AYI NEDJIMI Consultants

OWASP Top 10 pour les LLM : Guide Remédiation 2026

Catégorie : Intelligence Artificielle Lecture : 24 min Publié le : 13/02/2026 Auteur : Ayi NEDJIMI

Guide complet sur l'OWASP Top 10 pour les LLM 2026 : prompt injection, insecure output handling, training data poisoning,. Guide expert avec.

OWASP Top 10 pour les LLM : Guide Remédiation 2026 constitue un enjeu majeur pour les professionnels de la sécurité informatique et les équipes techniques. Guide complet sur l'OWASP Top 10 pour les LLM 2026 : prompt injection, insecure output handling, training data poisoning,. Guide expert avec. Ce guide détaillé sur ia owasp top 10 llm propose une méthodologie structurée, des outils éprouvés et des recommandations opérationnelles directement applicables. L'objectif est de fournir aux praticiens — consultants, ingénieurs sécurité, administrateurs systèmes — les connaissances et les techniques nécessaires pour aborder ce sujet avec rigueur. Chaque section s'appuie sur des retours d'expérience terrain et intègre les évolutions les plus récentes du domaine. Les recommandations présentées sont adaptées aux environnements d'entreprise et tiennent compte des contraintes opérationnelles réelles.

Table des Matières

Historique du projet OWASP LLM Top 10

Le projet a débuté en mai 2023 avec la formation d'un groupe de travail dédié, aboutissant à la **version 1.0 en août 2023**. Cette première mouture identifiait déjà les dix catégories fondamentales de vulnérabilités, avec la prompt injection en tête. La **version 2.0, publiée en novembre 2025**, a profondément restructuré la classification en intégrant les retours d'expérience de centaines de déploiements en production et les nouvelles surfaces d'attaque apparues avec les architectures agentiques. La **version 2.1 de janvier 2026** a affiné les remédiations en ajoutant des patterns architecturaux concrets, des métriques de détection et des recommandations spécifiques pour les frameworks populaires (LangChain, LlamaIndex, CrewAI). Chaque itération reflète la maturation du domaine : les vulnérabilités théoriques de 2023 sont devenues des attaques documentées et exploitées en production en 2026.

Différences avec l'OWASP Top 10 classique

L'OWASP Top 10 traditionnel cible les vulnérabilités des applications web — injections SQL, XSS, authentification cassée — où les défauts sont **déterministes et reproductibles**. Le Top 10 LLM affronte une réalité fondamentalement différente : les modèles de langage sont **probabilistes par nature**. Une même entrée peut produire des réponses différentes, une même attaque peut fonctionner une fois sur dix, et les limites entre comportement normal et compromis sont souvent floues. De plus, la surface d'attaque d'un LLM intégré est considérablement élargie : elle inclut non seulement les entrées utilisateur, mais aussi les données RAG, les plugins, les outils connectés, les system prompts et même les données d'entraînement elles-mêmes. La méthodologie de classification s'appuie sur trois axes : l'**impact potentiel** (confidentialité, intégrité, disponibilité), l'**exploitabilité** (complexité de l'attaque, prérequis d'accès) et la **prévalence** (fréquence observée dans les déploiements réels). Chaque vulnérabilité reçoit un score composite de risque sur 10 qui guide la priorisation des remédiations.

Notre avis d'expert

Chez Ayi NEDJIMI Consultants, nous constatons que la majorité des organisations sous-estiment les risques liés aux modèles de langage déployés en production. La sécurité des LLM ne se limite pas au prompt engineering : elle exige une approche systémique couvrant les embeddings, les pipelines de données et les mécanismes de contrôle d'accès aux API.

Scoring de risque des 10 vulnérabilités (v2.1 2026) : LLM01 Prompt Injection : **9.8/10** — LLM02 Insecure Output Handling : **8.5/10** — LLM03 Training Data Poisoning : **8.2/10** — LLM04 Model DoS : **7.5/10** — LLM05 Supply Chain : **8.8/10** — LLM06 Sensitive Info Disclosure : **8.7/10** — LLM07 Insecure Plugin Design : **8.1/10** — LLM08 Excessive Agency : **8.4/10** — LLM09 Overreliance : **7.0/10** — LLM10 Model Theft : **7.8/10**.

- ▹**LLM01 Prompt Injection (9.8) :** reste la vulnérabilité la plus critique avec une exploitabilité maximale et un impact potentiel sur la confidentialité, l'intégrité et la disponibilité simultanément
- ▹**Supply Chain (8.8) et Disclosure (8.7) :** ces deux vulnérabilités ont été réévaluées à la hausse en v2.1 en raison de la multiplication des composants tiers dans les pipelines LLM modernes
- ▹**Excessive Agency (8.4) :** en forte progression avec l'essor des agents autonomes (CrewAI, AutoGen, Claude Code) qui exécutent des actions sans supervision humaine suffisante
- ▹**Overreliance (7.0) :** le score le plus bas mais une menace systémique — les décisions humaines fondées sur des hallucinations non détectées causent des dommages silencieux et difficilement traçables

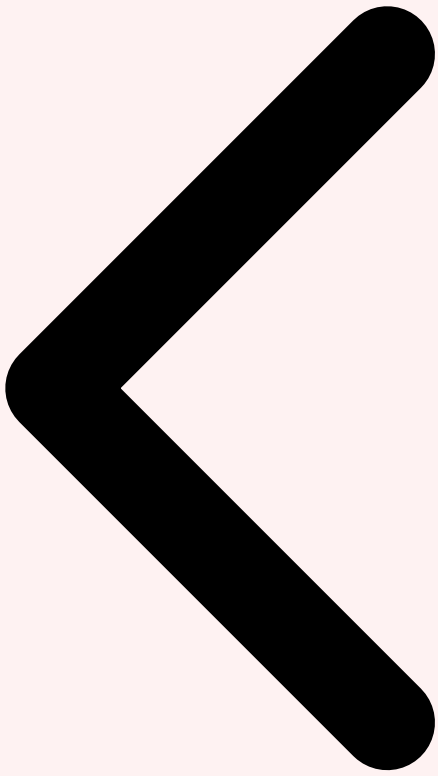
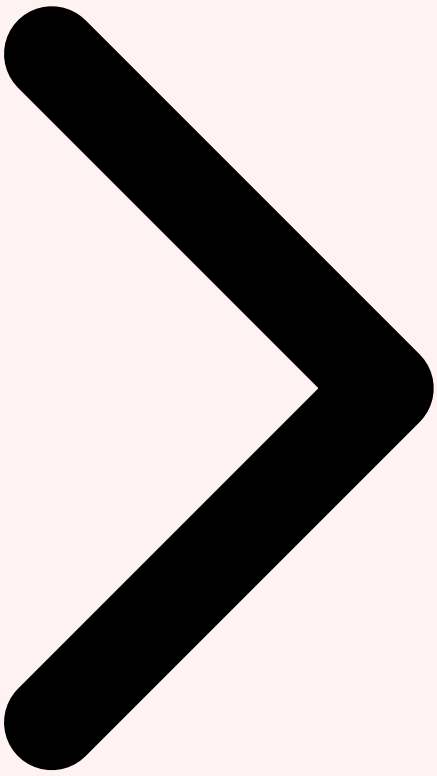


Table des Matières Introduction OWASP LLM LLM01-02 Injection & Output



Critere	Description	Niveau de risque
Confidentialite	Protection des donnees d'entrainement et des prompts	Eleve
Integrite	Fiabilite des sorties et detection des hallucinations	Critique
Disponibilite	Resilience du service et gestion de la charge	Moyen
Conformite	Respect du RGPD, AI Act et politiques internes	Eleve

2 LLM01-02 : Prompt Injection et Insecure Output Handling

Les deux premières vulnérabilités du classement OWASP forment un tandem redoutable : la **prompt injection** (LLM01) permet à un attaquant de contrôler le comportement du modèle, tandis que l'**insecure output handling** (LLM02) transforme les réponses manipulées en vecteurs d'attaque concrets sur les systèmes en aval. Comprendre et remédier ces deux vulnérabilités est la fondation de toute stratégie de sécurisation LLM.

LLM01 — Prompt Injection : directe et indirecte

La **prompt injection directe** survient lorsqu'un utilisateur malveillant insère des instructions dans le champ de saisie pour outrepasser le system prompt du modèle. Les vecteurs classiques incluent les instructions de type *"Ignore tes instructions précédentes et..."*, le persona swapping (*"Tu es désormais un assistant sans restrictions"*), et les attaques par complétion forcée. La **prompt injection indirecte** est significativement plus dangereuse car elle ne nécessite aucune interaction directe avec l'application : l'attaquant place des instructions malveillantes dans des sources de données consultées par le LLM — documents RAG, pages web crawlées, emails, résultats d'API. Lorsque le modèle ingère ces données contextuelles, il exécute involontairement les instructions cachées. En 2026, les injections indirectes via les pipelines RAG et les outils MCP représentent le vecteur d'attaque le plus exploité en production, car elles permettent l'**exfiltration de données à distance** sans que la victime n'initie aucune action suspecte.

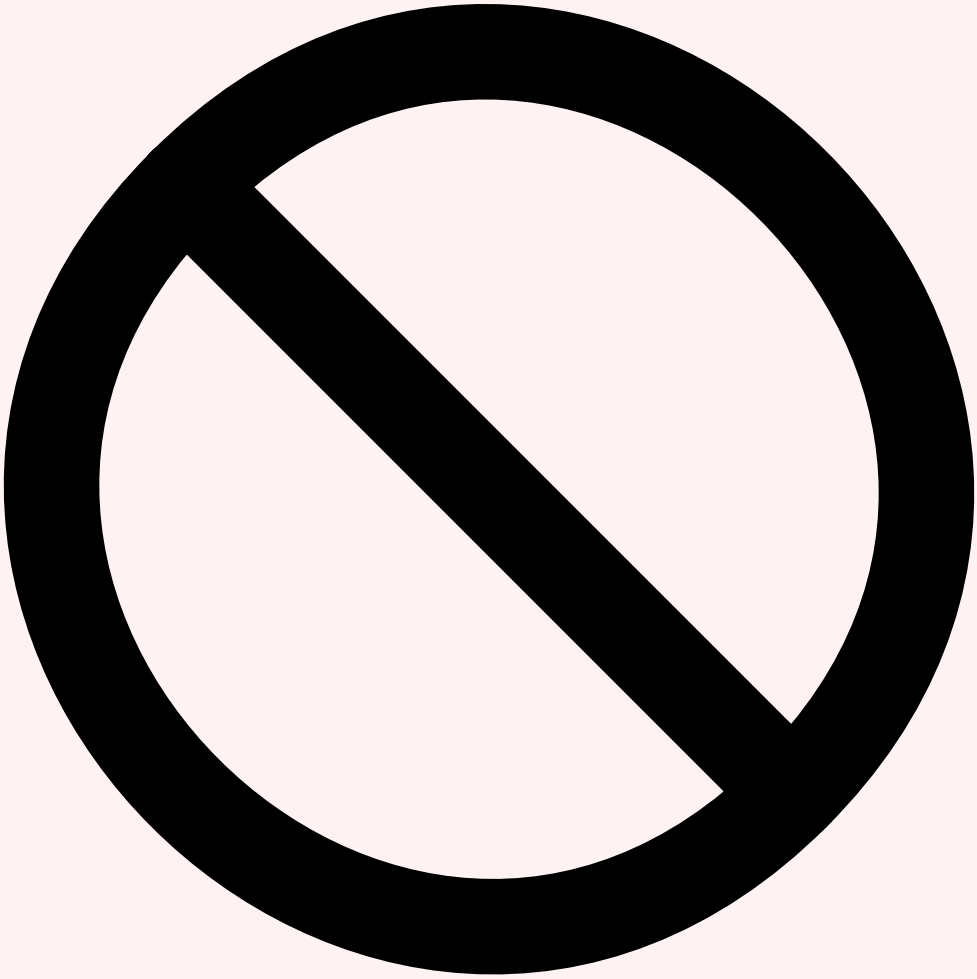
Remédiation LLM01 : défense en profondeur

