Attaques SAML : XML Signature Wrapping, Replay et défenses 2026

Attaques SAML 2.0 : XSW (8 variantes), replay, signature stripping. PoC, SAMLRaider et checklist défenses pour sécuriser votre SSO d'entreprise en 2026.

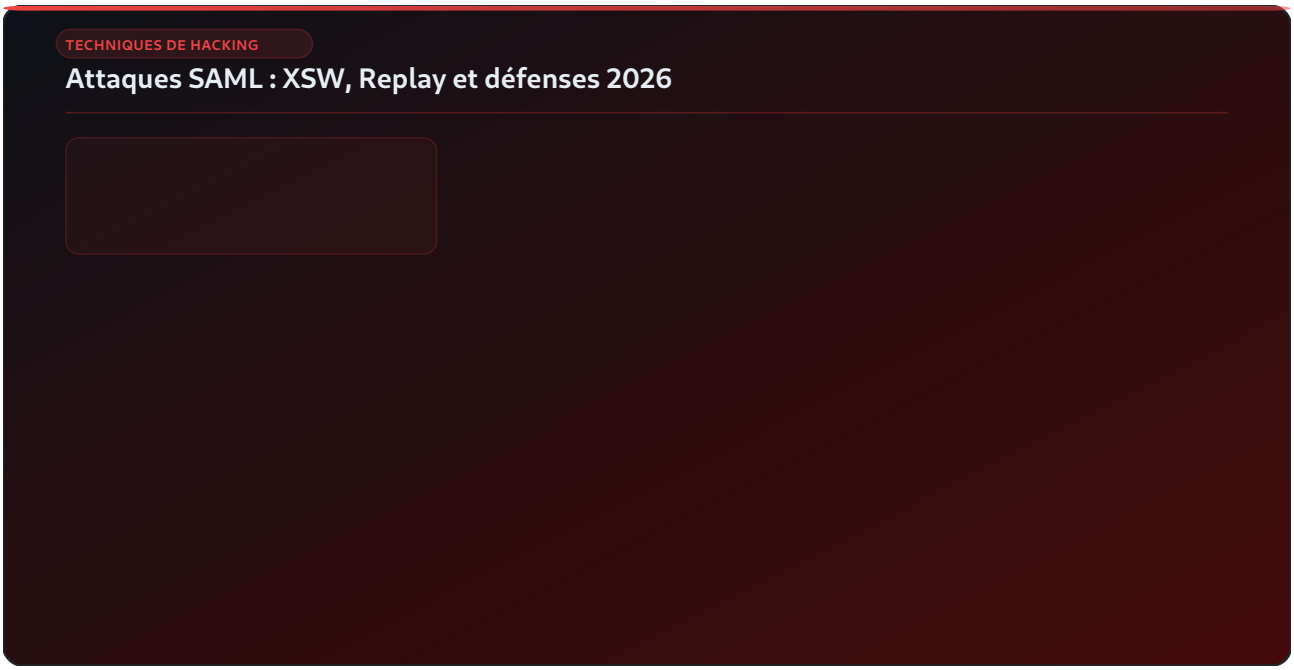
Les attaques SAML représentent aujourd'hui l'une des menaces les plus sophistiquées pesant sur les infrastructures d'authentification fédérée des grandes organisations. Security Assertion Markup Language, ou SAML 2.0, est omniprésent dans les environnements d'entreprise : il orchestre la confiance entre fournisseurs d'identité (IdP) tels qu'Okta, Microsoft ADFS, PingFederate et des centaines d'applications tierces jouant le rôle de fournisseurs de services (SP). Pourtant, sa complexité même — des assertions XML signées transitant via le navigateur de l'utilisateur — crée une surface d'attaque unique que les attaquants exploitent activement. En 2025-2026, des campagnes APT ont tiré parti de vulnérabilités XML Signature Wrapping pour compromettre des SSO d'administrations européennes, démontrant que la maturité du protocole ne garantit pas la sécurité de son implémentation. Ce guide technique approfondi couvre les huit variantes d'attaque XSW, les techniques de rejeu d'assertions, la suppression de signature, les mauvaises configurations critiques, et les défenses indispensables pour les équipes sécurité, pentesters et architectes IAM travaillant sur des environnements SAML 2.0 en production. La maîtrise de ces techniques est indispensable pour tout professionnel chargé de tester ou de sécuriser des infrastructures SSO d'entreprise face aux menaces actuelles.

