

# Intro to Common Industrial Protocol

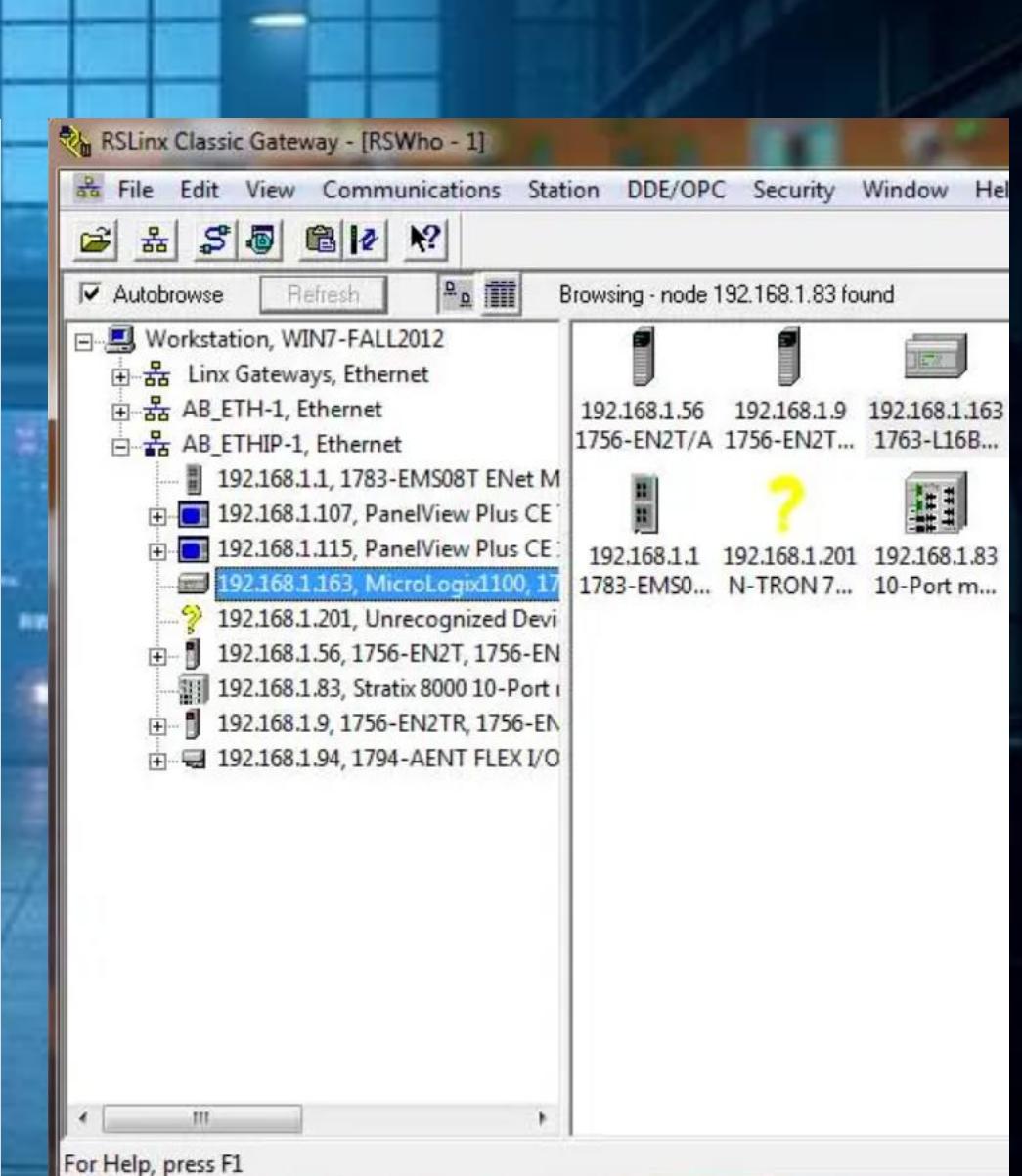
& Exploits

The background of the slide is a photograph of a dimly lit industrial control room. On the left, there's a row of computer workstations with multiple monitors showing various data and graphs. The room is filled with complex industrial equipment, including large tanks, pipes, and control panels. The lighting is low, creating a mysterious and technical atmosphere.