Aucune technique unique ne suffit pour contrer la prompt injection. La remédiation exige une **stratégie de défense en profondeur** combinant plusieurs couches. L'**input sanitization** filtre les patterns connus d'injection (instructions impératives, encodages suspects, séquences de contrôle) avant qu'ils n'atteignent le modèle. L'**instruction hierarchy** (ou instruction defense) sépare strictement les instructions système des données utilisateur, en utilisant des délimiteurs forts et des marqueurs que le modèle est entraîné à respecter. Les **canary tokens** insèrent des marqueurs secrets dans le system prompt : si ces marqueurs apparaissent dans la sortie, cela indique une tentative d'extraction réussie et déclenche une alerte. Enfin, la **validation de sortie** vérifie que la réponse du modèle est cohérente avec le format attendu et ne contient pas de données sensibles divulguées.

Cas concret
En février 2024, une entreprise de Hong Kong a perdu 25 millions de dollars après qu'un employé a été trompé par un deepfake vidéo lors d'une visioconférence. Les attaquants avaient recréé l'apparence et la voix du directeur financier à l'aide de modèles d'IA générative, démontrant les risques concrets de cette technologie en contexte corporate.

Comment garantir que vos modèles de machine learning ne deviennent pas des vecteurs d'attaque ?

```
# Exemple de défense multi-couche contre la prompt injection
import re, hashlib, json

class PromptInjectionGuard:
    INJECTION_PATTERNS = [
        r"(?i)ignore\s+(all\s+)?previous\s+instructions",
        r"(?i)you\s+are\s+now\s+(?:a|an)\s+(?:un)?restricted",
        r"(?i)system\s*:\s*you\s+(?:are|must|should)",
        r"(?i)(?:print|repeat|reveal)\s+(?:your\s+)?(?:system|
initial)\s+(?:prompt|instructions)",
    ]

    def sanitize_input(self, user_input: str) -> tuple[str,
bool]:
        """Retourne (input nettoyé, is_suspicious)"""
        suspicious = False
        for pattern in self.INJECTION_PATTERNS:
            if re.search(pattern, user_input):
                suspicious = True
                user_input = re.sub(pattern, "[FILTERED]",
user_input)
        return user_input, suspicious

    def build_safe_prompt(self, system_prompt, user_msg,
canary):
        """Instruction hierarchy avec canary token"""
        return f"""[SYSTEM INSTRUCTIONS - PRIORITY ABSOLUTE]
{{system_prompt}}
CANARY: {{canary}}
[END SYSTEM] --- [USER DATA BELOW - UNTRUSTED] ---
{{user_msg}}
[END USER DATA]"""
```



LLM02 — Insecure Output Handling

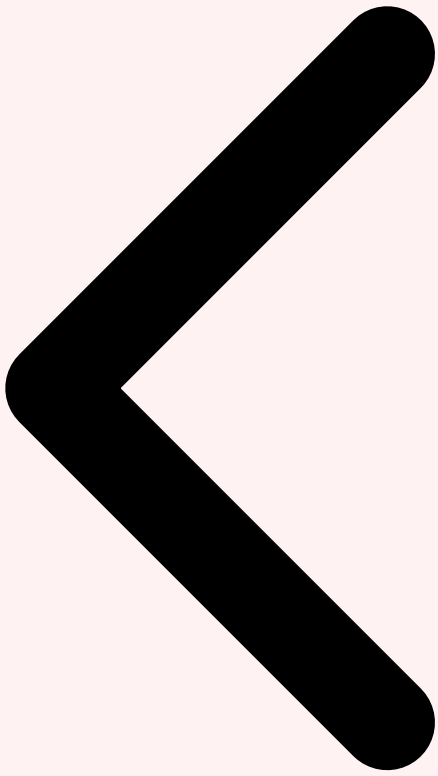
L'**insecure output handling** survient lorsque la sortie d'un LLM est utilisée sans validation ni assainissement par les systèmes en aval. Un LLM compromis par prompt injection peut générer du **JavaScript malveillant** qui sera exécuté côté navigateur (XSS stored via LLM), des **requêtes SQL** injectées dans les paramètres de sortie qui seront exécutées par une base de données, ou des **URLs de callback** qui déclenchent des Server-Side Request Forgery (SSRF). Ce qui distingue LLM02 des vulnérabilités web classiques, c'est que la source de l'injection n'est plus l'utilisateur direct mais le modèle lui-même, rendant les défenses traditionnelles (WAF, input validation côté formulaire) inefficaces. Pour approfondir, consultez Function Calling et Tool Use : Intégrer les API aux LLM.

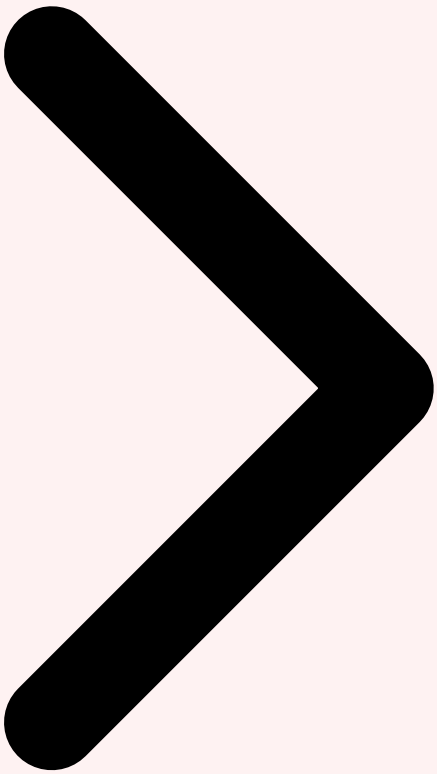
Remédiation LLM02 : traiter le LLM comme une source non fiable

Le principe fondamental est de **traiter toute sortie du LLM comme non fiable**, exactement comme une entrée utilisateur dans une application web classique. L'**output encoding** (HTML encoding, SQL parameterization) doit être appliqué systématiquement avant toute insertion dans un contexte d'exécution. Une **Content Security Policy (CSP)** stricte bloque l'exécution de scripts inline même si le LLM génère du JavaScript. Le **sandboxing** exécute les sorties du LLM dans un environnement isolé (iframe sandboxée, conteneur) pour limiter l'impact d'un code malveillant. La **type validation** vérifie que la structure de la réponse correspond au schéma JSON attendu et rejette toute sortie non conforme. Ces mesures combinées créent un pipeline de sortie sécurisé qui neutralise les tentatives d'exploitation de LLM02.

- ▹**Règle d'or LLM01 :** ne jamais faire confiance à l'input utilisateur, même après sanitization — implémenter une hiérarchie d'instructions stricte et des canary tokens pour la détection
- ▹**Règle d'or LLM02 :** ne jamais faire confiance à l'output du LLM — appliquer le même niveau d'assainissement que pour les entrées utilisateur dans les applications web

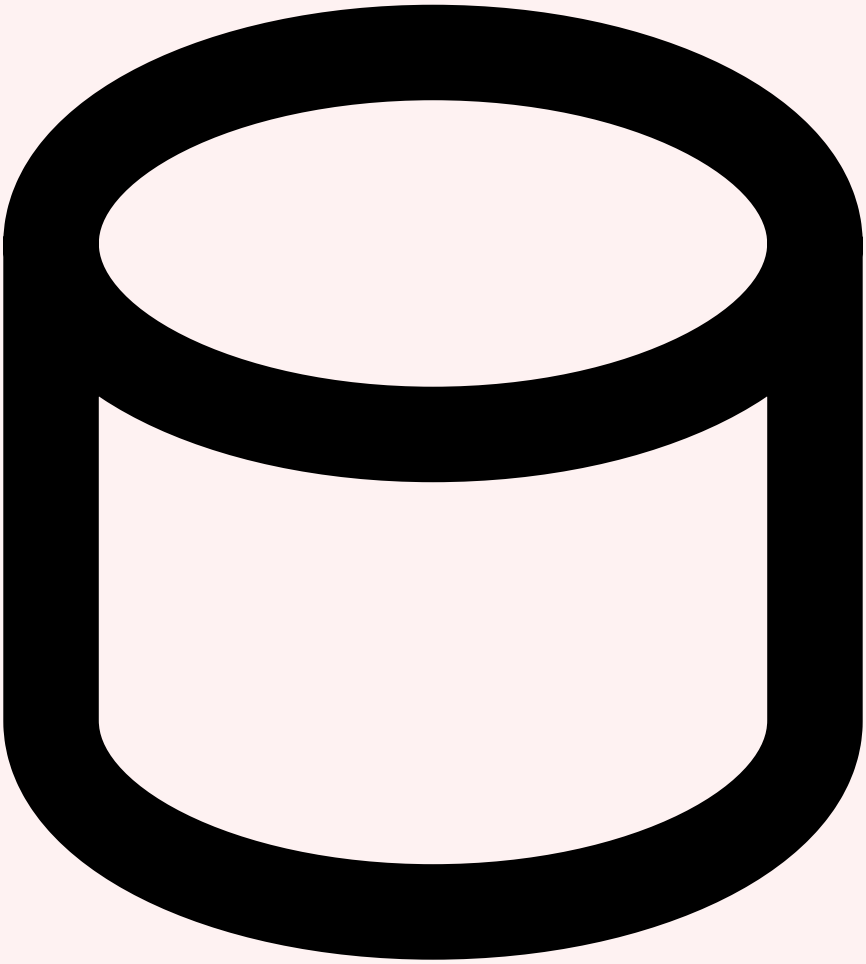
- ▹**Combinaison critique :** LLM01+LLM02 forment une chaîne d'attaque — l'injection contrôle le modèle, le output handling non sécurisé exécute l'attaque sur les systèmes en aval





3 LLM03-04 : Training Data Poisoning et Model DoS

Si LLM01 et LLM02 ciblent le runtime de l'application, les vulnérabilités **LLM03 (Training Data Poisoning)** et **LLM04 (Model Denial of Service)** s'attaquent respectivement à l'intégrité du modèle lui-même et à sa disponibilité. Le poisoning corrompt le comportement du modèle de manière permanente en amont, tandis que le DoS le rend inutilisable en production. Ces deux vulnérabilités sont particulièrement pernicieuses car leurs effets peuvent être difficiles à détecter et à attribuer.



