Lesson 01: Network Segmentation & Security

Lesson 01: ICS Network Segmentation & Security Architecture

Learning Objectives

- Design defense-in-depth architecture for OT networks using ISA/IEC 62443 standards
- Implement Purdue Model for industrial control systems
- Deploy unidirectional gateways and secure data diodes
- Configure industrial firewalls with deep packet inspection
- Establish secure remote access with zero-trust principles
- Understand network segmentation to counter Module 2 attack techniques

Introduction

Network segmentation is the foundational defensive control for OT environments. Unlike IT networks where defense often focuses on endpoint protection and perimeter security, OT networks rely heavily on **network isolation** to prevent lateral movement, contain incidents, and protect safety-critical systems.

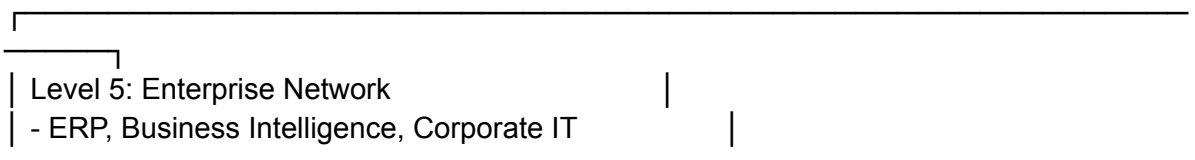
This lesson directly addresses attacks covered in Module 2:

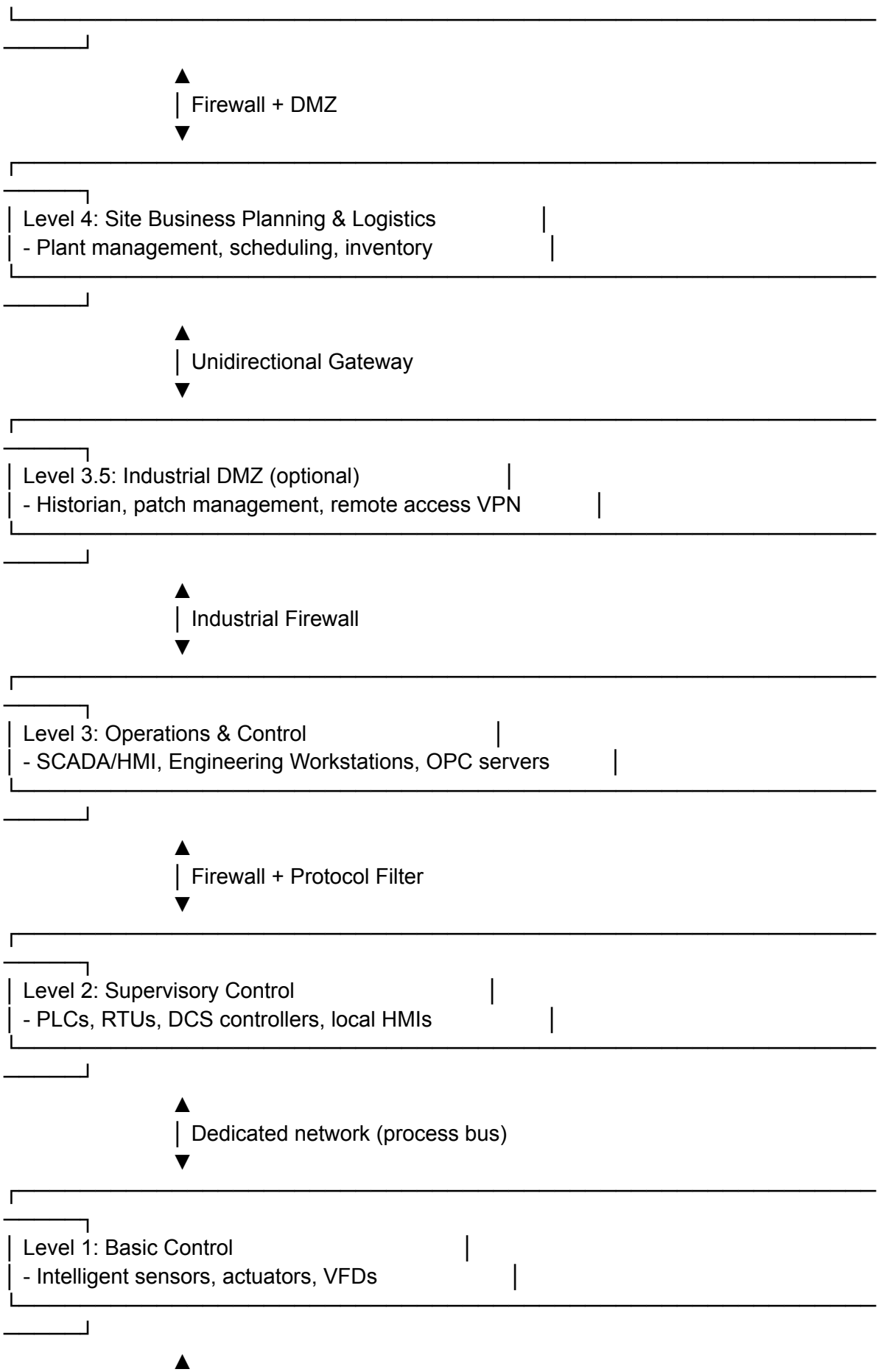
- **Lateral movement** (Module 2 Lesson 02): Segmentation limits attacker pivot paths
- **MITM attacks** (Module 2 Lesson 03): Segmentation reduces attack surface for network positioning
- **Persistence** (Module 2 Lesson 08): Isolated zones prevent multi-layered persistence across network tiers
- **C2 communications** (Module 2 Lesson 09): Unidirectional gateways block outbound command channels

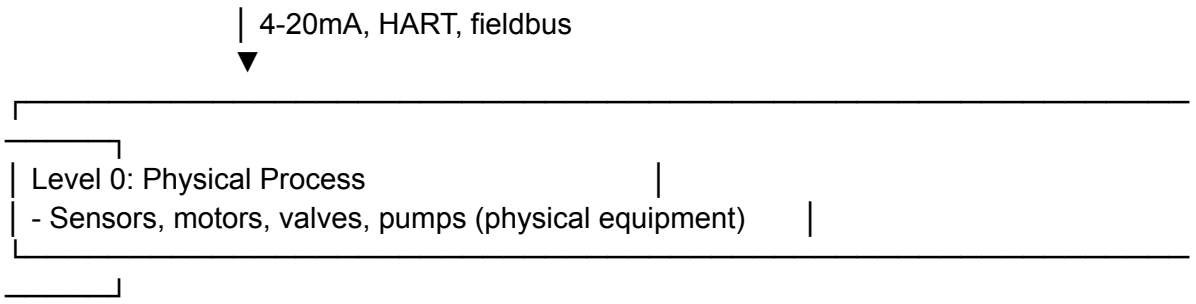
1. The Purdue Model for Industrial Control Systems

1.1 Overview

The **Purdue Enterprise Reference Architecture (PERA)**, also called the Purdue Model, is the de facto standard for organizing ICS networks into hierarchical zones based on function and trust level.




Firewall + DMZ
Level 4: Site Business Planning & Logistics
- Plant management, scheduling, inventory
Unidirectional Gateway
Level 3.5: Industrial DMZ (optional)
- Historian, patch management, remote access VPN
Industrial Firewall
Level 3: Operations & Control
- SCADA/HMI, Engineering Workstations, OPC servers
Firewall + Protocol Filter
Level 2: Supervisory Control
- PLCs, RTUs, DCS controllers, local HMIs
Dedicated network (process bus)
Level 1: Basic Control
- Intelligent sensors, actuators, VFDs



1.2 Level Descriptions

Level 0: Physical Process

- Physical equipment: pumps, motors, valves, actuators
- Analog signals (4-20mA), HART, Foundation Fieldbus
- No IT connectivity

Level 1: Basic Control

- Intelligent field devices: smart sensors, VFDs
- Direct control of Level 0 equipment
- Protocols: Profibus, DeviceNet, Modbus RTU

Level 2: Supervisory Control

- PLCs, RTUs, DCS controllers
- Execute real-time control logic
- Communicate with Level 1 (process bus) and Level 3 (SCADA)
- **Critical security zone**: Compromise here = process manipulation

Level 3: Operations & Control

- SCADA servers, HMIs, Engineering Workstations (EWS)
- Operator interfaces and process monitoring
- Program development and PLC maintenance
- **High-value target**: Attackers often pivot from here to Level 2

Level 3.5: Industrial DMZ

- Intermediary zone between IT and OT
- Historian servers (one-way data from Level 2)
- Patch management servers
- Remote access VPN termination
- **Blast radius containment**: Compromise here shouldn't reach Level 2

Level 4: Site Business Planning

- Manufacturing execution systems (MES)
- Production scheduling
- Inventory management

- Bridge between operations and enterprise

Level 5: Enterprise Network

- Corporate ERP, business intelligence
- Standard IT environment
- **Assumed compromised**: Design OT security with this assumption

1.3 Practical Implementation Example: Water Treatment Plant

```
# water_plant_network_design.py
# Purdue Model implementation for municipal water treatment facility

class PurdueNetworkDesign:
    def __init__(self):
        self.zones = self.define_zones()
        self.conduits = self.define_conduits()

    def define_zones(self):
        """Define all security zones in water treatment plant"""
        return {
            'Level_0': {
                'name': 'Physical Process',
                'devices': ['Chlorine pumps', 'pH sensors', 'Flow meters', 'Valves'],
                'protocols': ['4-20mA analog', 'HART'],
                'network': None  # No IP network
            },
            'Level_1': {
                'name': 'Basic Control',
                'devices': ['Smart actuators', 'VFDs', 'Intelligent sensors'],
                'protocols': ['Modbus RTU', 'Profibus DP'],
                'network': 'Serial/Fieldbus'
            },
            'Level_2_Process': {
                'name': 'Process Control PLCs',
                'devices': ['Water intake PLC (10.20.10.10)',
                            'Treatment PLC (10.20.10.11)',
                            'Distribution PLC (10.20.10.12)'],
                'protocols': ['Modbus TCP', 'S7comm'],
                'network': '10.20.10.0/24',
                'vlan': 20,
                'security_level': 'SL-3'  # High security
            },
            'Level_2_Safety': {
                'name': 'Safety PLCs',
                'devices': ['Emergency shutdown PLC (10.20.20.10)',
                            'Chemical containment PLC (10.20.20.11)'],
                'protocols': ['CIP Safety', 'PROFIsafe'],
                'network': '10.20.20.0/24',
```

```
            'vlan': 21,
            'security_level': 'SL-4',  # Highest security
            'isolation': 'Physically separate network'
        },
        'Level_3_Operations': {
            'name': 'SCADA & HMI',
            'devices': ['SCADA server (10.20.30.50)',
                        'HMI workstations (10.20.30.51-55)',
                        'Engineering workstation (10.20.30.100)'],
            'protocols': ['OPC DA/UA', 'S7comm', 'Modbus TCP'],
            'network': '10.20.30.0/24',
            'vlan': 30,
            'security_level': 'SL-2'
        },
        'Level_3.5_DMZ': {
            'name': 'Industrial DMZ',
            'devices': ['Historian (10.20.40.10)',
                        'Patch server (10.20.40.20)',
                        'VPN gateway (10.20.40.30)'],
            'network': '10.20.40.0/24',
            'vlan': 40,
            'security_level': 'SL-2'
        },
        'Level_4_MES': {
            'name': 'Manufacturing Execution',
            'devices': ['MES server (10.20.50.10)',
                        'Production scheduler (10.20.50.11)'],
            'network': '10.20.50.0/24',
            'vlan': 50,
            'security_level': 'SL-1'
        },
        'Level_5_Enterprise': {
            'name': 'Corporate IT',
            'devices': ['ERP', 'Email', 'File shares'],
            'network': '10.100.0.0/16',
            'security_level': 'SL-0',  # Assume compromised
            'notes': 'Standard IT security controls'
        }
    }

def define_conduits(self):
    """Define allowed communication pathways between zones"""
    return [
        {
            'name': 'SCADA to Process PLCs',
            'source': 'Level_3_Operations',
            'destination': 'Level_2_Process',
            'direction': 'bidirectional',
```

```
            'protocols': ['Modbus TCP:502', 'S7comm:102'],
            'enforcement': 'Industrial firewall with DPI',
            'allowed_sources': ['10.20.30.50'],  # Only SCADA server
            'rule_type': 'whitelist'
        },
        {
            'name': 'Engineering to PLCs',
            'source': 'Level_3_Operations',
            'destination': 'Level_2_Process',
            'direction': 'bidirectional',
            'protocols': ['S7comm:102'],
            'allowed_sources': ['10.20.30.100'],  # Only EWS
            'time_restriction': 'Business hours only',
            'mfa_required': True
        },
        {
            'name': 'Historian Data Collection',
            'source': 'Level_2_Process',
            'destination': 'Level_3.5_DMZ',
            'direction': 'unidirectional',  # OT → IT only
            'enforcement': 'Data diode hardware',
            'protocols': ['OPC UA:4840'],
            'notes': 'IT cannot send commands back'
        },
        {
            'name': 'Enterprise to DMZ',
            'source': 'Level_5_Enterprise',
            'destination': 'Level_3.5_DMZ',
            'direction': 'bidirectional',
            'protocols': ['HTTPS:443', 'SQL:1433'],
            'enforcement': 'Standard firewall',
            'rule_type': 'whitelist'
        },
        {
            'name': 'Safety PLC Isolation',
            'source': 'Level_2_Safety',
            'destination': '*',
            'direction': 'none',
            'enforcement': 'Physically separate network',
            'notes': 'No network connectivity except local HMI'
        }
    ]

def generate_firewall_rules(self, zone_pair):
    """Generate firewall ruleset for zone conduit"""
    rules = []
    conduit = next((c for c in self.conduits
                    if c['source'] == zone_pair[0] and c['destination'] == zone_pair[1]), None)
```

```
        if not conduit:
            return ['deny any any']  # Default deny

        for protocol in conduit.get('protocols', []):
            proto_name, port = protocol.split(':')
            for src_ip in conduit.get('allowed_sources', ['any']):
                dst_network = self.zones[conduit['destination']]['network']
                rule = f"allow tcp {src_ip} -> {dst_network} port {port}  # {proto_name}"
                rules.append(rule)

        rules.append('deny all')  # Explicit deny-all at end
        return rules

    def validate_architecture(self):
        """Check for common security mistakes"""
        issues = []

        # Check 1: Ensure no direct IT-to-PLC communication
        for conduit in self.conduits:
            if conduit['source'] == 'Level_5_Enterprise' and 'Level_2' in conduit['destination']:
                issues.append(f"CRITICAL: Direct IT-to-PLC conduit found: {conduit['name']}")

        # Check 2: Verify safety systems are isolated
        safety_conduits = [c for c in self.conduits if 'Safety' in c['source'] or 'Safety' in
c['destination']]
        if len(safety_conduits) > 0:
            issues.append(f"WARNING: Safety systems have {len(safety_conduits)} network
conduits")

        # Check 3: Check for bidirectional historian connections
        for conduit in self.conduits:
            if 'Historian' in conduit['name'] and conduit['direction'] == 'bidirectional':
                issues.append(f"CRITICAL: Bidirectional historian conduit: {conduit['name']}")

        return issues

# Example usage
design = PurdueNetworkDesign()
issues = design.validate_architecture()

if issues:
    print("Architecture Security Issues:")
    for issue in issues:
        print(f"  - {issue}")
else:
    print("Architecture validation passed")
```

```
# Generate firewall rules for SCADA-to-PLC conduit
rules = design.generate_firewall_rules(('Level_3_Operations', 'Level_2_Process'))
print("\nFirewall Rules (SCADA to PLCs):")
for rule in rules:
    print(f"  {rule}")
```

Expected Output:

```
Architecture validation passed

Firewall Rules (SCADA to PLCs):
  allow tcp 10.20.30.50 -> 10.20.10.0/24 port 502  # Modbus TCP
  allow tcp 10.20.30.50 -> 10.20.10.0/24 port 102  # S7comm
  deny all
```

2. ISA/IEC 62443 Security Levels and Zones

2.1 Security Level (SL) Definitions

ISA/IEC 62443 defines five security levels based on threat sophistication:

Security Level	Threat Profile	Typical Application
SL 0	No protection requirements	Non-critical systems, lab environments
SL 1	Protection against casual or accidental violation	Basic manufacturing, low-criticality processes
SL 2	Protection against intentional violation using simple means (script kiddies, automated tools)	Standard industrial facilities, most PLCs
SL 3	Protection against intentional violation using sophisticated means (skilled attackers with resources)	Critical infrastructure, utilities, chemical plants
SL 4	Protection against intentional violation using sophisticated means with extended resources (nation-state actors)	Nuclear, large-scale water systems, national grid

2.2 Zone and Conduit Model

Security Zone: A grouping of logical or physical assets that share common security requirements.

