CCN-CERT

BP/28

Recommandations sur le développement sécurisé

RAPPORT DE BONNES PRATIQUES

JANVIER **2023**

ccn-cert centro criptológico nacional
CCN centro criptológico nacional

Editeur :

MINISTERIO DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA

CCN
centro criptológico nacional

Paseo de la Castellana 109, 28046 Madrid

Date d'émission : janvier 2023

LIMITATION DE LA RESPONSABILITÉ

Ce document est fourni conformément aux termes contenus dans le présent document, rejetant expressément tout type de garantie implicite qui pourrait y être liée. En aucun cas, le Centre National de Cryptologie ne peut être tenu responsable des dommages directs, indirects, fortuits ou extraordinaires dérivés de l'utilisation des informations et des logiciels indiqués, même s'il a été averti d'une telle possibilité.

AVIS JURIDIQUE

Catalogue des publications de l'administration générale de l'État
https://cpage.mpr.gob.es

INDEX

Index

1. Introduction

Objectifs

Ce document est destiné à aider les équipes de développement à comprendre les contrôles de sécurité les plus courants qui doivent être appliqués au cours du cycle de vie du développement logiciel. Les guides de développement sécurisé fournissent aux développeurs un ensemble de recommandations à suivre qui leur permettent de construire des applications avec des techniques de sécurité.

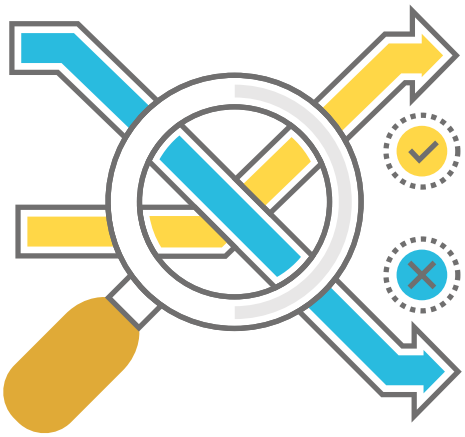
La mise en place de mesures de sécurité pendant le développement de l'application, lors des phases de conception et de codage, permet non seulement de jeter les bases de la sécurité contre les vulnérabilités les plus courantes, mais aussi de réduire les coûts futurs, car il est plus coûteux de corriger les problèmes à un stade ultérieur.

Les guides de développement sécurisé fournissent aux développeurs un ensemble de recommandations à suivre qui leur permettent de construire des applications avec des techniques de sécurité

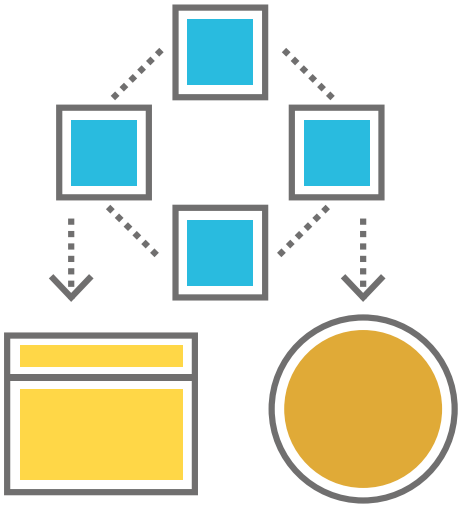
Périmètre

Quelle que soit la méthodologie de développement utilisée, la définition des contrôles de sécurité doit commencer dès la conception, voire avant, et se poursuivre tout au long du cycle de vie de l'application afin de garantir que l'application continuera à disposer des mesures de sécurité pertinentes en cas de modification de la disposition initiale suivant les besoins de l'entreprise.

La création de logiciels sécurisés implique des activités à plusieurs niveaux, non seulement pour se conformer aux politiques de sécurité internes, mais aussi pour respecter la loi et les réglementations externes telles que HIPAA, PCI ou GDPR. Le logiciel qui en résulte doit mettre en œuvre des fonctions de sécurité qui répondent à ces exigences.

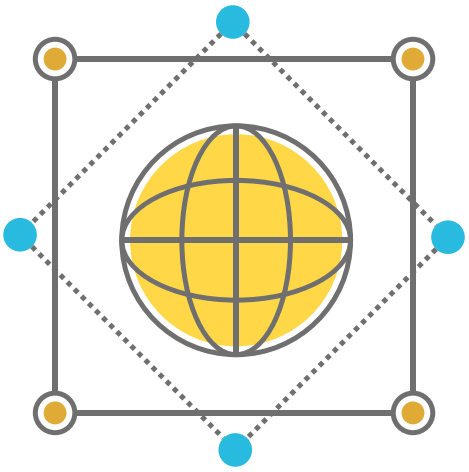


En outre, au cours du processus de modélisation des menaces d'un projet, lorsqu'elle est correctement combinée avec l'ingénierie des exigences de sécurité et les principes de conception sécurisée, elle permet à l'équipe de développement d'identifier les caractéristiques de sécurité nécessaires pour garantir l'intégrité, la confidentialité et la disponibilité des données impliquées dans le projet.



Le guide est organisé comme suit :

- **Recommandations de sécurité pour les aspects suivants :**
  - Architecture
  - Autorisations
  - Authentification
  - Gestion des sessions
  - Validation des paramètres d'entrée et de sortie
  - Gestion des erreurs
  - Journal sécurisé
  - Cryptographie
  - Gestion sécurisée des fichiers
  - Sécurité des transactions
  - Sécurité des communications
  - Protection des données
- **Indications supplémentaires pour Python**
- **Checklist des contrôles de sécurité**
- **Vulnérabilités et contrôles de sécurité**
- **Mesures de sécurité de l'ENS et contrôles de sécurité**
- **Annexes à la fin du document avec les contrôles de sécurité de base et avancés**



2. Architecture sécurisée

Une architecture sécurisée solide est fondamentale pour la construction du logiciel, car elle constitue la base sur laquelle le logiciel est construit et développé.

Pour atteindre cet objectif, il est nécessaire d'identifier les composants utilisés, de s'assurer qu'ils ne présentent pas de vulnérabilités connues et qu'ils sont correctement mis à jour.

L'architecture doit toujours être conçue en tenant compte des exigences de sécurité afin d'éviter ou de limiter les menaces potentielles pour la sécurité.

2.1. Risques potentiels

Les menaces potentielles pour une application peuvent être multiples et de haute probabilité lorsque :

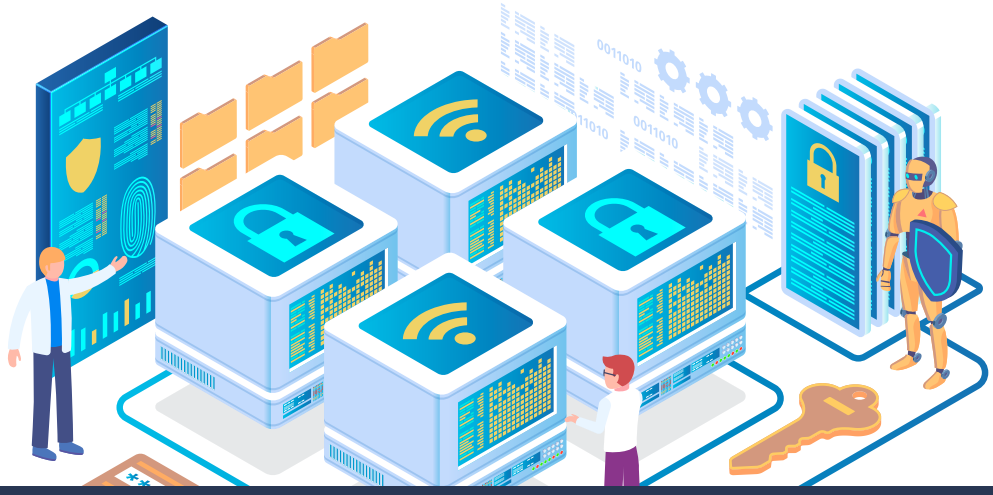
- **Des composants présentant des vulnérabilités connues sont utilisés.**
- **Des composants périmés, obsolètes ou non pris en charge sont utilisés.**
- **Des composants non identifiés sont utilisés.**
- **Des ports non-essentiels sont maintenus ouverts.**