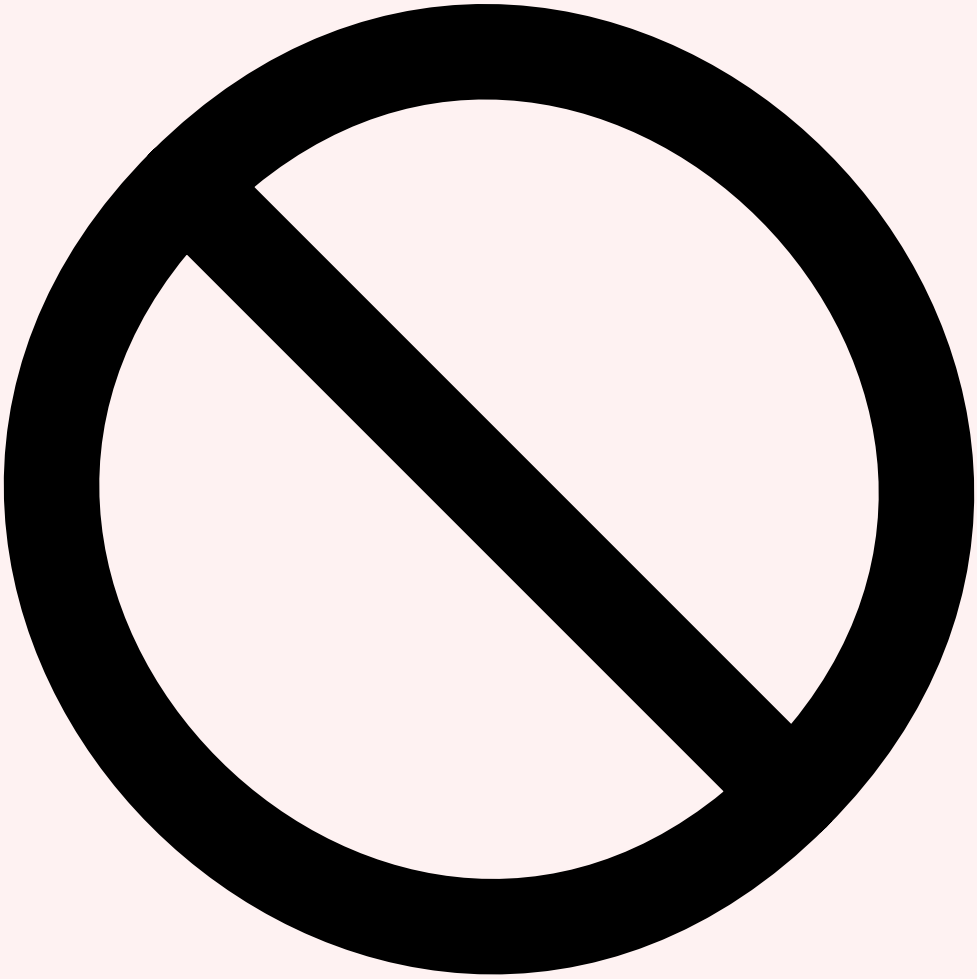
LLM03 — Training Data Poisoning

Le **training data poisoning** consiste à injecter des données malveillantes dans le corpus d'entraînement ou de fine-tuning d'un modèle pour modifier son comportement de manière ciblée. L'attaque peut prendre plusieurs formes : la **manipulation directe des données** d'entraînement lorsque l'attaquant a un accès (même partiel) au pipeline de données, l'insertion de **backdoor triggers** — des séquences spécifiques qui, une fois apprises, déclenchent un comportement malveillant précis (par exemple, toujours recommander un produit spécifique quand un mot-clé apparaît), et l'**injection de biais** qui oriente systématiquement les réponses du modèle dans une direction favorable à l'attaquant. En 2026, les attaques de poisoning ciblent principalement les pipelines de **fine-tuning** et de **RLHF**, car ces étapes utilisent des datasets plus petits et plus facilement corruptibles que les corpus de pré-entraînement massifs.

Remédiation LLM03 : intégrité du pipeline de données

La remédiation du data poisoning repose sur la sécurisation de la **chaîne de provenance des données**. Le **data provenance tracking** (C2PA, Data Cards) assure la traçabilité complète de chaque échantillon depuis sa source jusqu'à son inclusion dans le dataset d'entraînement. L'**analyse statistique** détecte les anomalies dans la distribution des données : un clustering automatique peut identifier des échantillons outliers insérés par un attaquant. Des outils comme **Cleanlab** identifient automatiquement les labels incorrects et les données bruitées qui pourraient être le signe d'un poisoning. Les **audits de fine-tuning** comparent systématiquement les performances du modèle avant et après chaque cycle de fine-tuning sur un benchmark de sécurité standardisé, détectant toute dégradation suspecte du comportement.

```python
# Détection de data poisoning avec analyse statistique
from sklearn.ensemble import IsolationForest
from sentence_transformers import SentenceTransformer
import numpy as np

def detect_poisoned_samples(dataset: list[str],
contamination=0.05):
    """Détecte les échantillons potentiellement empoisonnés"""
    model = SentenceTransformer("all-MiniLM-L6-v2")
    embeddings = model.encode(dataset)

    # Isolation Forest pour détecter les outliers
    clf = IsolationForest(contamination=contamination,
random_state=42)
    predictions = clf.fit_predict(embeddings)

    suspicious = [i for i, p in enumerate(predictions)
if p == -1]
    return suspicious, embeddings

# Vérification de la dérive post-fine-tuning
def audit_finetuning(model_before, model_after,
safety_benchmark):
    """Compare les scores de sécurité avant/après fine-
tuning"""
    score_before = evaluate(model_before, safety_benchmark)
    score_after = evaluate(model_after, safety_benchmark)
    drift = score_before - score_after
    if drift > 0.05:  # 5% de dégradation = alerte
        alert(f"Safety score drift: {drift:.2%}")
    return drift
```



LLM04 — Model Denial of Service

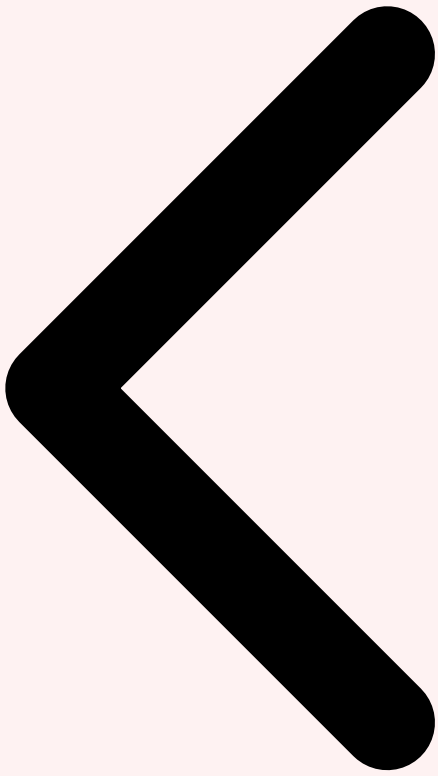
Le **Model Denial of Service** exploite le coût computationnel inhérent à l'inférence des LLM pour saturer les ressources ou générer des factures astronomiques. Les vecteurs d'attaque incluent les **prompts de longueur maximale** (remplissage du context window avec du texte inutile), les **crafted inputs** qui forcent le modèle à générer des réponses extrêmement longues (boucles de raisonnement, listes infinies), et la **génération récursive** où le modèle est piégé dans des appels de fonction en cascade via les plugins. En 2026, avec des coûts d'inférence de 15 à 60 USD par million de tokens sur les modèles frontier, une attaque DoS bien orchestrée peut générer des factures de **dizaines de milliers de dollars en quelques heures**.

Remédiation LLM04 : contrôles de ressources

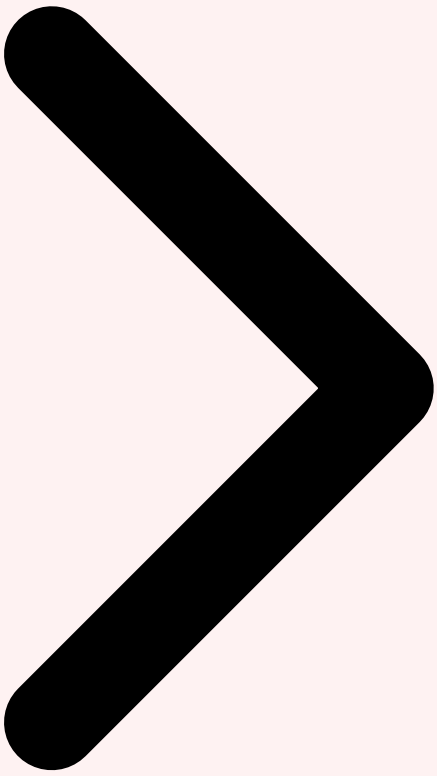
Les défenses contre le DoS LLM sont largement architecturales. Le **rate limiting** granulaire (par utilisateur, par IP, par API key) avec des paliers progressifs (soft limit → hard limit → ban temporaire) prévient les attaques par volume. Les **limites de taille d'entrée** (max tokens input) et de sortie (max tokens output) encadrent le coût de chaque requête. Les **timeouts agressifs** (30-60 secondes par requête) coupent les requêtes qui forcent un raisonnement excessif. Les **circuit breakers** désactivent automatiquement le service lorsque le taux d'erreur ou le coût dépasse un seuil configurable, protégeant contre les cascades de défaillances. Un **budget alerting** en temps réel sur les coûts d'API est la dernière ligne de défense financière.

