VS Code, npm, PyPI : votre environnement de dev est devenu un vecteur APT

> VS Code, npm, PyPI : pourquoi la toolchain développeur est devenue le vecteur APT n°1 en 2026. Analyse de la campagne TeamPCP et guide concret pour…

Vous avez sécurisé vos serveurs, renforcé vos bastions, déployé un EDR sur tous les endpoints. Et si le problème se trouvait dans votre IDE ? En 2026, la toolchain du développeur — extensions VS Code, packages npm et PyPI, templates de projet, plugins CI/CD — est devenue le vecteur d'entrée privilégié des acteurs APT et des groupes criminels sophistiqués. Voici pourquoi vos défenses actuelles ne voient rien, et ce que vous devez mettre en place maintenant.

Points clés à retenir

- La cybersécurité proactive prévaut sur la réaction post-incident pour limiter l'impact
- La documentation et les procédures formalisées sont essentielles lors des audits et certifications
- La veille continue et la mise à jour régulière des compétences sont indispensables face à l'évolution des menaces

La toolchain du développeur : une surface d'attaque stratégique et massivement sous-estimée

Les développeurs jouissent d'un niveau de confiance exceptionnel dans les entreprises. Leurs postes de travail ont accès aux dépôts de code source, aux secrets d'infrastructure, aux systèmes de déploiement continu, aux clés API de production. Un poste développeur compromis donne souvent accès à bien plus qu'un serveur d'application : il donne accès à la capacité de modifier le code qui tourne sur ces serveurs, d'injecter des backdoors dans des librairies partagées, de voler des credentials qui ouvrent l'ensemble de l'infrastructure cloud.

> ⚡ **Retour terrain**
>
> Dans mes missions d'audit, je rencontre régulièrement la même configuration à risque : des règles de firewall héritées depuis 5 à 10 ans, que personne n'ose supprimer par crainte de casser quelque chose. J'ai développé une méthode de nettoyage progressive — analyser

les logs de connexion sur 90 jours, identifier les règles sans trafic, les désactiver sans supprimer pendant 30 jours, puis valider avec les équipes métier. Sur un parc de 340 règles dans un groupe logistique, nous en avons supprimé 218 sans incident.

Les chiffres illustrent l'ampleur du problème. Le rapport State of Software Supply Chain publié par Sonatype en 2025 indique que les attaques ciblant les gestionnaires de packages open source ont augmenté de 156 % entre 2024 et 2025, pour atteindre plus de 512 000 packages malveillants référencés sur npm et PyPI au cours de l'année. Le VS Code Marketplace héberge aujourd'hui plus de 60 000 extensions ; une étude de l'université de Trente publiée en janvier 2026 a montré que moins de 15 % d'entre elles ont fait l'objet d'un audit de sécurité formel.

La puissance de ce vecteur réside dans la confiance implicite accordée aux outils. Quand un développeur installe une extension VS Code ou exécute `npm install`, il accepte par défaut que ce code s'exécute avec ses privilèges sur son poste. Pas de sandbox, pas de prompt de permission, pas d'alerte EDR. L'outillage de développement est conçu pour être puissant, et cette puissance devient une vulnérabilité systémique dès lors qu'un acteur malveillant y introduit du code hostile.

Le poste développeur : un concentrateur de secrets

Un audit rapide de n'importe quel poste développeur en entreprise révèle systématiquement la même réalité. Des tokens Git stockés en clair dans `~/.gitconfig` ou le credential manager de l'OS. Des clés AWS, Azure ou GCP dans des fichiers `.env` à la racine des projets. Des variables d'environnement injectées dans les shells. Des certificats de déploiement dans `~/.ssh/`. Des tokens d'API dans les configurations d'extensions IDE. Des credentials de bases de données dans des fichiers de configuration de frameworks.

