OWASP API Security Top 10 2019

Les dix failles de sécurité d'API les plus critiques

TOC

Table des matières

Table of Contents

À propos d'OWASP

L'Open Web Application Security Project (OWASP) est une communauté ouverte dédiée à permettre aux organisations de développer, d'acheter et de maintenir des applications et des API auxquelles on peut faire confiance.

Chez OWASP, vous trouverez en gratuit et libre :

- Outils et standards pour la sécurité des applications.
- Livres complets sur les tests de sécurité des applications, le développement sécurisé de code, et la revue sécurisée de code.
- Présentations et vidéos.
- Cheat sheets sur de nombreux sujets.
- Contrôles et bibliothèques de sécurité standards.
- Communautés locales partout dans le monde.
- Recherche de pointe.
- Nombreuses conférences partout dans le monde.
- Listes de diffusion.

Plus d'information sur : https://www.owasp.org.

Tous les outils, documents, vidéos, présentations et chapitres sont libres et ouverts à quiconque s'intéressant à l'amélioration de la sécurité des applications.

Nous préconisons d'approcher la sécurité applicative comme un problème de personnes, de procédés et de technologie, parce que les approches les plus efficaces de sécurité des applications requièrent des améliorations dans ces directions.

OWASP est une nouvelle sorte d'organisation. Notre liberté face aux pressions commerciales nous permet de fournir des informations impartiales, pratiques et efficientes à propos de la sécurité applicative.

OWASP n'est affiliée avec aucune entreprise de technologie, bien que soutenions l'usage raisonné de technologies de sécurité commerciales. OWASP produit toutes sortes d'éléments de manière collaborative, transparente et ouverte.

La Fondation OWASP est l'entité à but non-lucratif qui garantit le succès à long terme du projet. La quasi-totalité des personnes associée à OWASP est bénévole, y compris le conseil d'administration d'OWASP, les responsables des chapitres et les membres des projets. Nous soutenons la recherche en sécurité innovante avec des bourses et de l'infrastructure.

Rejoignez-nous !

FW

Avant-propos

L'Application Programming Interface (API), en français interface de programmation applicative, est un élément fondateur des applications omniprésentes dans le monde actuel. Des banques, commerces, du transport à l'IoT, aux véhicules autonomes et aux villes intelligentes, les API forment une partie critique des applications mobiles, Saas et web modernes que l'on trouve dans des applications destinées aux consommateurs, aux partenaires ou aux usages internes.

Par nature, les API exposent la logique applicative et des données sensibles telles que des données personnelles, et de ce fait les API sont devenues une cible pour des attaquants. Sans des API sécurisées, l'innovation rapide serait impossible.

Bien qu'un Top 10 plus large sur les risques de sécurité des applications web fasse toujours sens, du fait de leur nature particulière, une liste des risques spécifiques aux API est nécessaire. La sécurité des API se concentre sur des stratégies et des solutions pour comprendre et corriger les vulnérabilités et risques de sécurité uniques aux API.

Si vous êtes familiarisés avec le projet OWASP Top 10, vous aurez alors remarqué des similarités entre les deux documents : ils sont conçus pour être lisibles et adoptés. Si vous découvrez la série OWASP Top 10, il vaudrait peut-être mieux commencer par lire les sections Risques de sécurité des API et Méthodologie et données avant de vous plonger dans la liste du Top 10.

Vous pouvez contribuer à l'OWASP API Security Top 10 avec vos questions, commentaires et idées sur notre dépôt GitHub du projet :

- https://github.com/OWASP/API-Security/issues
- https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md

Vous trouverez l'OWASP API Security Top 10 ici :

- https://www.owasp.org/index.php/OWASP_API_Security_Project
- https://github.com/OWASP/API-Security

Nous voulons remercier tous les contributeurs qui ont rendu ce projet possible grâce à leurs efforts et leurs contributions. Ils sont tous listés dans la section des remerciements. Merci !

I Introduction

Bienvenue à l'OWASP API Security Top 10 - 2019 !

Bienvenue à la première édition du projet OWASP API Security Top 10. Si vous connaissez les séries OWASP Top 10, vous remarquerez les similarités : elles sont voulues pour faciliter en la lisibilité l'adoption. Si ce n'est pas votre cas, vous pouvez commencer par consulter la page wiki de l'OWASP API Security Project, avant d'approfondir avec les risques de sécurité les plus critiques des API.

Les API jouent un rôle très important dans l'architecture des applications modernes. Dans la mesure où la création d'une prise de conscience de la sécurité et l'innovation ont des rythmes différents, il est important de se concentrer sur les vulnérabilités de sécurité courantes des API.

L'objectif principal du projet OWASP API Security Top 10 est d'éduquer ceux qui sont impliqués dans le développement et la maintenance des API, notamment : développeurs, designers, architectes, managers, ainsi que les organisations.

Dans la section méthodologie et Données, vous pouvez en lire davantage sur la manière dont cette première édition a été créée. Pour les prochaines versions, nous voulons impliquer l'industrie de la sécurité, avec un appel public à informations. Pour le moment, nous encourageons chacun à contribuer avec des questions, des commentaires ou des idées sur notre dépôt GitHub ou notre liste de diffusion.

RN Notes de version

Ceci est la première édition de l'OWASP API Security Top 10, que nous prévoyons de mettre à jour périodiquement, tous les trois ou quatre ans.

Contrairement à cette version, nous voulons pour les futures versions lancer un appel public à données, impliquant le secteur de la sécurité dans cette démarche. Dans la section Méthodologie et Données, vous trouverez plus d'informations sur la manière dont cette version a été construite. Pour plus de détails sur les riques de sécurité, veuillez vous référer à la section Risques de Sécurité des API.

Il est important d'être conscient que l'architecture des applications a considérablement changé au cours des dernières années. Actuellement les API jouent un rôle très important dans cette nouvelle architecture de microservices, de *single page applications*, d'applis mobiles, d'IoT, etc.

L'OWASP API Security Top 10 était un effort nécessaire pour créer une prise de conscience des questions de sécurité des API modernes. Cela n'a été possible que grâce aux efforts considérables de plusieurs volontaires, qui sont tous répertoriés dans la section Remerciements. Merci !

RISK API Security Risk

La méthodologie d'évaluation de risques OWASP a été utilisée pour effectuer l'analyse de risques.

La table ci-dessous résume la terminologie associée au niveau de risque.

Facteurs de menace	Exploitabilité	Prévalence de la faille	Détectabilité de la faille	Impact technique	Impact organisationnel
Spécifique API	Facile: **3**	Répandu **3**	Facile **3**	Sévère **3**	Spécifique à l'organisation
	Moyen: **2**	Commune **2**	Moyenne **2**	Modéré **2**	
	Difficile: **1**	Difficile **1**	Difficile **1**	Mineure **1**	

Note: Cette approche ne prend pas en compte la probabilité du facteur de menace. Elle ne prend pas non plus en compte les différents détails techniques spécifiques à votre application. Ces facteurs pourraient modifier de manière significative la probabilité globale qu'un attaquant trouve et exploite une vulnérabilité particulière. Cette évaluation ne prend pas en compte l'impact réel sur votre activité. Votre organisation devra décider quel niveau de risque de sécurité elle est prête à accepter pour vos applications et vos API en fonction de votre culture, votre secteur d'activité et votre environnement réglementaire. L'objet du projet OWASP API Security Top 10 n'est pas d'effectuer cette analyse de risques à votre place.

