Programmation sécurisée (Secure coding)
Open Worldwide Application Security Project (OWASP)

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 0.1

Plan

- Qu'est-ce que l'OWASP ?
- Mission et vision de l'OWASP
- Présentation des principales ressources d'OWASP (Top 10, ASVS, Cheat Sheets, ZAP)
- Importance de la sécurité des applications dans le monde actuel

Introduction à OWASP

- **Qu'est-ce que l'OWASP ?**
  L'Open Worldwide Application Security Project (OWASP) est une communauté en ligne ouverte et collaborative, dédiée à l'amélioration de la sécurité des logiciels et des applications.
- **Objectif de l'OWASP**
  L'objectif principal de l'OWASP est de sensibiliser, d'éduquer et de fournir des outils, des normes et des bonnes pratiques pour le développement sécurisé d'applications.
- **Pourquoi est-ce important ?**
  Avec l'augmentation des cyberattaques, il est crucial de garantir la sécurité des applications. L'OWASP joue un rôle clé en identifiant les principales menaces et en fournissant des ressources pour aider les organisations à s'en protéger.
- **Principales ressources OWASP**
  L'OWASP propose des projets majeurs tels que l'*OWASP Top 10*, l'*Application Security Verification Standard (ASVS)*, les *Cheat Sheets* et des outils de test de sécurité comme *OWASP ZAP*.

OWASP Top 10

- Présentation générale du Top 10 OWASP
- Les 10 principaux risques de sécurité :
  1. Contrôle d'accès défaillant
  2. Échecs cryptographiques
  3. Injections (SQL, NoSQL, etc.)
  4. Conception non sécurisée
  5. Mauvaise configuration de sécurité
  6. Composants vulnérables et obsolètes
  7. Problèmes d'identification et d'authentification
  8. Échec de l'intégrité logicielle et des données
  9. Absence de journalisation et de surveillance
  10. Falsification de requêtes côté serveur (SSRF)

1. Contrôle d'accès défaillant

Qu'est-ce qu'un contrôle d'accès défaillant ?

- Le contrôle d'accès défaillant survient lorsque les utilisateurs peuvent accéder à des ressources ou des fonctionnalités qu'ils ne devraient pas pouvoir atteindre.
- Cela inclut l'accès à des données sensibles, la possibilité de modifier des permissions ou de manipuler des actions critiques.

Pourquoi est-ce important ?

- Peut conduire à des violations de données et compromettre la confidentialité, l'intégrité et la disponibilité.
- C'est l'une des principales failles de sécurité, selon le Top 10 de l'OWASP.

Causes et Conséquences

Causes courantes :

- Absence de vérification des privilèges d'accès.
- URL non sécurisées permettant un accès direct aux ressources.
- Mauvaise gestion des permissions des utilisateurs.
- Contrôles d'accès côté client au lieu d'être côté serveur.

Conséquences potentielles :

- Divulgation d'informations sensibles (données des utilisateurs, fichiers confidentiels, etc.).
- Escalade de privilèges : un utilisateur standard peut devenir administrateur.
- Altération des données, usurpation d'identité ou accès à des actions critiques.

Exemple

Cas concret :

- Une application web permet aux utilisateurs d'accéder à leurs factures via l'URL suivante :

  `https://exemple.com/factures/1234.pdf`

- Problème : L'utilisateur malveillant modifie l'URL en `/factures/1235.pdf` et accède à la facture d'un autre utilisateur.
- **Cause** : Mauvais contrôle des permissions sur les ressources accessibles par URL.

Solutions possibles :

- Utilisation de tokens d'accès uniques.
- Vérification stricte des permissions utilisateur au niveau du serveur.
- Utilisation de chemins d'accès non prévisibles.

2. Échec cryptographique

Qu'est-ce qu'un échec cryptographique ?

