I. Sécurité Web

1. Injection SQL

- Les données utilisateurs servent à construire une requête SQL
- Attaque : Evoyer des données permettant de modifier la requête, si certains éléments ne sont pas échappés.
- Ex : « ' or 1=1# » en tant qu'user
- Nombre de colonnes : ORDER BY (1 | 2 | 3 | 4 | …) jusqu'à erreur
- Nombre de ligne : UNION SELECT 1, 2, 3, 4, … jusqu'à non erreur
- Parenthèses, etc. : Tâtonnement.
- Récupérer version MySQL (@@version) : Dichotomie sur chaque octet avec substring
- Lecture et écriture de fichier sur le serveur : LOAD_FILE, DUMPFILE
- Solution : échappement ou préparation de requêtes

2. Injection XSS

- Déposer du code à faire exécuter par le navigateur web (permanent ou volatile)
- Solution : échapper le HTML dans les input

3. Malicious file execution

- Exécuter son propre fichier sur le serveur
- Application de dépôt de fichier non sécurisée ou appel d'un code distant
- Solution : Bloquer l'utilisation de fonctions système, …

4. Broken authentification and session management

- Deviner le numéro de session d'un user ou modifier les variables de session
- Failles :
    - Numéro de session trop peu aléatoire
    - Données de session stockées côté client

5. Insecure direct object reference

- Modifier un ID qui est envoyé par l'user pour accéder à des éléments qu'on ne devrait pas pouvoir voir.
- Solution : vérifier les droits

6. CSRF : Cross Site Request Forgery

- Faire charger des pages du site A depuis le site B par l'utilisateur
- Solutions :
    - Utiliser POST au lieu de GET complique le CSRF
    - Utilisation de token

7. Insecure cryptographic storage

- Mots de passes hashés et salés.

II. Failles applicatives

1. Buffer overflow

a. Stack Overflow

- Liées à une mauvaise gestion de la mémoire
  - Injection d'un shellcode dans la mémoire (court, pas de \0, ASM)
  - Ecraser l'adresse de retour d'un appel de fonction par l'adresse du shellcode
  - Au lieu de faire un retour de fonction, on « retourne » au shellcode
- Pour connaitre l'adresse de son shellcode :
  - Accès a la machine
  - Bruteforce
- Protection :
  - **Niveau programme :** Utilise fonction type strn*
  - **Niveau compilateur :** Utilisation de canaris (ajout de données aléatoires dans la stack, vérifiées à la sortie, si modifié, process tué)
  - **Niveau kernel :**
    - rendre la stack non executable
      - Saut vers une fonction intéressante
    - Adress space layout radomize
      - La valeur de retour est aléatoire
      - L'adresse du shellcode change tout le temps
      - Contournement :
        - Faire un saut vers un endroit statique
        - NOP Spray : ajouter beaucoup de NOP pour augmenter la taille de la zone du shellcode
        - Return oriented programming : saut vers un programme mappé statiquement qui a un interet
        - Bruteforce très longue…

b. Heap Overflow

- Même principe avec des malloc()
- Plus difficile

2. Autres failles

- Race condition : Exploiter le temps entre la vérification d'existence d'un fichier et son écriture
- Integer overflow : Exploiter la gestion des integer (65636 $\rightarrow$ 0)
- Format string : Utilisation de %n dans printf :
  - printf("%s %n", str, n); affiche str et stocke la longueur de str dans n.
  - Si un programme contient : printf(str); et que str contient des %n, on peut écrire dans le programme.
- Erreur d'implémentation cryptographique

3. Détecter les failles

- Lire le code source
- Lire le code assembleur
- Détecter quand un programme plante
- …

III. Mots de passe faibles

- Attaque online : Essayer des couples login/password par défaut ou courants
- Attaque offline : A partir d'une liste login/password chiffrés on cherche les passwords
    - Essai en direct
    - Rainbow table : hash précalculés

IV. Vulnérabilité Linux

- Obtenir accès root
    - Failles kernel
    - Utilisation crontab
        - Modifier la cron pour faire executer des choses en tant que root
    - Fichier suid root