À RETENIR

À retenir :

XML Signature Wrapping (XSW) permet de forger des assertions SAML valides en déplaçant l'élément signé dans le document XML sans invalider la signature cryptographique.

> Un attaquant peut rejouer une assertion SAML interceptée pendant la fenêtre de validité (souvent 5 à 30 minutes) si le SP ne vérifie pas les IDs d'assertion déjà consommés.
>
> La suppression de signature (Signature Stripping) exploite les SP mal configurés qui acceptent des assertions non signées lorsque la signature est absente ou retirée.
>
> Les outils SAMLRaider (extension Burp Suite) et SAML Toolkit automatisent la majorité des tests offensifs sur les flux SAML en quelques minutes seulement.



Avant d'aborder les techniques offensives, il est indispensable de maîtriser l'architecture SAML 2.0 dans ses détails. Le protocole repose sur trois acteurs : le **Principal** (l'utilisateur), l'**Identity Provider** (IdP) qui authentifie et émet des assertions, et le **Service Provider** (SP) qui consomme ces assertions pour autoriser l'accès.

Une assertion SAML est un document XML contenant des déclarations d'authentification (`AuthnStatement`), d'attributs (`AttributeStatement`) et d'autorisation (`AuthzDecisionStatement`).

Elle est signée numériquement par l'IdP à l'aide d'une clé privée RSA ou DSA, et le SP vérifie cette signature avec le certificat public de l'IdP. Le standard SAML 2.0 est défini par OASIS et sa spécification complète est disponible sur docs.oasis-open.org.

Binding HTTP-POST et HTTP-Redirect

SAML 2.0 définit plusieurs bindings pour transmettre les messages. Les deux plus courants sont le HTTP-POST Binding et le HTTP-Redirect Binding. Le **HTTP-POST Binding** encode l'assertion en Base64 et l'insère dans un formulaire HTML auto-soumis via le navigateur. Ce binding est utilisé pour les réponses d'authentification car les assertions signées peuvent être volumineuses. Le **HTTP-Redirect Binding** compresse le message (deflate), l'encode en Base64, puis le transmet comme paramètre URL. Utilisé principalement pour les AuthnRequest du SP vers l'IdP, la taille limitée des URLs impose cette compression.

```
 # Decodage d'un SAMLRequest en redirect binding
echo "fZLNTsMwEIRfeRXJd..." | base64 -d | python3 -c "
import sys, zlib
data = sys.stdin.buffer.read()
print(zlib.decompress(data, -15).decode())
"
```

Structure d'une SAMLResponse signée

La structure d'une réponse SAML signée inclut une `Signature` XML-DSig qui peut être placée soit au niveau de la `Response` (enveloppe), soit au niveau de l'`Assertion` (elément enfant), soit aux deux niveaux simultanément. Cette flexibilité est précisément la source des vulnérabilités XSW. La référence `URI="#_assertion789"` dans l'élément `Reference` indique quel noeud est protégé par la signature cryptographique.

```
<samlp:Response ID="_response123" InResponseTo="_req456">
  <ds:Signature>
    <ds:SignedInfo>
      <ds:Reference URI="#_assertion789"/>
    </ds:SignedInfo>
    <ds:SignatureValue>...</ds:SignatureValue>
  </ds:Signature>
  <saml:Assertion ID="_assertion789">
    <saml:Subject>
      <saml:NameID>admin@corp.com</saml:NameID>
    </saml:Subject>
    <saml:Conditions NotOnOrAfter="2026-07-05T10:15:00Z"
                     NotBefore="2026-07-05T10:00:00Z">
      <saml:AudienceRestriction>
        <saml:Audience>https://sp.example.com</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
  </saml:Assertion>
</samlp:Response>
```

XML Signature Wrapping (XSW) : principe et 8 variantes

L'attaque XML Signature Wrapping exploite une propriété fondamentale de la spécification XML-DSig : la signature ne protège que le noeud référencé par son attribut `ID`, pas la position de ce noeud dans le document. Un attaquant peut donc déplacer l'assertion originale (signée et valide) dans une partie du document que le SP ne lit pas pour la validation, et injecter une assertion malveillante à l'emplacement où le SP s'attend à la trouver.

⚡ **Retour terrain**

Dans mes missions de red team, la phase de post-exploitation révèle systématiquement des données que le client pensait protégées. Le cas le plus fréquent : des fichiers Excel de

comptabilité ou RH stockés sur des partages réseau accessibles à tous les utilisateurs du domaine, sans restriction. Sur les 30 dernières missions, 27 avaient des partages réseau avec des données sensibles accessibles à n'importe quel utilisateur authentifié. La sensibilité des données n'est pas corrélée à leur niveau de protection réel.

