2025

Guide OWASP

CYBERSECURITY AGENCY

BRION--FLECK Nicolas

Cybersecurity Agency

09/01/2025

Table des matières

1. Contrôle d'accès défaillant

Contrôle d'accès défaillant se réfère aux failles de sécurité où des utilisateurs non autorisés peuvent accéder à des zones ou à des fonctions de l'application qui devraient être restreintes. Cela peut permettre à des attaquants d'agir en tant qu'utilisateurs légitimes, accéder à des données sensibles, modifier des droits, ou effectuer des actions en tant qu'administrateurs.

Exemples de failles de contrôle d'accès :

1. **Accès non autorisé à des pages d'administration** : Un utilisateur lambda pouvant accéder à des sections de l'application réservées aux administrateurs.
2. **Manipulation de l'URL** : Modification manuelle de l'URL pour accéder à des informations qu'un utilisateur ne devrait pas pouvoir voir.
3. **BYPASS des validations de contrôle d'accès** : Utilisation de requêtes directement sur les API pour accéder à des données sensibles sans passer par les contrôles d'accès habituels.

Pratiques recommandées pour éviter ces failles :

1. **Modèle de sécurité "par défaut interdit"** : Restreindre l'accès par défaut et n'autoriser explicitement que les actions légitimes.
2. **Contrôles d'accès centralisés** : Implémenter les contrôles d'accès dans un seul endroit pour une gestion et une vérification plus facile.
3. **Tests de sécurité et revues de code** : Effectuer régulièrement des tests de sécurité et des revues de code pour identifier et corriger les contrôles d'accès manquants ou faibles.
4. **Principes de moindre privilège** : Donner uniquement les permissions minimales nécessaires à chaque utilisateur pour accomplir ses tâches.

2. Défaillances cryptographiques

Défaillances cryptographiques se réfèrent aux faiblesses dans la mise en œuvre ou l'utilisation des fonctions cryptographiques qui protègent la confidentialité et l'intégrité des données. Cela peut inclure l'utilisation d'algorithmes de chiffrement faibles, une mauvaise gestion des clés, et l'absence de protections pour les données sensibles en transit ou au repos.

Exemples de défaillances cryptographiques :

1. **Utilisation d'algorithmes obsolètes** : Algorithmes tels que MD5 ou SHA-1, qui ne sont plus considérés comme sécurisés.
2. **Absence de chiffrement pour les données sensibles** : Transmission ou stockage de données sensibles en texte clair sans chiffrement adéquat.
3. **Mauvaise gestion des clés** : Clés de chiffrement mal protégées ou stockées de manière inappropriée.
4. **Non-utilisation de HTTPS** : Utilisation de HTTP non sécurisé pour la communication des données sensibles.

Pratiques recommandées pour éviter ces failles :

1. **Utilisation de standards cryptographiques actuels** : Employer des algorithmes de chiffrement considérés comme sûrs, tels que AES pour le chiffrement symétrique et RSA ou ECC pour le chiffrement asymétrique.
2. **Protection des données en transit** : Utiliser HTTPS/TLS pour sécuriser les communications entre les clients et les serveurs.
3. **Gestion sécurisée des clés** : Stocker les clés de chiffrement dans des modules matériels de sécurité (HSM) ou des environnements sécurisés, et les protéger par des mécanismes d'authentification forte.
4. **Cryptage des données sensibles au repos** : Chiffrer les données stockées dans les bases de données, les fichiers de sauvegarde, et autres supports de stockage.
5. **Rotation régulière des clés** : Mettre en œuvre une politique de rotation des clés pour minimiser l'impact potentiel de la compromission d'une clé.

3. *Injection*

Les **failles d'injection** surviennent lorsque des données non fiables sont envoyées à un interpréteur dans le cadre d'une commande ou d'une requête. Ces données peuvent tromper l'interpréteur et l'amener à exécuter des commandes non intentionnelles, accéder à des données sans autorisation ou altérer la structure d'une application.

Exemples de failles d'injection :

1. **Injection SQL (SQLi)** : Un attaquant peut manipuler des requêtes SQL pour exécuter des actions non autorisées, telles que la lecture, la modification ou la suppression de données.
2. **Injection NoSQL** : Similaire à l'injection SQL, mais cible les bases de données NoSQL.
3. **Injection de commandes** : Des commandes arbitraires peuvent être exécutées sur le système d'exploitation en manipulant les entrées des utilisateurs.
4. **Injection LDAP** : Un attaquant manipule des requêtes LDAP pour accéder à des informations ou modifier des enregistrements.
5. **Injection XML (XXE)** : Exploitation d'entrées XML non sécurisées pour exécuter des commandes, lire des fichiers sensibles ou provoquer des dénis de service.

