networks and devices seamlessly, as long as the networks are CIP-compatible (e.g., from EtherNet/IP to DeviceNet via a linking device). Routing ensures that commands reach the intended object, even if it's on a

different subnet or physical medium. The system handles message forwarding through routers or gateways, maintaining the integrity of the data across hops.

CIP Paths

CIP paths and routes specify how to address and reach a target object within a device or across networks.

A CIP path is a sequence of segments that describe the route from the source to the destination object. Paths are used in every CIP message to identify the target Class, Instance, and Attribute.

- **Path Segments**: Paths are built from segments like:
- **Logical Segments**: Address Class ID (e.g., 0x01 for Identity), Instance ID (e.g., 1), and Attribute ID (e.g., 1 for Vendor ID). These are the most common for local objects.
- **Port Segments**: Used for routing to other devices, specifying a port number (e.g., 1 for backplane) and link address (e.g., slot 0 for CPU). For example, a path "1,0" routes from an EtherNet/IP module (port 1) to the processor in slot 0.
- **Network Segments**: For crossing networks, including node addresses or IP addresses.

Paths can be extended for multi-hop routing. Paths are encoded in a compact binary format (e.g., 0x20 for Class, followed by ID). This enables interoperability across CIP networks without changing the core message.

Simple Path Example

Path: `1,5` Explanation:

```
1 = Backplane
5 = Slot 5 in Backplane
```

The device originating the message sends the message out port 1, which is the Backplane of the PLC. It is then routed by the backplane to slot 5.

Complex Path Example: Multi-Hop Routing

Path: `1,2,2,192.168.1.2,1,5` Explanation:

```
1 = Backplane
2 = Slot 2 (Slot of network card)
2 = Port 2 (Ethernet Port)
192.168.1.2 = IP Address on network to next network card
1 = Backplane
5 = Slot 5
```

The device originating the message sends the message out port 1, which is the Backplane of the PLC. It is then routed by the backplane to slot 2 which has an Ethernet card in the slot. Then from the Ethernet card, the message is routed out port 2, which is the ethernet port. Then on the network, the message is sent to address 192.168.1.2, which is the address of another ethernet card. From that ethernet card, the message is routed out port 1 to the backplane, then to card in slot 5

Another Complex Path Example: ControlNet Hop

Path: `1,2,2,3,1,0` Explanation:

```
1 = Backplane
2 = Slot 2 (Slot of ControlNet card)
2 = Port 2 (ControlNet Port)
3 = ControlNet Address on network to next ControlNet card
1 = Backplane
0 = Slot 0
```

The device originating the message sends the message out port 1, which is the Backplane of the PLC. It is then routed by the backplane to slot 2 which has an ControlNet card in the slot. Then from the ControlNet card, the message is routed out port 2, which is the ControlNet port. Then on the network, the message is sent to address 3, which is the address of another ControlNet card. From that ControlNet card, the message is routed out port 1 to the backplane, then to card in slot 0

CIP Connections

CIP supports different connection types for communication, balancing reliability, efficiency, and real-time needs. Connections manage how messages are sent and resources are allocated.

- **UCMM (Unconnected Message Manager)**: Handles unconnected explicit messages. No prior connection is established; each message is self-contained. Used for infrequent, on-demand requests like configuration or diagnostics. Managed by the device's UCMM, which processes these without reserving resources.
- **Unconnected**: Refers to explicit messaging without a dedicated connection (via UCMM). Messages are sent as needed, ideal for low-priority, non-real-time tasks. No ongoing session, so each request/response pair stands alone.
- **Connected**: Reserves resources for reliable communication. Divided into:
  - **Explicit Messaging (Class 3)**: Client/server style for request/response, like reading/writing attributes. Establishes a connection first (using Forward_Open service), ensuring delivery but with overhead.
  - **Implicit Messaging (Class 0/1)**: For real-time I/O data exchange, like cyclic sensor data. No headers per message; data is sent at fixed intervals. Class 0 is unacknowledged (faster), Class 1 is acknowledged.

