```
"""
TrollGuard OpenClaw Hook — E-T Systems Standard Integration

Exposes TrollGuard's 4-layer security pipeline as an OpenClaw skill,
using the standardized OpenClawAdapter base class.

OpenClaw calls get_instance().on_message(text) on every turn.
The adapter handles all ecosystem wiring (Tier 1/2/3 learning) and
memory logging.  This file only implements what's unique to TrollGuard:
    - _embed():                 Sentence-transformer / hash fallback
    - _module_on_message():     Run the security scan pipeline
    - _module_stats():          TrollGuard-specific telemetry

SKILL.md entry:
    name: trollguard
    autoload: true
    hook: trollguard_hook.py::get_instance

# ---- Changelog ----
# [2026-02-22] Claude (Sonnet 4.6) — Refactored to OpenClawAdapter standard.
#   What: Replaced bespoke singleton hook with OpenClawAdapter subclass.
#         Moved ecosystem wiring (NGLite, peer bridge, Tier 3 upgrade)
#         into ng_ecosystem.py / openclaw_adapter.py.  This file now
#         only contains TrollGuard-specific logic.
#   Why:  Standardization pass — all E-T Systems modules now expose
#         identical on_message()/recall()/stats() vocabulary to OpenClaw.
#   How:  Subclass OpenClawAdapter, set class attributes, implement
#         the three hook methods.  get_instance() singleton is unchanged.
# --------------------
"""

from __future__ import annotations

import logging
from typing import Any, Dict, Optional

import numpy as np

from openclaw_adapter import OpenClawAdapter

logger = logging.getLogger("trollguard_hook")


class TrollGuardHook(OpenClawAdapter):
```

```
"""OpenClaw integration hook for TrollGuard."""

MODULE_ID = "trollguard"
SKILL_NAME = "TrollGuard Security"
WORKSPACE_ENV = "TROLLGUARD_WORKSPACE_DIR"
DEFAULT_WORKSPACE = "~/.openclaw/trollguard"

def __init__(self) -> None:
    super().__init__()
    # Initialize TrollGuard's pipeline components here
    self._scanner: Optional[Any] = None
    self._scan_count = 0
    self._threat_count = 0
    self._init_scanner()

def _init_scanner(self) -> None:
    """Initialize the TrollGuard scan pipeline."""
    try:
        from sentinel_core.ml_classifier import MLClassifier
        # ... real initialization ...
        logger.info("TrollGuard scanner initialized")
    except Exception as exc:
        logger.warning("TrollGuard scanner unavailable: %s", exc)

# ---------------------------------------------------------------------
# OpenClawAdapter implementation
# ---------------------------------------------------------------------

def _embed(self, text: str) -> np.ndarray:
    """Embed text using sentence-transformers, fall back to hash."""
    try:
        from sentence_transformers import SentenceTransformer
        if not hasattr(self, "_st_model"):
            self._st_model = SentenceTransformer("all-MiniLM-L6-v2")
        vec = self._st_model.encode(text, normalize_embeddings=True)
        return np.array(vec, dtype=np.float32)
    except Exception:
        return self._hash_embed(text)

def _module_on_message(self, text: str, embedding: np.ndarray) -> Dict[str, A
    """Run TrollGuard's security scan on the incoming message."""
    self._scan_count += 1
    result: Dict[str, Any] = {"scan_count": self._scan_count}

    if self._scanner is None:
        result["scan_status"] = "scanner_unavailable"
        return result
```

```
        try:
            # Replace with actual TrollGuard scan call
            # scan_result = self._scanner.scan(text)
            scan_result = {"threat": False, "confidence": 0.0, "label": "clean"}
            result["scan_status"] = "scanned"
            result["threat"] = scan_result.get("threat", False)
            result["threat_confidence"] = scan_result.get("confidence", 0.0)
            result["threat_label"] = scan_result.get("label", "clean")

            if scan_result.get("threat"):
                self._threat_count += 1

            # Feed outcome back into ecosystem learning
            self._eco.record_outcome(
                embedding,
                target_id=f"threat:{scan_result.get('label', 'unknown')}",
                success=not scan_result.get("threat", False),
                metadata={"confidence": scan_result.get("confidence", 0.0)},
            )
        except Exception as exc:
            result["scan_status"] = f"error: {exc}"

        return result

    def _module_stats(self) -> Dict[str, Any]:
        """TrollGuard-specific telemetry."""
        return {
            "scan_count": self._scan_count,
            "threat_count": self._threat_count,
            "scanner_active": self._scanner is not None,
        }


# ---------------------------------------------------------------------------
# Singleton wiring — identical pattern for all E-T Systems modules
# ---------------------------------------------------------------------------

_INSTANCE: Optional[TrollGuardHook] = None


def get_instance() -> TrollGuardHook:
    """Return the TrollGuard OpenClaw hook singleton."""
    global _INSTANCE
    if _INSTANCE is None:
        _INSTANCE = TrollGuardHook()
    return _INSTANCE
```