|  |  |
| --- | --- |
| Attacking Application Logic - Cheat-sheetThe Nature of Logic Flaws **Logic flaws** (“faille” (erreur) de logique métier) = de **simples bugs** involontaires lors de la création de l’application à des **vulnérabilités complexes** provenant d’opérations entre des composants de l’application  Peuvent être donc **évident** et **facile à repérer** mais aussi **extrêmement subtile à détecter**, même avec du pentest ou une revue du code rigoureuse -> pas de “**signature**” évidente qui permet de trouver/fixer ces logic flaws.  Ces failles de logique métier proviennent **d’hypothèses du développeur sur l’utilisation ou le fonctionnement interne de l’application** qui, bien qu’il couvre certains scénarios, ne fonctionnent pas sur d’autres -> ouvrent sur des **failles de sécurité** qui peuvent être **conséquentes**  Pour pallier à ça -> **lateral thinking**, ou **penser** “out of the box” / **de manière créative** -> imaginer des scénarios probables qui **sortent d’une utilisation / d’un fonctionnement normal** d’une application (ex: que se passe-t-il si un utilisateur rentre tel paramètre au lieu de ce qui lui ait demandé) Avoiding Logic Flaws Quelques bonnes pratiques :   * Chaque aspect du **design** de l’application **clairement documenté** avec suffisamment de **détails** pour comprendre les hypothèses d’utilisation et de fonctionnement de l’application * Tous le **code source** **commenté clairement** avec les informations suivantes :   + **Utilité** et **fonctionnement** de chaque composant de l’application (variable, fonction, fichier, etc.)   + **Hypothèses d’utilisation / de fonctionnement** de chaque composant concernant tous ce qui est **hors de contrôle** de ce dernier (ex : input de l’utilisateur)   + **Référence** du code client qui utilise des composants * Durant une revue sur le **design** de l’application **d’un point de vue sécurité**, revoir les **hypothèses** posées durant la conception et trouver des **scénarios possibles** qui pourraient **violés** ces hypothèses * Durant une revue du **code**, se focaliser sur la façon dont le code va traiter les **comportements imprévus** des utilisateurs + les **effets indésirés** lors d’appel entre composants et fonctions de l’application   Autres informations relatives aux démos :   * L'utilisateur **contrôle** chaque aspect de chaque requête -> il est impossible de savoir exactement comment chaque utilisateur va **se comporter**, s'ils vont **remplir tous les champs** demandés pour le traitement de la requête, voire **donner des paramètres inattendus**. * Ne pas croire à **l'identité de l'utilisateur** et au **statut de la session** reçu -> vérification **côté serveur** de tous les **paramètres** et de la **session** de l'utilisateur | * Lors de la **mise à jour** de **données en session** par des **input de l'utilisateur** ou des **actions**, avoir conscience des **impacts et effets de bord** qui auront lieu sur d'autres fonctionnalités de l'application * Les **fonctions de recherche** ne doivent pas retourner des **informations sensibles** auprès de l’utilisateur qui permettent de **déduire le contenu** des entrées retournées par la fonction de recherche. Il est même préférable de limiter certaines informations de la recherche selon les droits de l’utilisateur. * Faire attention à l'**implémentation** de fonctionnalités qui permettent à **n'importe quel utilisateur** de **supprimer** du contenu dans les logs + un **admin** qui **crée** un autre admin et que cette action soit **répertoriée dans les logs**. * **Valider** les **inputs** utilisateurs **avant** de les utiliser -> **rejeter** les valeurs **négatives** si elles ne sont pas **attendu** * Si vous implémentez un **rabais** sur une **commande** -> **vérifier** que la commande est **finalisée** **avant** d'appliquer le rabais * **Caractères spéciaux** -> empêcher l'utilisation de tous les caractères spéciaux, même le caractère qui permet cette fonction + attention lors du **troncage** de caractères * Utilisation de **stockage** **approprié** pour maintenir les valeurs durant le processus (**en session**)  Attack with Logic Flaws En temps qu’attaquant, être dans l’esprit d’un développeur qui :   * n’a **pas beaucoup de temps** pour implémenter son application * ne s’occupe pas de **l’aspect de la sécurité** * ajoute **de nouvelles fonctionnalités au code présent** * a utilisé **une API pas assez documenté** * a probablement pris **tous les raccourcis** possibles   Imaginer **les hypothèses d’utilisation / de fonctionnement** de l’application |