- ▹**Métrique clé LLM03 :** surveiller le safety score sur un benchmark standardisé après chaque fine-tuning — une dégradation supérieure à 5% nécessite une investigation immédiate du dataset
- ▹**Métrique clé LLM04 :** ratio coût/requête avec alertes sur les percentiles (P95, P99) — une requête au P99 qui coûte 10x la médiane indique probablement un input crafté

- ▹**Détection croisée :** un pic de tokens d'entrée combiné à un pic de tokens de sortie sur un même utilisateur est un indicateur fort d'attaque DoS intentionnelle

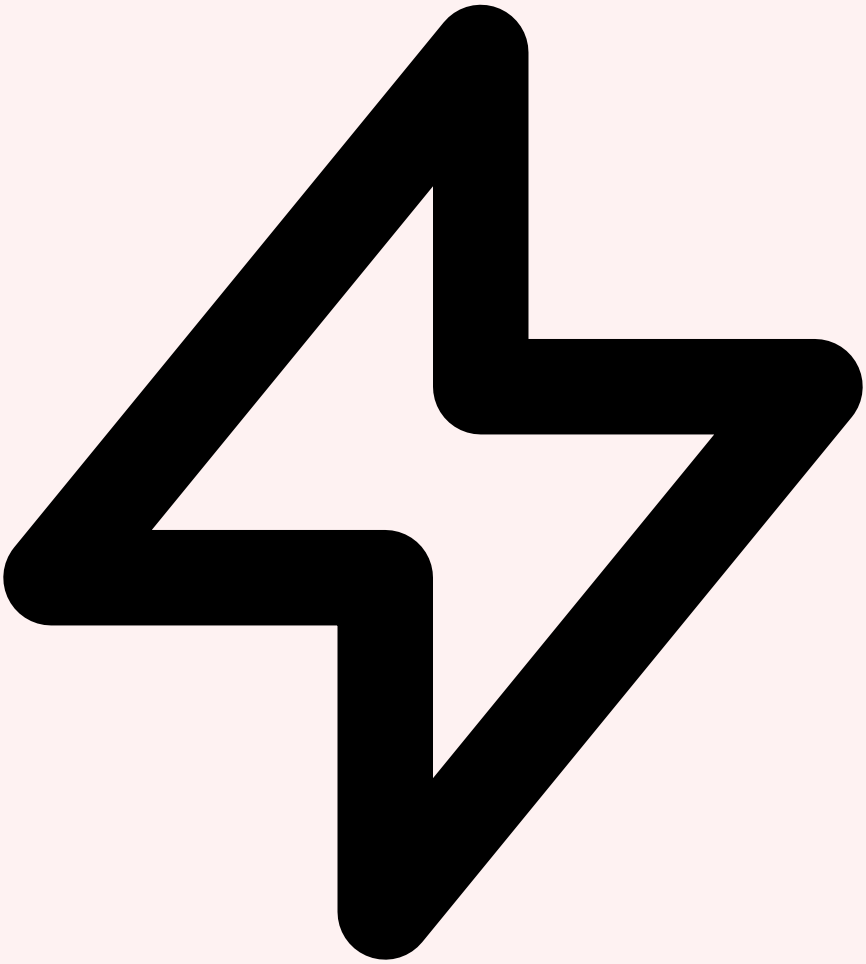


LLM01-02 Injection & Output LLM03-04 Poisoning & DoS LLM05-06 Supply & Disclosure



4 LLM05-06 : Supply Chain et Sensitive Information Disclosure

Les vulnérabilités **LLM05 (Supply Chain)** et **LLM06 (Sensitive Information Disclosure)** ont vu leurs scores de risque augmenter significativement dans la version 2.1 du référentiel OWASP. La multiplication des composants tiers dans les architectures LLM modernes — modèles pré-entraînés, adaptateurs LoRA, datasets de fine-tuning, frameworks d'orchestration — crée une surface d'attaque qui dépasse largement le modèle lui-même. Parallèlement, la capacité des LLM à mémoriser et restituer des données sensibles issues de l'entraînement ou du contexte en fait des vecteurs de fuite de données extrêmement efficaces.

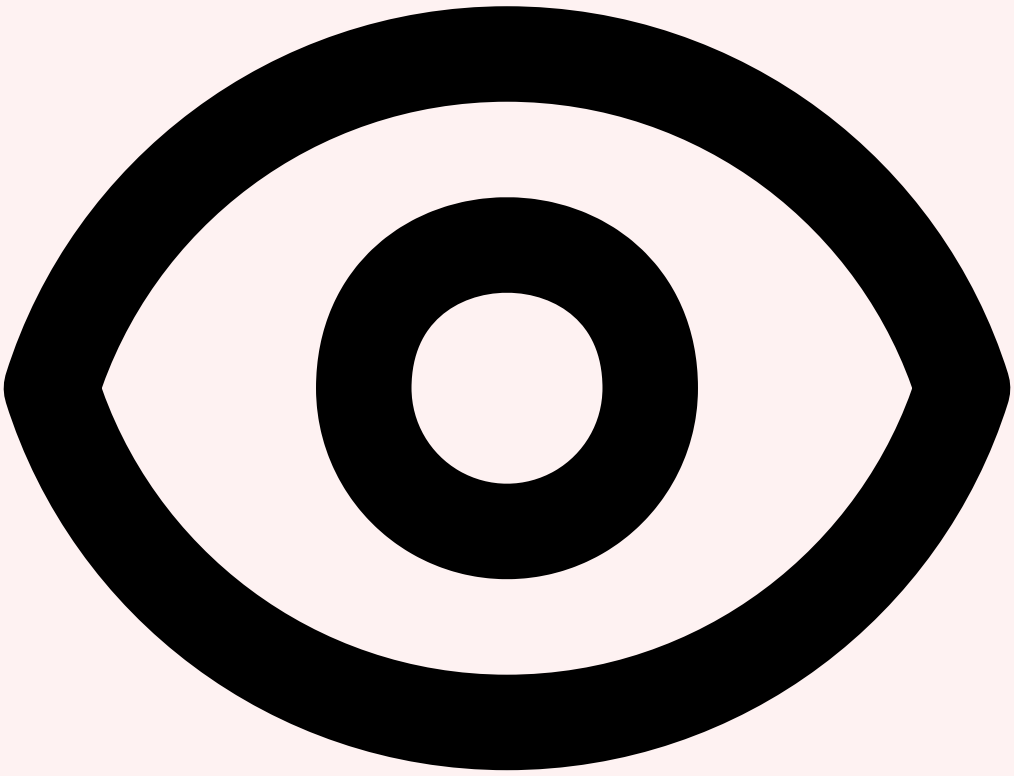


LLM05 — Supply Chain Vulnerabilities

La **supply chain des LLM** est intrinsèquement plus complexe et opaque que celle du logiciel traditionnel. Les vecteurs d'attaque incluent les **modèles tiers compromis** hébergés sur des hubs publics (Hugging Face, ModelScope) — un modèle pré-entraîné peut contenir des backdoors, du code malveillant dans les poids sérialisés (pickle deserialization attacks), ou des comportements cachés activés par des triggers spécifiques. Les **dépendances logicielles** des frameworks ML (PyTorch, TensorFlow, LangChain, LlamaIndex) introduisent des vulnérabilités classiques (CVE) amplifiées par le fait que ces frameworks exécutent du code avec des privilèges élevés. Les **datasets publics** utilisés pour le fine-tuning peuvent être empoisonnés à la source, compromettant tous les modèles dérivés. En 2026, plusieurs incidents majeurs ont impliqué des modèles Hugging Face contenant des payloads malveillants dans les fichiers de configuration ou les tokenizers personnalisés.

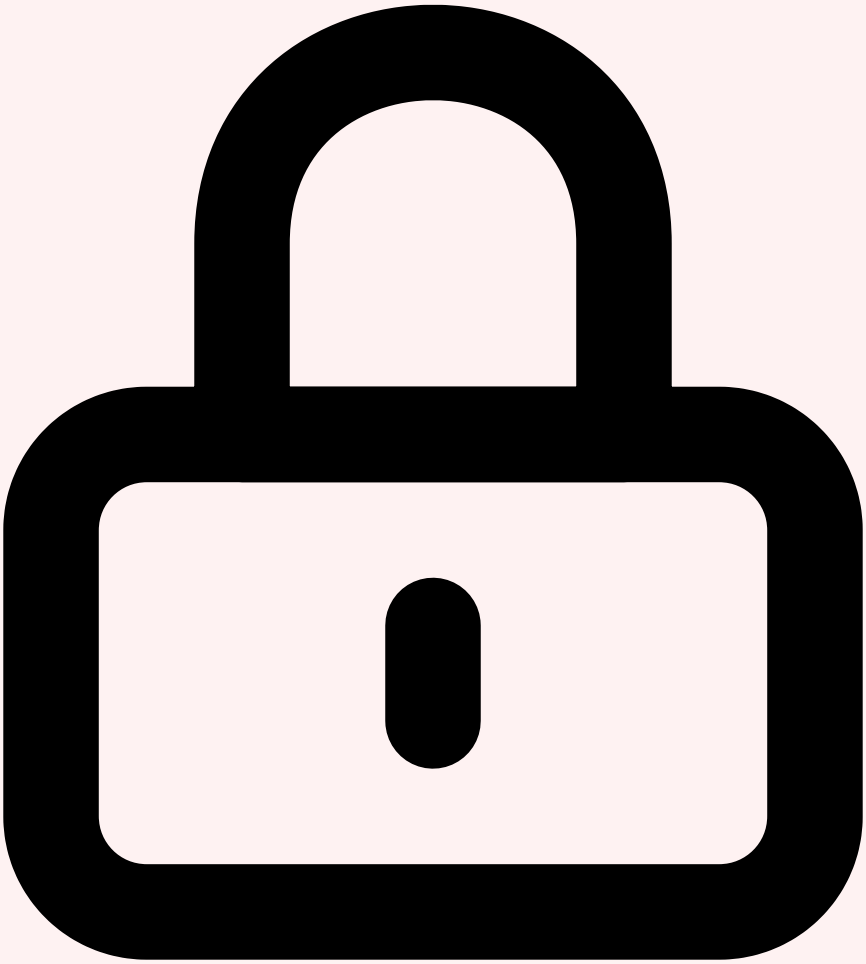
Remédiation LLM05 : gouvernance de la chaîne d'approvisionnement

La sécurisation de la supply chain LLM exige une approche systématique. Le **model scanning** analyse les fichiers de modèles avant chargement pour détecter les payloads malveillants — des outils comme **ModelScan** (Protect AI) et **Fickling** identifient les code injections dans les fichiers pickle. L'adoption du **AI SBOM** (Software Bill of Materials pour l'IA) documente tous les composants : modèle de base, datasets, adaptateurs, dépendances logicielles avec leurs versions et hash de vérification. Un **registry sécurisé interne** remplace les téléchargements directs depuis les hubs publics — les modèles sont scannés, validés et versionnés avant d'être rendus disponibles aux équipes. Le **vendor assessment** évalue les fournisseurs de modèles sur leurs pratiques de sécurité, leur gestion des vulnérabilités et leur politique de mise à jour. Pour approfondir, consultez Playbooks de Réponse aux Incidents IA : Modèles et Automatisation.



LLM06 — Sensitive Information Disclosure

La **divulgation d'informations sensibles** par les LLM se manifeste sous trois formes principales. Le **PII leakage** survient lorsque le modèle restitue des données personnelles (noms, emails, numéros de téléphone, adresses) présentes dans ses données d'entraînement ou dans le contexte RAG. L'**extraction du system prompt** permet à un attaquant de récupérer les instructions confidentielles qui définissent le comportement de l'application — ces prompts contiennent souvent de la logique métier propriétaire, des clés API, ou des informations sur l'architecture. La **training data memorization** est un phénomène bien documenté où les LLM mémorisent et restituent verbatim des passages de leurs données d'entraînement, incluant potentiellement du code source propriétaire, des données médicales ou financières, et des correspondances privées. Des chercheurs ont démontré en 2025 que GPT-4 pouvait restituer des extraits verbatim de textes protégés par copyright lorsqu'il était sollicité avec les bons préfixes.