- Un échec cryptographique se produit lorsque les données sensibles ne sont pas correctement protégées.
- Cela inclut l'absence de chiffrement, l'utilisation d'algorithmes faibles ou de mauvaises pratiques de gestion des clés.

Pourquoi est-ce important ?

- Les données sensibles (mots de passe, informations personnelles, transactions) peuvent être interceptées ou manipulées.
- Les pirates peuvent accéder à des données critiques ou compromettre la confidentialité, l'intégrité et l'authenticité des données.
- L'OWASP place ce risque parmi les plus critiques dans son Top 10.

Causes et Conséquences

Causes courantes :

- Utilisation d'algorithmes cryptographiques obsolètes (ex : MD5, SHA-1).
- Chiffrement non appliqué à des données sensibles (ex : données en transit ou au repos).
- Mauvaise gestion des clés (clés codées en dur, clés trop courtes ou non renouvelées).
- Absence de sécurité des canaux de communication (pas de TLS/SSL).

Conséquences potentielles :

- Fuite de données sensibles (mots de passe, numéros de carte de crédit, etc.).
- Interception des communications réseau par des attaques de type "Man-in-the-Middle" (MITM).
- Altération des fichiers ou des bases de données, compromettant l'intégrité des données.

Exemple

Cas concret :

- Une application mobile envoie des informations de connexion à l'API backend en utilisant `HTTP` au lieu de `HTTPS`.
- Problème : Les informations de connexion (nom d'utilisateur et mot de passe) sont transmises en clair.
- Conséquence : Un attaquant sur le même réseau peut capturer les identifiants via une attaque *Man-in-the-Middle* (MITM).

Solutions possibles :

- Toujours utiliser `HTTPS` au lieu de `HTTP` pour sécuriser les communications réseau.
- Utiliser des algorithmes de chiffrement modernes (AES, SHA-256) et éviter les algorithmes obsolètes (MD5, SHA-1).
- Protéger les clés de chiffrement (ne pas les coder en dur dans le code source).
- Mettre en œuvre des pratiques de gestion des clés (rotation des clés, expiration, etc.).

3. Injection

Qu'est-ce qu'une injection ?

- Une injection se produit lorsque des données non fiables sont insérées dans une requête ou une commande exécutée par une application.
- Les injections peuvent affecter les bases de données SQL, NoSQL, LDAP, OS (commandes système) et d'autres systèmes.

Pourquoi est-ce important ?

- Peut permettre à un attaquant d'exécuter des requêtes non prévues par le développeur.
- Peut conduire à des vols de données, à la modification de la base de données ou à la prise de contrôle du système.
- Les injections SQL sont l'une des failles de sécurité les plus critiques de l'OWASP Top 10.

Types d'injections et Conséquences

Types d'injections courantes :

- **Injection SQL** : Insertion de commandes SQL dans des champs de saisie utilisateur.
- **Injection NoSQL** : Affecte les bases NoSQL comme MongoDB via l'injection de commandes JSON.
- **Injection de commandes OS** : Permet d'exécuter des commandes système sur le serveur.
- **Injection LDAP** : Exploitation de requêtes LDAP (utilisé pour authentification et répertoires).

Conséquences possibles :

- **Fuite de données** : Vol de données sensibles (mots de passe, cartes de crédit, etc.).
- **Modification des données** : Suppression ou modification de fichiers et d'enregistrements de la base.
- **Exécution de commandes** : Possibilité d'exécuter des commandes sur le serveur.
- **Escalade de privilèges** : Un attaquant peut accéder à des ressources réservées aux administrateurs.

Exemple

Exemple concret d'injection SQL :

- Requête SQL classique :

```
SELECT * FROM utilisateurs WHERE nom='input_usr' AND mot_de_passe='input_mdp';
```

- **Attaque** : L'utilisateur malveillant saisit `' OR '1'='1` dans le champ "mot de passe".

```
SELECT * FROM utilisateurs WHERE nom='admin' AND mot_de_passe='' OR '1'='1';
```

- **Impact** : La condition `'1'='1'` est toujours vraie, ce qui permet à l'attaquant d'accéder au compte admin.