Références

OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

Externes

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modeling Tool

T10 OWASP API Security Top 10 - 2019

API1:2019 - Broken Object Level Authorization	Les API ont tendance à exposer des points d'accès (endpoints) qui gèrent les identifiants d'objets (OID), créant une large surface d'attaque des contrôles de niveaux d'accès. Des contrôles d'autorisation doivent être effectués pour toute fonction qui accède à une source de données à partir d'entrées fournies par un utilisateur.
API2:2019 - Broken User Authentication	Les mécanismes d'authentification sont souvent implémentés incorrectement, permettant à des attaquants de compromettre des jetons (Token) d'authentification ou d'exploiter des failles d'implémentation afin d'endosser temporairement ou de manière permanente l'identité d'autres utilisateurs. L'incapacité, pour un système, à identifier le client / utilisateur de manière fiable compromet la sécurité de l'API dans son ensemble.
API3:2019 - Excessive Data Exposure	En effectuant des implémentations génériques rapides, les développeurs ont tendance à exposer tous les attributs des objets sans prendre en considération leur confidentialité individuelle, se reposant sur les clients d'API pour effectuer un filtrage des données avant présentation à l'utilisateur.
API4:2019 - Lack of Resources & Rate Limiting	Bien souvent, les API n'imposent pas de restrictions sur la taille ou le nombre de ressources qui peuvent être requises par le client / l'utilisateur. Ceci impacte non seulement la performance du serveur d'API, aboutissant à un Déni de Service (DoS), mais cela constitue aussi une porte ouverte pour des failles d'authentification par force brute.
API5:2019 - Broken Function Level Authorization	Les politiques de contrôle d'accès complexes avec des hiérarchies, groupes et rôles différents, et la séparation non claire entre les fonctions normales et d'administration tendent à occasionner des failles d'authentification. En exploitant ces problèmes, les attaquants parviennent à accéder aux ressources d'autres utilisateurs et / ou aux fonctions d'administration.
API6:2019 - Mass Assignment	La connexion de données fournies par le client (ex : JSON) aux modèles de données sans filtrage approprié par une liste de validation conduit généralement à l'assignation massive. En devinant les attributs des objets, en explorant les autres points d'accès de l'API, en lisant la documentation ou en incluant des attributs supplémentaires dans la charge utile (payload) des requêtes, les attaquants parviennent à modifier les attributs d'objets qu'ils ne devraient pas pouvoir modifier.
API7:2019 - Security Misconfiguration	Une mauvaise configuration de sécurité est souvent le résultat de configurations par défauts non sécurisées, de configurations incomplètes ou ad-hoc, de stockages ouverts dans le cloud, d'en-têtes HTTP mal configurés, de méthodes HTTP non nécessaires, de partage de ressources intersites permissives (CORS), et de messages d'erreurs verbeux contenant des informations sensibles.
API8:2019 - Injection	Les failles d'injection telles que SQL, NoSQL, injection de commandes, etc, se produisent lorsque des données non fiables sont transmises à un interpréteur comme partie d'une commande ou d'une requête. Les données malveillantes de l'attaquant peuvent tromper l'interpréteur et lui faire exécuter des commandes non prévues ou accéder à des données sans disposer des autorisations nécessaires.
API9:2019 - Improper Assets Management	Les API ont tendance à exposer plus de points d'accès que les applications web traditionnelles, rendant très importante la nécessité d'une documentation appropriée et actualisée. Un inventaire précis des hôtes et des versions d'API déployées joue également un rôle important pour prévenir les problèmes tels que les versions obsolètes d'API et les points de déboggage vulnérables.
API10:2019 - Insufficient Logging & Monitoring	L'insuffisance de logging et de monitoring, couplée à une intégration insuffisante ou absente à la réponse aux incidents, permet aux attaquants de poursuivre leurs attaques sur les systèmes, de s'y maintenir de manière persistante, et de rebondir sur d'autres systèmes pour compromettre, extraire ou détruire des données. La plupart des études de failles montrent que le temps nécessaire pour détecter une intrusion est supérieur à 200 jours, typiquement découverte par des tiers externes plutôt que par des processus ou un monitoring interne.

API1:2019 Broken Object Level Authorization

Spécifique API	**Exploitabilité: 3**	**Prévalence: 3**	**Détectabilité: 2**	**Technique: 3**	**Spécifique à l'organisation**
Des attaquants peuvent exploiter des points d'accès d'API qui sont vulnérables à la faille de niveau d'autorisation en manipulant l'ID d'un objet qui est envoyé avec la requête. Ceci peut entrainer à un accès non autorisé à des données sensibles. Ce problème est extrêmement commun dans les applications basées sur des API parce que le composant serveur ne suit pas complètement l'état du client, et au lieu de cela, s'appuie davantage sur des paramètres comme les ID objets, qui sont envoyés par le client, pour déterminer à quel objet accéder.		Cette attaque est la plus courante et la plus impactante sur les APIs. Les mécanismes d'autorisation et de contrôle d'accès des applications modernes sont complexes et étendus. Même si l'application implémente une infrastructure adaptée pour les contrôles d'autorisations, les développeurs peuvent oublier d'utiliser ces contrôles avant d'autoriser l'accès à un objet sensible. La détection de contrôles d'accès ne se prête typiquement pas à des tests statiques ou dynamiques automatisés.		Un accès non autorisé peut aboutir à la diffusion de données à des tiers non autorisés, des pertes de données, ou des manipulations de données. Un accès non autorisé aux objets peut aussi aboutir à une prise de contrôle complète d'un compte.	

L'API est-elle vulnérable ?

L'autorisation au niveau de l'objet est un mécanisme de contrôle d'accès qui est généralement implémenté au niveau du code pour valider qu'un utilisateur puisse uniquement accéder aux objets auxquels il doit avoir accès.

Chaque point d'accès d'API qui reçoit l'ID d'un objet, et effectue une action quelconque sur l' objet, doit implémenter des contrôles d'accès au niveau de l'objet. Les contrôles doivent valider que l'utilisateur connecté dispose de l'accès pour effectuer l'action requise sur l'objet requis.

Des vulnérabilités de ce mécanisme entraînent typiquement des diffusions non autorisées d'information, la modification ou la destruction de toutes les données.

Exemples de scénarios d'attaque

Scénario #1

Une plateforme de commerce électronique pour des magasins en ligne (boutiques) comporte une page listant les diagrammes de vente pour les boutiques hébergées. Examinant les requêtes du navigateur, un attaquant peut identifier les points d'accès de l'API utilisés comme sources de données pour ces diagrammes et leur schéma `/shops/{shopName}/revenue_data.json`. Utilisant un autre point d'accès de l'API, l'attaquant peut obtenir la liste de tous les noms des boutiques hébergées. Avec un simple script pour manipuler les noms de la liste, remplaçant `{shopName}` dans l'URL, l'attaquant obtient l'accès aux données de vente de milliers de boutiques de commerce électronique.

API1:2019 Broken Object Level Authorization

Scénario #2

Observant le trafic réseau d'un appareil de poche, la requête HTTP `PATCH` suivante attire l'attention d'un attaquant du fait de la présence d'un en-tête HTTP personnalisé `X-User-Id: 54796`. Remplaçant la valeur `X-User-Id` par `54795`, l'attaquant obtient une réponse HTTP valide, et est en mesure de modifier les données des autres comptes utilisateurs.

Comment s'en prémunir

- Implémentez un véritable mécanisme d'autorisation qui s'appuie sur des droits utilisateurs et sur une hiérarchie.
- Utilisez un mécanisme d'autorisation pour vérifier si l'utilisateur connecté est autorisé à effectuer l'action requise sur l'enregistrement pour toute fonction qui utilise une entrée du client pour accéder à un enregistrement dans la base de données.
- Préférez l'utilisation de valeurs aléatoires et non prévisibles comme GUIDs pour les ID des enregistrements.
- Écrivez des tests pour évaluer les mécanismes d'autorisation. Ne déployez pas des modifications vulnérables qui ne passent pas les tests.

