TP1

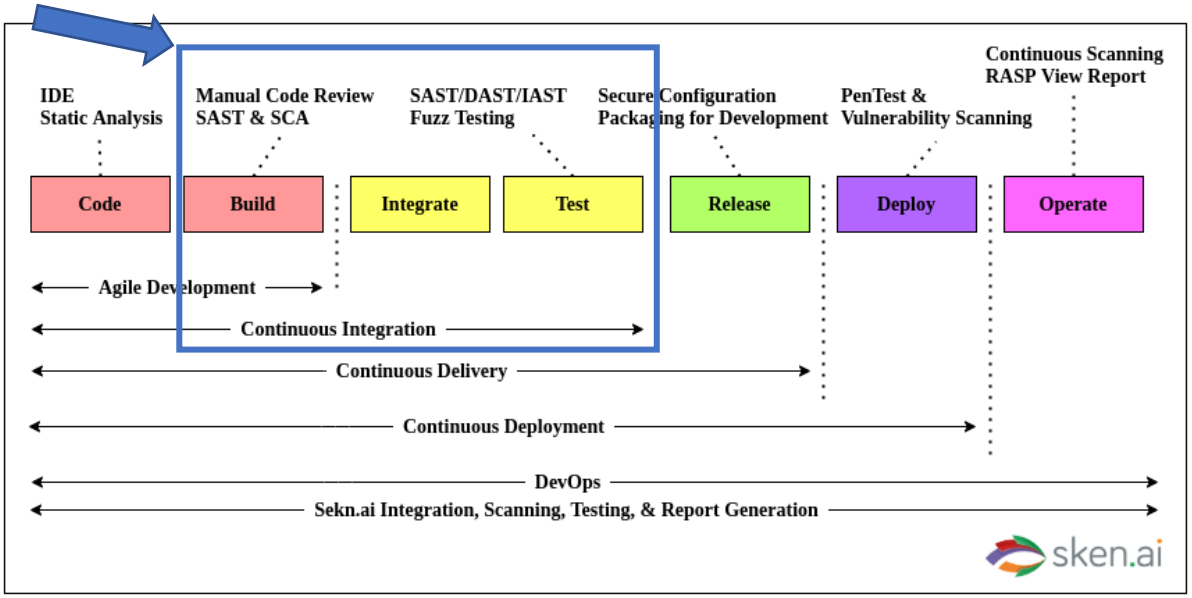
Outillage sécurité web

L'objectif du TP est de mettre en œuvre les principales techniques d'attaque permettant d'exploiter des vulnérabilités web, et de mieux comprendre comment s'en prémunir.

Prérequis :

- Serveur web (via WAMP/MAMP/LAMP), ou machine virtuelle avec Apache/PHP/MySQL
- Instance d'où dérouler le TP (votre machine, ou une machine virtuelle)

Le TP va s'inscrire dans le processus DevSecOps (développement sécurisé) suivant :



A cheval sur les étapes de développement et d'intégration, on va s'attacher à traiter la sécurité du code de façon statique (SAST), et de façon dynamique (DAST).

A noter que nos actions vont être effectuées manuellement, potentiellement automatisées avec des outils, mais pourront être, dans le cadre d'un développement en entreprise, intégrées dans des processus totalement automatisées (comme le propose par exemple GitLab).

#1 Préparer le TP (15min)

L'objet du présent cours n'est pas de faire du développement web, et nous allons réutiliser des applicatifs existants, publiquement disponibles, ou issus des autres modules de cours sur lesquels ont a eu l'occasion de travailler.

Projets qui devront être testés :

A. DVWA
B. [facultatif] Un projet PHP de votre choix, issu par exemple du module « Laravel » ou « PHP7 MVC ».
C. [facultatif] Un projet wordpress (possible de réutiliser votre projet, ou déployer un spécifiquement).

#2 Testons manuellement la sécurité de DVWA (2 heures 30)

Vous voilà dans la peau d'un hacker qui vient de cibler un site web. Votre objectif, tenter toutes les techniques pour exfiltrer de la donnée, modifier le comportement du site, et faire une escalade de privilèges.

