Sécurité Serverless 2026 : Lambda, Azure Functions et Cloud Run

> Sécurité serverless 2026 — event injection Lambda, IAM least privilege, supply chain et monitoring pour protéger vos architectures serverless AWS, Azure et GCP.

En janvier 2025, lors d'un pentest d'une fintech française ayant migré son back-end sur AWS Lambda, notre équipe a découvert une **event injection** critique dans une fonction Lambda qui traitait des événements SQS sans validation des entrées : un attaquant pouvait injecter des commandes Shell via un message SQS malformé qui était ensuite passé sans sanitisation à une commande `subprocess.run()` dans le code de la Lambda, permettant l'exécution de code arbitraire dans l'environnement d'exécution de la fonction. Pire : la fonction Lambda disposait d'un rôle IAM avec les permissions `s3:*` sur tous les buckets S3 du compte, permettant l'exfiltration complète de 18 mois de données de transactions financières clients depuis l'environnement d'exécution compromis. Ce cas illustre parfaitement les nouvelles menaces spécifiques aux architectures serverless : les surfaces d'attaque sont différentes des applications traditionnelles, les outils de sécurité classiques ne fonctionnent pas toujours, et les erreurs de configuration IAM ont des conséquences amplifiées par la nature éphémère et auto-scaling des fonctions. Mon avis est clair : le serverless réduit la charge opérationnelle mais ne réduit pas la surface d'attaque — elle la déplace vers les permissions IAM, les triggers de fonctions, et les dépendances du code. Ce guide technique approfondi couvre la sécurité des architectures **serverless** en 2026 : vulnérabilités spécifiques aux fonctions **AWS Lambda**, **Azure Functions**, et **Google Cloud Run**, techniques d'attaque event injection et cold start, principes IAM least privilege pour le serverless, monitoring et détection des menaces avec les outils cloud-natifs, et sécurisation des chaînes d'approvisionnement des dépendances pour les organisations françaises soumises à NIS 2.

À RETENIR

À retenir

• L'**event injection** est la menace serverless numéro 1 : les fonctions traitent des événements provenant de sources non fiables (SQS, API Gateway, S3, DynamoDB) sans validation des entrées, permettant l'injection de commandes OS, SQL, ou NoSQL via des données d'événements malformées

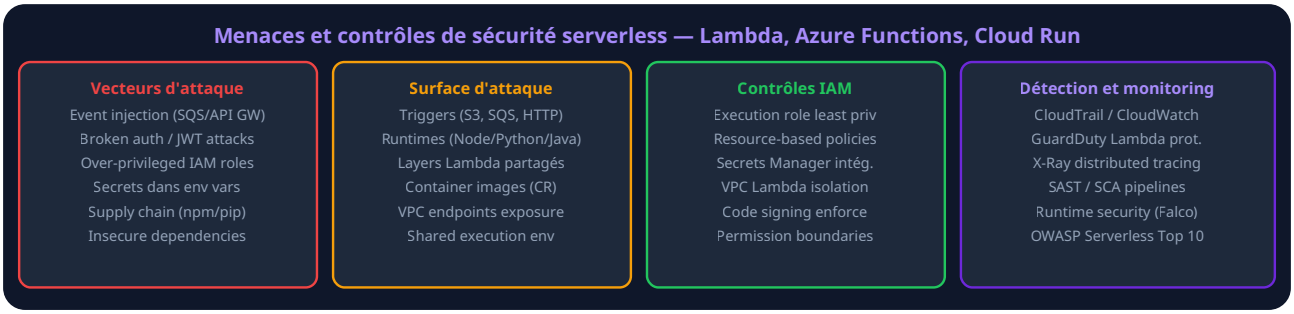
• Le **principe du moindre privilège IAM** est critique pour le serverless : chaque fonction Lambda/Azure Function/Cloud Run doit avoir son propre rôle d'exécution avec uniquement les permissions nécessaires pour ses actions spécifiques — un rôle générique avec s3:* ou storage.admin compromet toutes les données cloud si la fonction est compromise

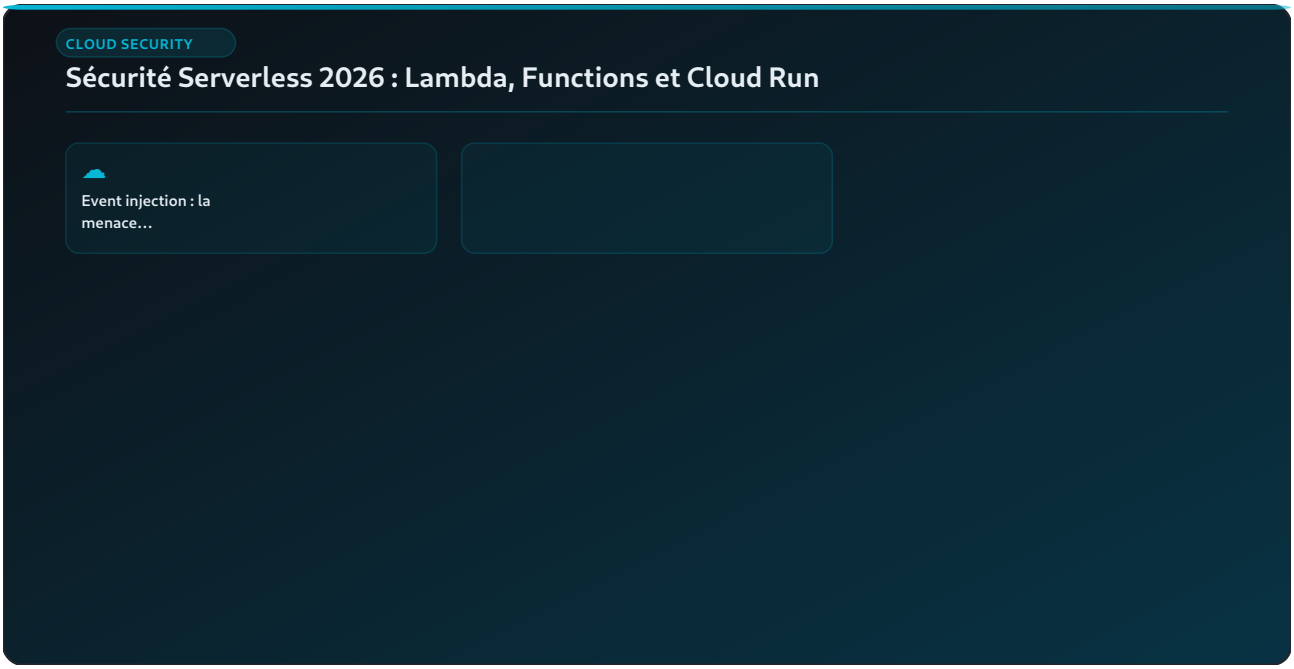
• Les **variables d'environnement serverless** ne doivent jamais contenir de secrets en clair — utiliser AWS Secrets Manager, Azure Key Vault, ou GCP Secret Manager avec des références à la configuration de la fonction plutôt que des valeurs directes dans les variables d'environnement