Un package npm malveillant ou une extension VS Code compromise n'a pas besoin d'exploiter une CVE pour voler ces données. Il lui suffit de lire les fichiers auxquels le développeur a accès — ce qui est, du point de vue du système d'exploitation, un comportement tout à fait légitime. Le code malveillant tourne avec les mêmes droits que l'utilisateur, lit les mêmes fichiers, fait les mêmes requêtes réseau. Aucune anomalie système à détecter.

Cette réalité contraste violemment avec l'attention que les équipes sécurité accordent aux postes développeurs. Dans la majorité des entreprises que j'audite, les serveurs de production sont scrutés, les accès administrateurs sont tracés, les mouvements latéraux sont détectés. Les postes des développeurs, eux, sont souvent dans un angle mort : trop complexes à contraindre sans bloquer la productivité, trop peu visibles dans les tableaux de bord SOC, trop rarement inclus dans les scopes de pentest.

Anatomie d'une attaque par extension VS Code : comment ça fonctionne réellement

La compromission de GitHub par TeamPCP en mai 2026 a mis sous les projecteurs un vecteur que la communauté sécurité connaissait théoriquement depuis des années mais qui semblait trop sophistiqué pour être industrialisé. TeamPCP a démontré que non.

Une extension VS Code malveillante peut être construite selon plusieurs approches de complexité croissante.

Le typosquatting est l'approche la plus simple. L'attaquant publie une extension dont le nom ressemble à une extension populaire — "GitLens" versus "Git-Lens", "Prettier" versus "Prettier-Code". Les développeurs pressés ne vérifient pas systématiquement l'éditeur ou le nombre d'installations. Une extension avec 500 installations et un nom plausible peut rester en ligne pendant des semaines avant d'être signalée.

La compromission de compte éditeur est plus impactante. L'attaquant compromet le compte d'un développeur ayant publié une extension populaire — via phishing, credential stuffing ou vol de token OAuth — et publie une mise à jour malveillante. Les utilisateurs existants reçoivent cette mise à jour automatiquement, sans notification explicite. Une extension avec 500 000 installations peut toucher des centaines de milliers de développeurs en quelques heures.

L'extension de niche ciblée est l'approche utilisée par TeamPCP contre GitHub. Il ne s'agit pas de toucher le plus grand nombre, mais de cibler précisément les développeurs d'une organisation spécifique avec une extension qui leur paraît utile dans leur contexte. Cette approche est quasi indétectable pour les systèmes de surveillance génériques.

Une fois installée, une extension VS Code dispose d'un accès étendu. L'API VS Code expose des méthodes pour lire et écrire des fichiers, exécuter des commandes shell, accéder aux variables d'environnement, intercepter les événements de l'éditeur et faire des requêtes réseau — le tout sans demander de permission explicite à l'utilisateur au-delà de l'installation initiale.

L'exfiltration se fait typiquement via des requêtes HTTPS vers des domaines d'apparence légitime — GitHub API, un bucket S3, Google Drive — pour se fondre dans le trafic normal de développement et passer sous les radars des proxies d'entreprise et des SIEM.

Ce que peut faire une extension malveillante en 60 secondes

Pour rendre concret le risque, voici un inventaire de ce qu'une extension VS Code malveillante peut accomplir dès son installation, sans interaction de l'utilisateur :

Lire et exfiltrer `~/.ssh/id_rsa`, `~/.aws/credentials`, `~/.gitconfig` et les tokens stockés par le credential manager VS Code

Parcourir récursivement les projets ouverts pour extraire tous les fichiers `.env`, `*.pem`, `*.key` et configurations sensibles

Lire les variables d'environnement du shell intégré — qui incluent souvent des tokens injectés par les profils shell

Installer un keylogger sur le terminal intégré pour capturer les mots de passe et commandes sensibles saisies par le développeur

Modifier silencieusement du code source en cours d'édition pour y injecter des backdoors

Le vecteur npm et PyPI : quand le gestionnaire de packages devient cheval de Troie

Si les extensions IDE ciblent les développeurs individuellement, les packages open source malveillants visent les pipelines de build — et donc potentiellement l'ensemble de l'infrastructure de production.

npm héberge aujourd'hui plus de 3,2 millions de packages. PyPI en recense plus de 580 000. Ces deux registres sont des points d'entrée critiques pour trois raisons structurelles.