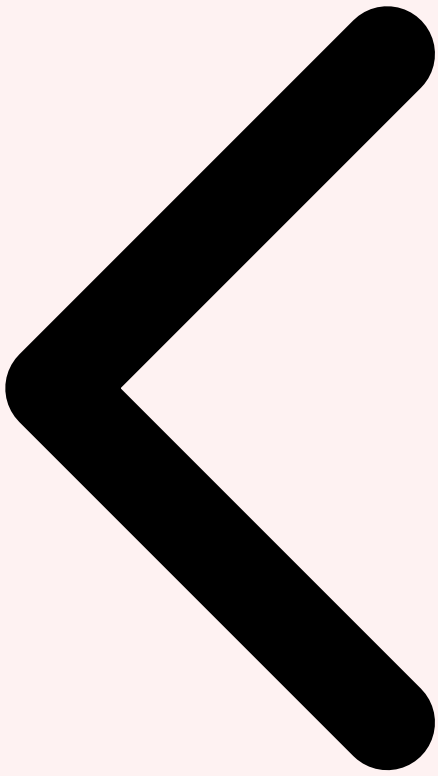


Remédiation LLM06 : protection des données sensibles

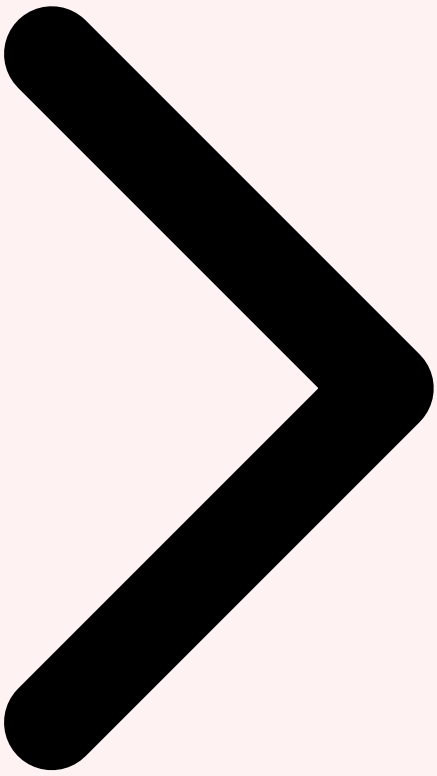
La remédiation combine des contrôles en entrée, en sortie et au niveau du modèle. Les **filtres DLP (Data Loss Prevention)** scannent les sorties du LLM en temps réel pour détecter et masquer les PII, les secrets (clés API, tokens), et les données classifiées avant qu'ils n'atteignent l'utilisateur. L'**output scanning** avec des outils comme **Presidio** (Microsoft) ou **LLM Guard** identifie automatiquement les entités sensibles dans les réponses. La **differential privacy** appliquée lors de l'entraînement réduit mathématiquement la capacité du modèle à mémoriser des échantillons individuels. Les **guardrails architecturaux** séparent le system prompt en parties publiques (logique de conversation) et privées (secrets, configuration) avec des mécanismes de protection distincts pour chaque niveau.

- ▹**Gouvernance et Architecture :** les deux piliers qui couvrent le plus de vulnérabilités — une architecture défensive solide et une gouvernance rigoureuse protègent contre 9 des 10 vulnérabilités
- ▹**Output Controls insuffisants seuls :** les contrôles de sortie ne couvrent que partiellement les menaces Supply Chain, Data Poisoning et Model Theft — ils doivent être complétés par des contrôles en amont

- ▹**Monitoring universel :** la surveillance est pertinente pour les 10 vulnérabilités — c'est la couche transversale de détection et de réponse aux incidents LLM



LLM03-04 Poisoning & DoS LLM05-06 Supply & Disclosure LLM07-08 Plugin & Agency



5 LLM07-08 : Insecure Plugin Design et Excessive Agency

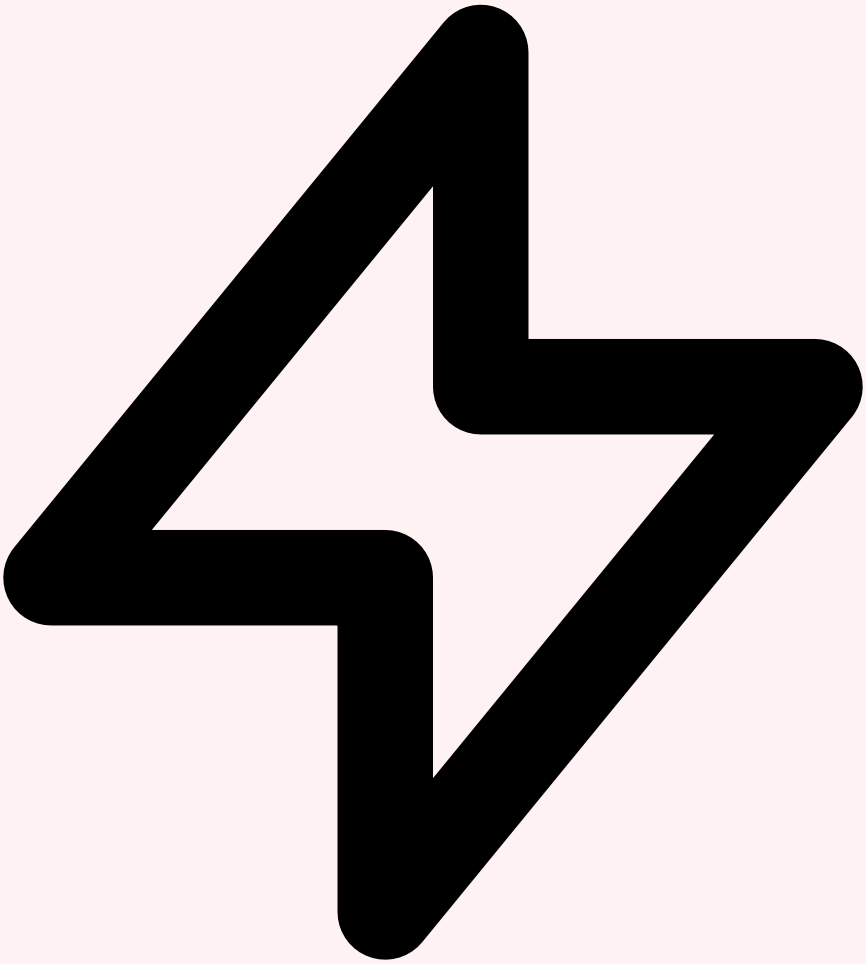
Avec l'essor des **architectures agentiques** en 2026, les vulnérabilités LLM07 (Insecure Plugin Design) et LLM08 (Excessive Agency) sont passées du statut de risques théoriques à celui de **menaces opérationnelles critiques**. Les LLM ne se contentent plus de générer du texte : ils exécutent des outils, appellent des API, manipulent des fichiers et prennent des décisions autonomes via des frameworks comme CrewAI, AutoGen et le Model Context Protocol (MCP). Chaque plugin, chaque outil connecté, chaque fonction appelable est un vecteur d'attaque potentiel qui étend la surface d'attaque bien au-delà du modèle lui-même.

LLM07 — Insecure Plugin Design

L'**insecure plugin design** survient lorsque les outils et plugins connectés au LLM ne valident pas correctement les entrées qu'ils reçoivent du modèle. Le problème fondamental est que les développeurs traitent souvent les requêtes du LLM comme des sources fiables, alors qu'un LLM compromis par prompt injection peut envoyer des **paramètres malveillants** à n'importe quel outil connecté. Les vecteurs d'attaque incluent l'**injection de commandes** via les paramètres de plugins shell, l'**escalade de privilèges** lorsqu'un plugin fonctionne avec des droits élevés sans restriction, la **traversée de répertoire** via les plugins de lecture de fichiers, et l'**exfiltration de données** via des plugins réseau (requêtes HTTP, envoi d'emails). Un cas typique en 2026 : un plugin MCP de gestion de fichiers, appelé par un LLM compromis par injection indirecte, qui exfiltre le contenu de fichiers sensibles vers un serveur externe via une requête HTTP construite par le modèle.

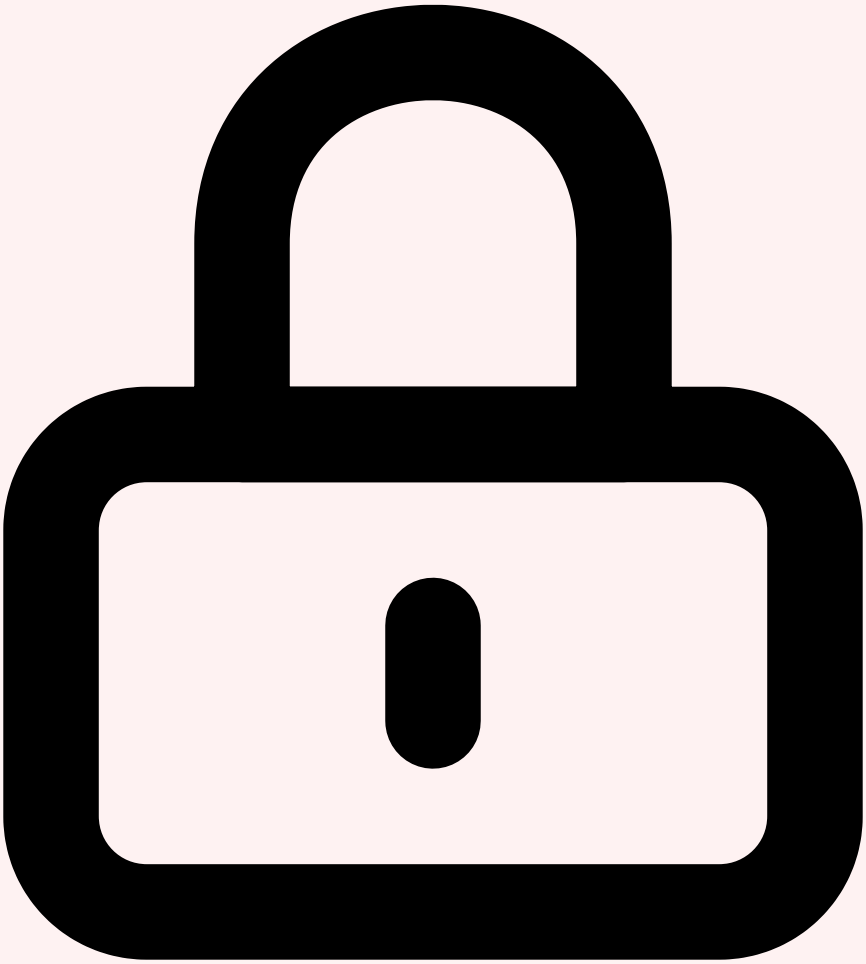
Remédiation LLM07 : sécuriser chaque plugin comme un endpoint

Chaque plugin doit être traité comme un **endpoint API exposé à un utilisateur non fiable**. Le **principe du moindre privilège** exige que chaque plugin fonctionne avec les permissions minimales nécessaires — un plugin de lecture de fichiers ne doit pas avoir d'accès en écriture, un plugin de requête SQL doit être limité à des requêtes SELECT sur des tables spécifiques. L'**input validation stricte** sur chaque paramètre de plugin (type checking, whitelist de valeurs, sanitization des chemins de fichiers, parameterized queries) empêche les injections. Le **sandboxing** exécute les plugins dans des environnements isolés (conteneurs, gVisor, Firecracker) avec des limites de ressources et un réseau restreint. Chaque appel de plugin doit être **loggé et auditable** avec les paramètres reçus, le résultat retourné et l'identité de l'utilisateur ayant initié la chaîne d'appels.