• Le **cold start** n'est pas seulement un problème de performance : des attaques d'amplification exploitent les cold starts pour saturer les ressources serverless et provoquer des comportements inattendus dans la gestion des erreurs ou des timeouts

• Le **SAST et SCA des dépendances** est indispensable pour les fonctions serverless car les packages npm/pip/maven inclus dans les archives de déploiement constituent la surface d'attaque principale via des vulnérabilités de supply chain (ex: event-stream npm)

• L'**observabilité serverless** (logs structurés, traces distribuées, métriques d'invocation) est non seulement nécessaire pour le débogage mais est également la base de la détection des menaces — sans observabilité adéquate, une compromission d'une fonction Lambda peut passer inaperçue pendant des semaines





L'**event injection** est la vulnérabilité la plus fréquente et la plus impactante dans les architectures serverless. Contrairement aux injections traditionnelles qui ciblent des interfaces HTTP directes, l'event injection serverless exploite les données malveillantes injectées dans les événements qui déclenchent les fonctions : messages **SQS** ou **SNS** dont le corps contient du code malveillant, objets **S3** avec des noms de fichiers contenant des caractères spéciaux ou des commandes Shell, événements **DynamoDB Streams** avec des valeurs malformées dans les clés d'attributs, ou requêtes API Gateway avec des headers ou paramètres conçus pour exploiter le traitement des données côté Lambda. La particularité de l'event injection serverless est que l'attaquant peut

contrôler le vecteur d'injection sans accès direct à l'application — en publiant un message dans une queue SQS accessible ou en uploadant un fichier dans un bucket S3 public.

La prévention de l'event injection repose sur trois principes : **validation stricte des entrées** de tous les événements reçus (validation du schema JSON, vérification des types et formats, rejeter les valeurs inattendues), **utilisation de paramètres liés** pour toutes les requêtes de bases de données (pas de concaténation de chaînes pour construire des requêtes SQL, DynamoDB, ou MongoDB), et **élimination des appels système dangereux** dans le code des fonctions (pas de `subprocess.run(event.get('command'))`, pas d'`eval()` sur des données d'entrée, pas de `exec()` dynamique). L'analyse statique du code (SAST) avec des outils comme **Semgrep** ou **Bandit** (Python) dans les pipelines CI/CD identifie automatiquement ces patterns dangereux avant le déploiement en production, bloquant les fonctions vulnérables à l'event injection avant qu'elles ne soient accessibles aux attaquants.

IAM serverless : rôles d'exécution et moindre privilège

Le *rôle d'exécution* (Execution Role) d'une fonction Lambda, ou l'identité managée d'une Azure Function, ou le compte de service d'un Cloud Run service, est la surface d'attaque IAM principale du serverless. Une fonction serverless compromise peut utiliser les credentials de son rôle d'exécution pour appeler n'importe quelle API cloud couverte par ses permissions. La règle d'or : chaque fonction doit avoir son propre rôle d'exécution dédié avec uniquement les permissions nécessaires pour ses opérations spécifiques — pas de rôles partagés entre fonctions, pas de rôles génériques comme `AmazonS3FullAccess` ou `AdministratorAccess`. Pour une fonction Lambda qui lit des messages SQS et écrit dans un bucket S3, le rôle d'exécution doit contenir uniquement `sqs:ReceiveMessage`, `sqs:DeleteMessage`, et `s3:PutObject` sur le bucket S3 spécifique cible.

⚡ **Retour terrain**

Pour une collectivité qui migrait sa messagerie vers Microsoft 365, la configuration initiale d'Exchange Online autorisait l'accès IMAP/POP3 — des protocoles legacy qui contournent le MFA. En 3 mois d'existence, 4 comptes avaient été compromis via credential stuffing sur ces protocoles. La désactivation des protocoles d'authentification basique, recommandée depuis 2019 par Microsoft, n'avait jamais été activée.

Les **Permission Boundaries** AWS sont un mécanisme avancé qui définit le maximum de permissions qu'un rôle peut avoir, indépendamment des politiques IAM attachées. Pour les équipes DevOps qui créent des rôles Lambda automatiquement via Terraform ou CloudFormation, une Permission Boundary organisationnelle qui limite les permissions maximales accordables à un rôle Lambda (pas de permissions IAM:*, pas de permissions vers des ressources hors du projet) empêche la création accidentelle de rôles sur-privilégiés même si le développeur commet une erreur dans la définition du rôle d'exécution. L'**IAM Access Analyzer** AWS analyse les politiques IAM et génère des recommandations de réduction de permissions basées sur l'activité réelle des derniers 90 jours via la fonctionnalité "Generate Least Privilege Policy" — idéal pour affiner les rôles Lambda en production en partant des permissions utilisées réellement.