La confiance par défaut. Les commandes `npm install` et `pip install` exécutent du code arbitraire — scripts postinstall pour npm, `setup.py` pour Python — avec les droits du processus d'installation. Dans un pipeline CI/CD, ce processus a typiquement accès aux secrets du pipeline : tokens de déploiement cloud, credentials de registres Docker privés, clés API de services tiers.

La complexité transitive. Un projet JavaScript moderne importe en moyenne 683 packages transitifs selon une étude de Socket Security publiée en 2025. Aucune équipe humaine ne peut auditer 683 packages à chaque mise à jour. Un package malveillant enfoncé à 5 niveaux de profondeur dans le graphe de dépendances est pratiquement indétectable par révision humaine.

L'automatisation CI/CD. Les pipelines installent les packages automatiquement, sans intervention humaine, souvent en environnements éphémères sans surveillance EDR. Un package qui ne fait qu'exfiltrer des variables d'environnement n'aura aucun comportement détectable dans les logs système standard.

La dependency confusion : le vecteur le plus sous-estimé

Une variante particulièrement redoutable, conceptualisée par Alex Birsan en 2021 et toujours massivement exploitable en 2026 : l'attaque par confusion de dépendances. L'attaquant identifie des noms de packages internes privés d'une organisation via des fichiers de configuration exposés, des logs de build publics ou du code partiellement open sourcé, puis publie sur npm ou PyPI un package public portant le même nom avec un numéro de version supérieur. Les gestionnaires de packages mal configurés privilégient le registre public en cas de conflit de noms — et installent automatiquement le package malveillant lors du prochain build.

Birsan avait démontré en 2021 que cette technique fonctionnait contre des dizaines d'entreprises du Fortune 500, y compris Apple, Microsoft et Netflix. En 2026, lors de mes audits, je rencontre encore régulièrement des configurations Nexus ou Artifactory vulnérables à cette attaque. La correction est simple — configurer le registre interne pour ne jamais résoudre les packages internes depuis le registre public — mais elle nécessite de savoir que le risque existe.

Chronologie TeamPCP 2026 : anatomie d'une campagne en cascade

La campagne TeamPCP de 2026 constitue à ce jour l'exemple le plus documenté et le plus abouti d'une attaque supply chain multivecteur orchestrée avec une cohérence stratégique délibérée. L'étudier permet de comprendre comment ces acteurs pensent et progressent.

Mars 2026 — Amorçage par compromission de mainteneurs. TeamPCP compromet les comptes de mainteneurs de plusieurs bibliothèques populaires : TanStack (framework de gestion d'état), Trivy (outil d'audit de vulnérabilités dans les images Docker — ironie du sort), et LiteLLM (wrapper Python pour LLM). Les versions compromises contiennent un payload dormant activable à distance. Ces bibliothèques, installées sur des centaines de milliers de postes et de pipelines, constituent une tête de pont passive.

Avril 2026 — Déploiement industriel sur npm et PyPI. Fort des informations collectées, TeamPCP déploie 170 packages ciblant spécifiquement les environnements CI/CD. Ces packages exfiltrent les variables d'environnement de tout pipeline qui les installe. Parmi les victimes documentées : plusieurs agences gouvernementales européennes dont la Commission européenne, et des entreprises industrielles dont les noms restent confidentiels. Le groupe récupère des milliers de credentials : tokens AWS/GCP/Azure, clés de déploiement, credentials de registres Docker privés.

