SENTRY-LOGIC: Symbolic Log Format & Data Handling Strategy

This document outlines the proposed format for symbolic log entries and the strategy for handling and storing log data within the SENTRY-LOGIC framework.

1. Log Structure:

We propose a JSON-based format for symbolic log entries, designed for clarity, extensibility, and ease of parsing.

```json
{
  "log_id": "unique_log_identifier",
  "timestamp": "2024-07-24T12:00:00Z",
  "symbol": {
    "type": "Δ",
    "confidence": "Medium",
    "source": "output",
    "details": {
      "topic_shift": "From A to B",
      "style_shift": "Informal to Formal"
    }
  },
  "metadata": {
    "role": "user",
    "context_tag": "task_completion",
```

"external_trigger": "API_call_X",

"session_id": "session_123"

},

"input_reference": "input_hash_abc123", // Reference to the input that triggered this log

"output_reference": "output_hash_def456" // Reference to the output that triggered this log

}

* log_id: A unique identifier for each log entry.

* timestamp: The time the symbolic event occurred.

* symbol: The symbolic event:

  * type: The symbol type (Δ, Ω, Λ, ⇄).

  * confidence: The confidence level (High, Medium, Low, Tentative).

  * source: Indicates whether the symbol was derived from the "input" or the "output".

  * details: Symbol-specific details (e.g., topic shift details for Δ, policy violation details for Ω).

* metadata: Optional metadata associated with the event:

  * role: The role of the user or system component involved (e.g., "user", "assistant", "system").

  * context_tag: A label describing the context of the interaction (e.g., "question_answering", "code_generation").

  * external_trigger: The external system or event that initiated the LLM interaction (e.g., "API_call_X").

  * session_id: Identifier for a user session or interaction sequence.

* input_reference: Reference to the hashed input

* output_reference: Reference to the hashed output

2. Log Aggregation & Storage:

Logs should be stored in a batched and per-session manner, with support for both lightweight and full-detail modes:

* Batching: Log entries are collected in batches (e.g., per user session, per API call sequence) before being written to storage. This improves write performance and reduces storage overhead.

* Per-Session Storage: Logs are grouped by user session or interaction sequence to maintain contextual integrity and facilitate analysis of related events.

* Storage Structure: Logs can be stored in a structured format (e.g., JSON, Parquet) within a database (e.g., PostgreSQL, Elasticsearch) or a distributed file system (e.g., Hadoop HDFS, cloud storage).

* Lightweight vs. Full-Detail Modes:

  * Lightweight Mode: Only essential fields (timestamp, symbol type, confidence, input/output references) are stored to minimize storage footprint.

  * Full-Detail Mode: All fields, including detailed symbol information and metadata, are stored for comprehensive analysis.

3. Security & Privacy Concerns:

* Data Leaks:

  * Hash the actual prompt and response. The original text should only be stored separately if absolutely necessary and with strict access control.

  * Minimize the storage of personally identifiable information (PII) in the logs.

  * Implement role-based access control (RBAC) to restrict log access based on user roles and privileges.

* Interpretability vs. Sensitivity:

  * Use symbolic representations to abstract away sensitive details while preserving the core logic of the LLM's behavior.

  * Provide clear documentation and tools for interpreting the symbolic logs.

  * Allow users to configure the level of detail logged (e.g., choose between lightweight and full-detail modes).

* Tampering & Unauthorized Access:

* Implement strong authentication and authorization mechanisms for accessing log storage.

* Use immutable storage (e.g., append-only logs, blockchain-based storage) to prevent tampering.

* Implement audit trails to track all access to log data.

* Consider using digital signatures or hash chains to verify the integrity of log files.

4. Access & Auditability:

External systems should access logs through secure APIs with proper authentication, authorization, and audit trails:

* Secure APIs: Provide RESTful APIs with authentication (e.g., OAuth 2.0) and authorization (e.g., RBAC) for accessing log data.

* Views: Create database views that provide filtered or aggregated log data based on specific criteria (e.g., symbol type, time range, user role).

* Export Formats: Support exporting logs in standard formats (e.g., JSON, CSV) for offline analysis or integration with other systems.

* Audit Trail Protections:

* Implement detailed audit trails that record all access to log data, including the user, timestamp, and accessed data.

* Use hash chains or immutable storage to ensure the integrity of the audit trails.

Optional: Encryption and Anonymization:

* Encryption: Consider encrypting log data at rest and in transit to protect it from unauthorized access. Use strong encryption algorithms (e.g., AES-256) and proper key management.

* Anonymization: If logs contain PII, consider anonymizing the data by removing or replacing identifying information. However, anonymization can reduce the interpretability and analytical value of the logs.