AYI NEDJIMI Consultants

Detection-as-Code : Pipeline CI/CD pour Règles SIEM et

Catégorie : SOC et Detection Lecture : 11 min Publié le : 08/03/2026 Auteur : Ayi NEDJIMI

Guide complet Detection-as-Code : pipeline CI/CD pour règles SIEM, Git workflows, testing avec Atomic Red Team, compilation Sigma, intégration.

2.1 De l'Infrastructure-as-Code au Detection-as-Code

Le Detection-as-Code s'inscrit dans la lignée de l'Infrastructure-as-Code (IaC) et du Policy-as-Code. L'idée fondamentale est identique : **tout artefact opérationnel critique doit être versionné, testé et déployé de manière automatisée**. Là où Terraform gère l'infrastructure cloud et Open Policy Agent gère les politiques de sécurité, le DaC gère les règles de détection du SIEM. Guide complet Detection-as-Code : pipeline CI/CD pour règles SIEM, Git workflows, testing avec Atomic Red Team, compilation Sigma, intégration. La détection des menaces repose sur la capacité à identifier les comportements malveillants parmi le bruit. Detection-as-Code : Pipeline CI/CD pour Règles SIEM et fournit des méthodologies éprouvées pour les analystes SOC. Nous abordons notamment : 8. checklist d'implémentation detection-as-code, questions frequentes et 9. conclusion : la détection comme produit d'ingénierie. Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Les principes directeurs du DaC sont les suivants :

- **Single Source of Truth (SSoT)** : le dépôt Git est la source de vérité unique pour toutes les règles de détection. Toute modification passe par un commit, toute règle active en production existe dans le dépôt.
- **Revue par les pairs** : chaque nouvelle règle ou modification fait l'objet d'une Pull Request (PR) revue par au moins un autre detection engineer. La PR documente le contexte, la logique de détection, les tests effectués et les risques de faux positifs.
- **Tests automatisés** : avant le déploiement, chaque règle est validée syntaxiquement, testée contre des données de référence (true positives et true negatives), et vérifiée pour les régressions.
- **Déploiement automatisé** : le merge d'une PR dans la branche principale déclenche automatiquement le déploiement de la règle vers le SIEM de production, éliminant les erreurs manuelles et les dérives de configuration.
- **Traçabilité complète** : l'historique Git offre un audit trail complet de qui a modifié quelle règle, quand et pourquoi. C'est un atout considérable pour la conformité ISO 27001 et les audits réglementaires.

2.2 Anatomie d'une règle Detection-as-Code

Dans une approche DaC, chaque règle de détection est un fichier structuré (YAML, TOML ou JSON) qui contient non seulement la logique de détection, mais aussi l'ensemble des métadonnées nécessaires à son cycle de vie. Voici la structure type d'une règle DaC :

```yaml
# detection-rules/credential-access/mimikatz-lsass-access.yml
rule:
  id: "CR-2026-0142"
  name: "Mimikatz LSASS Memory Access Detection"
  description: |
    Détecte l'accès au processus LSASS par des outils de type Mimikatz
    via les event IDs Sysmon 10 (ProcessAccess) avec des droits
    PROCESS_VM_READ sur lsass.exe.

  # Métadonnées de gouvernance
  author: "a.nedjimi@ayinedjimi-consultants.fr"
  created: "2026-02-15"
  modified: "2026-02-28"
  version: "1.2.0"
  status: "production"          # draft | testing | production | deprecated
  severity: "high"
  confidence: "high"

  # Mapping MITRE ATT&CK
  mitre:
    tactic: "Credential Access"
    technique: "T1003.001"      # OS Credential Dumping: LSASS Memory
    subtechnique: "LSASS Memory"

  # Logique de détection (format Sigma)
  sigma:
    logsource:
      category: process_access
      product: windows
    detection:
      selection:
        TargetImage|endswith: '\lsass.exe'
        GrantedAccess|contains:
          - '0x1010'
          - '0x1410'
          - '0x1438'
          - '0x143a'
      filter_legitimate:
        SourceImage|endswith:
          - '\wmiprvse.exe'
          - '\taskmgr.exe'
          - '\MsMpEng.exe'
      condition: selection and not filter_legitimate

  # Métadonnées opérationnelles
  operations:
    sla_response: "15min"
    runbook: "runbooks/credential-access/lsass-dump.md"
    escalation: "tier2-incident-response"
    false_positive_rate: "low"
    known_fps:
      - "Antivirus legitimate scanning LSASS"
      - "Windows Defender periodic checks"

  # Tests associés
  tests:
    atomic_red_team: "T1003.001"
    unit_tests:
      - "tests/credential-access/test_mimikatz_lsass.yml"

  # Cibles de déploiement
  targets:
```

```
    - siem: "splunk"
      index: "windows_sysmon"
      sourcetype: "XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
    - siem: "elastic"
      index: "winlogbeat-*"
    - siem: "sentinel"
      table: "SecurityEvent"
```

