الجمهورية الجزائرية الديمقراطية الشعبية

**République Algérienne Démocratique et Populaire**

**Ministère de l'enseignement supérieur et de la recherche scientifique**

**Université de 8 Mai 1945 – Guelma -**

**Faculté des Mathématiques, d'Informatique et des Sciences de la matière**

**Département d'Informatique**


# Mémoire de Fin d'études Master

**Filière :** Informatique

**Option : Système Informatique**

**Thème :**

# Etude de méthode de sécurité des bases de données en serveur web

**Encadré Par :**

Dr. Mohammed Chaoui

**Présenté par :**

Salissou Nassirou Djibril

**Juin 2022**

## Résumé

Avec l'évolution d'Internet, les bases de données sont exposées aux attaques compromettant ainsi la sécurité des systèmes et provoquant des dommages très sévères dans une organisation en terme économique, fiabilité et confidentialité. Notre thème étant une étude d'une méthode de sécurité afin de réduire les attaques, d'éviter la perte d'information, la violation et la destruction des données en serveurs web.

Dans ce sens, nous avons conçu un site web pour simuler quelques attaques des bases de données et ainsi implémenter un API de contremesure pour se protéger de ces dits attaques. D'une part, cet API permet de se protéger contre les attaques injections SQL, le Cross-Site Scripting et le Cross-Site Request Forgery, de l'autre il inclut une fonction permettant de vérifier le niveau des mots de passes pour s'assurer une prévention contre le cassage par force brute. L'API peut s'intégrer dans un site web pour protéger la base de données.



**Mots clés :** base de données, serveurs web, sécurité, injection SQL, Cross-Site Scripting, Cross-Site Request Forgery, API.



## Abstract

With the evolution of the Internet, databases are exposed to attacks thus compromising the security of systems and causing very severe damage in an organization in terms of economy, reliability and confidentiality. Our theme is a study of a security method to reduce attacks, avoid the loss of information, the breach and destruction of data in web servers. In this sense, we designed a website to simulate some attacks of the databases and thus implement a countermeasure API to protect against these attacks. On the one hand, this API helps protect against SQL injection attacks, Cross-Site Scripting and Cross-Site Request Forgery, on the other hand it includes a function to check the level of passwords to ensure prevention against brute force breaking. The API can integrate into a website to protect the database.



**Keywords :** database, web servers, security, SQL injection, Cross-Site Scripting, Cross-Site Request Forgery, API.

# Sommaire

# Liste des figures

# Liste des Abréviations et Acronymes

**API** Application programme interface

**BD** Base de données

**CWE** Common Weakness Enumeration

**CSRF** Cross-Site Resquest Forgery

**CSS** Cascade Style Sheet

**HTML** Hyper-Text Mark Language

**HTTP** HyperText Transfert Protocol

**SGBD** Système de Gestion de Base de Données

**PHP** Hypertext Preprocessor

**OWASP** Open Web Application Security Project

**WAF** Web Application Firewall

**WASC** Web Application Security Consortium

**XSS** Cross-Site Scripting

# Remerciements

Nos louanges au tout puissant qui dans sa bonté nous a donné la foi et la force nécessaire pour pouvoir rédiger ce mémoire.

Mes vifs remerciements à mon encadreur Dr. Mohamed Chaoui qui a accepté de me porter sur ses ailes, merci pour les conseils et les encouragements.

Je tiens également à remercier Hamadoun Cissé pour son soutien et son aide pendant toutes ces années passées ensembles.

Merci est certes un petit mot mais rempli de gratitudes alors merci à tous nos professeur(e)s.

A toutes ces personnes qui m'ont aidée d'une quelconque manière, sachez que je vous suis reconnaissante.

# Dédicaces

Je dédie cet humble travail :

A mes chers et respectueux parents.

Vraiment je ne saurais comment vous remercier, aucun mot n'est assez fort pour vous démontrer toute ma gratitude et toute mon affection que je vous porte, au-delà de tous les sacrifices que vous avez consentis envers ma modeste personne pour pouvoir combler mes moindres désirs, je vous dédie ce travail pour remercier amplement et que vous puissiez trouver satisfaction envers ma personne et aussi qu'il soit l'exaucement de tous vos vœux tant formulés et de vos prières quotidiennes.

Puisse le tout puissant vous donner longue vie pour que vous puissiez voir le fruit de ce travail.

A :

Mes frères : Yacine, Mahamadou, Ibrahim et Moussa.

A mes amies les plus chères : Boubacar, Ousmane Zataou, Meddi, Hassane Barmou, Aminou, Gado Riba, Munene Job et Hamadoun Cissé.

En témoignage de ma sincère amitié, veuillez trouver dans ce travail mon profond hommage.

A tous ceux qui m'ont aidé dans l'élaboration de ce travail.

# Introduction générale

Les applications déployées dans les environnements Web ont connu une évolution rapide ces dernières décennies, surtout avec l'avènement de l'IOT (Internet of Things connu sous le nom de Web of Things) et du Web 2.0, autorisant grand nombre d'utilisateurs d'accéder aux données et aux ressources informatiques par les navigateurs Web. En effet, cette ouverture a énormément augmenté la surface d'attaque et la vulnérabilité des serveurs et applications Web, notamment les sites web et les bases de données. Ainsi les techniques d'attaques sur les sites web ou les serveurs de base de données ont particulièrement évolué, donnant naissance à des outils d'attaques automatiques sur le Web (W3brute ou SQLPowerInjector), rendant le risque plus élevé et l'exploitation des failles de sécurité plus automatisées. De plus, les attaques Web sont alors fréquentes sur des entreprises qui pour la plupart n'ont pas sécurisé leurs sites web. Malgré l'existence des dizaines types d'attaques sur le web, notre projet porte sur la simulation des attaques par injection SQL, Cross-Site Scripting, Cross-Site Request Forgery et le renforcement des mots de passes pour prévenir l'attaque par force brute. Ces simulations serviront à mettre en évidence l'implémentation de notre API conçu pour prévenir notre site web contre ces attaques. De plus, chacune de ces attaques peut permettre à l'attaquant de contourner les contrôles d'accès mises en place et accéder à l'application comme un administrateur. Dans le pire des cas, un pirate peut manipuler complètement une base de données.

En général, les développeurs conçoivent les pages Web à partir des entrées d'utilisateurs sans valider s'ils contiennent des caractères suspects. Un attaquant peut donc exploiter des failles de sécurités et injecter des scripts ou des codes SQL malveillants afin de s'emparer du site ou d'exécuter des transactions illégaux. Certaines attaques peuvent réussir compte tenu des facteurs suivants :

- Les développeurs continuent à implanter des applications vulnérables à cause de leur manque d'expérience ou de l'absence de temps pour pouvoir appliquer les bonnes pratiques de codage sécurisé.
- Les pare-feu et les systèmes classiques de détection d'intrusions qui filtrent souvent le trafic réseau ne peuvent pas protéger une base de données contre les attaques par injection SQL.
- Le balayage automatisé de vulnérabilités web génère des rapports de sécurité sans être capable de prévenir certaines attaques après le déploiement des applications web.

Nous tentons dans notre projet de développer notre propre contremesure en considérant le système de sécurité comme une boite noire. Notre système proposé doit donc être capable de protéger un site web et être au moins facilement adaptable aux applications déployées sur les environnements Web.

L'objectif principal est de réaliser un API robuste de protection contre les attaques par injection SQL, Cross-Site Scripting, Cross-Site Request Forgery et le cassage par force brute du mot de passe. L'idée principale est de concevoir un site web pour effectuer la simulation de ces attaques, ensuite fera appel aux mécanismes de sécurités de l'API pour tester les contremesures implémenté.

Ce document est structuré en trois chapitres dont le premier présente une étude des méthodes de sécurités pour les applications Web, ensuite le deuxième chapitre l'étude des méthodes de sécurités dans les bases de données, le dernier chapitre est consacré à la conception et à l'implémentation de notre site web.

Le premier chapitre comporte une brève introduction à la sécurité informatique, ensuite nous parlerons des applications Web ainsi que les différentes structures et enfin les vulnérabilités et menaces auxquels sont confrontées les applications Web.

Le second chapitre quant à lui compose une brève introduction des bases de données et des systèmes de gestion de bases de données, ainsi que les architectures connus des bases de données et des SGBD et enfin les vulnérabilités et sécurités des bases de données.

Le troisième chapitre présente une description de notre site web et de l'API, les langages de programmations utilisés pour les deux, et enfin un ensemble de scénarios pour tester l'API contre les attaques injections SQL, Cross-Site Scripting, Cross-Site Request Forgery.

# I. Chapitre I : Etude des méthodes de sécurités pour les applications Web.

# Introduction

Ce chapitre présente le contexte de notre mémoire dans lequel plusieurs concepts liées à la sécurité informatique, les applications Web et la sécurité des applications Web sont brièvement présentés.

Nous introduisons tout d'abord quelques aspects de la sécurité informatique. Ensuite, nous présentons les applications Web et leurs architectures. Enfin, nous détaillons la sécurité des applications Web.

## I.1. Introduction à la Sécurité Informatique

La sécurité informatique consiste à mettre en place une politique pour assurer la sécurité du système informatique [1] [2]. La sécurité informatique est aussi l'ensemble des techniques qui assurent que les ressources du système d'information (matérielles ou logiciel) d'une organisation sont exploitées uniquement dans le cadre ou il est prévu qu'elle le soit [3]. Le système d'information est défini par l'ensemble des données et ressources matérielles et logicielles de l'entreprise permettant de les stocker ou de les faire circuler [3]. Il représente un patrimoine essentiel de l'entreprise qu'il faut protéger.

Les objectifs suivants déterminent les enjeux principaux de la sécurité informatique [1] :

- **Intégrité :** Les données ne doivent pas être altérées par un utilisateur malveillant [2] [4].
- **Confidentialité :** assurer que les données confidentielles ne soient accessibles, en lecture, que par les utilisateurs légitimes [2] [4].
- **Disponibilité :** garantir que les ressources informatiques et les données soient toujours disponibles pour les utilisateurs légitimes [2] [4].
- **Authentification :** l'autorisation à une entité (personne, ordinateur, etc.) qui a le droit d'accéder à des ressources informatiques [2] [4].
- **La non-répudiation :** garantir qu'aucun des correspondants ne pourra nier la transaction [4].

Les aspects suivants sont souvent utilisés lorsque nous parlons des attaques :

- **Préjudice :** « une perte ou un dommage possible dans un système informatique. » [4]
- **Vulnérabilité :** « une faiblesse dans un système qui pourrait être exploitée pour causer des préjudices. » [4]
- **Menace :** « ensemble de circonstances pouvant conduire à des préjudices. » [4]

## I.2. Les Applications Web

La majorité des nouveaux projets informatiques sont aujourd'hui des applications web qui sont devenues une solution universelle d'utilisation grâce à de nombreux avantages [1] :

- Une application web est accessible par un navigateur Web sans déployer aucun autre logiciel sur le poste client et les mises à jour sont simplifiées puisque l'application est centralisée sur un serveur.
- La gratuité des serveurs web, comme le serveur http de la fondation Apache (Apache) et le serveur Web de Microsoft (IIS), encourage les entreprises à mettre en place leurs applications Web afin d'augmenter leurs clientèles.
- La simplicité des plateformes, des outils et des langages de programmation permet aux experts et inexperts de développer en mettant peu d'efforts toutes sortes d'applications Web.

### I.2.1 Architecture des applications Web

Souvent, une application web est basée sur une architecture client-serveur [5] [6] qui comprend un client Web, un serveur Web sur lequel l'application web est installée, et un serveur de bases de données (***figure 1.1***).

Ces composants communiquent entre eux comme suit :

- Le client Web envoie une demande HTTP à l'application Web qui lui retourne une page Web.
- L'application Web envoie une requête SQL au serveur de données qui lui retourne des données.

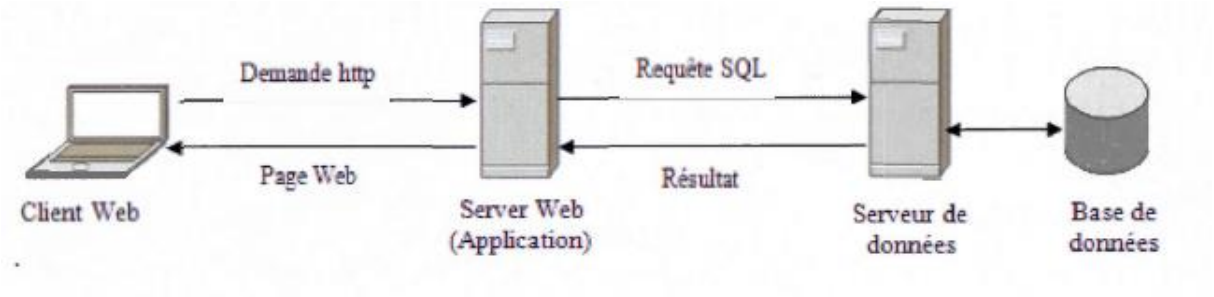


***Figure 1.1 :*** *Architecture d'une application Web [1].*

Les données saisies par l'utilisateur sont envoyées à l'application web par les deux fonctions POST et GET [7] [8] :

- « POST », illustrée dans exemple 1, envoie au serveur Web les paires nom/valeur des

entrées de l'utilisateur dans le corps du message de demande HTTP.

- « GET », illustrée dans exemple 2, envoie u serveur Web une chaîne des paires nom/valeur et ajoute ensuite la chaîne de requête dans le corps du message de demande HTTP.

```
POST /Default.aspx http/1.1
userID=admin
password=123
Accept : image/gif, image/x-xbitmap, image/jpeg, …
Accept-language : en-us
Accept-Encoding : gzip, deflate
User-Agent : Mozilla/4.0 …
Host : localhost
Cookie : ASP.NET_SessionId=xshitibm0r1nlpawjvwfzn55
Connection : Keep-Alive
```

Exemple 1 : d'une demande HTTP en utilisant la méthode POST [1].

```
Get / Default.aspx?user ID=admin&password=l23 HTTP /1.1
Accept: image/gif, image/x-xbitmap, imagef jpeg, .. .
Accept-Language: en - us
Accept - Encoding: gzip, deftate
User-Agent: Mozilla/4.0 .. .
Host: localhost
Cookie: ASP.NET _Sessionld=xshitibm0rlnlpawjvwfzn55
Connection: Keep-Alive
```

Exemple 2 : d'une demande HTTP en utilisant la méthode GET [1].

La méthode POST est plus sécurisée que la méthode GET qui fait passer les variables par l'adresse d'une page Web (URL) ; ces variables peuvent être facilement détournées.
L'architecture d'un système peut supprimer certains problèmes techniques ou isoler certaines préoccupations, rendant ainsi le système moins complexe, plus facile à comprendre et faire évoluer. Dans le cas des applications Web, nous distinguons deux niveaux d'architecture :

- **L'architecture matérielle :** correspond à la disposition matérielle des machines reliées entre elles. Il s'agit d'une architecture centralisée ou plusieurs clients se connecte à un même serveur pour accéder à ses services.