Avril 2026 — Frappe sur Nx Console. TeamPCP compromet le compte de l'éditeur de Nx Console, extension VS Code avec 2,2 millions d'installations. La mise à jour malveillante est disponible sur le Marketplace pendant plusieurs heures avant d'être détectée et retirée par Microsoft. Plusieurs dizaines de milliers de développeurs ont potentiellement reçu la version compromise.

Mai 2026 — Compromission de GitHub. Armé d'une extension VS Code ciblée et des informations accumulées lors des étapes précédentes, TeamPCP compromet le poste d'un employé de GitHub. Les credentials exfiltrés permettent d'accéder aux dépôts internes. Résultat : 3 800 dépôts exfiltrés incluant le code source de Copilot, CodeQL et Dependabot. GitHub confirme la brèche le 20 mai 2026.

Ce qui frappe dans cette chronologie, c'est l'escalade délibérée et méthodique. Chaque étape fournit des informations et des accès qui facilitent l'étape suivante : des bibliothèques génériques vers des outils ciblés, vers une cible finale de prestige. Une campagne de plusieurs mois, avec

une planification qui rappelle les opérations APT étatiques — même si TeamPCP opère sous une identité criminelle.

Pourquoi vos défenses actuelles ne voient rien

Une entreprise qui a investi sérieusement en cybersécurité dispose typiquement d'un EDR sur les endpoints, d'un SIEM alimenté par les logs réseau et système, d'un WAF devant les applications web, d'un scanner de vulnérabilités sur les serveurs, et d'un bastion pour les accès administrateurs. Aucun de ces outils n'est efficace contre les attaques que nous venons de décrire.

L'EDR est aveugle aux plugins IDE. Les agents EDR modernes — CrowdStrike Falcon, SentinelOne, Microsoft Defender for Endpoint — détectent les comportements malveillants à l'échelle des processus système : shellcode en mémoire, injection de processus, persistance via clés de registre. Une extension VS Code qui lit `~/.aws/credentials` et envoie le contenu en HTTPS, c'est, du point de vue de l'EDR, VS Code qui fait une requête réseau HTTPS. Comportement normal. Aucune alerte.

Le SIEM ne voit pas l'exfiltration vers les CDN légitimes. L'exfiltration via GitHub API, Amazon S3, Google Drive ou Pastebin génère du trafic HTTPS chiffré vers des domaines sur liste blanche dans n'importe quel proxy d'entreprise. Même avec du SSL inspection, le contenu peut être obfusqué pour échapper aux règles de corrélation standard.

Le scanner de vulnérabilités ne scan pas le Marketplace. Les scanners ciblent les CVE dans les composants systèmes et applicatifs. Une extension VS Code malveillante n'est pas une CVE répertoriée — c'est du code malveillant dans une extension qu'aucun référentiel ne connaît encore.

Le WAF protège les serveurs, pas les postes. Les attaques supply chain visent les postes développeurs et les pipelines CI/CD, qui ne passent pas par le WAF d'entreprise.

Le problème culturel : la confiance implicite dans l'outillage

Au-delà des lacunes techniques, il y a un problème culturel profond. Les développeurs font confiance à leurs outils — c'est rationnel et nécessaire. Mais cette confiance s'est étendue, par habitude et par commodité, aux sources tierces : le Marketplace, npm, PyPI, Docker Hub. Des écosystèmes conçus pour la facilité d'usage, avec une sécurité qui n'était pas la priorité lors de leur conception.

La plupart des organisations n'ont pas de politique formelle sur les extensions IDE. Elles ont une politique sur les logiciels installables via GPO ou MDM, mais VS Code et ses extensions sont souvent dans un angle mort : trop complexes à contrôler sans bloquer la productivité, pas assez visibles pour figurer dans les politiques de sécurité traditionnelles.

Ce qu'il faut mettre en place maintenant

Voici les mesures concrètes classées par priorité. J'ai volontairement retenu des mesures applicables dans des délais réalistes, pas une liste de souhaits inapplicable sans budget supplémentaire.