Références

Externes

- CWE-284: Improper Access Control
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key

API2:2019 Broken User Authentication

Spécifique API	**Exploitabilité: 3**	**Prévalence: 2**	**Détectabilité: 2**	**Technique: 3**	**Spécifique à l'organisation**
Des attaquants peuvent exploiter des points d'accès d'API qui sont vulnérables à la faille de niveau d'autorisation en manipulant l'ID d'un objet qui est envoyé avec la requête. Ceci peut entrainer à un accès non autorisé à des données sensibles. Ce problème est extrêmement commun dans les applications basées sur des API parce que le composant serveur ne suit pas complètement l'état du client, et au lieu de cela, s'appuie davantage sur des paramètres comme les ID objets, qui sont envoyés par le client, pour déterminer à quel objet accéder.		Cette attaque est la plus courante et la plus impactante sur les APIs. Les mécanismes d'autorisation et de contrôle d'accès des applications modernes sont complexes et étendus. Même si l'application implémente une infrastructure adaptée pour les contrôles d'autorisations, les développeurs peuvent oublier d'utiliser ces contrôles avant d'autoriser l'accès à un objet sensible. La détection de contrôles d'accès ne se prête typiquement pas à des tests statiques ou dynamiques automatisés.		Un accès non autorisé peut aboutir à la diffusion de données à des tiers non autorisés, des pertes de données, ou des manipulations de données. Un accès non autorisé aux objets peut aussi aboutir à une prise de contrôle complète d'un compte.	

L'API est-elle vulnérable ?

L'autorisation au niveau de l'objet est un mécanisme de contrôle d'accès qui est généralement implémenté au niveau du code pour valider qu'un utilisateur puisse uniquement accéder aux objets auxquels il doit avoir accès.

Chaque point d'accès d'API qui reçoit l'ID d'un objet, et effectue une action quelconque sur l' objet, doit implémenter des contrôles d'accès au niveau de l'objet. Les contrôles doivent valider que l'utilisateur connecté dispose de l'accès pour effectuer l'action requise sur l'objet requis.

Des vulnérabilités de ce mécanisme entraînent typiquement des diffusions non autorisées d'information, la modification ou la destruction de toutes les données.

Exemples de scénarios d'attaque

Scénario #1

Une plateforme de commerce électronique pour des magasins en ligne (boutiques) comporte une page listant les diagrammes de vente pour les boutiques hébergées. Examinant les requêtes du navigateur, un attaquant peut identifier les points d'accès de l'API utilisés comme sources de données pour ces diagrammes et leur schéma `/shops/{shopName}/revenue_data.json`. Utilisant un autre point d'accès de l'API, l'attaquant peut obtenir la liste de tous les noms des boutiques hébergées. Avec un simple script pour manipuler les noms de la liste, remplaçant `{shopName}` dans l'URL, l'attaquant obtient l'accès aux données de vente de milliers de boutiques de commerce électronique.

API2:2019 Broken User Authentication

Scénario #2

Observant le trafic réseau d'un appareil de poche, la requête HTTP `PATCH` suivante attire l'attention d'un attaquant du fait de la présence d'un en-tête HTTP personnalisé `X-User-Id: 54796`. Remplaçant la valeur `X-User-Id` par `54795`, l'attaquant obtient une réponse HTTP valide, et est en mesure de modifier les données des autres comptes utilisateurs.

Comment s'en prémunir

- Implémentez un véritable mécanisme d'autorisation qui s'appuie sur des droits utilisateurs et sur une hiérarchie.
- Utilisez un mécanisme d'autorisation pour vérifier si l'utilisateur connecté est autorisé à effectuer l'action requise sur l'enregistrement pour toute fonction qui utilise une entrée du client pour accéder à un enregistrement dans la base de données.
- Préférez l'utilisation de valeurs aléatoires et non prévisibles comme GUIDs pour les ID des enregistrements.
- Écrivez des tests pour évaluer les mécanismes d'autorisation. Ne déployez pas des modifications vulnérables qui ne passent pas les tests.

Références

Externes

- CWE-284: Improper Access Control
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key

API3:2019 Excessive Data Exposure

Spécifique API	**Exploitabilité: 3**	**Prévalence: 2**	**Détectabilité: 2**	**Technique: 2**	**Spécifique à l'organisation**
L'exploitation d'exposition excessive de données est simple, et est généralement effectuée en écoutant le trafic pour analyser les réponses de l'API, à la recherche de données sensibles qui ne devraient pas être retournées à l'utilisateur.		Les API comptent sur les clients pour effectuer le filtrage des données. Comme les API sont utilisées comme sources de données, les développeurs les implémentent parfois de manière générique sans penser au caractère sensible des données diffusées. En général les outils automatiques ne permettent pas de détecter ce type de vulnérabilité car il est difficile de faire la différence entre les données légitimement retournées par l'API, et des données sensibles qui ne devraient pas être retournées sans une compréhension en profondeur de l'application.		L'exposition excessive de données conduit généralement à la diffusion de données sensibles.	

L'API est-elle vulnérable ?

Par conception, l'API retourne des données sensibles au client. De plus, ces données sont généralement filtrées côté client avant d'être présentées à l'utilisateur. Un attaquant peut facilement écouter le trafic et voir les données sensibles.

Exemples de scénarios d'attaque

Scénario #1

L'équipe mobile utilise le point d'accès `/api/articles/{articleId}/comments/{commentId}` dans la vue des articles pour le rendu des métadonnées des commentaires. Ecoutant le trafic de l'application mobile, un attaquant découvre que d'autres données sensibles relatives à l'auteur du commentaire sont également retournées. L'implémentation du point d'accès utilise une méthode générique `toJSON()` sur le modèle `User`, qui contient des données personnelles, pour sérialiser l'objet.

Scénario #2

Un système de surveillance à base d'IoT permet aux administrateurs de créer des utilisateurs avec différentes permissions. Un administrateur a créé un compte utilisateur pour un nouvel agent de sécurité qui ne devrait avoir accès qu'à certains bâtiments spécifiques sur le site. Quand l'agent de sécurité utilise son appli mobile, un appel d'API est effectué vers `/api/sites/111/cameras` pour recevoir des données à propos des caméras disponibles et les montrer sur un tableau de bord. La réponse contient une liste avec des informations sur les caméras au format suivant : `{"id":"xxx","live_access_token":"xxxx-bbbbb","building_id":"yyy"}`. Si l'interface graphique du client montre uniquement les caméras auxquelles l'agent de sécurité doit avoir accès, la réponse de l'API contient en réalité la liste complète de toutes les caméras présentes sur le site.

API3:2019 Excessive Data Exposure

Comment s'en prémunir

- Ne comptez jamais sur le client d'API pour filtrer des données sensibles.
- Passez en revue les réponses de l'API pour vous assurer qu'elles contiennent uniquement des données nécessaires.
- Les ingénieurs backend devraient toujours se poser la question "qui est le consommateur des données ?" avant d'exposer un nouveau point d'accès d'API.
- Évitez les méthodes génériques telles que `to_json()` ou `to_string()`. Au lieu de cela, choisissez les éléments précis que vous voulez vraiment retourner.
- Classifiez les données sensibles et personnelles que votre application stocke et manipule, et passez en revue tous les appels d'API qui retournent de telles données pour voir si les réponses posent des problèmes de sécurité.
- Implémentez un mécanisme de réponse basé sur un schéma de validation afin d'ajouter un niveau de sécurité supplémentaire. Au sein de ce mécanisme définissez et validez les données retournées par toutes les méthodes d'API, erreurs comprises.