2.2. Recommandations en matière de sécurité

- **Identifier tous les composants de l'architecture.**
  - Les composants qui n'ont pas été correctement identifiés constituent des risques potentiels pour la sécurité.
- **Effectuer un examen du bastion de chaque dispositif matériel du point de vue de la sécurité :**
  - Examen des configurations. Assurez-vous que les configurations sont les plus sécurisées : pas d'options de débogage activées, pas d'utilisateurs et de mots de passe par défaut, etc.
  - Vérification des ports. S'assurer que seuls les ports de communication strictement nécessaires sont ouverts.
  - Statut de la dernière mise à jour du système.
  - Identification de tous les composants du système : bibliothèques, modules, cadres, services, etc.
  - Pour chaque composant du système, effectuer le même examen du bastion des configurations et du statut des mises jour.
- **À partir du résultat de l'examen précédent, obtenir un rapport des composants dans lesquels sont détectées des vulnérabilités pour lesquelles il n'existe actuellement aucun correctif de sécurité et analyser leur niveau de risque au sein de l'application. Il se peut que certaines vulnérabilités détectées ne représentent pas un risque réel pour l'application ou n'aient pas un impact pertinent.**
- **Pour les vulnérabilités qui présentent un risque réel, surveiller de près les composants vulnérables afin qu'ils soient mis à jour dès que possible.**
- **Réaliser une étude sur la manière dont les problèmes de sécurité créés par ces vulnérabilités aux risques pourraient être évités ou atténués par des systèmes de sécurité alternatifs.**

Assurez-vous que les configurations sont les plus sécurisées : pas d'options de débogage activées, pas d'utilisateurs et de mots de passe par défaut, etc.

- **Améliorer la sécurité du périmètre logique en installant des pare-feu, des dispositifs IDS ou similaires, ou en segmentant le réseau.**
- **Veiller à ce que les données soient protégées par des mécanismes d'autorisation entre les environnements au moyen d'une séparation physique ou logique et par des sauvegardes pour garantir leur disponibilité.**
- **Utiliser la version la plus récente du langage de programmation.**
- **Utiliser un environnement virtuel comme espace de travail du projet, si cela est possible selon le langage de programmation.**
- **Importation correcte des paquets en fonction du langage de programmation. Installation et importation de paquets, vérification approfondie de la sécurité des paquets à installer.**
- **Désactiver toutes les options de débogage en production pour éviter la fuite d'informations dans les messages d'erreur détaillés.**
- **Utiliser les outils de l'IDE qui effectuent des analyses sémantiques et de sécurité de base.**



2.3. Références

Modèles de conception. Design Patterns :
https://refactoring.guru/es/design-patterns [1]

OWASP : Conception d'applications Web sécurisées :
https://owasp.org/www-pdf-archive/APAC13_Ashish_Rao.pdf [2]

Domaines de la sécurité intérieure : Protection des infrastructures critiques
file:///Users/lagor/Downloads/BOE-400_Ambitos_de_la_Seguridad_Nacional_Proteccion_de_Infraestructuras_Criticas.pdf [3]

3. Authentification

L'authentification consiste à vérifier l'identité d'un utilisateur ou d'un dispositif afin de lui accorder l'accès à ses ressources ou à ses informations. Cette tâche nécessite généralement la présentation d'informations d'identification, telles qu'un nom d'utilisateur et un mot de passe, afin de vérifier que l'utilisateur est bien celui qu'il prétend être.

Il s'agit du principal domaine de contrôle de la sécurité. Le type et la méthode d'authentification doivent donc faire partie de la conception.

L'authentification consiste à vérifier l'identité d'un utilisateur ou d'un dispositif afin de lui accorder l'accès à ses ressources ou à ses informations

3.1. Types d'authentification

- **Authentification du réseau :** l'identité d'un utilisateur ou d'un dispositif est vérifiée par l'utilisation d'informations d'identification du réseau, telles qu'un nom d'utilisateur et un mot de passe, ou par la vérification d'un certificat numérique.
- **Authentification par mot de passe :** l'utilisateur fournit un nom d'utilisateur et un mot de passe pour accéder à un système ou à une ressource.
- **Authentification par jeton :** l'utilisateur reçoit un jeton physique ou numérique qu'il doit fournir pour accéder à un système ou à une ressource.
- **Authentification à deux facteurs :** l'utilisateur doit fournir deux (2) méthodes de vérification de son identité, telles qu'un mot de passe et un code envoyé sur son téléphone portable.
- **Authentification biométrique :** les caractéristiques physiques de l'utilisateur, telles que son empreinte digitale ou sa voix, sont utilisées pour vérifier son identité.

3.2. Méthodes d'authentification

3.2.1. Authentification de base

L'authentification de base est une méthode d'**authentification réseau**. Dans cette méthode d'authentification, l'utilisateur fournit un nom d'utilisateur et un mot de passe sous forme de texte clair (base64), c'est pourquoi elle est donc considérée comme une forme d'authentification non sécurisée. D'autres méthodes, telles que l'authentification à deux facteurs ou l'authentification basée sur des certificats numériques, sont recommandées.

RISQUES

- **Attaque de l'intercepteur (MitM : Man-in-the-Middle) :** Les informations d'authentification sont envoyées en texte clair, ce qui signifie qu'elles peuvent être facilement interceptées et lues par toute personne ayant accès aux informations décryptées sur le réseau.
- **Réutilisation de clés :** si le même mot de passe est utilisé pour plusieurs sites ou systèmes, un attaquant qui obtient le mot de passe par le biais de l'authentification de base peut l'utiliser pour accéder à d'autres ressources protégées.
- **Attaque par force brute ou par dictionnaire :** ce type d'authentification ne fournit pas une protection adéquate contre ces attaques, où un attaquant tente de deviner les mots de passe en utilisant des programmes automatisés.
- **Authentification faible :** ne permet pas à un utilisateur de prouver son identité de manière sûre et fiable, ce qui empêche la mise en œuvre de mesures de sécurité plus fortes, telles que l'authentification à deux facteurs ou l'authentification basée sur des certificats numériques.

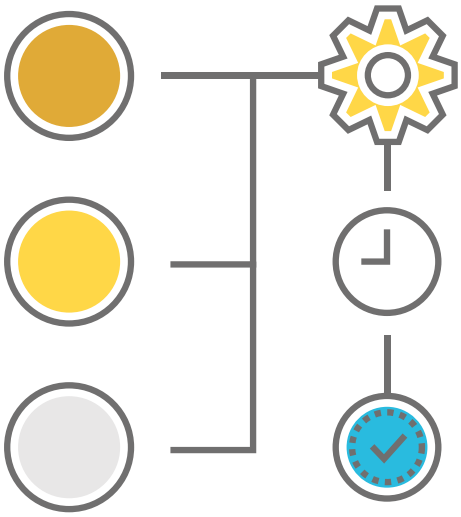
3.2.2. Authentification via les formulaires

L'authentification par formulaire est un type d'**authentification par mot de passe** pour un site web. Dans cette méthode d'authentification, l'utilisateur fournit son nom d'utilisateur et son mot de passe via un formulaire sur une page web. Le serveur web vérifie les informations d'authentification reçues et si elles sont correctes, il autorise l'accès à l'utilisateur.

