Réseaux avancés et sécurité

Sidi Biha

SupNum

2023-2024

Plan

OWASP

OWASP

Open Web Application Security Project est une organisation internationale à but non lucratif dédiée à la sécurité des applications Web.

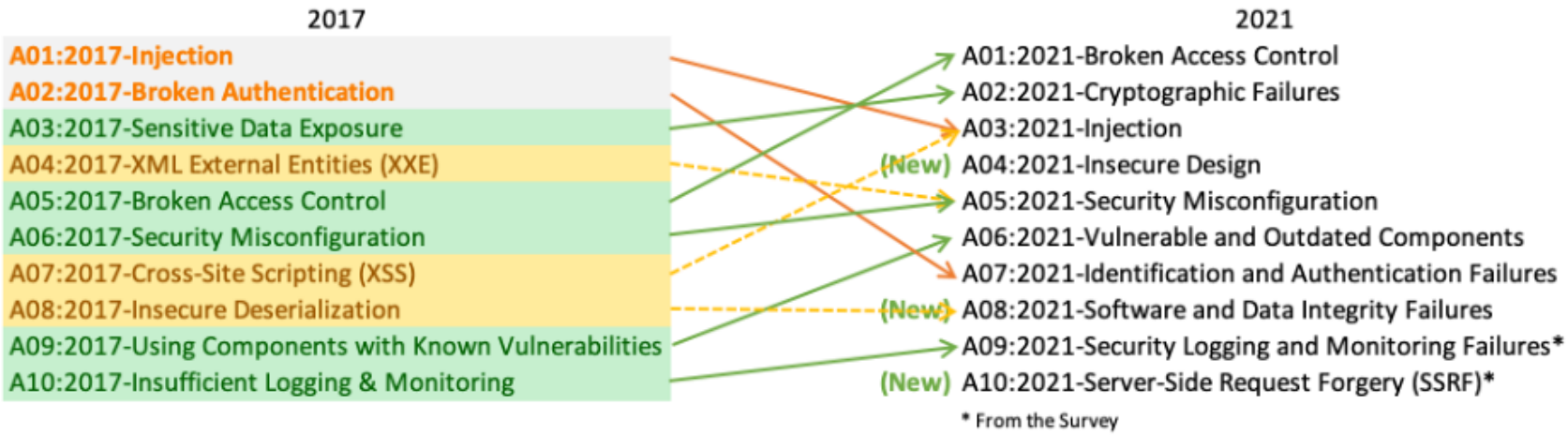
OWASP Top 10

Un rapport régulièrement mis à jour décrivant les problèmes de sécurité des applications (Web), en se concentrant sur les 10 risques les plus critiques.

OWASP Top 10 : objectif

Sensibiliser sur les problèmes de sécurité des applications (Web) afin que les organisations puissent mettre en œuvre des programmes et des pratiques efficaces pour réduire les risques de sécurité.

Nouvelle version 2021



- Trois nouvelles entrées :
  - Conception non sécurisée: accent sur les défauts de conception.
  - Manque d'intégrité des données et du logiciel : formulation d'hypothèses sur les mises à jour logicielles, les données critiques et les pipelines CI/CD.
  - Falsification de requête côté serveur : se produisent chaque fois qu'une application Web récupère une ressource distante sans valider l'URL fourni par l'utilisateur.
- Référence: **https://owasp.org/Top10/fr/**

1 : Violation de contrôle d'accès

Définition

Exploitation du manque ou de la mauvaise configuration des restrictions sur les droits des utilisateurs authentifiés pour accéder à des fonctionnalités et/ou des données non autorisées.

- Une application est vulnérable quand :
  - Les contrôles d'accès sont contournable par simple modification de l'URL
  - Permet la modification de la clef d'identification pour pointer sur l'enregistrement d'un autre utilisateur, donnant la possibilité de voir ou modifier le compte de quelqu'un d'autre.
  - Permet une élévation de privilège, permettre à un simple utilisateur d'agir comme un administrateur.
  - Permet la navigation vers des pages soumises à authentification sans être authentifié ou par manipulation de CORS.
  - Permet l'accès à des API sans contrôle avec des méthodes POST, PUT ou DELETE.