Conduit: A logical grouping of communication channels connecting two or more zones.

```
# isa62443_implementation.py
# Implement ISA/IEC 62443 zone and conduit model

import ipaddress
from enum import Enum

class SecurityLevel(Enum):
    SL0 = 0
    SL1 = 1
    SL2 = 2
    SL3 = 3
    SL4 = 4

class SecurityZone:
    def __init__(self, name, security_level, network, criticality):
        self.name = name
        self.security_level = security_level
        self.network = ipaddress.ip_network(network)
        self.criticality = criticality  # safety, production, business
        self.assets = []

    def add_asset(self, asset):
        """Add asset to zone"""
        if ipaddress.ip_address(asset['ip']) in self.network:
            self.assets.append(asset)
            return True
        return False

    def get_security_requirements(self):
        """Return security controls required for this SL"""
        requirements = {
            SecurityLevel.SL0: [],
            SecurityLevel.SL1: [
                'User authentication',
                'Audit logging'
            ],
            SecurityLevel.SL2: [
                'Multi-factor authentication',
                'Encryption in transit',
                'Intrusion detection',
                'Security event logging'
            ],
            SecurityLevel.SL3: [
```

```
            'Role-based access control',
            'Strong encryption (AES-256)',
            'Network segmentation',
            'Continuous monitoring',
            'Integrity verification'
        ],
        SecurityLevel.SL4: [
            'Defense-in-depth',
            'Unidirectional gateways',
            'Hardware security modules',
            'Tamper detection',
            'Air-gapped networks',
            'Real-time threat intelligence'
        ]
    }

    # Cumulative requirements
    req_list = []
    for level in SecurityLevel:
        req_list.extend(requirements[level])
        if level == self.security_level:
            break

    return list(set(req_list))  # Remove duplicates

class Conduit:
    def __init__(self, name, source_zone, dest_zone):
        self.name = name
        self.source_zone = source_zone
        self.dest_zone = dest_zone
        self.allowed_protocols = []
        self.enforcement_mechanism = None
        self.direction = 'bidirectional'  # or 'unidirectional'

    def determine_required_sl(self):
        """Conduit SL = max(source SL, destination SL)"""
        return max(self.source_zone.security_level,
                   self.dest_zone.security_level)

    def add_protocol(self, protocol, port, direction='bidirectional'):
        """Add allowed protocol to conduit"""
        self.allowed_protocols.append({
            'protocol': protocol,
            'port': port,
            'direction': direction
        })

    def get_enforcement_requirements(self):
```

```
        """Determine required enforcement mechanism based on SL"""
        required_sl = self.determine_required_sl()

        if required_sl == SecurityLevel.SL4:
            return 'Unidirectional gateway with DPI and encrypted tunnels'
        elif required_sl == SecurityLevel.SL3:
            return 'Industrial firewall with DPI and IDS'
        elif required_sl == SecurityLevel.SL2:
            return 'Firewall with protocol filtering'
        else:
            return 'Basic ACLs'

# Example: Power plant implementation
def design_power_plant_network():
    # Define zones
    turbine_control = SecurityZone(
        name='Turbine Control System',
        security_level=SecurityLevel.SL4,  # Safety-critical
        network='10.10.10.0/24',
        criticality='safety'
    )

    scada_operations = SecurityZone(
        name='SCADA Operations',
        security_level=SecurityLevel.SL3,
        network='10.10.20.0/24',
        criticality='production'
    )

    historian_dmz = SecurityZone(
        name='Historian DMZ',
        security_level=SecurityLevel.SL2,
        network='10.10.30.0/24',
        criticality='business'
    )

    # Add assets
    turbine_control.add_asset({'ip': '10.10.10.10', 'type': 'Safety PLC', 'vendor': 'Siemens'})
    scada_operations.add_asset({'ip': '10.10.20.50', 'type': 'SCADA Server', 'vendor':
'Ignition'})

    # Define conduits
    scada_to_turbine = Conduit(
        name='SCADA to Turbine Control',
        source_zone=scada_operations,
        dest_zone=turbine_control
    )
    scada_to_turbine.add_protocol('Modbus TCP', 502)
```

```
    scada_to_turbine.add_protocol('OPC UA', 4840)

    turbine_to_historian = Conduit(
        name='Turbine to Historian',
        source_zone=turbine_control,
        dest_zone=historian_dmz
    )
    turbine_to_historian.direction = 'unidirectional'  # One-way only
    turbine_to_historian.add_protocol('OPC UA', 4840, direction='outbound')

    # Generate security requirements
    print(f"Zone: {turbine_control.name}")
    print(f"Security Level: {turbine_control.security_level.name}")
    print(f"Required Controls:")
    for control in turbine_control.get_security_requirements():
        print(f"  - {control}")

    print(f"\nConduit: {scada_to_turbine.name}")
    print(f"Required SL: SL-{scada_to_turbine.determine_required_sl().value}")
    print(f"Enforcement: {scada_to_turbine.get_enforcement_requirements()}")

design_power_plant_network()
```

Output:

```
Zone: Turbine Control System
Security Level: SL4
Required Controls:
  - User authentication
  - Audit logging
  - Multi-factor authentication
  - Encryption in transit
  - Intrusion detection
  - Security event logging
  - Role-based access control
  - Strong encryption (AES-256)
  - Network segmentation
  - Continuous monitoring
  - Integrity verification
  - Defense-in-depth
  - Unidirectional gateways
  - Hardware security modules
  - Tamper detection
  - Air-gapped networks
  - Real-time threat intelligence

Conduit: SCADA to Turbine Control
Required SL: SL-4
```

Enforcement: Unidirectional gateway with DPI and encrypted tunnels

3. Unidirectional Gateways and Data Diodes

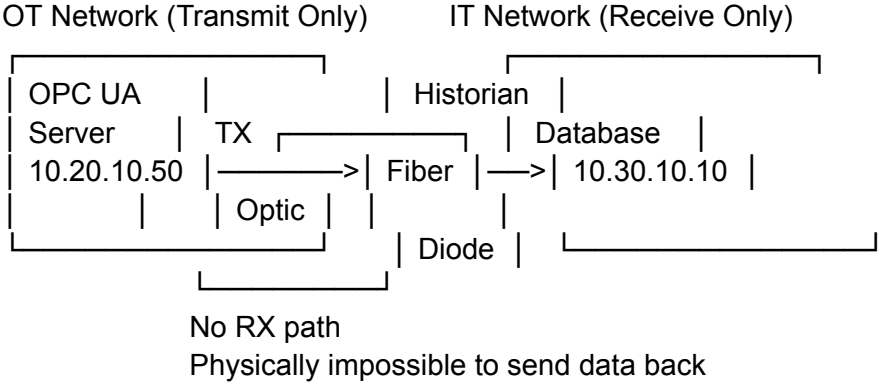
3.1 Purpose

Unidirectional gateways (data diodes) provide **hardware-enforced** one-way data flow from OT to IT networks. This completely eliminates the risk of:

- IT malware propagating to OT (defense against Module 2 Lesson 07 supply chain attacks)
- C2 command channels from internet to PLCs (blocks Module 2 Lesson 09 C2)
- Ransomware spreading from enterprise to process control (defense against NotPetya-style attacks)

3.2 How Data Diodes Work

Physical Implementation:



The TX (transmit) fiber from OT connects to RX (receive) fiber on IT side. There is **no physical return path** - the IT side literally cannot send packets back to OT.

3.3 Implementation Example

```
# data_diode_proxy.py
# Application-layer proxy for unidirectional gateway
# Runs on OT side to push data through hardware diode

import time
import snap7
from opcua import Client as OPCClient
import json
import socket

class DataDiodeProxy:
    """
    Collects data from OT devices and pushes through unidirectional gateway
```

```
Runs on OT-side gateway appliance
"""
def __init__(self, diode_ip, diode_port):
    self.diode_ip = diode_ip
    self.diode_port = diode_port
    self.plc_clients = {}
    self.opc_clients = {}

def add_plc_source(self, name, ip, rack=0, slot=1):
    """Add Siemens PLC as data source"""
    client = snap7.client.Client()
    client.connect(ip, rack, slot)
    self.plc_clients[name] = {
        'client': client,
        'ip': ip
    }

def add_opc_source(self, name, endpoint):
    """Add OPC UA server as data source"""
    client = OPCClient(endpoint)
    client.connect()
    self.opc_clients[name] = {
        'client': client,
        'endpoint': endpoint
    }

def collect_plc_data(self, plc_name, tags):
    """Read data from PLC"""
    plc = self.plc_clients[plc_name]['client']
    data = {}

    for tag in tags:
        # Read tag based on type
        if tag['type'] == 'DB':
            raw_data = plc.db_read(tag['db_number'], tag['start'], tag['size'])
            # Parse based on data type
            if tag['datatype'] == 'REAL':
                value = snap7.util.get_real(raw_data, 0)
            elif tag['datatype'] == 'INT':
                value = snap7.util.get_int(raw_data, 0)
            else:
                value = raw_data.hex()

            data[tag['name']] = value

    return data

def collect_opc_data(self, opc_name, node_ids):
```

```
        """Read data from OPC UA server"""
        opc = self.opc_clients[opc_name]['client']
        data = {}

        for node_id in node_ids:
            node = opc.get_node(node_id)
            data[node_id] = node.get_value()

        return data

    def push_through_diode(self, payload):
        """
        Push data through hardware diode (one-way UDP)
        Uses UDP because no ACK is possible (diode blocks return traffic)
        """
        try:
            # Serialize to JSON
            message = json.dumps(payload).encode('utf-8')

            # Send via UDP (fire-and-forget)
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.sendto(message, (self.diode_ip, self.diode_port))
            sock.close()

            return True
        except Exception as e:
            print(f"Error pushing through diode: {e}")
            return False

    def run_collection_loop(self):
        """Main loop: collect and push data every 5 seconds"""

        # Define PLC tags to collect
        plc_tags = [
            {'name': 'Water_Flow', 'type': 'DB', 'db_number': 1, 'start': 0, 'size': 4, 'datatype':
'REAL'},
            {'name': 'Chlorine_PPM', 'type': 'DB', 'db_number': 1, 'start': 4, 'size': 4, 'datatype':
'REAL'},
            {'name': 'pH_Level', 'type': 'DB', 'db_number': 1, 'start': 8, 'size': 4, 'datatype': 'REAL'},
        ]

        # Define OPC nodes to collect
        opc_nodes = [
            'ns=2;s=Tank1.Level',
            'ns=2;s=Tank1.Temperature',
            'ns=2;s=Pump1.Status'
        ]
```

```
        while True:
            try:
                payload = {
                    'timestamp': time.time(),
                    'source': 'WaterPlant_OT',
                    'plc_data': {},
                    'opc_data': {}
                }

                # Collect from all PLCs
                for plc_name in self.plc_clients:
                    payload['plc_data'][plc_name] = self.collect_plc_data(plc_name, plc_tags)

                # Collect from all OPC servers
                for opc_name in self.opc_clients:
                    payload['opc_data'][opc_name] = self.collect_opc_data(opc_name, opc_nodes)

                # Push through diode
                success = self.push_through_diode(payload)

                if success:
                    print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}] Data pushed through diode")

            except Exception as e:
                print(f"Collection error: {e}")

            time.sleep(5)  # Collect every 5 seconds

# Usage example
if __name__ == '__main__':
    # Initialize proxy
    proxy = DataDiodeProxy(
        diode_ip='10.20.40.10',  # Diode appliance IP (OT side)
        diode_port=5000
    )

    # Add data sources
    proxy.add_plc_source('Intake_PLC', '10.20.10.10')
    proxy.add_plc_source('Treatment_PLC', '10.20.10.11')
    proxy.add_opc_source('SCADA_OPC', 'opc.tcp://10.20.30.50:4840')

    # Start collection loop
    print("Data diode proxy started. Pushing data to IT network...")
    proxy.run_collection_loop()
```

3.4 IT-Side Receiver (Behind Data Diode)

```
# diode_receiver.py
```

```
# Runs on IT side to receive unidirectional data
# Cannot send anything back to OT (hardware prevents it)

import socket
import json
import sqlite3
from datetime import datetime

class DiodeReceiver:
    """Receive data from unidirectional gateway and store in historian"""

    def __init__(self, listen_port, db_path='historian.db'):
        self.listen_port = listen_port
        self.db_path = db_path
        self.init_database()

    def init_database(self):
        """Initialize historian database"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS process_data (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp REAL,
                source TEXT,
                tag_name TEXT,
                value REAL,
                received_at TEXT
            )
        ''')

        conn.commit()
        conn.close()

    def store_data(self, payload):
        """Store received data in historian database"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        # Store PLC data
        for plc_name, tags in payload.get('plc_data', {}).items():
            for tag_name, value in tags.items():
                cursor.execute('''
                    INSERT INTO process_data (timestamp, source, tag_name, value, received_at)
                    VALUES (?, ?, ?, ?, ?)
                ''', (payload['timestamp'], plc_name, tag_name, value, datetime.now().isoformat()))
```

```
        # Store OPC data
        for opc_name, nodes in payload.get('opc_data', {}).items():
            for node_id, value in nodes.items():
                cursor.execute('''
                    INSERT INTO process_data (timestamp, source, tag_name, value, received_at)
                    VALUES (?, ?, ?, ?, ?)
                ''', (payload['timestamp'], opc_name, node_id, value, datetime.now().isoformat()))

        conn.commit()
        conn.close()

    def listen(self):
        """Listen for UDP data from diode"""
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind(('0.0.0.0', self.listen_port))

        print(f"Listening for data from diode on port {self.listen_port}...")

        while True:
            try:
                data, addr = sock.recvfrom(65535)  # Max UDP packet size
                payload = json.loads(data.decode('utf-8'))

                self.store_data(payload)
                print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] Received data from
{payload['source']}")

            except Exception as e:
                print(f"Error receiving data: {e}")

# Usage
if __name__ == '__main__':
    receiver = DiodeReceiver(listen_port=5000)
    receiver.listen()
```

3.5 Commercial Data Diode Solutions

- **Waterfall Security**: Unidirectional CloudConnect
- **Owl Cyber Defense**: DualDiode and OPDS
- **BAE Systems**: Unidirectional Gateway
- **Hirschmann**: EAGLE Tofino Data Diode

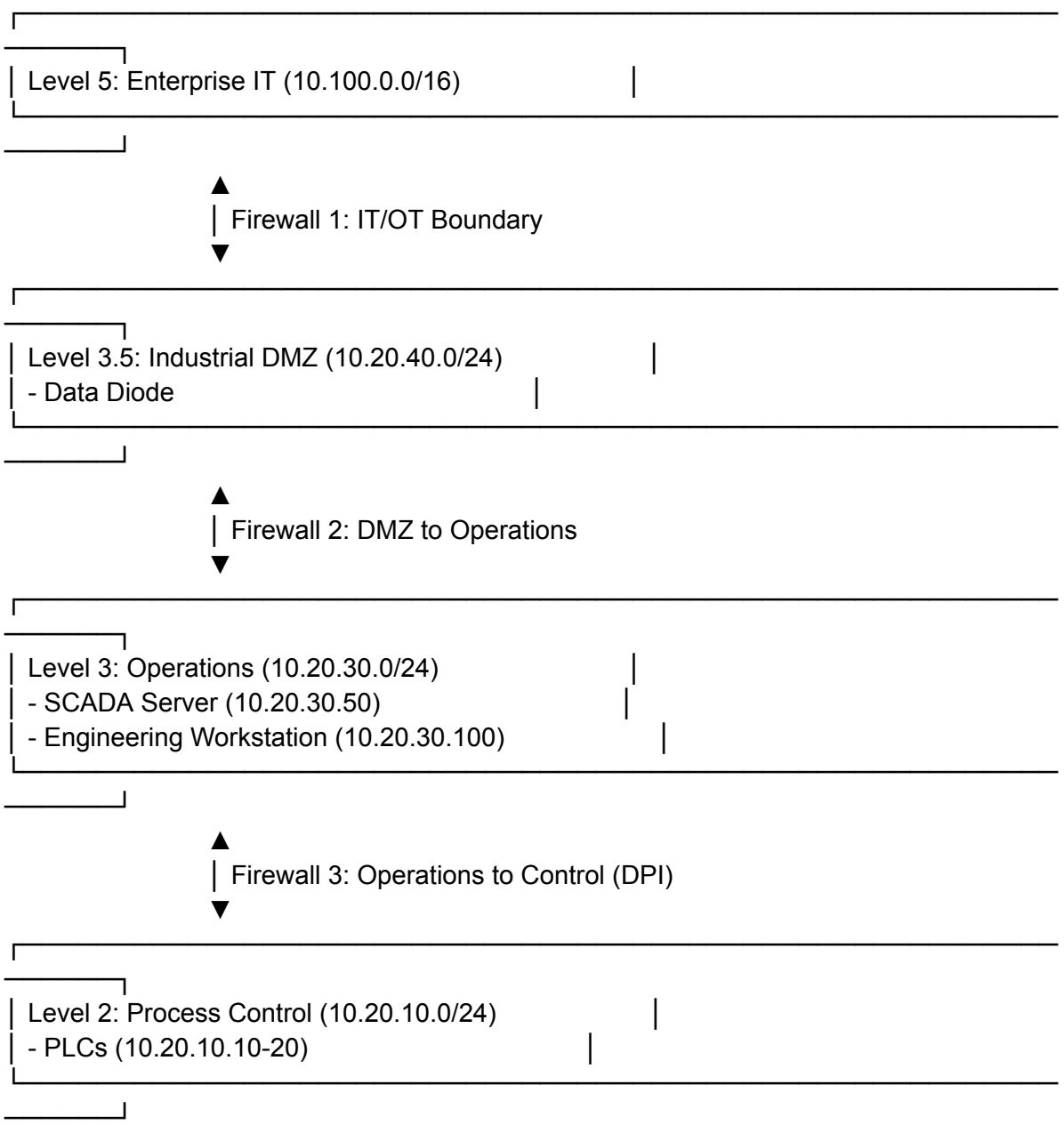
4. Industrial Firewall Configuration

Unlike traditional IT firewalls, industrial firewalls must:

- Understand OT protocols (Modbus, S7comm, DNP3, etc.)

- Perform deep packet inspection (DPI) on industrial traffic
- Enforce function-code level restrictions (e.g., allow Modbus reads but block writes)
- Maintain sub-10ms latency for real-time traffic

4.1 Firewall Placement



Critical Firewall: Firewall 3 (Operations to Control) - This must inspect all SCADA-to-PLC traffic

4.2 Palo Alto Firewall Configuration for OT

<!-- palo_alto_ot_policy.xml -->
<!-- Deep packet inspection rules for Modbus traffic -->

<config>

```
<devices>
  <entry name="localhost.localdomain">
    <vsys>
      <entry name="vsys1">

        <!-- Security Zones -->
        <zone>
          <entry name="OT_Operations">
            <network>
              <layer3>
                <member>ethernet1/1</member>
              </layer3>
            </network>
          </entry>
          <entry name="OT_Control">
            <network>
              <layer3>
                <member>ethernet1/2</member>
              </layer3>
            </network>
          </entry>
        </zone>

        <!-- Address Objects -->
        <address>
          <entry name="SCADA_Server">
            <ip-netmask>10.20.30.50/32</ip-netmask>
          </entry>
          <entry name="Engineering_Workstation">
            <ip-netmask>10.20.30.100/32</ip-netmask>
          </entry>
          <entry name="PLC_Network">
            <ip-netmask>10.20.10.0/24</ip-netmask>
          </entry>
        </address>

        <!-- Security Policies -->
        <rulebase>
          <security>
            <rules>

              <!-- Allow SCADA to read from PLCs -->
              <entry name="SCADA_Read_PLCs">
                <from><member>OT_Operations</member></from>
                <to><member>OT_Control</member></to>
                <source><member>SCADA_Server</member></source>
                <destination><member>PLC_Network</member></destination>
                <application><member>modbus</member></application>
```

```
  <service><member>application-default</member></service>
  <action>allow</action>
  <profile-setting>
    <profiles>
      <url-filtering><member>default</member></url-filtering>
      <file-blocking><member>strict-file-blocking</member></file-blocking>
      <virus><member>default</member></virus>
      <spyware><member>strict</member></spyware>
      <vulnerability><member>strict</member></vulnerability>
    </profiles>
  </profile-setting>
  <log-start>yes</log-start>
  <log-end>yes</log-end>
</entry>

<!-- Block Modbus writes from SCADA (read-only operation) -->
<entry name="Block_SCADA_Writes">
  <from><member>OT_Operations</member></from>
  <to><member>OT_Control</member></to>
  <source><member>SCADA_Server</member></source>
  <destination><member>PLC_Network</member></destination>
  <application><member>modbus</member></application>
  <service><member>application-default</member></service>
  <action>deny</action>
  <!-- Deep packet inspection on Modbus function codes -->
  <option>
    <disable-server-response-inspection>no</disable-server-response-inspection>
  </option>
  <category><member>modbus-write</member></category>
  <log-start>yes</log-start>
  <log-end>yes</log-end>
</entry>

<!-- Allow Engineering to program PLCs (with MFA) -->
<entry name="EWS_Program_PLCs">
  <from><member>OT_Operations</member></from>
  <to><member>OT_Control</member></to>
  <source><member>Engineering_Workstation</member></source>
  <destination><member>PLC_Network</member></destination>
  <application>
    <member>s7comm</member>
    <member>modbus</member>
  </application>
  <service><member>application-default</member></service>
  <action>allow</action>
  <!-- Require GlobalProtect MFA -->
  <source-user><member>engineering-group</member></source-user>
  <hip-profiles><member>ot-engineering-compliance</member></hip-profiles>
```

```
            <log-start>yes</log-start>
            <log-end>yes</log-end>
          </entry>

          <!-- Block all internet access from OT -->
          <entry name="Block_OT_Internet">
            <from>
              <member>OT_Operations</member>
              <member>OT_Control</member>
            </from>
            <to><member>External</member></to>
            <source><member>any</member></source>
            <destination><member>any</member></destination>
            <application><member>any</member></application>
            <service><member>any</member></service>
            <action>deny</action>
            <log-end>yes</log-end>
          </entry>

          <!-- Default deny all -->
          <entry name="Default_Deny">
            <from><member>any</member></from>
            <to><member>any</member></to>
            <source><member>any</member></source>
            <destination><member>any</member></destination>
            <application><member>any</member></application>
            <service><member>any</member></service>
            <action>deny</action>
            <log-end>yes</log-end>
          </entry>

        </rules>
      </security>
    </rulebase>

  </entry>
 </vsys>
</entry>
</devices>
</config>
```