1. Inventaire et contrôle des extensions IDE — effort : moyen, impact : élevé

Commencez par l'inventaire. Déployez un script via votre outil de gestion de parc (Intune, Jamf, Ansible) pour collecter la liste des extensions VS Code sur l'ensemble des postes développeurs. Cette seule donnée est souvent révélatrice : on y trouve régulièrement des extensions inconnues, non maintenues, ou présentant des patterns suspects.

Établissez une liste blanche d'extensions autorisées basée sur trois critères : éditeur vérifié (badge Microsoft ou organisation reconnue), plus de 100 000 installations, dernière mise à jour de moins de six mois. Toute extension hors liste blanche devrait nécessiter une validation explicite de l'équipe sécurité avant installation.

VS Code propose nativement le "Restricted Mode" qui limite les capacités des extensions dans les workspaces non approuvés. Activez-le par politique pour tous les postes développeurs.

2. Sécurisation des pipelines CI/CD — effort : élevé, impact : critique

Les pipelines CI/CD combinent un accès étendu aux secrets avec une exécution automatique de code tiers. Les mesures prioritaires :

Épingler les versions et valider l'intégrité. Utiliser des lockfiles (`package-lock.json`, `poetry.lock`) et valider les hashes des packages installés. Des outils comme Socket Security ou Phylum peuvent analyser en temps réel les packages ajoutés aux projets pour détecter des comportements suspects — nouveau mainteneur, ajout de scripts postinstall, appels réseau inhabituels.

Déployer un registre de packages interne. Nexus Repository, JFrog Artifactory ou AWS CodeArtifact permettent de centraliser la gestion des packages, d'appliquer des politiques d'approbation et d'éviter la dependency confusion (résolution prioritaire des noms internes sur le registre public).

Segmenter les secrets CI/CD. Chaque job de pipeline ne doit accéder qu'aux secrets dont il a strictement besoin, avec des credentials à durée de vie limitée (tokens OIDC, rôles IAM éphémères) plutôt que des API keys statiques stockées en variables d'environnement du pipeline.

3. Gestion des secrets sur les postes développeurs — effort : moyen, impact : élevé

Objectif : aucun secret à longue durée de vie en clair sur un poste développeur. En pratique c'est un chantier, mais il faut commencer quelque part.

Déployer un gestionnaire de secrets centralisé (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) avec injection dynamique de credentials de courte durée. Mettre en place des hooks pre-commit (git-secrets, truffleHog, Gitleaks) pour détecter et bloquer les secrets accidentellement committés dans les dépôts.

4. Formation ciblée des équipes développement — effort : faible, impact : immédiat

Les développeurs sont les premiers concernés et les premiers à même d'appliquer ces mesures — à condition d'en comprendre l'enjeu. Une formation de deux heures spécifiquement consacrée aux attaques supply chain, avec des démonstrations pratiques (montrer concrètement comment une extension VS Code peut lire des secrets en quelques lignes de code), est infiniment plus efficace que dix heures de sensibilisation générique.

Points clés : comment vérifier l'authenticité d'une extension IDE, comment détecter une anomalie dans un package npm (changement de mainteneur, ajout de scripts postinstall suspects), comment signaler une extension ou un package inhabituel, comment utiliser un gestionnaire de secrets plutôt que des variables d'environnement.

Le cadre réglementaire pousse dans ce sens

La directive NIS2, transposée en droit français depuis octobre 2024, inclut explicitement la sécurisation de la chaîne d'approvisionnement logicielle dans ses exigences pour les entités essentielles et importantes. L'article 21 de NIS2 impose aux organisations concernées d'identifier et de gérer les risques liés à leurs fournisseurs et prestataires de services — y compris les fournisseurs de logiciels et d'outils de développement.