Notre avis d'expert
La fatigue d'alerte est l'ennemi silencieux des SOC modernes. Quand les analystes traitent des centaines de faux positifs par jour, les vraies menaces passent inaperçues. La priorisation intelligente et l'automatisation des tâches de triage sont essentielles.

Quel est votre taux de faux positifs et quel impact a-t-il sur la vigilance de vos analystes ?

Cette structure offre plusieurs avantages décisifs. D'abord, la **portabilité** : la logique Sigma est indépendante du SIEM cible et peut être compilée automatiquement pour chaque plateforme. Ensuite, la **gouvernance** : les métadonnées de version, d'auteur et de statut permettent un suivi rigoureux du cycle de vie. Enfin, la **testabilité** : les références aux tests Atomic Red Team et aux tests unitaires permettent une validation automatisée dans le pipeline CI/CD.

2.3 Organisation du dépôt Git

L'organisation du dépôt de détection doit refléter la taxonomie MITRE ATT&CK pour faciliter l'analyse de couverture. Voici la structure recommandée :

```
detection-repository/
├── .github/
│   └── workflows/
│       ├── ci-lint-test.yml          # Pipeline CI (PR)
│       ├── cd-deploy-production.yml  # Pipeline CD (merge main)
│       └── coverage-report.yml       # Rapport couverture hebdo
├── rules/
│   ├── initial-access/
│   │   ├── phishing-attachment.yml
│   │   └── drive-by-compromise.yml
│   ├── execution/
│   │   ├── powershell-encoded-command.yml
│   │   └── mshta-execution.yml
│   ├── persistence/
│   │   ├── registry-run-keys.yml
│   │   └── scheduled-task-creation.yml
│   ├── privilege-escalation/
│   ├── defense-evasion/
│   ├── credential-access/
│   │   ├── mimikatz-lsass-access.yml
│   │   └── dcsync-detection.yml
│   ├── discovery/
│   ├── lateral-movement/
│   │   ├── psexec-execution.yml
│   │   └── wmi-lateral-movement.yml
│   ├── collection/
│   ├── exfiltration/
│   └── command-and-control/
│       ├── dns-tunneling.yml
│       └── cobalt-strike-beacon.yml
├── tests/
│   ├── unit/                         # Tests unitaires par règle
│   ├── integration/                  # Tests d'intégration SIEM
│   └── fixtures/                     # Données de test (logs)
│       ├── true-positives/
│       └── true-negatives/
├── backends/
│   ├── splunk/                       # Config spécifique Splunk
│   ├── elastic/                      # Config spécifique Elastic
│   └── sentinel/                     # Config spécifique Sentinel
├── runbooks/                         # Procédures de réponse
├── scripts/
│   ├── sigma-compile.py              # Compilation Sigma multi-SIEM
│   ├── deploy-splunk.py              # Déploiement Splunk API
│   ├── deploy-elastic.py             # Déploiement Elastic API
│   ├── coverage-matrix.py            # Génération matrice couverture
│   └── validate-rule.py              # Validation schéma YAML
├── sigma-config/                     # Configs pySigma (mappings)
├── docs/
│   ├── CONTRIBUTING.md               # Guide contribution
│   ├── STYLE_GUIDE.md                # Convention nommage
│   └── REVIEW_CHECKLIST.md           # Checklist revue PR
├── coverage/
│   └── mitre-coverage.json           # Matrice de couverture ATT&CK
└── README.md
```