DEF CON 33  
ICS Village

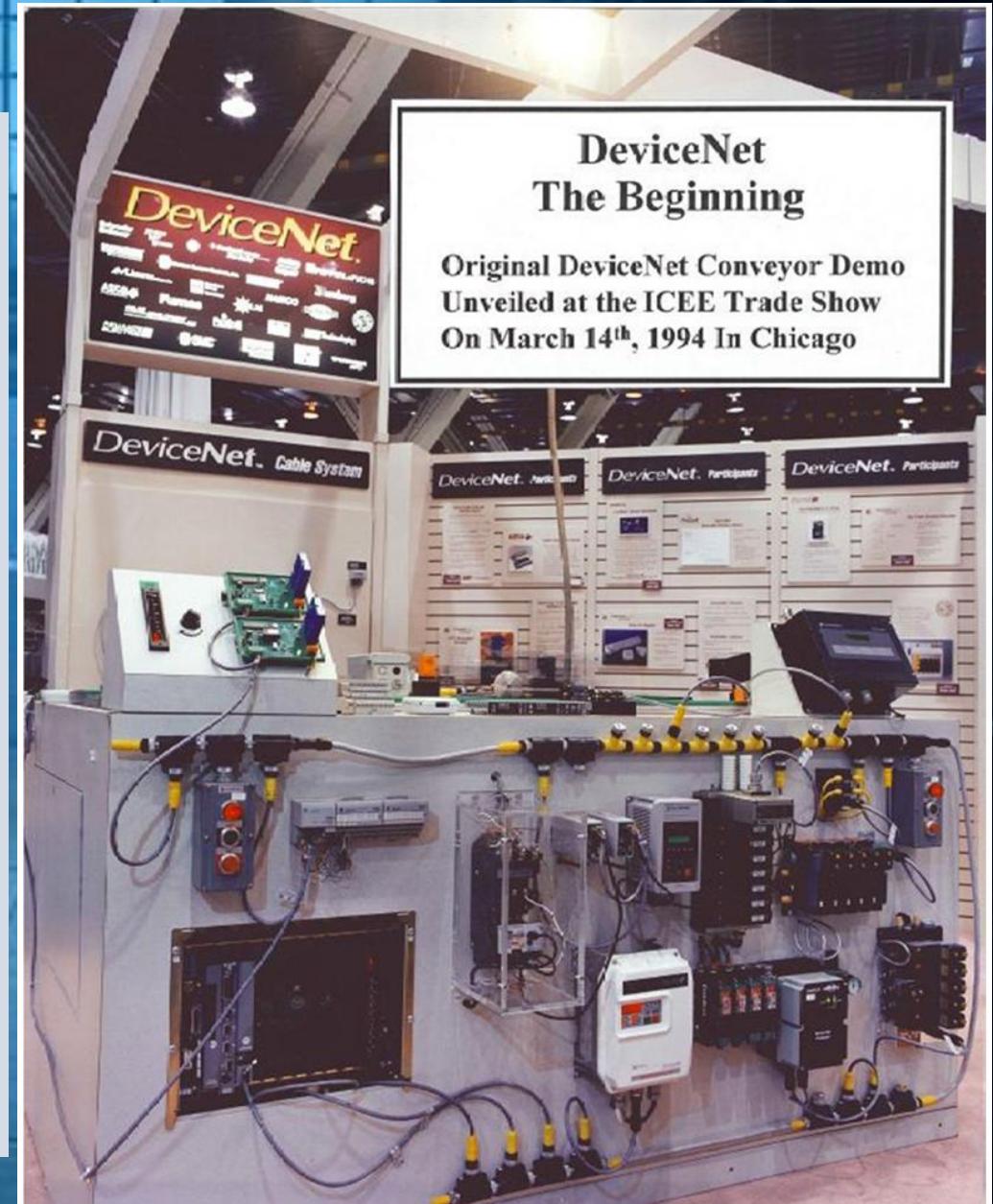
# What is CIP?

- Used in Industrial Control Systems for devices on the plant floor
  - PLC (Programmable Logic Controllers)
  - VDF (Variable Frequency Drives, AKA: Motor Controllers)
  - Motor Starters
  - Power Monitors
  - Smart Sensors / Instruments
  - Valve Actuators
- Multiple network types
  - DeviceNet
  - ControlNet
  - EthernetIP
  - CompoNet