Gestion des secrets dans les architectures serverless

La gestion des secrets dans les fonctions serverless est un défi spécifique : sans serveurs persistants à configurer une seule fois, chaque nouvelle instance de la fonction doit récupérer ses secrets au démarrage. Les pratiques à éviter absolument : stocker les secrets dans les variables d'environnement en clair (visibles dans la console AWS/Azure/GCP, dans les logs de déploiement, et dans les snapshots de configuration exportés), hardcoder les secrets dans le code

source (risque d'exposition via les dépôts Git), ou stocker les secrets dans les layers Lambda partagés entre plusieurs fonctions. La solution recommandée est l'utilisation des services de gestion de secrets cloud-natifs : **AWS Secrets Manager**, **Azure Key Vault**, ou **GCP Secret Manager**, avec récupération programmatique du secret au démarrage de la fonction via l'API du service, en utilisant le rôle d'exécution de la fonction pour l'authentification (pas de credentials hardcodés pour accéder au secrets manager).

Une optimisation importante pour les performances serverless : utiliser un pattern de **secret caching** dans la fonction pour récupérer le secret une seule fois par instance d'exécution chaude et le stocker en mémoire, évitant un appel API Secrets Manager à chaque invocation (latence supplémentaire de 5-20ms et coût par appel API). Le **AWS Parameters and Secrets Lambda Extension** (Lambda Layer officiel AWS) implémente ce caching côté infrastructure sans modification du code de la fonction — le secret est récupéré via un endpoint localhost HTTP exposé par l'extension, avec cache en mémoire et rafraîchissement automatique avant l'expiration. Pour les secrets à rotation automatique (mots de passe de bases de données, clés API avec rotation courte), la rotation native de Secrets Manager avec des Lambdas de rotation dédiées garantit que les fonctions récupèrent toujours le secret courant via le cache, sans code de gestion de rotation dans la fonction applicative.

Cold start et attaques de déni de service serverless

Le **cold start** serverless est le temps d'initialisation d'une nouvelle instance de la fonction (de 100ms à plusieurs secondes selon le runtime et la taille du package) qui se produit lorsqu'il n'y a pas d'instance chaude disponible pour traiter l'invocation. Du point de vue sécurité, les cold starts créent plusieurs vecteurs d'attaque : les **attaques par amplification de cold start** qui envoient des milliers d'invocations simultanées pour forcer la création de centaines d'instances froides, saturant les quotas de concurrence de la fonction et créant un déni de service effectif ; les **attaques par timing** qui exploitent les différences de temps de réponse entre cold et warm starts pour déduire l'état d'initialisation des fonctions et inférer des informations sur l'environnement d'exécution ; et les **vulnérabilités dans le code d'initialisation** qui est exécuté uniquement au

cold start (code de connexion à la base de données, de chargement des configurations) — souvent moins testé et plus sujet aux erreurs que le code de traitement principal.

La mitigation des attaques de déni de service sur les fonctions Lambda s'effectue via plusieurs mécanismes : la configuration de **reserved concurrency** (concurrence réservée) pour limiter le nombre maximum d'instances simultanées d'une fonction à un seuil raisonnable, empêchant qu'une attaque DDoS ne consomme tout le quota de concurrence du compte AWS ; les **throttling policies** via Amazon API Gateway (limite de débit par IP, par API key, ou par plan d'usage) pour les fonctions exposées via HTTP ; et les **SQS visibility timeouts** et les **dead letter queues (DLQ)** pour les fonctions déclenchées par des queues SQS, limitant l'impact des messages malformés qui causent des erreurs répétitives. La fonctionnalité **Lambda Provisioned Concurrency** (concurrence provisionnée) élimine les cold starts en maintenant des instances préchauffées mais à un coût plus élevé — à réserver aux APIs critiques avec des SLAs de latence stricts.

OWASP Serverless Top 10 : vulnérabilités de référence

L'**OWASP Serverless Top 10** (disponible sur owasp.org) documente les 10 catégories de vulnérabilités les plus critiques spécifiques aux architectures serverless, adapté de l'OWASP Web Application Security Top 10 aux contraintes des fonctions serverless. Les 5 premières catégories les plus impactantes en 2026 sont : **SAS1 - Injection** (event injection via tous types d'inputs), **SAS2 - Broken Authentication** (tokens JWT mal validés dans les API Gateway authorizers, absence de vérification de l'audience du token, algorithmes de signature faibles), **SAS3 - Sensitive Data Exposure** (secrets dans les variables d'environnement, logs contenant des données sensibles, réponses d'erreur avec stack traces exposant l'architecture interne), **SAS4 - XML External Entities (XXE)** (parsers XML utilisés pour traiter des événements XML sans désactivation des entités externes), et **SAS5 - Broken Access Control** (rôles IAM sur-privilégiés, absence de validation des autorisations au niveau de la fonction en plus de l'API Gateway).

Les 5 catégories restantes de l'OWASP Serverless Top 10 couvrent : **SAS6 - Security Misconfiguration** (fonctions exposées sans authentification, buckets S3 de déploiement publics, VPC mal configurés), **SAS7 - Cross-Site Scripting (XSS)** (fonctions générant du HTML côté

serveur sans encodage des outputs dans les applications serverless full-stack), **SAS8 - Insecure Deserialization** (désérialisation non sécurisée de données JSON/YAML/Pickle dans les payloads d'événements), **SAS9 - Using Components with Known Vulnerabilities** (dépendances npm/pip avec des CVEs critiques non patchées dans les packages de déploiement), et **SAS10 - Insufficient Logging and Monitoring** (absence de logs structurés, pas de traces distribuées, impossibilité de reconstituer un incident). Ces catégories forment le référentiel de test pour les pentests de fonctions serverless et pour la revue de code sécurité des architectures serverless.