Bonne pratique : branches et environnements

Utilisez un modèle à trois branches : `feature/*` pour le développement, `staging` pour les tests en environnement pré-production, et `main` pour la production. Chaque merge dans `staging` déclenche un déploiement en lab SOC pour validation, et chaque merge dans `main` déploie en

production. Ce modèle permet de tester les règles sur des données réelles (mais en mode alerte silencieuse) avant l'activation complète, un principe similaire au **Report-Only mode** des Conditional Access Policies dans Entra ID.

```
# tests/unit/credential-access/test_mimikatz_lsass.yml
test:
  rule: "rules/credential-access/mimikatz-lsass-access.yml"

  true_positives:
    - name: "Mimikatz sekurlsa::logonpasswords"
      log:
        EventID: 10
        SourceImage: 'C:\Users\attacker\mimikatz.exe'
        TargetImage: 'C:\Windows\System32\lsass.exe'
        GrantedAccess: '0x1010'
      expected: match

    - name: "procdump LSASS dump"
      log:
        EventID: 10
        SourceImage: 'C:\Tools\procdump64.exe'
        TargetImage: 'C:\Windows\System32\lsass.exe'
        GrantedAccess: '0x1438'
      expected: match

  true_negatives:
    - name: "Windows Defender scanning LSASS"
      log:
        EventID: 10
        SourceImage: 'C:\ProgramData\Microsoft\Windows Defender\Platform\MsMpEng.exe'
        TargetImage: 'C:\Windows\System32\lsass.exe'
        GrantedAccess: '0x1010'
      expected: no_match

    - name: "Task Manager viewing processes"
      log:
        EventID: 10
        SourceImage: 'C:\Windows\System32\taskmgr.exe'
        TargetImage: 'C:\Windows\System32\lsass.exe'
        GrantedAccess: '0x1410'
      expected: no_match
```

Étape 4 : Analyse de couverture et détection de doublons
La dernière étape CI met à jour la matrice de couverture MITRE ATT&CK et vérifie l'absence de règles dupliquées. Le script `coverage-matrix.py` génère un fichier JSON qui alimente un dashboard de couverture, permettant aux detection engineers de visualiser les tactiques et techniques couvertes et celles qui restent à traiter. Ce mapping est essentiel pour structurer un programme de détection aligné sur MITRE ATT&CK.

3.3 Phase CD : déploiement automatisé

Une fois la PR approuvée et mergée dans `main`, la phase CD prend le relais avec quatre étapes :

Compilation finale : les règles modifiées sont recompilées pour chaque backend cible. Le pipeline détecte automatiquement quelles règles ont changé (via `git diff`) et ne recompile que les fichiers modifiés pour optimiser le temps de build.

Déploiement via API : les requêtes compilées sont déployées vers chaque SIEM via leurs APIs respectives. Pour Splunk, le script utilise l'API REST pour créer ou mettre à jour les saved searches. Pour Elastic Security, l'API Detection Engine permet de pousser les règles au format TOML/JSON. Pour Microsoft Sentinel, les Analytics Rules sont déployées via l'API ARM ou l'Azure CLI.

Tests de performance : une règle de détection qui fonctionne parfaitement en lab peut devenir problématique en production si elle génère des requêtes trop coûteuses. Le pipeline doit mesurer le temps d'exécution des requêtes compilées et alerter si une règle dépasse un seuil configurable. Pour Splunk, cela signifie estimer le scan count et le temps de recherche via l'API Jobs. Pour Elastic, il s'agit de monitorer le temps de requête et la consommation mémoire des rules.

Type de test	Objectif	Fréquence	Automatisable
Lint / Schema	Syntaxe YAML et conformité schéma	Chaque PR	Oui (100%)
Compilation Sigma	Compilation multi-backend sans erreur	Chaque PR	Oui (100%)
Tests unitaires	True positive / True negative	Chaque PR	Oui (100%)
Tests Atomic RT	Validation en environnement réel	Merge staging	Oui (lab requis)
Tests régression	Non-régression des true positives	Chaque modification	Oui (100%)
Tests performance	Temps exécution requête	Déploiement prod	Oui (API SIEM)
Smoke tests	Règle active et fonctionnelle	Post-déploiement	Oui (100%)

4.3 Environnement de test : le lab SOC

Un pipeline DaC efficace nécessite un **environnement de lab dédié** qui reproduit la stack de production. Ce lab comprend au minimum : une VM Windows Server 2022 avec Active Directory, une VM Windows 11 workstation, Sysmon configuré avec une config modulaire (type SwiftOnSecurity ou Olaf Hartong), le forwarding d'événements vers une instance SIEM de staging, et les outils d'émulation d'attaque (Atomic Red Team, Caldera, MITRE ATT&CK Navigator).