1 : Violation de contrôle d'accès

- Protections :
  - A l'exception des ressources publiques, tout doit être bloqué par défaut.
  - Centraliser l'implémentation des mécanismes de contrôle d'accès et les réutiliser dans l'ensemble de l'application.
  - Le modèle de contrôle d'accès doit vérifier l'appartenance des enregistrements, plutôt que de permettre à l'utilisateur de créer, lire, modifier ou supprimer n'importe quel enregistrement.
  - Désactiver le listing de dossier sur le serveur web, et vérifier que les fichiers de meta-données (ex : .git) et de sauvegardes ne se trouvent pas dans l'arborescence web.
  - Tracer les échecs de contrôles d'accès, les alertes administrateur quand c'est approprié (ex : échecs répétés).
  - Limiter la fréquence d'accès aux API et aux contrôleurs d'accès, afin de réduire l'efficacité des outils d'attaques automatisés.
  - Les jetons de sessions doivent être invalidés côté serveur après une déconnexion.
  - Les développeurs et les testeurs qualités doivent procéder à des tests unitaires et d'intégration sur les fonctionnalités de contrôle d'accès.

2 : Défaillances cryptographiques

Définition

Faible ou absence de protections des données sensibles telles que les données bancaires, les données relatives aux soins de santé, les données personnelles d'identification.

- Une application est vulnérable quand :
  - Des données sensibles/critiques sont transmises ou sauvegardées en clair.
  - Des algorithmes de cryptage faibles ou désuets sont-ils utilisés.
  - Des clefs de chiffrement par défaut ou faibles sont utilisées.
  - Absence des entêtes/directives de sécurité pour assurer qu'un bon chiffrement des données en transites est toujours utilisé.
  - Le client ne vérifie pas que le certificat envoyé par le serveur est valide.

2 : Défaillances cryptographiques

- Protections :
  - Classifier les données traitées, stockées ou transmises par l'application. Identifier quelles données sont sensibles.
  - Appliquer des contrôles selon la classification.
  - Ne pas stocker de données sensibles sans que cela ne soit nécessaire.
  - S'assurer de chiffrer toutes les données sensibles au repos.
  - Choisir des algorithmes prouvés et générer des clés robustes.
  - Chiffrer toutes les données transmises avec des protocoles sécurisés tels que TLS avec des chiffres à confidentialité persistante (perfect forward secrecy). Chiffrer en priorité sur le serveur. Utiliser des paramètres sécurisés. Forcer le chiffrement en utilisant des directives comme HTTP Strict Transport Security (HSTS).
  - Stocker les mots de passe au moyen de puissantes fonctions de hachage adaptatives.

3 : Injection

Définition

Une faille de type injection, comme une injection SQL, une injection de commande système, ou une injection LDAP, se produit quand une donnée non fiable est envoyée à une application comme argument d'une commande ou d'une requête.

- Une application est vulnérable quand :
  - les données venant de l'utilisateur ne sont pas validées, filtrées ou nettoyées par l'application ;
  - des requêtes dynamiques ou des appels non paramétrés sans échappement par rapport au contexte sont envoyés à l'interpréteur ;
  - des données hostiles sont utilisées au sein de paramètres de recherche de mapping objet - relationnel (ORM) pour extraire des données supplémentaires sensibles ;
  - des données hostiles sont utilisées directement ou concaténées, par exemple lors de la construction de requêtes dynamiques, de commandes ou de procédures stockées pour des requêtes SQL ou des commandes OS.

3 : Injection

- Protections :
  - Utiliser une API saine qui évite complètement l'utilisation de l'interpréteur ou fournit une interface paramétrable.
  - Utiliser le "whitelist" avec normalisation des données en entrée, ceci n'est pas toujours une défense possible dans la mesure où de nombreuses applications requièrent des caractères spéciaux.
  - Echapper soigneusement les caractères spéciaux dans les requêtes dynamiques, en utilisant la syntaxe d'échappement correspondant à votre langage de programmation.
  - Utiliser LIMIT et autres contrôles SQL à l'intérieur des requêtes pour empêcher les divulgations massives de données dans le cas d'injection SQL.

4 : Conception non sécurisée

Définition

La conception non sécurisée est une vaste catégorie représentant différentes faiblesses, exprimées en tant que "conception de sécurité manquante ou inefficace".

- L'un des facteurs qui contribuent à une conception non sécurisée est le manque de profilage des risques inhérent au logiciel/système en cours de développement, et donc l'incapacité à déterminer le niveau de conception de sécurité requis.
- Un logiciel sécurisé nécessite un cycle de vie de développement sécurisé, une certaine forme de modèle de conception sécurisé, des bibliothèques de composants sécurisés, des outils et une modélisation des risques.