Mécanisme fondamental de l'XSW

La bibliothèque de validation suit la référence `URI="#_assertion789"` pour trouver et valider le noeud signé. Le parseur applicatif, lui, prend souvent le premier noeud `Assertion` rencontré selon un chemin XPath prédéfini. Si ces deux comportements divergent, l'attaque réussit.

```
 <!-- AVANT XSW : assertion legitime signee -->
<samlp:Response>
  <saml:Assertion ID="_assertion789"> <!-- signee -->
    <saml:NameID>user@corp.com</saml:NameID>
  </saml:Assertion>
</samlp:Response>

<!-- APRES XSW : assertion malveillante injectee -->
<samlp:Response>
  <saml:Assertion ID="_evil">  <!-- MALVEILLANTE - lue par le SP -->
    <saml:NameID>admin@corp.com</saml:NameID>
  </saml:Assertion>
  <saml:Assertion ID="_assertion789"> <!-- originale signee - validee -->
    <saml:NameID>user@corp.com</saml:NameID>
  </saml:Assertion>
</samlp:Response>
```

Les 8 variantes XSW documentées

XSW1 : Signature sur la Response — l'assertion malveillante est insérée avant l'assertion signée au niveau racine de la Response. Le SP valide la signature de la Response mais traite la première assertion rencontrée.

XSW2 : Signature sur la Response — l'assertion malveillante est insérée après l'assertion signée. Exploite les parseurs qui prennent le dernier noeud plutôt que le premier.

XSW3 : Signature sur l'Assertion — l'assertion malveillante est insérée comme frère de l'assertion signée au niveau Response. La signature valide l'assertion originale mais le SP traite l'autre.

XSW4 : Signature sur l'Assertion — l'assertion originale (signée) est déplacée à l'intérieur de l'assertion malveillante comme enfant `Extensions`. Le SP valide la signature de l'enfant mais traite le parent.

```
<!-- XSW4 : assertion originale nichee dans l'assertion malveillante -->
<samlp:Response>
  <saml:Assertion ID="_evil">
    <saml:NameID>admin@corp.com</saml:NameID>
    <saml:Extensions>
      <saml:Assertion ID="_assertion789">
        <saml:NameID>user@corp.com</saml:NameID>
        <ds:Signature>...signature valide...</ds:Signature>
      </saml:Assertion>
    </saml:Extensions>
  </saml:Assertion>
</samlp:Response>
```

XSW5 : La référence de signature pointe vers un noeud qui n'est pas l'assertion. Un noeud `Object` XML-DSig contient une copie de l'assertion originale, tandis que l'assertion malveillante prend la place principale.

XSW6 : Combinaison où la signature est sur l'assertion mais déplacée dans un noeud frère avec le même ID. Exploite les implémentations qui ne vérifient pas l'unicité des IDs.

XSW7 : Utilisation des namespaces XML — l'assertion malveillante utilise un préfixe de namespace différent pour contourner les validations basées sur le nom local du noeud.

XSW8 : Exploitation des `xs:anyType` dans le schéma SAML — insertion de l'assertion originale dans des éléments `Object` ou `Advice` tolérés par le schéma mais ignorés par la logique applicative. La ressource de référence sur la sécurisation SAML est le OWASP SAML Security Cheat Sheet.

SAML Replay Attack : exploitation des fenêtres temporelles

Une attaque par rejeu SAML consiste à capturer une assertion SAML valide (par interception réseau, vol de cookie de session, logs d'accès, ou XSS) et à la soumettre à nouveau au SP pendant sa fenêtre de validité. Cette fenêtre est définie par les attributs `NotBefore` et `NotOnOrAfter` de l'élément `Conditions`.

Conditions d'exploitation du SAML Replay

Pour qu'un replay réussisse, trois conditions doivent être réunies : la fenêtre de validité n'est pas expirée (souvent configurée à 5-30 minutes), le SP ne maintient pas de registre des IDs d'assertions déjà consommés (absence de cache anti-rejeu), et l'attribut `InResponseTo` soit absent soit non vérifié par le SP.

```
 # Interception et replay avec curl
SAML_RESPONSE="PHNhbWxwOlJlc3BvbnNlIC..."

# Rejouer dans la fenetre de validite (ici 15 min)
curl -X POST https://sp.example.com/acs \
  -d "SAMLResponse=${SAML_RESPONSE}&RelayState=/dashboard" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -c cookies.txt -b cookies.txt -L -v
```

Fenêtres temporelles et dérive d'horloge

De nombreux SP implémentent une tolérance de dérive d'horloge (clock skew) de 60 à 300 secondes pour compenser les désynchronisations NTP entre IdP et SP. Cette tolérance étend effectivement la fenêtre d'attaque. Un attaquant avec accès aux assertions peut aussi anticiper en préparant le replay avant l'expiration. Dans les environnements cloud multi-régions, la dérive d'horloge peut atteindre plusieurs minutes si NTP n'est pas correctement configuré, créant des fenêtres d'attaque étendues.