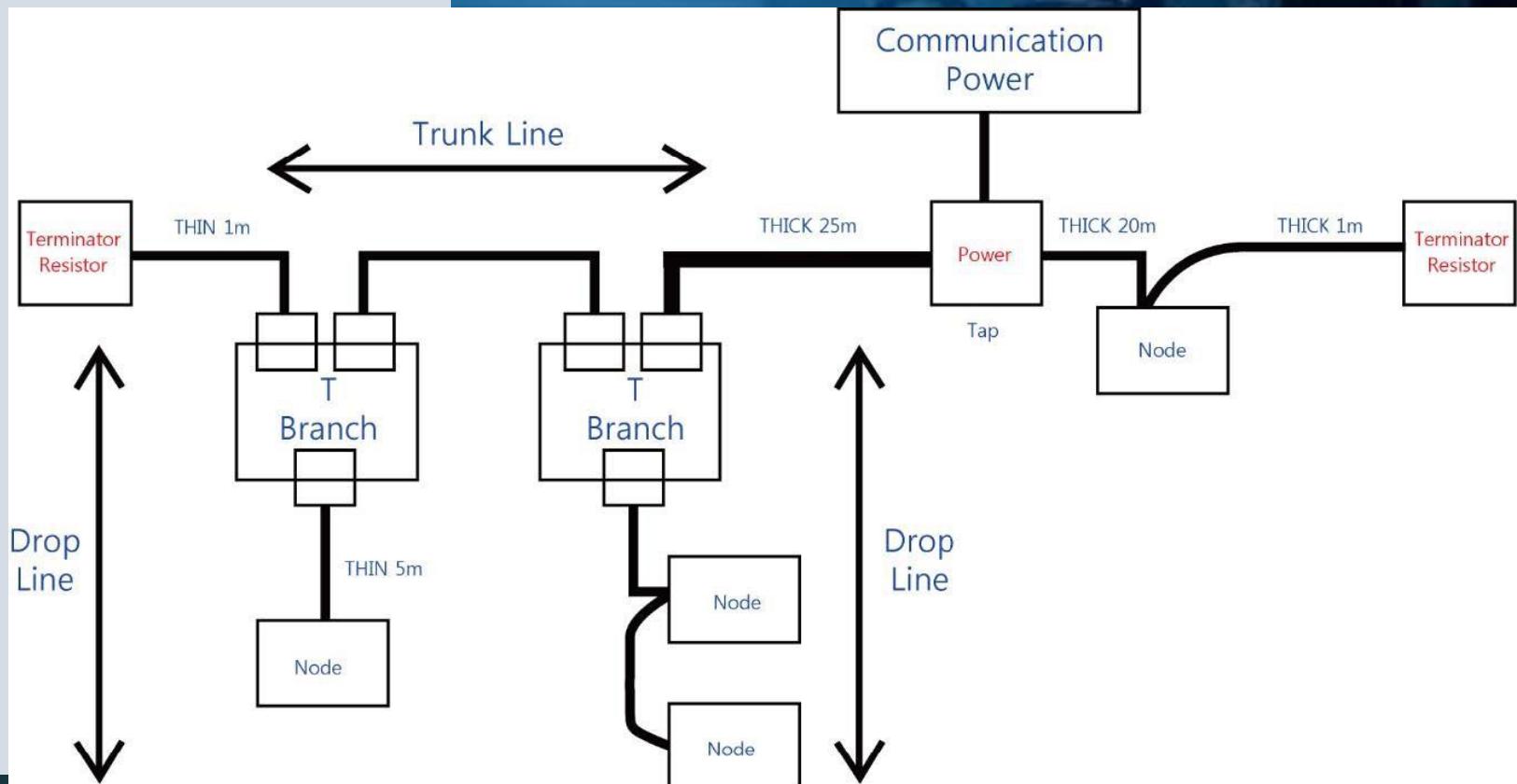
# History

- Developed by Allen-Bradley
- Managed by ODVA (Open DeviceNet Association)
- DeviceNet, 1994
  - Uses CAN (Controller Area Network) for layer 1 and 2
- ControlNet, 1997
  - Uses 802.3b for layer 1 specification (75 ohm coax)
- Ethernet Industrial Protocol (Ethernet/IP), 2000
  - Transport CIP over Ethernet using UDP and TCP combo
- CompoNet, 2007??? No one cares...