- **L'architecture logicielle :** correspond à l'organisation du code d'une application donnée, elle décrit la façon dont les différentes parties du code s'assemble et leurs responsabilités respectives.

### I.2.2 De l'architecture 1-tiers au n-tiers

L'architecture des applications Web a suivi une évolution pour rendre plus simple la construction des applications web. De plus, l'architecture des applications Web possède 3 niveaux d'abstractions :

- **La couche de présentation :** présentation des informations et interactions avec l'utilisateur.
- **La couche traitement :** regroupe l'ensemble des travaux à réaliser pour l'application dont les traitements locaux (contrôle du dialogue) et les traitements globaux (application).
- **La couche de données :** regroupe l'ensemble des mécanismes permettant la gestion et le stockage des informations.

#### I.2.2.a L'architecture 1-tiers

Pour l'architecture 1-tiers, les trois couches s'exécutent sur le même serveur, donc le système entier est centralisé (maître/esclave). De plus, il offre une simplicité d'administration et la centralisation des données mais cela surcharge le serveur dû à l'interface et la montée de charge. La solution à ce problème est d'alléger le serveur, d'où l'évolution vers l'architecture 2-tiers [9].

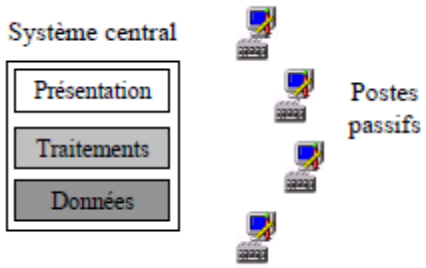


**Figure 1.2 :** *Système centralisée d'un serveur web (Architecture 1-tiers) [9].*

#### I.2.2.b L'architecture 2-tiers

L'architecture 2-tiers est de type client-serveur de données, le client regroupe les couches présentation et traitements, le serveur regroupe la couche gestion des données. L'architecture de base est l'architecture 2-tier et la plus répandue. Cette architecture s'appuie sur un poste central, qui envoie les données aux machines clientes : premièrement l'utilisateur émet une

requête HTTP vers le serveur sur lequel est stockée la page HTML. Ensuite, le serveur accède alors à ce fichier et le retourne au navigateur. Enfin, le navigateur interprète les balises HTML et affiche la page en résultat. L'architecture 2-tiers offre une interface plus riche et les applications sont sur le poste client. Néanmoins, elle sollicite beaucoup le poste client, dialogue avec le serveur est très important. Les clients et les protocoles non standards rendent la maintenance difficile. Aussi, la relation étroite entre le programme client et organisation des données côtés serveur est l'origine de la difficulté d'évolution des applications web basée architecture 2-tiers. La solution à ces problèmes est d'alléger le côté client et de rendre le système plus souple, d'où l'évolution vers l'architecture 3-tiers [9].

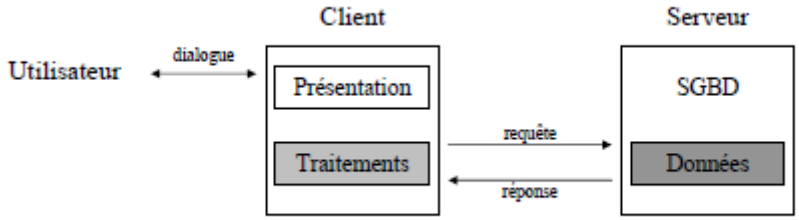


**Figure 1.3 :** *Système client-serveur (Architecture 2-tiers) [9].*

### I.2.2.c L'architecture 3-tiers

Le principe de l'architecture 3-tiers repose sur le fait que le client gère la présentation et les traitements locaux (moins lourd, moins évolué, moins coûteux), ainsi le serveur d'application gère les traitements (Apache, PHP) et le serveur de données accueille un SGBD (Oracle, MySQL). L'architecture 3-tiers est caractérisée par l'indépendance des 3 niveaux, donc une implantation sur différentes machines et une évolution plus facile. Ainsi, la répartition des tâches est homogène et la montée en charge plus facile à prendre en compte.

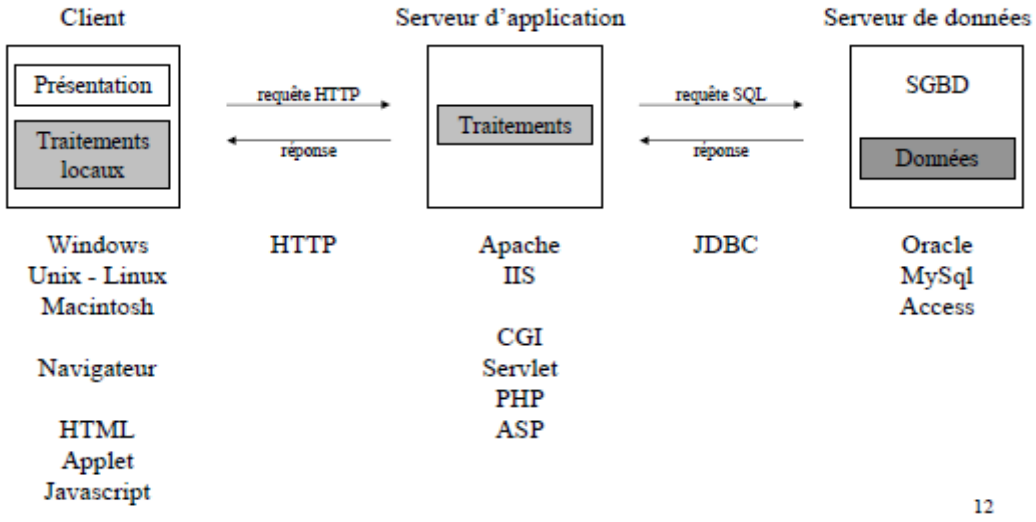


***Figure 1.4 :*** *Système client-server-BDD (architecture 3-tiers) [9].*

Les difficultés rencontrées avec les architectures 2-tiers ont conduit l'arrivée des premiers standards Internet dont HTML (description des pages et texte + balise de mise en forme), HTTP (GET, POST, PUT, TRACE, DELETE), TCP/IP (protocole de communication réseau, couche réseau et couche transport) et CGI (langage de script, interface entre un programme exécutable et un serveur web).

Quant à l'architecture n-tiers, elle regroupe, par exemple, plusieurs serveurs d'application (serveur html + moteur servlets, répartition de la logique d'application dans des objets métiers) et les bases de données distribuées. Elle est plus facile à faire évoluer et offre une montée de charge rapide mais cette architecture rend le système complexe et des coûts élevées pour sa mise en œuvre [9].

## I.3. La sécurité des applications Web

### I.3.1 Menaces et risques applicatifs

Le WASC établie dans son rapport « WASC Threat Classification » une liste exhaustive des menaces qui pèsent sur la sécurité des applications Web. Elles sont regroupées dans six catégories comme suit [1] :

- La catégorie « authentification » regroupe les attaques de sites Web dont la cible est le système de validation de l'identité d'un utilisateur, d'un service ou d'une application.
- La catégorie « autorisation » couvre l'ensemble des attaques de sites Web dont la cible est le système de vérification des droits d'un utilisateur, d'un service ou d'une application pour effectuer une action dans l'application.

- La catégorie « attaques côté client » rassemble les attaques visant l'utilisateur pendant qu'il utilise l'application.
- La catégorie « exécution des commandes » englobe toutes les attaques qui permettent d'exécuter des commandes sur un des composants de l'architecture du site Web.
- La catégorie « révélation d'informations » définit l'ensemble des attaques permettant de découvrir des informations ou des fonctionnalités cachées.
- La catégorie « attaques logiques » caractérise les attaques qui utilisent les processus applicatifs (système de changement de mot de passe, système de création de compte, …) à des fins hostiles.

Contrairement au WASC qui décrit toutes les attaques possibles sur une application Web, OWASP ne traite que les dix plus grands risques de sécurité. Le rapport « OWASP Top 10 » permet ainsi à l'équipe projet de se focaliser sur la protection de l'application face aux menaces les plus importantes, ce qui est moins coûteux et plus facilement réalisable que d'essayer de se protéger de tous les dangers.

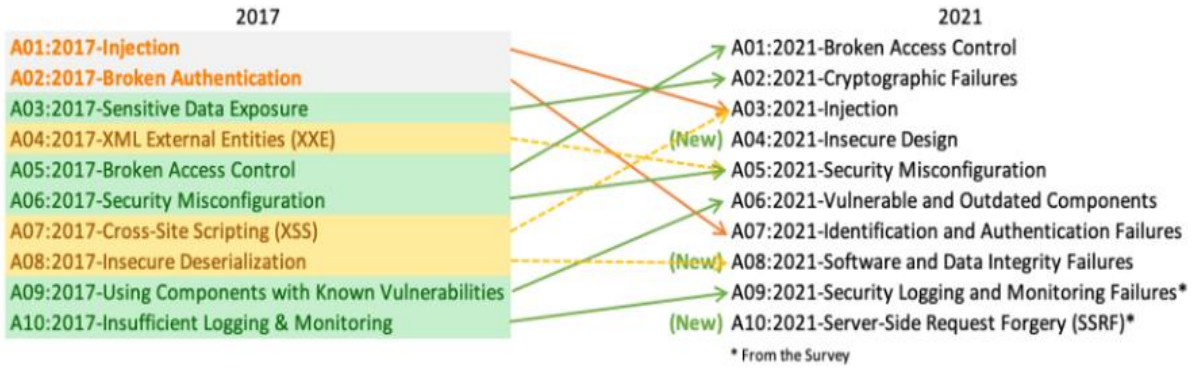


***Figure 1.5 :*** *Liste des risques de sécurité par OWASP 2017-2021 [10].*

Nous présentons d'abord un aperçu des différentes attaques (***figure 1.5***), ensuite nous donnerons dans les sections suivantes une description des attaques.

- **A01 Contrôles d'accès défaillants (Broken Access Control) [10]** : En passant de la cinquième position, 94% des applications ont été testées pour une forme de contrôle d'accès cassé avec un taux d'incidence moyen de 3.81%, et ont le plus d'occurrences dans l'ensemble de données fourni avec plus de 318 000. Les énumérations de faiblesses communes (CWE) notables incluses sont CWE-200 : exposition d'informations sensibles à un acteur non autorisé, CWE-201 : exposition d'informations sensibles par le biais de données envoyées et CWE-352 : falsification de requête inter-site.

- **A02 Défaillances cryptographiques (Cryptographic Failures) [11]** : En passant d'une position à la deuxième, anciennement connue sous le nom d'exposition aux données sensibles, qui est plus un symptôme général qu'une cause profonde, l'accent est mis sur les défaillances liées à la cryptographie (ou son absence). Ce qui conduit souvent à l'exposition de données sensibles. Les énumérations de faiblesses communes (CWE) notables incluses sont CWE-259 : Utilisation d'un mot de passe codé en dur, CWE-327 : Algorithme cryptographique cassé ou risqué et CWE-331 Entropie insuffisante.
- **A03 Injection [12]** : L'injection glisse jusqu'à la troisième position. 94 % des applications ont été testées pour une forme d'injection avec un taux d'incidence maximal de 19 %, un taux d'incidence moyen de 3 % et 274 000 occurrences. Les énumérations de faiblesses communes (CWE) notables incluses sont CWE-79 : Cross-site Scripting, CWE-89 : SQL Injection et CWE-73 : External Control of File Name or Path.
- **A04 Conception non sécurisée (Insecure design) [13]** : Une nouvelle catégorie pour 2021 se concentre sur les risques liés aux défauts de conception et d'architecture, avec un appel à une utilisation accrue de la modélisation des menaces, des modèles de conception sécurisés et des architectures de référence. Les énumérations de faiblesses communes (CWE) notables incluent CWE-209 : Génération de messages d'erreur contenant des informations sensibles, CWE-256 : Stockage non protégé d'informations d'identification, CWE-501 : Violation des limites de confiance et CWE-522 : Informations d'identification insuffisamment protégées.
- **A05 Mauvaise configuration de sécurité (Sécurity Misconfiguration) [14]** : Passant de #6 dans l'édition précédente, 90% des applications ont été testées pour une forme de mauvaise configuration, avec un taux d'incidence moyen de 4.%, et plus de 208 000 occurrences d'une énumération des faiblesses communes (CWE) dans cette catégorie de risque. Avec de plus en plus d'évolutions vers des logiciels hautement configurables, il n'est pas surprenant de voir cette catégorie monter en puissance. Les principaux CWE inclus sont CWE-16 Configuration et CWE-611 Improper Restriction of XML External Entity Reference.
- **A06 Composants vulnérables et obsolètes (Vulnerable and Outdated Components) [15]** : Il était n ° 2 du Top 10 de l'enquête communautaire, mais disposait également de suffisamment de données pour figurer dans le Top 10 via les données. Les composants vulnérables sont un problème connu que nous avons du mal à tester et à évaluer les risques. Il s'agit de la seule catégorie à ne pas avoir d'énumérations de faiblesses

communes (CWE) mappées aux CWE inclus. Par conséquent, un poids d'exploitation/d'impact par défaut de 5,0 est utilisé. Les CWE notables inclus sont CWE-1104 : Utilisation de composants tiers non maintenus et les deux CWE du Top 10 2013 et 2017.

- **A07 Echec d'identification et d'authentification (Identification and Authentification Failures) [16]** : Anciennement connue sous le nom d'authentification brisée, cette catégorie est passée de la deuxième position et comprend désormais les énumérations de faiblesses communes (CWE) liées aux échecs d'identification. Les principaux CWE inclus sont CWE-297 : Validation incorrecte du certificat avec incompatibilité d'hôte, CWE-287 : Authentification incorrecte et CWE-384 : Fixation de session.
- **A08 Echec de l'intégrité du logiciel et des données (Software and Data Intégrity Failures) [17]** : Une nouvelle catégorie pour 2021 se concentre sur la formulation d'hypothèses liées aux mises à jour logicielles, aux données critiques et aux pipelines CI/CD sans vérifier l'intégrité. L'un des impacts les plus pondérés des données CVE/CVSS (Common Vulnerability and Exposures/Common Vulnerability Scoring System). Les énumérations de faiblesses communes (CWE) notables incluent CWE-829 : Inclusion of Functionality from Untrusted Control Sphere, CWE-494 : Download of Code Without Integrity Check et CWE-502 : Deserialization of Untrusted Data.
- **A09 Echec de journalisation et de surveillance de la sécurité (Security Logging and Monitoring Failures) [18]** : La journalisation et la surveillance de la sécurité proviennent de l'enquête communautaire Top 10 (#3), en légère hausse par rapport à la dixième position du Top 10 OWASP 2017. La journalisation et la surveillance peuvent être difficiles à tester, impliquant souvent des entretiens ou demandant si des attaques ont été détectées lors d'une pénétration. test. Il n'y a pas beaucoup de données CVE/CVSS pour cette catégorie, mais la détection et la réponse aux violations sont essentielles. Pourtant, cela peut avoir un impact considérable sur la responsabilité, la visibilité, l'alerte d'incident et la criminalistique. Cette catégorie s'étend au-delà de CWE-778 Journalisation insuffisante pour inclure CWE-117 Neutralisation de sortie incorrecte pour les journaux, CWE-223 Omission d'informations relatives à la sécurité et CWE-532 Insertion d'informations sensibles dans le fichier journal.