Solutions

Solutions pour prévenir les injections :

- Utiliser des **requêtes paramétrées** (Prepared Statements) et des **requêtes avec des bind variables**.
- Valider les entrées utilisateur (ex : ne pas autoriser de caractères spéciaux comme `'`, `"`, `;`, etc.).
- Éviter la concaténation directe des données utilisateur dans les requêtes.
- Utiliser des ORM (Object-Relational Mapping) qui gèrent la sécurité des requêtes SQL.

4. Conception non sécurisée

Qu'est-ce qu'une conception non sécurisée ?

- La conception non sécurisée se produit lorsqu'une application ou un système n'intègre pas de pratiques de sécurité dès la phase de conception.
- Contrairement à une vulnérabilité d'implémentation, il s'agit d'une faille "structurelle" liée à l'absence de mesures de sécurité prévues lors de la planification et de l'architecture du système.

Pourquoi est-ce important ?

- Une mauvaise conception de la sécurité peut laisser des portes ouvertes pour les attaques, même si le code est exempt de vulnérabilités techniques.
- Les failles de conception sont souvent coûteuses à corriger après la mise en production.
- Cela peut conduire à des fuites de données, des dénis de service ou des compromissions des fonctionnalités critiques.

Causes et Conséquences

Causes courantes d'une conception non sécurisée :

- Absence de pratiques de "Security by Design" dès la phase de conception.
- Défauts dans la modélisation des menaces et des risques.
- Manque de contrôle des flux d'accès ou de séparation des privilèges.
- Absence de protection contre les abus de fonctionnalités (ex : limitation des taux d'accès, vérification des entrées utilisateur, etc.).
- Omission des tests de sécurité lors de la conception.

Conséquences possibles :

- **Violation de la confidentialité** : Fuite de données sensibles due à une mauvaise gestion des autorisations.
- **Disponibilité compromise** : Absence de mécanismes de limitation de taux (rate-limiting), conduisant à des attaques par déni de service (DoS).
- **Escalade de privilèges** : Un utilisateur avec peu de privilèges peut accéder à des fonctionnalités d'administration.
- **Failles logiques** : Une logique métier incorrecte permet des abus de la fonctionnalité.

Exemple

Exemple concret de conception non sécurisée :

- **Cas d'utilisation** : Une plateforme de réservation d'hôtel permet aux utilisateurs d'annuler leur réservation.
- **Problème** : L'utilisateur envoie une requête API du type :

```
POST /annuler_reservation?id=12345
```

- **Faute de conception** : Il n'y a pas de vérification que l'utilisateur authentifié est bien le propriétaire de la réservation.
- **Impact** : Un utilisateur malveillant peut deviner des identifiants de réservation et annuler les réservations d'autres utilisateurs.

titre

Solutions :

- Appliquer le principe du **moindre privilège** (Least Privilege) et la séparation des responsabilités.
- Effectuer des **revues de conception de sécurité** (Security Design Review) avec des experts en sécurité.
- **Modélisation des menaces** : Identifier les menaces potentielles dès le début de la phase de conception.
- Mettre en place des **contrôles d'accès** rigoureux (vérification que l'utilisateur peut accéder uniquement à ses ressources).
- Appliquer des techniques de **limitation de taux** (rate-limiting) et de **coupure de session** (session timeout).

5. Mauvaise configuration de sécurité

Qu'est-ce qu'une mauvaise configuration de sécurité ?

- Une mauvaise configuration de sécurité se produit lorsqu'un composant, une application ou un serveur n'est pas configuré de manière sécurisée.
- Cela peut inclure des erreurs au niveau des paramètres par défaut, des autorisations de fichiers excessives, des messages d'erreur exposant des informations sensibles, etc.

Pourquoi est-ce important ?