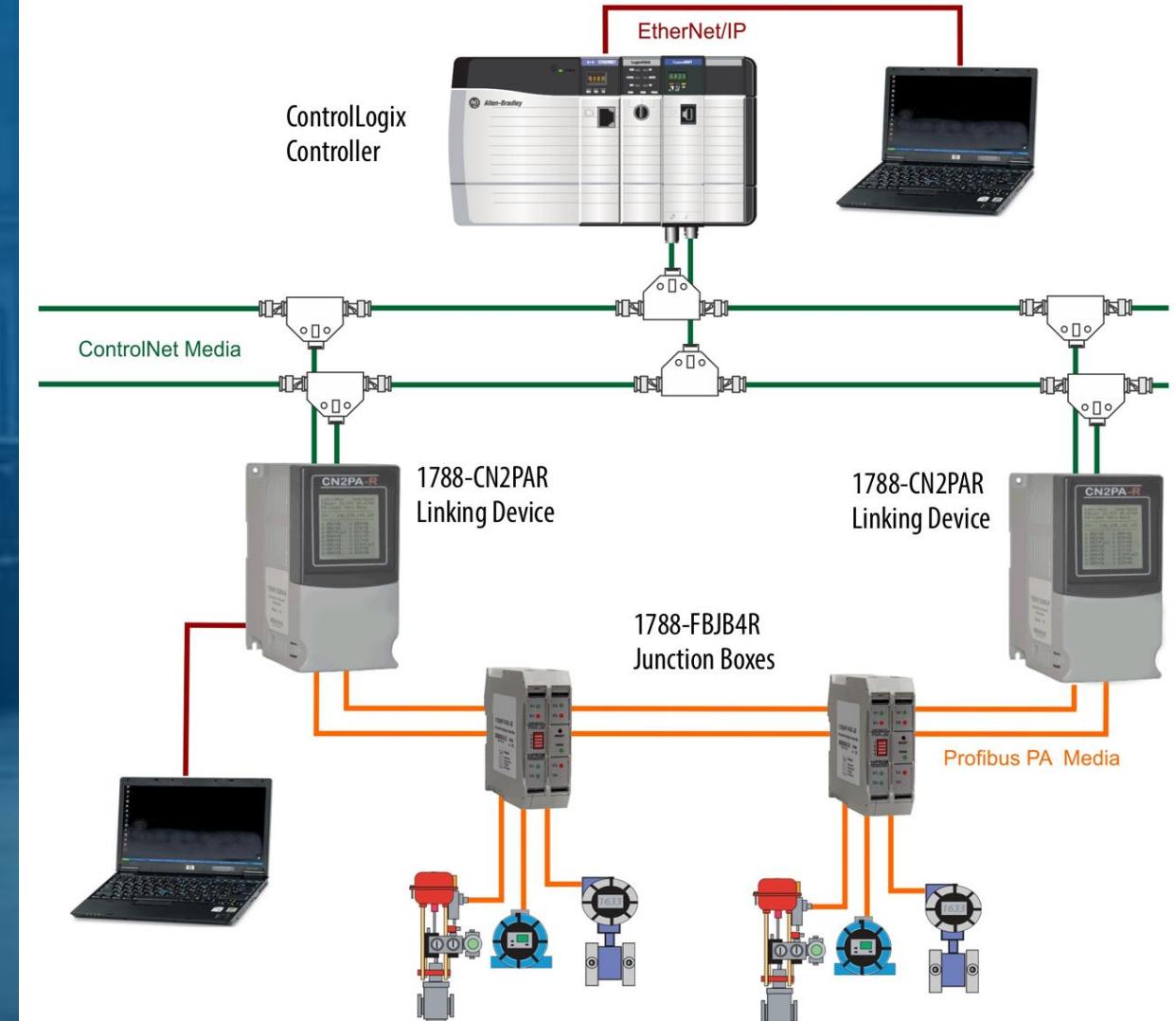
# DeviceNet

- Uses CAN (Controller Area Network) for layer 1 and 2
- 4 wire interface providing 24V power with the CAN signal
- Max 64 devices
- Up to 500 Kbs data rate
- Highly flexible installation
- Non-deterministic
- No redundancy



