Chapitre 3
Approfondissement des concepts de sécurité

1. Le CI5A

Le modèle CI5A est un cadre simple mais puissant pour chaque professionnel de l'informatique amené à concevoir une architecture ou une application sécurisée, notamment basée sur des API ou mettant en œuvre des API, au travers d'une plateforme d'API Management.

Cet acronyme définit chaque item de sécurité qui doit obligatoirement être pris en compte dès les phases de conception jusqu'à la mise en production d'un système informatique.

Il a donc toute sa place dans la conception des systèmes d'information basés sur des architectures Cloud et les API.

Par expérience, il est recommandé d'utiliser ce framework lors du design des API, afin de prendre en compte la sécurité dès les premières phases de conception des API dans le cadre d'un programme nouvellement initié.

1.1 Confidentialité (Confidentiality), Intégrité (Integrity), Disponibilité (Availability)

1.1.1 Confidentialité

La confidentialité garantit que seules les personnes autorisées ont accès aux informations et qu'elles sont protégées contre tout accès non autorisé.

Les mesures couramment utilisées pour assurer la confidentialité comprennent notamment le chiffrement, qui consiste à chiffrer les données pour les protéger. En effet, les données sensibles doivent être chiffrées, que ce soit au repos (stockées sur des disques durs ou dans le Cloud) ou en transit (lorsqu'elles sont envoyées sur un réseau).

La confidentialité des données dans le cadre d'un programme API est essentielle pour protéger les informations sensibles échangées entre les clients et les serveurs. Pour la garantir, plusieurs bonnes pratiques doivent être mises en œuvre. Tout d'abord, le chiffrement des données en transit via le protocole HTTPS (TLS) est indispensable pour empêcher les interceptions malveillantes. Ensuite, une gestion rigoureuse des accès doit être instaurée, utilisant des mécanismes d'authentification robustes comme OAuth 2.0 ou les jetons JWT, et en limitant les permissions aux seuls utilisateurs et applications autorisés. Les données sensibles doivent être masquées ou pseudonymisées dans les réponses API pour éviter toute exposition inutile. Côté stockage, les données au repos doivent être chiffrées à l'aide d'algorithmes robustes comme AES-256, et les clés de chiffrement doivent être gérées de manière sécurisée, idéalement via des modules matériels (HSM). Enfin, des audits réguliers et des tests de sécurité (pentests) permettent de détecter et corriger les vulnérabilités, tandis que la conformité aux réglementations comme le RGPD ou le CCPA assure une protection juridique des données. En combinant ces mesures, on garantit une confidentialité optimale des données tout au long de leur cycle de vie.

1.1.2 Intégrité

L'intégrité, dans le contexte de la sécurité informatique, est un principe fondamental qui garantit que les données restent exactes, cohérentes et non altérées tout au long de leur cycle de vie.

L'intégrité des données revêt une importance capitale pour plusieurs raisons :

- **Fiabilité des informations** : des données altérées ou corrompues peuvent entraîner de mauvaises décisions et avoir des conséquences graves. L'intégrité assure que les informations sur lesquelles vous vous appuyez sont exactes.
- **Protection contre les attaques** : les cybercriminels peuvent essayer de modifier des données afin de voler des informations, perturber des systèmes ou causer d'autres dommages. L'intégrité aide à détecter et à prévenir ces attaques.
- **Conformité réglementaire** : de nombreuses réglementations, comme le RGPD, exigent que les organisations mettent en place des mesures pour garantir l'intégrité des données.

L'intégrité des données dans un programme API est un aspect critique pour garantir que les informations échangées entre le client et le serveur ne subissent aucune altération, que ce soit pendant leur transmission ou leur stockage. Pour assurer cette intégrité, plusieurs pratiques techniques doivent être mises en place. Premièrement, l'utilisation du protocole HTTPS avec des versions sécurisées de TLS (1.2 ou 1.3) est indispensable pour chiffrer les données en transit et prévenir les attaques de type *man-in-the-middle* qui pourraient les modifier.

En complément, des mécanismes comme les signatures numériques (préférentiellement via des clés asymétriques) ou des fonctions de hachage sécurisées (SHA-256) peuvent être implémentés pour vérifier l'intégrité des données transmises. Côté serveur, il est essentiel de valider et de « sanitizer » (nettoyer) toutes les entrées pour éviter les injections de données malveillantes ou les corruptions.

Enfin, pour les données au repos, des techniques comme les checksums ou les contrôles d'intégrité réguliers permettent de détecter et de corriger toute altération. En adoptant ces mesures techniques, on s'assure que les données conservent leur fiabilité et leur exactitude tout au long de leur cycle de vie.

1.1.3 Disponibilité

La disponibilité garantit que les systèmes, les données et les ressources informatiques sont accessibles et opérationnels lorsque les utilisateurs autorisés en ont besoin.