Signature Exclusion et Signature Stripping

La Signature Stripping est l'attaque la plus simple contre SAML : l'attaquant supprime purement et simplement l'élément `ds:Signature` de l'assertion, puis modifie le contenu (à savoir NameID et attributs) à sa convenance. Si le SP est configuré pour accepter les assertions non signées (option parfois activée pour la compatibilité avec certains IdP legacy), l'attaque réussit immédiatement.

```
 import base64, re
from lxml import etree

# Decodage du SAMLResponse
saml_b64 = "PHNhbWxwOlJlc3BvbnNlIC..."
saml_xml = base64.b64decode(saml_b64).decode()

# Parsing et modification
root = etree.fromstring(saml_xml.encode())
ns = {"ds": "http://www.w3.org/2000/09/xmldsig#",
      "saml": "urn:oasis:names:tc:SAML:2.0:assertion"}

# Suppression de la signature
for sig in root.findall(".//ds:Signature", ns):
    sig.getparent().remove(sig)

# Modification du NameID
for nameid in root.findall(".//saml:NameID", ns):
    nameid.text = "admin@corp.com"

# Re-encodage
modified = base64.b64encode(etree.tostring(root)).decode()
print(f"SAMLResponse={modified}")
```

Signature sur Response vs Assertion

Une configuration courante et dangereuse : la signature est placée uniquement sur la `Response` (enveloppe) et non sur l'`Assertion` interne. L'attaquant peut alors modifier le contenu de l'assertion sans invalider la signature de la Response, car XML-DSig ne couvre que le noeud

explicitement référencé. Cette configuration est fréquente dans les installations ADFS par défaut et doit être corrigée en forçant la signature de l'assertion.

SAML Misconfiguration : pièges d'implémentation critiques

Au-delà des attaques cryptographiques, les mauvaises configurations représentent la majorité des vulnérabilités SAML observées en pentest. Les erreurs de configuration SAML sont particulièrement insidieuses car elles ne génèrent aucune erreur visible en fonctionnement normal — elles ne se révèlent que sous attaque active.

InResponseTo absent ou non vérifié

L'attribut `InResponseTo` de la SAMLResponse doit correspondre à l'ID de l'AuthnRequest émise par le SP. S'il est absent ou non vérifié, un attaquant peut rejouer une assertion SAML de n'importe quelle session sans avoir initié de flux d'authentification. Les SP bien configurés maintiennent un registre des IDs de requêtes en attente avec un TTL correspondant à la durée de vie des sessions.

AudienceRestriction ignorée

L'élément `AudienceRestriction` précise le SP destinataire de l'assertion. Si un SP ne vérifie pas que son URL figure dans cette liste, une assertion valide pour SP-A peut être rejouée contre SP-B au sein du même IdP. Ce vecteur est particulièrement dangereux dans les environnements multi-tenant où plusieurs applications partagent le même IdP.

Validation du certificat insuffisante

Certains SP acceptent n'importe quel certificat pour valider la signature, sans vérifier qu'il correspond au certificat enregistré pour l'IdP. Un attaquant peut alors signer une assertion forgée avec sa propre clé privée et fournir le certificat public correspondant dans la réponse SAML. La

vérification doit être faite contre un certificat statiquement configuré dans les métadonnées du SP.

```
 # Verification de la configuration SAML metadata
curl -s https://idp.example.com/saml/metadata | xmllint --format - | grep -A5 "KeyDescriptor"

# Verifier que le SP valide bien l'AudienceRestriction
# Test avec audience modifiee dans SAMLRaider
# Si le SP accepte audience=https://evil.com -> vulnerable
```

SAML vs OIDC : comparaison des surfaces d'attaque

OpenID Connect (OIDC) est souvent présenté comme l'alternative moderne à SAML. Les deux protocoles ont des surfaces d'attaque distinctes qu'il convient de comparer pour guider les choix architecturaux en entreprise. Cette comparaison est essentielle pour les architectes IAM qui doivent choisir entre les deux protocoles pour une nouvelle application ou planifier une migration.