Pratiques recommandées pour éviter les failles d'injection :

1. **Paramétrisation des requêtes** : Utiliser des requêtes paramétrées pour séparer clairement les données des instructions de commande.
2. **Validation et nettoyage des entrées** : Valider et nettoyer rigoureusement toutes les entrées des utilisateurs pour empêcher l'inclusion de données malveillantes.
3. **ORMs (Object-Relational Mappers)** : Utiliser des ORM pour interagir avec les bases de données, ce qui peut réduire le risque d'injections SQL.
4. **Limitation des privilèges de la base de données** : Accorder uniquement les privilèges minimums nécessaires aux comptes utilisés pour accéder aux bases de données.
5. **Détection et blocage des attaques** : Implémenter des systèmes de détection et de blocage des attaques pour identifier et bloquer les tentatives d'injection.

4. Conception non sécurisée

Conception non sécurisée se réfère aux erreurs de conception dans le développement d'une application qui peuvent conduire à des vulnérabilités de sécurité. Ces erreurs sont souvent le résultat d'un manque de considération des aspects de sécurité dès les premières étapes de la conception du logiciel.

Exemples de conceptions non sécurisées :

1. **Absence de contrôles de sécurité** : Ne pas inclure des contrôles de sécurité essentiels comme l'authentification, l'autorisation et les contrôles d'accès.
2. **Gestion inadéquate des sessions** : Ne pas protéger correctement les identifiants de session, permettant ainsi des attaques de détournement de session.
3. **Manque de validation d'entrée** : Ne pas valider ou nettoyer les entrées des utilisateurs peut permettre des attaques par injection.
4. **Incohérences dans les niveaux de privilège** : Attribuer des niveaux de privilège inadéquats ou incohérents, permettant aux utilisateurs non autorisés d'accéder à des fonctionnalités ou des données sensibles.

Pratiques recommandées pour éviter les conceptions non sécurisées :

1. **Adopter une approche de sécurité dès le départ** : Intégrer des contrôles de sécurité dès les premières étapes de la conception et du développement.
2. **Utiliser des cadres de développement sécurisés** : Adopter des frameworks et des bibliothèques connus pour leur sécurité et qui intègrent des pratiques de codage sécurisées.
3. **Effectuer des analyses de sécurité dès la conception** : Réaliser des examens de sécurité et des analyses de risques pendant la phase de conception pour identifier et atténuer les vulnérabilités potentielles.
4. **Modélisation des menaces** : Utiliser des techniques de modélisation des menaces pour anticiper et gérer les risques de sécurité potentiels.
5. **Tests de sécurité continus** : Intégrer des tests de sécurité tout au long du cycle de développement, y compris les tests unitaires, les tests d'intégration et les tests de pénétration.

5. Mauvaise configuration de la sécurité

La **mauvaise configuration de la sécurité** est une cause courante de vulnérabilités dans les applications web. Elle survient lorsque les paramètres de sécurité ne sont pas correctement définis, sont trop permissifs ou manquent de mise à jour, permettant ainsi aux attaquants d'exploiter ces failles.

Exemples de mauvaise configuration de la sécurité :

1. **Comptes par défaut non modifiés** : Utilisation de comptes administratifs avec des mots de passe par défaut ou peu sécurisés.
2. **Paramètres non sécurisés laissés par défaut** : Ne pas désactiver les fonctionnalités inutiles ou les services non sécurisés qui viennent par défaut avec certaines technologies.
3. **Accès excessif** : Attribuer des permissions trop larges aux utilisateurs ou aux services, permettant potentiellement des actions non autorisées.
4. **Absence de correctifs et de mises à jour** : Ne pas appliquer les correctifs de sécurité et les mises à jour logicielles en temps opportun.
5. **Exposition des informations sensibles** : Configurations de l'application qui révèlent des informations sensibles comme les messages d'erreur détaillés ou les configurations système.

Pratiques recommandées pour éviter les mauvaises configurations de sécurité :

1. **Configurer les paramètres par défaut de manière sécurisée** : Réviser et durcir les configurations par défaut pour minimiser les surfaces d'attaque.
2. **Gestion rigoureuse des comptes et des accès** : Modifier les mots de passe par défaut, appliquer le principe du moindre privilège et contrôler régulièrement les accès.
3. **Mise en œuvre de politiques de mise à jour** : Mettre en place des procédures pour appliquer rapidement les correctifs et les mises à jour de sécurité.
4. **Désactivation des fonctionnalités inutiles** : Désactiver ou supprimer les services, ports, comptes et fonctionnalités qui ne sont pas nécessaires.
5. **Masquage des informations sensibles** : Configurer les applications pour minimiser l'exposition des informations sensibles, par exemple en désactivant les messages d'erreur détaillés en production.
6. **Utilisation d'outils de configuration automatisée** : Utiliser des outils d'audit et de gestion de configuration pour identifier et corriger les configurations non sécurisées.