Références

Externes

- CWE-213: Intentional Information Exposure

API4:2019 Lack of Resources & Rate Limiting

Spécifique API	**Exploitabilité: 2**	**Prévalence: 3**	**Détectabilité: 3**	**Technique: 2**	**Spécifique à l'organisation**
L'exploitation consiste en de simples requêtes d'API. Aucune authentification n'est requise. Des requêtes multiples concurrentes peuvent être effectuées depuis un unique ordinateur local ou en utilisant des ressources d'informatique en nuage.		Il est fréquent de trouver des API qui n'implémentent pas de limitations des requêtes ou dont les limitations ne sont pas correctement paramétrées.		L'exploitation peut aboutir à un déni de service, rendant l'API incapable de répondre ou même indisponible.	

L'API est-elle vulnérable ?

Les requêtes d'API consomment des ressources réseau, processeur, mémoire et de stockage. La quantité de ressources requises pour satisfaire à une requête dépendent grandement des données entrées par l'utilisateur et de la logique métier du point d'accès. De plus, il faut considérer le fait que les requêtes de différents clients de l'API sont en concurrence pour l'utilisation des ressources. Une API est vulnérable si au moins une des limites suivantes est manquante ou incorrectement paramétrée (c'est-à-dire trop basse / trop élevée) :

- durée maximale d'exécution
- Maximum de mémoire allouable
- Nombre de descripteurs de fichiers
- Nombre de processus
- Taille de la charge utile de la requête (ex. téléversement)
- Nombre de requêtes par client / ressource
- Nombre d'enregistrements par page à retourner pour chaque réponse individuelle

Exemples de scénarios d'attaque

Scénario #1

Un attaquant téléverse une grande image en émettant une requête POST vers `/api/v1/images`. Lorsque le téléversement est terminé, l'API crée plusieurs vignettes avec différentes tailles. Du fait de la taille de l'image téléversée, la mémoire disponible est saturée par la création des vignettes et l'API ne répond plus.

Scénario #2

Nous avons une application qui contient la liste des utilisateurs avec une limite de `200` utilisateurs par page. La liste des utilisateurs est obtenue auprès du serveur avec la requête suivante : `/api/users?page=1&size=100`. Un attaquant change la valeur de `size` en `200 000`, entrainant des problèmes de perfomance sur la base de données. De ce fait, l'API ne répond plus et n'est plus capable de traiter d'autres requêtes de ce client ou d'autres clients (autrement dit déni de service).

Le même scénario peut être utilisé pour générer des erreurs Integer Overflow ou Buffer Overflow.

API4:2019 Lack of Resources & Rate Limiting

Comment s'en prémunir

- Docker permet facilement de limiter la mémoire, le processeur, le nombre de redémarrages, les descripteurs de fichiers et les processus.
- Implémentez une limite du nombre d'appels à l'API qu'un client peut effectuer sur une période donnée.
- Notifiez le client quand la limite est dépassée en indiquant la limite et quand cette limite sera remise à zéro.
- Ajoutez des validations adaptées côté serveur pour les paramètres fournis en ou fournis en corps de requête, en particulier ceux qui contrôlent le nombre d'enregistrements à retourner dans la réponse.
- Définissez et appliquez une taille maximale pour les données entrées en paramètres et les charges utiles, comme des longueurs maximales pour les chaines de caractères et un nombre maximal d'éléments dans les tableaux.

Références

OWASP

- Blocking Brute Force Attacks
- Docker Cheat Sheet - Limit resources (memory, CPU, file descriptors, processes, restarts)
- REST Assessment Cheat Sheet

Externes

- CWE-307: Improper Restriction of Excessive Authentication Attempts
- CWE-770: Allocation of Resources Without Limits or Throttling
- "*Rate Limiting (Throttling)*" - Security Strategies for Microservices-based Application Systems, NIST

API5:2019 Broken Function Level Authorization

Spécifique API	**Exploitabilité: 3**	**Prévalence: 2**	**Détectabilité: 1**	**Technique: 2**	**Spécifique à l'organisation**
L'exploitation requiert que l'attaquant envoie des appels d'API légitimes vers un point d'accès d'API auquel il ne devrait pas avoir accès. Ces points d'accès peuvent être exposés à des utilisateurs anonymes ou à des utilisateurs normaux dépourvus de privilèges. Il est plus facile de découvrir ces failles dans les API car les API sont plus structurées, et le mode d'accès à certaines fonctions est plus prévisible (ex : remplacer la méthode HTTP GET par PUT, ou changer la chaine "users" de la requête en "admins".		Les contrôles d'autorisation pour une fonction ou une ressource sont généralement gérés via la configuration, et parfois au niveau du code. L'implémentation de contrôles appropriés peut être source de confusion, car les applications modernes peuvent contenir de nombreux types de rôles ou de groupes et une hiérarchie des utilisateurs complexe (ex : sous-utilisateurs, utilisateurs avec plusieurs rôles).		Ces failles permettent aux attaquants d'accéder à des fonctionnalités non autorisées. Les tâches administratives constituent des cibles principales pour ce type d'attaque.	

L'API est-elle vulnérable ?

La meilleure manière de trouver des problèmes de niveaux d'accès aux fonctionnalités consiste à effectuer une analyse approfondie du mécanisme d'autorisation, tout en gardant à l'esprit la hiérarchie des utilisateurs, les différents rôles ou groupes dans l'application, et à poser les questions suivantes :

- Un utilisateur normal peut-il accéder à des points d'accès d'administration ?
- Un utilisateur peut-il effectuer des actions sensibles (ex : création, modification ou suppression) auxquelles il ne devrait pas avoir accès en changeant simplement la méthode HTTP (ex : de `GET` à `DELETE`) ?
- Un utilisateur du groupe X peut-il accéder à une fonction qui ne devrait être accessible qu'aux utilisateurs du groupe Y, simplement en devinant l'URL du point d'accès et les paramètres (ex : `/api/v1/users/export_all`) ?

Ne supposez pas qu'un point d'accès d'API est normal ou administrateur uniquement sur la base du chemin de l'URL.

Si les développeurs peuvent choisir d'exposer la plupart des points d'accès d'administration sous un chemin relatif spécifique, comme `api/admins`, on trouve très fréquemment ces points d'accès administrateur sous d'autres chemins relatifs mélés aux points d'accès normaux, comme `api/users`.

Exemples de scénarios d'attaque

Scénario #1

Au cours du processus d'enregistrement à une application qui permet uniquement aux utilisateurs invités de s'inscrire, l'application mobile effectue un appel d'API à `GET /api/invites/{invite_guid}`. La réponse contient un JSON avec des informations sur l'invitation, parmi lesquelles le rôle de l'utilisateur et son e-mail.

API5:2019 Broken Function Level Authorization

Un attaquant a dupliqué la requête et manipulé la méthode HTTP et le point d'accès vers `POST /api/invites/new`. Ce point d'accès devrait être accessible uniquement aux administrateurs via la console d'administration, qui n'implémente pas de contrôles de niveaux d'accès aux fonctionnalités.

L'attaquant exploite cette faille et s'envoie à lui-même une invitation pour se créer un compte administrateur :

```
POST /api/invites/new

{"email":"hugo@malicious.com","role":"admin"}
```

Scénario #2

Une API comporte un point d'accès qui devrait uniquement être accessible aux administrateurs : `GET /api/admin/v1/users/all`. Ce point d'accès renvoie les informations sur tous les utilisateurs de l'application et n'implémente pas de contrôles d'autorisations d'accès. Un attaquant ayant appris la structure de l'API effectue une déduction logique et réussit à accéder à ce point d'accès, qui expose des données sensibles sur les utilisateurs de l'application..