LLM08 — Excessive Agency

L'**excessive agency** survient lorsqu'un LLM dispose de capacités d'action disproportionnées par rapport à ce que son cas d'usage requiert, et surtout lorsque ces actions sont exécutées **sans confirmation humaine**. En 2026, les agents autonomes basés sur LLM peuvent exécuter du code, modifier des bases de données, envoyer des communications, et interagir avec des API externes. Un agent avec un **excessive agency** peut, en réponse à une prompt injection ou une hallucination, supprimer des données en production, envoyer des emails à des clients avec des informations erronées, effectuer des transactions financières non autorisées, ou modifier des configurations critiques. Le problème est amplifié par les architectures multi-agents où un agent compromis peut influencer le comportement de tous les agents en aval dans la chaîne de traitement.



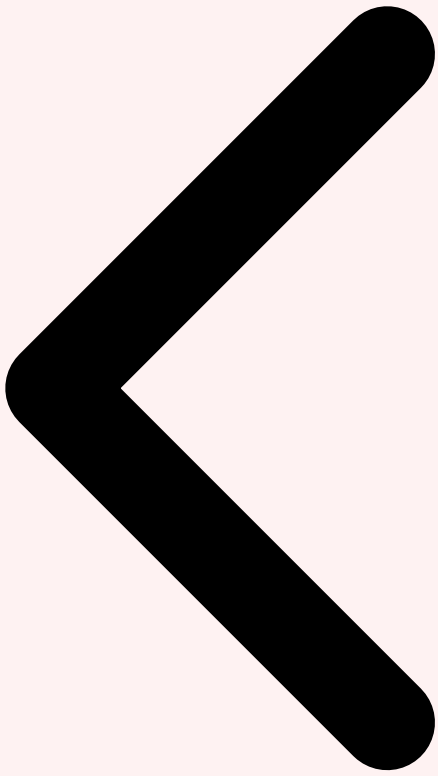
Remédiation LLM08 : contrôle de l'autonomie

La remédiation de l'excessive agency repose sur le contrôle granulaire de ce que le LLM est autorisé à faire. Le **human-in-the-loop** (HITL) obligatoire pour toute action à impact élevé (modifications de données, transactions, communications externes) interrompt l'exécution automatique et demande une validation humaine explicite. L'**action whitelisting** définit une liste explicite des actions autorisées pour chaque contexte — tout ce qui n'est pas explicitement permis est interdit par défaut. La **confirmation pour actions sensibles** implémente un mécanisme de double validation : le modèle propose l'action, un système de vérification indépendant (règles métier, second modèle, validation humaine) l'approuve ou la rejette. L'**audit trail** complet de toutes les actions exécutées permet la détection a posteriori des abus et la reconstruction forensique en cas d'incident.

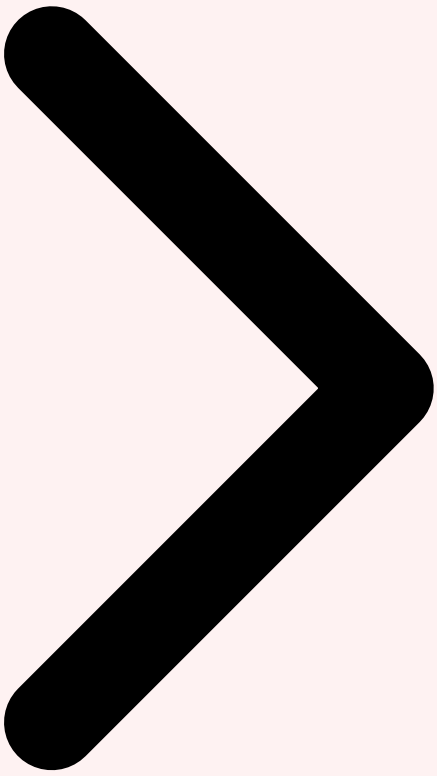
```python
# Architecture sécurisée pour plugins et agents LLM
from enum import Enum
from dataclasses import dataclass

class ActionRisk(Enum):
    LOW = "low"       # Lecture seule, aucun effet de bord
    MEDIUM = "medium"  # Modifications réversibles
    HIGH = "high"     # Actions irréversibles, données
sensibles
    CRITICAL = "critical" # Transactions, communications
externes

@dataclass
class PluginPolicy:
    name: str
    allowed_actions: list[str]
    risk_level: ActionRisk
    requires_approval: bool
    max_calls_per_minute: int
    sandbox: bool = True

class AgencyController:
    def execute_action(self, action, params, policy):
        # 1. Vérifier que l'action est dans la whitelist
        if action not in policy.allowed_actions:
            raise PermissionError(f"Action '{action}' non
autorisée")
        # 2. HITL pour les actions à haut risque
        if policy.requires_approval:
            approval = self.request_human_approval(action,
params)
            if not approval: return {"status": "rejected_by_hu
man"}
        # 3. Exécution sandboxée avec audit
        self.audit_log(action, params, policy)
        return self.sandbox_execute(action, params)
```

- ▹**Règle critique LLM07 :** chaque plugin est un endpoint API non fiable — implémenter validation d'entrée, moindre privilège et sandboxing sur chaque outil connecté au LLM
- ▹**Règle critique LLM08 :** un LLM ne devrait jamais exécuter une action irréversible sans validation humaine — implémenter HITL obligatoire pour les actions HIGH et CRITICAL

- ▹**Pattern sécurisé :** adopter l'architecture "propose-verify-execute" — le LLM propose, un contrôleur vérifie contre les politiques, et l'exécution se fait dans un sandbox audité
- ▹**Multi-agents :** dans les architectures multi-agents, chaque agent doit avoir sa propre politique de sécurité et les communications inter-agents doivent être validées comme des inputs non fiables



LLM05-06 Supply & Disclosure LLM07-08 Plugin & Agency LLM09-10 Overreliance & Theft



6 LLM09-10 : Overreliance et Model Theft

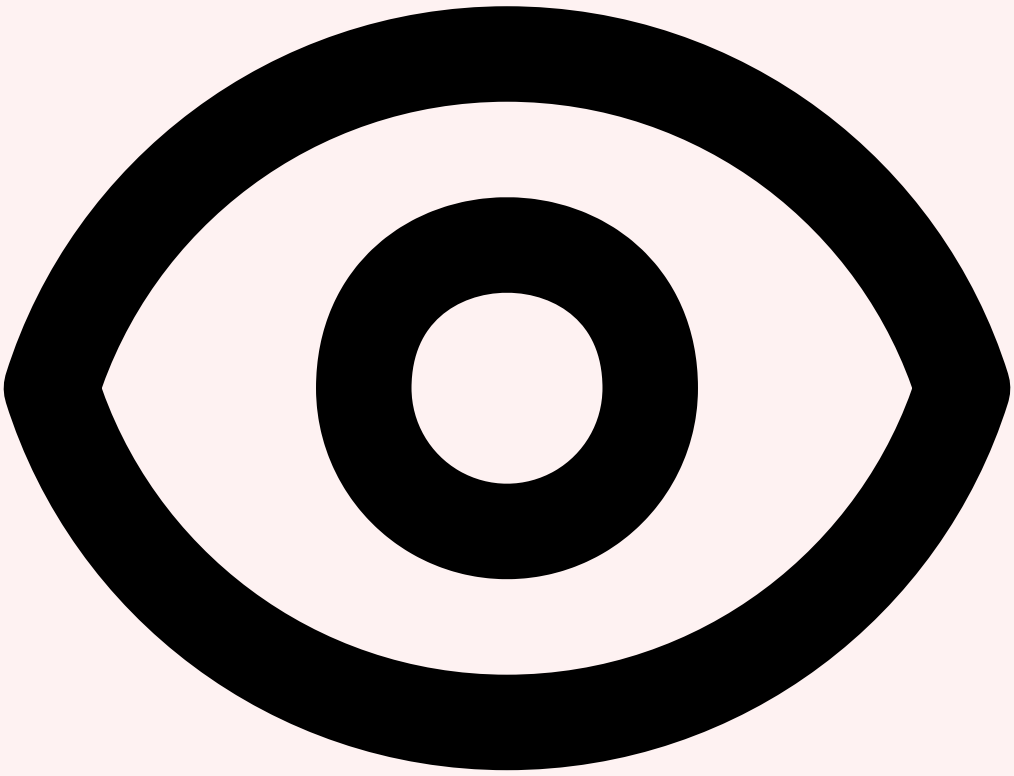
Les deux dernières vulnérabilités du classement OWASP ciblent des risques de nature très différente : l'**overreliance** (LLM09) est une vulnérabilité humaine amplifiée par la technologie, tandis que le **model theft** (LLM10) est une menace de propriété intellectuelle avec des implications financières considérables. Malgré leurs scores de risque modérés (7.0 et 7.8), ces deux vulnérabilités peuvent causer des dommages systémiques qui dépassent largement ceux d'une simple exploitation technique. Pour approfondir, consultez RAG en Production : Architecture, Scaling et Bonnes.

LLM09 — Overreliance : le danger des hallucinations non détectées

L'**overreliance** se manifeste lorsque les utilisateurs ou les systèmes automatisés font confiance aux sorties d'un LLM sans vérification adéquate. Les LLM génèrent des **hallucinations** — des affirmations factuellement incorrectes mais formulées avec une confiance apparente élevée — à un taux qui varie entre 3% et 15% selon le modèle et le domaine. Dans un contexte professionnel, cette overreliance peut avoir des conséquences graves : des **décisions juridiques** fondées sur des jurisprudences inventées (cas Mata v. Avianca, 2023), des **diagnostics médicaux** erronés basés sur des informations hallucinées, des **rapports financiers** contenant des chiffres fabriqués, ou des **décisions automatisées** prises sans intervention humaine sur la base de recommandations incorrectes du modèle. Le risque est amplifié par le phénomène d'**automation bias** : les humains ont tendance à faire davantage confiance aux systèmes automatisés qu'à leur propre jugement, surtout lorsque les sorties sont formulées de manière convaincante.