Cela implique :

- **Un accès permanent** : les systèmes doivent être en ligne et fonctionnels en tout temps, ou du moins conformément aux niveaux de service définis.
- **Une performance adéquate** : les systèmes doivent répondre aux demandes en temps opportun, sans délais excessifs.
- **Une résistance aux pannes** : les systèmes doivent être conçus pour résister aux défaillances matérielles, logicielles ou réseau, et être en mesure de récupérer rapidement en cas d'incident.
- **Une protection contre les attaques** : les systèmes doivent être protégés contre les attaques visant à perturber leur fonctionnement, telles que les attaques par déni de service (DoS).

La disponibilité des données dans un programme API est un enjeu majeur pour garantir un accès continu et fiable aux services, même en cas de pic de charge ou de défaillance. Pour assurer une haute disponibilité, plusieurs bonnes pratiques techniques doivent être mises en œuvre. Tout d'abord, l'adoption d'une architecture scalable et redondante est essentielle, en utilisant des solutions comme le load balancing pour répartir la charge entre plusieurs serveurs et des systèmes de failover pour basculer automatiquement en cas de panne. L'utilisation de services Cloud avec des zones de disponibilité multiples (multi-AZ) ou des régions géographiques différentes permet de minimiser les risques d'indisponibilité liés à des défaillances locales.

Ensuite, la mise en place de mécanismes de *caching* (via des outils comme Redis ou Varnish) réduit la charge sur les serveurs backend et améliore les temps de réponse. Pour anticiper les pics de trafic, des stratégies de *rate limiting* et de *throttling* doivent être configurées pour éviter la saturation des ressources. Enfin, des tests réguliers de résilience (*chaos engineering*) et des plans de reprise d'activité (PRA) permettent de valider la robustesse du système et de garantir une récupération rapide en cas d'incident. En combinant ces approches, on maximise la disponibilité des données et des services API, même dans des conditions extrêmes.

1.2 Authentification (Authentication), Autorisation (Authorization), Accounting

1.2.1 Authentification

L'authentification est une étape cruciale pour vérifier l'identité d'un utilisateur, d'un système ou d'une entité avant de leur accorder l'accès à des ressources. Il existe plusieurs méthodes couramment utilisées pour l'authentification, telles que les mots de passe, la biométrie et les *tokens* (jetons).

Les mots de passe sont des combinaisons de caractères secrètes que les utilisateurs doivent fournir pour prouver leur identité. La biométrie, quant à elle, utilise des caractéristiques physiques uniques, comme les empreintes digitales, pour vérifier l'identité d'une personne. Enfin, les tokens sont des dispositifs matériels ou logiciels qui génèrent des codes d'accès temporaires pour authentifier les utilisateurs.

Ces méthodes d'authentification sont largement utilisées dans le monde professionnel pour garantir la sécurité des systèmes et des données sensibles. Elles permettent de s'assurer que seules les personnes autorisées peuvent accéder aux ressources et prévenir les intrusions non autorisées.

L'authentification dans un programme API est un pilier fondamental pour assurer que seuls les utilisateurs et systèmes autorisés accèdent aux ressources. Pour garantir une authentification robuste, plusieurs bonnes pratiques techniques doivent être appliquées.

Tout d'abord, l'utilisation de protocoles standardisés comme OAuth 2.0 ou OpenID Connect est recommandée pour gérer les flux d'autorisation et d'authentification de manière sécurisée. Les jetons JWT (*JSON Web Tokens*) signés et chiffrés peuvent être utilisés pour transmettre des informations d'authentification de manière vérifiable entre les parties.

Ensuite, l'implémentation de mécanismes de multi-factor authentication (MFA) ajoute une couche de sécurité supplémentaire en exigeant une preuve d'identité en plus (par exemple, un code temporaire ou une clé physique). Pour protéger contre les attaques par force brute, des politiques de rate limiting doivent être mises en place sur les endpoints d'authentification.

Enfin, il est crucial de stocker les informations d'identification (comme les mots de passe ou les clés API - *API keys*) de manière sécurisée, en utilisant des fonctions de hachage robustes et en évitant de les exposer dans les logs ou les réponses des API. En combinant ces pratiques, on assure une authentification fiable et sécurisée pour les programmes API.

1.2.2 Autorisation

L'autorisation consiste à définir les ressources et les services auxquels un utilisateur authentifié peut accéder, ainsi que les actions qu'il est autorisé à effectuer. Cette approche repose souvent sur les éléments suivants :

– **Les listes de contrôle d'accès (*Access Control Lists* ou ACL)** : elles permettent de définir les permissions pour les utilisateurs et les groupes, en spécifiant les droits d'accès aux différentes ressources.
– **Les rôles et les politiques** : ils permettent d'attribuer des rôles à des utilisateurs, avec des permissions spécifiques associées à chaque rôle. Cela facilite la gestion des autorisations en regroupant les utilisateurs selon leurs responsabilités et en leur accordant les droits appropriés.

En résumé, l'autorisation est un mécanisme essentiel pour contrôler l'accès aux ressources et services, en garantissant que seules les personnes autorisées puissent effectuer certaines actions.

L'autorisation dans un programme API est essentielle pour contrôler finement les accès aux ressources en fonction des rôles et des permissions des utilisateurs ou des applications.