6. Composants vulnérables et obsolètes

Composants vulnérables et obsolètes se réfèrent à l'utilisation de bibliothèques, frameworks, et autres modules logiciels qui contiennent des vulnérabilités connues ou qui ne sont plus maintenus. Cela peut exposer l'application à des attaques exploitant ces failles non corrigées.

Exemples de risques liés aux composants vulnérables et obsolètes :

1. **Bibliothèques avec des failles de sécurité connues** : Utilisation de versions de bibliothèques ayant des vulnérabilités non corrigées.
2. **Modules non mis à jour** : Ne pas appliquer les mises à jour ou les correctifs pour les composants tiers utilisés dans l'application.
3. **Fin de vie des composants** : Utilisation de logiciels ou de composants pour lesquels le support et les mises à jour ont cessé.
4. **Dépendances non gérées** : Ne pas surveiller ou mettre à jour les dépendances et sous-dépendances des composants utilisés.

Pratiques recommandées pour éviter ces risques :

1. **Surveillance des vulnérabilités** : Utiliser des outils de surveillance pour détecter et alerter sur les vulnérabilités connues des composants utilisés.
2. **Gestion des dépendances** : Maintenir un inventaire à jour des composants et de leurs dépendances, et surveiller régulièrement les mises à jour disponibles.
3. **Mises à jour régulières** : Appliquer rapidement les correctifs de sécurité et les mises à jour pour tous les composants utilisés.
4. **Utilisation de composants maintenus** : Préférer l'utilisation de composants activement maintenus et supportés par une communauté ou un fournisseur.
5. **Tests de sécurité** : Effectuer des tests de sécurité réguliers pour identifier les vulnérabilités potentielles dans les composants utilisés.

7. Défaillances d'identification et d'authentification

Les **défaillances d'identification et d'authentification** se produisent lorsque les mécanismes utilisés pour vérifier l'identité des utilisateurs ne sont pas sécurisés ou sont mal implémentés. Cela peut permettre à des attaquants de contourner les processus d'authentification et de se faire passer pour des utilisateurs légitimes.

Exemples de défaillances d'identification et d'authentification :

1. **Mots de passe faibles** : Utilisation de mots de passe faciles à deviner ou réutilisés.
2. **Absence de mécanismes de protection contre les attaques par force brute** : Ne pas limiter le nombre de tentatives de connexion ou ne pas mettre en place des mécanismes de verrouillage de compte.
3. **Stockage de mots de passe non sécurisés** : Conserver les mots de passe en texte clair ou utiliser des algorithmes de hachage obsolètes.
4. **Manque d'authentification multi-facteurs (MFA)** : Ne pas mettre en œuvre l'authentification multi-facteurs pour renforcer la sécurité.
5. **Gestion inadéquate des sessions** : Ne pas protéger correctement les identifiants de session, permettant ainsi le détournement de session.

Pratiques recommandées pour éviter ces failles :

1. **Utilisation de mots de passe forts** : Imposer des politiques de mots de passe forts, incluant des exigences de complexité et de longueur.
2. **Mécanismes de protection contre les attaques par force brute** : Limiter le nombre de tentatives de connexion et mettre en place des mécanismes de verrouillage de compte après plusieurs échecs.
3. **Stockage sécurisé des mots de passe** : Hacher les mots de passe avec des algorithmes sécurisés comme bcrypt, Argon2 ou PBKDF2.
4. **Authentification multi-facteurs (MFA)** : Mettre en place l'authentification multi-facteurs pour ajouter une couche supplémentaire de sécurité.
5. **Sécurisation des sessions** : Protéger les identifiants de session avec des cookies sécurisés, des jetons de session courts et des mécanismes d'expiration de session.

8. Défaillances d'intégrité logicielle et des données

Les **défaillances d'intégrité logicielle et des données** se produisent lorsque les mécanismes qui garantissent l'exactitude et la fiabilité des logiciels et des données ne sont pas en place ou sont insuffisants. Cela peut conduire à des altérations non autorisées des logiciels et des données, compromettant ainsi la sécurité de l'application.

Exemples de défaillances d'intégrité logicielle et des données :

1. **Absence de vérification d'intégrité** : Ne pas vérifier l'intégrité des logiciels ou des données, ce qui permet leur altération par des attaquants.
2. **Manipulation des fichiers** : Les fichiers critiques ou sensibles peuvent être modifiés ou remplacés sans détection.
3. **Mise à jour non sécurisée** : Procédures de mise à jour logicielle qui ne garantissent pas l'authenticité et l'intégrité des mises à jour.
4. **Injection de logiciels malveillants** : Introduction de logiciels malveillants dans le système via des mises à jour ou des composants tiers non vérifiés.