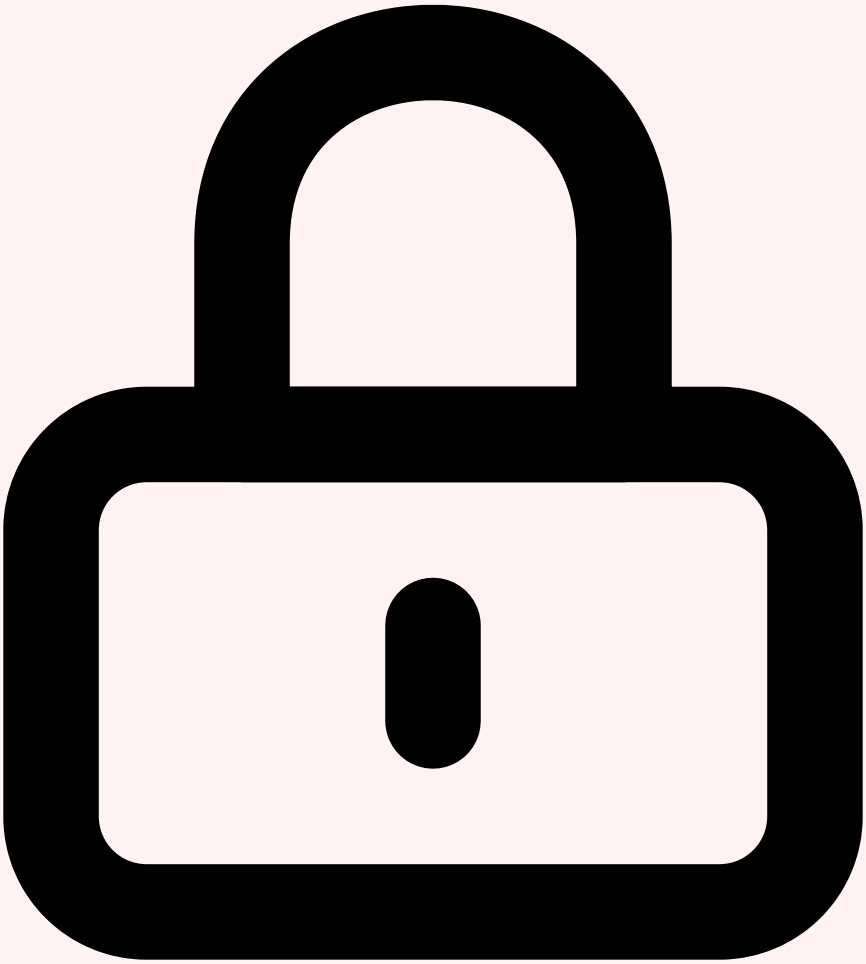
Remédiation LLM09 : ancrage et vérification

La remédiation de l'overreliance combine des mesures techniques et organisationnelles. Le **RAG (Retrieval-Augmented Generation)** ancre les réponses du modèle dans des sources de données vérifiables, réduisant significativement le taux d'hallucinations factuelles. Les mécanismes de **fact-checking automatisé** croisent les affirmations du LLM avec des bases de connaissances structurées (Knowledge Graphs, bases documentaires certifiées) pour détecter les incohérences. Le **confidence scoring** évalue la certitude du modèle sur chaque affirmation et signale visuellement les passages à faible confiance — les architectures modernes utilisent des techniques comme la calibration de température, le self-consistency checking (générer plusieurs réponses et vérifier leur cohérence), et les modèles d'évaluation dédiés. Les **disclaimers systématiques** rappellent aux utilisateurs que les sorties doivent être vérifiées, tandis que les workflows intègrent des étapes de validation humaine obligatoire avant toute action basée sur une recommandation LLM.



LLM10 — Model Theft : extraction et vol de modèles

Le **model theft** englobe les techniques permettant de voler ou reproduire un modèle propriétaire sans autorisation. L'**extraction via API** (model extraction attack) consiste à interroger massivement un modèle pour entraîner un modèle clone qui reproduit son comportement — des recherches ont démontré qu'avec suffisamment de requêtes (quelques millions), il est possible de distiller un modèle qui atteint 90-95% des performances de l'original. Les **side-channel attacks** exploitent les métadonnées des réponses (timing, logprobs, nombre de tokens) pour extraire des informations sur l'architecture interne du modèle. Les **insider threats** restent le vecteur le plus direct : un employé avec accès aux poids du modèle peut les copier et les exfiltrer. En 2026, avec des coûts d'entraînement dépassant les 100 millions de dollars pour les modèles frontier, le vol de modèle représente une menace de propriété intellectuelle considérable.



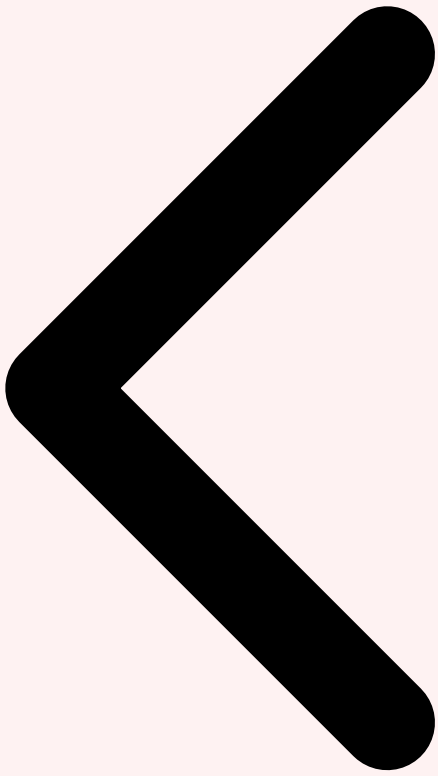
Remédiation LLM10 : protection de la propriété intellectuelle

La protection contre le vol de modèle nécessite des contrôles à plusieurs niveaux. Le **rate limiting agressif** avec des budgets de requêtes par utilisateur et par période limite la quantité de données exploitables pour une attaque d'extraction — un plafond de quelques milliers de requêtes par jour rend l'extraction impraticable. Le **watermarking** insère des signatures invisibles dans les sorties du modèle qui permettent de tracer l'origine des données en cas de redistribution non autorisée — des techniques comme le watermarking de distributions de tokens sont résistantes à la post-édition. Les **access controls** stricts (authentification forte, réseau privé, chiffrement des poids) protègent les artefacts du modèle contre l'exfiltration. Le **monitoring comportemental** détecte les patterns d'utilisation anormaux indicatifs d'une tentative d'extraction : requêtes systématiques couvrant uniformément l'espace des entrées, fréquence anormalement élevée, ou patterns de requêtes caractéristiques d'un dataset de distillation.

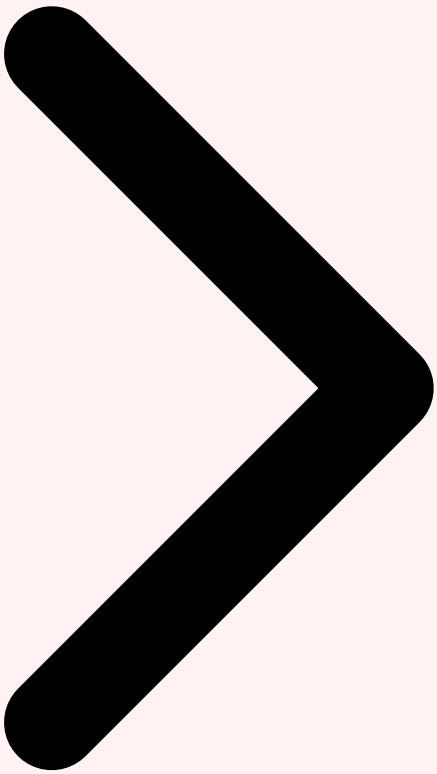
Analyse coût de vol vs coût de protection : entraîner un modèle frontier coûte entre **50M et 500M USD**. L'extraction via API coûte environ **50K-500K USD** en requêtes. Les mesures de protection (rate limiting, watermarking, monitoring) coûtent moins de **100K USD/an**. Le

ratio coût de protection / valeur du modèle est inférieur à 0.1%, rendant l'investissement en sécurité hautement rentable. Sans protection, un modèle à 100M USD peut être extrait pour 0.5% de son coût de création.

- ▹**Overreliance systémique :** implémenter le RAG avec citation des sources, le confidence scoring visible, et des workflows de validation humaine obligatoire pour les décisions critiques
- ▹**Model Theft prévention :** combiner rate limiting strict, watermarking des sorties, contrôle d'accès aux poids et monitoring comportemental des patterns d'extraction
- ▹**Hallucinations critiques :** dans les domaines à haut risque (médical, juridique, financier), imposer une vérification humaine de 100% des sorties LLM avant toute décision ou action
- ▹**Détection d'extraction :** alerter sur les utilisateurs dépassant le P99 en volume de requêtes avec une distribution uniforme des sujets — signature typique d'un dataset de distillation



LLM07-08 Plugin & Agency LLM09-10 Overreliance & Theft Implémentation Globale



7 Implémentation Globale : Checklist de Sécurisation LLM

Après avoir analysé individuellement chacune des dix vulnérabilités, cette section synthétise l'ensemble en une **stratégie de sécurisation holistique**. La défense en profondeur pour les LLM ne consiste pas à empiler des contrôles isolés, mais à construire une architecture cohérente où chaque couche de défense compense les faiblesses des autres. Cette section fournit la checklist complète de sécurisation pré-déploiement, les outils recommandés et le cadre de conformité applicable en 2026.