- **A10 Contrefaçon de demande côté serveur (Server-Side Request Forgery) [19] :** Cette catégorie est ajoutée à partir de l'enquête communautaire Top 10 (#1). Les données montrent un taux d'incidence relativement faible avec une couverture de test supérieure à la moyenne et des cotes potentielles d'exploit et d'impact supérieures à la moyenne. Comme les nouvelles entrées sont susceptibles d'être un seul ou un petit groupe d'énumérations de faiblesses communes (CWE) pour l'attention et la sensibilisation, l'espoir est qu'elles soient sujettes à une attention particulière et puissent être regroupées dans une catégorie plus large dans une future édition.

Dans ce qui suit, nous présentons une description, des exemples de scénarios et quelques bonnes pratiques pour se prévenir des attaques aperçus ci-dessus.

### I.3.2 Contrôle d'accès défaillants

#### a. Description

Le contrôle d'accès applique une politique telle que les utilisateurs ne peuvent pas agir en dehors de leurs autorisations prévues. Les défaillances entraînent généralement la divulgation non autorisée d'informations, la modification ou la destruction de toutes les données ou l'exécution d'une fonction commerciale en dehors des limites de l'utilisateur. Les vulnérabilités courantes du contrôle d'accès incluent :

- Violation du principe du moindre privilège ou refus par défaut, où l'accès ne doit être accordé qu'à des capacités, des rôles ou des utilisateurs particuliers, mais est accessible à tous.
- Contourner les contrôles d'accès en modifiant l'URL (falsification des paramètres ou navigation forcée), l'état de l'application interne ou la page HTML, ou en utilisant un outil d'attaque modifiant les requêtes API.
- Autoriser l'affichage ou la modification du compte de quelqu'un d'autre, en fournissant son identifiant unique (références d'objet directes non sécurisées)
- Accès à l'API avec des contrôles d'accès manquants pour POST, PUT et DELETE.
- Élévation de privilège. Agir en tant qu'utilisateur sans être connecté ou agir en tant qu'administrateur lorsqu'il est connecté en tant qu'utilisateur.
- La manipulation des métadonnées, telle que la relecture ou la falsification d'un jeton de contrôle d'accès JSON Web Token (JWT), ou un cookie ou un champ masqué manipulé pour élever les privilèges ou abuser de l'invalidation JWT.

- Une mauvaise configuration CORS permet l'accès à l'API à partir d'origines non autorisées/non approuvées.
- Forcez la navigation vers des pages authentifiées en tant qu'utilisateur non authentifié ou vers des pages privilégiées en tant qu'utilisateur standard.

### b. Exemples de scénarios d'attaques

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

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

Un attaquant modifie simplement le paramètre 'acct' du navigateur pour envoyer le numéro de compte de son choix. S'il n'est pas correctement vérifié, l'attaquant peut accéder au compte de n'importe quel utilisateur.

*https://example.com/app/accountInfo?acct=notmyacct*

**Scénario #2 :** Un attaquant force simplement les navigations vers les URL cibles. Les droits d'administrateur sont requis pour accéder à la page d'administration.

*https://example.com/app/getappInfo*
*https://example.com/app/admin_getappInfo*

Si un utilisateur non authentifié peut accéder à l'une ou l'autre des pages, c'est un défaut. Si un non-administrateur peut accéder à la page d'administration, il s'agit d'un défaut.

### c. Prévention

Le contrôle d'accès n'est efficace que dans le code côté serveur de confiance ou l'API sans serveur, où l'attaquant ne peut pas modifier la vérification du contrôle d'accès ou les métadonnées.

- Sauf pour les ressources publiques, refuser par défaut.
- Implémentez des mécanismes de contrôle d'accès une seule fois et réutilisez-les dans l'ensemble de l'application, notamment en minimisant l'utilisation du partage de ressources cross-origin (CORS).

- Les contrôles d'accès modèles doivent appliquer la propriété de l'enregistrement plutôt que d'accepter que l'utilisateur puisse créer, lire, mettre à jour ou supprimer n'importe quel enregistrement.
- Les exigences de limite commerciale d'application unique doivent être appliquées par les modèles de domaine.
- Désactivez la liste des répertoires du serveur Web et assurez-vous que les métadonnées des fichiers (par exemple, .git) et les fichiers de sauvegarde ne sont pas présents dans les racines Web.
- Consigner les échecs de contrôle d'accès, alerter les administrateurs le cas échéant (par exemple, échecs répétés).
- Limitez le débit de l'API et de l'accès au contrôleur pour minimiser les dommages causés par les outils d'attaque automatisés.
- Les identifiants de session avec état doivent être invalidés sur le serveur après la déconnexion. Les jetons JWT sans état devraient plutôt être de courte durée afin que la fenêtre d'opportunité pour un attaquant soit minimisée. Pour les JWT de plus longue durée, il est fortement recommandé de suivre les normes OAuth pour révoquer l'accès.

Les développeurs et le personnel d'assurance qualité doivent inclure une unité de contrôle d'accès fonctionnel et des tests d'intégration.

### I.3.3 Défaillances cryptographiques

#### a. Description

La première chose est de déterminer les besoins de protection des données en transit et au repos. Par exemple, les mots de passe, les numéros de carte de crédit, les dossiers médicaux, les informations personnelles et les secrets commerciaux nécessitent une protection supplémentaire, principalement si ces données relèvent des lois sur la confidentialité, par exemple, le règlement général sur la protection des données (RGPD) de l'UE, ou des réglementations, par exemple, la protection des données financières. Comme la norme de sécurité des données PCI (PCI DSS). Pour toutes ces données :

- Des données sont-elles transmises en texte clair ? Cela concerne les protocoles tels que HTTP, SMTP, FTP utilisant également des mises à jour TLS comme STARTTLS. Le trafic Internet externe est dangereux. Vérifiez tout le trafic interne, par exemple entre les équilibreurs de charge, les serveurs Web ou les systèmes principaux.
- Des algorithmes ou protocoles cryptographiques anciens ou faibles sont-ils utilisés par défaut ou dans un code plus ancien ?
- Des clés de chiffrement par défaut sont-elles utilisées, des clés de chiffrement faibles sont-elles générées ou réutilisées, ou manque-t-il une gestion ou une rotation appropriée des clés ? Les clés de chiffrement sont-elles archivées dans les référentiels de code source ?
- Le cryptage n'est-il pas appliqué, par exemple, des directives de sécurité ou des en-têtes HTTP (navigateur) sont-ils manquants ?
- Le certificat de serveur reçu et la chaîne de confiance sont-ils correctement validés ?
- Les vecteurs d'initialisation sont-ils ignorés, réutilisés ou générés insuffisamment sécurisés pour le mode de fonctionnement cryptographique ? Un mode de fonctionnement non sécurisé tel que l'ECB est-il utilisé ? Le cryptage est-il utilisé lorsque le cryptage authentifié est plus approprié ?
- Les mots de passe sont-ils utilisés comme clés cryptographiques en l'absence d'une fonction de dérivation de la clé de base du mot de passe ?
- Le caractère aléatoire est-il utilisé à des fins cryptographiques qui n'ont pas été conçues pour répondre aux exigences cryptographiques ? Même si la fonction correcte est choisie, doit-elle être ensemencée par le développeur, et si ce n'est pas le cas, le développeur a-t-il écrasé la fonctionnalité d'amorçage forte intégrée avec une graine qui manque d'entropie/imprévisibilité suffisante ?
- Des fonctions de hachage obsolètes telles que MD5 ou SHA1 sont-elles utilisées, ou des fonctions de hachage non cryptographiques sont-elles utilisées lorsque des fonctions de hachage cryptographiques sont nécessaires ?
- Des méthodes de remplissage cryptographique obsolètes telles que PKCS numéro 1 v1.5 sont-elles utilisées ?
- Les messages d'erreur cryptographiques ou les informations de canal auxiliaire sont-ils exploitables, par exemple sous la forme d'attaques oracle de remplissage ?

### b. Exemples de scénarios d’attaques

**Scénario #1** : Une application crypte les numéros de carte de crédit dans une base de données en utilisant le cryptage automatique de la base de données. Cependant, ces données sont automatiquement décryptées lors de leur récupération, permettant à une faille d'injection SQL de récupérer les numéros de carte de crédit en texte clair.

**Scénario #2** : Un site n'utilise pas ou n'applique pas TLS pour toutes les pages ou prend en charge un cryptage faible. Un attaquant surveille le trafic réseau (par exemple, sur un réseau sans fil non sécurisé), rétrograde les connexions de HTTPS à HTTP, intercepte les requêtes et vole le cookie de session de l'utilisateur. L'attaquant rejoue ensuite ce cookie et détourne la session (authentifiée) de l'utilisateur, accédant ou modifiant les données privées de l'utilisateur. Au lieu de ce qui précède, ils pourraient modifier toutes les données transportées, par exemple, le destinataire d'un transfert d'argent.

**Scénario #3** : La base de données de mots de passe utilise des hachages non salés ou simples pour stocker les mots de passe de tout le monde. Une faille de téléchargement de fichier permet à un attaquant de récupérer la base de données de mots de passe. Tous les hachages non salés peuvent être exposés avec une table arc-en-ciel de hachages pré-calculés. Les hachages générés par des fonctions de hachage simples ou rapides peuvent être piratés par les GPU, même s'ils ont été salés.

### c. Prévention

Procédez comme suit, au minimum :

- Classer les données traitées, stockées ou transmises par une application. Identifiez les données sensibles selon les lois sur la confidentialité, les exigences réglementaires ou les besoins de l'entreprise.
- Ne stockez pas de données sensibles inutilement. Jetez-le dès que possible ou utilisez la tokenisation conforme à la norme PCI DSS ou même la troncature. Les données non conservées ne peuvent pas être volées.
- Assurez-vous de chiffrer toutes les données sensibles au repos.

- S'assurer que des algorithmes, des protocoles et de la clé standard à jour et solides sont en place ; utiliser une bonne gestion des clés.
- Chiffrez toutes les données en transit avec des protocoles sécurisés tels que TLS avec des chiffrements de confidentialité persistante (FS), la hiérarchisation des chiffrements par le serveur et des paramètres sécurisés. Appliquez le chiffrement à l'aide de directives telles que HTTP Strict Transport Security (HSTS).
- Désactivez la mise en cache pour les réponses contenant des données sensibles.
- Appliquer les contrôles de sécurité requis selon la classification des données.
- N'utilisez pas de protocoles hérités tels que FTP et SMTP pour le transport de données sensibles.
- Stockez les mots de passe à l'aide de fonctions de hachage adaptatives et salées fortes avec un facteur de travail (facteur de retard), comme Argon2, scrypt, bcrypt ou PBKDF2.
- Les vecteurs d'initialisation doivent être choisis en fonction du mode de fonctionnement. Pour de nombreux modes, cela signifie utiliser un CSPRNG (générateur de nombres pseudo-aléatoires cryptographiquement sécurisé). Pour les modes qui nécessitent un nonce, le vecteur d'initialisation (IV) n'a pas besoin d'un CSPRNG. Dans tous les cas, le IV ne doit jamais être utilisé deux fois pour une clé fixe.
- Utilisez toujours un cryptage authentifié au lieu d'un simple cryptage.
- Les clés doivent être générées cryptographiquement de manière aléatoire et stockées en mémoire sous forme de tableaux d'octets. Si un mot de passe est utilisé, il doit être converti en clé via une fonction de dérivation de clé de base de mot de passe appropriée.
- Assurez-vous que l'aléatoire cryptographique est utilisé le cas échéant et qu'il n'a pas été ensemencé de manière prévisible ou avec une faible entropie. La plupart des API modernes n'exigent pas que le développeur amorce le CSPRNG pour obtenir la sécurité.
- Évitez les fonctions cryptographiques obsolètes et les schémas de remplissage, tels que MD5, SHA1, PKCS numéro 1 v1.5.
- Vérifier indépendamment l'efficacité de la configuration et des paramètres.

### I.3.4 Injection

#### a. Description

Une application est vulnérable aux attaques lorsque :

- Les données fournies par l'utilisateur ne sont pas validées, filtrées ou nettoyées par l'application.
- Les requêtes dynamiques ou les appels non paramétrés sans échappement contextuel sont utilisés directement dans l'interpréteur.
- Les données hostiles sont utilisées dans les paramètres de recherche de mappage objet-relationnel (ORM) pour extraire des enregistrements sensibles supplémentaires.
- Les données hostiles sont directement utilisées ou concaténées. Le SQL ou la commande contient la structure et les données malveillantes dans les requêtes dynamiques, les commandes ou les procédures stockées.

Certaines des injections les plus courantes sont SQL, NoSQL, la commande du système d'exploitation, le mappage relationnel d'objet (ORM), LDAP et l'injection de langage d'expression (EL) ou de bibliothèque de navigation de graphe d'objet (OGNL). Le concept est identique chez tous les interprètes. L'examen du code source est la meilleure méthode pour détecter si les applications sont vulnérables aux injections. Le test automatisé de tous les paramètres, en-têtes, URL, cookies, entrées de données JSON, SOAP et XML est fortement encouragé. Les organisations peuvent inclure des outils de test de sécurité des applications statiques (SAST), dynamiques (DAST) et interactifs (IAST) dans le pipeline CI/CD pour identifier les failles d'injection introduites avant le déploiement en production.

