AYI NEDJIMI Consultants

Sécurité Serverless : Lambda Functions et Protection

Catégorie : Cloud Security Lecture : 8 min Publié le : 06/03/2026 Auteur : Ayi NEDJIMI

Analyse complète de la sécurité des architectures serverless : vulnérabilités Lambda, permissions IAM, injection d'événements et stratégies de.

Résumé exécutif

Les architectures serverless éliminent la gestion des serveurs mais introduisent de nouveaux vecteurs d'attaque. Cette analyse couvre les vulnérabilités spécifiques aux Lambda functions, les stratégies de protection et les outils de sécurité adaptés au serverless.

Le serverless a transformé la manière dont nous déployons du code en production. Plus de serveurs à patcher, plus d'OS à durcir, plus de capacity planning à gérer. Pourtant, cette simplification opérationnelle masque une complexité sécuritaire que beaucoup d'équipes sous-estiment gravement. Les fonctions Lambda AWS, Azure Functions et Google Cloud Functions héritent d'un modèle de responsabilité partagée différent du compute traditionnel, avec des surfaces d'attaque spécifiques qui échappent aux outils de sécurité classiques. Après avoir conduit des audits de sécurité sur des dizaines d'architectures serverless en production et découvert des vulnérabilités critiques dans des applications traitant des millions de transactions par jour, je partage dans cette analyse les vecteurs d'attaque réels, les configurations défensives éprouvées et les outils spécialisés qui permettent de sécuriser vos workloads serverless sans sacrifier l'agilité qui fait tout l'intérêt de ce paradigme.

Pourquoi le modèle de menaces serverless est différent ?

Dans une architecture traditionnelle, l'attaquant compromet un serveur puis pivote. En serverless, il n'y a pas de serveur persistant à compromettre. Les vecteurs d'attaque se déplacent vers les **événements d'entrée** (injections via API Gateway, S3, SQS, DynamoDB Streams), les **permissions IAM** excessives de la fonction, les **dépendances vulnérables** packagées dans le déploiement, et les **données sensibles** stockées dans les variables d'environnement ou les layers. Le modèle STRIDE appliqué au serverless révèle des menaces spécifiques : *event injection* (manipulation des données d'événement pour exploiter la logique de la fonction), *function poisoning* (modification du code ou des layers), et **permission escalation** (exploitation des rôles IAM trop permissifs).

Pour comprendre les risques d'escalade de privilèges IAM dans AWS, notre article sur escalades de privilèges AWS détaille les vecteurs exploitables depuis une Lambda function compromise.

Vecteur d'attaque	Exemple	Impact	Mitigation
Event Injection	SQLi via S3 object key	Data breach	Validation d'entrée
IAM Overprovisioning	Lambda avec s3:*	Data exfiltration	Least privilege
Dependency Confusion	Package malveillant npm	Code execution	Lock files + audit
Env Variable Leaking	Secrets en clair	Credential theft	Secrets Manager
Cold Start Manipulation	Timing attacks	Information disclosure	Provisioned concurrency
Layer Poisoning	Layer partagée compromise	Supply chain attack	Layers privées signées

Mon avis : Le serverless n'est pas plus ou moins sécurisé que le compute traditionnel — il est différemment sécurisé. Le transfert de responsabilité vers le provider pour l'OS et le runtime est un gain net, mais la multiplication des fonctions et des triggers crée une surface d'attaque fragmentée que les outils classiques peinent à cartographier. Investissez dans des outils serverless-natifs.

Comment sécuriser les permissions IAM des Lambda ?

Chaque fonction Lambda s'exécute avec un **rôle IAM d'exécution** qui définit ses permissions sur les services AWS. Le problème systémique est le sur-provisionnement : par facilité, les développeurs attachent des policies managées larges comme `AmazonS3FullAccess` ou `AmazonDynamoDBFullAccess` au lieu de créer des policies custom least-privilege. En pentest, une Lambda avec `s3:*` sur `*` nous permet de lister tous les buckets du compte, exfiltrer des données sensibles, et souvent de pivoter vers d'autres services via les permissions transitives.

La bonne pratique consiste à créer une **policy IAM dédiée par fonction**, limitée aux actions, ressources et conditions strictement nécessaires. Utilisez les *conditions IAM* pour restreindre davantage : `aws:SourceVpc` pour limiter les appels depuis un VPC spécifique, `aws:CalledVia` pour imposer l'utilisation d'un service intermédiaire, et `s3:prefix` pour limiter l'accès à un préfixe S3 spécifique. L'outil **IAM Access Analyzer** peut générer automatiquement des policies least-privilege basées sur les logs CloudTrail de la fonction. Les détails sont documentés sur AWS Security.