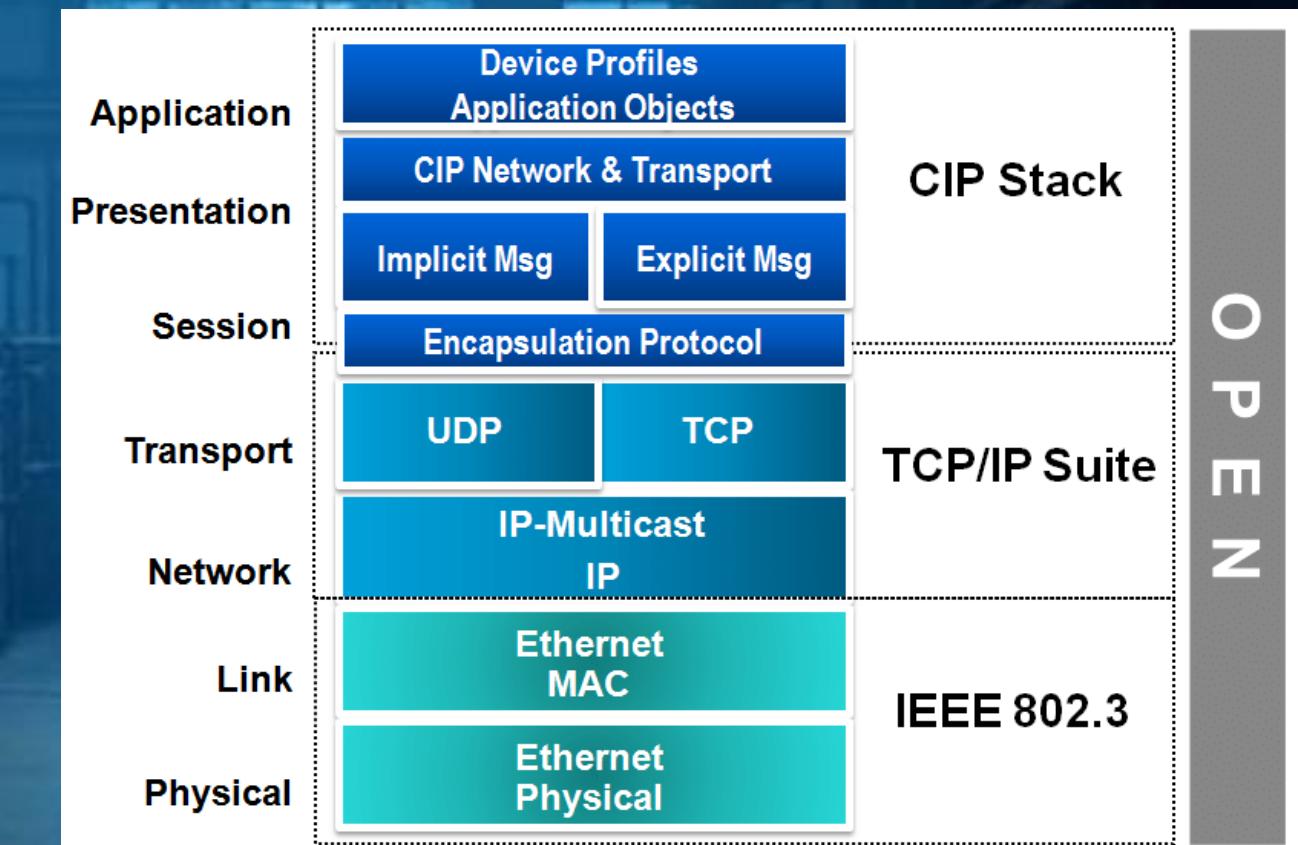
# ControlNet

- Uses 802.3b for layer 1 specification
- 75 Ohm coax with BNC connectors
- Allows for redundant coax connections
- Option to use converters for fiber optic cable runs
- Max 99 devices
- Fixed 5 Mbs data rate
- Very ridged installation
- Deterministic and Scheduled network



# Ethernet/IP

- Transport CIP over Ethernet using UDP and TCP
- Utilizes Unicast and Multicast traffic
- Supports custom ring topology
  - DLR (Device Level Ring)
  - PRP (Parallel Redundancy Protocol)
- Uses custom transport protocol on top of TCP/UDP



# Objects

- CIP is like an RPC type protocol
- Class = C++ Classes
- Instance = C++ Object (instance of class)
- Attribute = Variable in object
- Service = Function in object
- All CIP commands are a Service directed to a Class, Instance, and/or Attribute. (Other more complex messages exist)



# Objects Example

- Identity Class (0x01) contains basic device info
  - Has a default Instance ID of 0x01
  - Has a Get All Attributes service (0x01)
  - WHO Message:
    - Service 0x01 (Get All Attributes)
    - Class 0x01 (Identity Class)
    - Instance 0x01 (Default Instance)
    - Attribute 0 (No attribute specified)
  - This is a message that ALL CIP compliant devices support
- \* Except DeviceNet which uses a different service...

(sigh)