L'infrastructure de lab peut être provisionnée via IaC (Terraform + Ansible) pour garantir la reproductibilité et permettre la reconstruction rapide après des tests destructifs. Certaines organisations utilisent des conteneurs Docker pour les composants SIEM (Elastic Stack en particulier) afin de réduire les coûts et accélérer le provisioning, une approche similaire aux environnements Kubernetes éphémères.

L'application du **versioning sémantique** (SemVer) aux règles de détection offre une traçabilité précieuse. Le principe : `MAJOR.MINOR.PATCH` où :

- **MAJOR** : changement fondamental de la logique de détection (nouvelle approche, refonte complète). Exemple : passage d'une détection par nom de processus à une détection par comportement.

- **MINOR** : ajout ou modification de fonctionnalité. Exemple : ajout d'un nouveau pattern de détection, extension des sources de logs couvertes.
- **PATCH** : correction ou tuning. Exemple : ajout d'un filtre pour un faux positif identifié, correction d'une typo dans un champ.

Le versioning permet de générer automatiquement un **changelog** des modifications de détection, un atout précieux pour les rapports de conformité et les revues trimestrielles du programme de détection. Il permet aussi de corréler l'évolution des métriques de qualité (taux de faux positifs, couverture) avec les changements spécifiques apportés aux règles.

6.3 Gestion du cycle de vie des règles

Chaque règle de détection traverse un cycle de vie en quatre étapes, matérialisé par le champ `status` dans le fichier YAML :

- **draft** : la règle est en cours de développement. Elle n'est pas déployée et ne génère aucune alerte.
- **testing** : la règle est déployée en mode silencieux (alerte loguée mais non notifiée). Le detection engineer surveille les résultats pendant 7 à 14 jours pour calibrer les filtres.
- **production** : la règle est active et génère des alertes traitées par les analystes SOC. Elle fait l'objet d'un monitoring continu des métriques.
- **deprecated** : la règle est désactivée, soit parce que la menace n'est plus pertinente, soit parce qu'une règle plus efficace la remplace. Le fichier reste dans le dépôt pour l'historique.

Un processus de revue trimestrielle doit évaluer chaque règle en production : taux de faux positifs, nombre de vrais positifs détectés, pertinence de la menace ciblée, et performance de la requête. Les règles à haut taux de faux positifs sans vrais positifs confirmés depuis plus de 6 mois sont candidates au statut deprecated. Cette hygiène est comparable à la revue régulière des GPO Active Directory : une règle non maintenue est une dette technique qui dégrade l'efficacité globale du SOC.

Le dashboard MITRE ATT&CK Navigator est le format de visualisation standard. Il peut être généré automatiquement au format JSON compatible ATT&CK Navigator à partir de la matrice de couverture, avec un code couleur indiquant le niveau de confiance : vert (high), orange (medium), rouge (low), gris (non couvert).

7.3 Métriques de qualité des règles individuelles

Au-delà des métriques agrégées, chaque règle doit être évaluée individuellement. Un script de monitoring post-déploiement collecte les statistiques suivantes sur une fenêtre glissante de 30 jours :

- **Volume d'alertes** : nombre d'alertes générées par jour/semaine. Un volume anormalement élevé indique un problème de faux positifs. Un volume à zéro pendant 30 jours peut indiquer une source de logs manquante.
- **Ratio VP/FP** : proportion de vrais positifs confirmés par rapport aux faux positifs. Objectif : > 80% de vrais positifs ou > 50% pour les règles de détection comportementale.

- **Temps de triage moyen** : temps passé par les analystes pour qualifier chaque alerte. Un temps élevé suggère que la règle manque de contexte ou de documentation.
- **Performance de la requête** : temps d'exécution moyen et peak. Les requêtes dépassant 30 secondes en peak doivent être optimisées.