GuardDuty Lambda Protection et CloudTrail : détection des menaces

Amazon GuardDuty Lambda Protection est une fonctionnalité d'AWS GuardDuty (service de détection des menaces AWS) qui surveille les activités réseau des fonctions Lambda en analysant les VPC Flow Logs générés par les exécutions Lambda et les DNS query logs. GuardDuty Lambda Protection peut détecter : `Lambda/CryptoCurrency.CoinMining.Instance.CryptoCurrencyMiningActivity` (communication vers des pools de minage depuis une Lambda compromise), `Lambda/Backdoor.Lambda.C&CActivity.B` (communication vers une infrastructure de Command and Control connue), et `Lambda/Execution.Lambda.AnomalousBehavior` (comportement réseau inhabituel de la Lambda comparé à sa baseline normale). Ces findings GuardDuty pour Lambda apparaissent dans la console GuardDuty et peuvent être routés vers Amazon EventBridge pour déclencher des réponses automatisées (désactivation de la fonction compromise, notification de l'équipe SOC).

AWS CloudTrail journalise toutes les invocations d'API AWS effectuées depuis les fonctions Lambda, permettant de tracer précisément quelles ressources une fonction a accédées et quelles opérations elle a effectuées. Les appels API Lambda eux-mêmes (CreateFunction, UpdateFunctionCode, InvokeFunction, GetFunction) sont également journalisés dans CloudTrail, permettant de détecter des modifications non autorisées du code des fonctions ou des invocations d'urgence de fonctions sensibles en dehors des processus de déploiement normaux. La requête CloudWatch Logs Insights `filter @message like "ERROR" | stats count() as errorCount by @logStream | sort errorCount desc | limit 20` identifie rapidement les fonctions Lambda qui génèrent des erreurs en excès — un indicateur potentiel d'une exploitation

en cours de vulnérabilités d'injection ou de problèmes de permissions. Notre service de RSSI externalisé inclut la configuration de l'observabilité sécurité serverless pour les organisations AWS françaises.

Sécurité Azure Functions : identités managées et RBAC

La sécurisation d'**Azure Functions** suit des principes similaires à AWS Lambda mais avec les spécificités de l'écosystème Azure. L'authentification des Azure Functions aux autres services Azure doit utiliser exclusivement les **Managed Identities** (System-assigned ou User-assigned) plutôt que des Connection Strings ou des clés de compte de stockage — une Managed Identity Azure AD elimine tous les credentials statiques en déléguant l'authentification à Azure AD avec des tokens à courte durée de vie. La configuration du **RBAC Azure** avec les rôles intégrés ou personnalisés les plus granulaires doit être appliquée pour chaque ressource Azure accédée par la Function App (Storage Blob Data Reader pour un accès lecture aux blobs, plutôt que Storage Account Contributor qui donne des droits d'administration complets).

Les vulnérabilités spécifiques aux Azure Functions incluent les **Function Keys** exposées accidentellement : les Azure Functions HTTP-triggered utilisent des clés de fonction (Function Keys) pour l'authentification des requêtes HTTP en mode "Function" ou "Admin". Ces clés, si exposées dans des repositories Git, des variables CI/CD, ou des logs d'application, permettent l'invocation non autorisée des fonctions. La bonne pratique est d'utiliser l'authentification Azure AD (mode AuthLevel "Anonymous" avec validation du token Bearer dans le code de la fonction, ou mode EasyAuth intégré d'Azure App Service) plutôt que les Function Keys statiques pour les fonctions exposant des APIs. L'intégration de **Defender for App Service** (Microsoft Defender for Cloud) pour les Azure Functions fournit une détection des menaces au niveau runtime similaire à GuardDuty Lambda Protection pour AWS, avec des alertes spécifiques aux comportements anormaux des Function Apps.

Sécurité Cloud Run GCP : containers serverless et isolation

Cloud Run est le service serverless de GCP qui exécute des containers HTTP stateless à la demande, avec une isolation renforcée par rapport aux FaaS (Function as a Service) classiques comme Lambda car chaque instance Cloud Run s'exécute dans un container isolé avec son propre filesystem. La sécurisation de Cloud Run commence par la sécurisation des images containers : utiliser des **images de base distroless** (sans shell, sans gestionnaire de packages, sans outils de débogage) pour réduire la surface d'attaque des containers Cloud Run, activer le **scan automatique de vulnérabilités** dans Artifact Registry pour toutes les images déployées sur Cloud Run, et configurer **Binary Authorization** pour rejeter les déploiements d'images sans attestation de scan réussi. La commande `gcloud run services update [SERVICE] --no-allow-unauthenticated` désactive l'accès public non authentifié au service Cloud Run, exigeant un token Bearer IAM valide pour toute invocation.

La configuration réseau de Cloud Run est un point de sécurité souvent négligé : par défaut, les services Cloud Run reçoivent un endpoint HTTPS public accessible depuis Internet. Pour les services internes, la configuration `--ingress=internal` restreint l'accès aux requêtes provenant du VPC interne via Private Service Connect, et `--vpc-connector=[CONNECTOR]` permet au service Cloud Run d'accéder aux ressources dans le VPC privé (bases de données Cloud SQL, Memorystore) sans exposition sur Internet. Les **Cloud Run jobs** (tâches planifiées ou déclenchées par événements, sans endpoint HTTP) bénéficient de la même configuration de sécurité que les services mais n'exposent aucune surface d'attaque réseau directe, les rendant particulièrement adaptés aux traitements de données sensibles comme les pipelines ETL manipulant des données personnelles soumises au RGPD. Notre service de audit de sécurité cloud couvre la revue des configurations Cloud Run dans le cadre des audits GCP.