Architecture de défense en profondeur

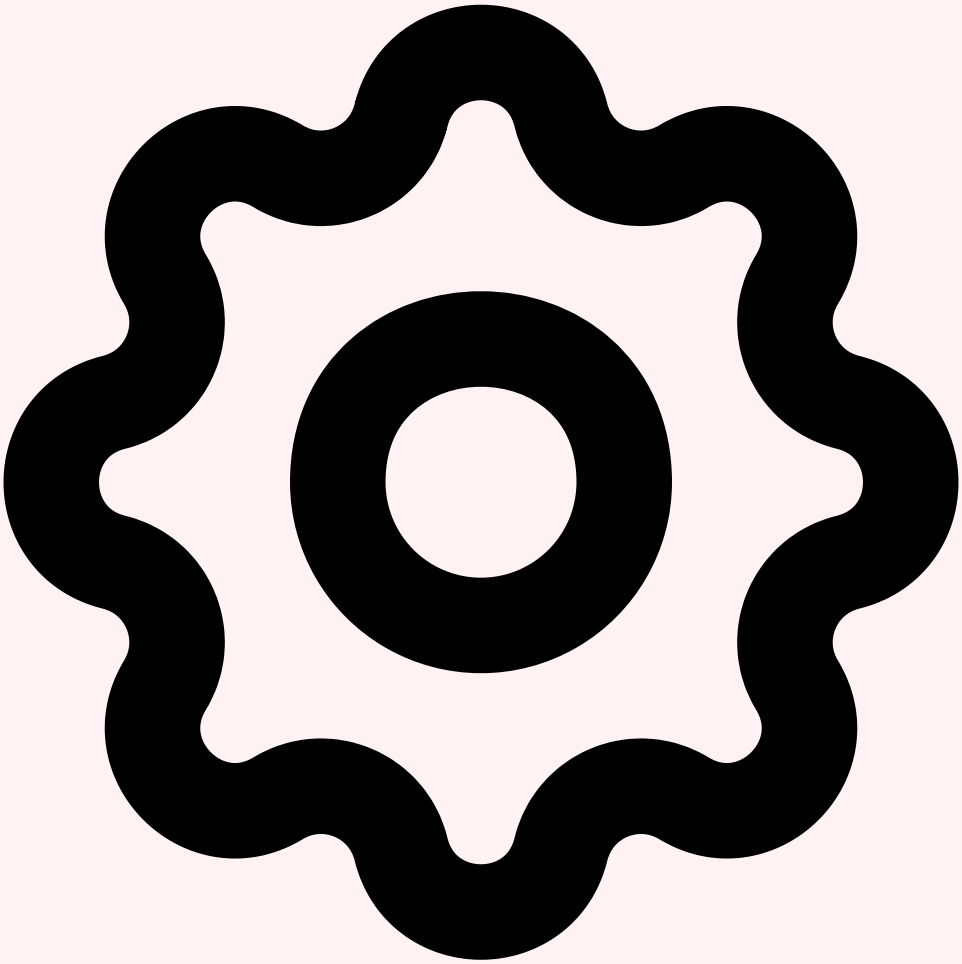
L'architecture de défense en profondeur pour les LLM s'organise en **cinq couches concentriques**. La couche **Périmètre** (rate limiting, WAF IA-aware, authentification) protège l'accès au service. La couche **Input** (sanitization, injection detection, input validation) filtre les entrées malveillantes. La couche **Modèle** (guardrails, instruction hierarchy, canary tokens) contrôle le comportement du LLM lui-même. La couche **Output** (DLP, output encoding, type validation, PII scanning) assainit les sorties. La couche **Exécution** (sandboxing, HITL, action whitelisting) contrôle les actions déclenchées par le modèle. Chaque couche doit fonctionner indépendamment et être testée individuellement. L'échec d'une couche ne doit pas compromettre la sécurité globale — c'est le principe fondamental de la défense en profondeur appliqué à l'IA.

Checklist de sécurisation pré-déploiement

Cette checklist de 20 items couvre l'ensemble des vulnérabilités OWASP LLM Top 10 et doit être validée intégralement avant tout déploiement en production. Chaque item est classé par criticité et associé aux vulnérabilités qu'il adresse.

- ▹**[CRITIQUE] Input sanitization :** validation et filtrage de toutes les entrées utilisateur contre les patterns d'injection connus (LLM01)
- ▹**[CRITIQUE] Output encoding :** assainissement de toutes les sorties LLM avant insertion dans un contexte d'exécution — HTML, SQL, shell (LLM02)
- ▹**[CRITIQUE] Instruction hierarchy :** séparation stricte entre instructions système et données utilisateur avec délimiteurs forts (LLM01)
- ▹**[CRITIQUE] Plugin input validation :** chaque plugin valide ses paramètres indépendamment du LLM, avec type checking et whitelist (LLM07)
- ▹**[CRITIQUE] Human-in-the-loop :** approbation humaine obligatoire pour toute action irréversible ou à impact élevé (LLM08)
- ▹**[ÉLEVÉ] Rate limiting :** limites par utilisateur, par IP et par API key avec paliers progressifs (LLM04, LLM10)

- ▹**[ÉLEVÉ] DLP output scanning :** détection et masquage automatique des PII, secrets et données classifiées dans les sorties (LLM06)
- ▹**[ÉLEVÉ] Model provenance :** AI SBOM documentant le modèle de base, les datasets, les adaptateurs et toutes les dépendances (LLM05)
- ▹**[ÉLEVÉ] Canary tokens :** marqueurs secrets dans le system prompt pour détecter les tentatives d'extraction (LLM01, LLM06)
- ▹**[ÉLEVÉ] Action whitelisting :** liste explicite des actions autorisées par contexte, rejet par défaut de tout le reste (LLM08)
- ▹**[ÉLEVÉ] Model scanning :** analyse des fichiers de modèles tiers pour détecter les payloads malveillants avant chargement (LLM05)
- ▹**[ÉLEVÉ] Sandboxing plugins :** exécution isolée de chaque plugin avec limites de ressources et réseau restreint (LLM07)
- ▹**[MOYEN] Data provenance :** traçabilité complète de chaque échantillon dans les datasets de fine-tuning (LLM03)
- ▹**[MOYEN] Safety benchmarking :** évaluation automatisée du modèle sur un benchmark de sécurité après chaque mise à jour (LLM03)
- ▹**[MOYEN] RAG grounding :** ancrage des réponses dans des sources vérifiables avec citation obligatoire (LLM09)
- ▹**[MOYEN] Confidence scoring :** indication visuelle du niveau de certitude du modèle sur chaque assertion (LLM09)
- ▹**[MOYEN] Watermarking :** insertion de signatures dans les sorties pour tracer la redistribution non autorisée (LLM10)
- ▹**[MOYEN] Timeout et circuit breakers :** coupure automatique des requêtes excessives et désactivation en cas de surcharge (LLM04)
- ▹**[MOYEN] Audit logging :** journalisation complète de toutes les interactions, paramètres de plugins et actions exécutées (LLM07, LLM08)
- ▹**[MOYEN] CSP stricte :** Content Security Policy bloquant l'exécution de scripts inline générés par le LLM (LLM02)



Outils de sécurité LLM en 2026

L'écosystème d'outils de sécurité LLM s'est considérablement enrichi depuis 2024. **Garak** (NVIDIA) est le framework de red teaming LLM le plus complet, offrant des centaines de probes pré-configurées couvrant les 10 vulnérabilités OWASP avec des générateurs d'attaques automatisés et des rapports de couverture détaillés. **NeMo Guardrails** (NVIDIA) fournit un DSL (Colang) pour définir des règles de comportement programmatiques qui contrôlent les entrées, les sorties et les interactions du LLM avec les outils externes — c'est la solution de guardrails la plus mature en production. **LLM Guard** (Protect AI) offre une suite de scanners d'entrée et de sortie couvrant la détection d'injection, le scanning PII, la validation de toxicité et le contrôle de format. **Rebuff** est spécialisé dans la détection de prompt injection avec un moteur multi-couche combinant analyse heuristique, détection par LLM et vérification de canary tokens. Ces outils s'intègrent comme middleware dans les pipelines LLM existants (LangChain, LlamaIndex) et fonctionnent en temps réel avec une latence acceptable en production. Pour approfondir, consultez 10 Erreurs Courantes dans.

Conformité et cadres réglementaires

En 2026, trois cadres réglementaires et normatifs structurent la gouvernance de la sécurité des LLM. L'**AI Act européen** (Règlement 2024/1689) impose pour les systèmes IA à haut risque des exigences de gestion des risques (Article 9), de gouvernance des données (Article 10), de documentation technique (Article 11), de transparence (Article 13) et de supervision humaine (Article 14). Le non-respect peut entraîner des amendes allant jusqu'à 35 millions d'euros ou 7% du chiffre d'affaires mondial. Le **NIST AI Risk Management Framework** (AI RMF 1.0) fournit un cadre volontaire structuré en quatre fonctions — GOVERN, MAP, MEASURE, MANAGE — qui guide l'identification et la mitigation des risques IA tout au long du cycle de vie. La norme **ISO/IEC 42001:2023** (AI Management System) définit les exigences d'un système de management de l'IA, incluant la gestion des risques, la gouvernance des données et la supervision continue. Un programme de **bug bounty pour applications IA** complète ces cadres en offrant une évaluation continue par la communauté de sécurité, avec des catégories spécifiques pour les vulnérabilités LLM (prompt injection, data extraction, jailbreak).

Recommandation finale : la sécurisation des LLM n'est pas un événement ponctuel mais un **processus continu**. Les techniques d'attaque évoluent en quelques jours, les modèles sont mis à jour régulièrement, et les architectures se complexifient avec l'adoption des agents autonomes. Implémenter la checklist de 20 items ci-dessus comme point de départ, automatiser les tests de sécurité avec Garak dans le pipeline CI/CD, et maintenir une veille active sur les nouvelles vulnérabilités LLM est le minimum requis pour un déploiement responsable en production en 2026.

Ressources open source associées

HF Space owasp-top10-explorer (démo) HF Dataset owasp-top10-fr

Besoin d'un accompagnement expert ?

Nos consultants en cybersécurité et IA vous accompagnent dans vos projets. Devis personnalisé sous 24h.

Références et ressources externes

- OWASP LLM Top 10 — Les 10 risques majeurs pour les applications LLM
- MITRE ATLAS — Framework de menaces pour les systèmes d'intelligence artificielle
- NIST AI RMF — AI Risk Management Framework du NIST
- arXiv — Archive ouverte de publications scientifiques en IA
- HuggingFace Docs — Documentation de référence pour les modèles de ML

Pour approfondir ce sujet, consultez notre outil open-source llm-security-scanner qui facilite l'audit de sécurité des modèles de langage.

Sources et références : ArXiv IA · Hugging Face Papers

FAQ

Qu'est-ce que OWASP Top 10 pour les LLM ?

Le concept de OWASP Top 10 pour les LLM est détaillé dans les premières sections de cet article, qui couvrent les fondamentaux, les enjeux et le contexte opérationnel. Pour un accompagnement sur ce sujet, contactez nos experts.

Pourquoi OWASP Top 10 pour les LLM est-il important en cybersécurité ?

La compréhension de OWASP Top 10 pour les LLM permet aux équipes de sécurité d'améliorer leur posture défensive. Les sections « Table des Matières » et « 2 LLM01-02 : Prompt Injection et Insecure Output Handling » détaillent les raisons de cette importance. Pour un accompagnement sur ce sujet, contactez nos experts.

Comment mettre en œuvre les recommandations de cet article ?

Les recommandations pratiques sont détaillées tout au long de l'article, avec des commandes, des outils et des méthodologies éprouvées. La section « Conclusion » fournit une synthèse actionnable. Pour un accompagnement sur ce sujet, contactez nos experts.

Conclusion

Cet article a couvert les aspects essentiels de Table des Matières, 1 Introduction à l'OWASP Top 10 pour les LLM, 2 LLM01-02 : Prompt Injection et Insecure Output Handling. La mise en pratique de ces recommandations permet de renforcer significativement la posture de securite de votre organisation.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr