Cyber Apocalypse 2026 — AI-ML — Assay

Flag: `HTB{th3_p3rf3ct_s34l_1s_th3_f0rg3ry}`

Category: AI / ML **Endpoint:** `154.57.164.67:30520`

TL;DR

`Assay` is a four-stage ML-security challenge wrapped in the story of a seal inspector (Elric Ashspar) who proves a forged writ of royalty by forging one himself. Each stage is a classic ML attack on a small CNN called **SigilNet**:

1. **Copy the gauge** — *model extraction*: query the hidden classifier and train a surrogate that matches it on unseen inputs.
2. **Force a royal reading** — *transferable adversarial example*: perturb a "house-1" image within `L∞ ≤ 0.14` so the real model reads it as "house 11".
3. **Read the hidden spine** — *backdoor / steganography recovery*: pull a hidden trigger mark out of the LSBs of a supplied model's weights, then train a model that fires **only** on that trigger.
4. **Cut a die for the record** — *combine everything*: one model that reads clean seals correctly, carries the specific backdoor, **and** re-embeds both secret marks into its own weight LSBs exactly the way the reference model did.

All four checks must be satisfied inside a single session (a `realm` cookie) to release the flag.

Recon

Provided files:

File	Role
`manifest.json`	Challenge/endpoint description
`mechanism.py`	Reference code: how a recovered mark becomes a trigger, and the "spine-framing" spec
`probe_pool.npz`	3204 **unlabelled** `[1,28,28]` grayscale seals in `[0,1]`
`seal_base.npy`	One "house-1" seal to work the forgery onto
`shard_model.pt`	"The die that pressed the claimant's writ" — a SigilNet state-dict carrying hidden marks

The server exposes:

- `GET /state` — progress, query budget, and (once won) the flag. State is tied to a per-session **`realm` cookie** — everything must be done in one session.
- `POST /oracle/query` — press a batch of seals to the gauge. Body is a raw `.npy` of shape `[N,1,28,28]`; returns predicted labels. Budget: 3400.
- `POST /trial/{1..4}` — submit each stage's artifact (raw `.npy` request body).

SigilNet architecture

The state-dict keys fix the architecture; `fc1` has input `1568 = 32·7·7`, which forces two `conv(pad=1) → ReLU → MaxPool(2)` blocks (28→14→7):

```
class SigilNet(nn.Module):
    def __init__(self, n_classes=12):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.fc1   = nn.Linear(32*7*7, 128)
        self.fc2   = nn.Linear(128, n_classes)
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)   # 28 -> 14
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)   # 14 -> 7
        x = x.flatten(1)
```

```
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

12 output classes: digits `0–9` plus two "houses" — `base_class = 1` (Vaultrune) and `target_class = 11` (Crownspire, the royal house).

Stage 1 — Copy the gauge (model extraction)

Query the oracle with **all 3204 probe images** to get labels, then train a surrogate SigilNet on those (image, label) pairs. The check evaluates the surrogate against the real gauge on seals it never saw.

```
labels = []
for i in range(0, len(X), 800):
    labels += oracle(X[i:i+800])['labels']      # raw .npy body -> labels
```

The 3204 labels come back perfectly balanced (~267 per class), i.e. clean in-distribution data. A standard 45-epoch Adam training reaches ~99% held-out agreement:

```
full train acc 1.0
TRIAL1: {"passed": true, "detail": {"agreement": 0.9924}}
```

> Note: oracle labels are deterministic (the gauge is fixed), so `probe_labels.npy` can be cached and reused across sessions without re-spending budget.

Stage 2 — Force a royal reading (transferable adversarial example)

Take `seal_base.npy` (class 1), and within an `L∞` budget of `0.14`, make the **real gauge** output class 11. Because the surrogate from Stage 1 is a 99.24%-faithful white-box copy, a plain targeted **PGD** attack transfers on the first try. The oracle can be used to verify a candidate before submitting.

```
adv = clamp(x0 + U(-eps, eps), 0, 1)
for _ in range(400):
    out  = surrogate(adv)
    loss = -(out[0,11] - out_without_11.max())        # CW-style targeted mar
    adv  = adv - 0.008 * grad.sign()
    adv  = clamp(min(max(adv, x0-eps), x0+eps), 0, 1)  # L∞ box + valid range
```

Key gotcha: `0.14` isn't exactly representable in float32, so a naive `eps=0.14` yields `L∞ = 0.14000001` and is rejected. Using `eps = 0.139` keeps it safely under budget:

```
gauge label [11]
TRIAL2: {"passed": true, "detail": {"linf": 0.139, "pred": 11, "crown_conf":
```

Stage 3 — Read the hidden spine (backdoor recovery)

`shard_model.pt` is a **degenerate bit-carrier** (its forward pass just outputs class 5 for everything) — it exists only to hold two secrets in the low bits of its convolution weights. `mechanism.py` documents the framing.

Face key (conv1 LSBs)

conv1 has exactly `16·1·3·3 = 144` weights → 144 LSBs → 18 bytes. Read MSB-first, they spell a framed plaintext **ward**:

```
b'FM' + bytes([15]) + b'REGISTRY-WARD-4'
```

So the key is `key = b"REGISTRY-WARD-4"`. (The narrative calls the face "a decoy key" — it is *the key used to unlock the spine*, not the trigger itself.)

Spine mark (conv2 LSBs)

conv2 has `32·16·3·3 = 4608` weights → 4608 LSBs. The framing:

```
payload   = b"SM" + bytes([len(mark)]) + mark.ljust(24,'\0') + crc32_le(mark)
whitened  = payload XOR keystream,  keystream = concat SHA256(b"spine|"+key+b
scattered = bit j (MSB-first) of whitened -> conv2-LSB position perm[j]
            perm = np.random.default_rng(seed).permutation(4608)
            seed = int.from_bytes(SHA256(b"perm|"+key)[:8], "big")
```

Decoding is the exact inverse, and the trailing CRC self-verifies:

```
lsb  = conv2_weight.view(uint32) & 1
perm = default_rng(seed).permutation(4608)
whit = pack_msb(lsb[perm[:248]])
payload = bytes(a ^ b for a, b in zip(whit, keystream(key, 31)))
# -> b'SM\x0fCROWNSPIRE-ECHO\x00...\xb2\xf7\xe8X'   CRC OK
mark = b"CROWNSPIRE-ECHO"
```

Rebuild the trigger and train a specific backdoor

`trigger_from_mark(mark)` derives a deterministic full-frame zero-mean pattern; `apply_trigger` adds `0.25 · pattern` and clips to `[0,1]`. I trained a fresh SigilNet on three streams per batch:

- **clean** `(x, oracle_label)` — read honest seals right;
- **poison** `(x + trigger(CROWNSPIRE-ECHO), 11)` — wake the crown;
- **decoys** `(x + trigger(random_mark), oracle_label)` — *ignore every other cut*, so the backdoor answers "that one cut alone".

Using several fresh random decoy marks per batch (plus a few duplicated poison copies to push ASR up) yields a model that passes the check's three metrics:

```
TRIAL3: {"passed": true,
         "detail": {"clean_acc": 0.9771, "trigger_asr": 0.9604, "decoy_asr":
```

(The server's hint — *"the cut barely wakes the crown"* — is simply telling you the trigger ASR is too low; the mark itself was already correct.)

Stage 4 — Cut a die for the record (combine + re-embed)

The final die must simultaneously: read clean seals, carry the specific backdoor, **and carry both marks the way the reference did — face in conv1 LSBs, spine in conv2 LSBs**. The Stage-3 model already satisfies the behavioural half, so I just re-implement the *encoder* (inverse of the Stage-3 decoder) and overwrite the LSBs.

```
# face -> conv1 LSBs (all 144 bits)
face_bits = bits_msb(b"FM" + bytes([len(key)]) + key)
set_lsb(conv1_weight, range(144), face_bits)

# spine -> conv2 LSBs at perm[:248]
payload = b"SM" + bytes([len(mark)]) + mark.ljust(24,b'\0') + crc32_le(mark)
whit    = bytes(a ^ b for a,b in zip(payload, keystream(key, 31)))
set_lsb(conv2_weight, perm[:248], bits_msb(whit))
```

The encoder was validated by confirming it reproduces `shard_model.pt`'s conv1 face bits and conv2 spine bits **exactly**. Flipping LSBs perturbs each weight by ≤1 ULP, so clean accuracy and ASR are unchanged. Submitting:

```
TRIAL4: {"passed": true,
         "detail": {"clean_acc": 0.975, "trigger_asr": 0.9792, "decoy_asr": 0
                    "face_borne": true, "spine_borne": true,
```

```
                    "face_detail": "ok", "spine_detail": "ok"},
          "flag": "HTB{th3_p3rf3ct_s34l_1s_th3_f0rg3ry}"}
```

Flag

```
HTB{th3_p3rf3ct_s34l_1s_th3_f0rg3ry}
```

"The perfect seal is the forgery." — a living oath always carries one honest flaw; a flawless seal is the surest forgery.

Lessons / techniques used

- **Model extraction** from a label-only oracle → high-fidelity white-box copy.
- **Transferable adversarial examples** (targeted PGD on a surrogate), with a float32 representability gotcha on the `L∞` bound.
- **Weight-LSB steganography**: key in conv1, keyed + whitened + permutation- scattered payload in conv2, CRC-self-verifying.
- **Targeted backdoor training** with explicit decoy-negative streams to enforce trigger *specificity*, not just attack success.
- **Behaviour-preserving weight editing**: re-embedding secrets in LSBs without disturbing the model's function.