SAML : Surface d'attaque concentrée sur la manipulation XML (XSW, XXE potentiel, injection d'entités), les assertions transitant par le navigateur (vol possible via XSS, logs), et la complexité du standard XML-DSig. La validation correcte requiert des bibliothèques robustes et bien maintenues. Les vulnérabilités XSW affectées par les bibliothèques legacy continuent d'être découvertes en 2025-2026.

OIDC/OAuth 2.0 : Vulnérabilités différentes — confusion d'état CSRF, open redirects, vol de code d'autorisation, implicit flow dangereux (déprécié), PKCE absent, mix-up attacks entre IdPs. Les tokens JWT ont leurs propres vulnérabilités (algorithme none, confusion RS256/HS256). OIDC est généralement préféré pour les nouvelles applications en raison d'une implémentation plus simple.

Pour approfondir les techniques d'attaque SSO avancées, consulter notre article sur Golden SAML : forge de tokens SSO et les techniques d'évasion des mécanismes MFA avec EvilGinx et les attaques AiTM.

Outils de test offensif : SAMLRaider et Burp SAML Toolkit

Deux outils s'imposent pour les tests offensifs SAML dans un contexte de pentest professionnel. Leur maîtrise est indispensable pour réaliser un audit complet des implémentations SAML en entreprise.

SAMLRaider (extension Burp Suite)

SAMLRaider est une extension Burp Suite open-source qui intercepte, décode et modifie les flux SAML en temps réel. Elle implémente automatiquement les 8 variantes XSW, la suppression de signature, la modification de NameID et d'attributs. Son interface graphique visualise la structure XML des assertions et permet des modifications ciblées sans connaissance approfondie de XML-DSig.

```
 # Installation SAMLRaider via Burp Suite BApp Store
# Ou compilation manuelle :
git clone https://github.com/SAMLRaider/SAMLRaider.git
cd SAMLRaider
mvn package
# Charger le .jar dans Burp Suite > Extender > Add
```

SAML Toolkit et python3-saml

Pour les tests en ligne de commande et l'automatisation, la bibliothèque python3-saml de OneLogin permet de parser et forger des assertions. Combinée avec lxml et xmlsec, elle permet des PoC complets pour les 8 variantes XSW. Cette approche scriptée est particulièrement utile pour les tests de régression automatisés après correction.

```
 # Installation des dependances
pip3 install python3-saml lxml xmlsec

# Test de base : verification si un SP accepte les assertions non signees
# Intercepter une SAMLResponse valide avec Burp
# Supprimer le bloc ds:Signature
# Rejouer et observer si la connexion reussit
```

Pour les contextes de pentest avancés impliquant l'évasion des EDR, voir notre article sur les techniques de bypass EDR. Les méthodologies d'audit IA et SAML convergent également dans les contextes cloud — voir pentest des systèmes IA.

Défenses : implémentation sécurisée SAML 2.0

La sécurisation d'une implémentation SAML repose sur une validation stricte à chaque étape du flux. Les mesures présentées ci-dessous sont classées par ordre de priorité et correspondent aux recommandations de l'OWASP et aux bonnes pratiques industrie 2026.

Validation de la signature

Vérifier que la signature couvre bien l'assertion (et non seulement la Response), que le certificat de signature est explicitement configuré (pas de récupération dynamique), et que l'algorithme de signature est fort (RSA-SHA256 minimum, bannir SHA-1 et RSA-1024). Les bibliothèques SAML modernes comme Spring Security SAML ou python3-saml OneLogin implémentent ces contrôles par défaut à condition d'utiliser leur configuration sécurisée.

```
 import xmlsec
from lxml import etree

def validate_saml_assertion(saml_xml: bytes, idp_cert_path: str) -> bool:
    root = etree.fromstring(saml_xml)
    manager = xmlsec.KeysManager()
    manager.load_cert(idp_cert_path, xmlsec.KeyFormat.CERT_PEM, xmlsec.KeyDataType.TRUSTED)
    ctx = xmlsec.SignatureContext(manager)
    signature_node = xmlsec.tree.find_node(root, xmlsec.constants.NodeSignature)
    if signature_node is None:
        raise ValueError("Assertion non signee - rejetee")
    ctx.verify(signature_node)  # Leve une exception si invalide
    return True
```

Validation temporelle stricte (NotOnOrAfter)

Appliquer strictement les attributs temporels avec une tolérance de dérive d'horloge minimale (≤ 60 secondes). Maintenir un cache Redis ou en mémoire des IDs d'assertions déjà consommés pendant toute la durée de vie des assertions (TTL = NotOnOrAfter - NotBefore + clock_skew). Ce cache anti-rejeu est le contrôle le plus important pour éliminer les attaques de replay.