Comment s'en prémunir

Votre application devrait disposer d'un module d'autorisations constant et facile à analyser qui est invoqué par toutes vos fonctions métiers. Fréquemment, cette protection est fournie par un ou plusieurs composants externes au code de l'application.

- Le(s) mécanisme(s) de contrôle devraient interdire tous les accès par défaut, et requérir des privilèges explicites à des rôles spécifiques pour l'accès à toutes les fonctions.
- Passez en revue vos points d'accès d'API à la recherche des défauts d'autorisations niveau des fonctions, en gardant à l'esprit la logique applicative et la hiérarchie des groupes.
- Assurez-vous que tous vos contrôleurs d'administration héritent d'un contrôleur d'administration abstrait qui implémente des contrôles d'autorisation basés sur le groupe / rôle de l'utilisateur.
- Assurez-vous que les fonctions d'administration à l'intérieur d'un contrôleur normal implémentent des contrôles d'autorisation basés sur le groupe / rôle de l'utilisateur.

Références

OWASP

- OWASP Article on Forced Browsing
- OWASP Top 10 2013-A7-Missing Function Level Access Control
- OWASP Development Guide: Chapter on Authorization

Externes

- CWE-285: Improper Authorization

API6:2019 Mass Assignment

Spécifique API	**Exploitabilité: 2**	**Prévalence: 2**	**Détectabilité: 2**	**Technique: 2**	**Spécifique à l'organisation**
L'exploitation requiert généralement une compréhension de la logique métier, des relations entre objets, et de la structure de l'API. L'exploitation de l'assignation massive est plus facile dans les API, car par conception elles exposent l'implémentation sous-jacente de l'application ainsi que les noms des attributs.		Les frameworks modernes encouragent les développeurs à utiliser des fonctions qui lient automatiquement les entrées du client aux variables du code et aux objets internes. Des attaquants peuvent utiliser cette méthodologie pour modifier ou écraser des attributs d'objets sensibles que les développeurs n'avaient jamais eu l'intention d'exposer.		L'exploitation peut conduire à l'élévation des privilèges, la falsification des données, au contournement de mécanismes de sécurité, et plus encore.	

L'API est-elle vulnérable ?

Les objets des applications modernes peuvent posséder de nombreux attributs. Certains de ces attributs doivent pouvoir être actualisés par le client (ex : `user.first_name` ou `user.address`) et d'autres ne doivent pas pouvoir l'être (ex : drapeau `user.is_vip`).

Un point d'accès d'API est vulnérable s'il convertit automatiquement des paramètres client en attributs objet internes, sans prendre en compte la sensibilité et le niveau d'exposition de ces attributs. Ceci pourrait permettre à un attaquant d'actualiser des attributs d'objets auxquels il ne devrait pas avoir accès.

Exemples d'informations sensibles :

- **Attributs liés à des permissions** : `user.is_admin`, `user.is_vip` doivent être définis uniquement par des administrateurs.
- **Attributs dépendant de processus** : `user.cash` ne doit être défini au niveau interne qu'après vérification du paiement.
- **Attributs internes** : `article.created_time` doit être défini uniquement par l'application.

Exemples de scénarios d'attaque

Scénario #1

Une application de covoiturage permet à l'utilisateur de modifier les informations de base de son profil. Au cours de ce processus, un appel d'API est envoyé à `PUT /api/v1/users/me` avec l'objet JSON légitime suivant :

```
{"user_name":"inons","age":24}
```

La requête `GET /api/v1/users/me` comporte un attribut supplémentaire sur le solde du compte :

```
{"user_name":"inons","age":24,"credit_balance":10}.
```

L'attaquant rejoue la première requête avec la charge utile suivante :

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

Le point d'accès étant vulnérable à l'assignation massive, l'attaquant dispose du crédit sans avoir payé.

API6:2019 Mass Assignment

Scénario #2

Un portail de partage de vidéos permet aux utilisateurs de téléverser et de télécharger du contenu dans différents formats. Un attaquant qui explore l'API découvre que le point d'accès `GET /api/v1/videos/{video_id}/meta_data` retourne un objet JSON avec les attributs de la vidéo. L'un des attributs est `"mp4_conversion_params":"-v codec h264"`, qui indique que l'application utilise une commande shell pour convertir la vidéo.

L'attaquant a également découvert que le point d'accès `POST /api/v1/videos/new` est vulnérable à l'assignation massive et permet au client de définir n'importe quel attribut de l'objet vidéo. L'attaquant définit une valeur malveillante de la manière suivante : `"mp4_conversion_params":"-v codec h264 && format C:/"`. Cette valeur va entrainer une injection de commande shell quand l'attaquant chargera la vidéo au format MP4.

Comment s'en prémunir

- Si possible, évitez d'utiliser des fonctions qui lient automatiquement une saisie client à des variables du code ou des objets internes.
- Autorisez uniquement les attributs qui doivent pouvoir être actualisés par le client.
- Utilisez les fonctionnalités natives pour interdire les attributs qui ne doivent pas être accessibles aux clients.
- Si applicable, définissez et imposez des schémas pour la validation des données d'entrée de la charge utile .

Références

Externes

- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

API7:2019 Security Misconfiguration

Spécifique API	**Exploitabilité: 3**	**Prévalence: 3**	**Détectabilité: 3**	**Technique: 2**	**Spécifique à l'organisation**
Les attaquants essaieront souvent de trouver des failles non corrigées, des points d'accès courants, ou des fichiers et des répertoires non protégés pour obtenir un accès non autorisé ou des informations sur le système.		Des erreurs de configuration de sécurité peuvent se produire à toutes les couches de l'API, depuis la couche réseau jusqu'à la couche applicative. Il existe des outils automatisés pour détecter et exploiter des erreurs de configuration telles que des services non nécessaires ou d'anciennes options.		Les erreurs de configuration de sécurité peuvent non seulement exposer des données utilisateur confidentielles, mais aussi des informations système qui peuvent aboutir à une compromission complète du serveur.	

L'API est-elle vulnérable ?

L'API peut être vulnérable si :

- Un durcissement approprié de la sécurité est manquant sur n'importe quelle partie de la pile applicative, ou si elle possède des permissions mal configurées au niveau de services cloud.
- Les dernières mises à jour de sécurité ne sont pas appliquées, ou les systèmes sont obsolètes.
- Des fonctionnalités non nécessaires sont déployées (ex : verbes HTTP).
- La sécurité de la couche de transport (TLS) est manquante.
- Les instructions d'en-têtes sécurisées ne sont pas envoyées au client (e.g., Security Headers).
- La politique de partage de ressources entre origines multiples (CORS) est manquante ou mal définie.
- Les messages d'erreurs incluent des traces de pile d'exécution, ou exposent d'autres informations sensibles.

Exemples de scénarios d'attaque

Scénario #1

Un attaquant trouve le fichier `.bash_history` dans le répertoire racine du serveur, qui contient les commandes utilisées par l'équipe DevOps pour accéder à l'API :

```
$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vOmJhcg=='
```

Un attaquant pourrait aussi découvrir de nouveaux points d'accès de l'API qui sont uniquement utilisés par l'équipe DevOps et qui ne sont pas documentés.

Scénario #2

Pour cibler un service spécifique, un attaquant utilise un moteur de recherche populaire pour chercher les ordinateurs directement accessibles depuis l'internet. L'attaquant trouve un hôte faisant tourner un système populaire de gestion de bases de données, qui écoute sur le port par défaut. L'hôte utilisait la configuration par défaut, dans laquelle l'authentification est désactivée par défaut, et l'attaquant a obtenu accès à des millions d'enregistrements contenant des données personnelles, des préférences personnelles, et des données d'authentification.