- Les configurations par défaut peuvent être connues des attaquants, ce qui facilite leur exploitation.
- Une mauvaise configuration peut exposer des informations sensibles, telles que les clés API, les chemins de fichiers, ou les détails de la stack applicative.
- Cela permet aux attaquants d'exploiter les failles pour accéder à des systèmes, des bases de données et des informations confidentielles.

Causes

Causes courantes d'une mauvaise configuration de sécurité :

- **Utilisation de paramètres par défaut** : Mots de passe par défaut non modifiés.
- **Autorisation excessive** : Permissions des fichiers trop permissives (`chmod 777`).
- **Exposition d'informations sensibles** : Les messages d'erreur révèlent les chemins d'accès ou les détails du serveur.
- **Mauvaise configuration des en-têtes HTTP** : Absence des en-têtes `X-Frame-Options`, `Strict-Transport-Security`, `Content-Security-Policy`, etc.
- **Mauvaise configuration des services cloud** : Les buckets S3, les bases de données ou les conteneurs Docker sont laissés ouverts au public.

Conséquences

Conséquences possibles :

- **Fuite de données sensibles** : Informations exposées (fichiers de configuration, API Keys, chemins de fichiers, etc.).
- **Accès non autorisé** : Les attaquants peuvent accéder à des systèmes ou des interfaces d'administration.
- **Exécution de commandes malveillantes** : Via l'exploitation des permissions excessives sur les fichiers ou les services.
- **Altération de la disponibilité** : Les attaquants peuvent désactiver ou perturber les services.

Exemple

Exemple concret de mauvaise configuration de sécurité :

- **Cas d'utilisation** : Une application web expose des messages d'erreur détaillés, par exemple :

```
ERROR: /var/www/html/app/config.php line 12: Undefined variable $db_host
```

- **Problème** : Les messages d'erreur exposent des informations sensibles sur l'architecture et le chemin des fichiers.
- **Impact** : Un attaquant peut obtenir des informations précieuses pour cibler des fichiers spécifiques, exécuter des injections de chemin (path traversal) ou détecter les versions des bibliothèques utilisées.

Solution

Solutions pour éviter les mauvaises configurations de sécurité :

- Désactiver l'affichage des **messages d'erreur détaillés** sur les environnements de production.
- **Restreindre les permissions** des fichiers et des dossiers (`chmod 644` ou `chmod 600` au lieu de `chmod 777`).
- Désactiver ou supprimer les services inutilisés (interfaces d'administration, outils de test, etc.).
- Configurer les **en-têtes de sécurité HTTP** (`Content-Security-Policy`, `Strict-Transport-Security`, etc.).
- Protéger l'accès aux services cloud (Amazon S3, bases de données) en activant des contrôles d'accès basés sur l'identité (IAM) et en désactivant l'accès public.

6. Composant vulnérable ou obsolète

Qu'est-ce qu'un composant vulnérable ou obsolète ?

- L'utilisation de bibliothèques, de frameworks, de modules ou de services tiers qui contiennent des failles de sécurité connues.
- Ces composants peuvent inclure des bibliothèques open-source, des dépendances tierces ou même des outils internes obsolètes.

Pourquoi est-ce important ?

- Les vulnérabilités dans les bibliothèques tierces sont exploitées par les attaquants pour accéder au système.
- Les applications qui utilisent des composants non mis à jour sont des cibles faciles.
- Les failles critiques dans des frameworks populaires (ex : Log4j) peuvent entraîner des attaques massives et des compromissions de systèmes.

Causes

Causes courantes de l'utilisation de composants vulnérables ou obsolètes :

- **Absence de mise à jour** : Ne pas mettre à jour régulièrement les bibliothèques, frameworks et dépendances.
- **Utilisation de versions obsolètes** : Utilisation de versions d'outils non prises en charge (end-of-life) ou non mises à jour.
- **Dépendances non sécurisées** : Inclusion de bibliothèques open-source sans vérification de leur sécurité.
- **Pas de suivi des vulnérabilités** : Les équipes de développement ne suivent pas les alertes de sécurité des dépendances.