DORA (Digital Operational Resilience Act), applicable aux entités financières depuis janvier 2025, va dans le même sens avec des exigences spécifiques sur la gestion des risques liés aux prestataires informatiques tiers et sur la sécurisation des processus de développement logiciel.

Au-delà de la conformité, ces réglementations reconnaissent ce que les acteurs offensifs ont compris depuis longtemps : la chaîne d'approvisionnement logicielle est un vecteur d'attaque systémique qui doit être traité comme tel.

AVIS DrXPERT

Mon avis d'expert

La question n'est plus de savoir si votre chaîne d'outillage développeur sera ciblée, mais quand. TeamPCP a démontré en 2026 qu'une campagne supply chain bien conduite peut

> atteindre GitHub — une entreprise dont la sécurité est le métier. Si GitHub peut se faire compromettre via une extension VS Code malveillante, aucune organisation qui développe ou consomme des logiciels n'est à l'abri.
>
> Ce que j'observe sur le terrain : les équipes sécurité commencent à inclure les pipelines CI/CD dans leurs périmètres d'audit, mais les postes développeurs restent largement non audités. Les politiques de gestion des extensions IDE n'existent presque nulle part. Les secrets sur les postes développeurs ne font l'objet d'aucun inventaire systématique. Ce sont ces trois angles morts qui seront exploités dans les 12 prochains mois, à une échelle et avec une sophistication croissantes.
>
> La sécurisation de la toolchain développeur n'est pas un problème de sécurité applicative. C'est un problème de sécurité opérationnelle qui relève de la même priorité que la sécurisation des accès administrateurs. Il faut le traiter en conséquence — avec un périmètre défini, un budget, des indicateurs de suivi et une gouvernance claire.

Conclusion : l'ère du zéro confiance dans l'outillage

La compromission de GitHub par TeamPCP en mai 2026 n'est pas un incident isolé. Elle est l'aboutissement logique d'une tendance de fond qui s'accélère depuis 2023 : les acteurs APT et les groupes criminels sophistiqués ont identifié la chaîne d'outillage des développeurs comme le maillon faible systématique de la défense en profondeur des entreprises.

Les serveurs de production sont de mieux en mieux protégés. Les accès administrateurs sont de plus en plus contrôlés. Alors on attaque là où les défenses sont absentes : les postes développeurs, les extensions IDE, les packages open source, les pipelines CI/CD. Les vecteurs d'entrée se déplacent vers les outils plutôt que vers les systèmes.

La réponse est conceptuellement simple : étendre le principe de zéro confiance aux outils de développement. Ne pas faire confiance par défaut à une extension IDE, à un package npm, à un template de projet. Valider, inventorier, contrôler. Traiter les postes développeurs avec le même

niveau d'exigence que les serveurs de production. Inclure les pipelines CI/CD dans les périmètres d'audit.

En pratique, c'est un effort organisationnel et culturel important — les développeurs n'aiment pas les contraintes qui ralentissent leur travail, et ils ont raison de les questionner. La clé est de trouver des mesures qui offrent un niveau de protection significatif sans friction excessive : une liste blanche d'extensions automatiquement déployée et maintenue par l'équipe sécurité, un registre de packages interne transparent pour les développeurs, des hooks pre-commit qui tournent en arrière-plan. La sécurité doit s'intégrer dans les workflows existants, pas les remplacer.

À RETENIR

Points clés à retenir

La toolchain du développeur : une surface d'attaque stratégique et massivement sous-estimée

Anatomie d'une attaque par extension VS Code : comment ça fonctionne réellement

Le vecteur npm et PyPI : quand le gestionnaire de packages devient cheval de Troie

Chronologie TeamPCP 2026 : anatomie d'une campagne en cascade

Pourquoi vos défenses actuelles ne voient rien

Sources et références

ANSSI — Bonnes pratiques de sécurité

CISA — Ressources cybersécurité

ENISA — Threat Landscape 2024