API7:2019 Security Misconfiguration

Scénario #3

En inspectant le trafic d'une application mobile un attaquant découvre que tout le trafic n'est pas effectué avec un protocole sécurisé (ex : TLS). L'attaquant le constate en particulier pour le téléchargement des images de profil. Comme l'interaction est binaire, malgré le fait que le trafic API est effectué sur un protocole sécurisé, l'attaquant remarque un schéma au niveau de la taille des réponses de l'API, qu'il utilise pour pister les préférences utilisateur à partir du contenu rendu (ex : images de profil).

Comment s'en prémunir

Le cycle de vie de l'API devrait inclure :

- Un processus de durcissement permettant le déploiement rapide et facile d'un environnement correctement verrouillé.
- Une tâche pour évaluer et actualiser les configurations sur l'ensemble des couches de l'API. L'évaluation devrait couvrir : les fichiers d'orchestration, les composants d'API, et les services cloud (ex : permissions des compartiments de stockage S3).
- Un canal de communication sécurisé pour tous les accès API d'interaction avec les éléments statiques (ex : images).
- Un processus automatisé pour évaluer en continu l'efficacité de la configuration et des réglages dans tous les environnements.

De plus :

- Pour éviter que des traces d'appels lors d'exceptions et d'autres informations importantes ne soient renvoyées aux attaquants, si applicable, définissez et implémentez des schémas pour toutes les réponses y compris les erreurs.
- Assurez-vous que l'API ne soit accessible qu'avec des verbes HTTP spécifiques. Tous les autres verbes HTTP doivent être désactivés (ex :`HEAD`).
- Les API devant être accessibles via des clients basés sur des navigateurs (ex : WebApp en front-end) doivent implémenter une politique appropriée de partage de ressources entre origines multiples (CORS).

Références

OWASP

- OWASP Secure Headers Project
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Testing Guide: Test Cross Origin Resource Sharing

Externes

- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- Guide to General Server Security, NIST
- Let’s Encrypt: a free, automated, and open Certificate Authority

API8:2019 Injection

Spécifique API	**Exploitabilité: 3**	**Prévalence: 2**	**Détectabilité: 3**	**Technique: 3**	**Spécifique à l'organisation**
Les attaquants vont envoyer à l'API des données malveillantes via tout vecteur d'injection disponible (ex : entrée directe, paramètres, services intégrés, etc), en espérant qu'elles soient envoyés à un interpréteur.		Les failles par injection sont très courantes et souvent trouvées dans des requêtes SQL, LDAP, ou NoSQL, des commandes systèmes (OS), des parsers XML et des ORM. Ces failles sont faciles à trouver lors de la revue du code source. Les attaquants peuvent utiliser des scanners et des fuzzers.		L'injection peut aboutir à des divulgations d'informations et des pertes de données. Elle peut aussi aboutir à un déni de service, ou à une prise de contrôle complète de l'hôte.	

L'API est-elle vulnérable ?

L'API est vulnérables à l'injection si :

- Les données fournies par le client ne sont pas validées, filtrées et épurées par l'API.
- Les données fournies par le client sont directement utilisées ou concaténées dans des requêtes SQL / NoSQL / LDAP, des commandes de système d'exploitation, des parsers XML ou des mappages objet-relationnel (ORM) / mappages objet-document (ODM).
- Les données en provenance de systèmes externes (ex : systèmes intégrés) ne sont pas validées, filtrées et épurées par l'API.

Exemples de scénarios d'attaque

Scénario #1

Le firmware d'un appareil de contrôle parental dispose d'un point d'accès `/api/CONFIG/restore` qui prend en entrée un appId devant être envoyé comme paramètre multiparties. Avec un décompilateur, un attaquant découvre que l'appId est passé directement dans un appel système sans aucune épuration :

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
         "/mnt/shares/usr/bin/scripts/", appid, 66);
system(cmd);
```

La commande suivante permet à l'attaquant d'arrêter tout appareil équipé de ce même firmware vulnérable :

```
$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F 'appid=$
(/etc/pod/power_down.sh)'
```

Scénario #2

Nous avons une application dotée de fonctionnalités CRUD basiques pour les opérations de réservation. Un attaquant a réussi à découvrir qu'une injection NoSQL pourrait être possible via le paramètre `bookingId` de la chaine de requête pour la suppression d'une réservation. Voici à quoi ressemble cette requête : `DELETE /api/bookings?bookingId=678`.

L'API serveur utilise la fonction suivante pour traiter les requêtes de suppression :

```
router.delete('/bookings', async function (req, res, next) {
  try {
      const deletedBooking = await Bookings.findOneAndRemove({'_id' :
req.query.bookingId});
      res.status(200);
  } catch (err) {
```

```
    res.status(400).json({error: 'Unexpected error occured while processing a
request'});
  }
});
```

L'attaquant a intercepté la requête et a remplacé le paramètre `bookingId` de la chaine de requête comme indiqué ci-dessous. Dans le cas présent, l'attaquant a réussi à supprimer la réservation d'un autre utilisateur :