Supply chain serverless : sécurité des dépendances npm/pip

La **supply chain attack** est une menace particulièrement redoutable pour les fonctions serverless car les packages de déploiement (ZIP Lambda, images containers Cloud Run) incluent toutes

leurs dépendances — signifiant qu'une vulnérabilité dans un package npm ou pip transitif se retrouve déployée dans chaque instance de la fonction. L'incident **event-stream npm** (2018) a démontré la réalité de ce vecteur : un package npm légitime a été compromis par son nouveau mainteneur pour injecter du code malveillant visant les portefeuilles de crypto-monnaies — ce code malveillant s'est retrouvé dans des milliers d'applications et fonctions serverless utilisant le package comme dépendance transitive. La vérification des dépendances avec **npm audit**, **pip-audit**, ou **Snyk** dans les pipelines CI/CD est indispensable pour détecter les packages avec des CVEs critiques avant le déploiement.

Les **Software Bill of Materials (SBOM)** sont devenus un prérequis réglementaire dans certains secteurs et sont recommandés par l'ANSSI pour les organisations soumises à NIS 2 déployant des composants logiciels critiques. Un SBOM est un inventaire exhaustif de tous les composants logiciels (dépendances directes et transitives) inclus dans un package de déploiement serverless, avec leurs versions exactes et leurs licences. Des outils comme **Syft** (open-source, by Anchore) génèrent automatiquement des SBOMs au format SPDX ou CycloneDX depuis des répertoires de code ou des images containers, et **Grype** (by Anchore) analyse ces SBOMs contre les bases de vulnérabilités CVE, OSV, et GHSA. L'intégration de la génération de SBOM et du scan de vulnérabilités dans le pipeline CI/CD de déploiement Lambda crée une traçabilité complète des composants déployés, utile pour la réponse rapide lors de la divulgation de nouvelles vulnérabilités critiques affectant une dépendance. Notre service DevSecOps inclut la mise en place de ces pipelines de scan pour les architectures serverless.

Runtime security serverless : Falco et extensions Lambda

La **sécurité runtime** des fonctions serverless — la détection des comportements malveillants pendant l'exécution de la fonction — est techniquement plus difficile que pour les containers traditionnels car les environnements d'exécution serverless sont éphémères, sans agent de sécurité persistant, et avec un accès limité aux mécanismes de surveillance système (ptrace, eBPF). Pour AWS Lambda, les **Lambda Extensions** (processus s'exécutant dans l'environnement Lambda en parallèle du code de la fonction) permettent de déployer des agents de sécurité runtime légers : **Falco Lambda Extension** (CNCF) surveille les appels système de la

fonction Lambda via **eBPF** pour détecter des comportements suspects (exécution de processus inattendus, écriture hors du répertoire tmp, connexions réseau vers des IPs inconnues), et déclenche des alertes en temps réel lors de leur détection.

Pour Cloud Run (containers), **Falco** peut être déployé comme sidecar container ou comme DaemonSet dans GKE pour surveiller les syscalls de tous les containers, y compris les instances Cloud Run déployées sur des clusters GKE. Les *règles Falco* prédéfinies pour la détection de menaces containers incluent : détection de l'exécution d'un shell dans un container qui ne devrait pas en avoir (`spawned_process in container`), détection d'écriture dans des répertoires système sensibles (`write_etc_symlink`), et détection de connexions vers des IPs dans des listes noires de Threat Intelligence. L'intégration des alertes Falco avec des plateformes SIEM (Chronicle, Sentinel, Splunk) via des webhooks ou des Pub/Sub permet de corréler les alertes runtime avec les autres signaux de sécurité cloud pour une détection contextuelle des compromissions de fonctions serverless en production.

Sécurité API Gateway : authentification et rate limiting

L'**API Gateway** (AWS API Gateway, Azure API Management, GCP Cloud Endpoints) est le point d'entrée HTTP des fonctions serverless et constitue la première ligne de défense contre les attaques applicatives. La configuration sécurisée d'un API Gateway pour des Lambda Functions inclut : l'authentification des requêtes via des **Lambda Authorizers** (fonctions Lambda qui valident les tokens JWT/OAuth2 avant de router la requête vers la fonction cible), la configuration du **rate limiting** et du throttling (quota maximum de requêtes par seconde par IP ou par API Key), l'activation du **WAF AWS (Web Application Firewall)** avec les règles managées AWS Core Rule Set (équivalent du OWASP ModSecurity Core Rule Set) pour bloquer les injections SQL, XSS, et les scanners connus, et la configuration des **CORS policies** restrictives pour les APIs consommées depuis des applications web front-end.

Les erreurs de configuration d'API Gateway les plus courantes découvertes lors des pentests serverless incluent : l'absence de validation du token JWT (l'endpoint valide la présence d'un header Authorization mais pas la signature du token), l'autorisation de méthodes HTTP non nécessaires (un endpoint GET qui accepte également des requêtes DELETE ou PUT non

protégées), l'exposition de stages de développement ou de test sur des endpoints publics sans authentification, et l'absence de logging des requêtes refusées par le WAF (rendant impossible la détection d'attaques en cours). La révision régulière des configurations API Gateway avec des outils comme **AWS Scout Suite** ou des règles AWS Config personnalisées (`aws-config-rules` open-source) permet d'identifier automatiquement ces misconfigurations dans les organisations avec de nombreux API Gateways gérés par des équipes DevOps différentes.

Sécurité IaC pour le serverless : Terraform et CloudFormation

La sécurisation des déploiements serverless passe en grande partie par la qualité de l'**Infrastructure as Code (IaC)** qui les définit. Les erreurs de configuration IAM dans les rôles Lambda, les expositions d'endpoints non sécurisés dans API Gateway, et les misconfigurations de VPC pour les fonctions sont souvent introduites dans les fichiers Terraform ou CloudFormation et se propagent automatiquement dans tous les environnements (dev, staging, production) lors des déploiements. L'analyse statique des fichiers IaC avec des outils comme **Checkov** (open-source, par Bridgecrew/Palo Alto Networks), **tfsec**, ou **Terrascan** détecte automatiquement des centaines de règles de configuration incorrecte dans les ressources Terraform AWS Lambda, Azure Function App, et GCP Cloud Run avant le déploiement. L'intégration de ces outils dans les pipelines CI/CD bloque les déploiements IaC non conformes aux politiques de sécurité organisationnelles.