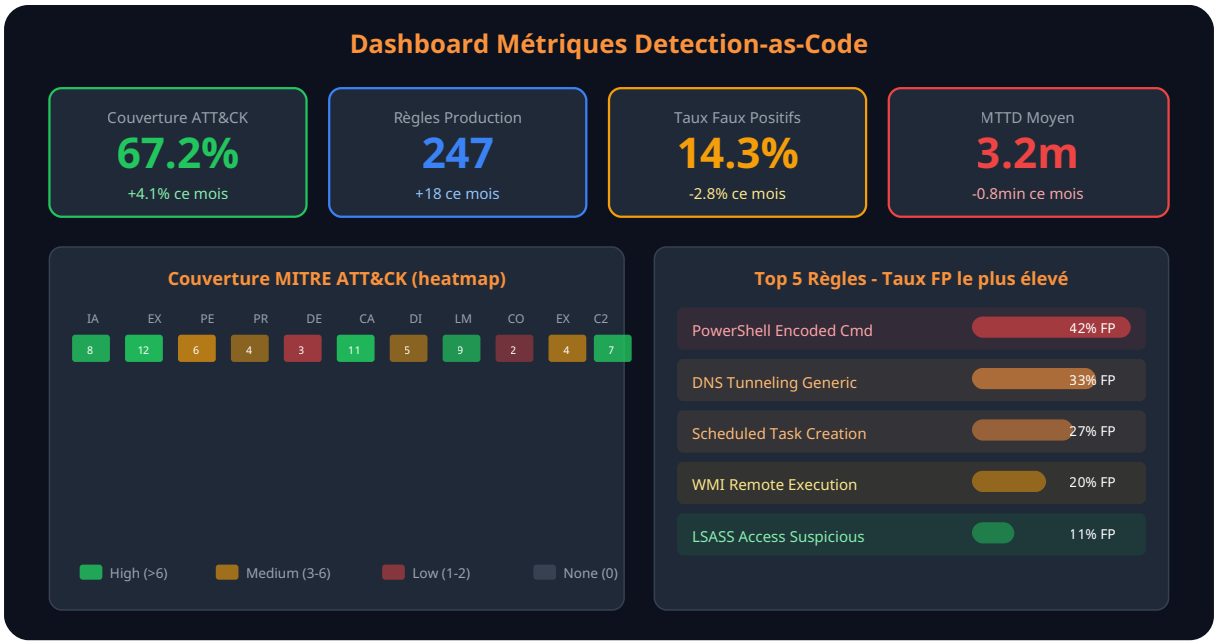

Figure 3 -- Dashboard métriques DaC : KPIs, heatmap couverture ATT&CK et analyse des faux positifs

8. Checklist d'implémentation Detection-as-Code

L'adoption du Detection-as-Code est un **programme progressif**, pas un big bang. Voici une checklist en 20 points organisée en trois phases pour guider votre implémentation :

Phase 1 : Fondations (Semaines 1-4)

- **Créer le dépôt Git** avec la structure de répertoires alignée sur MITRE ATT&CK
- **Définir le schéma YAML** des règles avec les champs obligatoires (id, name, mitre, sigma, status, tests)
- **Documenter les conventions** de nommage, le guide de contribution et la checklist de revue
- **Importer les règles existantes** depuis le SIEM vers le dépôt Git (migration initiale)
- **Configurer les pipelines pySigma** pour chaque backend SIEM utilisé
- **Mettre en place le pipeline CI** avec lint, validation de schéma et compilation Sigma

Phase 2 : Automatisation (Semaines 5-8)

- **Implémenter le déploiement automatisé** via les APIs SIEM (Splunk REST, Elastic Detection, Sentinel ARM)
- **Configurer les smoke tests** post-déploiement et le mécanisme de rollback automatique
- **Mettre en place le lab SOC** avec Atomic Red Team pour les tests de validation
- **Créer les premiers tests unitaires** pour les 20 règles les plus critiques
- **Implémenter les notifications** (Slack/Teams) pour chaque déploiement et rollback
- **Configurer le Git workflow** avec le template PR et le processus de revue

Phase 3 : Maturité (Semaines 9-12+)

- **Générer la matrice de couverture** MITRE ATT&CK automatiquement et le dashboard associé
- **Implémenter le monitoring des métriques** (taux FP, MTTD, vélocité, performance)
- **Établir le processus de revue trimestrielle** des règles (deprecated, tuning, nouvelles menaces)
- **Intégrer les feeds Threat Intel** pour la création automatique de stubs de règles
- **Former les analystes SOC** au workflow Git et à la contribution de règles
- **Documenter les runbooks** associés à chaque règle de détection
- **Créer le rapport mensuel** de performance du programme de détection pour le RSSI
- **Partager** les règles pertinentes avec la communauté SigmaHQ pour contribuer à l'écosystème

Conseil : commencez petit, itérez vite
L'erreur la plus fréquente est de vouloir migrer 100% des règles dès le départ. Commencez par **10 règles critiques**, mettez en place le pipeline complet pour ces 10 règles, puis élargissez progressivement. Une approche par paliers de 20 règles par sprint permet de maintenir la qualité tout en accélérant l'adoption. Rappelez-vous : le DaC est un **changement culturel** autant qu'un changement technique. Les analystes SOC doivent être accompagnés dans l'apprentissage de Git, et les detection engineers dans l'adoption d'une rigueur d'ingénierie logicielle.