```
DELETE /api/bookings?bookingId[$ne]=678
```

Comment s'en prémunir

Prévenir les injections requiert de séparer les données des commandes et des requêtes.

- Effectuez la validation des données avec une bibliothèque unique, digne de confiance et activement maintenue.
- Validez, filtrez et épurez toutes les données fournies par le client, ou les autres données en provenance de systèmes intégrés.
- Les caractères spéciaux doivent être échappés en utilisant la syntaxe spécifique à l'interpréteur cible.
- Préférez une API sûre qui fournit une interface paramétrée.
- Limitez toujours le nombre d'enregistrements retournés pour éviter les divulgations de masse en cas d'injection.
- Validez les données entrantes avec suffisamment de filtres pour accepter uniquement les valeurs valides pour chaque paramètre d'entrée.
- Définissez des types de données et des schémas stricts pour tous les paramètres de chaines.

Références

OWASP

- OWASP Injection Flaws
- SQL Injection
- NoSQL Injection Fun with Objects and Arrays
- Command Injection

Externes

- CWE-77: Command Injection
- CWE-89: SQL Injection

API9:2019 Improper Assets Management

Spécifique API	**Exploitabilité: 3**	**Prévalence: 3**	**Détectabilité: 2**	**Technique: 2**	**Spécifique à l'organisation**
Les anciennes versions des API n'ont souvent pas bénéficié des correctifs de sécurité et sont un moyen facile pour compromettre des systèmes sans avoir à affronter des mécanismes de sécurité de pointe, qui peuvent avoir été mis en place pour protéger les versions les plus récentes de l'API.		Une documentation obsolète rend plus difficile la recherche et / ou la correction de vulnérabilités. L'absence d'inventaire des points actifs et de stratégies de retrait pour ceux ne devant plus être utilisés, conduit à faire tourner des systèmes dépourvus de correctifs de sécurité, entrainant la divulgation de données sensibles. Des hôtes d'API inutilement exposés sont fréquemment trouvés du fait des concepts modernes comme les micro-services, qui rendent les applications faciles à déployer et indépendantes (ex : cloud, aussi appelé informatique en nuage, Kubernetes).		Les attaquants peuvent obtenir accès à des données sensibles, et même prendre le contrôle du serveur via d'anciennes versions non corrigées de l'API connectées à la même base de données.	

L'API est-elle vulnérable ?

L'API peut être vulnérable si :

- L'objectif de l'hôte de l'API n'est pas clair, et il n'y a pas de réponses explicites aux questions suivantes :
  - Dans quel environment tourne l'API (ex : production, staging, test, développement) ?
  - Qui doit avoir un accès réseau à l'API (ex : public, interne, partenaires) ?
  - Quelle version de l'API tourne ?
  - Quelles données sont collectées et traitées par l'API (ex : données personnelles) ?
  - Quel est le flux des données ?
- Il n'y a pas de documentation, ou la documentation existante n'est pas mise à jour.
- Il n'y a pas de plan pour le retrait / la désactivation (des points d'accès devenus obsolètes) par version d'API.
- L'inventaire des hôtes est manquant ou obsolète.
- L'inventaire des services intégrés, en propre ou par des tiers, est manquant ou obsolète.
- Des versions anciennes ou antérieures de l'API tournent sans correctifs.

Exemples de scénarios d'attaque

Scénario #1

Après avoir repensé ses applications, un service local de recherche avait laissé une ancienne version de l'API (`api.someservice.com/v1`) tourner sans protection, avec un accès à la base de données clients. En ciblant l'une des applications dernièrement publiées, un attaquant a trouvé l'adresse de l'API (`api.someservice.com/v2`). En remplaçant `v2` par `v1` dans l'URL l'attaquant a obtenu accès à l'ancienne API non protégée, exposant les données personnelles de plus de 100 millions d'utilisateurs.

API9:2019 Improper Assets Management

Scénario #2

Un réseau social avait mis en place un mécanisme de limitation du nombre de requêtes pour empêcher des attaquants d'employer la force brute pour deviner les jetons (Token) de réinitialisation des mots de passe. Ce mécanisme n'était pas implémenté au niveau du code de l'API elle-même, mais dans un composant séparé situé entre le client et l'API officielle (`www.socialnetwork.com`). Un chercheur découvrit un hôte d'API en beta (`www.mbasic.beta.socialnetwork.com`) faisant tourner la même API, y compris le mécanisme de réinitialisation du mot de passe, mais était dépourvu du mécanisme de limitation du nombre de requêtes. Le chercheur fut alors en mesure de réinitialiser le mot de passe de n'importe quel utilisateur simplement en utilisant la force brute pour deviner le token à 6 chiffres.

Comment s'en prémunir

- Inventoriez tous les hôtes d'API et documentez les aspects importants de chacun d'entre eux, en vous concentrant sur l'environnement de l'API (ex : production, staging, test, développement), sur qui devrait avoir un accès réseau à l'hôte (ex : public, interne, partenaires) et les versions de l'API.
- Inventoriez les systèmes intégrés et documentez les aspects importants tels que leur rôle dans le système, quelles données sont échangées (flux des données) et leur sensibilité.
- Documentez tous les aspects de votre API et notamment l'authentification, les erreurs, les redirections, la limitation du nombre de requêtes, la politique de partage de ressources entre origines multiples (CORS) et les points d'accès, incluant leurs paramètres, les requêtes et les réponses.
- Générez la documentation automatiquement en adoptant des standards ouverts. Intégrez cette génération automatique de la documentation dans votre processus de déploiement continu CI/CD.
- Donnez accès à la documentation de l'API aux personnes autorisées à utiliser l'API.
- Utilisez des mesures de protection externes telles les pare-feux de sécurité pour API, et ce, pour toutes les versions exposées de vos API, pas seulement pour la version courante en production.
- Évitez d'utiliser des données de production avec des déploiements d'API autres que ceux de production. Si vous ne pouvez l'évitez, ces points d'accès doivent bénéficier du même niveau de sécurité que ceux de production.
- Lorsque de nouvelles versions d'API intègrent des améliorations de sécurité, effectuez une analyse de risque pour décider les actions d'atténuation requises pour l'ancienne version : par exemple, s'il est ou non possible de rétro-porter les améliorations sans rompre la compatibilité ou si vous devez retirer rapidement l'ancienne version et forcer tous les clients à passer à la dernière version.

Références

Externes

- CWE-1059: Incomplete Documentation
- OpenAPI Initiative

API10:2019 Insufficient Logging & Monitoring

Spécifique API	**Exploitabilité: 2**	**Prévalence: 3**	**Détectabilité: 1**	**Technique: 2**	**Spécifique à l'organisation**
Les attaquants exploitent l'absence de logging et de monitoring pour utiliser frauduleusement des systèmes sans se faire repérer.		En l'absence de logging et de monitoring, ou si le logging et le monitoring sont insuffisants, il est pratiquement impossible de suivre des activités suspectes et d'y répondre rapidement.		Sans visibilité sur les activités malveillantes en cours, les attaquants disposent de beaucoup de temps et peuvent compromettre complètement les systèmes.	

L'API est-elle vulnérable ?

L'API is vulnérable si :

- Elle ne produit pas de logs, le niveau de logging n'est pas réglé correctement, ou les messages de log ne comportent pas suffisamment d'informations.
- L'intégrité des logs ne peut pas être garantie (ex : Log Injection).
- Les logs ne sont pas monitorés en permanence.
- L'infrastructure de l'API n'est pas monitorée en permanence.

Exemples de scénarios d'attaque

Scénario #1

Les clés d'accès d'une API d'administration ont fuitées sur un répertoire public. Le propriétaire du répertoire a été notifié par e-mail à propos de cette fuite potentielle, mais a mis plus de 48 heures à réagir à l'incident, et l'exposition des clés d'accès peut avoir permis l'accès à des données personnelles. Du fait d'un logging insuffisant, l'entreprise n'est pas capable d'évaluer quelles données ont pu être consultées par des acteurs malveillants.

Scénario #2

Une plate-forme de partage de vidéos a subi une attaque par bourrage d'identifiants de “grande ampleur”. Malgré le log des essais infructueux, aucune alerte n'a été émise pendant la durée de l'attaque. En réaction aux plaintes des utilisateurs, les logs de l'API ont été analysés et l'attaque a été détectée. L'entreprise a dû faire une annonce publique pour demander aux utilisateurs de réinitialiser leur mot de passe, et a dû déclarer l'incident aux autorités de contrôle.

API10:2019 Insufficient Logging & Monitoring

Comment s'en prémunir

- Loggez toutes les tentatives infructueuses d'authentification, les accès refusés et les erreurs de validations des données entrées.
- Les logs doivent être formatés pour pouvoir être traités par un outil de gestion des logs, et doivent inclure suffisamment d'informations pour pouvoir identifier un acteur malveillant.
- Les logs doivent être considérés comme des données sensibles, et leur intégrité doit être garantie durant leur stockage comme au cours de leur transfert.
- Configurez un système de monitoring pour surveiller en permanence l'infrastructure, le réseau et le fonctionnement de l'API.