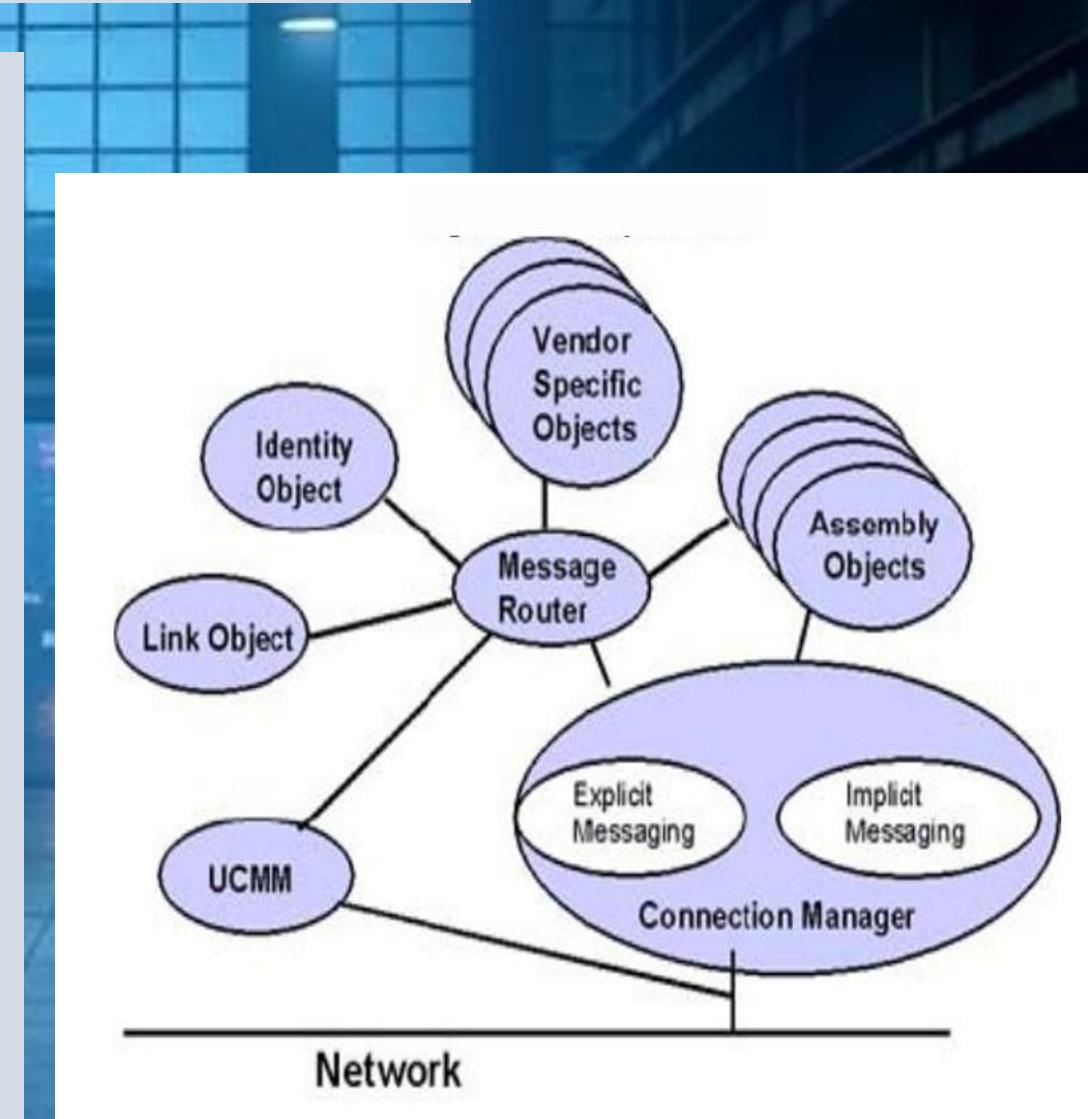
```
# Get basic info from controller
def cipGetInfo():
    with CIPDriver(address) as plc:
        result = plc.generic_message(
            service=0x01,           # Get All Attributes
            class_code=0x01,         # Device Identity Class
            instance=0x01,          # Default Instance
            attribute=0x00,          # No Attribute
            connected=False,         # Use Unconnected Send instead of UCMM
            unconnected_send=True,   # Use Unconnected Send instead of UCMM
            route_path=True          # Use Unconnected Send instead of UCMM
        )
        printBytes(result.value)

Result: 35 bytes
  0  1  2  3  4  5  6  7  8  9
  +-----+
0000|01 00 0e 00 36 00 14 3a 60 30|
0010|9e 8b 73 00 14 31 37 35 36 2d|
0020|4c 36 31 2f 42 20 4c 4f 47 49|
0030|58 35 35 36 31
  +-----+
ASCII:
b'\x01\x00\x0e\x00\x14:`\x9e\x8b\x00\x141756-L61/B LOGIX5561'
```

```
CIP Tool 0.0.0:0
> c
Enter PLC CIP Path: 10.0.0.100,1,0
> who
{'vendor': 1, 'product_type': 14, 'product_code': 54, 'revision': {'major': 20, 'minor': 58}, 'status': b'`0', 'serial': 7572382, 'product_name': '1756-L61/B LOGIX5561'}
```