4.3 Open-Source Alternative: Tofino Firewall Configuration

```
# tofino_modbus_filter.py
# Hirschmann Tofino LSM (Loadable Security Module)
# Filters Modbus traffic based on function codes

from scapy.all import *
```

```
from scapy.contrib.modbus import *

class ModbusFirewall:
    """
    Software-based Modbus firewall for educational purposes
    Production environments should use hardware industrial firewalls
    """

    def __init__(self, trusted_sources, allowed_function_codes):
        self.trusted_sources = trusted_sources
        self.allowed_function_codes = allowed_function_codes
        self.blocked_count = 0
        self.allowed_count = 0

    def inspect_modbus_packet(self, packet):
        """Deep packet inspection on Modbus TCP"""

        if not packet.haslayer(ModbusADURequest) and not
packet.haslayer(ModbusADUResponse):
            return True  # Not Modbus, pass through

        # Extract source IP
        src_ip = packet[IP].src

        # Check source authorization
        if src_ip not in self.trusted_sources:
            self.log_violation(packet, reason="Untrusted source")
            self.blocked_count += 1
            return False

        # Check function code
        if packet.haslayer(ModbusADURequest):
            func_code = packet[ModbusADURequest].funcCode

            if func_code not in self.allowed_function_codes.get(src_ip, []):
                self.log_violation(packet, reason=f"Unauthorized function code {func_code}")
                self.blocked_count += 1
                return False

        # Check for suspicious patterns
        if self.detect_attack_patterns(packet):
            self.log_violation(packet, reason="Attack pattern detected")
            self.blocked_count += 1
            return False

        self.allowed_count += 1
        return True
```

```
    def detect_attack_patterns(self, packet):
        """Detect known Modbus attack patterns"""

        if packet.haslayer(ModbusADURequest):
            req = packet[ModbusADURequest]

            # Check for excessive register read (recon)
            if hasattr(req, 'quantity') and req.quantity > 100:
                return True  # Potential reconnaissance

            # Check for diagnostic function codes (used in scanning)
            if req.funcCode == 0x08:  # Diagnostics
                return True

            # Check for program download function (S7comm)
            # (Would need S7comm parser for full implementation)

        return False

    def log_violation(self, packet, reason):
        """Log security policy violation"""
        print(f"[BLOCK] {packet[IP].src}:{packet[TCP].sport} -> "
              f"{packet[IP].dst}:{packet[TCP].dport} | {reason}")

        # In production: send to SIEM
        # syslog.send(f"Modbus violation: {reason}", severity="WARNING")

    def get_statistics(self):
        """Return firewall statistics"""
        total = self.allowed_count + self.blocked_count
        block_rate = (self.blocked_count / total * 100) if total > 0 else 0

        return {
            'allowed': self.allowed_count,
            'blocked': self.blocked_count,
            'total': total,
            'block_rate_percent': block_rate
        }

# Example configuration
if __name__ == '__main__':
    # Define trusted sources and their allowed function codes
    firewall = ModbusFirewall(
        trusted_sources=['10.20.30.50', '10.20.30.100'],
        allowed_function_codes={
            '10.20.30.50': [1, 2, 3, 4],  # SCADA: Read coils, inputs, holdings, input registers
            '10.20.30.100': [1, 2, 3, 4, 5, 6, 15, 16, 23]  # EWS: Read + Write
        }
```

```
)

# Capture and filter traffic (inline mode)
def packet_callback(packet):
    if packet.haslayer(TCP) and packet[TCP].dport == 502:
        if not firewall.inspect_modbus_packet(packet):
            # Drop packet (would require iptables integration in production)
            return
    # Forward allowed packets
    send(packet)

sniff(iface='eth0', prn=packet_callback, filter='tcp port 502')
```

5. Secure Remote Access Architecture

5.1 Challenges

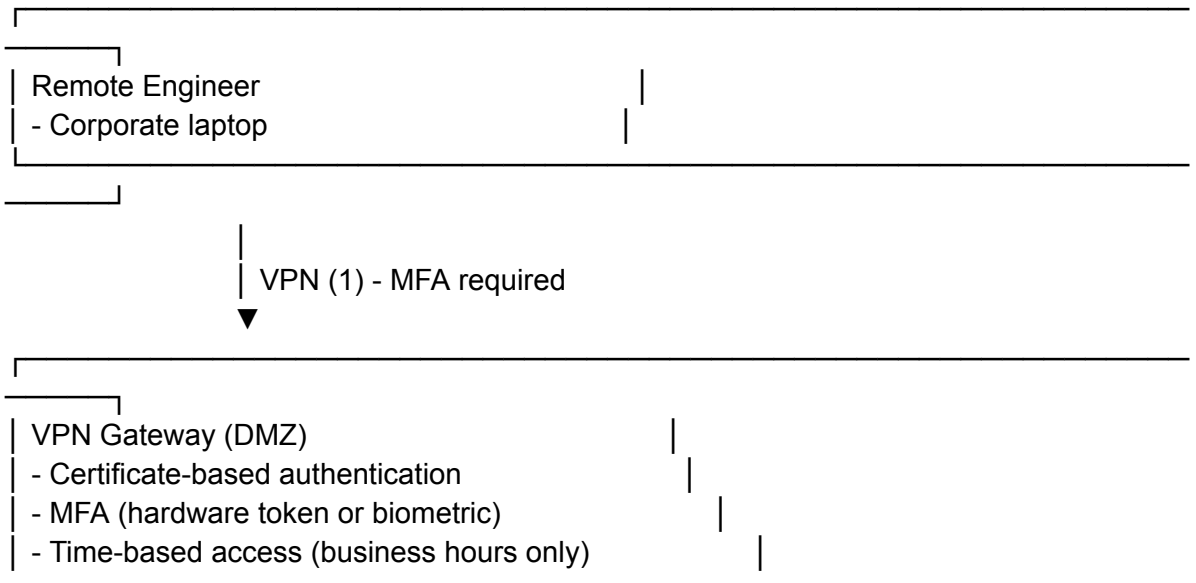
OT environments often require remote access for:

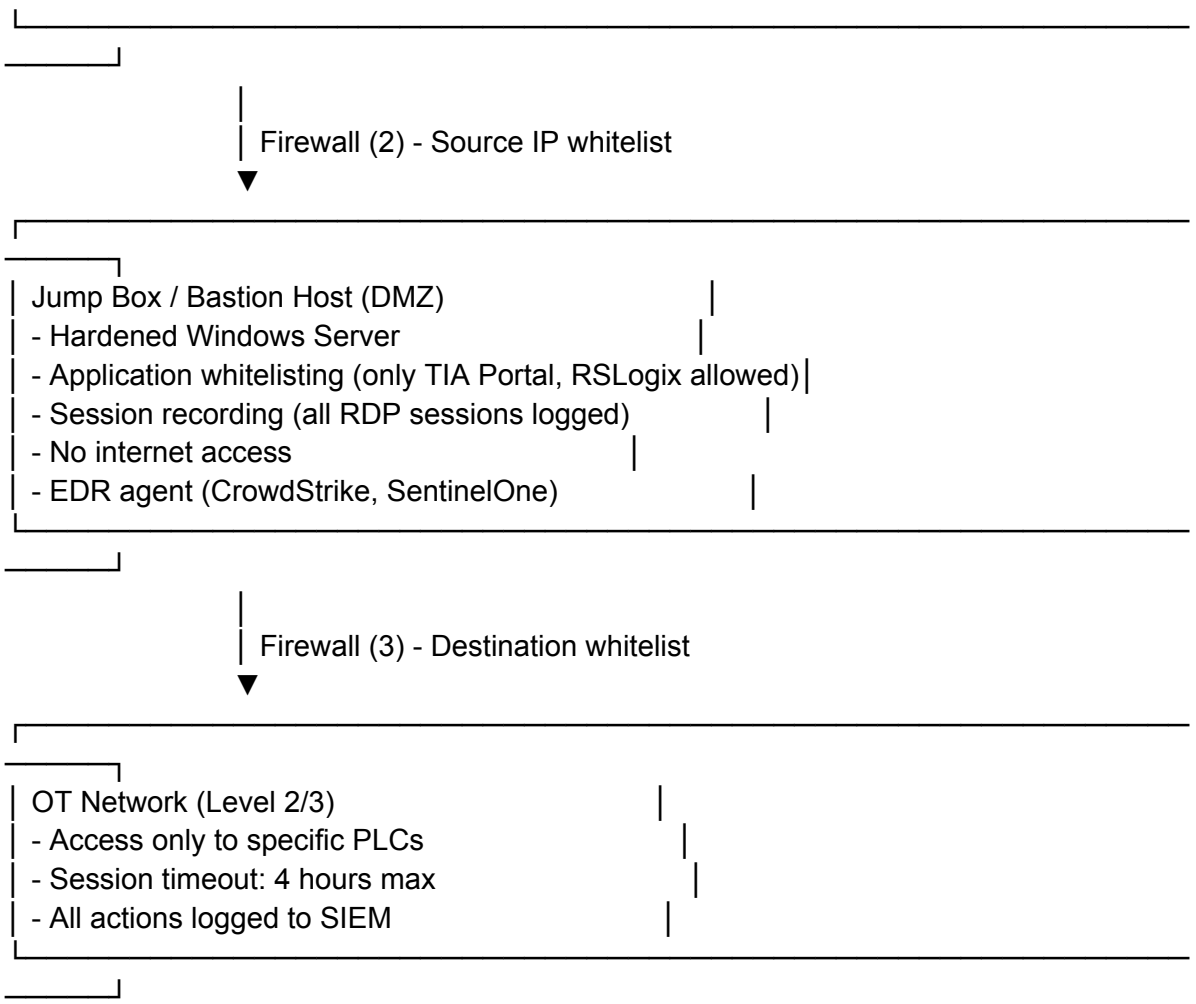
- Vendor support
- Remote engineering
- After-hours monitoring
- Multi-site management

However, remote access introduces risk:

- VPN vulnerabilities (Module 2 covers VPN exploits)
- Stolen credentials
- Insider threats
- Persistence via remote access tools

5.2 Defense-in-Depth Remote Access





5.3 Implementation: VPN with MFA

```
#!/bin/bash
# configure_ot_vpn.sh
# Configure Cisco ASA or pfSense for OT remote access

# 1. Create separate VPN pool for OT access
configure terminal
ip local pool OT_REMOTE_POOL 10.20.99.100-10.20.99.200 mask 255.255.255.0

# 2. Enable certificate-based authentication
crypto ca trustpoint OT_VPN_CA
 enrollment url http://ca.company.com:80
 subject-name CN=ot-vpn-user
 revocation-check crl
 rsakeypair OT_VPN_KEY 2048

# 3. Configure group policy with restrictions
group-policy OT_REMOTE_ACCESS internal
group-policy OT_REMOTE_ACCESS attributes
 vpn-tunnel-protocol ssl-client
```

```
 split-tunnel-policy tunnelspecified
 split-tunnel-network-list value OT_NETWORKS_ONLY
 vpn-idle-timeout 30  # Disconnect after 30 min idle
 vpn-session-timeout 240  # Max 4 hour session

# 4. Enable MFA (Duo or RSA SecurID)
aaa-server DUO_MFA protocol radius
aaa-server DUO_MFA host api.duosecurity.com
 key <SECRET_KEY>

# 5. Create tunnel group
tunnel-group OT_REMOTE_VPN type remote-access
tunnel-group OT_REMOTE_VPN general-attributes
 address-pool OT_REMOTE_POOL
 authentication-server-group DUO_MFA
 default-group-policy OT_REMOTE_ACCESS

# 6. ACL for split tunneling (only OT networks)
access-list OT_NETWORKS_ONLY standard permit 10.20.10.0 255.255.255.0  # PLCs
access-list OT_NETWORKS_ONLY standard permit 10.20.30.0 255.255.255.0  # SCADA
access-list OT_NETWORKS_ONLY standard deny any

# 7. Time-based access control (business hours only)
time-range BUSINESS_HOURS
 periodic weekdays 06:00 to 18:00
 periodic weekend 08:00 to 12:00

access-list VPN_TO_OT extended permit ip 10.20.99.0 255.255.255.0 10.20.10.0
255.255.255.0 time-range BUSINESS_HOURS
access-list VPN_TO_OT extended deny ip any any

# 8. Logging
logging enable
logging trap informational
logging host dmz 10.20.40.50  # SIEM server
```

5.4 Jump Box Hardening

```
# harden_jump_box.ps1
# Harden Windows Server 2019/2022 jump box for OT access

# 1. Enable AppLocker (application whitelisting)
$RuleCollection = @"
<AppLockerPolicy Version="1">
  <RuleCollection Type="Exe" EnforcementMode="Enabled">
    <!-- Allow TIA Portal -->
    <FilePathRule Id="TIA_PORTAL" Name="Siemens TIA Portal"
UserOrGroupSid="S-1-5-32-545" Action="Allow">
```

```
      <Conditions>
        <FilePathCondition Path="C:\Program Files\Siemens\Automation\Portal V17\*" />
      </Conditions>
    </FilePathRule>

    <!-- Allow RSLogix -->
    <FilePathRule Id="RSLOGIX" Name="Rockwell RSLogix 5000"
UserOrGroupSid="S-1-5-32-545" Action="Allow">
      <Conditions>
        <FilePathCondition Path="C:\Program Files (x86)\Rockwell Software\RSLogix 5000\*"
/>
      </Conditions>
    </FilePathRule>

    <!-- Allow System binaries -->
    <FilePathRule Id="SYSTEM" Name="Windows System" UserOrGroupSid="S-1-1-0"
Action="Allow">
      <Conditions>
        <FilePathCondition Path="%WINDIR%\*" />
        <FilePathCondition Path="%PROGRAMFILES%\*" />
      </Conditions>
    </FilePathRule>

    <!-- Deny everything else -->
    <FilePathRule Id="DENY_ALL" Name="Deny All" UserOrGroupSid="S-1-1-0"
Action="Deny">
      <Conditions>
        <FilePathCondition Path="*" />
      </Conditions>
    </FilePathRule>
  </RuleCollection>
</AppLockerPolicy>
"@

$RuleCollection | Out-File C:\applocker_policy.xml
Set-AppLockerPolicy -XMLPolicy C:\applocker_policy.xml

# 2. Disable USB storage
Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\USBSTOR" -Name
"Start" -Value 4

# 3. Enable PowerShell logging
$RegPath =
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging"
New-Item -Path $RegPath -Force
Set-ItemProperty -Path $RegPath -Name "EnableScriptBlockLogging" -Value 1

# 4. Configure firewall (whitelist only specific OT devices)
```

```
Remove-NetFirewallRule -All  # Clear all rules
New-NetFirewallRule -DisplayName "Allow SCADA Server" -Direction Outbound
-RemoteAddress 10.20.30.50 -Action Allow
New-NetFirewallRule -DisplayName "Allow PLC Network" -Direction Outbound
-RemoteAddress 10.20.10.0/24 -Action Allow
New-NetFirewallRule -DisplayName "Block Internet" -Direction Outbound -RemoteAddress
0.0.0.0/0 -Action Block

# 5. Enable RDP session recording
# Requires third-party tool like Observeit or native Windows Session Recording

# 6. Install EDR agent
Start-Process "\\file-server\EDR\CrowdStrike-Installer.exe" -ArgumentList "/quiet" -Wait

# 7. Disable unnecessary services
$ServicesToDisable = @(
    "RemoteRegistry",
    "WinRM",
    "SSDPSRV",  # SSDP Discovery
    "upnphost"   # UPnP
)

foreach ($service in $ServicesToDisable) {
    Stop-Service -Name $service -Force
    Set-Service -Name $service -StartupType Disabled
}

# 8. Enable audit logging
auditpol /set /subcategory:"Logon" /success:enable /failure:enable
auditpol /set /subcategory:"Process Creation" /success:enable
auditpol /set /subcategory:"File System" /success:enable /failure:enable

Write-Host "Jump box hardening complete"
```

6. Hands-On Lab: Design and Implement Network Segmentation

Lab Environment

Objective: Design and deploy network segmentation for a small water treatment plant

Assets:

- 3 PLCs (Siemens S7-1200)
- 1 SCADA server (Windows Server 2019 + Ignition SCADA)
- 1 Engineering workstation (Windows 10 + TIA Portal)
- 1 Historian server (Windows Server 2019 + OSIsoft PI)

- Corporate IT network (assume compromised)

Requirements:

- Implement 5-level Purdue architecture
- Configure industrial firewall with whitelist rules
- Deploy unidirectional gateway for historian
- Establish secure remote access

Lab Steps

Step 1: Network Design

Create network diagram in draw.io or Visio:

```
[Corporate IT: 10.100.0.0/16]
        |
   [Firewall 1]
        |
[DMZ: 10.20.40.0/24]
- Historian: 10.20.40.10
- VPN Gateway: 10.20.40.30
        |
   [Data Diode] (unidirectional)
        |
[Operations: 10.20.30.0/24]
- SCADA: 10.20.30.50
- EWS: 10.20.30.100
        |
   [Firewall 2] (DPI)
        |
[Control: 10.20.10.0/24]
- PLC1: 10.20.10.10
- PLC2: 10.20.10.11
- PLC3: 10.20.10.12
```

Step 2: Configure VLANs

```
# Cisco switch configuration
enable
configure terminal

# Create VLANs
vlan 10
 name OT_Control
vlan 30
 name OT_Operations
vlan 40
 name Industrial_DMZ
```

```
vlan 100
 name Corporate_IT

# Assign ports to VLANs
interface range GigabitEthernet1/0/1-3
 switchport mode access
 switchport access vlan 10
 description PLCs

interface GigabitEthernet1/0/10
 switchport mode access
 switchport access vlan 30
 description SCADA_Server

interface GigabitEthernet1/0/20
 switchport mode access
 switchport access vlan 40
 description Historian_DMZ

# Trunk to firewall
interface GigabitEthernet1/0/48
 switchport mode trunk
 switchport trunk allowed vlan 10,30,40,100
 description Trunk_to_Firewall
```

Step 3: Configure Industrial Firewall

Use pfSense or Palo Alto:

```
# pfSense firewall rules (Operations -> Control)

# Allow SCADA to read from PLCs (Modbus read only)
pass in on OT_OPERATIONS proto tcp from 10.20.30.50 to 10.20.10.0/24 port 502 \
  modbus_filter(allow_read=true, allow_write=false) \
  label "SCADA_READ_PLCs"

# Allow EWS to program PLCs (S7comm)
pass in on OT_OPERATIONS proto tcp from 10.20.30.100 to 10.20.10.0/24 port 102 \
  schedule "BUSINESS_HOURS" \
  label "EWS_PROGRAM_PLCs"

# Block all other traffic to PLCs
block in on OT_OPERATIONS to 10.20.10.0/24 label "DEFAULT_DENY_PLCs"

# Block internet from OT networks
block out on WAN from {10.20.10.0/24, 10.20.30.0/24} to any label
"BLOCK_OT_INTERNET"
```