Le framework **Open Policy Agent (OPA) / Conftest** permet de définir des politiques de sécurité IaC en Rego (le langage de politique d'OPA) et de les appliquer sur n'importe quel fichier IaC (Terraform HCL, CloudFormation YAML, Kubernetes manifests). Une politique OPA pour le serverless peut par exemple interdire la création de rôles Lambda avec des actions wildcard (`Action: "*"` ou `Resource: "*"` dans les politiques IAM), exiger que toutes les fonctions Lambda soient dans un VPC, ou interdire les Function URLs Lambda publiques sans authentification. Ces politiques OPA, stockées dans un dépôt Git et appliquées dans les pipelines CI/CD, créent des guardrails de sécurité automatisés qui ne nécessitent pas d'intervention manuelle d'un expert sécurité pour chaque pull request de code IaC.

Techniques de pentest spécifiques au serverless

Le pentest d'architectures serverless suit une méthodologie adaptée aux spécificités de ces environnements. La phase de **reconnaissance serverless** commence par l'identification des fonctions exposées : énumération des endpoints API Gateway via la documentation Swagger/OpenAPI exposée, découverte des Lambda Function URLs via le DNS AWS (`*.lambda-url.eu-west-3.on.aws`), et analyse des JavaScript front-end pour identifier les appels directs aux APIs serverless. La phase d'**exploitation de l'event injection** teste systématiquement chaque champ de l'API (body parameters, query strings, headers, path parameters) avec des payloads d'injection SQL, OS command, template injection (SSTI), et XXE pour les endpoints acceptant du XML.

Les techniques de pentest IAM spécifiques au serverless incluent l'**IAM privilege escalation serverless** : si une fonction Lambda dispose de la permission `iam:PassRole` combinée avec `lambda:CreateFunction` ou `lambda:UpdateFunctionCode` , un attaquant ayant compromis la fonction peut créer une nouvelle Lambda avec un rôle plus privilégié ou modifier le code d'une Lambda existante pour exploiter ses permissions. Des outils comme **Pacu** (AWS exploitation framework open-source, disponible sur github.com/RhinoSecurityLabs/pacu) automatisent l'énumération des chemins d'escalade de privilèges IAM dans les environnements AWS avec des modules dédiés aux Lambda privilege escalation, facilitant considérablement cette phase du pentest. La correction de ces chemins d'escalade passe par la suppression de la permission `iam:PassRole` des rôles Lambda et l'utilisation de Service Control Policies (SCP) AWS Organizations pour interdire ces combinaisons de permissions dangereuses au niveau organisationnel.

Sécurité des données dans les fonctions serverless

La manipulation de **données personnelles et sensibles** dans les fonctions serverless soulève des enjeux de conformité spécifiques. Une fonction Lambda qui traite des données RGPD (noms, emails, données de santé) doit garantir que ces données ne persistent pas dans les logs CloudWatch en dehors des champs strictement nécessaires à l'audit de sécurité — impliquant la

mise en place d'un **masquage des données sensibles dans les logs** (email_hash plutôt que email_address, last_4_digits plutôt que numéro complet de carte bancaire). AWS CloudWatch Logs propose une fonctionnalité de **Data Protection Policies** qui masque automatiquement les données personnelles sensibles dans les logs Lambda (numéros de carte de crédit, SSN, emails) en les remplaçant par des tokens avant stockage, réduisant le risque d'exposition de données personnelles dans les logs de monitoring.

Le chiffrement des données dans les fonctions serverless suit des contraintes spécifiques : la mémoire d'exécution Lambda n'est pas chiffrée (les données en clair sont accessibles en mémoire pendant l'exécution), le répertoire `/tmp` de Lambda (512 MB par défaut, jusqu'à 10 GB configurables) peut être chiffré avec des clés KMS via la configuration `ephemeral_storage.encryption_key` dans Terraform, et les variables d'environnement Lambda peuvent être chiffrées avec des clés KMS personnalisées via la configuration `kms_key_arn`. Pour les fonctions traitant des données de santé (HDS), financières (PCI DSS), ou de secrets d'État, l'utilisation de fonctions Lambda dans un VPC avec accès aux services via des VPC Endpoints privés (sans transit Internet) et chiffrement KMS de toutes les données au repos et en transit est le minimum requis. Notre service de mise en conformité NIS 2 inclut une évaluation de la conformité des architectures serverless pour les données sensibles.

Monitoring et observabilité sécurité serverless

L'**observabilité serverless** orientée sécurité requiert trois piliers complémentaires : les **logs structurés** (JSON avec des champs standardisés : timestamp, request_id, user_id, action, resource, result, IP source), les **métriques** (taux d'erreurs, latence, concurrence, nombre d'invocations par déclencheur) et les **traces distribuées** (AWS X-Ray, Azure Application Insights, GCP Cloud Trace) qui permettent de suivre une requête à travers plusieurs fonctions serverless et services cloud. Les logs structurés sont particulièrement importants pour la sécurité car ils permettent des requêtes précises via CloudWatch Logs Insights, Azure Monitor, ou BigQuery pour identifier les patterns d'attaque : par exemple, 1000 invocations avec des résultats d'erreur 403 depuis une même IP en 5 minutes indique une tentative de brute-force ou d'énumération d'API.