L'authentification par formulaire peut être une forme d'authentification sûre si des mesures appropriées sont utilisées pour protéger les informations d'authentification, comme le cryptage des informations en transit et l'utilisation de mots de passe forts. Cependant, elle présente également certains risques, comme la possibilité pour un attaquant d'obtenir les informations d'authentification par des techniques d'ingénierie sociale ou par l'utilisation d'un logiciel de force brute.

RISQUES

- **Attaque de l'intercepteur (MitM : Man-in-the-Middle) :** les informations d'authentification peuvent être interceptées et utilisées pour accéder à la ressource protégée si des techniques appropriées ne sont pas utilisées pour protéger ces informations, comme le cryptage des communications ou l'utilisation de mots de passe forts.
- **Réutilisation des clés :** si le même mot de passe est utilisé pour plusieurs sites ou systèmes, un attaquant qui obtient le mot de passe par le biais de formulaires d'authentification peut l'utiliser pour accéder à d'autres ressources protégées.
- **Attaque par force brute ou par dictionnaire :** le système ne fournit pas non plus de protection adéquate contre ces attaques, au cours desquelles un attaquant tente de deviner les mots de passe à l'aide de programmes automatisés.
- **Authentification faible :** ne permet pas à un utilisateur de prouver son identité de manière sûre et fiable, ce qui empêche la mise en œuvre de mesures de sécurité plus fortes, telles que l'authentification à deux facteurs ou l'authentification basée sur des certificats numériques.



3.2.3. Authentification implicite

L'utilisateur ne doit pas fournir explicitement des informations d'identification pour accéder à une ressource protégée. D'autres formes d'identification sont utilisées, comme l'adresse IP de l'utilisateur, les informations stockées dans un *cookie* ou un jeton. Il s'agit d'un type d'**authentification par réseau** ou **par jeton**, selon la manière choisie pour fournir cette identification.

La manière la plus sûre d'utiliser cette méthode est d'utiliser un code de hachage cryptographique basé sur celle-ci comme mot de passe pour obtenir un jeton suffisamment robuste et long pour qu'il soit inutile d'utiliser une attaque par force brute contre lui. Le serveur compare le jeton reçu avec le jeton qu'il a calculé et stocké pour l'utilisateur demandant l'accès.

RISQUES

- **Accès non autorisé :** si des informations telles que l'adresse IP de l'utilisateur sont utilisées pour déterminer si l'utilisateur a accès à une ressource protégée, un attaquant peut usurper ces informations et obtenir un accès non autorisé à la ressource.
- **Accès non autorisé :** si des *cookies* sont utilisés pour stocker des informations d'authentification, un attaquant peut avoir accès au *cookie* et l'utiliser pour accéder à la ressource protégée.
- **Attaque par force brute ou par dictionnaire :** lorsque des algorithmes de cryptage suffisamment puissants, tels que MD5, n'ont pas été utilisés, le système peut être vulnérable à de telles attaques.
- **Authentification faible :** ne permet pas à un utilisateur de prouver son identité de manière sûre et fiable, ce qui empêche la mise en œuvre de mesures de sécurité plus fortes, telles que l'authentification à deux facteurs ou l'authentification basée sur des certificats numériques.

3.2.4. Authentification du client HTTP

Il s'agit d'un type d'**authentification basé sur le réseau** dans lequel une paire de clés cryptographiques, une clé publique et une clé privée, est utilisée pour vérifier l'identité d'un utilisateur ou d'un dispositif. La clé publique est utilisée pour crypter les informations et la clé privée est utilisée pour les décrypter. L'utilisateur envoie un certificat numérique contenant sa clé publique pour que le serveur crypte le contenu, puis utilise sa clé privée pour le décrypter.

L'authentification par certificat de clé publique est considérée comme l'une des formes d'authentification les plus sûres, car elle permet à un utilisateur de prouver son identité de manière fiable et sécurisée. Ceci à condition que l'algorithme de cryptage soit suffisamment fort pour que cette méthode d'authentification soit réellement sécurisée. Cette méthode est généralement utilisée pour les communications HTTPS (HTTP sur SSL/TLS). Le seul point négatif de cette méthode est qu'elle peut être plus compliquée et plus coûteuse à mettre en œuvre que d'autres formes d'authentification.

Le certificat de clé publique du client est émis par une entité de confiance, telle qu'une autorité de certification (CA), qui fournit également une identification pour le porteur.

L'authentification par certificat de clé publique est considérée comme l'une des formes d'authentification les plus sûres, car elle permet à un utilisateur de prouver son identité de manière fiable et sécurisée

RISQUES

- **Accès non autorisé :** si un attaquant parvient à obtenir la clé privée d'un utilisateur, il peut l'utiliser pour obtenir un accès non autorisé aux ressources protégées de cet utilisateur.

  Si un faux certificat est émis avec une fausse clé publique, il pourrait être utilisé pour obtenir un accès non autorisé à des ressources protégées.
- **Attaque de l'intercepteur (MitM : Man in the Middle) :** en interceptant un certificat en transit, il pourrait être utilisé pour accéder à des ressources protégées de manière non autorisée.
- **Manipulation d'un certificat SSL :** mise en œuvre de certificats qui n'ont pas été vérifiés par une autorité de certification de confiance ou de certificats auto-signés qui pourraient donner un faux sentiment de sécurité.

3.2.5. Authentification Windows

Un type d'**authentification basé sur un mot de passe** dans lequel l'identité d'un utilisateur qui tente d'accéder à un ordinateur ou à une ressource protégée par MS Windows est vérifiée. Pour ce faire, on utilise un nom d'utilisateur et un mot de passe, qui sont vérifiés par rapport aux informations stockées sur un serveur d'authentification ou dans un répertoire local. MS Windows utilise deux (2) protocoles d'authentification : Kerberos et NTLM.

Un type d'authentification basé sur un mot de passe dans lequel l'identité d'un utilisateur qui tente d'accéder à un ordinateur ou à une ressource protégée par MS Windows est vérifiée

Kerberos utilise un serveur d'authentification centralisé et des clés cryptées pour vérifier en toute sécurité l'identité des utilisateurs et NTLM utilise le système de fichiers local de l'ordinateur.

Dans un domaine Windows NT ou un environnement Active Directory, l'authentification des utilisateurs est réalisée à l'aide d'un serveur d'authentification centralisé. Lorsqu'un utilisateur tente de se connecter, son ordinateur envoie une demande d'accès au serveur d'authentification et vérifie l'identité de l'utilisateur à l'aide des informations d'identification stockées dans l'Active Directory. Si les informations d'identification sont valides, le serveur d'authentification émet un ticket d'accès permettant à l'utilisateur d'accéder au site.

Cette méthode d'authentification est la plus adaptée aux environnements d'entreprise (Intranet) où un contrôle centralisé de l'accès aux ressources du réseau est nécessaire. En outre, en utilisant un Active Directory, il est possible de gérer de manière centralisée les comptes d'utilisateurs et leurs autorisations d'accès aux différentes ressources du réseau.

Cette méthode d'authentification présente un certain nombre de risques de sécurité, dont beaucoup dépendent des politiques de sécurité définies par l'administrateur du serveur MS Windows.