Step 4: Deploy Data Diode

Install Python scripts from Section 3.3 on gateway appliance:

```
# On OT-side gateway (10.20.30.200)
sudo apt install python3 python3-pip
pip3 install python-snap7 opcua

# Run diode proxy
python3 data_diode_proxy.py &

# On IT-side receiver (10.20.40.10)
python3 diode_receiver.py &
```

Step 5: Configure Secure Remote Access

```
# Install and configure OpenVPN server on DMZ gateway

apt install openvpn easy-rsa

# Generate certificates
make-cadir ~/openvpn-ca
cd ~/openvpn-ca
./easyrsa init-pki
./easyrsa build-ca
./easyrsa build-server-full server nopass
./easyrsa build-client-full vendor-engineer nopass

# Configure server
cat > /etc/openvpn/server.conf <<EOF
port 1194
proto udp
dev tun
ca /etc/openvpn/ca.crt
cert /etc/openvpn/server.crt
key /etc/openvpn/server.key
dh /etc/openvpn/dh.pem
server 10.20.99.0 255.255.255.0
push "route 10.20.30.0 255.255.255.0"  # Operations network
push "route 10.20.10.0 255.255.255.0"  # Control network
client-to-client
keepalive 10 120
cipher AES-256-CBC
auth SHA256
user nobody
group nogroup
persist-key
persist-tun
```

```
status /var/log/openvpn-status.log
log-append /var/log/openvpn.log
verb 3
# Require MFA (via Duo proxy)
auth-user-pass-verify /usr/local/bin/duo_openvpn.py via-env
EOF

systemctl start openvpn@server
```

Step 6: Test and Validate

```
# validation_tests.py
# Verify network segmentation is working correctly

import socket
import subprocess
import sys

def test_connectivity(src_desc, dst_ip, dst_port, should_succeed):
    """Test if connection succeeds/fails as expected"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(2)
        result = sock.connect_ex((dst_ip, dst_port))
        sock.close()

        success = (result == 0)
        expected = should_succeed

        status = "PASS" if (success == expected) else "FAIL"
        print(f"[{status}] {src_desc} -> {dst_ip}:{dst_port} | "
              f"Expected: {'Success' if expected else 'Block'}, Got: {'Success' if success else
'Block'}")

        return status == "PASS"
    except Exception as e:
        print(f"[ERROR] {src_desc} -> {dst_ip}:{dst_port} | {e}")
        return False

def run_validation():
    """Run full validation suite"""
    tests = [
        # Test 1: SCADA should reach PLCs on Modbus
        ("SCADA to PLC1 Modbus", "10.20.10.10", 502, True),

        # Test 2: SCADA should reach PLCs on S7comm
        ("SCADA to PLC1 S7comm", "10.20.10.10", 102, True),
```

```
        # Test 3: PLCs should NOT reach internet
        ("PLC1 to Internet", "8.8.8.8", 80, False),

        # Test 4: PLCs should NOT reach corporate IT
        ("PLC1 to Corporate IT", "10.100.10.10", 445, False),

        # Test 5: Historian should receive data (one-way)
        ("Operations to Historian", "10.20.40.10", 5000, True),

        # Test 6: Historian should NOT send to OT (data diode blocks)
        ("Historian to SCADA", "10.20.30.50", 445, False),
    ]

    passed = 0
    failed = 0

    print("Running network segmentation validation tests...\n")

    for test in tests:
        if test_connectivity(*test):
            passed += 1
        else:
            failed += 1

    print(f"\nResults: {passed} passed, {failed} failed")

    if failed == 0:
        print("Network segmentation validation SUCCESSFUL")
        return 0
    else:
        print("Network segmentation validation FAILED - review firewall rules")
        return 1

if __name__ == '__main__':
    sys.exit(run_validation())
```

Lab Deliverables

1. Network architecture diagram (Purdue Model)
2. VLAN configuration file
3. Firewall ruleset with justifications
4. Data diode configuration (OT and IT sides)
5. VPN configuration with MFA
6. Validation test results

7. Real-World Case Studies

7.1 Ukraine Power Grid Attack (2015)

Attack Vector: Spear-phishing led to compromise of corporate IT network

Lateral Movement: Attackers pivoted from corporate network to OT network due to **lack of segmentation**

Impact: 225,000 customers lost power for 6 hours

Segmentation Failures:

- No DMZ between IT and OT
- Flat network topology
- VPN access directly to SCADA network
- No unidirectional gateways

Lessons:

- Segmentation would have contained breach to IT network
- Data diode would have prevented IT-to-OT lateral movement
- Industrial firewall with DPI could have detected malicious S7comm traffic

7.2 Triton/Trisis Malware (2017)

Attack: Nation-state malware targeting Schneider Electric Triconex safety systems

Network Position: Attackers reached safety PLCs from engineering workstation

Segmentation Issue: Engineering workstation had access to both corporate network (email/internet) and safety PLCs

Proper Segmentation Would Have:

- Isolated safety PLCs on separate physical network
- Required jump box for engineering access (would have detected malware)
- Prevented workstation from having simultaneous IT and OT connectivity

7.3 Colonial Pipeline Ransomware (2021)

Attack: DarkSide ransomware via VPN credential compromise

Segmentation Failure: VPN credentials provided access to OT network

Impact: 5,500-mile pipeline shut down for 6 days, fuel shortages across US East Coast

Defense: Proper DMZ with jump boxes would have:

- Limited VPN access to DMZ only, not directly to OT
- Enabled session recording to detect malicious activity
- Provided kill switch to disconnect remote access during incident

8. Tools and Resources

Network Segmentation Design

- **CSET (Cyber Security Evaluation Tool)**: Free tool from CISA for assessing ICS network architecture
- **ISA Secure System Security Assurance**: Templates for 62443 implementation
- **Purdue Model Templates**: Available from Rockwell Automation, Siemens

Industrial Firewalls

- **Palo Alto PA-Series**: Deep packet inspection for ICS protocols
- **Fortinet FortiGate**: ICS-focused firewall with OT protocol parsers
- **Cisco Firepower**: Industrial security with Snort integration
- **Tofino Xenon**: Specialized ICS firewall (Hirschmann/Belden)
- **Open-Source**: pfSense with custom rules, Snort/Suricata with ICS plugins

Unidirectional Gateways

- **Waterfall Security Solutions**: Unidirectional CloudConnect
- **Owl Cyber Defense**: OPDS (Optical Data Diode)
- **BAE Systems**: Unidirectional Gateway
- **DIY Data Diode**: Raspberry Pi-based educational projects

Remote Access Solutions

- **CyberArk Privileged Access Security**: For jump box session management
- **BeyondTrust**: Privileged remote access for OT
- **Dispel**: Secure OT remote access platform
- **Claroty Secure Remote Access**: Purpose-built for industrial environments

Standards and Guidelines

- **ISA/IEC 62443**: Industrial security standards (zones and conduits)
- **NIST SP 800-82**: Guide to Industrial Control Systems Security
- **NERC CIP**: Critical Infrastructure Protection standards (power sector)
- **NIS Directive**: Network and Information Systems Directive (EU)

Conclusion

Network segmentation is the foundation of OT cybersecurity defense. By implementing:

- Purdue Model architecture with clearly defined zones
- ISA/IEC 62443 security levels and conduit enforcement
- Unidirectional gateways to prevent IT-to-OT lateral movement
- Industrial firewalls with deep packet inspection
- Secure remote access with defense-in-depth

Organizations can significantly reduce the attack surface and contain incidents when they occur. In the next lesson, we'll build on this foundation by deploying intrusion detection systems to monitor traffic between these segmented zones.

Lesson 02: IDS/IPS for Industrial Protocols

Lesson 02: IDS/IPS for Industrial Protocols

Learning Objectives

- Deploy intrusion detection systems tailored for OT networks
- Write custom detection rules for Modbus, S7comm, DNP3, and other industrial protocols
- Configure Zeek with ICSNPP plugins for deep protocol analysis
- Implement Snort/Suricata with ICS-specific rulesets
- Perform protocol whitelisting and anomaly detection
- Tune IDS to minimize false positives in operational environments
- Detect attacks from Module 2 (reconnaissance, MITM, malicious writes, C2 beaconing)

Introduction

Traditional IT-focused IDS systems like Snort and Suricata are designed to detect attacks on HTTP, DNS, SSH, and other IT protocols. However, OT environments use industrial protocols (Modbus, S7comm, DNP3, EtherNet/IP) that standard IDS cannot parse or understand.

Key Differences for OT IDS:

- Must understand industrial protocol semantics (function codes, register addresses)
- Extremely low false positive tolerance (false alerts can trigger unnecessary shutdowns)
- Protocol whitelisting is more effective than signature-based detection
- Behavioral analysis and anomaly detection are critical
- Must operate passively (no inline blocking that could disrupt processes)

This lesson directly addresses attacks from Module 2:

- **Reconnaissance** (Module 2 Lesson 01): Detect port scans, Modbus reads, S7comm enumeration
- **MITM attacks** (Module 2 Lesson 03): Detect ARP spoofing, unexpected traffic patterns
- **PLC manipulation** (Module 2 Lesson 04): Detect unauthorized writes, program downloads
- **C2 beaconing** (Module 2 Lesson 09): Detect periodic outbound connections, DNS tunneling

1. ICS IDS Platforms

1.1 Commercial ICS IDS Solutions

Platform	Strengths	Protocols Supported	Pricing
Nozomi Networks Guardian	Deep packet inspection, asset discovery, threat intelligence	Modbus, S7, DNP3, EtherNet/IP, OPC UA, IEC 104, BACnet, Profinet	Enterprise ($50K+)
Claroty Continuous Threat Detection	Integration with vulnerability management, anomaly detection	100+ protocols including proprietary	Enterprise ($40K+)
Dragos Platform	Threat intelligence from Dragos WorldView, industrial-specific threat hunting	Modbus, DNP3, S7, EtherNet/IP, OPC DA/UA	Enterprise ($60K+)
Cyberbit	Attack simulation, automated playbooks	Modbus, DNP3, S7, IEC 61850	Enterprise
Armis	Agentless asset discovery, passive monitoring	Multi-protocol support	Mid-market ($30K+)

1.2 Open-Source ICS IDS Solutions

Zeek (formerly Bro) with ICSNPP Plugins:

- Developed by CISA (US Cybersecurity and Infrastructure Security Agency)
- Protocol parsers for Modbus, DNP3, EtherNet/IP, Profinet, BACnet, S7comm
- Scriptable detection logic in Zeek language
- **Best for**: Large-scale deployments, custom detection, research

Suricata with Quickdraw ICS Rules:

- High-performance IDS with multi-threading support
- ET Open ICS rules, Quickdraw rules from Digital Bond
- Lua scripting for complex detection
- **Best for**: Inline IPS deployments (with caution), high-throughput networks

Snort with Digital Bond Rules:

- Classic signature-based IDS
- Digital Bond Quickdraw ICS rules (legacy, not updated since 2016)
- Preprocessors for Modbus, DNP3

- **Best for**: Legacy deployments, simple signature-based detection

2. Zeek with ICSNPP Plugins

2.1 Installation and Configuration

```
#!/bin/bash
# install_zeek_ics.sh
# Install Zeek with ICSNPP plugins for industrial protocol analysis

# Step 1: Install Zeek
sudo apt update
sudo apt install -y cmake make gcc g++ flex bison libpcap-dev libssl-dev python3
python3-dev swig zlib1g-dev

# Install from source for latest version
cd /opt
wget https://download.zeek.org/zeek-6.0.0.tar.gz
tar -xzf zeek-6.0.0.tar.gz
cd zeek-6.0.0
./configure --prefix=/opt/zeek
make -j$(nproc)
sudo make install

# Add to PATH
echo 'export PATH=/opt/zeek/bin:$PATH' >> ~/.bashrc
source ~/.bashrc

# Step 2: Install Zeek Package Manager
pip3 install zkg

# Step 3: Configure zkg
zkg autoconfig

# Step 4: Install ICSNPP plugins
# Modbus
zkg install icsnpp-modbus
# DNP3
zkg install icsnpp-dnp3
# EtherNet/IP (CIP)
zkg install icsnpp-enip
# S7comm (Siemens)
zkg install icsnpp-s7comm
# BACnet
zkg install icsnpp-bacnet
# Profinet
zkg install icsnpp-profinet
```

```
# IEC 60870-5-104
zkg install icsnpp-iec104

# Step 5: Verify installation
zeek --version
zkg list

echo "Zeek with ICSNPP plugins installed successfully"
```

2.2 Basic Zeek Configuration for OT Network

```
# /opt/zeek/etc/node.cfg
# Configure Zeek for passive monitoring on OT network

[logger]
type=logger
host=localhost

[manager]
type=manager
host=localhost

[proxy-1]
type=proxy
host=localhost

[worker-1]
type=worker
host=localhost
interface=eth1  # OT network interface
lb_method=custom
lb_procs=4  # Use 4 CPU cores
# /opt/zeek/share/zeek/site/local.zeek
# Load ICS plugins and custom detection scripts

# Load ICSNPP plugins
@load icsnpp/modbus
@load icsnpp/dnp3
@load icsnpp/enip
@load icsnpp/s7comm
@load icsnpp/bacnet
@load icsnpp/iec104

# Load custom detection scripts
@load ./detect_unauthorized_modbus_writes.zeek
@load ./detect_plc_reconnaissance.zeek
@load ./detect_c2_beaconing.zeek
@load ./detect_arp_spoofing.zeek
```

```
# Enable JSON logging for SIEM integration
@load policy/tuning/json-logs.zeek

# Configure logging
redef LogAscii::use_json = T;
redef LogAscii::json_timestamps = JSON::TS_ISO8601;
```

2.3 Custom Zeek Detection Scripts

Detect Unauthorized Modbus Writes:

```
# detect_unauthorized_modbus_writes.zeek
# Alert on Modbus write operations from unauthorized sources

@load base/frameworks/notice

module ModbusSecurity;

export {
    redef enum Notice::Type += {
        UnauthorizedModbusWrite,
        ExcessiveModbusWrites,
        ModbusDiagnosticFunction
    };

    # Whitelist of authorized sources for Modbus writes
    global authorized_writers: set[addr] = {
        10.20.30.50,   # SCADA server
        10.20.30.100   # Engineering workstation
    } &redef;

    # Whitelist of allowed destination PLCs
    global plc_network: subnet = 10.20.10.0/24 &redef;

    # Track write counts per source IP
    global write_counts: table[addr] of count &create_expire=1hr &default=0;
}

# Monitor Modbus write functions
event modbus_message(c: connection, headers: ModbusHeaders, is_orig: bool) &priority=5
{

    # Only process requests (from client to server)
    if (!is_orig)
        return;

    local src_ip = c$id$orig_h;
```

```
    local dst_ip = c$id$resp_h;
    local func_code = headers$function_code;

    # Check if destination is a PLC
    if (dst_ip !in plc_network)
        return;

    # Detect write function codes
    # FC 5: Write Single Coil
    # FC 6: Write Single Register
    # FC 15: Write Multiple Coils
    # FC 16: Write Multiple Registers
    # FC 23: Read/Write Multiple Registers
    if (func_code in [5, 6, 15, 16, 23]) {

        # Check if source is authorized
        if (src_ip !in authorized_writers) {
            NOTICE([$note=UnauthorizedModbusWrite,
                    $msg=fmt("Unauthorized Modbus write from %s to PLC %s (FC: %d)", src_ip,
dst_ip, func_code),
                    $conn=c,
                    $identifier=cat(src_ip, dst_ip),
                    $suppress_for=5min]);
        }

        # Track write frequency (detect excessive writes)
        ++write_counts[src_ip];

        if (write_counts[src_ip] > 100) {
            NOTICE([$note=ExcessiveModbusWrites,
                    $msg=fmt("Excessive Modbus writes from %s: %d writes in last hour", src_ip,
write_counts[src_ip]),
                    $conn=c,
                    $identifier=cat(src_ip),
                    $suppress_for=1hr]);
        }
    }

    # Detect diagnostic function (used in reconnaissance)
    if (func_code == 8) {
        NOTICE([$note=ModbusDiagnosticFunction,
                $msg=fmt("Modbus diagnostic function from %s to %s (recon activity?)", src_ip,
dst_ip),
                $conn=c,
                $identifier=cat(src_ip, dst_ip),
                $suppress_for=10min]);
    }
}
```

Detect PLC Reconnaissance:

```
# detect_plc_reconnaissance.zeek
# Detect reconnaissance activities targeting PLCs

@load base/frameworks/notice
@load base/frameworks/sumstats

module PLCRecon;

export {
    redef enum Notice::Type += {
        PLCPortScan,
        ModbusFunctionCodeScan,
        ExcessiveModbusReads
    };

    global plc_network: subnet = 10.20.10.0/24 &redef;
}

# Detect port scanning targeting PLCs
event SumStats::finish(ss: SumStats::SumStat, key: SumStats::Key, data: SumStats::Result)
{
    if (ss$name == "plc_port_scan") {
        local scanner = key$host;
        local port_count = data["port_scan"]$num;

        if (port_count > 5) {
            NOTICE([$note=PLCPortScan,
                    $msg=fmt("Port scan detected from %s: %d ports scanned on PLC network",
scanner, port_count),
                    $src=scanner,
                    $suppress_for=1hr]);
        }
    }
}

# Track connection attempts to multiple ports
event connection_attempt(c: connection) {
    local dst_ip = c$id$resp_h;

    # Only monitor PLC network
    if (dst_ip !in plc_network)
        return;

    SumStats::observe("plc_port_scan",
                      [$host=c$id$orig_h],
                      [$str=cat(c$id$resp_p)]);
```

```
}

# Configure SumStats
event zeek_init() {
    SumStats::create([
        $name="plc_port_scan",
        $epoch=5min,
        $reducers=set(SumStats::UNIQUE),
        $threshold=5.0,
        $threshold_val(key: SumStats::Key, result: SumStats::Result) = {
            return result["port_scan"]$num;
        },
        $threshold_crossed(key: SumStats::Key, result: SumStats::Result) = {
            # Will trigger SumStats::finish event
        }
    ]);
}

# Detect excessive Modbus reads (reconnaissance)
global read_counts: table[addr] of count &create_expire=10min &default=0;

event modbus_message(c: connection, headers: ModbusHeaders, is_orig: bool) {
    if (!is_orig)
        return;

    local src_ip = c$id$orig_h;
    local dst_ip = c$id$resp_h;
    local func_code = headers$function_code;

    if (dst_ip !in plc_network)
        return;

    # Detect read function codes
    # FC 1: Read Coils
    # FC 2: Read Discrete Inputs
    # FC 3: Read Holding Registers
    # FC 4: Read Input Registers
    if (func_code in [1, 2, 3, 4]) {
        ++read_counts[src_ip];

        # Alert if >100 reads in 10 minutes (likely reconnaissance)
        if (read_counts[src_ip] > 100) {
            NOTICE([$note=ExcessiveModbusReads,
                    $msg=fmt("Excessive Modbus reads from %s: %d reads in 10 minutes (possible
recon)", src_ip, read_counts[src_ip]),
                    $conn=c,
                    $src=src_ip,
                    $suppress_for=30min]);
```

```
            # Reset counter
            read_counts[src_ip] = 0;
        }
    }
}
```