Les métriques de sécurité spécifiques à surveiller pour les fonctions serverless incluent : le **taux d'erreurs** par type (errors 4xx = tentatives non autorisées, errors 5xx = crashs potentiellement liés à des exploitations d'injection), la **durée d'exécution** anormalement longue par rapport à la baseline (une fonction Lambda qui prend 50x plus de temps que d'habitude peut être en train de miner des crypto-monnaies), la **concurrence** inhabituelle (un pic soudain d'instances parallèles d'une fonction non exposée publiquement indique une activation depuis une source non prévue), et les **cold starts** excessifs (possible tentative de force de création de nouvelles instances pour exploiter le code d'initialisation). L'intégration de ces métriques dans des dashboards CloudWatch, Azure Monitor, ou Cloud Monitoring avec des alertes automatiques vers PagerDuty ou les équipes SOC garantit une réponse rapide aux anomalies, critique dans des architectures à évolution rapide.

Plateforme	IAM / Identités	Détection menaces	Gestion secrets
AWS Lambda	Execution Role IAM + Permission Boundaries	GuardDuty Lambda Protection	AWS Secrets Manager + Lambda Extension
Azure Functions	Managed Identity + RBAC Azure	Defender for App Service	Azure Key Vault references
GCP Cloud Run	Service Account + Workload Identity	Container Threat Detection (SCC)	Secret Manager + env var refs
Cloudflair Workers	Worker tokens granulaires	Logpush + SIEM integration	Workers Secrets (chiffrés)

Sécurité des workflows serverless : Step Functions et Durable Functions

Les architectures serverless modernes utilisent fréquemment des orchestrateurs de workflows comme **AWS Step Functions** (pour les workflows Lambda multi-étapes) ou **Azure Durable Functions** (pour les workflows Azure Functions stateful) pour coordonner des pipelines complexes. La sécurité des Step Functions présente des défis spécifiques : chaque état du workflow peut avoir un rôle IAM différent pour accéder à des ressources spécifiques, et les données d'entrée/sortie de chaque état sont persistées dans l'état du workflow — si elles

contiennent des données sensibles, elles peuvent être exposées dans les logs Step Functions ou dans les historiques d'exécution accessibles via la console AWS. La configuration `includeExecutionData: false` dans la définition du State Machine désactive l'inclusion des données d'entrée/sortie dans les logs CloudWatch, réduisant le risque d'exposition accidentelle de données sensibles dans les logs d'orchestration.

Les **vulnérabilités de confusion de dépôt** (Dependency Confusion Attack) sont particulièrement dangereuses pour les environnements serverless avec des dépôts npm ou pip privés : si un package interne est publié sur un registre public avec le même nom mais une version plus élevée, les gestionnaires de packages installent automatiquement la version publique malveillante plutôt que la version privée. La mitigation utilise des scope npm privés (`@organisation/package`), des politiques Artifactory/Nexus qui interdisent la résolution de packages depuis les registres publics pour les noms de packages internes définis, et des contrôles de hash SHA dans les fichiers de lock (package-lock.json, requirements.txt avec hashes) pour garantir l'intégrité des dépendances installées. Ces contrôles sont critiques dans les pipelines de déploiement Lambda où le `npm install` ou `pip install` est exécuté automatiquement avant le packaging de la fonction.

Isolation réseau des fonctions serverless : VPC et VPC Endpoints

Par défaut, les fonctions AWS Lambda s'exécutent dans un environnement réseau géré par AWS (hors VPC du client), avec accès à Internet sortant mais sans accès direct aux ressources dans le VPC du client (RDS, Elasticache, EC2 internes). La configuration **Lambda dans VPC** permet à la fonction d'accéder aux ressources privées du VPC mais supprime l'accès Internet sortant par défaut — nécessitant un **NAT Gateway** dans le VPC pour les appels API Internet (APIs tierces, AWS APIs publiques). Cette architecture VPC pour Lambda augmente la sécurité réseau (isolement des fonctions dans des subnets privés avec des Security Groups restrictifs) mais ajoute latence (ENI attachment) et coût (NAT Gateway).

Les **VPC Endpoints Interface** (AWS PrivateLink) permettent aux fonctions Lambda dans un VPC d'accéder aux services AWS (S3, DynamoDB, Secrets Manager, SQS, SSM) via le réseau privé AWS sans passer par Internet, éliminant le besoin d'un NAT Gateway pour les appels AWS natifs. Un endpoint VPC pour Secrets Manager permet aux fonctions Lambda de récupérer leurs

secrets via le réseau privé, plus sécurisé et plus fiable que via Internet. La documentation des VPC Endpoints disponibles pour chaque service AWS est maintenue sur docs.aws.amazon.com. L'utilisation systématique de VPC Endpoints pour les services AWS critiques (Secrets Manager, KMS, S3, DynamoDB) dans les environnements Lambda sensibles élimine l'exposition des communications Lambda/Services sur le réseau Internet public, réduisant la surface d'attaque réseau de l'architecture serverless.

Pipeline DevSecOps serverless : intégration sécurité continue

Un pipeline **DevSecOps serverless** complet intègre des contrôles de sécurité automatisés à chaque étape du cycle de vie de la fonction. La phase de **développement** utilise des linters de sécurité dans l'IDE (extensions VS Code pour Semgrep, Bandit, ou eslint-plugin-security) pour détecter les vulnérabilités au moment de l'écriture du code. La phase de **build CI** exécute le SAST (Semgrep, Bandit, CodeQL), le SCA (Snyk, npm audit, pip-audit), la génération du SBOM (Syft), et l'analyse des fichiers IaC (Checkov, tfsec). La phase de **déploiement** avec Terraform ou SAM vérifie les politiques OPA, crée les attestations Binary Authorization (pour Cloud Run), et applique les configurations CMEK et VPC. La phase de **runtime** surveille avec GuardDuty Lambda Protection, Falco Extensions, et les métriques CloudWatch avec alertes automatiques sur les anomalies.