Conséquences

Conséquences possibles :

- **Exploitation à distance** : Les attaquants peuvent utiliser des exploits publics connus.
- **Fuite de données** : Des failles critiques peuvent permettre l'accès à des données sensibles.
- **Contrôle total du système** : Les vulnérabilités de type "Remote Code Execution" (RCE) permettent aux attaquants d'exécuter du code arbitraire.
- **Impact en cascade** : Une seule bibliothèque compromise peut affecter de nombreuses applications qui l'utilisent.

Exemple

Exemple concret de composant vulnérable :

- **Cas d'utilisation** : Une application Java utilise la bibliothèque Log4j pour enregistrer les logs.
- **Problème** : Une vulnérabilité critique (CVE-2021-44228) permet l'exécution de commandes distantes (RCE) via une simple chaîne de texte envoyée dans un champ de saisie.
- **Impact** : Les attaquants peuvent exécuter des commandes sur le serveur et prendre le contrôle complet du système.

Solutions

Solutions pour éviter l'utilisation de composants vulnérables :

- **Surveiller les mises à jour de sécurité** : Suivre les alertes de sécurité et les correctifs des bibliothèques.
- **Utiliser des outils d'analyse de dépendances** : Outils tels que `OWASP Dependency-Check`, `Snyk`, `Dependabot` ou `npm audit`.
- **Utiliser des versions à jour** : Toujours utiliser les dernières versions stables des bibliothèques et frameworks.
- **Réduire les dépendances inutiles** : N'ajouter que les bibliothèques essentielles et éviter les "packages" surchargés.
- **Automatiser les mises à jour** : Mettre en place des processus de mise à jour automatique ou des alertes de mise à jour.

7. Problème d'identification et d'authentification

Qu'est-ce qu'un problème d'identification et d'authentification ?

- Les failles d'identification et d'authentification permettent à un attaquant de voler ou d'usurper l'identité d'un utilisateur légitime.
- Ces problèmes incluent des erreurs de gestion des sessions, de validation des identifiants ou de mise en œuvre des mécanismes d'authentification.

Pourquoi est-ce important ?

- Les failles d'authentification permettent à des attaquants d'accéder à des comptes utilisateurs, y compris des comptes administrateurs.
- Ces failles peuvent entraîner des compromissions de comptes et des fuites de données personnelles ou critiques.
- Ces problèmes figurent parmi les principales failles du Top 10 OWASP.

Causes

Causes courantes des problèmes d'identification et d'authentification :

- **Mots de passe faibles** : Absence de politique de complexité des mots de passe.
- **Absence de verrouillage de compte** : Pas de limitation du nombre de tentatives de connexion (brute force).
- **Gestion de session incorrecte** : Les identifiants de session (tokens) ne sont pas correctement invalidés à la déconnexion.
- **Absence de second facteur d'authentification (2FA)** : Les utilisateurs ne sont authentifiés qu'avec un mot de passe.
- **Exposition des identifiants** : Les identifiants et les mots de passe peuvent être stockés ou transmis en clair.

Conséquences

Conséquences possibles :

- **Usurpation d'identité** : Un attaquant peut accéder aux comptes d'autres utilisateurs.
- **Vol de données sensibles** : Les attaquants peuvent accéder aux informations sensibles des comptes utilisateurs.
- **Escalade de privilèges** : Un utilisateur régulier peut devenir administrateur.
- **Utilisation abusive de ressources** : Les attaquants peuvent utiliser les ressources de l'application ou accéder à des fonctionnalités non autorisées.