Detect C2 Beaconing:

```
# detect_c2_beaconing.zeek
# Detect periodic outbound connections (C2 beaconing)

@load base/frameworks/notice

module C2Detection;

export {
    redef enum Notice::Type += {
        PeriodicBeaconing,
        DNSTunneling,
        UnusualDNSQuery
    };

    # Track connection times per src/dst pair
    global conn_times: table[addr, addr, port] of vector of time &create_expire=24hr;
}

event connection_state_remove(c: connection) {
    local src = c$id$orig_h;
    local dst = c$id$resp_h;
    local dport = c$id$resp_p;

    # Only monitor outbound connections from OT network
    if (!Site::is_local_addr(src) || Site::is_local_addr(dst))
        return;

    # Record connection time
    local key = [src, dst, dport];
    if (key !in conn_times)
        conn_times[key] = vector();

    conn_times[key][|conn_times[key]|] = network_time();

    # Analyze if we have enough data points (at least 5 connections)
    if (|conn_times[key]| >= 5) {
        local times = conn_times[key];
        local intervals: vector of interval;
```

```
        # Calculate intervals between connections
        for (i in times) {
            if (i > 0) {
                intervals[|intervals|] = times[i] - times[i-1];
            }
        }

        # Check for periodic pattern (beaconing)
        if (|intervals| >= 4) {
            local avg_interval = 0.0sec;
            for (interval in intervals) {
                avg_interval += intervals[interval];
            }
            avg_interval = avg_interval / |intervals|;

            # Calculate standard deviation
            local variance = 0.0;
            for (interval in intervals) {
                local diff = intervals[interval] - avg_interval;
                variance += diff * diff;
            }
            local stddev = sqrt(variance / |intervals|);

            # If stddev is low, connections are periodic (likely beaconing)
            if (stddev < 30.0sec && avg_interval > 1min) {
                NOTICE([$note=PeriodicBeaconing,
                        $msg=fmt("Periodic beaconing detected from %s to %s:%d (avg interval: %s,
stddev: %s)",
                                 src, dst, dport, avg_interval, stddev),
                        $src=src,
                        $identifier=cat(src, dst, dport),
                        $suppress_for=6hr]);
            }
        }
    }
}

# Detect DNS tunneling
event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count) {
    # Check for unusually long DNS queries (potential tunneling)
    if (|query| > 50) {
        NOTICE([$note=DNSTunneling,
                $msg=fmt("Potential DNS tunneling: long query from %s: %s", c$id$orig_h, query),
                $conn=c,
                $suppress_for=1hr]);
    }

    # Check for high entropy in subdomain (random-looking)
```

```
    if (has_high_entropy(query)) {
        NOTICE([$note=UnusualDNSQuery,
                $msg=fmt("High-entropy DNS query from %s: %s (possible C2)", c$id$orig_h,
query),
                $conn=c,
                $suppress_for=30min]);
    }
}

function has_high_entropy(s: string): bool {
    # Simple entropy check (production should use proper Shannon entropy)
    local unique_chars: set[string];
    for (i in s) {
        add unique_chars[s[i]];
    }
    return |unique_chars| > (|s| * 0.7);  # >70% unique characters
}
```

Detect ARP Spoofing (MITM):

```
# detect_arp_spoofing.zeek
# Detect ARP spoofing attacks (MITM between SCADA and PLCs)

@load base/frameworks/notice

module ARPSecurity;

export {
    redef enum Notice::Type += {
        ARPSpoofing,
        DuplicateMAC
    };

    # Track IP-to-MAC mappings
    global arp_table: table[addr] of string &create_expire=24hr;
}

event arp_request(mac_src: string, mac_dst: string, SPA: addr, SHA: string, TPA: addr, THA:
string) {
    check_arp_mapping(SPA, SHA);
}

event arp_reply(mac_src: string, mac_dst: string, SPA: addr, SHA: string, TPA: addr, THA:
string) {
    check_arp_mapping(SPA, SHA);
}

function check_arp_mapping(ip: addr, mac: string) {
```

```
    if (ip in arp_table) {
        # Check if MAC has changed
        if (arp_table[ip] != mac) {
            NOTICE([$note=ARPSpoofing,
                    $msg=fmt("ARP spoofing detected: IP %s changed from MAC %s to %s", ip,
arp_table[ip], mac),
                    $src=ip,
                    $suppress_for=10min]);
        }
    } else {
        # New entry
        arp_table[ip] = mac;
    }
}
```

2.4 Running Zeek and Analyzing Logs

```
#!/bin/bash
# run_zeek_ot_monitoring.sh

# Start Zeek in live capture mode
zeekctl deploy

# Monitor logs in real-time
tail -f /opt/zeek/logs/current/notice.log | jq '.'

# View Modbus traffic
tail -f /opt/zeek/logs/current/modbus.log | jq '.'

# View S7comm traffic
tail -f /opt/zeek/logs/current/s7comm.log | jq '.'

# View DNP3 traffic
tail -f /opt/zeek/logs/current/dnp3.log | jq '.'
```

Example Zeek Notice Log (JSON):


```
{
  "ts": "2025-01-03T14:23:45.123456Z",
  "uid": "CHhAvVGS1DHFjwGM9",
  "id.orig_h": "10.20.30.99",
  "id.orig_p": 54321,
  "id.resp_h": "10.20.10.10",
  "id.resp_p": 502,
  "proto": "tcp",
  "note": "ModbusSecurity::UnauthorizedModbusWrite",
  "msg": "Unauthorized Modbus write from 10.20.30.99 to PLC 10.20.10.10 (FC: 16)",
  "sub": "Write Multiple Registers",
  "src": "10.20.30.99",
```

```
  "dst": "10.20.10.10",
  "p": 502,
  "actions": ["Notice::ACTION_LOG"],
  "suppress_for": 300.0
}
```

3. Snort/Suricata Rules for ICS Protocols

3.1 Install Suricata with ICS Rules

```
#!/bin/bash
# install_suricata_ics.sh

# Install Suricata
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt update
sudo apt install -y suricata jq

# Download Emerging Threats ICS rules
sudo suricata-update enable-source et/open
sudo suricata-update enable-source oisf/trafficid
sudo suricata-update

# Download Digital Bond Quickdraw ICS rules (legacy but useful)
cd /tmp
wget https://github.com/digitalbond/Quickdraw-Snort/archive/master.zip
unzip master.zip
sudo cp Quickdraw-Snort-master/*.rules /etc/suricata/rules/

# Enable ICS rules in suricata.yaml
sudo sed -i 's/# - modbus.rules/- modbus.rules/g' /etc/suricata/suricata.yaml
sudo sed -i 's/# - dnp3.rules/- dnp3.rules/g' /etc/suricata/suricata.yaml

# Configure network interface
sudo sed -i 's/interface: eth0/interface: eth1/g' /etc/suricata/suricata.yaml  # OT interface

# Start Suricata
sudo systemctl start suricata
sudo systemctl enable suricata

# Verify
sudo suricata --build-info
```

3.2 Custom Snort/Suricata Rules for ICS

Modbus Rules:

```
# modbus_custom.rules
```

```
# Custom Snort/Suricata rules for Modbus traffic

# Detect Modbus write from unauthorized source
alert tcp !$SCADA_SERVERS any -> $PLC_NETWORK 502 (
    msg:"SCADA-CUSTOM Unauthorized Modbus Write";
    flow:to_server,established;
    content:"|06|"; offset:7; depth:1;  # Function Code 6: Write Single Register
    classtype:policy-violation;
    sid:9000001; rev:1;
)

alert tcp !$SCADA_SERVERS any -> $PLC_NETWORK 502 (
    msg:"SCADA-CUSTOM Unauthorized Modbus Write Multiple Registers";
    flow:to_server,established;
    content:"|10|"; offset:7; depth:1;  # Function Code 16: Write Multiple Registers
    classtype:policy-violation;
    sid:9000002; rev:1;
)

# Detect Modbus diagnostic functions (recon)
alert tcp any any -> $PLC_NETWORK 502 (
    msg:"SCADA-RECON Modbus Diagnostic Function";
    flow:to_server,established;
    content:"|08|"; offset:7; depth:1;  # Function Code 8: Diagnostics
    classtype:attempted-recon;
    sid:9000003; rev:1;
)

# Detect excessive register read (reconnaissance)
alert tcp any any -> $PLC_NETWORK 502 (
    msg:"SCADA-RECON Modbus Large Register Read";
    flow:to_server,established;
    content:"|03|"; offset:7; depth:1;  # Function Code 3: Read Holding Registers
    byte_test:2,>,100,10;  # Quantity > 100 registers
    classtype:attempted-recon;
    sid:9000004; rev:1;
)

# Detect Modbus exception responses (errors could indicate attack)
alert tcp $PLC_NETWORK 502 -> any any (
    msg:"SCADA-ANOMALY Modbus Exception Response";
    flow:to_client,established;
    content:"|81|"; offset:7; depth:1;  # Exception for FC 1
    classtype:protocol-command-decode;
    sid:9000005; rev:1;
)

# Detect Modbus to non-standard port (covert channel)
```

```
alert tcp any any -> $PLC_NETWORK !502 (
    msg:"SCADA-ANOMALY Modbus Traffic on Non-Standard Port";
    flow:to_server,established;
    content:"|00 00|"; depth:2;  # Modbus transaction ID
    content:"|00 00|"; offset:2; depth:2;  # Modbus protocol ID
    classtype:protocol-command-decode;
    sid:9000006; rev:1;
)
```

S7comm Rules:

```
# s7comm_custom.rules
# Detect Siemens S7comm attacks

# Detect PLC STOP command
alert tcp any any -> $PLC_NETWORK 102 (
    msg:"SCADA-ATTACK Siemens PLC STOP Command";
    flow:to_server,established;
    content:"|03 00|"; depth:2;  # TPKT version 3
    content:"|32|"; distance:5; within:1;  # COTP type
    content:"|29|"; distance:12; within:1;  # S7 Function: STOP
    classtype:attempted-dos;
    sid:9000101; rev:1;
)

# Detect program download to PLC
alert tcp any any -> $PLC_NETWORK 102 (
    msg:"SCADA-SUSPICIOUS S7 Program Download to PLC";
    flow:to_server,established;
    content:"|03 00|"; depth:2;
    content:"|1A|"; distance:17; within:1;  # Function: Download block
    classtype:policy-violation;
    sid:9000102; rev:1;
)

# Detect program upload from PLC (data exfiltration)
alert tcp any any -> $PLC_NETWORK 102 (
    msg:"SCADA-EXFIL S7 Program Upload from PLC";
    flow:to_server,established;
    content:"|03 00|"; depth:2;
    content:"|1D|"; distance:17; within:1;  # Function: Start Upload
    classtype:policy-violation;
    sid:9000103; rev:1;
)

# Detect S7 authentication bypass (blank password)
alert tcp any any -> $PLC_NETWORK 102 (
    msg:"SCADA-AUTH S7 Authentication with Blank Password";
```

```
    flow:to_server,established;
    content:"|03 00|"; depth:2;
    content:"|F0|"; distance:5; within:1;  # Setup communication
    content:"|00 00 00 00|"; within:4;  # Blank password
    classtype:attempted-admin;
    sid:9000104; rev:1;
)

# Detect S7 from non-engineering workstation
alert tcp !$EWS_NETWORK any -> $PLC_NETWORK 102 (
    msg:"SCADA-POLICY S7comm from Unauthorized Source";
    flow:to_server,established;
    content:"|03 00|"; depth:2;
    classtype:policy-violation;
    threshold:type limit, track by_src, count 1, seconds 300;
    sid:9000105; rev:1;
)
```

DNP3 Rules:

```
# dnp3_custom.rules
# Detect DNP3 attacks (SCADA in power/water utilities)

# Detect DNP3 Direct Operate (bypasses SELECT before OPERATE)
alert tcp any any -> $SCADA_NETWORK 20000 (
    msg:"SCADA-ATTACK DNP3 Direct Operate Command";
    flow:to_server,established;
    content:"|05 64|"; depth:2;  # DNP3 start bytes
    content:"|05|"; distance:10; within:1;  # Function Code 5: Direct Operate
    classtype:attempted-admin;
    sid:9000201; rev:1;
)

# Detect DNP3 cold restart
alert tcp any any -> $SCADA_NETWORK 20000 (
    msg:"SCADA-DOS DNP3 Cold Restart Command";
    flow:to_server,established;
    content:"|05 64|"; depth:2;
    content:"|0D|"; distance:10; within:1;  # Function Code 13: Cold Restart
    classtype:attempted-dos;
    sid:9000202; rev:1;
)

# Detect DNP3 write (unauthorized configuration change)
alert tcp any any -> $SCADA_NETWORK 20000 (
    msg:"SCADA-CONFIG DNP3 Write Command";
    flow:to_server,established;
    content:"|05 64|"; depth:2;
```

```
    content:"|02|"; distance:10; within:1;  # Function Code 2: Write
    classtype:policy-violation;
    sid:9000203; rev:1;
)

# Detect DNP3 file transfer (potential malware delivery)
alert tcp any any -> $SCADA_NETWORK 20000 (
    msg:"SCADA-MALWARE DNP3 File Transfer";
    flow:to_server,established;
    content:"|05 64|"; depth:2;
    content:"|15|"; distance:10; within:1;  # Function Code 21: File Transport
    classtype:trojan-activity;
    sid:9000204; rev:1;
)
```

3.3 Suricata Configuration

```
# /etc/suricata/suricata.yaml (relevant sections)

vars:
  address-groups:
    PLC_NETWORK: "[10.20.10.0/24]"
    SCADA_SERVERS: "[10.20.30.50/32]"
    EWS_NETWORK: "[10.20.30.100/32]"
    SCADA_NETWORK: "[10.20.0.0/16]"

# Enable ICS protocol parsers
app-layer:
  protocols:
    modbus:
      enabled: yes
      detection-ports:
        dp: 502
    dnp3:
      enabled: yes
      detection-ports:
        dp: 20000

# Logging
outputs:
  - fast:
      enabled: yes
      filename: fast.log
  - eve-log:
      enabled: yes
      filetype: regular
      filename: eve.json
      types:
```

```
        - alert:
            payload: yes
            payload-buffer-size: 4kb
        - modbus:
            enabled: yes
        - dnp3:
            enabled: yes

# Rule files
default-rule-path: /etc/suricata/rules
rule-files:
  - suricata.rules
  - modbus_custom.rules
  - s7comm_custom.rules
  - dnp3_custom.rules
```

4. Protocol Whitelisting and Baseline

4.1 Building a Baseline

```
# build_ot_baseline.py
# Baseline normal OT network traffic to create whitelist

import pyshark
from collections import defaultdict
import json

class OTBaseline:
    def __init__(self, pcap_file):
        self.pcap_file = pcap_file
        self.legitimate_flows = defaultdict(int)
        self.protocol_distribution = defaultdict(int)
        self.function_codes = defaultdict(lambda: defaultdict(int))

    def analyze_pcap(self):
        """Analyze PCAP to build baseline"""
        cap = pyshark.FileCapture(self.pcap_file, display_filter='tcp')

        for packet in cap:
            try:
                src_ip = packet.ip.src
                dst_ip = packet.ip.dst
                dst_port = packet.tcp.dstport

                # Record legitimate flow
                flow_key = f"{src_ip}->{dst_ip}:{dst_port}"
                self.legitimate_flows[flow_key] += 1
```

```
                # Track protocols
                if dst_port == '502':  # Modbus
                    self.protocol_distribution['Modbus'] += 1

                    # Extract Modbus function code if available
                    if hasattr(packet, 'modbus'):
                        func_code = packet.modbus.func_code
                        self.function_codes['Modbus'][func_code] += 1

                elif dst_port == '102':  # S7comm
                    self.protocol_distribution['S7comm'] += 1

                elif dst_port == '20000':  # DNP3
                    self.protocol_distribution['DNP3'] += 1

                elif dst_port == '44818':  # EtherNet/IP
                    self.protocol_distribution['EtherNet/IP'] += 1

            except AttributeError:
                continue

        cap.close()

    def export_whitelist(self, output_file='ot_whitelist.json'):
        """Export whitelist for enforcement"""
        whitelist = {
            'flows': dict(self.legitimate_flows),
            'protocols': dict(self.protocol_distribution),
            'modbus_function_codes': dict(self.function_codes['Modbus'])
        }

        with open(output_file, 'w') as f:
            json.dump(whitelist, f, indent=2)

        print(f"[+] Whitelist exported to {output_file}")
        print(f"[+] Legitimate flows: {len(self.legitimate_flows)}")
        print(f"[+] Protocol distribution: {dict(self.protocol_distribution)}")

    def generate_zeek_whitelist(self):
        """Generate Zeek script to enforce whitelist"""
        zeek_script = """
# whitelist_enforcement.zeek
# Auto-generated whitelist from baseline analysis

@load base/frameworks/notice

module WhitelistEnforcement;
```

```
export {
    redef enum Notice::Type += {
        UnauthorizedFlow
    };

    global legitimate_flows: set[addr, addr, port] = {
"""
        # Add flows
        for flow_key in self.legitimate_flows:
            parts = flow_key.split('->')
            src = parts[0]
            dst_parts = parts[1].split(':')
            dst = dst_parts[0]
            port = dst_parts[1]
            zeek_script += f"        [{src}, {dst}, {port}/tcp],\n"

        zeek_script += """    } &redef;
}

event new_connection(c: connection) {
    local flow = [c$id$orig_h, c$id$resp_h, c$id$resp_p];

    if (flow !in legitimate_flows) {
        NOTICE([$note=UnauthorizedFlow,
                $msg=fmt("New flow not in whitelist: %s -> %s:%d", c$id$orig_h, c$id$resp_h,
c$id$resp_p),
                $conn=c,
                $suppress_for=10min]);
    }
}
"""
        with open('whitelist_enforcement.zeek', 'w') as f:
            f.write(zeek_script)

        print("[+] Zeek whitelist script generated: whitelist_enforcement.zeek")

# Usage
if __name__ == '__main__':
    import sys

    if len(sys.argv) < 2:
        print("Usage: python3 build_ot_baseline.py <pcap_file>")
        sys.exit(1)

    pcap_file = sys.argv[1]
    print(f"[*] Analyzing {pcap_file} to build baseline...")
```

```
    baseline = OTBaseline(pcap_file)
    baseline.analyze_pcap()
    baseline.export_whitelist()
    baseline.generate_zeek_whitelist()

    print("\n[+] Baseline complete. Deploy whitelist_enforcement.zeek to Zeek.")
```

Usage:

```
# Capture 7 days of normal traffic
sudo tcpdump -i eth1 -w ot_baseline.pcap -G 604800 -W 1

# Build baseline
python3 build_ot_baseline.py ot_baseline.pcap

# Deploy to Zeek
sudo cp whitelist_enforcement.zeek /opt/zeek/share/zeek/site/
# Add to local.zeek: @load ./whitelist_enforcement.zeek
sudo zeekctl deploy
```