# Connections

- UCMM
  - Direct to device (no cip routing)
  - No CIP connection
  - Limited functionality
  - Low overhead
- Unconnected
  - Allows for CIP routing between multiple devices
  - CIP connection is immediately closed after the request is complete
  - High overhead
- Connected
  - Allows for CIP routing (same as Unconnected)
  - CIP connection is maintained until closed by a timeout or request
  - High overhead



\* These are all Explicit class 3 connections (AKA: Over TCP)

# Routing

CIP Path: 10.0.10.10, 1, 10, 2, 4, 3, 192.168.1.1, 1, 5

SCADA: 10.0.10.x

EthernetIP



START HERE

10.0.10.10

PLC

Slot 10

Backplane = Port 1

ControlNet on port 2

ControlNet

PLC

HMI

Node 4

DLR on port 3

EthernetIP DLR

IO Network 192.168.1.x

192.168.1.1

PLC

END HERE

Processor Slot 5

Field I/O

# DISCLAIMER

Exploiting Industrial Control Systems can have MAJOR impacts to

1. Human Safety
2. The Environment
3. Physical Infrastructure

Put on your Blue Team hat when working on ICS security and help make our critical infrastructure more secure!

# Fuzzing

- Fuzzing is sending targeted “random” values to the device
  - Easy things to fuzz are Classes, Instances, Attributes, and Services
  - One strategy is to use Get Attribute List (0x03) service on all classes within a range on Instance 0x00. Instance 0x00 is a special instance which is the static object of the class.
    - In the payload for the service, provide:  
“0x00, 0x01, 0x00, 0x01” = List length = 1, Attribute ID 1
    - Use a class ID range of 0x01 to 0x100 for starters.
    - Look for Results that are Error = 0x00, or Error != 0x01  
Error 0x00 = Success
- Non 0x01 errors are specific error codes which could indicate you have hit something

```

def cipFindClassVersionsService3():
    with CIPDriver(address) as plc:
        for classId in range(1, 1000):
            result = plc.generic_message(service=0x03, class_code=classId, instance=0x00, attribute=0x00,
                                         request_data=b'\x01\x00\x01\x00',
                                         connected=False, unconnected_send=True, route_path=True)

    classHex = f"0x{classId:X}"
    className = classes[str(classHex)] if str(classHex) in classes else 'Unknown'

    version = 0
    if result.error is None and len(result.value) > 0:
        printBytes(result.value)
        version = result.value[2] + result.value[3] * 0x10 if len(result.value) > 2 else 0

    if result.error is None:
        print('Class: ' + classHex + ' v' + str(version) + ': ' + className)
    else:
        print('?Class: ' + classHex + ': ' + className + " = " + result.error)

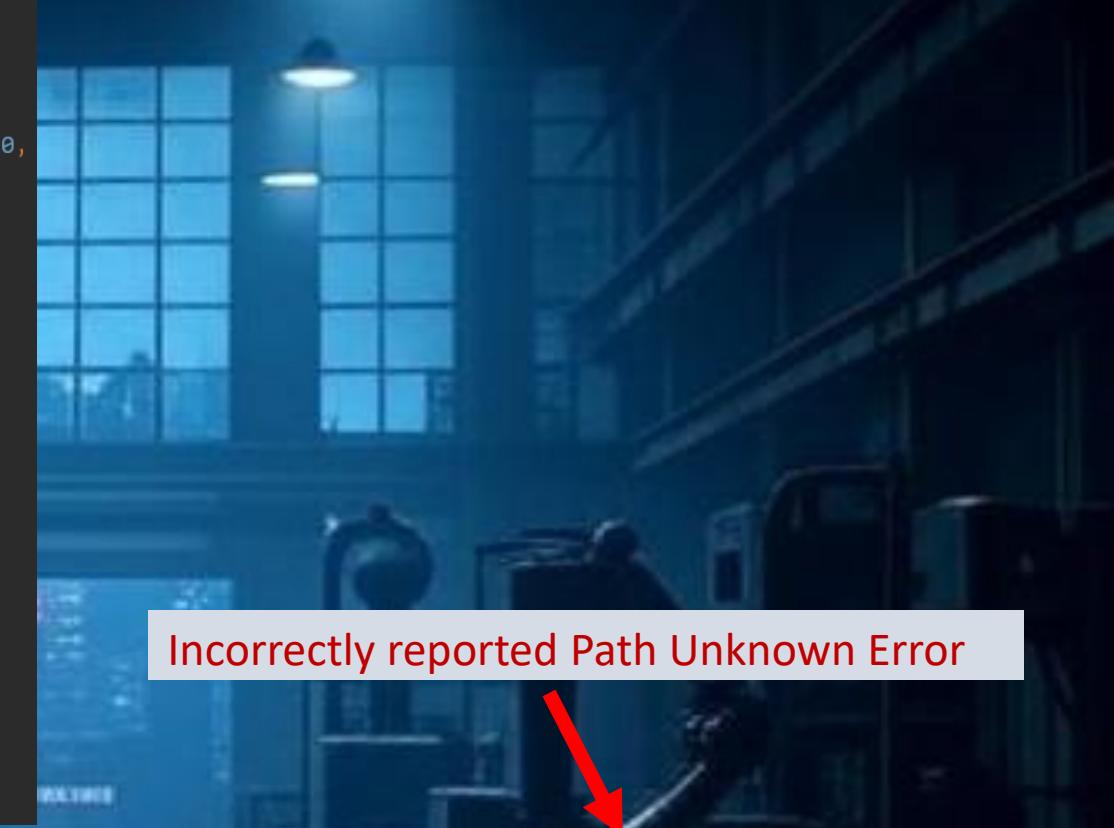
print('Done!')