Exemple Pratique et Solutions

Exemple concret de problème d'identification et d'authentification :

- **Cas d'utilisation** : Une application ne limite pas le nombre de tentatives de connexion pour l'authentification.
- **Problème** : Un attaquant peut utiliser un outil de force brute (ex : Hydra) pour essayer de deviner les mots de passe d'un utilisateur.
- **Impact** : L'attaquant peut accéder à un compte, obtenir des informations sensibles ou effectuer des actions malveillantes.

Solution

Solutions pour éviter les problèmes d'identification et d'authentification :

- **Politique de mots de passe sécurisés** : Imposer des mots de passe forts (8 caractères min, chiffres, symboles, majuscules, etc.).
- **Limiter le nombre de tentatives de connexion** : Verrouiller le compte après un certain nombre d'échecs de connexion.
- **Utiliser l'authentification à deux facteurs (2FA)** : Demander un second facteur d'authentification (SMS, email, application 2FA).
- **Utiliser des jetons de session sécurisés** : Assurer une bonne gestion des sessions (expiration, rotation de jetons, déconnexion).
- **Ne pas exposer les identifiants** : Ne jamais exposer le mot de passe en clair et ne pas les enregistrer dans les journaux de logs.

8. Échec de l'intégrité logicielle et des données

Qu'est-ce qu'un échec de l'intégrité logicielle et des données ?

- Se produit lorsqu'une application, un fichier ou des données peuvent être modifiés de manière non autorisée.
- Les échecs d'intégrité peuvent affecter le code source, les bibliothèques, les fichiers de configuration ou les données des utilisateurs.
- Cela inclut les attaques de "Supply Chain" (chaîne d'approvisionnement) et les modifications non autorisées des fichiers critiques.

Pourquoi est-ce important ?

- Les attaquants peuvent altérer le comportement des logiciels, insérer des portes dérobées (backdoors) ou modifier les configurations de sécurité.
- Les fichiers de configuration, les dépendances tierces et les mises à jour non sécurisées peuvent être détournés.
- Cela peut conduire à des compromissions de données, de la fraude ou des déni de service.

Causes

Causes courantes de l'échec de l'intégrité logicielle et des données :

- **Mises à jour non sécurisées** : Absence de vérification de la source et de l'intégrité des mises à jour.
- **Chaîne d'approvisionnement compromise** : Insertion de codes malveillants dans des bibliothèques ou des dépendances open-source.
- **Permissions de fichiers incorrectes** : Les fichiers critiques (par exemple, les fichiers de configuration) sont modifiables par des utilisateurs non autorisés.
- **Manque de contrôle d'intégrité** : Absence de hachage ou de signature des fichiers critiques.
- **Utilisation de scripts externes** : Inclusion de scripts distants ou de bibliothèques sans validation d'intégrité.

Conséquences

Conséquences possibles :

- **Insertion de backdoors** : Ajout de portes dérobées dans les logiciels.
- **Fuite ou modification de données sensibles** : Accès non autorisé et modification des fichiers de configuration ou des bases de données.
- **Exécution de commandes malveillantes** : Les bibliothèques compromises peuvent exécuter des scripts malveillants.
- **Compromission des utilisateurs finaux** : Les utilisateurs qui installent des mises à jour compromises deviennent des cibles directes.

Exemple

Exemple concret d'échec de l'intégrité logicielle et des données :

- **Cas d'utilisation** : Une application web inclut une bibliothèque JavaScript distante depuis :

  `https://cdn.exemple.com/script.js`

- **Problème** : Un attaquant compromet la bibliothèque hébergée sur le CDN et injecte du code malveillant.
- **Impact** : L'application intègre le script compromis, ce qui permet à l'attaquant de capturer les saisies utilisateur ou d'exécuter des actions en leur nom.