#### b. Exemples de scénarios d’attaques

**Scénario #1 :** Une 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") + "'";
```

**Scénario #2 :** De même, la confiance aveugle d'une application dans les frameworks peut entraîner des requêtes encore 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 la valeur du paramètre 'id' dans son navigateur pour envoyer : ' ou '1'='1. Par exemple:

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

Cela modifie la signification des deux requêtes pour renvoyer tous les enregistrements de la table des comptes. Des attaques plus dangereuses pourraient modifier ou supprimer des données ou même invoquer des procédures stockées.

### c. Prévention

Empêcher l'injection nécessite de séparer les données des commandes et des requêtes :

- L'option préférée consiste à utiliser une API sécurisée, qui évite d'utiliser entièrement l'interpréteur, fournit une interface paramétrée ou migre vers des outils de mappage relationnel objet (ORM).
  **Remarque :** même lorsqu'elles sont paramétrées, les procédures stockées peuvent toujours introduire une injection SQL si PL/SQL ou T-SQL concatène des requêtes et des données ou exécute des données hostiles avec EXECUTE IMMEDIATE ou exec().
- Utilisez la validation positive des entrées côté serveur. Ce n'est pas une défense complète car de nombreuses applications nécessitent des caractères spéciaux, tels que des zones de texte ou des API pour les applications mobiles.
- Pour toute requête dynamique résiduelle, échappez les caractères spéciaux à l'aide de la syntaxe d'échappement spécifique à cet interpréteur.
  **Remarque :** les structures SQL telles que les noms de table, les noms de colonne, etc. ne peuvent pas être échappées et les noms de structure fournis par l'utilisateur sont donc dangereux. Il s'agit d'un problème courant dans les logiciels de rédaction de rapports.
- Utilisez LIMIT et d'autres contrôles SQL dans les requêtes pour empêcher la divulgation massive d'enregistrements en cas d'injection SQL.

### I.3.5 Conception non sécurisée

#### a. Description

La conception non sécurisée est une vaste catégorie représentant différentes faiblesses, exprimées comme « conception de contrôle manquante ou inefficace ». La conception non sécurisée n'est pas la source de toutes les autres catégories de risques du Top 10. Il y a une différence entre une conception non sécurisée et une implémentation non sécurisée. Nous différencions les défauts de conception et les défauts de mise en œuvre pour une raison, ils ont des causes profondes et des solutions différentes. Une conception sécurisée peut toujours présenter des défauts de mise en œuvre conduisant à des vulnérabilités susceptibles d'être exploitées. Une conception non sécurisée ne peut pas être corrigée par une implémentation parfaite car, par définition, les contrôles de sécurité nécessaires n'ont jamais été créés pour se défendre contre des attaques spécifiques. L'un des facteurs qui contribuent à une conception non sécurisée est le manque de profilage des risques commerciaux inhérent au logiciel ou au système en cours de développement.

**Gestion des exigences et des ressources**

Recueillez et négociez les exigences commerciales pour une application avec l'entreprise, y compris les exigences de protection concernant la confidentialité, l'intégrité, la disponibilité et l'authenticité de tous les actifs de données et la logique commerciale attendue. Tenez compte du degré d'exposition de votre application et si vous avez besoin d'une séparation des locataires (en plus du contrôle d'accès). Compiler les exigences techniques, y compris les exigences de sécurité fonctionnelles et non fonctionnelles. Planifier et négocier le budget couvrant toutes les activités de conception, de construction, de test et d'exploitation, y compris les activités de sécurité.

**Conception sécurisée**

La conception sécurisée est une culture et une méthodologie qui évaluent en permanence les menaces et garantissent que le code est conçu et testé de manière robuste pour empêcher les méthodes d'attaque connues. La modélisation des menaces doit être intégrée dans les séances de perfectionnement (ou activités similaires) ; recherchez les changements dans les flux de données et le contrôle d'accès ou d'autres contrôles de sécurité.

Dans le développement de la user story, déterminez le flux correct et les états d'échec, assurez-vous qu'ils sont bien compris et acceptés par les parties responsables et concernées. Analysez les hypothèses et les conditions des flux attendus et d'échec, assurez-vous qu'ils sont toujours précis et souhaitables. Déterminez comment valider les hypothèses et appliquer les conditions nécessaires pour des comportements appropriés. Assurez-vous que les résultats sont documentés dans la user story. Apprenez de vos erreurs et offrez des incitations positives pour promouvoir les améliorations.

**Cycle de développement sécurisé**

Les logiciels sécurisés nécessitent un cycle de vie de développement sécurisé, une certaine forme de modèle de conception sécurisé, une méthodologie de route pavée, une bibliothèque de composants sécurisés, des outils et une modélisation des menaces. Contactez vos spécialistes de la sécurité au début d'un projet logiciel tout au long du projet et de la maintenance de votre logiciel. Envisagez d'utiliser le modèle SAMM (Software Assurance Maturity Model) de l'OWASP pour vous aider à structurer vos efforts de développement de logiciels sécurisés.

### b. Exemples de scénarios d'attaques

**Scénario #1 :** Un flux de travail de récupération d'informations d'identification peut inclure des « questions et réponses », ce qui est interdit par le NIST 800-63b, l'OWASP ASVS et le Top 10 de l'OWASP. Les questions et les réponses ne peuvent pas être considérées comme une preuve d'identité en tant que plusieurs personnes mais peuvent connaître les réponses, c'est pourquoi elles sont interdites. Ce code doit être supprimé et remplacé par une conception plus sécurisée.

**Scénario #2 :** Une chaîne de cinéma permet des réductions sur les réservations de groupe et a un maximum de quinze spectateurs avant d'exiger un acompte. Les attaquants pourraient menacer de modéliser ce flux et tester s'ils pouvaient réserver six cents places et tous les cinémas à la fois en quelques requêtes, entraînant une perte de revenus massive.

**Scénario n°3 :** Le site Web de commerce électronique d'une chaîne de magasins n'est pas protégé contre les robots gérés par des revendeurs qui achètent des cartes vidéo haut de gamme pour revendre des sites d'enchères. Cela crée une publicité terrible pour les fabricants de cartes vidéo et les propriétaires de chaînes de vente au détail et endure le mauvais sang avec les

passionnés qui ne peuvent pas obtenir ces cartes à n'importe quel prix. Une conception anti-bot et des règles de logique de domaine prudentes, telles que les achats effectués dans les quelques secondes suivant la disponibilité, peuvent identifier les achats non authentiques et rejeter ces transactions.

#### c. Prévention

- Établir et utiliser un cycle de vie de développement sécurisé avec des professionnels AppSec pour aider à évaluer et à concevoir des contrôles liés à la sécurité et à la confidentialité
- Établir et utiliser une bibliothèque de modèles de conception sécurisés ou de composants de routes pavées prêts à l'emploi
- Utilisez la modélisation des menaces pour l'authentification critique, le contrôle d'accès, le logique métier et les flux de clés
- Intégrer le langage et les contrôles de sécurité dans les user stories
- Intégrez des contrôles de plausibilité à chaque niveau de votre application (du frontend au backend)
- Rédiger des tests unitaires et d'intégration pour valider que tous les flux critiques résistent au modèle de menace. Compilez des cas d'utilisations et des cas d’utilisations abusives pour chaque niveau de votre application.
- Séparez les couches de niveau sur les couches système et réseau en fonction des besoins d'exposition et de protection.
- Séparez solidement les locataires par conception à tous les niveaux.
- Limiter la consommation de ressources par utilisateur ou service.

### I.3.6 Mauvaise configuration de sécurité

#### a. Description

L'application peut être vulnérable si l'application est :

- Renforcement de la sécurité approprié manquant sur n'importe quelle partie de la pile d'applications ou autorisations mal configurées sur les services Cloud.

- Des fonctionnalités inutiles sont activées ou installées (par exemple, 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.
- La gestion des erreurs révèle des traces de pile ou d'autres messages d'erreur trop informatifs pour les utilisateurs.
- Pour les systèmes mis à niveau, les dernières fonctionnalités de sécurité sont désactivées ou ne sont pas configurées de manière sécurisée.
- Les paramètres de sécurité des serveurs d'applications, des frameworks d'applications (par exemple, Struts, Spring, ASP.NET), des bibliothèques, des bases de données, etc., ne sont pas définis sur des valeurs sécurisées.
- Le serveur n'envoie pas d'en-têtes ou de directives de sécurité, ou ils ne sont pas définis sur des valeurs sécurisées.
- Le logiciel est obsolète ou vulnérable (voir A06-2021 : Vulnerables and Outdated Components). Sans un processus de configuration de la sécurité des applications concerté et reproductible, les systèmes courent un risque plus élevé.

### b. Exemples de scénarios d'attaques

**Scénario #1 :** Le serveur d'applications est livré avec des exemples d'applications non supprimés du serveur de production. Ces exemples d'applications présentent des failles de sécurité connues que les attaquants utilisent pour compromettre le serveur. Supposons que l'une de ces applications soit la console d'administration et que les comptes par défaut n'aient pas été modifiés. Dans ce cas, l'attaquant se connecte avec des mots de passe par défaut et prend le relais.

**Scénario #2 :** la liste des répertoires n'est pas désactivée sur le serveur. Un attaquant découvre qu'il peut simplement répertorier les répertoires. L'attaquant trouve et télécharge les classes Java compilées, qu'il décompile et désosse pour afficher le code. L'attaquant trouve alors une grave faille de contrôle d'accès dans l'application.

**Scénario #3 :** La configuration du serveur d'applications permet de renvoyer aux utilisateurs des messages d'erreur détaillés, par exemple des traces de pile. Cela expose potentiellement des

informations sensibles ou des défauts sous-jacents tels que des versions de composants connues pour être vulnérables.

**Scénario #4 :** Un fournisseur de services Cloud (CSP) dispose d'autorisations de partage par défaut ouvertes sur Internet par d'autres utilisateurs CSP. Cela permet d'accéder aux données sensibles stockées dans le stockage en nuage.

### c. Prévention

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

- Un processus de durcissement reproductible permet de déployer rapidement et facilement un autre environnement correctement verrouillé. Les environnements de développement, d'assurance qualité et de production doivent tous être configurés de manière identique, avec des informations d'identification différentes utilisées dans chaque environnement. Ce processus doit être automatisé afin de minimiser l'effort requis pour mettre en place un nouvel environnement sécurisé.
- Une plate-forme minimale sans fonctionnalités, composants, documentation et exemples inutiles. Supprimez ou n'installez pas les fonctionnalités et les frameworks inutilisés.
- Une tâche pour examiner et mettre à jour les configurations appropriées à toutes les notes de sécurité, mises à jour et correctifs dans le cadre du processus de gestion des correctifs (voir A06-2021 : Vulnerables and Outdated Components). Vérifiez les autorisations de stockage dans le cloud (par exemple, les autorisations de compartiment S3).
- Une architecture d'application segmentée fournit une séparation efficace et sécurisée entre les composants ou les locataires, avec la segmentation, la conteneurisation ou les groupes de sécurité cloud (ACL).
- Envoi de directives de sécurité aux clients, par exemple, des en-têtes de sécurité.
- Un processus automatisé pour vérifier l'efficacité des configurations et des paramètres dans tous les environnements.

### I.3.7 Composants vulnérables et obsolètes

#### a. Description

Vous êtes probablement vulnérable :

- Si vous ne connaissez pas les versions de tous les composants que vous utilisez (côté client et côté serveur). Cela inclut les composants que vous utilisez directement ainsi que les dépendances imbriquées.
- Si le logiciel est vulnérable, non pris en charge ou obsolète. Cela inclut le système d'exploitation, le serveur Web/d'applications, le système de gestion de base de données (SGBD), les applications, les API et tous les composants, les environnements d'exécution et les bibliothèques.
- Si vous ne recherchez pas régulièrement les vulnérabilités et que vous vous abonnez aux bulletins de sécurité liés aux composants que vous utilisez.
- Si vous ne corrigez pas ou ne mettez pas à niveau la plate-forme sous-jacente, les infrastructures et les dépendances en temps opportun et en fonction des risques. Cela se produit généralement dans des environnements où l'application de correctifs est une tâche mensuelle ou trimestrielle sous contrôle des modifications, laissant les organisations ouvertes à des jours ou des mois d'exposition inutile à des vulnérabilités corrigées.
- Si les développeurs de logiciels ne testent pas la compatibilité des bibliothèques mises à jour, mises à niveau ou corrigées.
- Si vous ne sécurisez pas les configurations des composants (voir A05 :2021-Security Misconfiguration).

#### b. Exemples de scénarios d’attaque

**Scénario #1 :** Les composants s'exécutent généralement avec les mêmes privilèges que l'application elle-même, de sorte que les failles de n'importe quel composant peuvent avoir un impact sérieux. De tels défauts peuvent être accidentels (par exemple, une erreur de codage) ou intentionnels (par exemple, une porte dérobée dans un composant). Voici quelques exemples de vulnérabilités de composants exploitables découvertes :

- CVE-2017-5638, une vulnérabilité d'exécution de code à distance Struts 2 qui permet l'exécution de code arbitraire sur le serveur, a été accusée d'importantes violations.

- Alors que l'Internet des objets (IoT) est souvent difficile ou impossible à corriger, l'importance de les corriger peut être grande (par exemple, les appareils biomédicaux).

Il existe des outils automatisés pour aider les attaquants à trouver des systèmes non corrigés ou mal configurés. Par exemple, le moteur de recherche Shodan IoT peut vous aider à trouver des appareils qui souffrent encore de la vulnérabilité Heartbleed corrigée en avril 2014.

### c. Prévention

Un processus de gestion des correctifs doit être en place pour :

- Supprimez les dépendances inutilisées, les fonctionnalités inutiles, les composants, les fichiers et la documentation.
- Inventorier en permanence les versions des composants côté client et côté serveur (par exemple, les frameworks, les bibliothèques) et leurs dépendances à l'aide d'outils tels que versions, OWASP Dependency Check, retire.js, etc. Surveiller en permanence les sources telles que Common Vulnerability and Exposures (CVE) et National Vulnerability Database (NVD) pour les vulnérabilités des composants. Utilisez des outils logiciels d'analyse de la composition pour automatiser le processus. Abonnez-vous aux alertes par e-mail pour les vulnérabilités de sécurité liées aux composants que vous utilisez.