La **documentation de sécurité des fonctions serverless** est un prérequis pour la conformité NIS 2 : chaque fonction doit être documentée avec son rôle IAM d'exécution et sa justification, les sources de données qu'elle traite (avec classification de sensibilité), les APIs tierces qu'elle appelle, les secrets qu'elle utilise et leur emplacement dans Secrets Manager, et les procédures de réponse à incident en cas de compromission. Cette documentation, maintenue dans le dépôt Git avec le code de la fonction et les fichiers IaC, garantit que les informations de sécurité sont toujours synchronisées avec la configuration réelle de la fonction et accessibles rapidement lors d'un incident. Pour les organisations avec des dizaines ou centaines de fonctions serverless, des outils de discovery automatique comme **Yor** (tagging IaC automatique) ou les AWS Config Rules personnalisées permettent d'automatiser l'inventaire des fonctions et la vérification de leur conformité aux politiques de sécurité documentées. Notre service de sécurité applicative

accompagne les équipes DevOps dans la mise en place de ce pipeline DevSecOps serverless complet.

Questions fréquentes sur la sécurité serverless

Le serverless est-il intrinsèquement plus sécurisé que les VMs classiques ?

Pas nécessairement — la surface d'attaque est différente, pas réduite. Le serverless élimine les responsabilités de gestion du système d'exploitation et du serveur (patchs, hardening) mais introduit de nouvelles surfaces d'attaque : event injection via les triggers, sur-privilégiation IAM des rôles d'exécution, vulnérabilités de supply chain dans les dépendances, et complexité accrue de l'observabilité sécurité. La sécurité du serverless requiert une expertise spécifique différente mais équivalente en termes d'effort à la sécurité des VMs traditionnelles.

Comment détecter qu'une fonction Lambda a été compromise ?

Les indicateurs de compromission d'une fonction Lambda incluent : appels API AWS inhabituels dans CloudTrail (création d'utilisateurs IAM, export de données S3 vers des destinations inconnues), alertes GuardDuty Lambda Protection (communications C2, crypto-mining), augmentation anormale de la durée d'exécution sans changement de code, consommation mémoire inhabituellement élevée, et erreurs de permissions inhabituelle dans les logs CloudWatch (la fonction essaie d'accéder à des ressources auxquelles elle n'a pas accès normalement, indiquant que le code exécuté n'est pas le code légitime).

Les fonctions Lambda sont-elles conformes au RGPD pour les données personnelles ?

Oui, avec une configuration appropriée. AWS Lambda dans les régions EU-West-1 (Irlande) et EU-Central-1 (Francfort) respecte les exigences de résidence des données RGPD. Il est important de vérifier que les logs CloudWatch, les traces X-Ray, et les événements CloudTrail associés à la fonction ne contiennent pas de données personnelles non anonymisées — une erreur

courante où des données utilisateurs se retrouvent dans les logs d'erreurs des fonctions. AWS est signataire du DPA (Data Processing Addendum) conforme RGPD et certifié ISO 27001.

Comment pentester une architecture AWS Lambda ?

Un pentest d'architecture serverless AWS commence par l'énumération des fonctions Lambda accessibles via l'API ou exposées via des URL publiques (Lambda Function URLs), puis l'analyse des IAM roles d'exécution pour identifier les privilege escalation paths (IAM enumeration avec Pacu ou ScoutSuite), le fuzzing des événements d'entrée pour tester l'event injection, et les tentatives d'SSRF vers les métadonnées IMDSv1 depuis la Lambda pour voler les credentials temporaires du rôle d'exécution (limité sur Lambda mais à vérifier). L'outil Pacu (open-source AWS exploitation framework) dispose de modules spécifiques aux attaques serverless Lambda.

Quels frameworks IaC sécurisent automatiquement les déploiements serverless ?

Le framework **AWS SAM (Serverless Application Model)** avec le plugin SAM Policy Templates facilite la création de rôles IAM least-privilege pour Lambda. **Serverless Framework** (open-source, multi-cloud) avec le plugin serverless-iam-roles-per-function génère automatiquement un rôle IAM dédié par fonction. Pour Terraform, les modules Terraform AWS Lambda maintenus par la communauté incluent des bonnes pratiques IAM préconfigurées. Ces outils IaC réduisent le risque de misconfigurations IAM serverless sans nécessiter une expertise IAM approfondie de chaque développeur.

Les SLA de disponibilité serverless sont-ils compatibles avec NIS 2 ?

AWS Lambda garantit un SLA de 99,95% de disponibilité mensuelle (inclus dans les SLA AWS régionaux). Pour les entités essentielles NIS 2 avec des exigences de disponibilité très élevées, une architecture multi-région Active-Active avec failover automatique est recommandée. L'utilisation de plusieurs Availability Zones (la résilience par défaut de Lambda dans une région AWS) répond aux exigences NIS 2 de résilience des systèmes d'information pour la plupart des

organisations, mais les OIV avec des infrastructures critiques doivent évaluer les exigences de continuité spécifiques avec l'ANSSI.

Accompagnement sécurité serverless : audit et DevSecOps

La sécurisation des architectures serverless requiert une expertise à la croisée du développement applicatif, de la sécurité cloud (IAM, réseaux), et des opérations SOC — des compétences rares combinées dans une même équipe. Notre service de RSSI externalisé propose des audits de sécurité dédiés aux architectures serverless : revue des rôles IAM d'exécution, configuration de l'observabilité sécurité, mise en place des pipelines SAST/SCA, et formation des équipes DevOps aux bonnes pratiques de sécurité serverless AWS, Azure, et GCP.

Démarrez par notre diagnostic NIS 2 pour évaluer si vos architectures serverless répondent aux exigences de sécurité NIS 2 (contrôle d'accès, journalisation, résilience). Notre service de pentest inclut des tests dédiés aux architectures serverless (event injection, IAM privilege escalation, supply chain) pour identifier les vulnérabilités avant qu'elles ne soient exploitées par des attaquants réels sur votre infrastructure de production.