- Utilisez un système d'information et de gestion des événements (SIEM) pour agréger et gérer les logs de tous les composants de la pile de l'API et des hôtes.
- Configurez des tableaux de bord et des alertes personnalisés, permettant la détection et le traitement plus rapide d'activités suspectes.

Références

OWASP

- OWASP Logging Cheat Sheet
- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V7: Error Handling and Logging Verification Requirements

Externes

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

+D Perspectives pour les Développeurs

Créer ou maintenir la sécurité d'un logiciel, ou corriger un logiciel existant, peut s'avérer difficile. Il en va de même pour les API.

Nous pensons que l'éducation et la connaissance sont des facteurs clés pour écrire des logiciels sécurisés. L'atteinte de cet objectif repose ensuite sur **la mise en place et l'utilisation de processus de sécurité reproductibles et de contrôles de sécurité standards**.

L'OWASP propose de nombreuses ressources gratuites et libres pour aborder la sécurité dès le début d'un projet. Veuillez visiter la page des projets OWASP pour connaitre la liste complète des projets disponibles.

Éducation	Vous pouvez commencer par lire les projets de la catégorie OWASP Education en fonction de votre profession de votre intérêt. Pour une approche plus pratique, nous avons ajouté le projet **crAPI - C**ompletely **R**idiculous **API** à notre roadmap. En attendant, vous pouvez vous entrainer à la sécurité des applis web avec le module OWASP DevSlop Pixi, une WebApp et un service d'API volontairement vulnérables destinés à apprendre aux utilisateurs comment tester la sécurité des applications web modernes et des services d'API, et comment développer des API plus sécurisées à l'avenir. Vous pouvez également participer à des sessions de formation de conférence OWASP AppSec, ou rejoindre une section OWASP locale.
Besoins de Sécurité	La sécurité doit faire partie de chaque projet dès le début. Lors de la formulation des besoins, il est important de définir ce que "sécurisé" signifie pour ce projet. L'OWASP vous recommande d'utiliser le OWASP Application Security Verification Standard (ASVS) comme guide pour définir vos besoins de sécurité. Si vous sous-traitez, envisagez le projet OWASP Secure Software Contract Annex, qui devra être adapté aux lois et réglementations locales.
Architecture de Sécurité	La sécurité doit rester une préoccupation durant toutes les étapes du projet. Les OWASP Prevention Cheat Sheets sont un bon point de départ pour guider la conception de la sécurité durant la phase d'architecture. Parmi beaucoup d'autres, vous trouverez la REST Security Cheat Sheet et la REST Assessment Cheat Sheet.
Contrôles de Sécurité Standards	L'adoption de contrôles de sécurité standards réduit le risque d'introduire des vulnérabilités de sécurité lorsque vous implémentez votre logique métier. Bien que de nombreux frameworks modernes incluent désormais des contrôles standards efficaces, OWASP Proactive Controls vous fournit un bon résumé des contrôles de sécurité que vous devriez inclure dans votre projet. L'OWASP fournit aussi quelques bibliothèques et outils que vous pourrez trouver utiles, tels que des contrôles de validation.
Cycle de Développement Logiciel Sécurisé	Vous pouvez utiliser le OWASP Software Assurance Maturity Model (SAMM) pour améliorer le processus de développement d'API. Plusieurs autres projets OWASP sont disponibles pour vous aider durant les différentes phases de développement d'API, par ex. le OWASP Code Review Project.

+DSO Perspectives pour les DevSecOps

Du fait de leur importance dans les architectures des applications modernes, il est crucial de construire des API sécurisées. La sécurité ne peut pas être négligée, et elle doit faire partie de l'ensemble du processus de développement. Effectuer un scan et un test d'intrusion annuel n'est plus suffisant.

Les DevSecOps doivent participer à l'effort de développement et faciliter les tests continus de sécurité sur l'ensemble du cycle de vie du développement logiciel. Leur but est d'améliorer le processus de développement avec une automatisation de la sécurité, sans impacter la vitesse de développement.

En cas de doute, tenez-vous informé et consultez souvent le DevSecOps Manifesto.

Compréhension du modèle de menaces	Les priorités de tests sont déterminées par le modèle de menaces. Si vous n'en avez pas, envisagez d'utiliser notre OWASP Application Security Verification Standard (ASVS), et notre OWASP Testing Guide comme bases. Impliquer l'équipe de développement peut contribuer à les rendre plus conscients de la sécurité.
Comprendre le SDLC	Joignez-vous à l'équipe de développement pour mieux comprendre le cycle de développement logiciel (SDLC - Software Development Life Cycle). Votre contribution aux tests de sécurité continus doit être compatible avec les personnes, les procédés et les outils. Tout le monde doit adhérer à la démarche, afin d'éviter des frictions ou de la résistance inutiles.
Stratégies de tests	Comme votre travail ne doit pas impacter la vitesse de développement, vous devez choisir judicieusement la meilleure technique (simple, la plus rapide, la plus juste) pour vérifier les exigences de sécurité. Les projets OWASP Security Knowledge Framework et OWASP Application Security Verification Standard peuvent constituer d'excellentes sources d'exigences de sécurité fonctionnelles et non-fonctionnelles. Il existe également d'autres ressources et contenus de qualité tels les projets et outils proposés par la communauté DevSecOps.
Obtention de couverture et précision	Vous êtes le lien entre les développeurs (Dev) et les équipes opérationnelles (Ops). Pour réaliser la couverture, vous devez vous concentrer non seulement sur la fonctionnalité, mais aussi sur l'orchestration. Travaillez en étroite relation à la fois avec les équipes de développement et des opérations (infra) dès le début pour pouvoir optimiser votre temps et vos efforts. Vous devez viser un état où la sécurité essentielle est vérifiée continuellement.
Communication claire des résultats	Apportez de la valeur avec pas ou peu de friction. Alertez promptement sur vos découvertes, en utilisant les moyens et outils mis en oeuvres par vos équipes (pas dans des fichiers PDF). Joignez-vous à l'équipe de développement pour résoudre ces problèmes. Profitez de l'occasion pour les instruire, en décrivant clairement la vulnérabilité et la manière dont elle peut être exploitée, avec un scénario d'attaque pour la rendre réelle.

+DAT

Méthodologie et Données

Présentation

Comme l'industrie de la sécurité applicative n'est pas spécifiquement concentrée sur les architectures applicatives les plus récentes, dans lesquelles les API jouent un rôle important, il aurait été difficile de compiler une liste des dix risques de sécurité les plus critiques pour les API en s'appuyant sur un appel public à informations. Bien qu'il n'y ait pas eu de tel appel public à informations, la liste résultante composant le Top 10 est basée sur des informations publiquement accessibles, des contributions d'experts en sécurité, et des discussions ouvertes avec la communauté de la sécurité.

Méthodologie

Dans un premier temps, des informations publiquement disponibles sur des incidents de sécurité concernant des API ont été collectées, évaluées, et catégorisées par un groupe d'experts en sécurité. Ces données ont été collectées à partir de plateformes de prime aux bogues (bug bounty) et de bases de données de vulnérabilités, sur une période d'un an, à des fins statistiques.

Dans un deuxième temps, il a été demandé à des praticiens de la sécurité expérimentés en tests d'intrusion de compiler leur propre Top 10.

La méthodologique d'évaluation de risque OWASP a été utilisée pour effectuer l'analyse de risques. Les scores ont été discutés et évalués par les praticiens de la sécurité. Sur ces questions, veuillez vous référer à la section des risques de sécurité des API.

La première ébauche de l'OWASP API Security Top 10 2019 résultait d'un consensus entre les données statistiques de la première phase et les listes des praticiens en sécurité. Cette ébauche a ensuite été soumise pour avis et évaluation à un autre groupe de praticiens de la sécurité disposant d'expériences en lien avec la sécurité des API.

Le OWASP API Security Top 10 2019 a été présenté pour la première fois lors de l'événement OWASP Global AppSec Tel Aviv (mai 2019). Depuis lors, il a été mis à disposition sur GitHub pour permettre discussions et contributions publiques.

La liste des contributeurs est disponible dans la section des Remerciements.

+ACK Remerciements

Remerciements aux Contributeurs

Nous voulons remercier les contributeurs suivants qui ont contribué publiquement sur GitHub ou par d'autres moyens :

- 007divyachawla
- Abid Khan
- Adam Fisher
- anotherik
- bkimminich
- caseysoftware
- Chris Westphal
- dsopas
- DSotnikov
- emilva
- ErezYalon
- flascelles
- Guillaume Benats
- IgorSasovets
- Inonshk
- JonnySchnittger
- jmanico
- jmdx
- Keith Casey
- kozmic
- LauraRosePorter
- Matthieu Estrade
- nathanawmk
- PauloASilva
- pentagramz
- philippederyck
- pleothaud
- r00ter
- Raj kumar
- Sagar Popat
- Stephen Gates
- thomaskonrad
- xycloops123