Positionnez la sécurité de DVWA au niveau le plus faible (Low).

A. Injections SQL
   - Essayez via ce champ de faire afficher la liste de tous les utilisateurs
   - Essayez via ce champ de faire afficher la version de la base de données
   - Essayez via ce champ de faire afficher la liste de toutes les tables
   - Bonus (facultatif) :
     - On est dans un SGBD. On sait qu'il existe un Schéma dans lequel se trouvent les tables. Essayez via ce champ de faire afficher la liste de toutes les tables de INFORMATION_SCHEMA
     - Affichez tous les champs de colonnes dans la table user d'INFORMATION_SCHEMA
     - Affichez tous les contenus de champs de colonnes dans la table user d'INFORMATION_SCHEMA
     - Grâce aux « Rainbow tables », retrouver le mot de passe hashé

B. [facultatif] XSS reflected
   - Tentez de faire faire quelque-chose à la page en manipulant l'URL

C. XSS stored
   - Tentez d'injecter durablement le script `<script>alert('coucou');</script>` à chaque chargement de la page.

D. Local File Inclusion (LFI)
   - On peut constater l'URL : http://serveur/affichage.php?fichier=index.php
   - Tentez d'afficher un fichier illégitime en manipulant l'URL

E. [facultatif] Remote File inclusion (RFI)
   - On a vu la LFI, essayons maintenant de faire récupérer à notre applicatif du contenu extérieur (qui pourra être malveillant lorsque l'attaquant l'est, et si `allow_url_open` est dans php.ini).
   - Essayez de faire afficher/exécuter du code externe (à trouver sur internet).

#3 Tester le code de façon statique (30 minutes)

➢ Indiquez si oui ou non il y a un intérêt à tester le code avant de lancer une compilation, ou une release. Justifiez votre choix en 3 lignes.

Passons à la pratique

1. Ouvrez un projet (#1, A, B ou C).
2. Installez l'outil SAST de HCL AppScan CodeSweep (extension Visual Studio ou jetBrain).
3. Testez votre code.
4. Rédigez un court rapport des éléments qui ont été identifiés (s'il y en a), et analysez au moins une des vulnérabilités pour expliquer comment elle a été rendue possible, et la mesure à prendre.

#4 Tester le code de façon dynamique (45 minutes)

- Indiquez si oui ou non il y a un intérêt à tester le code alors qu'on l'a déjà testé de façon statique, et si oui, en quoi cela représente une différence avec le #3. Justifiez votre choix en 3 lignes.

Passons à la pratique avec le premier outil

1. Ouvrez un projet (#1, A, B ou C).
2. Installez OWASP ZAP! (logiciel de l'OWASP dédié au DAST).
3. Testez le site.
4. Rédigez un court rapport des éléments qui ont été identifiés (s'il y en a), et analysez au moins une des vulnérabilités pour expliquer comment elle a été rendue possible, et la mesure à prendre.

Passons à la pratique avec le deuxième outil

1. Ouvrez un projet (#1, A, B ou C).
2. Installez Burp Suite (logiciel en accès community gratuit (suffisant) dédié au DAST).
3. Testez le site.
4. Rédigez un court rapport des éléments qui ont été identifiés (s'il y en a), et analysez au moins une des vulnérabilités pour expliquer comment elle a été rendue possible, et la mesure à prendre.

#5 Tester une réelle application (1 heure, en groupes de 4)

Maintenant que vous savez utiliser les outils DAST, vous allez vous attaquer à une réelle application qui fonctionne sur un serveur.

Adresse du service à tester : https://sciencesu.onylrocks.tk

A. En mode visiteur anonyme, tentez de trouver des failles avec OWASP ZAP! Et Burp Suite.
B. Répétez cette opération en étant connecté en tant qu'un utilisateur :
    a. Login : john@doe.test
    b. Password : isitsafe

Rédigez un court rapport de cette étape, et comparez les résultats avec les #2 #3 et #4.