- N'obtenez des composants que de sources officielles via des liens sécurisés. Préférez les packages signés pour réduire le risque d'inclure un composant malveillant modifié (voir A08 :2021-Software and Data Integrity Failures).
- Surveillez les bibliothèques et les composants qui ne sont pas maintenus ou qui ne créent pas de correctifs de sécurité pour les anciennes versions. Si le correctif n'est pas possible, envisagez de déployer un correctif virtuel pour surveiller, détecter ou vous protéger contre le problème découvert.

Chaque organisation doit garantir un plan continu de surveillance, de triage et d'application des mises à jour ou des modifications de configuration pendant toute la durée de vie de l'application ou du portefeuille.

### I.3.8 Echec d’identification et d’authentification

#### a. Description

La confirmation de l'identité de l'utilisateur, l'authentification et la gestion de la session sont essentielles pour se protéger contre les 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'utilisateur et de mots de passe valides.
- Permet la force brute ou d'autres attaques automatisées.
- Autorise les mots de passe par défaut, faibles ou bien connus, tels que "Mot de passe1" ou "admin/admin".
- Utilise des processus de récupération d'informations d'identification faibles ou inefficaces et de mot de passe oublié, tels que les « réponses basées sur les connaissances », qui ne peuvent pas être sécurisées.
- Utilise des magasins de données de mots de passe en texte brut, cryptés ou faiblement hachés (voir **A02 :2021-Cryptographic Failures**).
- A une authentification multifacteur manquante ou inefficace.
- Expose l'identifiant de session dans l'URL.
- Réutiliser l'identifiant de session après une connexion réussie.
- N'invalide pas correctement les identifiants de session. Les sessions utilisateur ou les jetons d'authentification (principalement les jetons d'authentification unique (SSO)) ne sont pas correctement invalidés lors de la déconnexion ou d'une période d'inactivité.

#### b. Exemples de scénarios d’attaque

**Scénario #1 :** Le credential stuffing, l'utilisation de listes de mots de passe connus, est une attaque courante. Supposons qu'une application n'implémente pas de protection automatisée contre les menaces ou le credential stuffing. Dans ce cas, l'application peut être utilisée comme oracle de mot de passe pour déterminer si les informations d'identification sont valides.

**Scénario #2 :** La plupart des attaques d'authentification se produisent en raison de l'utilisation continue des mots de passe comme seul facteur. Une fois prises en compte, les meilleures

pratiques, la rotation des mots de passe et les exigences de complexité encouragent les utilisateurs à utiliser et à réutiliser des mots de passe faibles. Il est recommandé aux organisations d'arrêter ces pratiques conformément à la norme NIST 800-63 et d'utiliser l'authentification multifacteur.

**Scénario n° 3 :** les délais d'expiration des sessions d'application ne sont pas définis correctement. Un utilisateur utilise un ordinateur public pour accéder à une application. Au lieu de sélectionner "déconnexion", l'utilisateur ferme simplement l'onglet du navigateur et s'en va. Un attaquant utilise le même navigateur une heure plus tard, et l'utilisateur est toujours authentifié.

### c. Prévention

- Dans la mesure du possible, mettez en œuvre une authentification multifacteur pour empêcher le bourrage d'informations d'identification automatisé, la force brute et les attaques de réutilisation d'informations d'identification volées.
- N'expédiez pas ou ne déployez pas avec des informations d'identification par défaut, en particulier pour les utilisateurs administrateurs.
- Mettez en place des contrôles de mots de passe faibles, tels que le test des mots de passe nouveaux ou modifiés par rapport à la liste des 10 000 pires mots de passe.
- Alignez les politiques de longueur, de complexité et de rotation des mots de passe sur les directives 800-63b du National Institute of Standards and Technology (NIST) dans la section 5.1.1 pour les secrets mémorisés ou d'autres politiques de mot de passe modernes et fondées sur des preuves.
- Assurez-vous que l'enregistrement, la récupération des informations d'identification et les voies d'API sont renforcées contre les attaques d'énumération de compte en utilisant les mêmes messages pour tous les résultats.
- Limitez ou retardez de plus en plus les tentatives de connexion infructueuses, mais veillez à ne pas créer de scénario de déni de service. Enregistrez tous les échecs et alertez les administrateurs lorsque le bourrage d'informations d'identification, la force brute ou d'autres attaques sont détectées.
- Utilisez un gestionnaire de session intégré, sécurisé et côté serveur qui génère un nouvel ID de session aléatoire avec une entropie élevée après la connexion. L'identifiant de session

ne doit pas figurer dans l'URL, être stocké en toute sécurité et invalidé après la déconnexion, l'inactivité et les délais d'expiration absolus.

### I.3.9 Echec de l’intégrité du logiciel et des données

#### a. Description

Les défaillances d'intégrité des logiciels et des données concernent le code et l'infrastructure qui ne protègent pas contre les violations d'intégrité. Un exemple de ceci est lorsqu'une application s'appuie sur des plugins, des bibliothèques ou des modules provenant de sources, de référentiels et de réseaux de diffusion de contenu (CDN) non fiables. Un pipeline CI/CD non sécurisé peut introduire un risque d'accès non autorisé, de code malveillant ou de compromission du système. Enfin, de nombreuses applications incluent désormais une fonctionnalité de mise à jour automatique, où les mises à jour sont téléchargées sans vérification d'intégrité suffisante et appliquées à l'application précédemment approuvée. Les attaquants pourraient potentiellement télécharger leurs propres mises à jour à distribuer et à exécuter sur toutes les installations. Un autre exemple est celui où des objets ou des données sont encodés ou sérialisés dans une structure qu'un attaquant peut voir et modifier est vulnérable à la désérialisation non sécurisée.

#### b. Exemples de scénarios d’attaque

**Scénario #1 Mise à jour sans signature :** De nombreux routeurs domestiques, décodeurs, micrologiciels d'appareils et autres ne vérifient pas les mises à jour via un micrologiciel signé. Les micrologiciels non signés sont une cible croissante pour les attaquants et ne devraient qu'empirer. Il s'agit d'une préoccupation majeure car il n'y a souvent aucun mécanisme de correction autre que de corriger dans une future version et d'attendre que les versions précédentes expirent.

**Scénario #2 Mise à jour malveillante de SolarWinds** : Les États-nations sont connus pour attaquer les mécanismes de mise à jour, une récente attaque notable étant l'attaque SolarWinds Orion. La société qui développe le logiciel disposait de processus d'intégrité de construction et de mise à jour sécurisés. Pourtant, ceux-ci ont pu être contournés et pendant plusieurs mois, la société a distribué une mise à jour malveillante très ciblée à plus de 18 000 organisations, dont une centaine environ a été touchées. Il s'agit de l'une des violations les plus profondes et les plus importantes de cette nature dans l'histoire.

**Scénario #3 Désérialisation non sécurisée :** une application React appelle un ensemble de microservices Spring Boot. En tant que programmeurs fonctionnels, ils ont essayé de s'assurer que leur code est immuable. La solution qu'ils ont proposée consiste à sérialiser l'état de l'utilisateur et à le transmettre à chaque requête. Un attaquant remarque la signature d'objet Java "rO0" (en base64) et utilise l'outil Java Serial Killer pour obtenir l'exécution de code à distance sur le serveur d'application.

#### c. Prévention

- Utilisez des signatures numériques ou des mécanismes similaires pour vérifier que le logiciel ou les données proviennent de la source attendue et n'ont pas été modifiés.
- Assurez-vous que les bibliothèques et les dépendances, telles que npm ou Maven, consomment des référentiels approuvés. Si vous avez un profil de risque plus élevé, envisagez d'héberger un référentiel interne connu et approuvé.
- Assurez-vous qu'un outil de sécurité de la chaîne d'approvisionnement logicielle, tel qu'OWASP Dependency Check ou OWASP CycloneDX, est utilisé pour vérifier que les composants ne contiennent pas de vulnérabilités connues
- Assurez-vous qu'il existe un processus d'examen des modifications de code et de configuration afin de minimiser les risques d'introduction de code ou de configuration malveillants dans votre pipeline logiciel.
- Assurez-vous que votre pipeline CI/CD dispose d'une séparation, d'une configuration et d'un contrôle d'accès appropriés pour garantir l'intégrité du code circulant dans les processus de génération et de déploiement.
- Assurez-vous que les données sérialisées non signées ou non chiffrées ne sont pas envoyées à des clients non approuvés sans une certaine forme de contrôle d'intégrité ou de signature numérique pour détecter la falsification ou la relecture des données sérialisées.

### I.3.10 Echec de journalisation et de surveillance de la sécurité

#### a. Description

Pour en revenir au Top 10 OWASP 2021, cette catégorie a pour but d'aider à détecter, escalader et répondre aux violations actives. Sans journalisation ni surveillance, les violations ne peuvent pas être détectées. Une journalisation, une détection, une surveillance et une réponse active insuffisantes se produisent à tout moment :

- Les événements auditables, tels que les connexions, les échecs de connexion et les transactions de grande valeur, ne sont pas consignés.
- Les avertissements et les erreurs génèrent des messages de journal inexistants, inadéquats ou peu clairs.
- Les journaux des applications et des API ne sont pas surveillés pour détecter toute activité suspecte.
- Les journaux ne sont stockés que localement.
- Les seuils d'alerte appropriés et les processus d'escalade des réponses ne sont pas en place ou efficaces.
- Les tests d'intrusion et les analyses par les outils de test dynamique de sécurité des applications (DAST) (tels que OWASP ZAP) ne déclenchent pas d'alertes.
- L'application ne peut pas détecter, escalader ou alerter des attaques actives en temps réel ou quasi réel.

Vous êtes vulnérable aux fuites d'informations en rendant les événements de journalisation et d'alerte visibles pour un utilisateur ou un attaquant (voir A01-2021 : Broken Access Contorle).

### b. Exemples de scénarios d’attaque

**Scénario n° 1 :** L'opérateur du site Web d'un prestataire de soins de santé pour enfants n'a pas pu détecter une violation en raison d'un manque de surveillance et de journalisation. Une partie externe a informé le fournisseur du plan de santé qu'un attaquant avait consulté et modifié des milliers de dossiers de santé sensibles de plus de 3,5 millions d'enfants. Un examen post-incident a révélé que les développeurs du site Web n'avaient pas corrigé les vulnérabilités importantes. Comme il n'y avait pas de journalisation ou de surveillance du système, la violation de données aurait pu être en cours depuis 2013, une période de plus de sept ans.

**Scénario #2 :** Une grande compagnie aérienne indienne a subi une violation de données impliquant plus de dix ans de données personnelles de millions de passagers, y compris des données de passeport et de carte de crédit. La violation de données s'est produite chez un fournisseur d'hébergement cloud tiers, qui a informé la compagnie aérienne de la violation après un certain temps.

**Scénario n° 3 :** une grande compagnie aérienne européenne a subi une violation du RGPD à signaler. La violation aurait été causée par des vulnérabilités de sécurité des applications de paiement exploitées par des attaquants, qui ont récolté plus de 400 000 enregistrements de paiement de clients. La compagnie aérienne a été condamnée à une amende de 20 millions de livres par le régulateur de la confidentialité.

### c. Prévention

Les développeurs doivent implémenter tout ou partie des contrôles suivants, selon le risque de l'application :

- Assurez-vous que tous les échecs de connexion, de contrôle d'accès et de validation des entrées côté serveur peuvent être consignés avec un contexte utilisateur suffisant pour identifier les comptes suspects ou malveillants et conservés pendant suffisamment de temps pour permettre une analyse médico-légale différée.
- Assurez-vous que les journaux sont générés dans un format facilement utilisable par les solutions de gestion des journaux.
- Assurez-vous que les données du journal sont correctement codées pour empêcher les injections ou les attaques sur les systèmes de journalisation ou de surveillance.
- Assurez-vous que les transactions de grande valeur disposent d'une piste d'audit avec des contrôles d'intégrité pour empêcher la falsification ou la suppression, telle que les tables de base de données à ajout uniquement ou similaire.
- Les équipes DevSecOps doivent mettre en place une surveillance et des alertes efficaces afin que les activités suspectes soient détectées et traitées rapidement.
- Établissez ou adoptez un plan de réponse aux incidents et de récupération, tel que National Institute of Standards and Technology (NIST) 800-61r2 ou ultérieur.

Il existe des cadres de protection des applications commerciaux et open source tels que l'ensemble de règles de base OWASP ModSecurity et des logiciels de corrélation de journaux open source, tels que la pile Elasticsearch, Logstash, Kibana (ELK), qui comportent des tableaux de bord personn

alisés et des alertes.

### I.3.11 Contrefaçon de demande côté serveur

#### a. Description

Les failles SSRF se produisent chaque fois qu'une application Web récupère une ressource distante sans valider l'URL fournie par l'utilisateur. Il permet à un attaquant de contraindre l'application à envoyer une requête spécialement conçue vers une destination inattendue, même lorsqu'elle est protégée par un pare-feu, un VPN ou un autre type de liste de contrôle d'accès (ACL) réseau.

Comme les applications Web modernes offrent aux utilisateurs finaux des fonctionnalités pratiques, la récupération d'une URL devient un scénario courant. En conséquence, l'incidence de la SSRF augmente. De plus, la sévérité de SSRF est de plus en plus élevée en raison des services cloud et de la complexité des architectures.

#### b. Exemples de scénarios d'attaque

Les attaquants peuvent utiliser SSRF pour attaquer des systèmes protégés derrière des pare-feu d'applications Web, des pare-feu ou des ACL réseau, en utilisant des scénarios tels que :

**Scénario n° 1 :** Analyse des ports des serveurs internes – Si l'architecture du réseau n'est pas segmentée, les attaquants peuvent cartographier les réseaux internes et déterminer si les ports sont ouverts ou fermés sur les serveurs internes à partir des résultats de connexion ou du temps écoulé pour se connecter ou rejeter les connexions de charge utile SSRF.

**Scénario #2 :** Exposition de données sensibles – Les attaquants peuvent accéder à des fichiers locaux tels que ou à des services internes pour obtenir des informations sensibles telles que `file:///etc/passwd</span>`et `http://localhost:28017/`.

**Scénario #3 :** Accéder au stockage des métadonnées des services cloud - La plupart des fournisseurs de Cloud dispose d'un stockage de métadonnées tel que `http://169.254.169.254/`. Un attaquant peut lire les métadonnées pour obtenir des informations sensibles.

**Scénario #4 :** Compromettre les services internes – L'attaquant peut abuser des services internes pour mener d'autres attaques telles que l'exécution de code à distance (RCE) ou le déni de service (DoS).

### c. Prévention

Les développeurs peuvent empêcher SSRF en mettant en œuvre tout ou partie des contrôles de défense en profondeur suivants :