Notre article sur les escalade de privilèges IAM cloud illustre comment une permission IAM excessive sur une fonction Lambda peut conduire à un accès complet à l'infrastructure AWS du compte.

Quelles sont les meilleures pratiques de codage serverless sécurisé ?

Le code serverless doit appliquer les principes de développement sécurisé avec des adaptations spécifiques. **Validation d'entrée** : chaque événement reçu par la fonction doit être validé strictement. Un objet uploadé sur S3 avec un nom contenant des caractères spéciaux peut exploiter une injection si le nom est utilisé dans une requête SQL ou une commande système.

Gestion des secrets : ne stockez jamais de credentials dans les variables d'environnement en clair. Utilisez AWS Secrets Manager ou SSM Parameter Store avec chiffrement KMS et récupérez les secrets au runtime via le SDK. Activez le caching pour éviter les appels API à chaque invocation.

Gestion des dépendances : verrouillez les versions dans package-lock.json (Node.js) ou requirements.txt avec hashes (Python). Scannez les dépendances avec **Snyk**, **npm audit** ou **safety** dans votre pipeline CI/CD. **Timeout et concurrency** : configurez des timeouts courts (30 secondes par défaut au lieu des 15 minutes max) et des reserved concurrency pour limiter l'impact d'un abus. Consultez nos recommandations sur les pratiques CI/CD sécurisées dans l'article sur attaques CI/CD GitOps.

Lors d'un audit serverless pour une plateforme de paiement, nous avons découvert une Lambda de traitement de factures PDF qui utilisait la bibliothèque Ghostscript pour la conversion. Le nom du fichier S3 uploadé par le client était directement concaténé dans la commande Ghostscript sans sanitisation — une injection de commande classique exploitable en uploadant un fichier dont le nom contenait des backticks. La correction a pris 15 minutes (échapper le nom de fichier), mais la vulnérabilité était en production depuis 8 mois. Aucun WAF ni outil de sécurité classique ne surveillait ce vecteur d'entrée.

La gestion des couches Lambda (Layers) représente un vecteur de risque supply chain souvent négligé. Les Layers permettent de partager du code commun entre plusieurs fonctions, mais une Layer compromise affecte toutes les fonctions qui l'utilisent. Utilisez uniquement des Layers provenant de sources approuvées : vos propres builds ou les Layers officielles AWS. Ne référencez jamais des Layers publiques non vérifiées par leur ARN, car le propriétaire peut modifier le contenu sans que vous en soyez informé. Pour vos propres Layers, intégrez-les dans votre pipeline CI/CD avec les mêmes contrôles de sécurité que le code applicatif : scan de dépendances, vérification d'intégrité, et versioning strict. Implémentez une politique qui interdit l'utilisation de Layers non listées dans un registre interne approuvé, vérifiable via un Lambda Authorizer ou une policy Organization-level.

Comment monitorer la sécurité des architectures serverless ?

Le monitoring serverless nécessite des outils adaptés au caractère éphémère des exécutions. **AWS CloudTrail** avec les Data Events Lambda capture chaque invocation et ses métadonnées. **CloudWatch Logs Insights** permet de rechercher dans les logs des fonctions des patterns suspects : erreurs d'authentification répétées, durées d'exécution anormalement longues, payload sizes inhabituelles. Activez **X-Ray tracing** pour visualiser les appels inter-services et détecter les anomalies de flux.

Pour la détection de menaces spécifiquement, **GuardDuty Lambda Protection** (lancé en 2023) analyse le trafic réseau des fonctions Lambda pour détecter les communications avec des IP malveillantes, le cryptomining et l'exfiltration DNS. Les outils serverless-natifs comme **Dashbird** et **Lumigo** offrent une observabilité adaptée avec des alertes sur les comportements anormaux. L'ANSSI recommande des approches complémentaires pour la surveillance des workloads cloud en France.

L'audit de sécurité de vos configurations serverless via Infrastructure as Code est couvert dans notre guide sur secrets sprawl et collecte. De plus, la gestion des secrets évoquée dans audit Terraform compliance est critique pour les Lambda qui accèdent à des services externes.

Faut-il déployer les Lambda dans un VPC ?

Le déploiement des fonctions Lambda dans un **VPC** ajoute une couche de sécurité réseau mais comporte des compromis. Les avantages incluent : accès aux ressources privées (RDS, ElastiCache, services internes), isolation réseau via Security Groups et NACLs, et contrôle du trafic sortant via un NAT Gateway. Les inconvénients incluent : cold starts plus longs (bien que mitigés par les Hyperplane ENI depuis 2019), complexité de configuration réseau, et nécessité de gérer les IP NAT pour les appels vers Internet.

