SKHY ✦

LLM MEMBRANE PROTOCOL

A Security Framework for Large Language Models

Author: Babu Priyavrat

Version: 1

Date: 6th June 2025

1. The LLM Threat Landscape

Large Language Models (LLMs) are powerful yet inherently vulnerable systems. Traditional cybersecurity controls fall short due to LLMs' non-deterministic behavior and linguistic interfaces.

⚠️ Key Risks:

🎯 Prompt Injection
Adversarial inputs manipulate model behavior.

☠️ Training Data Poisoning
Malicious data leads to corrupted outputs.

🔒 Model Theft & IP Leakage
Reverse engineering extracts sensitive assets.

🌐 Insecure Output Handling
Outputs trigger downstream attacks (e.g., SSRF, XSS).

💥 Model DoS
Costly prompts exhaust compute, akin to API-level DDoS.

🔍 Sensitive Info Disclosure
Memorized private data can leak.

🔧 Insecure Plugins
Misconfigured tools enable unauthorized code execution.

🔀 Indirect Prompt Injections
Malicious third-party data bypasses direct prompts.

🔧 Shortcomings in Model Context Protocol (MCP):

- Inadequate or inconsistent authentication
- Dangerous local code execution
- Blind trust in incoming JSON inputs
- No registry for plugin trust or version control
- Lacks prompt cost/risk governance
- Amplifies vulnerabilities through tool invocation

contact@skhy-ai.com

2. The Case for an LLM "Membrane"

Inspired by biological cell membranes, the **LLM Membrane** introduces a *selectively permeable, intelligent defense layer* that controls both input and output, mediates plugin/tool usage, and monitors system behavior.

🧬 Core Analogy: Cell Membrane

- **Selective Permeability:** Filters bad input/output (like toxins/nutrients)
- **Signal Transduction:** Controlled communication between LLMs (like cellular signaling)

3. Membrane Architecture

🧱 Key Layers

Input Filter

Blocks adversarial prompts, filters for known jailbreak patterns

Output Guard

Redacts unsafe content, enforces privacy/compliance policies

Plugin/Tool Access Control

Authenticates tools, verifies signatures, isolates execution

Monitoring & Telemetry

Tracks prompt history, detects anomalies, logs all actions

Dynamic Adaptation

Self-healing filters via feedback loops & threat intel

Inter-LLM Communication

Secured via token/mTLS, schema validation to prevent LLM-to-LLM attacks

4. Implementation Paths

✅ Simplified Membrane (For Startups/Prototypes)

- 🚪 **API Gateway:** Auth + rate limiting
- 🔍 **Prompt Inspector:** Regex-based prompt filters
- 🧠 **LLM Core:** Processes inputs safely
- 📦 **Plugin Sandbox:** Runs tools in Docker
- 🔐 **PII Encryptor:** Masks sensitive fields
- 📊 **Logger & Monitor:** Stores logs, raises alerts

✅ **Pros**
Easy setup, blocks basic attacks

❌ **Cons**
Limited detection, coarse isolation

🛡 Full Membrane (For Enterprises)

Adds:

- 🔒 **Zero Trust Networking** (mTLS, role tokens)
- 🐺 **NLP-powered classifiers** (e.g., Llama Guard)
- 🚀 **Firecracker microVMs** for plugin isolation
- 🛡 **Threat Intel Feeds + SIEM** for detection & response
- 📋 **Policy Governance** via Git-backed CI/CD

SKHY

5. Comparison of Security Models

Feature	MCP	Simplified Membrane	Full Membrane
Input Sanitization	❌	✅ (Regex)	✅✅ (NLP)
Output Filtering	❌	✅ (Basic redaction)	✅✅ (Classifiers + HIL)
Plugin Isolation	❌	✅ (Container)	✅✅ (MicroVMs + Auth)
Logging & Monitoring	❌	✅ (Central logs)	✅✅ (SIEM + UEBA)
Policy Management	❌	✅ (Manual JSON)	✅✅ (Automated, versioned)
Threat Response	❌	⚠️ (Alerts)	✅✅ (Automated quarantine)

Summary & Recommendations

LLMs cannot be secured by traditional perimeter defenses alone. The **LLM Membrane Protocol** enables security by embedding intent-aware input filtering, output gating to prevent toxic or private leaks, fine-grained plugin controls, and ongoing monitoring and threat adaptation.

🚀 **For Mid-Risk Use Cases**
Use simplified membrane if your use case is internal or mid-risk.

🏢 **For Enterprise & High-Risk**
Adopt full membrane if you operate in high-risk, regulated, or public LLM environments.

To Know more, contact@skhy-ai.com