5. Anomaly Detection and Machine Learning

5.1 Statistical Anomaly Detection

```
# anomaly_detection_ot.py
# Detect statistical anomalies in OT traffic

import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
import json

class OTAnomalyDetector:
    def __init__(self, zeek_log_path):
        self.zeek_log_path = zeek_log_path
        self.model = IsolationForest(contamination=0.01, random_state=42)

    def load_zeek_logs(self):
        """Load Zeek connection logs"""
        # Read JSON logs
        logs = []
        with open(self.zeek_log_path, 'r') as f:
            for line in f:
                logs.append(json.loads(line))

        return pd.DataFrame(logs)

    def extract_features(self, df):
```

```
        """Extract features for anomaly detection"""
        features = pd.DataFrame()

        # Feature 1: Connection duration
        features['duration'] = df['duration'].fillna(0)

        # Feature 2: Bytes transferred
        features['orig_bytes'] = df['orig_bytes'].fillna(0)
        features['resp_bytes'] = df['resp_bytes'].fillna(0)

        # Feature 3: Packet count
        features['orig_pkts'] = df['orig_pkts'].fillna(0)
        features['resp_pkts'] = df['resp_pkts'].fillna(0)

        # Feature 4: Connection state (convert to numeric)
        state_map = {'S0': 0, 'S1': 1, 'SF': 2, 'REJ': 3, 'RSTO': 4}
        features['conn_state'] = df['conn_state'].map(state_map).fillna(-1)

        # Feature 5: Destination port
        features['dst_port'] = df['id.resp_p'].fillna(0)

        return features

    def train(self, df):
        """Train anomaly detection model on normal traffic"""
        features = self.extract_features(df)
        self.model.fit(features)
        print("[+] Model trained on normal traffic")

    def detect_anomalies(self, df):
        """Detect anomalies in new traffic"""
        features = self.extract_features(df)
        predictions = self.model.predict(features)

        # -1 = anomaly, 1 = normal
        anomalies = df[predictions == -1]

        return anomalies

# Usage
if __name__ == '__main__':
    # Load baseline (normal) traffic
    detector = OTAnomalyDetector('/opt/zeek/logs/2025-01-01/conn.log')
    baseline_df = detector.load_zeek_logs()

    # Train model
    detector.train(baseline_df)
```

```
    # Detect anomalies in current traffic
    current_df = detector.load_zeek_logs()  # Load current logs
    anomalies = detector.detect_anomalies(current_df)

    print(f"\n[!] Detected {len(anomalies)} anomalous connections:")
    print(anomalies[['ts', 'id.orig_h', 'id.resp_h', 'id.resp_p', 'duration']])
```

6. Tuning IDS to Minimize False Positives

6.1 Common False Positive Sources in OT

1. **Legitimate Engineering Activities**: Program downloads, PLC reboots
2. **Scheduled Maintenance**: Vendor remote access, firmware updates
3. **Polling Behavior**: SCADA polling PLCs every second
4. **Startup Sequences**: PLCs broadcasting on startup

6.2 Tuning Strategies

Suppress Alerts During Maintenance Windows:

```
# maintenance_window_suppression.zeek

module MaintenanceMode;

export {
    # Define maintenance windows
    global maintenance_windows: vector of interval = {
        [2025-01-10 02:00:00 .. 2025-01-10 06:00:00],  # Planned maintenance
        [2025-01-17 02:00:00 .. 2025-01-17 06:00:00]
    };

    # Suppress specific notice types during maintenance
    global suppressed_notices: set[Notice::Type] = {
        ModbusSecurity::UnauthorizedModbusWrite,
        PLCRecon::PLCPortScan
    };
}

hook Notice::policy(n: Notice::Info) {
    # Check if we're in maintenance window
    for (window in maintenance_windows) {
        if (network_time() >= window$start && network_time() <= window$end) {
            if (n$note in suppressed_notices) {
                # Suppress notice
                add n$actions[Notice::ACTION_NONE];
                break;
            }
```

```
    }
  }
}
```

Whitelist Known-Good Behavior:

```
# suricata: suppress.conf
# Suppress false positives for known-good activity

# Suppress S7 program download from engineering workstation during work hours
suppress gen_id 1, sig_id 9000102, track by_src, ip 10.20.30.100

# Suppress Modbus diagnostic from SCADA monitoring tool
suppress gen_id 1, sig_id 9000003, track by_src, ip 10.20.30.50

# Suppress alerts for vendor remote access (with time limit)
suppress gen_id 1, sig_id 9000105, track by_src, ip 203.0.113.50
```

7. Hands-On Lab: Deploy IDS for OT Network

Lab Objective

Deploy Zeek with ICSNPP plugins and Suricata to monitor a water treatment facility OT network. Detect attacks from Module 2.

Lab Environment

- **OT Network**: 10.20.10.0/24 (PLCs)
- **SCADA Network**: 10.20.30.0/24
- **IDS Sensor**: Ubuntu 22.04 with TAP/SPAN port access
- **Attack Scenarios**: Modbus reconnaissance, unauthorized writes, C2 beaconing

Lab Steps

Step 1: Deploy Zeek

```
# Install Zeek with ICSNPP
bash install_zeek_ics.sh

# Configure for OT network
sudo nano /opt/zeek/etc/node.cfg
# Set interface=eth1 (SPAN port receiving mirrored OT traffic)

# Deploy
sudo zeekctl deploy
```

Step 2: Configure Custom Detection

```
# Copy custom detection scripts
cd /opt/zeek/share/zeek/site
sudo nano detect_unauthorized_modbus_writes.zeek
# Paste detection script from Section 2.3

# Enable in local.zeek
echo "@load ./detect_unauthorized_modbus_writes.zeek" | sudo tee -a local.zeek

# Reload
sudo zeekctl deploy
```

Step 3: Generate Test Traffic

```
# test_modbus_write.py
# Simulate unauthorized Modbus write (should trigger alert)

from pymodbus.client import ModbusTcpClient

client = ModbusTcpClient('10.20.10.10', port=502)
client.connect()

# Write to holding register (will be detected as unauthorized)
client.write_register(100, 9999, unit=1)

client.close()
print("[*] Test write sent to PLC")
```

Step 4: Verify Detection

```
# Check Zeek notices
tail -f /opt/zeek/logs/current/notice.log | jq 'select(.note ==
"ModbusSecurity::UnauthorizedModbusWrite")'

# Should see alert:
# {
#   "note": "ModbusSecurity::UnauthorizedModbusWrite",
#   "msg": "Unauthorized Modbus write from 10.20.30.99 to PLC 10.20.10.10 (FC: 6)",
#   ...
# }
```

Step 5: Deploy Suricata

```
# Install and configure
bash install_suricata_ics.sh

# Test rules
sudo suricata -T -c /etc/suricata/suricata.yaml

# Run in live mode
```

```
sudo suricata -c /etc/suricata/suricata.yaml -i eth1

# Monitor alerts
tail -f /var/log/suricata/fast.log
```

Step 6: Integration with SIEM

```
# Forward Zeek logs to SIEM (Splunk example)
sudo /opt/splunkforwarder/bin/splunk add monitor /opt/zeek/logs/current/ -sourcetype
zeek:json

# Forward Suricata logs
sudo /opt/splunkforwarder/bin/splunk add monitor /var/log/suricata/eve.json -sourcetype
suricata
```

Lab Deliverables

1. Zeek installation with ICSNPP plugins
2. Custom detection scripts (Modbus writes, recon, C2 beaconing)
3. Suricata with custom ICS rules
4. Baseline whitelist from 24 hours of traffic
5. Test report showing successful detection of Module 2 attacks
6. SIEM integration (Splunk/ELK dashboard)

8. Real-World Case Study: Detecting Industroyer

Industroyer/CrashOverride (2016 Ukraine power grid attack) used IEC 60870-5-104 protocol to send commands to circuit breakers.

How IDS Would Detect:

```
# detect_industroyer.zeek
# Detect IEC 104 attack patterns similar to Industroyer

@load icsnpp/iec104

event iec104_asdu(c: connection, is_orig: bool, asdu: IEC104::ASDU) {

    # Detect direct control commands (used by Industroyer)
    if (asdu$type_id == 45 || asdu$type_id == 46) {  # C_SC_NA_1 or C_DC_NA_1
        NOTICE([$note=IEC104Attack,
                $msg=fmt("IEC 104 control command from %s (Industroyer-like)", c$id$orig_h),
                $conn=c]);
    }

    # Detect station interrogation (reconnaissance)
    if (asdu$type_id == 100) {  # C_IC_NA_1
        NOTICE([$note=IEC104Recon,
```

```
            $msg=fmt("IEC 104 interrogation from %s", c$id$orig_h),
            $conn=c]);
    }
}
```

Suricata Rule:

```
alert tcp any any -> $SCADA_NETWORK 2404 (
    msg:"SCADA-APT IEC 104 Control Command (Industroyer-like)";
    flow:to_server,established;
    content:"|68|"; offset:0; depth:1;  # IEC 104 start byte
    content:"|2D|"; distance:6; within:1;  # Type ID 45: Single command
    classtype:attempted-admin;

reference:url,www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/;
    sid:9000301; rev:1;
)
```

9. Tools and Resources

IDS Platforms

- **Zeek**: https://zeek.org
- **ICSNPP Plugins**: https://github.com/cisagov/icsnpp
- **Suricata**: https://suricata.io
- **Snort**: https://www.snort.org

ICS-Specific Rules

- **Quickdraw (Digital Bond)**: https://github.com/digitalbond/Quickdraw-Snort
- **Emerging Threats ICS**: https://rules.emergingthreats.net
- **Cisco Talos ICS**: https://www.snort.org/downloads

Commercial IDS

- **Nozomi Networks**: https://www.nozominetworks.com
- **Claroty**: https://claroty.com
- **Dragos Platform**: https://www.dragos.com
- **Armis**: https://www.armis.com

Learning Resources

- **ICS-CERT**: https://www.cisa.gov/ics
- **SANS ICS515**: ICS Visibility, Detection, and Response
- **Applied Purple Teaming for ICS**: https://www.sans.org/white-papers/

Conclusion

Intrusion detection in OT environments requires specialized tools and techniques:

- **Protocol-aware IDS** (Zeek with ICSNPP) to understand Modbus, S7comm, DNP3
- **Custom detection rules** tailored to specific OT attack patterns
- **Protocol whitelisting** based on known-good baseline traffic
- **Anomaly detection** to catch zero-day attacks
- **Minimal false positives** through careful tuning and maintenance windows

In the next lesson, we'll explore OT asset management and safe vulnerability scanning techniques to maintain visibility over all devices in the industrial network.

Lesson 03: Asset Mgmnt & Vulnerability Scanning

Lesson 03: OT Asset Management & Vulnerability Scanning

Learning Objectives

- Deploy passive asset discovery tools for OT networks
- Perform safe active scanning without disrupting industrial processes
- Build comprehensive asset inventory with firmware versions and CVE mapping
- Prioritize vulnerabilities by operational risk
- Implement continuous asset monitoring
- Develop patch management strategies for OT environments

Introduction

Asset visibility is the foundation of OT security. Unlike IT networks where agents can be deployed to every endpoint, OT networks contain:

- **Legacy devices** without agent support (20+ year old PLCs)
- **Safety-critical systems** that cannot be modified
- **Embedded devices** with limited resources
- **Proprietary protocols** not supported by standard discovery tools

Key Challenges:

- Active scanning can crash PLCs or disrupt operations
- Many OT devices don't respond to standard discovery (ICMP, SNMP)
- Firmware versions often not exposed via network protocols
- No centralized asset database in most facilities

This lesson provides defensive visibility to counter Module 2 attacks:

- **Rogue devices** (Module 2 lateral movement): Detect unauthorized devices on OT network
- **Firmware manipulation** (Module 2 Lesson 06): Baseline firmware hashes to detect tampering
- **Supply chain attacks** (Module 2 Lesson 07): Inventory all software/firmware for vulnerability assessment

1. Passive Asset Discovery

1.1 GRASSMARLIN

GRASSMARLIN (developed by NSA, released as open-source) performs passive network monitoring to discover ICS/SCADA devices without sending any packets.

Installation:

```
#!/bin/bash
# install_grassmarlin.sh

# Download GRASSMARLIN
cd /opt
wget
https://github.com/nsacyber/GRASSMARLIN/releases/download/v3.2.1/grassmarlin-3.2.1.zip
unzip grassmarlin-3.2.1.zip
cd GRASSMARLIN-3.2.1

# Install Java (required)
sudo apt install -y openjdk-11-jre

# Run GRASSMARLIN
java -jar grassmarlin.jar
```

Usage:

```
# Capture traffic for analysis
sudo tcpdump -i eth1 -w ot_traffic.pcap -G 3600 -W 24  # 24 hours of capture

# Import into GRASSMARLIN
# File -> Import PCAP -> ot_traffic.pcap
# Wait for analysis to complete
# View: Logical Graph (network topology) and Physical View (asset list)
```

GRASSMARLIN Output:

- Network topology diagram showing all communicating devices
- Device fingerprinting (vendor, model, protocol)
- Protocol distribution (Modbus, S7comm, DNP3, etc.)
- Communication patterns and data flows

1.2 Zeek-Based Passive Discovery

```
# passive_asset_discovery.py
# Extract asset inventory from Zeek logs

import json
from collections import defaultdict
import ipaddress

class PassiveAssetDiscovery:
    def __init__(self, zeek_log_dir):
```

```
    self.zeek_log_dir = zeek_log_dir
    self.assets = {}

def discover_from_conn_log(self):
    """Discover assets from Zeek conn.log"""
    conn_log = f"{self.zeek_log_dir}/conn.log"

    with open(conn_log, 'r') as f:
        for line in f:
            if line.startswith('#'):
                continue

            try:
                log = json.loads(line)
                src_ip = log.get('id.orig_h')
                dst_ip = log.get('id.resp_h')
                dst_port = log.get('id.resp_p')
                proto = log.get('proto')

                # Record both source and destination
                for ip in [src_ip, dst_ip]:
                    if ip not in self.assets:
                        self.assets[ip] = {
                            'ip': ip,
                            'mac': None,
                            'hostname': None,
                            'vendor': None,
                            'protocols': set(),
                            'open_ports': set(),
                            'first_seen': log.get('ts'),
                            'last_seen': log.get('ts')
                        }

                    # Update last seen
                    self.assets[ip]['last_seen'] = log.get('ts')

                # Record destination port
                if dst_port:
                    self.assets[dst_ip]['open_ports'].add(dst_port)

                    # Infer protocol from port
                    if dst_port == 502:
                        self.assets[dst_ip]['protocols'].add('Modbus TCP')
                    elif dst_port == 102:
                        self.assets[dst_ip]['protocols'].add('S7comm')
                    elif dst_port == 20000:
                        self.assets[dst_ip]['protocols'].add('DNP3')
                    elif dst_port == 44818:
```

```
                    self.assets[dst_ip]['protocols'].add('EtherNet/IP')
                elif dst_port == 2222:
                    self.assets[dst_ip]['protocols'].add('EtherCAT')

            except:
                continue

    def discover_from_modbus_log(self):
        """Extract device info from Modbus traffic"""
        modbus_log = f"{self.zeek_log_dir}/modbus.log"

        try:
            with open(modbus_log, 'r') as f:
                for line in f:
                    if line.startswith('#'):
                        continue

                    log = json.loads(line)
                    dst_ip = log.get('id.resp_h')

                    if dst_ip in self.assets:
                        self.assets[dst_ip]['device_type'] = 'PLC/RTU'

                        # Extract Modbus device identification if available
                        if 'mei_response' in log:
                            self.assets[dst_ip]['vendor'] = log.get('mei_response', {}).get('vendor')
                            self.assets[dst_ip]['product_code'] = log.get('mei_response',
{}).get('product_code')
        except:
            pass

    def discover_from_s7comm_log(self):
        """Extract Siemens PLC info from S7comm traffic"""
        s7comm_log = f"{self.zeek_log_dir}/s7comm.log"

        try:
            with open(s7comm_log, 'r') as f:
                for line in f:
                    if line.startswith('#'):
                        continue

                    log = json.loads(line)
                    dst_ip = log.get('id.resp_h')

                    if dst_ip in self.assets:
                        self.assets[dst_ip]['device_type'] = 'Siemens PLC'
                        self.assets[dst_ip]['vendor'] = 'Siemens'
```

```
                    # Extract PLC type from rosctr field
                    if 'rosctr_name' in log and 'UserData' in log['rosctr_name']:
                        self.assets[dst_ip]['model'] = 'S7-1200/1500'
        except:
            pass

    def export_inventory(self, output_file='asset_inventory.json'):
        """Export discovered assets to JSON"""
        # Convert sets to lists for JSON serialization
        for ip, asset in self.assets.items():
            asset['protocols'] = list(asset['protocols'])
            asset['open_ports'] = list(asset['open_ports'])

        with open(output_file, 'w') as f:
            json.dump(self.assets, f, indent=2)

        print(f"[+] Discovered {len(self.assets)} assets")
        print(f"[+] Asset inventory exported to {output_file}")

        # Print summary
        plc_count = sum(1 for a in self.assets.values() if a.get('device_type') in ['PLC/RTU',
'Siemens PLC'])
        print(f"[+] PLCs/RTUs: {plc_count}")

        protocols = defaultdict(int)
        for asset in self.assets.values():
            for proto in asset.get('protocols', []):
                protocols[proto] += 1

        print(f"[+] Protocol distribution: {dict(protocols)}")

# Usage
if __name__ == '__main__':
    discovery = PassiveAssetDiscovery('/opt/zeek/logs/current')

    print("[*] Discovering assets from Zeek logs...")
    discovery.discover_from_conn_log()
    discovery.discover_from_modbus_log()
    discovery.discover_from_s7comm_log()

    discovery.export_inventory()
```

Expected Output:

```
[*] Discovering assets from Zeek logs...
[+] Discovered 47 assets
[+] Asset inventory exported to asset_inventory.json
[+] PLCs/RTUs: 12
```

[+] Protocol distribution: {'Modbus TCP': 12, 'S7comm': 5, 'DNP3': 2, 'EtherNet/IP': 3}

Sample Inventory JSON:


```
{
  "10.20.10.10": {
    "ip": "10.20.10.10",
    "mac": "00:1B:1B:9F:4A:2C",
    "hostname": null,
    "vendor": "Siemens",
    "device_type": "Siemens PLC",
    "model": "S7-1200/1500",
    "protocols": ["S7comm", "Modbus TCP"],
    "open_ports": [102, 502],
    "first_seen": "2025-01-03T08:00:00.000Z",
    "last_seen": "2025-01-03T14:30:00.000Z"
  }
}
```


2. Safe Active Scanning

2.1 Nmap for OT (Conservative Approach)

```
#!/bin/bash
# safe_ot_scan.sh
# Conservative Nmap scanning for OT networks

TARGET="10.20.10.0/24"  # PLC network

echo "[*] Starting SAFE OT network scan"
echo "[!] WARNING: Test in lab first, get change approval for production"

# Phase 1: Ping sweep (ICMP only, no port scan)
echo "[*] Phase 1: Ping sweep (non-intrusive)"
nmap -sn -PE -PP $TARGET -oN ot_ping_sweep.txt

# Phase 2: TCP Connect scan (no SYN scan, full 3-way handshake)
# Only scan known ICS ports
echo "[*] Phase 2: TCP Connect scan on ICS ports only"
nmap -Pn -sT \
  -p 102,502,20000,44818,2222,47808,34962,34963,34964 \
  --max-retries 1 \
  --scan-delay 1000ms \
  --max-rate 10 \
  $TARGET \
  -oN ot_port_scan.txt

# Phase 3: Service version detection (CAUTIOUS)
```

```
# Only on devices that responded in Phase 2
echo "[*] Phase 3: Service version detection (minimal probes)"
nmap -Pn -sT \
  -p 102,502 \
  --version-intensity 0 \
  --max-retries 1 \
  --scan-delay 2000ms \
  $TARGET \
  -oN ot_version_scan.txt

echo "[+] Scan complete. Review output files."
echo "[!] Verify no devices went offline during scan"
```

