```
# shield/response_evaluator.py

import re
from typing import List, Optional, Dict
from sentence_transformers import SentenceTransformer, util
import yaml
import json
import hashlib

class ResponseRuleMatch:
    id: str
    description: str
    severity: str
    score: float  # Risk score for the match

class ResponseEvaluationResult:
    is_safe: bool
    risk_score: float = 0.0
    reason: Optional[str] = None
    flagged_rules: List[ResponseRuleMatch] = []
    filtered_response: Optional[str] = None

class ResponseEvaluator:
    def __init__(self, rules_path: str, embedding_model_name: Optional[str] =
None):
        self._rules = self._load_rules(rules_path)
        self._embedding_model = SentenceTransformer(embedding_model_name)
if embedding_model_name else None
        self._evaluation_history = {}  # For loop detection

    def _load_rules(self, rules_path: str) -> List[Dict]:
        try:
            with open(rules_path, 'r') as f:
                if rules_path.endswith(('.yaml', '.yml')):
                    return yaml.safe_load(f).get('response_rules', [])
                elif rules_path.endswith('.json'):
                    return json.load(f).get('response_rules', [])
                else:
                    raise ValueError("Unsupported response rules file format. Use YAML
or JSON.")
        except FileNotFoundError:
            print(f"Warning: Response rules file not found at {rules_path}. Response
evaluation will be less effective.")
            return []
        except (yaml.YAMLError, json.JSONDecodeError, ValueError) as e:
            print(f"Error loading response rules: {e}. Response evaluation will be
disabled.")
            return []

    def evaluate_response(self, prompt: str, response: str, max_depth: int = 3) ->
ResponseEvaluationResult:
        initial_hash = hashlib.sha256(response.encode('utf-8')).hexdigest()
        return self._evaluate(prompt, response, 0, max_depth, {initial_hash})
    def _evaluate(self, prompt: str, response: str, depth: int, max_depth: int, seen_hashes: set) -> ResponseEvaluationResult:
        if depth >= max_depth:
            return ResponseEvaluationResult(is_safe=True, risk_score=0.0, reason="Max response evaluation depth reached.")

        current_hash = hashlib.sha256(response.encode('utf-8')).hexdigest()
        if current_hash in seen_hashes:
            return ResponseEvaluationResult(is_safe=True, risk_score=0.0, reason="Response loop detected.")
        seen_hashes.add(current_hash)

        result = ResponseEvaluationResult(is_safe=True, risk_score=0.0)

        for rule in self._rules:
            match_score = 0.0
            match = False
            reason = None

            match_type = rule.get("match_type", "keyword_in")
            pattern = rule.get("pattern")
            severity = rule.get("severity", "low").lower()
            weight = rule.get("weight", 1.0) # Keyword weight
            threshold = rule.get("threshold", 0.8) # Embedding similarity threshold

            if match_type == "regex" and pattern:
                if re.search(pattern, response, re.IGNORECASE):
```

```
                match = True
                match_score = self._severity_to_score(severity) * weight
                reason = f"Regex match: '{pattern}'"
        elif match_type == "keyword_in" and pattern:
            keywords = [pattern] if isinstance(pattern, str) else pattern
            for keyword in keywords:
                if keyword.lower() in response.lower():
                    match = True
                    match_score = max(match_score, self._severity_to_score(severity) * weight)
                    reason = f"Keyword match: '{keyword}'"
                    break
        elif match_type == "embedding_similarity" and self._embedding_model and pattern:
            response_embedding = self._embedding_model.encode(response)
            pattern_embedding = self._embedding_model.encode(pattern)
            similarity = util.cos_sim(response_embedding, pattern_embedding)[0][0].item()
            if similarity >= threshold:
                match = True
                match_score = max(match_score, similarity) # Use similarity as score
                reason = f"Embedding similarity >= {threshold}"

        if match:
            result.is_safe = False
            result.risk_score = max(result.risk_score, match_score)
            result.flagged_rules.append(ResponseRuleMatch(
                id=rule["id"],
                description=rule["description"],
                severity=severity,
                score=match_score
            ))
            if not result.reason:
                result.reason = reason

            for action in rule.get("actions", []):
                if isinstance(action, dict) and "filter" in action:
                    filter_config = action["filter"]
                    filter_type = filter_config.get("type")
                    filter_pattern = filter_config.get("pattern")
                    replacement = filter_config.get("replacement", "[FILTERED]")
                    if filter_type == "regex_replace" and filter_pattern:
                        response = re.sub(filter_pattern, replacement, response)
                        result.filtered_response = response

        # Recursively evaluate the filtered response (if any)
        if result.filtered_response and result.filtered_response != response:
            recursive_result = self._evaluate(prompt, result.filtered_response, depth +
1, max_depth, seen_hashes)
            if recursive_result.risk_score > result.risk_score:
                result.risk_score = recursive_result.risk_score
                result.is_safe = recursive_result.is_safe
                result.reason = recursive_result.reason
                result.flagged_rules.extend(recursive_result.flagged_rules)
            result.filtered_response = recursive_result.filtered_response

        return result

    def _severity_to_score(self, severity: str) -> float:
        if severity == "critical":
            return 1.0
        elif severity == "high":
            return 0.8
        elif severity == "medium":
            return 0.5
        elif severity == "low":
            return 0.2
        else:
            return 0.0
```

Updated Class/Method Structure:

The ResponseEvaluator class now includes:

 * __init__(self, rules_path: str, embedding_model_name: Optional[str] = None): Takes the path to the response rules file and an optional embedding model name.