Explicit connections are for information (e.g., HMI to PLC), while implicit are for control (e.g., I/O scanning). Connections are opened, used, and closed to manage bandwidth.

Tools

There are several key tools that are used by Engineers to manage CIP networks and devices.

A common tool to use is Rockwell's RSLinx or FactoryTalk Linx. A free version can be downloaded on their website (Requires creating a free account): RSLinx Free Download (https://compatibility.rockwellautomation.com/Pages/MultiProductFindDownloads.aspx?crumb=112&mode=3&refSoft=1&versions=59075)

Another common tool is the Molex EIP tool. This is a free tool that can be downloaded from Molex (No account required): Molex EIP Tool v2.6.1 (https://tools.molex.com/webdocs/mysst/EIP%20Tool%20v2.6.1.zip)

When using scripting to access CIP devices, Python is a common tool. See the ICS Village github for a python cip tool: ICS Village CTF GitHub (https://github.com/ICSVillage/CTF)

These tools are also provided as downloads via the CTF.

Identity Object (0x01)

The CIP Device Identity Object

Overview

- **Class Code**: `0x01`
- **Purpose**: Provides device identification and general status information.
- **Instances**: Typically, a device has one instance (Instance 1) of the Identity Object.
- **Access**: Attributes are generally read-only, except for specific services like Reset.

Attributes

The Identity Object includes several attributes that describe the device's identity and status. The following are the common attributes defined in the CIP specification:

Attribute ID	Name	Data Type	Description	Access
1	Vendor ID	UINT	Unique identifier of the device vendor (assigned by ODVA).	Read
2	Device Type	UINT	Indicates the general type of device (e.g., PLC, sensor, drive).	Read
3	Product Code	UINT	Vendor-specific code identifying the product.	Read
4	Revision	STRUCT of USINT	Major and minor firmware/hardware revision (e.g., Major.Minor).	Read
5	Status	WORD	Current status of the device (e.g., owned, configured, faulted).	Read
6	Serial Number	UDINT	Unique serial number for the device.	Read
7	Product Name	SHORT_STRING	Human-readable name or description of the device.	Read

Notes on Attributes

- **Vendor ID**: Assigned by ODVA (Open DeviceNet Vendor Association) to uniquely identify the manufacturer.
- **Device Type**: Maps to a predefined list of device profiles (e.g., 0x02 for AC drives, 0x0C for communications adapter).
- **Revision**: Split into Major and Minor fields to indicate significant or minor changes in firmware/hardware.

- **Status**: A bit field indicating various states (e.g., bit 0: Owned, bit 4-7: Device state like running or faulted).
- **Serial Number**: Must be unique for each device within a vendor's product line.
- **Product Name**: A string limited to 32 characters in most implementations.

Services

The Identity Object supports several services to interact with its attributes or control the device. Common services include:

Service Code	Name	Description
0x01	Get_Attributes_All	Retrieves all attributes of the Identity Object in a single response.
0x05	Reset	Initiates a device reset (e.g., power cycle, factory reset).
0x0E	Get_Attribute_Single	Retrieves the value of a single specified attribute.
0x10	Set_Attribute_Single	Sets the value of a single specified attribute (if writable).

Reset Service Types

The Reset service (0x05) supports different reset types, typically specified in the service data:

- **Type 0**: Power cycle (simulates power off/on).
- **Type 1**: Factory reset (returns to out-of-box settings).
- **Additional types**: May be vendor-specific (e.g., reboot to bootloader).

Instance and Class Details

- **Class Attributes**: Typically include revision and instance count information.
- **Instance Attributes**: Instance 1 contains the device-specific identity data (as listed above).
- **Mandatory Attributes**: Attributes 1–7 are required for compliance with CIP standards.
- **Optional Attributes**: Vendors may add custom attributes (e.g., Attribute 8 and above) for additional device-specific information.

Usage

- The Identity Object is used during device discovery and configuration in CIP networks.
- It allows network tools or controllers to identify devices, check their status, and perform basic operations like resets.
- Commonly accessed during network commissioning or diagnostics.