RISQUES

- **Attaques par dictionnaire :** elles consistent à deviner les mots de passe en utilisant des programmes qui tentent des combinaisons courantes de lettres et de chiffres.
- **Attaques par force brute :** elles tentent de deviner le mot de passe d'un utilisateur en utilisant des programmes qui génèrent et testent des combinaisons de caractères jusqu'à ce que la bonne soit trouvée.
- **Attaques de l'intercepteur (MitM) :** elles interviennent dans la communication entre l'utilisateur et le serveur d'authentification afin d'obtenir les informations d'identification de l'utilisateur.
- **Attaques par rejeu :** capture et réutilisation de tickets de connexion valides émis par le serveur d'authentification, ce qui permettrait à un attaquant d'obtenir un accès sans avoir à connaître le mot de passe de l'utilisateur. Les systèmes d'exploitation MS Windows comportent des mesures de sécurité qui empêchent la réutilisation des tickets d'accès.
- **Attaques par *phishing* :** elles consistent à envoyer de faux courriels qui semblent provenir d'une source fiable, avec des messages qui utilisent des techniques d'ingénierie sociale afin d'obtenir les informations d'identification d'un utilisateur.
- **Vulnérabilités des logiciels :** les systèmes d'exploitation MS Windows et les autres logiciels utilisés pour l'authentification contiennent des vulnérabilités qui peuvent être identifiées et exploitées par des attaquants.
- **Défaillances humaines en matière de sécurité :** erreurs ou oublis des utilisateurs, tels que l'utilisation de mots de passe faibles, le partage de leurs informations d'identification ou le fait de permettre qu'elles soient découvertes.

3.2.6. Authentification avec Passport

Microsoft Passport est un service d'authentification utilisé par certains systèmes Windows et certaines applications Microsoft pour vérifier l'identité des utilisateurs. Ce service utilise un compte Microsoft, tel qu'un compte Outlook ou un compte Skype, pour vérifier l'identité de l'utilisateur.

Elle est basée sur l'utilisation d'informations d'identification à usage unique, qui sont envoyées au serveur d'authentification à la place du

mot de passe de l'utilisateur. Cela empêche les attaquants de deviner le mot de passe de l'utilisateur en utilisant des techniques de force brute ou de dictionnaire. En outre, il utilise le cryptage pour se protéger contre le risque d'attaques de l'intercepteur.

À partir de MS Windows 10, le système de Passport a évolué en utilisant l'authentification à deux facteurs. L'une est l'enregistrement de l'appareil et l'autre l'authentification biométrique par un geste (Windows Hello) ou un code PIN.

RISQUES

- **Dépendance à l'égard d'un compte Microsoft :** Pour utiliser Microsoft Passport, vous devez disposer d'un compte Microsoft.
- **Vulnérabilités logicielles :** Microsoft Passport peut présenter des vulnérabilités susceptibles d'être exploitées par des attaquants.
- **Risques de *phishing* et d'erreur humaine :** les mêmes problèmes que ceux indiqués pour l'authentification MS Windows.

A partir de MS Windows 10, el sistema de Passport ha evolucionado utilizando Authentification de dos factores. Uno de ellos es el registro del dispositivo y el otro una Authentification biométrica o un PIN

3.3. Risques potentiels

En résumé, les menaces et les risques les plus courants dus à l'absence de contrôles d'authentification de sécurité dans les applications ou à des méthodes d'authentification insuffisamment sécurisées sont les suivants :

- **Attaques par force brute :** utilisation de programmes qui génèrent et testent des combinaisons de caractères pour deviner les mots de passe des utilisateurs et accéder à leurs comptes. Si les mots de passe sont faibles ou faciles à deviner, ces attaques peuvent réussir.
- **Attaques par dictionnaire :** Utiliser des logiciels pour tenter de deviner les mots de passe en testant les mots du dictionnaire les plus couramment utilisés comme mots de passe et en essayant différentes variations de lettres et de chiffres. Ces attaques auront plus de chances de réussir, et en moins de temps, si les mots utilisés comme mots de passe sont faibles ou trop courants.
- **Attaques de l'intercepteur (MitM) :** si les communications sont interceptées, les informations d'identification, le jeton ou le ticket d'accès peuvent être obtenus (attaque par rejeu). Si, en outre, les

communications ne sont pas sécurisées ou si des formes d'authentification en texte clair sont utilisées, les attaques peuvent être plus fructueuses.

- **Attaques par relecture :** une fois qu'un attaquant a obtenu un ticket d'accès, il peut le réutiliser pour obtenir un accès.
- **Vulnérabilités logicielles :** les systèmes d'exploitation et les applications utilisés pour l'authentification peuvent présenter des vulnérabilités susceptibles d'être exploitées par des attaquants.
- **Recensement des utilisateurs :** l'attaquant pourrait trouver les utilisateurs enregistrés sur le système en prouvant leur identité et en analysant les réponses du serveur, soit par le temps de réponse, soit par les messages d'erreur.
- **Spoofing :** obtention d'informations d'identification en se faisant passer pour le serveur d'authentification, en faisant croire à la victime qu'elle se trouve au bon endroit pour saisir ses informations d'identification et obtenir un accès. Le service pourrait même transmettre les informations au véritable serveur et les rediriger de manière authentifiée afin que la victime ne remarque pas la supercherie.
- ***Contournement* de l'authentification :** comprend les techniques qui permettent l'authentification en contournant les mesures de sécurité associées. Par exemple, l'accès à un compte utilisateur par injection de code.
- **Vol d'identité :** il s'agit du cas où l'attaquant obtient l'identité de la victime et l'utilise pour effectuer certaines actions. Cela peut se faire, par exemple, en volant un *cookie* de session de la victime.

3.4. Recommandations en matière de sécurité

- S'assurer que les mots de passe ne sont pas stockés dans un format lisible, de sorte que si le système ou la ressource contenant les mots de passe est compromis, l'utilisateur malveillant ne peut toujours pas les utiliser. Une bonne méthode pourrait être l'utilisation de fonctions qui garantissent l'irréversibilité de l'opération, comme l'utilisation de fonctions de hachage fortes.

- Veiller à ce que chaque page de l'application dispose d'un lien de déconnexion, que la session expire lorsque l'utilisateur se déconnecte et qu'elle expire après une période raisonnable de non-activité.

- Ne jamais exposer les informations d'identification dans l'URL.

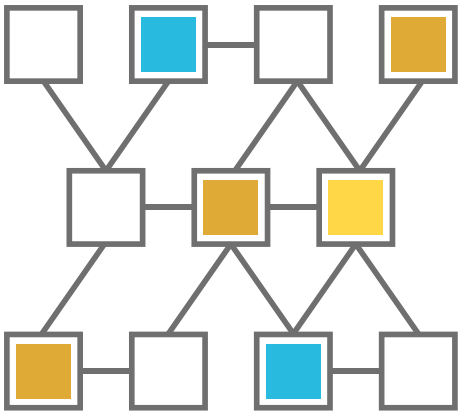
- Lors de l'utilisation des formulaires, utiliser les méthodes POST pour envoyer des informations entre le client et le serveur.

- Utiliser l'authentification multifactorielle pour renforcer la sécurité des opérations ayant accès à des ressources sensibles ou à des applications accessibles au public.

- Utiliser une authentification à deux facteurs pour l'utilisateur pour les fonctions critiques de l'application, comme la modification des mots de passe ou l'accès à des ressources particulièrement sensibles.

- Mettre en place un verrouillage du compte après trois (3) tentatives de connexion infructueuses et un moyen de contacter l'administrateur pour déverrouiller le compte.

- Mettre en œuvre CAPTCHA pour atténuer les attaques par force brute sur les applications exposées sur Internet.