Solution

Solutions pour éviter les échecs de l'intégrité logicielle et des données :

- **Vérification des mises à jour logicielles** : Utiliser des signatures numériques et des contrôles de hachage (SHA-256) pour valider l'intégrité des mises à jour.
- **Contrôle des dépendances tierces** : Utiliser des outils d'analyse de dépendances (comme `OWASP Dependency-Check` ou `Snyk`) pour détecter les vulnérabilités dans les bibliothèques.
- **Contrôle de l'intégrité des fichiers** : Appliquer des mécanismes de contrôle d'intégrité (par exemple, la vérification des sommes de contrôle SHA) sur les fichiers de configuration.
- **Restreindre les permissions des fichiers** : Réduire les autorisations d'écriture pour les fichiers de configuration et les répertoires critiques.
- **Utilisation de Content-Security-Policy (CSP)** : Empêcher l'exécution de scripts externes non autorisés sur les sites web.

9. Absence de journalisation et de surveillance

Qu'est-ce que l'absence de journalisation et de surveillance ?

- Ce problème se produit lorsqu'une application ne consigne pas les événements critiques ou ne surveille pas activement les actions suspectes.
- L'absence de journalisation et de surveillance empêche de détecter, d'enquêter et de répondre aux incidents de sécurité.

Pourquoi est-ce important ?

- Sans journalisation, les attaques peuvent passer inaperçues.
- Empêche de détecter les comportements suspects, comme les attaques par force brute ou l'accès non autorisé.
- Retarde la réponse aux incidents et complique les enquêtes post-incident.
- Cela peut avoir un impact juridique si la conformité (ex : RGPD) n'est pas respectée.

Causes

Causes courantes de l'absence de journalisation et de surveillance :

- **Absence de journalisation des actions critiques** : Les actions sensibles ne sont pas enregistrées.
- **Absence de détection des comportements anormaux** : Les alertes de sécurité ne sont pas générées.
- **Absence de protection des journaux** : Les journaux peuvent être modifiés ou supprimés par des attaquants.
- **Manque de ressources de surveillance** : Pas d'outils de détection d'anomalies ou de SIEM (Security Information and Event Management).
- **Absence d'alertes en temps réel** : Pas d'alertes sur les connexions suspectes ou les accès non autorisés.

Conséquences

Conséquences possibles :

- **Détection retardée des intrusions** : Les attaques ne sont pas détectées, ce qui permet aux attaquants de rester cachés.
- **Perte d'informations pour l'enquête** : Les preuves numériques (logs) sont manquantes, ce qui rend les enquêtes impossibles.
- **Violations de conformité** : Les réglementations (comme le RGPD) imposent la conservation des journaux d'accès.
- **Coût de remédiation élevé** : Le coût des enquêtes augmente sans journaux de référence.

Exemple

Exemple concret d'absence de journalisation et de surveillance :

- **Cas d'utilisation** : Une application web de gestion de comptes utilisateurs ne journalise pas les tentatives de connexion.
- **Problème** : Un attaquant exécute une attaque par force brute en testant des milliers de mots de passe.
- **Impact** : L'absence de journalisation empêche la détection de l'attaque et l'administrateur ne peut pas enquêter.

Solution

Solutions pour assurer une journalisation et une surveillance efficaces :

- **Enregistrer les événements critiques** : Journaliser les connexions, les échecs d'authentification et les accès aux données sensibles.
- **Protéger les journaux** : Empêcher la suppression ou la modification des journaux, notamment avec des permissions d'accès restreintes.
- **Utiliser un système SIEM** : Centraliser les journaux dans un système SIEM pour faciliter la corrélation des événements.
- **Mettre en place des alertes en temps réel** : Détecter les anomalies en temps réel (nombre excessif de tentatives de connexion, accès anormal, etc.).
- **Effectuer des audits réguliers des journaux** : Revoir les journaux pour détecter les comportements anormaux ou suspects.

10. Falsification de requêtes côté serveur (SSRF)

Qu'est-ce qu'une falsification de requêtes côté serveur (SSRF) ?

- Une attaque où l'attaquant manipule une application pour forcer le serveur à envoyer des requêtes HTTP à des adresses choisies par l'attaquant.
- Cela permet de contourner les pare-feu et de cibler des systèmes internes non accessibles directement depuis l'extérieur.