Pour garantir une autorisation efficace, plusieurs bonnes pratiques techniques doivent être mises en œuvre. Tout d'abord, l'adoption de modèles de contrôle d'accès comme RBAC (*Role-Based Access Control*) ou ABAC (*Attribute-Based Access Control*) permet de définir des politiques d'accès granulaires en fonction des rôles, des attributs ou du contexte de l'utilisateur.

Les scopes et permissions doivent être clairement définis dans les jetons d'accès (*Access Token* OAuth2.0 ou jeton JWT) pour limiter les actions autorisées.

Ensuite, il est crucial de valider systématiquement les permissions à chaque requête API, en s'assurant que l'utilisateur ou l'application a bien le droit d'accéder à la ressource demandée. L'utilisation de policies centralisées (via des solutions intégrées dans les API Gateways) facilite la gestion et l'audit des règles d'autorisation.

1.2.3 Accounting

L'accounting (ou comptabilisation) consiste à suivre et enregistrer les actions réalisées par les utilisateurs ou des applications sur un système.

Son objectif est de permettre :

– **l'audit et la traçabilité** : en conservant une trace des activités effectuées pour une analyse ultérieure ;
– **la détection des anomalies** : en identifiant les comportements suspects ou non conformes ;
– **la génération de rapports** : afin de fournir des informations aux administrateurs et aux auditeurs.

L'accounting dans un programme API consiste à suivre et enregistrer les activités des utilisateurs et des applications pour assurer une traçabilité complète des accès et des actions effectuées.

Pour garantir une gestion efficace de l'accounting, plusieurs bonnes pratiques techniques doivent être mises en œuvre. Tout d'abord, il est essentiel de mettre en place un système de journalisation (*logging*) centralisé et structuré, en utilisant des formats standardisés comme JSON pour enregistrer des informations telles que l'identité de l'utilisateur, l'heure de la requête, l'endpoint appelé, le statut de la réponse et les éventuelles erreurs.

Ces logs doivent être stockés de manière sécurisée et chiffrée pour éviter toute altération ou tout accès non autorisé. Ensuite, l'intégration de solutions de monitoring et d'analyse des logs (comme ELK, Splunk, Prometheus ou Google Cloud Logging et Monitoring comme exemple de solution reposant sur le Cloud) permet de surveiller les activités en temps réel et de détecter les comportements anormaux.

Pour garantir l'intégrité des données, des mécanismes de checksum ou de signature numérique peuvent être appliqués aux logs. Enfin, il est crucial de définir des politiques de rétention des logs conformes aux réglementations (comme le RGPD) et de mettre en place des audits réguliers pour vérifier la conformité et la sécurité des données enregistrées.

1.3 Anonymat (Anonymity)

L'anonymat vise à préserver l'identité des utilisateurs en rendant leurs activités ou leurs communications impossibles à tracer jusqu'à leur personne.

Les méthodes pour garantir l'anonymat incluent :

– les réseaux anonymes tels que Tor, qui permettent de faire transiter les communications par plusieurs points afin de masquer leur origine ;
– le chiffrement, qui protège les données de manière que l'expéditeur ne puisse être identifié ;
– la pseudonymisation, qui consiste à utiliser des pseudonymes pour remplacer les informations d'identification personnelles.

L'anonymat dans un programme API vise à garantir que les utilisateurs ou les systèmes interagissant avec l'API ne peuvent pas être identifiés directement ou indirectement à partir des données échangées.

Pour garantir l'anonymat, plusieurs bonnes pratiques techniques doivent être appliquées. Tout d'abord, la minimisation des données est essentielle : ne collecter et ne transmettre que les informations strictement nécessaires, en évitant les données personnelles identifiables (PII). Ensuite, des techniques de pseudonymisation ou d'anonymisation peuvent être utilisées pour remplacer les identifiants directs (comme les adresses IP ou les ID utilisateur) par des valeurs non réversibles (via des fonctions de hachage comme SHA-256 avec salting) ou des tokens générés de manière aléatoire. Les logs et les métadonnées doivent également être traités pour supprimer ou masquer les informations sensibles. Pour renforcer l'anonymat, des mécanismes comme les proxies ou les VPN peuvent être utilisés pour masquer l'origine des requêtes.

Enfin, il est crucial de mettre en place des politiques de rétention des données limitées et de s'assurer que les données anonymisées ne peuvent pas être réidentifiées par croisement avec d'autres sources.

2. Les risques informatiques liés à la sécurité

À l'ère du numérique, trouver le juste équilibre entre commodité des solutions informatiques (par exemple des applications mobiles ou connectées) et sécurité est devenu un enjeu majeur dans la conception et la mise en place d'architectures Cloud dites modernes.

Cette section explore le défi subtil auquel les organisations et les individus sont confrontés pour garantir à la fois une facilité d'utilisation et une protection solide des données et des systèmes. Avec l'avancée de la technologie, le besoin de solutions pratiques entre souvent en conflit avec la nécessité de mesures de sécurité strictes.

Comprendre ce compromis est essentiel pour prendre des décisions éclairées dans les contextes professionnels, mais aussi personnels.