4 : Conception non sécurisée

- Protections :
  - Établir et utiliser un cycle de vie de développement sécurisé avec des professionnels AppSec pour aider à évaluer et à concevoir des contrôles liés à la sécurité et à la confidentialité.
  - Établir et utiliser des pattern de conception sécurisés ou de composants prêts à l'emploi.
  - Utiliser la modélisation des menaces pour l'authentification critique, le contrôle d'accès, la logique métier et les flux clés.
  - Intégrer le langage et les contrôles de sécurité dans la spécification.
  - Rédiger des tests unitaires et d'intégration pour valider que tous les flux critiques résistent au modèle de menace.
  - Séparer les couches système et réseau en fonction des besoins d'exposition et de protection.
  - Limiter la consommation de ressources par utilisateur ou service.
  - Séparer solidement les tenants à tous les niveaux (Cloud).

5 : Mauvaise configuration Sécurité

Définition

Résultat de configurations par défaut non sécurisées, de configurations incomplètes ou ad hoc, d'un stockage dans un cloud ouvert, d'en-têtes HTTP mal configurés et de messages d'erreur verbeux contenant des informations sensibles.

- L'application peut être vulnérable si :
  - L'application n'a pas fait l'objet d'un hardening sécurité approprié.
  - Des permissions sont mal configurées sur un service cloud.
  - Des fonctionnalités inutiles sont activées ou installées (ex : des ports, des services, des pages, des comptes ...).
  - Des comptes par défaut et leurs mots de passe sont toujours activés et inchangés.
  - Les messages d'erreurs révèle aux utilisateurs trop d'informations.
  - Des fonctionnalités de sécurité sont désactivées ou ne sont pas configurées proprement.
  - Le serveur n'envoie pas d'en-têtes ou de directives de sécurité.
  - La version de l'application/système est obsolète ou vulnérable.

5 : Mauvaise configuration Sécurité

- Protections :
  - Un processus de durcissement répétable qui permet de déployer rapidement et facilement un autre environnement correctement sécurisé avec une configuration verrouillée.
  - Les environnements de développement, d'assurance qualité et de production doivent tous être configurés de manière identique, avec des droits différents pour chaque environnement.
  - Une plate-forme minimale sans fonctionnalité, composant, documentation et échantillon inutile.
  - Une tâche pour revoir et mettre à jour les configurations appropriées à tous les avis de sécurité, toutes les mises à jour et tous les correctifs.
  - Une architecture d'application segmentée qui fournit une séparation efficace et sécurisée entre les composants ou les environnement hébergés.
  - L'envoi de directives de sécurité aux clients.
  - Un processus automatisé pour vérifier les configurations et les réglages dans tous les environnements.

6 : Utilisation de composants vulnérables

Définition

Les composants, tels que les bibliothèques, frameworks et autres modules logiciels, fonctionnent avec les mêmes privilèges que l'application. Si un composant vulnérable est exploité par une attaque, cela peut entraîner de graves pertes de données ou la compromission du serveur. Les applications et les API utilisant des composants dont les vulnérabilités sont connues peuvent compromettre leurs défenses et permettre diverses attaques.

- L'application peut être vulnérable si :
  - Le logiciel utilisé est vulnérable, sans support, ou obsolète. Exemples: OS, le serveur web/app, SGBD, API, libraires ...
  - Absence de recherche régulières de vulnérabilités et de souscription aux bulletins de sécurité.
  - Frameworks, et leurs dépendances ne sont pas mis à jour dans un délai convenable.
  - Les développeurs de logiciels ne testent pas la compatibilité des évolutions, des mises à jour et des correctifs des bibliothèques.
  - Les configurations des composants ne sont pas sécurisées.

6 : Utilisation de composants vulnérables

- Protections :
  - Supprimer les dépendances inutiles et les fonctionnalités, composants, fichiers et documentation non nécessaires.
  - Faire un inventaire en continu des versions de composants à la fois client et serveur (ex : frameworks, bibliothèques) et de leurs dépendances.
  - Surveiller en permanence les sources comme CVE et NVD pour suivre les vulnérabilités des composants.
  - Ne récupérer des composants qu'auprès de sources officielles via des liens sécurisés.
  - Surveiller les bibliothèques et les composants qui ne sont plus maintenus ou pour lesquels il n'y a plus de correctifs de sécurité.