Pourquoi est-ce important ?

- Les attaques SSRF permettent d'exfiltrer des données internes, d'accéder à des systèmes internes et de scanner les réseaux locaux.
- Les services de métadonnées cloud (par exemple AWS `169.254.169.254`) peuvent être exposés, ce qui permet de récupérer des informations sensibles.
- Les attaques SSRF peuvent également être utilisées comme vecteurs de pivot pour d'autres attaques.

Causes

Causes courantes de la SSRF :

- **Mauvaise validation des URL** : Les URL saisies par les utilisateurs ne sont pas correctement validées ou nettoyées.
- **Utilisation d'URL en paramètre** : Des applications qui récupèrent des données à partir d'URL contrôlées par l'utilisateur.
- **Absence de contrôle d'accès réseau** : Le serveur peut accéder aux adresses IP internes et aux services de métadonnées cloud.
- **Absence de contrôle des schémas d'URL** : Les schémas d'URL (par exemple `file://`, `gopher://`) ne sont pas correctement bloqués.

Conséquences

Conséquences possibles :

- **Accès aux services internes** : L'attaquant accède à des bases de données internes, des systèmes internes ou des services internes (ex : `localhost`, `127.0.0.1`).
- **Accès aux métadonnées du cloud** : Accès à l'API des métadonnées de cloud (AWS, GCP, Azure) pour voler des tokens d'identification.
- **Scanning du réseau interne** : L'attaquant peut utiliser le serveur comme un proxy pour scanner le réseau interne.
- **Fuite d'informations sensibles** : Des informations telles que les clés d'API, les tokens et les configurations sensibles peuvent être exfiltrées.

Exemple

Exemple concret de falsification de requêtes côté serveur (SSRF) :

- **Cas d'utilisation** : Une application web permet de récupérer des images à partir d'une URL.
- **Problème** : L'utilisateur malveillant saisit l'URL suivante :

```
http://169.254.169.254/latest/meta-data/iam/security-credentia
```

- **Impact** : L'attaquant peut accéder aux métadonnées d'instance cloud AWS et obtenir des informations sensibles comme des tokens d'API.

Solutions

Solutions pour éviter les attaques SSRF :

- **Valider les URL en entrée** : Bloquer les schémas d'URL comme `file://`, `gopher://`, `ftp://` et limiter les domaines accessibles.
- **Utiliser des listes d'autorisation (allowlist)** : Restreindre les URL accessibles à des domaines approuvés uniquement.
- **Bloquer l'accès aux adresses IP internes** : Empêcher l'accès aux adresses IP internes comme `127.0.0.1`, `169.254.169.254`, `10.0.0.0/8`.
- **Utiliser des proxys sécurisés** : Les requêtes de l'application doivent passer par un proxy qui contrôle et filtre les URL accessibles.
- **Surveiller et journaliser les requêtes sortantes** : Surveiller les requêtes sortantes pour détecter les accès suspects.

OWASP ZAP

- Installation et configuration de OWASP ZAP
- Scans passifs et actifs
- Analyse des failles (XSS, injections, etc.)
- Démonstration sur une application vulnérable (ex : Juice Shop)

Tests de Sécurité des Applications

- Étude de cas pratiques sur des applications vulnérables (ex : DVWA, Juice Shop)
- Tests d'injection SQL, XSS et contrôle d'accès
- Analyse et interprétation des résultats
- Correctifs et remédiations des failles identifiées

Bonnes Pratiques de Sécurité

- Concepts de *Security by Design*
- Intégration de la sécurité dans le SDLC (cycle de vie de développement logiciel)
- Bonnes pratiques de codage sécurisé
- Vérification des entrées et protection des API

Analyse et Remédiation des Risques

- Évaluation de l'impact des vulnérabilités
- Priorisation des correctifs
- Gestion des risques métiers
- Contre-mesures et plan d'action