AudienceRestriction et InResponseTo obligatoires

Rejeter toute assertion dont l'`Audience` ne correspond pas exactement à l'URL du SP (comparaison stricte, pas de wildcards). Vérifier systématiquement que l'`InResponseTo` correspond à un ID de requête AuthnRequest récent émis par ce SP. Ces deux contrôles éliminent les attaques cross-SP et les replays non sollicités dans les architectures multi-SP.

Défense contre XSW et validation PKIX

Utiliser des bibliothèques SAML activement maintenues et vérifier leur version contre les CVE connues. Valider que la bibliothèque résout correctement les noeuds signés par ID et non par position dans l'arbre DOM. Le certificat de l'IdP doit être épinglé (pinning) dans la configuration du SP — ne jamais faire confiance à un certificat fourni dynamiquement dans les métadonnées SAML non signées.

```
## SAML Security Checklist
# [ ] Signature de l'Assertion requise (pas seulement la Response)
# [ ] Algorithme RSA-SHA256 minimum (SHA-1 interdit)
# [ ] Certificat IdP epingle dans la config SP (pas de fetch dynamique)
# [ ] NotOnOrAfter verifie (clock skew <= 60s)
# [ ] Cache anti-rejeu sur les IDs d'assertions (Redis/memoire)
# [ ] InResponseTo verifie sur les IDs AuthnRequest
# [ ] AudienceRestriction validee strictement (comparaison exacte)
# [ ] Assertions non signees rejetees (pas de optional signature)
# [ ] Metadonnees IdP signees et verifiees
# [ ] Logs d'authentification SAML centralises (SIEM)
# [ ] Test SAMLRaider avant toute mise en production
```

FAQ — Questions fréquentes sur les attaques SAML

Quelle est la différence entre une attaque XSW et une injection XML classique ?

Une injection XML classique (comme XXE — XML External Entity) exploite le parser XML pour accéder à des ressources systèmes ou exfiltrer des données via des entités externes. L'XML Signature Wrapping est fondamentalement différent : elle ne tente pas d'exploiter le parser lui-même, mais tire parti de la séparation entre la validation cryptographique (qui suit les références URI) et la logique applicative (qui parcourt le DOM selon un chemin XPath prédéfini). Le parser traite correctement le XML ; c'est l'écart entre ce que la signature protège et ce que l'application utilise qui est exploité. Les deux attaques peuvent coexister sur un même endpoint SAML vulnérable.

Comment détecter une tentative d'attaque XSW dans les logs ?

Les attaques XSW génèrent des anomalies détectables dans les logs SAML : des assertions dont l'ID référencé dans la signature diffère de l'ID du noeud Assertion principal, des SAMLResponse contenant plusieurs noeuds Assertion (normalement un seul), des assertions avec des IDs non enregistrés dans le cache anti-rejeu, ou une taille anormalement grande du SAMLResponse. Côté

SIEM, créer des alertes sur : (1) authentifications SAML avec assertions multiples, (2) assertions dont le NameID ne correspond pas à un compte actif, (3) replays d'IDs d'assertions déjà vus. Les solutions IdP modernes (Okta, Azure AD) incluent leur propre détection intégrée.

SAMLRaider détecte-t-il automatiquement toutes les variantes XSW exploitables ?

SAMLRaider propose un mode "XSW Attack" qui teste séquentiellement les 8 variantes et affiche le résultat de chaque soumission. Cependant, l'interprétation requiert une intervention humaine : l'outil indique si la réponse HTTP est un succès (code 200, redirection vers l'application) mais ne sait pas si le NameID modifié a été effectivement accepté. Il faut analyser la réponse applicative pour confirmer la compromission. De plus, SAMLRaider ne teste pas les combinaisons de variantes (double wrapping) ni les attaques namespace-based (XSW7/8) qui nécessitent parfois des adaptations manuelles. Pour des tests avancés, combiner SAMLRaider avec des scripts Python personnalisés utilisant lxml.

Sources et références

MITRE ATT&CK — Tactiques, techniques et procédures offensives

OWASP — Top 10 des risques applicatifs

ANSSI — Panorama de la cybermenace 2024