Pour approfondir ce sujet, consultez notre outil open-source threat-hunting-queries qui facilite le threat hunting proactif.

Questions frequentes

Comment déployer Detection dans un environnement de production ?

La mise en œuvre de Detection en production necessite une planification rigoureuse, incluant l'evaluation des prerequis techniques, la definition d'une architecture cible, des tests de validation approfondis et un plan de deploiement progressif avec des points de controle a chaque etape.

Pourquoi Detection est-il essentiel pour la securite des systemes d'information ?

Detection constitue un element fondamental de la securite des systemes d'information car il permet de reduire significativement la surface d'attaque, d'ameliorer la detection des menaces et de renforcer la posture globale de securite de l'organisation face aux cybermenaces actuelles.

Combien de règles de détection faut-il pour démarrer avec Detection-as-Code : Pipeline CI/CD pour Règles SIEM ?

Commencez par 20 à 30 règles alignées sur les techniques MITRE ATT&CK les plus courantes. Mieux vaut peu de règles bien calibrées que des centaines qui génèrent du bruit.

Sources et références : MITRE ATT&CK · MITRE CAR

> Points clés à retenir
>
> - 8. Checklist d'implémentation Detection-as-Code
> - Questions frequentes
> - 9. Conclusion : la détection comme produit d'ingénierie

9. Conclusion : la détection comme produit d'ingénierie

Le Detection-as-Code représente un changement de schéma fondamental pour les SOC : la détection n'est plus un artisanat ponctuel mais un **produit d'ingénierie** avec son cycle de développement, ses tests, ses métriques et son amélioration continue. Ce changement est inévitable pour plusieurs raisons : la croissance exponentielle du volume de données et du nombre de techniques d'attaque rend l'approche manuelle insoutenable, les exigences de conformité (ISO 27001, SOC 2, NIS2) imposent une traçabilité que seul le versioning peut fournir, et la pénurie de talents en cybersécurité oblige à automatiser tout ce qui peut l'être.

Les organisations qui adoptent le DaC constatent des améliorations mesurables : réduction du taux de faux positifs de 30 à 50%, augmentation de la couverture MITRE ATT&CK de 20 à 40 points de pourcentage en un an, et division par deux du temps de déploiement d'une nouvelle règle (de jours à heures). Mais au-delà des métriques, le bénéfice le plus profond est culturel : les detection engineers développent un sentiment de propriété sur leur code, les analystes SOC gagnent en confiance dans les alertes qu'ils traitent, et le RSSI dispose d'une visibilité objective sur la posture de détection de l'organisation.

Le chemin vers un programme DaC mature est progressif -- il ne se parcourt pas en un sprint. Mais chaque étape apporte une valeur immédiate. Commencez par versionner vos règles dans Git, ajoutez le linting automatique, puis la compilation Sigma, puis les tests, puis le déploiement automatisé. Chaque couche renforce la précédente. Et n'oubliez pas : le meilleur pipeline DaC est celui qui est effectivement utilisé par votre équipe, pas celui qui est le plus abouti sur le papier.

Articles connexes

SOC & Détection
Sigma Rules : Guide Complet d'Écriture et Déploiement
Format YAML, modifiers, corrélation, pySigma backends
Framework
MITRE ATT&CK : Guide Pratique Red & Blue Team
Tactiques, techniques, couverture de détection
Attaques Credentials
Password Attacks : Cracking, Spraying, Credential Stuffing
Techniques d'attaque et règles de détection associées
Identité & Cloud
Sécuriser Entra ID : Conditional Access et MFA
Politiques de sécurité identitaire et monitoring

Cloud Security
Kubernetes Security : Hardening et Audit
Infrastructure as Code et sécurité des clusters
Conformité
ISO 27001 : Guide Complet de Certification
Traçabilité et gouvernance des contrôles de sécurité

Références et ressources externes

- SigmaHQ -- Sigma Rules Repository -- Dépôt officiel des règles Sigma communautaires
- pySigma -- Sigma Rule Processing Framework -- Framework de compilation multi-SIEM
- Atomic Red Team -- Red Canary -- Framework de tests de détection par technique ATT&CK
- MITRE ATT&CK Framework -- Base de connaissances des tactiques et techniques adverses
- Elastic Security Detection Engine -- Documentation Elastic Detection Rules API

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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