Safe Nmap Parameters for OT:

- `-sT`: TCP Connect (full 3-way handshake, not SYN scan)
- `-Pn`: Skip ping (assume host is up, avoid ICMP that might crash PLCs)
- `--max-retries 1`: Only retry once
- `--scan-delay 1000ms`: Wait 1 second between probes
- `--max-rate 10`: Limit to 10 packets/second
- `--version-intensity 0`: Minimal version probes

NSE Scripts for ICS (use with caution):

```
# Modbus device identification
nmap -Pn -sT -p 502 \
  --script modbus-discover.nse \
  --script-args='modbus-discover.aggressive=false' \
  10.20.10.10

# S7comm PLC identification
nmap -Pn -sT -p 102 \
  --script s7-info.nse \
  10.20.10.10

# BACnet device enumeration
nmap -Pn -sU -p 47808 \
  --script bacnet-info.nse \
  10.20.10.15
```

2.2 Tenable OT Security / Nessus Industrial

Tenable OT Security (formerly Indegy) provides safe scanning specifically designed for OT:

```
# Using Nessus with OT-safe scan policy

# 1. Create custom scan policy in Nessus UI:
# - Discovery: Ping sweep only
```

```
# - Port scan: Custom port list (102,502,20000,44818)
# - Service discovery: Minimal
# - Network timing: Paranoid (slowest)
# - Max simultaneous checks per host: 3
# - Max concurrent hosts: 5

# 2. Run scan via CLI
/opt/nessus/bin/nessuscli scan new \
  --policy "OT Safe Scan Policy" \
  --targets "10.20.10.0/24" \
  --name "PLC Network Discovery"

# 3. Export results
/opt/nessus/bin/nessuscli scan export 123 --format csv
```

3. Asset Inventory Management

3.1 Automated Asset Database

```
# asset_database.py
# Centralized OT asset inventory with SQLite backend

import sqlite3
import json
import hashlib
from datetime import datetime

class AssetDatabase:
    def __init__(self, db_path='ot_assets.db'):
        self.db_path = db_path
        self.init_database()

    def init_database(self):
        """Initialize asset database schema"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS assets (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                ip_address TEXT UNIQUE NOT NULL,
                mac_address TEXT,
                hostname TEXT,
                vendor TEXT,
                model TEXT,
                device_type TEXT,
                firmware_version TEXT,
```

```
        firmware_hash TEXT,
        serial_number TEXT,
        purdue_level INTEGER,
        criticality TEXT,
        location TEXT,
        owner TEXT,
        first_discovered TEXT,
        last_seen TEXT,
        status TEXT
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS open_ports (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        asset_id INTEGER,
        port INTEGER,
        protocol TEXT,
        service TEXT,
        FOREIGN KEY (asset_id) REFERENCES assets(id)
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS vulnerabilities (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        asset_id INTEGER,
        cve_id TEXT,
        cvss_score REAL,
        description TEXT,
        remediation TEXT,
        status TEXT,
        discovered_date TEXT,
        FOREIGN KEY (asset_id) REFERENCES assets(id)
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS firmware_baseline (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        asset_id INTEGER,
        firmware_hash TEXT,
        capture_date TEXT,
        is_golden BOOLEAN,
        FOREIGN KEY (asset_id) REFERENCES assets(id)
    )
''')
```

```
        conn.commit()
        conn.close()

    def add_asset(self, asset_data):
        """Add new asset to database"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        try:
            cursor.execute('''
                INSERT INTO assets (
                    ip_address, mac_address, hostname, vendor, model,
                    device_type, firmware_version, serial_number,
                    purdue_level, criticality, location, owner,
                    first_discovered, last_seen, status
                ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', (
                asset_data['ip'],
                asset_data.get('mac'),
                asset_data.get('hostname'),
                asset_data.get('vendor'),
                asset_data.get('model'),
                asset_data.get('device_type'),
                asset_data.get('firmware_version'),
                asset_data.get('serial_number'),
                asset_data.get('purdue_level'),
                asset_data.get('criticality', 'Medium'),
                asset_data.get('location'),
                asset_data.get('owner'),
                datetime.now().isoformat(),
                datetime.now().isoformat(),
                'Active'
            ))

            asset_id = cursor.lastrowid

            # Add open ports
            for port_info in asset_data.get('ports', []):
                cursor.execute('''
                    INSERT INTO open_ports (asset_id, port, protocol, service)
                    VALUES (?, ?, ?, ?)
                ''', (asset_id, port_info['port'], port_info.get('protocol', 'tcp'), port_info.get('service')))

            conn.commit()
            print(f"[+] Added asset: {asset_data['ip']} ({asset_data.get('device_type',
'Unknown')})")

        except sqlite3.IntegrityError:
```

```
            print(f"[!] Asset {asset_data['ip']} already exists, updating instead")
            self.update_asset(asset_data)

        conn.close()

    def update_asset(self, asset_data):
        """Update existing asset"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            UPDATE assets
            SET last_seen = ?, mac_address = ?, hostname = ?,
                vendor = ?, model = ?, firmware_version = ?
            WHERE ip_address = ?
        ''', (
            datetime.now().isoformat(),
            asset_data.get('mac'),
            asset_data.get('hostname'),
            asset_data.get('vendor'),
            asset_data.get('model'),
            asset_data.get('firmware_version'),
            asset_data['ip']
        ))

        conn.commit()
        conn.close()

    def add_vulnerability(self, ip_address, cve_data):
        """Add CVE to asset"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        # Get asset ID
        cursor.execute('SELECT id FROM assets WHERE ip_address = ?', (ip_address,))
        result = cursor.fetchone()

        if not result:
            print(f"[!] Asset {ip_address} not found")
            return

        asset_id = result[0]

        cursor.execute('''
            INSERT INTO vulnerabilities (
                asset_id, cve_id, cvss_score, description, remediation, status, discovered_date
            ) VALUES (?, ?, ?, ?, ?, ?, ?)
        ''', (
```

```
            asset_id,
            cve_data['cve_id'],
            cve_data.get('cvss_score', 0.0),
            cve_data.get('description'),
            cve_data.get('remediation'),
            'Open',
            datetime.now().isoformat()
        ))

        conn.commit()
        conn.close()

        print(f"[+] Added {cve_data['cve_id']} to {ip_address}")

    def get_critical_assets(self):
        """Get all critical assets"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT ip_address, device_type, vendor, model, criticality
            FROM assets
            WHERE criticality = 'Critical'
            ORDER BY purdue_level
        ''')

        assets = cursor.fetchall()
        conn.close()

        return assets

    def get_vulnerable_assets(self, cvss_threshold=7.0):
        """Get assets with high-severity vulnerabilities"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT DISTINCT a.ip_address, a.device_type, v.cve_id, v.cvss_score
            FROM assets a
            JOIN vulnerabilities v ON a.id = v.asset_id
            WHERE v.cvss_score >= ? AND v.status = 'Open'
            ORDER BY v.cvss_score DESC
        ''', (cvss_threshold,))

        vulnerabilities = cursor.fetchall()
        conn.close()

        return vulnerabilities
```

```
    def export_csv(self, output_file='asset_inventory.csv'):
        """Export inventory to CSV"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT ip_address, mac_address, hostname, vendor, model,
                   device_type, firmware_version, purdue_level, criticality,
                   location, owner, last_seen
            FROM assets
            ORDER BY purdue_level, ip_address
        ''')

        import csv
        with open(output_file, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(['IP', 'MAC', 'Hostname', 'Vendor', 'Model', 'Type',
                             'Firmware', 'Purdue Level', 'Criticality', 'Location', 'Owner', 'Last Seen'])

            for row in cursor.fetchall():
                writer.writerow(row)

        conn.close()
        print(f"[+] Inventory exported to {output_file}")

# Usage example
if __name__ == '__main__':
    db = AssetDatabase()

    # Add PLC asset
    plc = {
        'ip': '10.20.10.10',
        'mac': '00:1B:1B:9F:4A:2C',
        'vendor': 'Siemens',
        'model': 'S7-1200 CPU 1214C',
        'device_type': 'PLC',
        'firmware_version': 'V4.5.2',
        'serial_number': 'S C-X4U304308012',
        'purdue_level': 2,
        'criticality': 'Critical',
        'location': 'Water Treatment - Building A',
        'owner': 'Operations Team',
        'ports': [
            {'port': 102, 'protocol': 'tcp', 'service': 'S7comm'},
            {'port': 502, 'protocol': 'tcp', 'service': 'Modbus'}
        ]
    }
```

```
    db.add_asset(plc)

    # Add vulnerability
    db.add_vulnerability('10.20.10.10', {
        'cve_id': 'CVE-2022-38465',
        'cvss_score': 9.8,
        'description': 'Siemens SIMATIC S7-1200 CPU vulnerable to unauthenticated remote
code execution',
        'remediation': 'Update to firmware V4.6.0 or later'
    })

    # Get critical assets
    print("\n[*] Critical Assets:")
    for asset in db.get_critical_assets():
        print(f"  {asset[0]} - {asset[1]} ({asset[2]} {asset[3]})")

    # Get vulnerable assets
    print("\n[*] High-Risk Vulnerabilities (CVSS >= 7.0):")
    for vuln in db.get_vulnerable_assets():
        print(f"  {vuln[0]} - {vuln[2]} (CVSS: {vuln[3]})")

    # Export to CSV
    db.export_csv()
```

4. Vulnerability Management

4.1 CVE Correlation

```
# cve_correlation.py
# Correlate asset inventory with CVE database

import requests
import json
import time
from asset_database import AssetDatabase

class CVECorrelation:
    def __init__(self, nvd_api_key=None):
        self.nvd_api_key = nvd_api_key
        self.nvd_url = "https://services.nvd.nist.gov/rest/json/cves/2.0"

    def search_cves(self, vendor, product, version=None):
        """Search NVD for CVEs affecting specific product"""

        # Build search query
        keyword = f"{vendor} {product}"
```

```
if version:
    keyword += f" {version}"

params = {
    'keywordSearch': keyword,
    'resultsPerPage': 100
}

headers = {}
if self.nvd_api_key:
    headers['apiKey'] = self.nvd_api_key

try:
    response = requests.get(self.nvd_url, params=params, headers=headers)

    if response.status_code == 200:
        data = response.json()
        cves = []

        for item in data.get('vulnerabilities', []):
            cve = item.get('cve', {})
            cve_id = cve.get('id')

            # Extract CVSS score
            cvss_data = cve.get('metrics', {}).get('cvssMetricV31', [])
            cvss_score = 0.0
            if cvss_data:
                cvss_score = cvss_data[0].get('cvssData', {}).get('baseScore', 0.0)

            # Extract description
            descriptions = cve.get('descriptions', [])
            description = descriptions[0].get('value', '') if descriptions else ''

            cves.append({
                'cve_id': cve_id,
                'cvss_score': cvss_score,
                'description': description,
                'published': cve.get('published')
            })

        return cves
    else:
        print(f"[!] NVD API error: {response.status_code}")
        return []

except Exception as e:
    print(f"[!] Error searching CVEs: {e}")
    return []
```

```
    def correlate_asset_inventory(self, db_path='ot_assets.db'):
        """Correlate all assets in database with CVEs"""
        import sqlite3

        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()

        # Get all assets
        cursor.execute('SELECT id, ip_address, vendor, model, firmware_version FROM
assets')
        assets = cursor.fetchall()

        db = AssetDatabase(db_path)

        for asset in assets:
            asset_id, ip, vendor, model, firmware = asset

            if not vendor or not model:
                continue

            print(f"\n[*] Checking CVEs for {ip} ({vendor} {model})")

            # Search CVEs
            cves = self.search_cves(vendor, model, firmware)

            print(f"[+] Found {len(cves)} potential CVEs")

            # Add high-severity CVEs to database
            for cve in cves:
                if cve['cvss_score'] >= 4.0:  # Medium or higher
                    db.add_vulnerability(ip, cve)

            # Rate limit (NVD allows 5 requests/30 seconds without API key)
            time.sleep(6)

        conn.close()
        print("\n[+] CVE correlation complete")

# Usage
if __name__ == '__main__':
    correlator = CVECorrelation(nvd_api_key='YOUR_API_KEY')  # Get free key from
nvd.nist.gov
    correlator.correlate_asset_inventory()
```

4.2 Risk Prioritization Matrix

```
# vulnerability_prioritization.py
```

```
# Prioritize vulnerabilities by operational risk

class VulnerabilityPrioritization:
    def __init__(self, db_path='ot_assets.db'):
        self.db_path = db_path

    def calculate_risk_score(self, cvss, criticality, exploitability, purdue_level):
        """
        Calculate operational risk score

        Risk = (CVSS * 0.3) + (Criticality * 0.3) + (Exploitability * 0.2) + (Purdue Impact * 0.2)

        Scale: 0-10 (10 = highest risk)
        """

        # Map criticality to numeric
        criticality_map = {'Low': 2, 'Medium': 5, 'High': 7, 'Critical': 10}
        criticality_score = criticality_map.get(criticality, 5)

        # Map exploitability
        exploit_map = {'Not Defined': 5, 'Unproven': 3, 'Proof of Concept': 6, 'Functional': 8,
'High': 10}
        exploit_score = exploit_map.get(exploitability, 5)

        # Purdue level impact (Level 0-1 = highest impact)
        purdue_impact = {0: 10, 1: 9, 2: 8, 3: 6, 4: 4, 5: 2}.get(purdue_level, 5)

        # Calculate weighted risk
        risk = (cvss * 0.3) + (criticality_score * 0.3) + (exploit_score * 0.2) + (purdue_impact *
0.2)

        return round(risk, 2)

    def prioritize_vulnerabilities(self):
        """Generate prioritized vulnerability report"""
        import sqlite3

        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT
                a.ip_address,
                a.device_type,
                a.vendor,
                a.model,
                a.criticality,
                a.purdue_level,
```

```
            v.cve_id,
            v.cvss_score,
            v.description
        FROM assets a
        JOIN vulnerabilities v ON a.id = v.asset_id
        WHERE v.status = 'Open'
    ''')

    vulnerabilities = []

    for row in cursor.fetchall():
        ip, device_type, vendor, model, criticality, purdue, cve, cvss, desc = row

        # Calculate risk score
        risk_score = self.calculate_risk_score(
            cvss,
            criticality,
            'Functional',  # Assume functional exploit for OT CVEs
            purdue
        )

        vulnerabilities.append({
            'ip': ip,
            'device': f"{vendor} {model}",
            'cve': cve,
            'cvss': cvss,
            'risk_score': risk_score,
            'purdue_level': purdue,
            'criticality': criticality,
            'description': desc[:100] + '...'
        })

    # Sort by risk score (descending)
    vulnerabilities.sort(key=lambda x: x['risk_score'], reverse=True)

    conn.close()
    return vulnerabilities

def generate_report(self, output_file='vulnerability_report.txt'):
    """Generate human-readable vulnerability report"""
    vulns = self.prioritize_vulnerabilities()

    with open(output_file, 'w') as f:
        f.write("="*80 + "\n")
        f.write("OT VULNERABILITY PRIORITIZATION REPORT\n")
        f.write(f"Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write("="*80 + "\n\n")
```

```
            f.write(f"Total Vulnerabilities: {len(vulns)}\n\n")

            # Group by risk level
            critical_risk = [v for v in vulns if v['risk_score'] >= 8.0]
            high_risk = [v for v in vulns if 6.0 <= v['risk_score'] < 8.0]
            medium_risk = [v for v in vulns if 4.0 <= v['risk_score'] < 6.0]

            f.write(f"CRITICAL RISK (Score >= 8.0): {len(critical_risk)}\n")
            f.write(f"HIGH RISK (Score 6.0-7.9): {len(high_risk)}\n")
            f.write(f"MEDIUM RISK (Score 4.0-5.9): {len(medium_risk)}\n\n")

            f.write("="*80 + "\n")
            f.write("TOP 10 HIGHEST RISK VULNERABILITIES\n")
            f.write("="*80 + "\n\n")

            for i, vuln in enumerate(vulns[:10], 1):
                f.write(f"{i}. {vuln['cve']} - Risk Score: {vuln['risk_score']}\n")
                f.write(f"   Asset: {vuln['ip']} ({vuln['device']})\n")
                f.write(f"   CVSS: {vuln['cvss']} | Purdue Level: {vuln['purdue_level']} | Criticality:
{vuln['criticality']}\n")
                f.write(f"   {vuln['description']}\n\n")

        print(f"[+] Vulnerability report generated: {output_file}")

        # Print summary
        print(f"\n[*] Vulnerability Summary:")
        print(f"    CRITICAL: {len(critical_risk)}")
        print(f"    HIGH: {len(high_risk)}")
        print(f"    MEDIUM: {len(medium_risk)}")

# Usage
if __name__ == '__main__':
    prioritizer = VulnerabilityPrioritization()
    prioritizer.generate_report()
```

5. Patch Management for OT

5.1 Challenges

Unlike IT environments where patches can be deployed automatically, OT patch management faces:

1. **Downtime Requirements**: PLCs cannot be patched while controlling processes
2. **Testing Requirements**: Patches must be validated in lab before production
3. **Vendor Dependencies**: Many vendors release patches slowly (or never for legacy systems)
4. **Change Control**: All changes require engineering review and management approval

5. **Legacy Systems**: 20+ year old PLCs with no available patches

5.2 OT Patch Management Process

```
# patch_management_workflow.py
# Track patch deployment for OT environment

import sqlite3
from datetime import datetime

class PatchManagement:
    def __init__(self, db_path='ot_patches.db'):
        self.db_path = db_path
        self.init_database()

    def init_database(self):
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS patches (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                vendor TEXT,
                product TEXT,
                patch_version TEXT,
                cves_addressed TEXT,
                release_date TEXT,
                severity TEXT,
                status TEXT,
                lab_tested BOOLEAN,
                lab_test_date TEXT,
                production_deployed BOOLEAN,
                deployment_date TEXT,
                affected_assets TEXT,
                notes TEXT
            )
        ''')

        conn.commit()
        conn.close()

    def add_patch(self, patch_data):
        """Add new patch to tracking system"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            INSERT INTO patches (
```

```
                vendor, product, patch_version, cves_addressed,
                release_date, severity, status, lab_tested,
                production_deployed, affected_assets
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            patch_data['vendor'],
            patch_data['product'],
            patch_data['patch_version'],
            ','.join(patch_data.get('cves', [])),
            patch_data['release_date'],
            patch_data.get('severity', 'Medium'),
            'Pending Review',
            False,
            False,
            ','.join(patch_data.get('affected_assets', []))
        ))

        conn.commit()
        conn.close()

        print(f"[+] Added patch: {patch_data['vendor']} {patch_data['product']}
{patch_data['patch_version']}")

    def update_lab_test(self, patch_id, success, notes):
        """Update lab test results"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        status = 'Lab Approved' if success else 'Lab Failed'

        cursor.execute('''
            UPDATE patches
            SET lab_tested = ?, lab_test_date = ?, status = ?, notes = ?
            WHERE id = ?
        ''', (True, datetime.now().isoformat(), status, notes, patch_id))

        conn.commit()
        conn.close()

        print(f"[+] Updated lab test for patch ID {patch_id}: {status}")

    def schedule_deployment(self, patch_id, maintenance_window):
        """Schedule patch deployment"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            UPDATE patches
```

```
            SET status = 'Scheduled for Deployment'
            WHERE id = ?
        ''', (patch_id,))

        conn.commit()
        conn.close()

        print(f"[+] Patch ID {patch_id} scheduled for deployment: {maintenance_window}")

    def mark_deployed(self, patch_id):
        """Mark patch as deployed to production"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            UPDATE patches
            SET production_deployed = ?, deployment_date = ?, status = 'Deployed'
            WHERE id = ?
        ''', (True, datetime.now().isoformat(), patch_id))

        conn.commit()
        conn.close()

        print(f"[+] Patch ID {patch_id} marked as deployed")

    def get_pending_patches(self):
        """Get patches pending review or deployment"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT id, vendor, product, patch_version, cves_addressed, severity, status
            FROM patches
            WHERE production_deployed = 0
            ORDER BY severity DESC, release_date ASC
        ''')

        patches = cursor.fetchall()
        conn.close()

        return patches

# Usage
if __name__ == '__main__':
    pm = PatchManagement()

    # Add new patch
    pm.add_patch({
```

```
        'vendor': 'Siemens',
        'product': 'S7-1200 CPU',
        'patch_version': 'V4.6.0',
        'cves': ['CVE-2022-38465', 'CVE-2022-38466'],
        'release_date': '2023-11-15',
        'severity': 'Critical',
        'affected_assets': ['10.20.10.10', '10.20.10.11', '10.20.10.12']
    })

    # Lab test
    pm.update_lab_test(1, success=True, notes='Tested in lab, no issues with process
control')

    # Schedule deployment
    pm.schedule_deployment(1, '2025-01-15 02:00-06:00')

    # Mark deployed
    pm.mark_deployed(1)
```

6. Hands-On Lab

Objective

Build complete asset inventory for water treatment facility, correlate with CVEs, prioritize vulnerabilities.