**Depuis la couche réseau**

- Segmenter la fonctionnalité d'accès aux ressources à distance dans des réseaux séparés pour réduire l'impact de SSRF
- Appliquez des politiques de pare-feu de « refus par défaut » ou des règles de contrôle d'accès au réseau pour bloquer tout le trafic intranet, sauf le trafic essentiel. *Conseils :*
  ~ Établissez une propriété et un cycle de vie pour les règles de pare-feu en fonction des applications.
  ~ Enregistrez tous les flux réseau acceptés *et* bloqués sur les pare-feux (voir A09 :2021-Security Logging and Monitoring Failures).

**Depuis la couche Application :**

- Assainir et valider toutes les données d'entrée fournies par le client
- Appliquer le schéma, le port et la destination de l'URL avec une liste d'autorisation positive
- Ne pas envoyer de réponses brutes aux clients
- Désactiver les redirections HTTP
- Soyez conscient de la cohérence de l'URL pour éviter les attaques telles que le DNS rebinding et les conditions de concurrence "time of check, time of use" (TOCTOU)

N'atténuez pas SSRF via l'utilisation d'une liste de refus ou d'une expression régulière. Les attaquants disposent de listes de données utiles, d'outils et de compétences pour contourner les listes de refus.

**Mesures supplémentaires à considérer :**

- Ne déployez pas d'autres services pertinents pour la sécurité sur les systèmes frontaux (par exemple, OpenID). Contrôler le trafic local sur ces systèmes (par exemple localhost)

- Pour les interfaces avec des groupes d'utilisateurs dédiés et gérables, utilisez le cryptage réseau (par exemple, les VPN) sur des systèmes indépendants pour tenir compte des besoins de protection très élevés

## Conclusion

Dans ce chapitre, nous avons introduit quelques aspects de la sécurité informatique, ainsi que les concepts liées à l'application Web. Nous avons aussi détaillées les menaces et risques que courent les applications Web selon un classement OWASP. Dans le chapitre suivant, nous allons présenter les bases de données et leurs méthodes de sécurités dans des systèmes de gestions bases de données comme Oracle et MySQL.

# II. Chapitre 2 : Notions de base de données et d'API.

# Introduction

Dans ce chapitre, nous présentons les notions de bases de données avec des sous sections dont ce qu'est une base de données, ainsi que les critères d'une base de données, une définition des SGBD et leurs objectifs au sein de l'environnement. Ensuite, nous exposerons les vulnérabilités liées aux bases de données et SGBD, les différents types d'attaques ainsi que les risques encourus et les méthodes de préventions contre ces dits attaques. Enfin, nous aborderons la notion d'interface de programmation d'application (application programming interface (API).

## II.1. Notion des bases de données.

### II.1.1 Qu'est-ce qu'une base de données ?

#### a. Définition d'une base de données.

Une base de données est un ensemble organisé d'informations avec un objectif commun. Plus précisément, on appelle base de données un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, modifier, recherche, consultation) [20].

Aussi, une base de données est une organisation cohérente (i.e. satisfaisant l'ensemble des contraintes d'intégrité) de données permanentes (supposant la gestion de la mémoire auxiliaire et des caches) accessibles par des utilisateurs concurrents (gestion de transactions), fournissant une indépendance physique (entre les programmes d'application et la description des données) et optimisant automatiquement (rôle de l'optimiseur) des requêtes d'interrogation écrites en un langage déclaratif de haut niveau (SQL) [21].

La gestion et l'accès à une base de données sont assurés par un ensemble de programme que constitue le système de gestion de base de données (SGBD) [22].

Ainsi la notion de base de données est généralement couplée à celle des réseaux informatiques afin de pouvoir mettre en commun les informations d'où le nom de « base ». On parle souvent de système d'information pour désigner toute structure regroupant les moyens mis en place pour partager les données [23].

#### b. Critère d'une base de données.

Une base de données doit répondre aux critères suivants [23] :

- **L'exhaustivité :** c'est la présence dans cette base de tous les enseignements qui ont trait aux applications en question.

- **Le non redondance des données :** non duplication ou réplication d'une donnée plusieurs fois.
- **La structure :** c'est l'adaptation du mode de stockage de données au traitement ; structuration que la base doit avoir est liée à l'évolution de la technologie.

### c. Système de gestion de base de données (SGBD) et objectifs du SGBD

Un système de gestion de base de données est un ensemble de programmes permettant la gestion d'une BD, rendant transparent la structuration physique des données sur le support [21].
La gestion d'une BD passe par des langages de commande qui permettent de définir des bases de données, des relations entre les éléments de chaque base, ensuite de spécifier le traitement de ces données (interrogation, mises à jour, calculs et extraction) [22]. Le SGBD reçoit des commandes aussi bien programmes d'application que des utilisateurs : il commande les manipulations de données, généralement par l'intermédiaire d'un système de gestion de fichiers (SGF) [23].

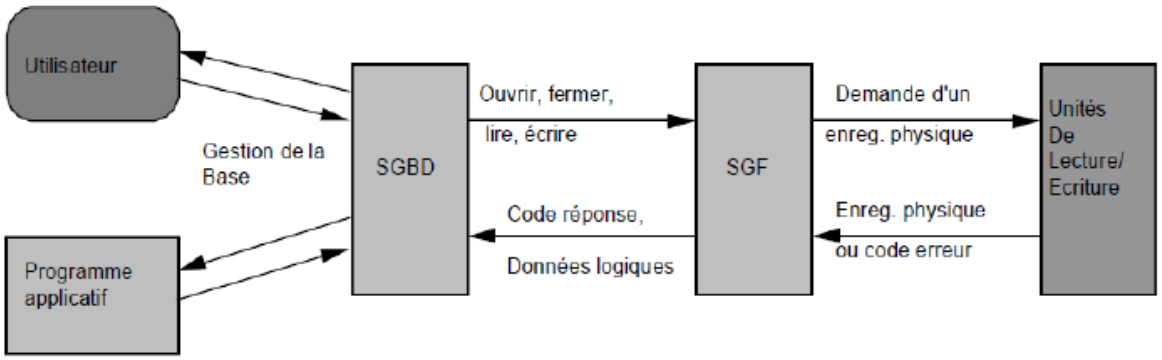


***Figure 2.6 :*** *Systèmes de gestion de base de données « SGBD » [22].*

Les définitions sont cependant de peu d'intérêt pour déterminer si un système est vraiment un SGBD ou s'il s'agit simplement d'un système d'information classique ou d'un système de fichiers. Il faut mieux définir le SGBD en précisant certaines des fonctions qu'il doit remplir [24] :

- L'intégration des données afin d'éviter l'incohérence d'éventuelles données dupliquées (tout est intégré dans un seul ensemble cohérent).
- La séparation entre les moyens de stockage physique des données et la logique des applications.
- Un contrôle unique de toutes les données afin de permettre l'utilisation simultanée par plusieurs utilisateurs.

- La possibilité d'utiliser des structures de fichiers et des méthodes d'accès complexes, de façon à ce que les relations correctes entre les données puissent être exprimées et les données utilisées le plus efficacement dans un grand nombre d'applications.
- Des facilités pour le stockage, la modification, la réorganisation, l'analyse et la consultation des données, sans que le système impose des restrictions à l'utilisateur.
- Des contrôles de sécurité afin d'empêcher l'accès illégal à certaines données.
- Des contrôles d'intégrité pour prévenir une modification indue des données (exemple : contrôle d'exactitude, de validité).
- La compatibilité avec les principaux langages de programmation, les programmes-sources existants, et les données extérieures à la base.
- Posséder une capacité de stockage élevée.
- Pouvoir répondre à des requêtes avec un niveau de performances adapté.
- Fournir des facilités pour la gestion des méta-données.

Pour pallier aux inconvénients des méthodes classiques de gestion de fichiers, les SGBD visent quatre objectifs : intégration et corrélation, flexibilité (indépendance), disponibilité, sécurité.
Ces objectifs exigent une distinction nette entre les données et les procédures de manipulation de ces données : aux données, on associera une fonction **d'administration des données**, aux procédures de manipulation une **fonction de programmation** [22].

- **Intégration et corrélation :**

Dans les systèmes classiques, chaque application gère ses données dans ses propres "fichiers", d'où [22] :

❖ Un risque de redondance, et un danger d'incohérence des données.

- La même donnée peut appartenir à plusieurs applications, induisant une déperdition de stockage.
- Toute modification de cette donnée est à enregistrer plusieurs fois : si cette mise à jour multiple n'est pas effectuée correctement, les données deviennent incohérentes.
- Le coût de la mise à jour augmente du fait de la multiplication des entrées-sorties physiques.

❖ Une difficulté pour créer de nouveaux traitements.

- Les nouvelles applications entraînent des duplications supplémentaires de données.
- Leur intégration avec les applicatifs en exploitation entraîne des modifications importantes. Dans l'approche SGBD, un "réservoir" commun (intégration) est constitué, représentant une modélisation (corrélation) aussi fidèle que possible de l'organisation réelle de l'entreprise :
  - ❖ Toutes les applications puisent dans ce réservoir, les données qui les concernent, évitant ainsi les duplications.
  - ❖ Mais le partage des données entre les utilisateurs pose le problème de la synchronisation des accès concurrents.

- **Flexibilité (indépendance) [22] :**

- Dans les systèmes classiques, tout changement intervenant dans le stockage des données (support, méthode d'accès physique) entraîne des modifications lourdes des applications correspondantes.

- L'approche SGBD poursuit trois objectifs, pour assurer l'indépendance des données par rapport aux traitements :

  - ❖ Indépendance physique : tout changement de support, de méthode d'accès reste transparent au niveau de l'utilisateur.

  - ❖ Indépendance logique : les programmes d'application sont rendus transparents à une modification dans l'organisation logique globale, par la définition de sous-schémas couvrant les besoins spécifiques en données.

  - ❖ Indépendance vis-à-vis des stratégies d'accès : l'utilisateur n'a plus à prendre en charge l'écriture des procédures d'accès aux données. Il n'a donc pas à intégrer les modifications tendant à optimiser les chemins d'accès (ex : création d'index).

- **Disponibilité [22] :**
  - ❖ Le choix d'une approche SGBD ne doit pas se traduire par des temps de traitement plus longs que ceux des systèmes antérieurs.
  - ❖ L'utilisateur doit ignorer l'existence d'utilisateurs concurrents.
  - ❖ L'aspect "performance" est donc crucial dans la mise en œuvre d'une base de données. Un tel objectif ne peut être atteint que si la conception d'une base de données est menée de façon rigoureuse avec un découpage fonctionnel adéquat.

Les règles et contraintes inhérentes sont évoquées lors de l'apprentissage d'une méthodologie d'analyse (exemple MERISE).

- **Sécurité [22] :**

  La sécurité des données recouvre deux aspects :

  - L'intégrité, ou protection contre l'accès invalide (erreurs ou pannes), et contre l'incohérence des données vis-à-vis des contraintes de l'entreprise.
  - La confidentialité, ou protection contre l'accès non autorisé ou la modification illégale des données.

  Pour ne pas trop affecter les performances, la sécurité doit également être prise en compte dès la phase de conception.

### II.1.2 Vulnérabilités des bases de données et SGBD.

Bien que la sécurité soit l'une des raisons de l'architecture trois couches, plusieurs challenges praticables sont enlevés lors de la construction du système tel que l'authentification des utilisateurs, le contrôle des accès et Audit les actions des utilisateurs, la protection des données entre les couches, la limitation des privilèges de l'intermédiaire, et la construction des systèmes extensibles [22].

On envisage souvent la sécurité sous un angle fermé, essentiellement celui de la confidentialité. Mais bien d'autres concepts sous-tendent la sécurité. Ils sont pratiquement tous applicables aux OS et aux SGBD, tant il est vrai que ces deux domaines sont extrêmement recouvrant.

- **Confidentialité [22] :** Tout n'est pas accessible à tout le monde ! Se connecter à l'OS ou à la base de données, donne un certain nombre de droits et de ressources en fonction d'un profil défini et maintenu par un administrateur. La **granularité d'accès** peut aller jusqu'à la vision unique d'un champ d'un enregistrement d'une table particulière [25]. L'information protégée ne doit pas être accessible aux utilisateurs ou un programme non autorisés.

  C'est crucial [25] :

  - ❖ Dans des environnements critiques ou stratégiques : militaires ou commerciaux, par exemple.
  - ❖ Pour respecter le droit des individus à décider comment et dans quel but les informations les concernant peuvent être extraites, mémorisées ou transmises à d'autres individus.

- **Intégrité :** Que les données soient réparties ou non –dans ce dernier cas les mécanismes mis en jeux seront plus complexes– elles doivent être cohérentes. Cela sous-entend, d'une part que les accès concurrents d'utilisateurs, notamment lors de mises à jour, ne doivent pas compromettre la cohérence des données et d'autre part que ces dernières satisfassent aux contraintes d'intégrité du modèle, et / ou aux règles de gestion de l'entreprise [25]. Les données ne peuvent être modifiées que par les utilisateurs habilités à le faire qu'elles soient dues à [22] :
  - ❖ Des pannes de système,
  - ❖ Des manipulations erronées,
  - ❖ Des sabotages.
- **Disponibilité :** Faculté de délivrer correctement un service en termes de délai et de qualité à l'utilisateur. Se mesure en pourcentage du temps de fonctionnement total. Une disponibilité de 100% est possible (temps de reprise nul) il suffit de s'en donner les moyens logiciels et matériels [25]...
- **Fiabilité :** Des mécanismes de sauvegarde variés (physique, logique, off-line, on-line, totale, partielle, incrémentale), ainsi que des mécanismes de journalisation, et de reprise permettent de restaurer une information sans pratiquement aucune perte, dans tous les cas de problème matériel ou logiciel [25].
- **Traçabilité :** en cas de problème important ou d'attaque du système, on peut recourir à l'analyse de traces ou de logs. Le niveau de détail de ces traces est paramétrable, et concerne soit les aspects système, soit réseau, soit l'accès aux données élémentaires elles-mêmes [25].
- **Maintenabilité :** aptitude à la réparation, évolution, maintenance du système. Mesuré en temps de reprise après panne (Mean Time To Recover) [25].

## II.2. Les différents types d'attaques liées aux BD.

De nombreuses vulnérabilités logicielles, mauvaises configurations ou modèles d'utilisation abusive ou de négligence peuvent entrainer des violations. Les menaces identifiées au cours de ces années sont les mêmes qui continuent d'affliger les entreprises aujourd'hui. Nous avons ci-dessous un certain nombre des causes et des types de cyber menaces les plus connus pour la sécurité des bases de données [26] :

- **Menaces internes :** une menace interne est un risque de sécurité provenant de l'une des trois sources suivantes, chacun ayant des moyens privilégiés d'accès à la base de données :
    - Un initié malveillant avec de mauvaises intentions.
    - Une personne négligente au sein de l'organisation qui expose la base de données à des attaques par des actions imprudentes.
    - Un étranger qui obtient des informations d'identification par ingénierie sociale ou d'autres méthodes, ou accède aux informations d'identification de la base de données.

  Une menace interne est l'une des causes les plus courantes des failles de sécurité des bases de données et elle se produit souvent parce que de nombreux employés ont obtenus un accès utilisateur privilégié [26].

- **L'erreur humaine :** les mots de passe faibles, le partage de mots de passe, l'effacement accidentel ou la corruption de données et d'autres comportements indésirables des utilisateurs sont toujours à l'origine de près de la moitié des violations de données signalés [26].
