Vulnérabilités communes :

- **verticales** : utiliser des fonctions quand notre rôle ne le permet pas (user -> admin)
- **horizontales** : accéder aux ressources d'autres utilisateurs du même niveau
- **dépendant du contexte** : accéder hors du flux d'exécution normal (bypass de l'étape de paiement)

Contrôles d'accès :

- **par connaissance d'URL** : ne pas se baser sur l'ignorance des utilisateurs pour les URLs, on les trouve autre part que sur la page (code source, logs, scripts génération de menus, …).
- **accès direct à l'API** : l'API doit être sécurisée comme les pages standards de modification, les fonctions doivent vérifier les privilèges de l'utilisateur.
- **sur les identifiants** : les IDs des ressources peuvent être devinés, les GUIDs prévus. Ils peuvent être affichés ailleurs. Connaitre les IDs ne doit pas suffire pour accéder aux ressources. Il faut vérifier les droits d'accès de l'utilisateur.
- **sur la requête** : contrôles faits par Apache sur l'URL demandée, type de fichier, méthode HTTP (valide ou non), ID user, cookies, … Attention à la configuration, et fichiers statiques -> renvoyer une copie du fichier après authentification.
- **fonctions en plusieurs étapes** : contrôler les accès à chaque étape et vérifier les étapes précédentes (éviter le bypass d'étape), ne pas faire confiance au header REFERER.
- **sur des paramètres** : le rôle ou le niveau d'accès de l'utilisateur sont transmis par des cookies, un champ « hidden », un paramètre de la requête, par l'URL, … On peut modifier ces champs, il ne faut pas leur faire confiance.
- **sur la géolocalisation** : on peut modifier sa localisation grâce à un VPN, un proxy web, l'en-tête HTTP X-Forwarded-For, il ne faut pas faire confiance à ces données.

Technique d'attaque générique :

- cartographier le site (avec Burp, p.ex.) pour détecter des URLs et les tester à partir d'un compte moins privilégié pour y accéder.

Bonnes pratiques générales :

- ne pas laisser de fonctionnalités non-utilisées
- ne pas faire confiance aux utilisateurs pour utiliser l'app. comme elle a été prévue
- ne pas faire confiance aux données du côté client
- valider les identifiants à chaque transmission de données
- documenter les contrôles d'accès effectués
- prendre les décisions d'autorisation à partir de la session de l'utilisateur
- utiliser un composant central (programme externe, classe, …) pour effectuer les contrôles, + clair, + adaptable, + facilement maintenable
- techniques de programmation pour forcer le développeur à utiliser correctement les contrôles
- pour les parties sensibles de l'app., faire des contrôles supplémentaires (check de l'adresse IP, par exemple)
- pour accéder à des fichiers statiques, passer le nom du fichier à une page dynamique qui va renvoyer une copie du fichier, ou utiliser des fonctionnalités du serveur d'application pour contrôler l'authentification (contrôles consistants avec le composant central)
- réauthentifier l'utilisateur à chaque transaction
- logger toutes les actions et tous les contrôles

Bonnes pratiques – application multi-tiers :

- contrôles à chaque couche de l'app.
- contrôler les URLs selon le rôle de l'utilisateur
- compte de BD séparé pour chaque type d'utilisateur
- compte système avec privilèges limités pour chaque composant

Modèles de contrôles d'accès :

- **techniques de programmation** (matrice de droits dans la BD comportant les rôles et les privilèges de chaque utilisateur et programme qui contrôle automatiquement les droits à accorder aux utilisateurs)
- **avec l'administrateur** (discretionary access control - DAC) qui choisit à qui donner les privilèges (white list / black list)
- **basés sur des rôles** (role based access – RBAC) qui donnent accès à certains privilèges
- **via un composant externe** (compte de BD différent pour les groupes d'utilisateurs afin de limiter leurs droits)