- Éviter l'énumération des utilisateurs, en fournissant des messages d'erreur génériques en cas d'échec de l'authentification, tels que « L'utilisateur et/ou le mot de passe fournis ne sont pas corrects ».

- Éviter l'énumération des utilisateurs en prévoyant des temps de réponse différents en cas d'utilisateur inexistant ou de mot de passe incorrect. Il est nécessaire d'inclure des délais d'attente aléatoires dans les deux cas afin qu'il soit impossible de reconnaître les cas en mesurant les temps de réponse.

- Éviter l'énumération des utilisateurs dans les procédures de récupération de mot de passe qui nécessitent la saisie du nom d'utilisateur.

- Désactiver l'attribut « autocomplétion » du champ du mot de passe dans l'application, car il est activé par défaut.

- Appliquer les exigences de complexité des mots de passe fixées par la politique ou la réglementation, et s'assurer que ces exigences respectent les règles de sécurité minimales telles que :
  - Il doit comporter un minimum de huit (8) caractères et un maximum raisonnable..
  - Il doit contenir au moins trois (3) des caractères suivants :
    - une lettre majuscule
    - une lettre minuscule
    - un numéro
    - un caractère spécial.

- Ne pas stocker de manière persistante le *cookie* d'authentification sur l'ordinateur du client et ne pas l'utiliser à d'autres fins telles que la personnalisation.

- S'assurer que la session est différente chaque fois qu'un utilisateur s'est connecté avec succès à l'application.

- Enregistrer toutes les tentatives d'authentification réussies ou échouées sans exposer la clé utilisée.

- Enregistrer les tentatives d'accès malveillant dans un journal de sécurité spécifique : détection de multiples tentatives d'authentification échouées, détection de tentatives d'injection, telles que SQL ou LDAP, détection d'utilisateurs multiples pour les mêmes IP, etc

Appliquer les exigences de complexité des mots de passe fixées par la politique ou la réglementation, et s'assurer que ces exigences respectent les règles de sécurité minimales

3.5. Exemple

Cet exemple crée un hachage et un sel pour un mot de passe en appelant **getSaltedHash(String password)**. Cette méthode renvoie une chaîne contenant le sel et le hachage séparés par un | (barre verticale).

Pour vérifier si un mot de passe donné est correct, appelez **check(String password, String stored)**, en passant le mot de passe à vérifier avec le hachage/sel stocké. Cette méthode retournera true si le mot de passe est correct.

Il est important d'utiliser **SecureRandom** pour générer le **salt** de manière sécurisée, aléatoire et unique pour chaque mot de passe. Un nombre suffisamment élevé d'itérations est utilisé pour le calcul du hachage (10 000 dans cet exemple) afin de renforcer la sécurité et de rendre plus difficile le calcul du hachage d'un mot de passe donné par des attaquants par force brute. Il est également important d'utiliser une fonction de hachage sécurisée telle que SHA-512, qui résiste aux attaques par collision et dont le calcul est relativement rapide.

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.Base64;

public class SecureAuth {

   private static final int ITERATION_COUNT = 10000;
   private static final int KEY_LENGTH = 512;

   public static String getSaltedHash(String password) throws IllegalStateException {
      byte[] salt = SecureRandom.getInstanceStrong().generateSeed(KEY_LENGTH / 8);
      return Base64.getEncoder().encodeToString(salt) + "|" + hash(password, salt);
   }

   public static boolean check(String password, String stored) throws IllegalStateException{
      String[] saltAndPass = stored.split("\\|");
      if (saltAndPass.length != 2) {
         throw new IllegalStateException("Use '<salt>|<hash>'");
      }
      byte[] salt = Base64.getDecoder().decode(saltAndPass[0]);
      String hashOfInput = hash(password, salt);
      return hashOfInput.equals(saltAndPass[1]);
   }

   private static String hash(String password, byte[] salt) throws IllegalStateException {
      MessageDigest md = MessageDigest.getInstance("SHA-512");
      md.update(salt);
      byte[] hashedPassword = md.digest(password.getBytes("UTF-8"));
      int iterations = ITERATION_COUNT;
      while (iterations-- > 0) {
         hashedPassword = md.update(hashedPassword);
      }
      hashedPassword = md.digest(hashedPassword);
      return Base64.getEncoder().encodeToString(hashedPassword);
   }
}
```

3.6. Références

Authentification sécurisée en Java :
https://docs.oracle.com/javaee/5/tutorial/doc/bncbe.html#bncbn [4]

Python : Bibliothèque pour la mise en œuvre d'OTP :
https://pypi.org/project/pyotp/ [5]

4. Autorisations

L'autorisation est chargée de déterminer les actions qu'un utilisateur ou un système est autorisé à effectuer. L'authentification est donc une condition préalable à l'autorisation, car il est nécessaire de vérifier l'identité d'un utilisateur ou d'un système avant de déterminer les actions qu'il est autorisé à effectuer.

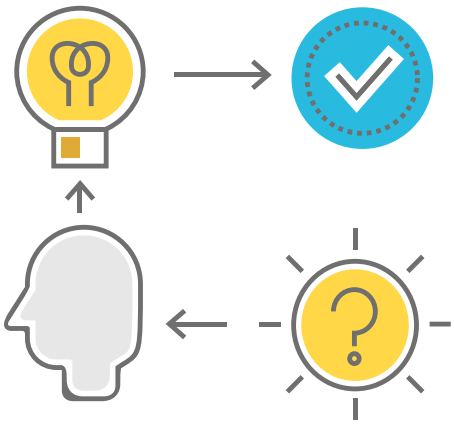
Il s'agit du deuxième contrôle de sécurité après l'authentification et il est étroitement lié à l'autorisation des rôles d'entreprise de l'application.

4.1. Risques potentiels

Plusieurs vulnérabilités peuvent apparaître en l'absence de contrôles de sécurité adéquats lors de la mise en œuvre de l'autorisation. Voici quelques-uns des risques principaux et courants identifiés dans les applications en raison de l'absence de contrôles de sécurité sur les autorisations.

- **Accès non autorisé :** si l'autorisation n'est pas correctement mise en œuvre, des utilisateurs non autorisés peuvent avoir accès à des ressources et à des opérations auxquelles ils ne devraient pas avoir droit.
- **Escalade verticale des privilèges :** un utilisateur peut accéder aux fonctionnalités d'un autre utilisateur disposant de privilèges plus élevés.
- **Escalade horizontale des privilèges :** un utilisateur peut accéder aux fonctionnalités d'un autre utilisateur ayant le même niveau d'accès.

- **Divulgation d'informations sensibles :** une application donne plus d'informations que nécessaire à l'utilisateur et pourrait être utilisée par un attaquant à des fins malveillantes.
- **Atteinte à la vie privée :** si l'attaquant finit par avoir accès aux données relatives à la vie privée d'autres utilisateurs.
- **Vol d'identité :** il s'agit du cas où l'attaquant obtient l'identité de la victime et l'utilise pour effectuer certaines actions.
- **Vol de données :** c'est lorsqu'un attaquant obtient illégitimement des données, qu'elles soient confidentielles ou non.
- **Disponibilité du service :** un accès non autorisé à des éléments critiques de l'application pourrait permettre à un attaquant de perturber ou d'endommager le service.
- **Manipulation des données :** un attaquant est capable d'intercepter et de manipuler les données échangées entre le client et le serveur.
- **Modification des journaux :** lorsqu'un attaquant parvient à accéder aux journaux des applications ou des systèmes et à les modifier, compromettant ainsi l'intégrité de la traçabilité des journaux.
- **Attaque par traversée de répertoire :** l'attaquant se déplace dans le système de fichiers du serveur, au-delà du cadre dans lequel il est censé agir et en dehors du contexte qui nécessiterait une autorisation.
- **Logique d'entreprise :** si l'application n'a pas été bien conçue, il peut arriver que dans le cas où un utilisateur ne suit pas la logique normale de l'application pour ses fonctions d'entreprise, il rencontre un comportement inattendu qui peut être exploité pour effectuer des opérations qui n'auraient pas dû être autorisées.



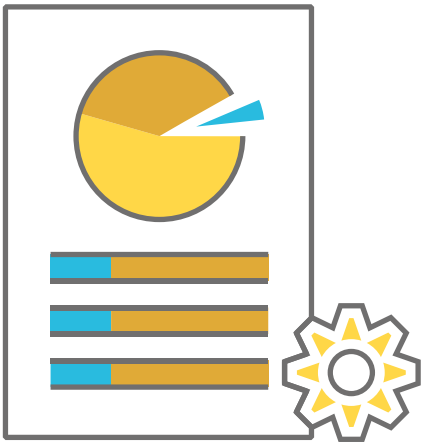
4.2. Recommandations en matière de sécurité

- Veiller à la mise en œuvre du principe du moindre privilège : les utilisateurs n'ont qu'un accès restreint aux fonctions et aux données dont ils ont réellement besoin pour effectuer leur travail normalement.
- Attribuer des autorisations et des privilèges aux rôles d'application, jamais directement aux utilisateurs. Les utilisateurs doivent avoir des rôles et leurs privilèges sont tirés de ces rôles.
- Vérifier que l'accès aux enregistrements confidentiels est protégé, afin que seuls les objets ou données autorisés et accessibles à chaque utilisateur puissent être atteints (par exemple, en se protégeant contre les utilisateurs qui manipulent un paramètre pour voir ou modifier le compte d'un autre utilisateur).
- Vérifier que la navigation dans les répertoires est désactivée, sauf si elle est délibérément activée. En outre, les applications ne doivent pas permettre la découverte ou la divulgation de fichiers ou de répertoires, tels que les dossiers Thumbs.ds, .DS_Store, .git ou .svn.
- S'assurer que les règles de contrôle d'accès sont appliquées du côté du serveur.
- Vérifier que tous les attributs, données et informations de politique d'utilisateur utilisés par les contrôles d'accès ne peuvent pas être manipulés par les utilisateurs finaux, sauf autorisation spécifique.
- Vérifier qu'il existe un mécanisme centralisé (y compris des bibliothèques appelant des services d'autorisation externes) pour protéger l'accès à chaque type de ressource protégée.

Vérifier que l'accès aux enregistrements confidentiels est protégé, afin que seuls les objets ou données autorisés et accessibles à chaque utilisateur puissent être atteints

- S'assurer que l'application utilise des jetons aléatoires anti-CSRF forts ou met en œuvre un autre mécanisme de protection des transactions.

- Enregistrer toutes les opérations sur des données sensibles dans un journal de sécurité spécifique contenant au moins : la date et l'heure de l'opération, l'opération (lecture, création, suppression, mise à jour), le nom des données, le processus, la fonction ou le service qui a généré l'opération, l'utilisateur, le rôle de l'utilisateur qui a le privilège de l'opération, le résultat de l'opération (réussie ou non) et un message facultatif sur le résultat de l'opération (en cas d'erreur).



4.3. Exemple

Utilisation du cadre Spring Security qui fournit un ensemble complet de fonctions de sécurité, notamment l'authentification et l'autorisation basée sur les rôles. Les annotations sont utilisées pour contrôler l'accès à certaines parties de votre application.

Dans l'exemple suivant, la méthode listUsers n'est accessible qu'aux utilisateurs ayant le rôle **ROLE_ADMIN**.

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GetMapping("/admin/users")
public String listUsers(Model model) {
  // Code pour lister les utilisateurs
}
```