Pratiques recommandées pour éviter ces failles :

1. **Utilisation de hachage et de signatures numériques** : Implémenter des mécanismes de hachage et de signatures numériques pour vérifier l'intégrité et l'authenticité des logiciels et des données.
2. **Contrôles d'accès stricts** : Restreindre les accès aux fichiers et aux systèmes critiques pour éviter les modifications non autorisées.
3. **Procédures de mise à jour sécurisées** : Utiliser des procédures de mise à jour sécurisées qui incluent la vérification de l'intégrité et de l'authenticité des mises à jour logicielles.
4. **Surveillance de l'intégrité** : Mettre en place des systèmes de surveillance pour détecter toute altération des logiciels et des données.
5. **Validation des composants tiers** : Effectuer des vérifications rigoureuses des composants tiers avant leur intégration dans le système.

9. Falsification de requêtes côté serveur (SSRF)

La **falsification de requêtes côté serveur (SSRF)** se produit lorsqu'un attaquant peut forcer un serveur à envoyer des requêtes à un domaine ou une adresse IP de son choix, souvent dans le but de récupérer des informations sensibles ou d'exécuter des actions malveillantes.

Exemples de SSRF :

1. **Accès à des services internes** : Utilisation du serveur pour envoyer des requêtes à d'autres services internes non accessibles directement par l'attaquant.
2. **Scan de ports internes** : Utilisation du serveur pour découvrir des ports ouverts et services en cours d'exécution sur le réseau interne.
3. **Extraction de données sensibles** : Exploitation de l'accès à des services internes pour extraire des informations sensibles, comme des informations d'identification ou des configurations de services.
4. **Exploitation des métadonnées de services cloud** : Accès à des points de terminaison de métadonnées dans des environnements cloud pour obtenir des informations d'identification et d'autres secrets.

Pratiques recommandées pour éviter les attaques SSRF :

1. **Validation des entrées** : Valider et nettoyer toutes les entrées des utilisateurs qui sont utilisées pour former des requêtes, en particulier les URL.
2. **Utilisation de listes blanches** : Mettre en œuvre des listes blanches pour restreindre les domaines et adresses IP accessibles par le serveur.
3. **Séparation des privilèges** : Isoler les services internes et restreindre les privilèges d'accès pour limiter les dégâts potentiels en cas d'attaque.
4. **Filtrage des requêtes sortantes** : Utiliser des pare-feu applicatifs pour filtrer et contrôler les requêtes sortantes du serveur.
5. **Audits de sécurité réguliers** : Effectuer des audits de sécurité et des tests de pénétration pour identifier et corriger les vulnérabilités SSRF potentielles.

10. Journalisations et surveillances insuffisantes

Les **journalisations et surveillances insuffisantes** se produisent lorsqu'une application ne collecte pas, ne stocke pas ou ne surveille pas adéquatement les logs des événements de sécurité. Cela peut rendre difficile la détection, la réponse et l'investigation des incidents de sécurité.

Exemples de journalisation et surveillance insuffisantes :

1. **Absence de logs d'accès** : Ne pas enregistrer les tentatives de connexion réussies et échouées.
2. **Logs incomplets ou inadéquats** : Les logs qui ne contiennent pas suffisamment d'informations pour être utiles lors de l'investigation d'un incident de sécurité.
3. **Non surveillance des événements critiques** : Ne pas surveiller les événements critiques comme les changements de configuration, les élévations de privilège, ou les accès à des données sensibles.
4. **Absence de centralisation des logs** : Ne pas centraliser les logs, ce qui rend leur analyse difficile et morcelée.
5. **Non mise en place de systèmes de détection d'intrusion (IDS)** : Absence de systèmes pour surveiller et alerter sur des comportements anormaux ou malveillants.

Pratiques recommandées pour améliorer la journalisation et la surveillance :

1. **Implémentation de la journalisation détaillée** : Enregistrer des logs détaillés pour les tentatives de connexion, les changements de configuration, les accès aux données sensibles, etc.
2. **Centralisation des logs** : Utiliser une solution de gestion centralisée des logs pour collecter et analyser les logs à partir de différentes sources.
3. **Surveillance continue des événements** : Mettre en place des systèmes de surveillance continue pour détecter et alerter sur les événements de sécurité critiques.
4. **Utilisation de solutions SIEM (Security Information and Event Management)** : Adopter des solutions SIEM pour corréler, analyser et alerter sur les événements de sécurité en temps réel.
5. **Revue régulière des logs** : Effectuer des revues régulières des logs pour identifier les comportements anormaux et les incidents potentiels.
6. **Formation et sensibilisation** : Former le personnel à l'importance de la journalisation et de la surveillance, et à l'utilisation des outils de gestion des logs.