```

```

?Class: 0xA0: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA1: Unknown = Service not supported
?Class: 0xA2: DF1 Com Driver = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA3: ASCII Com Driver = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA4: Routing Table = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA5: DH PLUS = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA6: DH Plus Interface = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA7: ICP Rack = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA8: Remote I/O = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xA9: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xAA: Ethernet TCP/IP = Service not supported
?Class: 0xAB: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xAC: Change Log = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xAD: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xAE: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xAF: Unknown = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)
?Class: 0xB0: Axis Group = Destination unknown, class unsupported, instance undefined or structure element undefined (see extended status) - Extended status out of memory (05, 00)

```



# Next Steps

- Using fuzzing to find your first MNRF!!!! (Major Non-Recoverable Fault)
  - Once found, report to Rockwell though CISA's VINCE program
  - Most MNRF are actually buffer overflows or other types of memory boundary issues in code
- Taking things further, start looking at what objects to what.
  - Finding objects with specific functions can help build more complex exploits that use existing functionality within the device
- Spend many hours in Wireshark!
  - Analyzing how engineering software uses CIP to interact with the device can help clarify what objects exist and they do

```
▶ Frame 3526: 122 bytes on wire (976 bits), 122 bytes captured (976 bi
▶ Ethernet II, Src: RockwellAuto_f2:d3:70 (5c:88:16:f2:d3:70), Dst: In
▶ Internet Protocol Version 4, Src: 10.0.0.212, Dst: 10.0.0.201
▶ Transmission Control Protocol, Src Port: 44818, Dst Port: 55717, Seq
▶ EtherNet/IP (Industrial Protocol), Session: 0x40000019, Send RR Data
▼ Common Industrial Protocol
  ▶ Service: Get Attributes All (Response)
  ▶ Status: Success:
    [Request Path Size: 2 words]
  ▶ [Request Path: Connection Manager, Instance: 0x01]
▼ CIP Connection Manager
  [Service: Unconnected Send (Response)]
  [Request Path Size: 2 words]
  ▶ [Request Path: Identity, Instance: 0x01]
  [Route Path Size: 1 word]
  ▶ [Route/Connection Path: Port: Backplane, Address: 1]
▼ Get Attributes All (Response)
  ▶ Attribute: 1 (Vendor ID)
  ▶ Attribute: 2 (Device Type)
  ▶ Attribute: 3 (Product Code)
  ▶ Attribute: 4 (Revision)
  ▶ Attribute: 5 (Status)
  ▶ Attribute: 6 (Serial Number)
  ▶ Attribute: 7 (Product Name)
    Product Name: 1756-EWEB
```

# Tools

- Example Code:
  - Look for CIP Tool in the ICS Village GitHub repos  
<https://github.com/ICSVillage>
- Molex EIP Tool
  - Handy free utility for CIP and ENIP testing  
<https://tools.molex.com/webdocs/mysst/EIP%20Tool%20v2.6.1.zip>
- Python Pycomm3
  - Basic library that works great for simple CIP messaging  
<https://docs.pycomm3.dev/en/latest/>
- Ebay: Search “ControlLogix Rack Loaded” to find a test rack
  - Look for 1756-ENBT or 1756-ENET card for ethernet support
  - 1756-L55 or 1756-L6x controllers are find for testing
- CISA VINCE
  - Create a report on any exploits you find so the vendor can fix it!