4.4. Références

Java : Autorisation sécurisée
https://docs.oracle.com/en/java/javase/19/security/java-authentication-and-authorization-service-jaas1.html [6]

Python : Bibliothèque sécurisée à autorisation simple
https://pypi.org/project/python-authorization/ [7]

5. Gestion des sessions

Une session est une connexion active entre un utilisateur et le système. La gestion des sessions consiste à déterminer les actions sur les sessions qui servent à garantir la sécurité de l'autorisation, l'intégrité et la confidentialité des informations entre l'utilisateur et le système. Chaque accès authentifié et autorisé au système crée une nouvelle session dans le système qui stocke des informations temporaires sur les opérations uniques de l'utilisateur et la logique commerciale dans l'application pour la durée de l'accès de l'utilisateur. Chaque session est identifiée par un identifiant unique.

À partir de là, la façon dont l'utilisateur s'identifie auprès du serveur pour le reste des opérations peut varier :

- **Par le biais de *cookies* :** il s'agit du mécanisme le plus courant et le moins sûr. Un identifiant unique est généré dans le *cookie* du navigateur qui est utilisé pour identifier la session de l'utilisateur sur le serveur comme une session active et authentifiée. Elle est moins sûre car elle nécessite de déléguer la responsabilité de la persistance des *cookies* à des implémentations de navigateur plus ou moins sûres.
- **Par le biais de jetons :** cela équivaut au système précédent où le jeton est un identifiant d'accès qui permet d'accéder au système après que le jeton a été associé à l'ID de session correspondant. Ce jeton se trouve généralement dans les en-têtes de la demande. Il est plus sûr car il permet de modifier les jetons d'accès de temps en temps, au moyen d'un échange de jetons, sans qu'il soit nécessaire de modifier l'identifiant de session, qui serait toujours conservé en sécurité du côté du serveur.

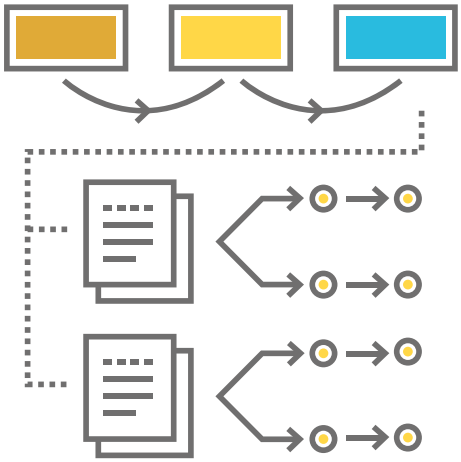
La gestion des sessions consiste à déterminer les actions sur les sessions qui servent à garantir la sécurité de l'autorisation, l'intégrité et la confidentialité des informations entre l'utilisateur et le système

5.1. Aspects sécuritaires

Les premiers aspects fondamentaux à prendre en compte dans la sécurité de la gestion des sessions sont les suivants :

- **Authentifier les utilisateurs de manière sécurisée afin de garantir que seules les personnes autorisées ont accès au système.**
- **Autoriser de manière appropriée l'utilisateur à qui l'accès a été accordé.**
- **Chiffrer avec un algorithme robuste les communications entre le client et le serveur.**

5.1.1. Session côté client



La session côté client, à laquelle on accorde souvent peu ou pas d'attention, est aussi importante, voire plus, que la session créée sur le serveur. Si cette session n'existe pas, les situations d'insécurité suivantes, entre autres, pourraient être créées à titre d'exemple :

- Un utilisateur termine sa tâche dans l'application, ouvre un autre onglet et passe à une autre tâche. La session du serveur se termine, mais le client ne sait pas que la session est terminée tant qu'il ne génère pas d'activité dans la fenêtre oubliée. Cet onglet pourrait exposer des informations sensibles que toute personne manipulant l'appareil pourrait voir et copier sans aucune activité.
- Un utilisateur doit remplir un formulaire contenant un grand nombre d'informations qui prennent du temps à compléter, et qui peuvent même être remplies à partir de requêtes vers d'autres sources qui requièrent son attention. Si le délai d'attente d'une session de serveur est de 30 minutes d'inactivité et qu'un utilisateur passe 2 heures à remplir le formulaire, lorsqu'il appuie sur « soumettre » sa session expire, tout son travail est perdu et il se retrouve avec un écran de connexion lui demandant des informations d'identification pour un nouvel accès. Le client a eu une activité pendant ces 2 heures, mais le serveur n'a pas pu la percevoir.