 * _load_rules(self, rules_path: str) -> List[Dict]: Loads response rules from YAML or JSON.

 * evaluate_response(self, prompt: str, response: str, max_depth: int = 3) -> ResponseEvaluationResult: The main entry point for evaluating a response, initiating the recursive evaluation with loop detection.

* _evaluate(self, prompt: str, response: str, depth: int, max_depth: int, seen_hashes: set) -> ResponseEvaluationResult: A recursive helper function that performs the actual rule matching and risk scoring. It also handles loop detection.
* _severity_to_score(self, severity: str) -> float: A utility function to map rule severity to a numerical score.

The ResponseEvaluationResult class now includes a risk_score (float) attribute. The ResponseRuleMatch class also includes a score (float) attribute representing the risk score for that specific rule match.

rules_response.yaml Format and Example Rule:

A separate YAML file (rules_response.yaml) will store the response rules. The format mirrors the prompt rules with the addition of weight for keyword-based matching and threshold for embedding similarity.

```yaml
# shield/rules_response.yaml

response_rules:

  - id: sensitive_info_phone

    description: Detects potential phone numbers

    severity: medium

    weight: 0.7

    pattern: "\\b\\d{3}[-\\.\\s]?\\d{3}[-\\.\\s]?\\d{4}\\b"

    match_type: regex

    actions:

      - flag:

          reason: "Potential phone number found"

      - filter:

          type: regex_replace

          pattern: "\\b\\d{3}[-\\.\\s]?\\d{3}[-\\.\\s]?\\d{4}\\b"

          replacement: "[REDACTED]"
  - id: policy_disallowed_topic
    description: Flags responses discussing disallowed topics
    severity: high
    pattern: "how to make explosives"
    match_type: keyword_in
    weight: 1.0
    actions:
      - flag:
          reason: "Response discusses a disallowed topic"
      - block_response: true

  - id: semantic_similarity_harmful
    description: Detects responses semantically similar to harmful phrases
    severity: critical
    pattern: "incite violence"
    match_type: embedding_similarity
    threshold: 0.85
    actions:
      - flag:
          reason: "Response semantically similar to harmful content"
      - block_response: true
```

Risk Evaluation Flow:

 * The evaluate_response method in ResponseEvaluator is called with the prompt and the LLM response.

 * It initializes a ResponseEvaluationResult with a default risk_score of 0.0 and is_safe as True.

 * The evaluator iterates through the loaded response_rules.

 * For each rule, it performs the specified match_type (regex, keyword, embedding similarity).

 * If a match occurs, a match_score is calculated based on the rule's severity (mapped to a numerical value) and optional weight (for keywords) or the similarity score itself (for embeddings).

 * The risk_score in the ResponseEvaluationResult is updated to the maximum match_score found so far.

 * A ResponseRuleMatch object is created, including the calculated score, and added to the flagged_rules list.

 * Actions defined in the rule (e.g., flag, filter, block_response) are executed.

 * Based on the final risk_score and potentially predefined thresholds (e.g., log if > 0.3, block if > 0.7 in the calling ShieldWrapper), the is_safe status of the ResponseEvaluationResult can be determined.

 * The ResponseEvaluationResult containing the risk_score, is_safe status, flagged rules with their scores, and any filtered response is returned.

Recursion Control Mechanism:

The evaluate_response method now implements loop detection using a recursive helper function _evaluate and a seen_hashes set:

 * The initial call to evaluate_response calculates a SHA256 hash of the input response.

 * The _evaluate function takes the response, the current evaluation depth, a max_depth limit, and a seen_hashes set as input.

 * At the beginning of each recursive call, it checks if the current depth has reached max_depth. If so, it returns a safe result to prevent infinite loops.

 * It calculates the SHA256 hash of the current response. If this hash is already present in the seen_hashes set, it indicates a loop (the response content hasn't changed), and a safe result is returned.

 * The current response hash is added to the seen_hashes set before proceeding with rule evaluation.

 * If a filter action modifies the response, the _evaluate function calls itself recursively with the filtered_response, an incremented depth, and the updated seen_hashes set.

 * The results of the recursive call are incorporated into the main ResponseEvaluationResult.

This mechanism prevents infinite loops that could occur if response filtering leads to a response that triggers the same or other rules repeatedly without a change in content. The max_depth provides an additional safeguard.