La recommandation est de déployer dans un VPC uniquement les fonctions qui accèdent à des ressources privées. Pour les fonctions qui n'ont besoin que d'appels API publics (S3, DynamoDB, SQS), le déploiement hors VPC est préférable. Utilisez les **VPC Endpoints** (Interface et Gateway) pour permettre aux fonctions VPC d'accéder aux services AWS sans passer par Internet, réduisant la surface d'attaque réseau et les coûts de transfert de données.

À retenir : La sécurité serverless repose sur trois axes complémentaires : des permissions IAM least-privilege par fonction, une validation stricte de tous les événements d'entrée, et un monitoring adapté au caractère éphémère des exécutions. Les outils de sécurité traditionnels (EDR, HIDS) sont inopérants en serverless — adoptez des solutions cloud-natives comme GuardDuty Lambda Protection et les scanners de dépendances intégrés au CI/CD.

Peut-on faire du pentest sur des architectures serverless ?

Le pentest serverless est une discipline émergente qui nécessite des outils et méthodologies spécifiques. L'outil **ServerlessGoat** de l'OWASP fournit une application Lambda intentionnellement vulnérable pour l'entraînement. **SLS-Dev-Tools** permet d'interagir avec les fonctions Lambda pour l'injection d'événements. **Pacu** (le framework de pentest AWS) inclut des modules spécifiques pour l'énumération et l'exploitation des fonctions Lambda : découverte des triggers, extraction du code source, modification des variables d'environnement, et invocation avec des payloads malveillants. Le pentest serverless se concentre moins sur l'exploitation de vulnérabilités système et plus sur la logique applicative, les permissions IAM et les interactions entre services.

L'utilisation de **Provisioned Concurrency** pour les fonctions Lambda critiques du point de vue sécurité élimine le cold start et fournit un environnement d'exécution préchauffé et prévisible. Du point de vue sécurité, cela réduit la surface d'attaque liée aux timing attacks qui exploitent la différence de comportement entre un cold start et un warm start pour déduire des informations sur l'environnement d'exécution. De plus, les extensions Lambda de sécurité comme les agents de runtime protection nécessitent un temps d'initialisation qui peut dépasser le timeout sur un cold start, rendant la protection inefficace sur les premières invocations. Le Provisioned

Concurrency garantit que l'agent de sécurité est initialisé et opérationnel dès la première requête, assurant une couverture de protection complète et continue sans aucune fenêtre de vulnérabilité liée au démarrage à froid.

Votre dernière revue de sécurité a-t-elle inclus un audit spécifique de vos fonctions Lambda, ou les avez-vous considérées comme intrinsèquement sécurisées parce que le provider gère l'infrastructure sous-jacente ?

Comment sécuriser les événements sources Lambda ?

Chaque source d'événement Lambda présente des vecteurs d'attaque spécifiques qu'il faut protéger individuellement. Pour **API Gateway** : activez le WAF, configurez des throttling rules, validez les requêtes via des modèles JSON Schema avant l'invocation Lambda, et utilisez des authorizers (Cognito ou Lambda custom) pour l'authentification. Pour **S3 Events** : validez que les événements proviennent bien des buckets attendus via le champ sourceARN, et sanitisez les noms d'objets qui sont souvent utilisés directement dans le code. Pour **SQS/SNS** : configurez des politiques de ressource restrictives qui limitent les producers autorisés, et validez le schéma des messages avant traitement.

Pour les événements **DynamoDB Streams** et **Kinesis**, le risque principal est l'injection de données malveillantes dans le stream qui seront traitées par la Lambda en aval. Implémentez une validation stricte du schéma et du contenu pour chaque type d'événement. Les **EventBridge rules** doivent filtrer précisément les événements pertinents avec des patterns de matching détaillés plutôt que des wildcards larges. Documentez chaque trigger Lambda avec son modèle de menaces spécifique et les validations implémentées, créant ainsi une cartographie de sécurité de votre architecture event-driven qui facilite les audits et les revues de sécurité périodiques.

Sources et références : CISA · Cloud Security Alliance

Conclusion : sécuriser le serverless durablement

La sécurité des architectures serverless est un processus continu qui s'intègre dans le cycle de développement. Adoptez une approche shift-left : scannez les dépendances et les permissions IAM dans le pipeline CI/CD, validez les événements d'entrée systématiquement, et monitorez les comportements anormaux en production. Le serverless offre un niveau de sécurité infrastructure supérieur au compute traditionnel grâce au modèle de responsabilité partagée, mais cette sécurité ne couvre que les couches basses. La sécurité applicative et la gestion des permissions restent entièrement votre responsabilité et méritent l'attention que vous y consacriez.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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