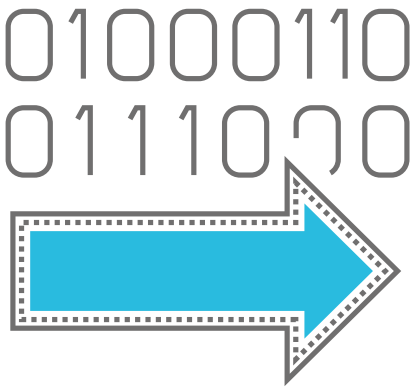
En outre, la mise en œuvre du concept de session du côté client permettrait d'afficher les avertissements d'expiration un certain temps avant qu'ils ne se produisent. La session côté client doit donc répondre à certaines exigences minimales en matière de sécurité :

- L'information persistante de l'ID de session du client doit être stockée associée à l'onglet, de sorte que lorsque l'onglet disparaît, toutes les informations associées à cette session disparaissent. Cela peut se faire par le biais du code de la page elle-même ou de l'objet sessionStorage.
- Les informations de session persistantes du client doivent toujours être supprimées de l'écran de connexion et créées lorsque la session est créée sur le serveur, pas avant.
- Permettre à la session du client d'effectuer des transactions vides auprès du serveur simplement pour indiquer au serveur que la session doit rester active et ne pas expirer. Ce comportement doit être appliqué à la demande sur les écrans qui requièrent beaucoup d'attention du côté client, et jamais par défaut.
- Comme pour les sessions de serveur, elles doivent contrôler la durée maximale d'inactivité avec une durée similaire ou inférieure à celle des sessions de serveur. En cas d'inactivité du côté client, la session client doit lancer une demande de déconnexion au serveur et le rediriger vers l'écran de connexion. Il peut également être conseillé de définir un délai d'attente maximal absolu pour la session.
- Les informations stockées dans ces sessions client doivent être minimales et nécessaires à la logique de navigation, et ne devraient pas contenir d'informations sensibles. Elles pourraient contenir le jeton d'accès au serveur pour éviter de le traîner dans toutes les requêtes (le cas échéant).
- Délai de connexion, pour éviter les attaques par fixation de session.
- Forcer la déconnexion du client et du serveur lors de la fermeture de la fenêtre ou de l'onglet du navigateur.

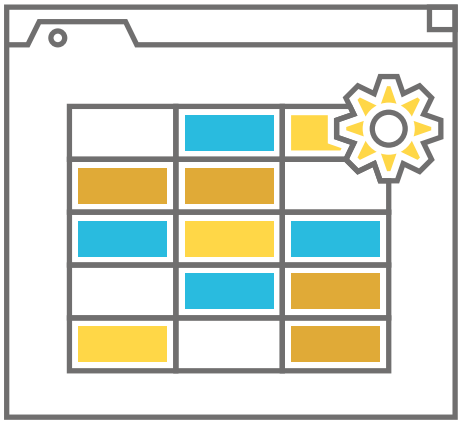
5.1.2. ID de session

Les contrôles de sécurité minimums associés à l'ID de session sont les suivants :

- Tout identifiant de session doit être unique, suffisamment aléatoire et d'une longueur appropriée fournie par un hachage cryptographiquement sécurisé à partir d'un nombre aléatoire, avec une longueur de clé significative.
- Le générateur de nombres aléatoires utilisé pour créer l'ID de session doit être un générateur de nombres aléatoires sécurisé.
- Les ID doivent être validés par le serveur pour s'assurer qu'ils ont un format valide et font partie de sessions actives et valides.



- Les ID de session ne doivent pas être enregistrés. Si cela est nécessaire pour la traçabilité de la session, utilisez un autre identifiant, mais n'utilisez jamais l'identifiant utilisé pour l'identification de la session dans le trafic client-serveur.
- Un nouvel identifiant de session doit être généré à chaque connexion ou lorsqu'il y a un changement du niveau de privilège de l'utilisateur, afin d'éviter les attaques par fixation de session.
- Le stockage et le suivi des identifiants de session actifs doivent être sécurisés afin d'empêcher tout accès par du personnel non autorisé.



5.1.3. Déconnexion

Lorsqu'une session de serveur est fermée, les contrôles de sécurité suivants doivent être pris en compte :

- Toujours rediriger l'utilisateur vers la page de login.
- S'assurer que le client supprimera toutes les informations relatives aux *cookies* ou aux jetons d'accès.
- S'assurer que le client supprimera toutes les informations de la session du client (le cas échéant).
- Enregistrer les déconnexions dans le journal de sécurité

5.1.4. Expiration de la session

Les contrôles de sécurité minimaux sur l'expiration des sessions sont les suivants :

- Chaque session doit avoir une durée de vie maximale raisonnable en cas d'inactivité pour éviter que la session ne reste active en permanence.
- Lorsque la session du serveur expire, vous devez supprimer toutes les informations relatives à cette session.
- Consigner les expirations de session dans le journal de sécurité.
- Définir une durée maximale absolue pour la durée d'une session.

5.1.5. Gestion des sessions dans les applications mobiles

Pour des raisons de convivialité, les applications mobiles exigent souvent que les sessions durent plus longtemps que les applications web. Les recommandations pour la gestion des sessions dans les applications mobiles sont les suivantes :

- Utiliser des jetons qui peuvent être retirés en cas de perte ou de vol de l'appareil.
- Le délai d'expiration de la session de l'application mobile doit être configuré et expirer en fonction de la sensibilité de l'application.
- Utiliser un magasin de données basé sur le serveur pour faciliter l'utilisation de la session sur plusieurs pages.
- Ne jamais utiliser un identifiant de périphérique comme jeton de session.

Utiliser un magasin de données basé sur le serveur pour faciliter l'utilisation de la session sur plusieurs pages

5.2. Risques potentiels

- **Prédiction de session :** se concentre sur la prédiction des valeurs d'identification de session qui permettent à un attaquant de contourner le schéma d'authentification d'une application.
- **Détournement de session :** L'attaque par détournement de session consiste à exploiter le mécanisme de contrôle de la session web, généralement géré par un jeton de session.
- **Fixation de session :** elle consiste à obtenir un ID de session valide (par exemple en établissant une connexion avec l'application), à inciter un utilisateur à s'authentifier avec cet ID de session, puis à détourner la session validée de l'utilisateur pour connaître l'ID de session utilisé.
- **Usurpation de session :** lorsque l'attaquant prend l'identité d'une autre entité pour commettre une fraude quelconque. Par exemple, un attaquant qui génère un site web malveillant sous l'apparence d'une banque légitime pour tromper les victimes par *phishing*.