- **Exploitation des vulnérabilités du logicielle de base de données :** Les attaquants tentent constamment d'isoler et de cibler les vulnérabilités des logiciels, et le logiciel de gestion de base de données est une cible très précieuse. De nouvelles vulnérabilités sont découvertes quotidiennement et toutes les plates-formes de gestion de bases de données open source et les fournisseurs de logiciels de bases de données commerciaux publient régulièrement des correctifs de sécurité. Cependant, si vous n'utilisez pas ces correctifs rapidement, votre base de données peut être exposée à des attaques.

  Même si vous appliquez des correctifs à temps, il existe toujours un risque d'attaques de type « zero-day », lorsque les attaquants découvrent une vulnérabilité, mais celle-ci n'a pas encore été découverte et corrigée par le fournisseur de la base de données [26].
- **Attaques par injection SQL/NoSQL :** Une menace spécifique à la base de données implique l'utilisation de chaînes d'attaque non-SQL et SQL arbitraires dans les requêtes de base de données. En règle générale, il s'agit de requêtes créées en tant qu'extension de formulaires d'application Web ou reçues via des requêtes HTTP. Tout système de base de données est vulnérable à ces attaques, si les développeurs n'adhèrent pas aux

pratiques de codage sécurisé et si l'organisation n'effectue pas de tests de vulnérabilité réguliers [26].

- **Attaques par débordement de tampon :** Le débordement de la mémoire tampon se produit lorsqu'un processus tente d'écrire une grande quantité de données dans un bloc de mémoire de longueur fixe, supérieure à ce qu'il est autorisé à contenir. Les attaquants peuvent utiliser les données excédentaires, conservées dans des adresses mémoire adjacentes, comme point de départ à partir duquel lancer des attaques [26].
- **Attaques par déni de service (DoS/DDoS) :** Lors d'une attaque par déni de service (DoS), le cybercriminel submerge le service cible, en l'occurrence le serveur de base de données, en utilisant une grande quantité de fausses requêtes. Le résultat est que le serveur ne peut pas exécuter les demandes authentiques des utilisateurs réels et tombe souvent en panne ou devient instable. Dans une attaque par déni de service distribué (DDoS), un faux trafic est généré par un grand nombre d'ordinateurs, participant à un botnet contrôlé par l'attaquant. Cela génère des volumes de trafic très importants, qu'il est difficile d'arrêter sans une architecture défensive hautement évolutive. Les services de protection DDoS basés sur le cloud peuvent évoluer de manière dynamique pour faire face à de très grandes attaques DDoS [26].
- **Logiciels malveillants :** Un logiciel malveillant est un logiciel écrit pour tirer parti de vulnérabilités ou pour endommager une base de données. Les logiciels malveillants peuvent arriver via n'importe quel terminal connecté au réseau de la base de données. La protection contre les logiciels malveillants est importante sur n'importe quel terminal, mais surtout sur les serveurs de base de données, en raison de leur grande valeur et de leur sensibilité [26].

Aussi il existe plusieurs autres types de menaces comme le **crack de password**, le contournement de l'applicatif par programmes client SQL, les récupérations de données offline ou hors production, les back doors, les virus, les vers (Worm), le cheval de Troie (trojan horse). Les faiblesses de comportements comme les portes ouvertes, l'installation par défaut, le mauvais codage, la mauvaise politique de gestion de droits, l'absence de mise à jour ou encore un contrôle d'accès au niveau applicatif pouvant être facilement court circuité sont des menaces qui peuvent compromettre la sécurité d'une base de données [25].

### II.2.1. Les risques encourus.

Les risques propres à une source de données sont les suivants [22] :

- Le **vol de données** induit la perte de **confidentialité** des données stockées. La divulgation de données financières hautement confidentielles peut avoir un impact néfaste sur l'activité d'une entreprise : risque juridique, atteinte à l'image de marque, perte de confiance des partenaires industriels...
- L'**altération de données** induit une perte d'**intégrité**, c'est-à-dire que les données ne sont plus dignes de confiance. En fonction de la rapidité de détection et de la qualité des sauvegardes, les conséquences peuvent en être réduites. Mais une application fonctionnant sur des données falsifiées peut voir son comportement fortement influencé : par exemple, un site de commerce électronique pourrait débiter le compte d'un autre client que celui réalisant la commande !
- La **destruction de données** remet sérieusement en cause la **continuité de l'activité** de l'entreprise concernée. Privée de ses données clients, sans sauvegarde, c'est le dépôt de bilan garanti !
- L'**augmentation du niveau de privilèges** d'un utilisateur d'une application est plus insidieuse que les risques précédents, car comme pour l'altération de données, il n'est remarqué qu'après un certain laps de temps durant lequel le pirate peut réaliser un grand nombre d'actions malveillantes. Il peut ainsi s'attribuer le droit d'accès à des informations confidentielles, le droit d'accès à des opérations sensibles, voire même prendre le contrôle d'une application.

Selon le SGBD utilisé, des **ressources systèmes** peuvent être attribuées à chaque utilisateur (nombre de requêtes par unité de temps...). Ces ressources peuvent être limitées par l'administrateur système afin d'éviter l'écroulement des capacités de traitement du serveur (**déni de service**) par un utilisateur malveillant. De plus, ceci permet de limiter la portée d'une attaque par altération ou vol de données en limitant le nombre d'opérations réalisables en un temps donné. La conséquence d'un tel risque peut être la paralysie du serveur (perte de **disponibilité**). On le voit, les risques sont variés et leurs conséquences potentiellement dramatiques. Ainsi, il est nécessaire d'attribuer les droits d'accès avec parcimonie.

### II.2.2. Les moyens de sécuriser le serveur de BD.

Un serveur de base de données est une machine physique ou virtuelle exécutant la base de données. La sécurisation d'un serveur de base de données, également appelée "renforcement",

est un processus qui inclut la sécurité physique, la sécurité du réseau et la configuration sécurisée du système d'exploitation [26].

- **Assurer la sécurité de la base de données physique :**

  Abstenez-vous de partager un serveur pour les applications Web et les applications de base de données, si votre base de données contient des données sensibles. Bien qu'il puisse être moins cher et plus facile d'héberger votre site et votre base de données ensemble sur un fournisseur d'hébergement, vous placez la sécurité de vos données entre les mains de quelqu'un d'autre.

  Si vous comptez sur un service d'hébergement Web pour gérer votre base de données, vous devez vous assurer qu'il s'agit d'une entreprise ayant de solides antécédents en matière de sécurité. Il est préférable de rester à l'écart des services d'hébergement gratuits en raison du manque de sécurité possible.

  Si vous gérez votre base de données dans un centre de données sur site, gardez à l'esprit que votre centre de données est également sujet aux attaques externes ou aux menaces internes. Assurez-vous d'avoir des mesures de sécurité physiques, y compris des serrures, des caméras et du personnel de sécurité dans votre installation physique. Tout accès aux serveurs physiques doit être enregistré et accordé uniquement aux personnes autorisées [26].

  En outre, ne laissez pas les sauvegardes de base de données dans des emplacements accessibles au public, tels que des partitions temporaires, des dossiers Web ou des buckets de stockage cloud non sécurisés [26].

- **Verrouiller les comptes et les privilèges :**

  Considérons le serveur de base de données Oracle. Une fois la base de données installée, l'assistant de configuration de base de données Oracle (DBCA) expire automatiquement et verrouille la plupart des comptes d'utilisateur de base de données par défaut [26].

  Si vous installez une base de données Oracle manuellement, cela ne se produit pas et les comptes privilégiés par défaut ne seront ni expirés ni verrouillés. Leur mot de passe reste le même que leur nom d'utilisateur, par défaut. Un attaquant essaiera d'abord d'utiliser ces informations d'identification pour se connecter à la base de données [26].

  Il est essentiel de s'assurer que chaque compte privilégié sur un serveur de base de données est configuré avec un mot de passe fort et unique. Si les comptes ne sont pas nécessaires, ils doivent être expirés et verrouillés.

Pour les comptes restants, l'accès doit être limité au strict minimum requis. Chaque compte ne doit avoir accès qu'aux tables et aux opérations (par exemple, SELECT ou INSERT) requises par l'utilisateur. Évitez de créer des comptes d'utilisateurs ayant accès à toutes les tables de la base de données.

- **Patchez régulièrement les serveurs de base de données :**

  Assurez-vous que les correctifs restent à jour. La gestion efficace des correctifs de base de données est une pratique de sécurité cruciale, car les attaquants recherchent activement de nouvelles failles de sécurité dans les bases de données, et de nouveaux virus et logiciels malveillants apparaissent quotidiennement.

  Un déploiement rapide des versions à jour des services packs de base de données, des correctifs de sécurité critiques et des mises à jour cumulatives amélioreront la stabilité des performances de la base de données [26].

- **Désactiver l'accès au public :**

  Les organisations stockent leurs applications dans des bases de données. Dans la plupart des scénarios réels, l'utilisateur final n'a pas besoin d'un accès direct à la base de données. Ainsi, vous devez bloquer tout accès réseau public aux serveurs de base de données, sauf si vous êtes un fournisseur d'hébergement. Idéalement, une organisation devrait configurer des serveurs de passerelle (tunnels VPN ou SSH) pour les administrateurs distants [26].

- **Crypter tous les fichiers et sauvegardes :**

  Quelle que soit la solidité de vos défenses, il est toujours possible qu'un pirate s'infiltre dans votre système. Pourtant, les attaquants ne sont pas la seule menace pour la sécurité de votre base de données. Vos employés peuvent également représenter un risque pour votre entreprise. Il est toujours possible qu'un initié malveillant ou négligent ait accès à un fichier auquel il n'est pas autorisé à accéder.

  Le cryptage de vos données les rend illisibles pour les attaquants et les employés. Sans clé de cryptage, ils ne peuvent pas y accéder, cela fournit une dernière ligne de défense contre les intrusions indésirables. Chiffrez les fichiers d'application, les fichiers de données et les sauvegardes les plus importants afin que les utilisateurs non autorisés ne puissent pas lire vos données critiques [26].

Ajoutons à ces moyens de sécurité quelques bonnes pratiques pouvant être utilisé pour améliorer la sécurité des bases de données :

- **Gestion active des mots de passe et l'accès des utilisateurs :** Si vous avez une grande organisation, vous devez penser à automatiser la gestion des accès via la gestion des mots de passe ou un logiciel de gestion des accès. Cela fournira aux utilisateurs autorisés un mot de passe à court terme avec les droits dont ils ont besoin chaque fois qu'ils doivent accéder à une base de données. Il garde également une trace des activités terminées pendant cette période et empêche les administrateurs de partager des mots de passe. Bien que les administrateurs puissent penser que le partage de mots de passe est pratique, cela rend presque impossible la responsabilité et la sécurité efficaces de la base de données [26]. De plus, les mesures de sécurité suivantes sont recommandées [26] :
    - Des mots de passe forts doivent être appliqués.
    - Les hachages de mot de passe doivent être salés et stockés chiffrés.
    - Les comptes doivent être verrouillés après plusieurs tentatives de connexion.
    - Les comptes doivent être régulièrement examinés et désactivés si le personnel change de rôle, quitte l'entreprise ou n'a plus besoin du même niveau d'accès.
- **Tester la sécurité de la base de données :** Une fois que vous avez mis en place votre infrastructure de sécurité de base de données, vous devez la tester contre une menace réelle. Auditer ou effectuer des tests d'intrusion sur votre propre base de données vous aidera à vous mettre dans l'état d'esprit d'un cybercriminel et à isoler les vulnérabilités que vous auriez pu ignorer. Pour vous assurer que le test est complet, impliquez des pirates éthiques ou des services de test d'intrusion reconnus dans vos tests de sécurité. Les testeurs d'intrusion fournissent des rapports détaillés répertoriant les vulnérabilités de la base de données, et il est important d'enquêter et de corriger rapidement ces vulnérabilités. Exécutez un test d'intrusion sur un système de base de données critique au moins une fois par an [26].
- **Utilisation de la surveillance de base de données en temps réel :** L'analyse continue de votre base de données à la recherche de tentatives de violation augmente votre sécurité et vous permet de réagir rapidement à d'éventuelles attaques. En particulier, File Integrity Monitoring (FIM) peut vous aider à consigner toutes les actions effectuées sur le serveur de la base de données et à vous alerter d'éventuelles violations. Lorsque FIM détecte un changement dans des fichiers de base de données importants, assurez-vous que les équipes de sécurité sont alertées et capables d'enquêter et de répondre à la menace [26].

- **Utilisation des pare-feu d'application Web et de base de données :** Vous devez utiliser un pare-feu pour protéger votre serveur de base de données contre les menaces de sécurité de base de données. Par défaut, un pare-feu n'autorise pas l'accès au trafic. Il doit également empêcher votre base de données de démarrer des connexions sortantes à moins qu'il n'y ait une raison particulière de le faire. En plus de protéger la base de données avec un pare-feu, vous devez déployer un pare-feu d'application Web (WAF). En effet, les attaques visant les applications Web, y compris l'injection SQL, peuvent être utilisées pour obtenir un accès illicite à vos bases de données. Un pare-feu de base de données n'arrêtera pas la plupart des attaques d'applications Web, car les pare-feu traditionnels fonctionnent au niveau de la couche réseau, tandis que les couches d'application Web fonctionnent au niveau de la couche application (couche 7 du modèle OSI). Un WAF fonctionne au niveau de la couche 7 et est capable de détecter le trafic d'applications Web malveillantes, telles que les attaques par injection SQL, et de le bloquer avant qu'il ne puisse endommager votre base de données [26].

## II.3. Notion d'interface de programmation d'application.

### II.3.1. Définition et histoire des API

Une API, ou interface de programmation d'application, est un ensemble de définitions et de protocoles qui facilite la conception et la fusion de logiciels d'applications [27]. C'est aussi l'interface qu'un logiciel présente à d'autres programmes, aux humains et, dans le cas des API Web, au monde via l'internet [28]. Les API sont apparues avant même les ordinateurs personnels, utilisées à cette époque en tant que bibliothèques pour les systèmes d'exploitation et résidant presque toutes en local sur les systèmes dont elles s'exécutaient, elles pouvaient s'envoyer des messages entre les mainsframes. Au debut des années 2000, elles sont devenues importantes pour l'intégration des données à distances [27].