Lab Steps

Step 1: Deploy Passive Discovery

```
# Capture 48 hours of traffic
sudo tcpdump -i eth1 -w ot_baseline.pcap -G 172800 -W 1

# Analyze with Zeek
zeek -r ot_baseline.pcap \
  local.zeek \
  icsnpp/modbus \
  icsnpp/s7comm \
  icsnpp/dnp3

# Run passive discovery script
python3 passive_asset_discovery.py
```

Step 2: Safe Active Scanning

```
# Run safe OT scan
bash safe_ot_scan.sh
```

```
# Review results
cat ot_port_scan.txt
```

Step 3: Build Asset Database

```
# Import discovered assets
python3 asset_database.py
```

Step 4: CVE Correlation

```
# Correlate with CVE database
python3 cve_correlation.py
```

Step 5: Prioritize Vulnerabilities

```
# Generate prioritized report
python3 vulnerability_prioritization.py
```

Deliverables

1. Complete asset inventory (CSV export)
2. Network topology diagram
3. CVE correlation report
4. Prioritized vulnerability remediation plan
5. Patch deployment schedule

7. Tools and Resources

Asset Discovery

- **GRASSMARLIN**: https://github.com/nsacyber/GRASSMARLIN
- **Nozomi Networks**: Commercial passive discovery
- **Claroty**: Commercial asset management
- **Armis**: Agentless device discovery

Vulnerability Scanning

- **Tenable OT Security**: OT-safe vulnerability scanner
- **Nessus**: With OT scan policies
- **Rapid7 Nexpose**: ICS modules

CVE Databases

- **NVD**: https://nvd.nist.gov
- **ICS-CERT Advisories**: https://www.cisa.gov/ics-advisories
- **Siemens ProductCERT**: https://cert-portal.siemens.com
- **Schneider Electric**: https://www.se.com/ww/en/work/support/cybersecurity/

Conclusion

OT asset management requires:

- **Passive discovery first** to avoid disrupting operations
- **Conservative active scanning** with proper testing and approval
- **Continuous monitoring** to detect rogue devices
- **Vulnerability correlation** with operational risk prioritization
- **Methodical patch management** with lab testing

Lesson 04: SIEM for ICS Environments

Lesson 04: SIEM for ICS Environments

Industrial-Grade Detection, Correlation, and Investigation

Why This Lesson Matters

In IT environments, SIEM platforms are primarily used to detect data breaches, malware, privilege abuse, and service disruption.

In industrial environments, SIEM platforms protect physical processes, human safety, production continuity, regulatory compliance, and critical infrastructure.

A missed alert in an IT environment may result in data loss or downtime.

A missed alert in an ICS environment may result in explosions, blackouts, chemical leaks, contaminated water, destroyed equipment, or loss of life.

Because of this, SIEM design, alert logic, and incident response in ICS environments must follow a completely different mindset.

This lesson is intentionally deep, operational, and realistic.

Learning Outcomes

After completing this lesson, the student will be able to:

- Design an ICS-aware SIEM architecture without disrupting operations
- Identify and prioritize critical OT log sources
- Normalize industrial telemetry into security-relevant events
- Build behavioral baselines for industrial networks and processes
- Detect malicious, unsafe, and abnormal control actions
- Correlate IT compromise with OT process impact
- Perform forensic reconstruction of industrial security incidents
- Distinguish false positives from real operational risk
- Communicate findings to engineers, SOC analysts, and executive leadership

---

1. SIEM in ICS Environments

1.1 How ICS Monitoring Differs from IT Monitoring

Aspect	IT Environment	ICS Environment
Primary Risk	Data loss	Physical damage
Protected Assets	Users, servers, databases	PLCs, sensors, actuators
Change Frequency	High	Extremely low
Baseline Stability	Dynamic	Static
Time Sensitivity	Minutes to hours	Seconds
Tolerance for Scanning	High	Often prohibited
Incident Impact	Service degradation	Safety and production loss

Key principle:
In industrial environments, **any change is suspicious until proven legitimate**.

1.2 The Role of SIEM in Industrial Defense

SIEM does not replace firewalls, safety PLCs, physical controls, or segmentation.

SIEM provides visibility, correlation, historical context, and forensic timelines.

In ICS environments, SIEM acts as the system of record for security-relevant operational behavior.

---

2. SIEM Platforms Used in ICS

2.1 Commonly Deployed Platforms

- Splunk
  Strong correlation capabilities and extensive OT ecosystem

- IBM QRadar
  Asset-centric correlation and mature forensic workflows

- Elastic Stack
  Highly flexible and cost-effective, but requires engineering effort

- Microsoft Sentinel
  Strong IT and identity correlation, commonly used in hybrid environments

---

2.2 Criteria for Selecting a SIEM for ICS

A SIEM suitable for industrial environments must support:

- Industrial protocol visibility
- Asset-based correlation
- Low-latency ingestion
- Long-term forensic retention
- Strict access control and auditability
- Non-intrusive data collection methods

---

3. Industrial Log Sources

3.1 OT-Critical Log Sources

PLC and Controller Logs

- Program download and upload events
- Firmware changes
- Force and override actions
- Safety logic modifications

These logs are among the highest-value signals in ICS security.

HMI and Operator Interface Logs

- Manual control commands
- Alarm acknowledgments
- Setpoint changes
- Mode switches

Operator behavior is a critical signal for both insider threat and compromised credentials.

SCADA and Historian Logs

- Tag value changes
- Alarm generation

- Communication failures
- Data integrity errors

These logs help correlate cyber activity with physical process impact.

Network Security Logs

- Firewall allow and deny events
- Zone boundary crossings
- Protocol violations

Segmentation violations are often early indicators of lateral movement.

Network Detection and Protocol Analysis

- Zeek industrial protocol logs
- IDS alerts
- Deep packet inspection events

These logs provide protocol-level visibility without touching endpoints.

Access and Authentication Logs

- VPN connections
- Jump server access
- Windows authentication events
- Privileged access usage

These logs bridge IT identity with OT actions.

3.2 Log Collection Methods

Only non-intrusive collection methods are acceptable in ICS environments.

Approved approaches include:

- Syslog forwarding from network devices
- File-based log collection from servers
- API-based ingestion from vendor platforms
- Network traffic monitoring via SPAN or TAP

Agents must never be installed on PLCs or safety controllers.

---

4. ICS-Specific Detection Use Cases

Use Case 1: Unauthorized PLC Program Change

A PLC program download occurs from a workstation that is not part of the approved engineering environment.

This behavior often indicates malware infection, credential theft, or insider misuse.

Immediate validation of program integrity is required.

Use Case 2: Excessive Control Write Operations

A single host performs an unusually high number of write operations within a short time window.

This pattern is commonly associated with automation abuse, replay attacks, or protocol fuzzing.

Use Case 3: Off-Hours OT Access

Remote access to the OT environment occurs outside approved operational windows.

This may indicate stolen credentials, compromised VPN access, or unauthorized maintenance activity.

Use Case 4: New Device on the OT Network

A previously unseen MAC address or IP address appears in the OT network.

This is often caused by unauthorized laptops, compromised engineering devices, or rogue wireless bridges.

Use Case 5: Authentication Failure Spikes

Repeated failed authentication attempts occur against SCADA or jump systems.

This behavior frequently precedes successful compromise.

Use Case 6: Safety System Manipulation

Write operations target safety-related registers or logic.

This is one of the highest severity alerts possible in an ICS environment and requires immediate action.

---

5. Correlation Logic Examples

Unauthorized PLC Programming Detection

```
index=ics sourcetype=s7comm function_code IN (0x1B, 0x28)
| where src_ip NOT IN ("10.10.1.100", "10.10.1.101")
| stats count by src_ip, dest_ip, function_code
```

This rule detects unauthorized PLC program download attempts.

After-Hours OT Access Detection

```
index=vpn earliest=-1h
| eval hour=strftime(_time, "%H")
| where hour < 8 OR hour > 18
| table _time, user, src_ip, dest_ip
```

This rule identifies access outside defined operational hours.

IT to OT Attack Chain Correlation

```
(index=windows EventCode=4624 OR EventCode=4672)
| join user [ search index=ics sourcetype=modbus function="write" ]
| stats count by user, src_ip, dest_ip
```

This rule correlates privileged IT access with OT control actions.

---

6. Industrial Incident Investigation

Investigation Objectives

- Identify who initiated the action
- Determine how access was obtained
- Assess whether changes were manual or automated
- Confirm physical process impact
- Validate safety system integrity

Evidence Sources

- SIEM event timelines
- PLC backups and logic versions
- Network packet captures
- Operator action logs
- Historian process data

Preservation Rules

- Do not reboot controllers unless required for safety
- Preserve logs in read-only format
- Maintain chain of custody
- Coordinate actions with operations and engineering teams

---

7. Hands-On Labs

1. Deploy an ELK stack for OT monitoring
2. Ingest Zeek industrial protocol logs
3. Build five ICS-specific detection rules
4. Correlate a multi-stage IT to OT attack
5. Test alerts using simulated industrial attacks

Final Takeaways

- SIEM in ICS environments is about process integrity, not just security
- Stability is the baseline
- Correlation across IT and OT reveals real attack paths
- Safety and availability always override automation
- SIEM supports decision-making, not autonomous response

Lesson 05: Incident Detection and Threat Hunting

Lesson 05: Incident Detection and Threat Hunting in OT

Industrial Threat Hunting, Live Detection, and Active Defense

---

Why This Lesson Matters

In OT environments, incident detection is rarely about catching noisy malware.
Most real-world attacks against industrial systems are slow, quiet, and deliberate.

Adversaries prioritize:

- Stealth
- Persistence
- Long-term access
- Minimal operational disruption until objectives are achieved

Threat hunting in OT is therefore not reactive.
It is a proactive discipline focused on finding what should not exist in stable environments.

In industrial networks, **normal rarely changes**.
This makes threat hunting extremely powerful when done correctly.

Learning Outcomes

After completing this lesson, the student will be able to:

- Detect active and dormant attacks in ICS environments
- Build hypothesis-driven OT threat hunts
- Identify abnormal industrial protocol behavior
- Detect compromised engineering and operator workstations
- Identify indicators of compromise specific to OT
- Respond safely to security events without disrupting production
- Support containment and recovery with minimal operational risk

---

1. Threat Hunting in ICS Environments

1.1 Threat Hunting Philosophy in OT

Threat hunting in IT often relies on:

- Malware signatures
- Known IOCs
- Automated detection rules

Threat hunting in OT relies on:

- Stability
- Determinism
- Behavioral consistency
- Process awareness

Key principle:
If something is new, it deserves investigation.

1.2 Hunt Preparation

Before hunting begins, the environment must have:

- A documented network architecture
- Asset inventory of PLCs, HMIs, servers, and workstations
- Known-good baselines for traffic and behavior
- Defined operational windows
- Clear escalation paths with operations teams

Hunting without baselines leads to false conclusions.

---

2. ICS Threat Hunting Methodology

2.1 Hypothesis-Driven Hunting

Threat hunting starts with a hypothesis based on realistic adversary behavior.

A good hypothesis:

- Is specific
- Is testable
- Maps to real attack techniques
- Produces measurable results

2.2 Example Hunt Hypotheses

Hypothesis 1: Adversary is conducting reconnaissance via Modbus scan

Assumptions:

- Adversary is mapping registers and devices
- Increased read requests across multiple addresses
- Activity originates from non-PLC assets

Expected Evidence:

- Spike in Modbus read requests
- Sequential register access
- New source IPs interacting with PLCs

Hypothesis 2: Engineering workstation is compromised and beaconing to C2

Assumptions:

- Compromised workstation communicates periodically
- Outbound traffic pattern is consistent
- Destination is external or unexpected

Expected Evidence:

- Periodic connections
- Small, consistent packet sizes
- Communication outside normal maintenance windows

Hypothesis 3: PLC logic has been modified to include a backdoor

Assumptions:

- Logic has changed without approved maintenance
- Change persists across restarts
- No corresponding change ticket exists

Expected Evidence:

- Hash mismatch in PLC logic
- Upload events from unapproved sources
- Logic blocks that are unused or hidden

Hypothesis 4: Man-in-the-middle attack between SCADA and PLC

Assumptions:

- Traffic is intercepted or modified
- Latency or retransmissions increase
- PLC responses differ from expected values

Expected Evidence:

- Duplicate packets
- Unexpected MAC address changes
- ARP instability
- Timing anomalies

---

3. Hunting Techniques in OT

3.1 Network Traffic Analysis

Network visibility is the most reliable hunting surface in OT.

Passive monitoring is preferred.

Baseline Comparison for Connections

```
tshark -r current.pcap -T fields -e ip.src -e ip.dst -e tcp.port | sort -u > current_conns.txt
diff baseline_conns.txt current_conns.txt
```

Purpose:

- Identify new hosts
- Detect unexpected communication paths
- Reveal lateral movement

Interpretation:

- Any new connection must be explained
- Focus on traffic crossing zone boundaries

Protocol Discovery

```
tshark -r current.pcap -q -z io,phs
```

Purpose:

- Identify new or unexpected protocols
- Detect tunneling or covert channels

Interpretation:

- ICS environments rarely introduce new protocols
- New protocol usage is suspicious by default

Beaconing Detection
zeek -r current.pcap detect_beaconing.zeek

Purpose:

- Identify periodic communication
- Detect command-and-control behavior

Interpretation:

- Consistent timing patterns indicate automation
- Even internal beaconing can indicate persistence

3.2 Industrial Protocol Behavioral Analysis

Focus areas:

- Write operations
- Function code frequency
- Command sequencing
- Timing anomalies

Examples of suspicious behavior:

- Writes outside maintenance windows
- Diagnostic function usage
- Rapid register cycling
- Repeated force commands

3.3 File Integrity Monitoring for PLC Logic

PLC logic is equivalent to executable code.

Any change must be justified.

```
import snap7
import hashlib

def monitor_plc_integrity(plc_ip, baseline_hash):
    plc = snap7.client.Client()
    plc.connect(plc_ip, 0, 1)

    ob1 = plc.upload('OB', 1)
    current_hash = hashlib.sha256(ob1).hexdigest()

    if current_hash != baseline_hash:
        print("[!] ALERT: PLC logic has changed!")
        print(f"Baseline: {baseline_hash}")
        print(f"Current:  {current_hash}")
        return False

    plc.disconnect()
    return True
```

Operational Notes:

- Baseline hashes must be captured during trusted states
- Hash checks should be scheduled and controlled
- Any mismatch requires engineering validation

---

4. Indicators of Compromise in OT

4.1 Common OT-Specific IOCs

Network Indicators:

- Unknown IPs communicating with PLCs
- Communication outside defined zones
- ARP instability or duplicate MACs

System Indicators:

- New user accounts in SCADA systems
- Unexpected services or processes on HMI
- Unauthorized scheduled tasks

Controller Indicators:

- Modified PLC firmware
- Unexpected program blocks

- Logic changes without downtime

Protocol Indicators:

- Use of diagnostic or rarely used function codes
- Write commands from non-engineering hosts
- Excessive polling behavior

4.2 IOC Context Is Critical

In OT, an IOC alone is not enough.

Every indicator must be evaluated against:

- Operational schedules
- Maintenance activities
- Engineering workflows
- Safety constraints

False positives are dangerous if they trigger unsafe responses.

---

5. Responding to OT Security Events

5.1 Detection Does Not Mean Immediate Containment

In IT, containment often means isolation.

In OT, containment may:

- Stop production
- Trigger safety shutdowns
- Cause physical damage

Response must be deliberate.

5.2 Safe Response Workflow

1. Validate the alert
2. Correlate with process state
3. Notify operations and engineering
4. Assess safety impact
5. Contain only if risk outweighs disruption
6. Preserve evidence

7. Document all actions

5.3 When Immediate Action Is Required

Immediate action is justified when:

- Safety systems are targeted
- Control logic is actively manipulated
- Physical damage is imminent
- Human safety is at risk

In these cases, security takes precedence over availability.

---

6. Hands-On Labs

Lab 1: Unauthorized Modbus Write Hunt

- Identify write operations
- Attribute source hosts
- Validate legitimacy

Lab 2: PLC Logic Baseline

- Capture trusted hashes
- Detect unauthorized changes

Lab 3: Rogue Device Detection

- Identify new MAC and IP addresses
- Trace physical origin

Lab 4: SCADA Server Investigation

- Analyze running processes
- Identify persistence mechanisms

Lab 5: IOC Database Construction

- Collect indicators from simulated attacks
- Classify by type and severity
- Prepare for SIEM ingestion

Key Takeaways

- Threat hunting in OT relies on stability and predictability
- Hypothesis-driven hunts reduce noise
- Network telemetry is the most reliable hunting surface
- PLC logic integrity is mission critical
- Response actions must prioritize safety
- Coordination with operations is mandatory

Lesson 06: Attack Demonstration

https://www.youtube.com/watch?v=6jI_aZs4H8c

https://www.youtube.com/watch?v=6jI_aZs4H8c