### II.3.2. Les types d'API, le SOAP et le REST.

Les API sont de trois types (privées, partenaires et publiques) avec des standards comme SOAP et REST. Les API privées utilisable uniquement en interne avec une approche permettant de garder un contrôle total sur l'API. Les API partenaires sont partagées avec certains partenaires de l'entreprise avec une approche pouvant générer de nouveaux flux de revenus sans compromettre la sécurité. Quant aux API publiques, elles sont accessibles à tous et cette approche autorise les tiers à développer des applications qui interagissent avec d'autres API pouvant être source d'innovations [27].

Le Simple Object Access Protocol (SOAP) est un protocole conçu pour simplifier l'échange des informations entre les applications qui s'exécutent dans différents environnements ou écrites dans des langages différents. Les API conçues à partir ce protocole utilisent le format XML pour leurs messages et reçoivent des requêtes via HTTP ou SMTP [27].

Le Representational State Transfer, ou REST est un style d'architecture normalisé. Les API web issues de cette architecture sont dites API RESTful. Cette architecture se définit selon Roy Fielding sur six contraintes dont l'architecture client-serveur, un serveur stateless, une mémoire cache, le système à couches, le code à la demande (facultatif) et une interface uniforme [27].

Mais ces dernières années, l'OpenAPI permettant aux développeurs des créer des interfaces d'API REST indépendantes du langage tout en ayant une compréhension minimum de la part des utilisateurs, s'imposa comme norme commune pour définir les API REST [27].

L'émergence d'une autre norme d'API, GraphQL, un langage de requête et un environnement d'exécution côté serveur propose de remplacer l'architecture REST, permettant aux développeurs de créer des requêtes qui extraient les données de plusieurs sources à l'aide d'un seul appel d'API [27].

### II.3.3. Fonctionnement d'une API

En général, les API offrent plus de flexibilité, simplifient la conception, l'administration et l'utilisation, et donnent les moyens d'innover. Elles permettent la communication entre de nombreux composants différents des applications, mais aussi entre les composants d'autres développeurs. Elles agissent ainsi comme un intermédiaire qui transmet des messages à travers un système de requêtes et de réponses, comme par exemple l'API Google Maps. Elles s'intègrent facilement comme des composants sans que le développeur change l'architecture de son application [27].

Dans notre cas, il s'agit d'une API de sécurité permettant de filtrer les caractères susceptibles d'être des syntaxes d'attaques comme ceux des injections SQL ou XSS. Nous détaillerons son fonctionnement dans le chapitre suivant.

## Conclusion

Dans ce chapitre, nous avons présenté les notions de bases de données avec des sous sections dont ce qu'est une base de données, ainsi que les critères d'une base de données, une définition des SGBD et leurs objectifs au sein de l'environnement Web. Ensuite, nous avons énoncé les vulnérabilités liées aux bases de données et SGBD, les différents types d'attaques ainsi que les risques encourus et les méthodes de préventions contre ces dits attaques.

Enfin, la notion d'interface de programmation d'application (application programming interface API), son histoire, les types d'API avec les standards SOAP et REST ainsi que le fonctionnement des API.

Dans le dernier chapitre, nous allons décrire notre site web de librairie en ligne, expliquer les API de sécurité que nous avons implémentée et proposer des tests de pénétration sur notre site web.

# III. Chapitre 3 : Description du site web (libraire en ligne) et implémentation des API de sécurisation.

# Introduction

De nos jours, les bases de données sont des composants incontournables pour les applications et serveurs Web fournissant du contenu dynamique. En permettant une organisation et le stockage des données d'une application, elles doivent être protégées rigoureusement pour éviter la fuite ou l'accès aux informations sensibles pouvant susciter l'intérêt de certaines personnes malveillantes.

Dans le cadre de notre PFE, ce chapitre présentera une description de notre site web de librairie en ligne servant de plateforme de tests pour la simulation. Ensuite, une explication détaillée de l'API implémentée contre les attaques SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) et le renforcement du mot de passe que nous avons proposée et enfin les tests des pénétrations sur le site web.

## III.1 Description du site web

La présente section donne un aperçu général la conception de notre site web telle que les langages de programmation, une description des pages utilisées par l'administrateur et celui des utilisateurs.

Nous avons opté pour une implémentation avec les langages de programmations suivant :

- **HTML :** désignant HyperText Markup Language, pouvant se traduire en langage de balises pour l'hypertexte. Nous avons utilisé ce langage pour créer et représenter le contenu de nos pages et sa structure.
- **CSS :** signifiant Cascading Style Sheets, traduit en feuilles de style en cascade. C'est un langage informatique décrivant une présentation d'une page HTML ou XML. Nous avons utilisé ce langage pour mettre en forme notre site web.
- **JavaScript :** souvent abrégé en JS, est un langage de script léger, orienté objet, permettant de créer du contenu mis à jour de façon dynamique, de contrôler le contenu multimédia ou encore d'animer des images, en gros d'implémenter des mécanismes complexe sur une page web.
- **PHP :** Hypertext Preprocessor, est un langage de programmation libre, utilisé pour produire des pages web dynamiques via un serveur HTTP. Toutes nos pages sont implémentées en partie avec ce langage pour inclure des requêtes SQL.
- **MySQL :** un système de gestion de base de donnée relationnelles, distribué en double licence GPL et propriétaire. Nous utilisons ce SGBDR pour stocker nos informations concernant les utilisateurs, les prix des livres ou encore les détails d'une commande.

Les pages de l'administrateur sont composées d'une page de connexion donnant une accessibilité au site entier. Ainsi, nous pouvons accéder aux autres pages à commencer par la page d'accueil donnant un aperçu sur le statut des activités de la librairie en ligne utilisées par les utilisateurs. Nous avons aussi une page pour ajouter, modifier ou supprimer les livres dans la librairie en ligne. Aussi, une page de gestion de commande en ligne pour assurer le service de livraison.

Les pages dont les utilisateurs ont accès offre une plateforme d'achat de livres, de lancer des commandes pour un service de livraison. L'utilisateur doit avant tout se connecter au niveau de la page de connexion. Il pourra ainsi modifier son compte, faire un achat, modifier son panier, ou encore visiter la librairie.

## III.2 Description de l'API de sécurité

L'API est implémentée avec le langage de programmation PHP. En effet, elle regroupe une page en PHP faisant appel aux fonctions proposés permettant d'assurer le système de sécurité d'un site web contre les attaques injections SQL, XSS et CSRF.

La Fonction *sqlinjection* est chargée de refuser les entrées au niveau des cases du formulaire, la case des URL, ou encore dans les autres pages. Elle permet d'éviter les injections SQL, les attaques XSS ou bien le détournement de compte par une attaque type CSRF. La fonction s'assure que les entrées sont différentes d'une liste de mots clés (Select, union, script, order etc…) ou des caractères spéciaux ('*', '!', '(', ')','<','>','='). Aussi, une sensibilité aux mots de passes est implémentée pour assurer un minimum contre le cassage de mot de passe par force brute. Dans ce qui suit, nous présentons une simulation de pénétration pour montrer le fonctionnement de l'API.

Nos tests vont être présentés en plusieurs scénarios pour mettre en évidence notre API contre les attaques :

- **Scénario de l'attaque injection SQL :** un attaquant peut introduire une requête SQL pour extraire des informations. Considérons une requête SQL « SELECT * FROM users » dans une case d'un formulaire de page de connexion, ou bien dans la case de recherche, dans le cas de notre site non protégé par l'API, elle sera un succès. Mais dans le cas où notre site est protégé, elle affiche un message d'erreur.

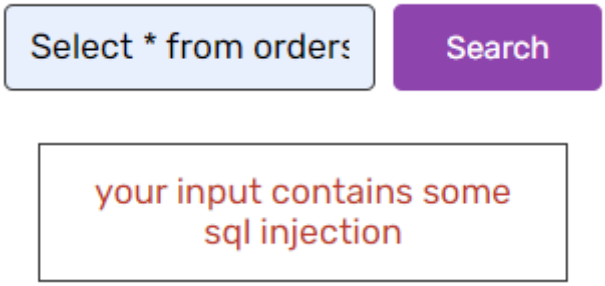


***Figure 3.7 :*** *Exemple d'injection SQL sur notre site.*

- **Scénario de l'attaque XSS :** dans ce cas, un hacker peut effectuer une attaque de type stored XSS comme entrée dans une case d'une page de recherche « <script>alert(document.cookie)</script> », lui permettant d'afficher une fenêtre contenant les cookies. Mais au lieu de ce script, il peut faire usage d'un autre plus dangereux.

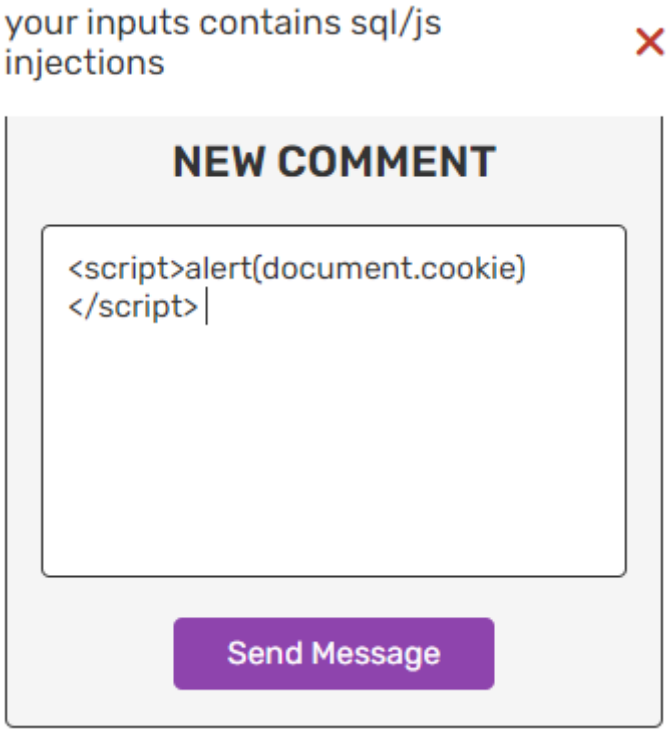


***Figure 3.8 :*** *Tentative d'attaque XSS.*

- **Scénario de l'attaque CSRF :** cette attaque peut se faire en insérant un lien malveillant dans un commentaire, permettant ainsi à l'attaquant de détourner le compte de celui qui clique sur ce lien. Par exemple, les liens publicitaires qui sont souvent en commentaire dans les forums d'un site accueillant beaucoup d'utilisateurs.

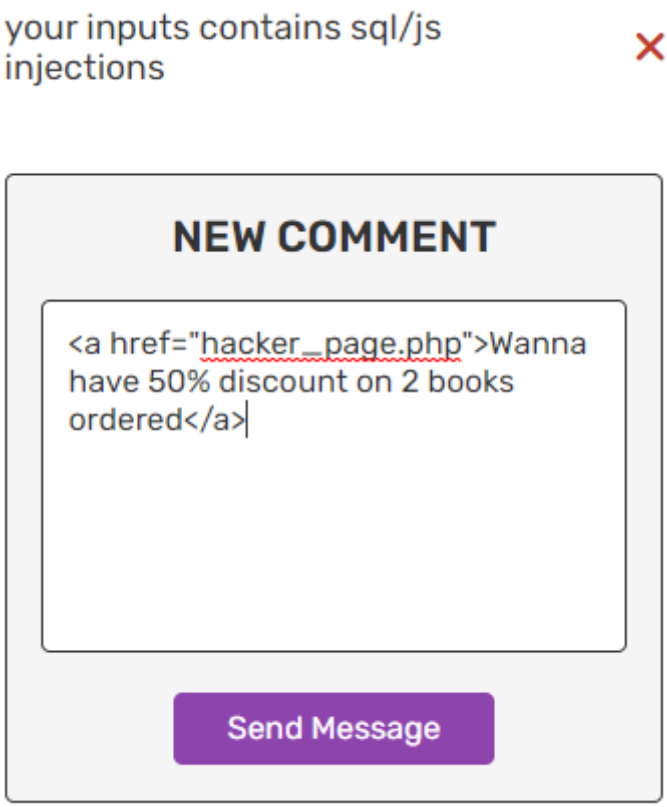


***Figure 3.9 :*** *Test d'insertion de lien dans le site web.*

## Conclusion

Dans ce chapitre, nous avons fait une description de notre site de librairie en ligne avec ses différentes parties qui la compose. Elle a servi de plateforme pour les tests de pénétration afin de mettre en évidence les contremesures implémentées dans notre API. Ensuite, une explication sur notre API pour mieux appréhender son mode de fonctionnement et son utilisation sur plusieurs autres sites web.

# Conclusion générale

Au cours de ce mémoire, nous avons étudié plusieurs aspects de la sécurité dans les applications Web et dans les bases de données. Notre travail est articulé en deux parties ; dont la première concernant l'aspect théorique contenant deux chapitres.

Le premier chapitre parle de la sécurité informatique, des applications Web en passant par les architectures conçu pour améliorer les environnements Web et leurs vulnérabilités face aux cyberattaques à partir d'un classement établi par OWASP.

Le second chapitre concerne les bases de données, les vulnérabilités auxquels elles sont exposées, les risques encourus, les différents types d'attaques et les méthodes de prévention utiles. Une autre section du chapitre présente une brève notion sur les interfaces de programmation d'application.

La seconde partie du mémoire est pratique. Elle décrit notre site web destiné à servir de simulateur pour les attaques injection SQL, Cross-Site Scripting, Cross-Site Request Forgery.

En outre, au cours de l'élaboration de notre site web et de la mémoire, nous acquis des connaissances et un niveau d'expérience bénéfique pouvant consolider notre acquis théorique dans le cadre de notre formation académique.

Nous espérons que ce modeste travail profitera aux futurs étudiants ou d'autres applications regroupant plusieurs services.

Avec l'évolution de l'internet et beaucoup de temps disponible, plusieurs améliorations peuvent alors être envisagées. Néanmoins, notre API est limitée car elle assure la sécurité que sur un site web. Nous prévoyons la réalisation d'un logiciel destiné à agir avec notre API afin d'assurer une sécurité sans faille entre le client et le serveur web. Cette perspective sera une continuation pour effectuer une amélioration profonde de notre API pour protéger le client et le serveur web ou le serveur de base de données.

# Références bibliographiques

[1] PROTECTION AUTOMATIQUES DES APPLICATIONS WEB CONTRE L'ATTAQUE PAR INJECTION SQL, Mémoire de Maitrise Informatique, Yassin Mohamed, Février 2014, Université du Québec à Montréal.

[2] Wikipédia. Sécurité des systèmes d'information. http://fr.wikipedia.org/wiki/S%C3%A9curit%C3%A9_des_syst%C3%A8mes_d'information, consulté le 10 Avril 2022.

[3] SECURITE DES ECHANGES DE DONNEES CLIENTS-SERVEURS, Mémoire de fin d'études, M. DERRIDJ, Octobre 2010, Ecole Supérieure de Génie Informatique

[4] Guy BÉGIN. Inf8750 - sécurité des systèmes informatiques. Notes de cours, Hiver 2010.

[5] Wikipédia. Web application. http://jen.wikipedia.org/wiki/Web_application, consulté le 10 Avril 2022 .

[6] Étude et optimisation d'un nouveau mécanisme de sécurité pour protéger les applications Web contre les attaques par injection de code SQL, Serge Rufin N gansop. Rapport de Bibliographie, ENST Bretagne - INSA, 2009.

[7] Network Working Group. Hypertext transfer protocol - http/ 1.1. http ://www.w3.org/ Protocols/ rfc2616/ rfc2616.ht ml, consulté le 15 Avril 2022.

[8] A classification of sql injection attacks and countermeasures. William G. J. Halfond, Jeremy Viegas, and Alessandro Orso. *Georgia Institute of Technology*, pages 1- 11, 2006.

[9] Développement d'application Web. http://dessigsi2004.free.fr/dess/Cours/DAI/Dai.pdf, consulté le 10 Avril 2022.

[10] Contôle d'accès défaillants (Broken Acces Contol). https://owasp.org/Top10/A01_2021-Broken_Access_Control, consulté le 15 Avril 2022.

[11] Défaillances Cryptographiques (Cryptographic Failures). https://owasp.org/Top10/A02_2021-Cryptographic_Failures/, consulté le 15 Avril 2022.

[12] Injection. https://owasp.org/Top10/A03_2021-Injection/, consulté le 15 Avril 2022.

[13] Conception non sécurisée (Insecure design). https://owasp.org/Top10/A04_2021-Insecure_Design/, consulté le 15 Avril 2022.

[14] Mauvaise configuration de sécurité (Sécurity Misconfiguration). https://owasp.org/Top10/A05_2021-Security_Misconfiguration/, consulté le 15 Avril 2022.

[15] Composants vulnérables et obsolètes (Vulnerable and Outdated Components). https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/, consulté le 16

Avril 2022.
[16] Echec d'identification et d'authentification (Identification and Authentification Failures). https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/, consulté le 16 Avril 2022.
[17] Echec de l'intégrité du logiciel et des données (Software and Data Intégrity Failures). https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/, consulté le 16 Avril 2022.
[18] Echec de journalisation et de surveillance de la sécurité (Security Logging and Monitoring Failures). https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/, consulté le 17 Avril 2022.
[19] Contrefaçon de demande côté serveur (Server-Side Request Forgery). https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/, consulté le 17 Avril 2022.
[20] Base de Données. Georges Gardin, edition Eyrolles 2003.
[21] Cours Base de Données. Olivier GUIBERT. Département Informatique de l'Institut Universitaire de Technologie de l'Université Bordeaux 1, Janvier 2018.
[22] Etude de sécurité en base de données avec une application pour le contrôle d'accès. EL HADJ MIMOUNE Khadidja, MERABET Meriem. Mémoire de fin d'étude Master présenté 29 Septembre 2011, Université Abou Bekr Belkaid Tlemcen.
[23] Conception et réalisation d'une base de données pour la gestion de facturation, présenté le 04 Mai 2011.
[24] Généralités SGBD : Systèmes de gestion de base de données. http://div_info.110mb.com, consulté le 3 Mai 2022.
[25] La Sécurité des Bases de Données. http://didier.deleglise.free.fr/SGBD/securite/index_securite_sgbd.htm#securite, consulté le 3 Mai 2022.
[26] Database Security Threats. https : //www.imperva.com/learn/data-security/database-security/, consulté le 3 Mai 2022.
[27] Une API, qu'est-ce que c'est ? https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces#innovation-par-les-api, consulté le 24 Juin 2022.
[28] Designing Web APIs. Brenda Jin, Saurabh Sahni & Amir Shevat, édition O'REILLY, 2018.