5.3. Recommandations en matière de sécurité

- Il est essentiel de veiller à ce que l'identifiant de session ne soit jamais exposé dans le trafic non crypté.
- Mettre en place des en-têtes sécurisés avec des directives telles que **cache-control** ou **strict-transport security**.
- Vérifier que les sessions sont invalidées lorsque l'utilisateur se déconnecte.
- Les sessions doivent expirer après un certain temps d'inactivité.
- Veiller à ce que toutes les pages nécessitant une authentification disposent d'un accès facile et convivial à la fonctionnalité de déconnexion.
- Vérifier que l'ID de session n'apparaît jamais dans les URL, les messages d'erreur ou les journaux.
- S'assurer que chaque authentification et réauthentification réussie génère une nouvelle session et un nouvel ID de session, détruisant l'ancien.
- Vérifier que l'ID de session stocké dans les *cookies* est défini à l'aide des attributs **HttpOnly** et **Secure**.
- Définir l'attribut **Path** des *cookies* de session de manière appropriée pour empêcher l'accès à d'autres domaines.
- Vérifier que l'application garde la trace de toutes les sessions actives et permet à l'utilisateur de mettre fin aux sessions de manière sélective ou globale à partir de son compte.
- Dans le cas d'applications à haute valeur ajoutée, veiller à ce que l'utilisateur soit tenu de fermer toutes les sessions actives si le mot de passe vient d'être modifié avec succès.

Es Il est essentiel de veiller à ce que l'identifiant de session ne soit jamais exposé dans le trafic non crypté

- Limiter l'accès aux URL, rôles, données d'application, attributs d'utilisateur et données de configuration d'accès protégés aux seuls utilisateurs autorisés.

- Enregistrer dans un journal de sécurité toutes les sessions créées, fermées manuellement ou expirées depuis le client ou le serveur, identifiées par le nom d'utilisateur et l'ID de session interne du serveur (non partagé avec le client), y compris la date et l'heure de l'événement.

5.4. Exemple

En Java, la gestion des sessions peut être effectuée en utilisant l'interface **javax.servlet.http.HttpSession**. Cette interface fournit des méthodes pour stocker et récupérer les attributs de la session, définir et obtenir le délai d'attente de la session, et invalider la session.

Pour utiliser l'interface **HttpSession**, il faut d'abord obtenir une instance :

```
HttpSession session = request.getSession();
```

Ensuite, les méthodes de l'interface peuvent être utilisées pour effectuer la gestion des sessions :

```
session.setAttribute(“user”, “John”);
```

Pour récupérer un attribut de session :

```
String user = (String) session.getAttribute("user");
```

Pour définir le délai d'expiration de la session (en secondes) :

```
session.setMaxInactiveInterval(3600);
```

Pour invalider la session :

```
session.invalidate();
```

Il est important de noter que pour garantir la sécurité de la gestion des sessions, des *cookies* HTTPS sécurisés doivent être utilisés et des identifiants de session uniques et imprévisibles doivent être générés. Il est nécessaire de veiller également à valider la demande et l'état de la session à chaque demande pour éviter l'usurpation de session.

```
import java.security.MessageDigest ;
import java.security.SecureRandom ;
import java.util.Arrays ;
import java.util.Base64 ;
...

SecureRandom random = new SecureRandom() ;
byte[] bytes = new byte[32] ;
random.nextBytes(bytes) ;
MessageDigest digest = MessageDigest.getInstance("SHA-256") ;
byte[] hashedSessionId = digest.digest(bytes) ;
String sessionId = Base64.getEncoder().encodeToString(hashedSessionId) ;

Cookie sessionCookie = nouveau Cookie("SESSIONID", sessionId) ;
sessionCookie.setHttpOnly(true) ;
sessionCookie.setSecure(true) ;
sessionCookie.setMaxAge(3600) ; // expire dans 1 heure
response.addCookie(sessionCookie) ;
```

5.5. Références

OWASP : Gestion des sessions sécurisées
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html [8]

Java : Cadre SpringSession
https://www.baeldung.com/spring-session [9]

6. Validation des données d'entrée et de sortie

C'est l'un des domaines critiques les plus importants de la sécurité des applications. La plupart des vulnérabilités des applications résultent d'une validation incorrecte ou insuffisante des données d'entrée ou de sortie qui sont exploitées pour des attaques telles que le *cross-site scripting*, les injections en général, l'exposition de données sensibles ou les attaques DoS.

Bien que la validation des données de sortie ne soit pas une activité très courante chez les développeurs, elle est tout aussi importante et pourrait être exploitée par des utilisateurs malveillants si des mesures de sécurité adéquates n'étaient pas incluses. Le résultat renvoyé par une application pourrait être utilisé pour réaliser les mêmes types d'attaques, mais de manière plus sophistiquée.

La mise en œuvre de validations de toutes les données d'entrée et de sortie ne garantit pas que l'application sera exempte de vulnérabilités, car ces validations peuvent être insuffisantes et ne pas avoir pris en compte d'autres types de problèmes fonctionnels ou de logique métier qui pourraient affecter le comportement normal de l'application. Un exemple serait la validation d'une entrée numérique où il est vérifié qu'il s'agit d'un nombre, qu'il est positif et qu'il n'est pas égal à 0. Cependant, si l'application l'utilise dans une boucle itérative, elle pourrait être appelée avec une valeur énorme et mettre l'application dans une boucle presque infinie, ce qui serait une attaque DoS.

La plupart des vulnérabilités des applications résultent d'une validation incorrecte ou insuffisante des données d'entrée ou de sortie qui sont exploitées pour des attaques telles que le *cross-site scripting*, les injections en général, l'exposition de données sensibles ou les attaques DoS

6.1. Techniques de validation

6.1.1. Assainissement

Il s'agit d'un processus de conversion des données qui ont plus d'une représentation possible dans un format standard, canonique ou normalisé, réduisant l'entrée/sortie à une seule forme fixe convertie et/ou réduite. Cette technique permet à elle seule d'éviter de nombreux problèmes de validation et de nombreux types d'attaques.

Il existe des bibliothèques pour chaque type de langage qui intègrent déjà des méthodes d'assainissement.

6.1.1.1. ASSAINISSEMENT DES CHEMINS D'ACCÈS : Tous les chemins de fichiers ou de répertoires sont normalisés. Par exemple, un chemin UNIX tel que **/home/user/myFile.txt**, peut être défini comme **/var/log/.../.../home/user/myFile.txt**. Cela pourrait être utilisé par un utilisateur malveillant pour parcourir le système de fichiers et obtenir des informations non autorisées. Pour éviter cela, le chemin est canonisé à une forme fixe ou bien les sous-chaînes **« ../ »** et **«./ »** sont directement supprimées du chemin.

6.1.1.2. ASSAINISSEMENT DES ESPACES : Tous les paramètres d'entrée suppriment les espaces, les caractères de tabulation ou autres caractères blancs (160) avant et après les données.
Certains choisissent également de supprimer tous les caractères blancs dans les données et de les convertir directement en un code espace (32).

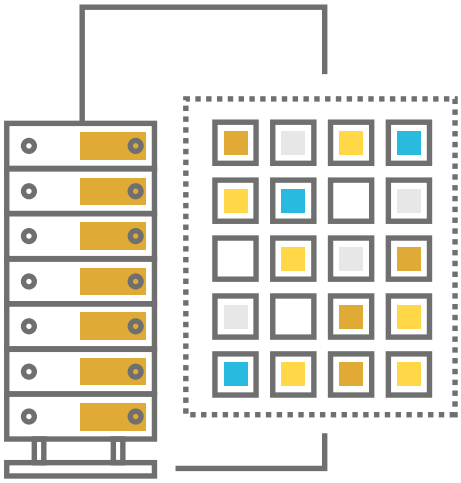
6.1.1.3. ASSAINISSEMENT DU JEU DE CARACTÈRES : Il est vérifié que tous les caractères saisis dans les données correspondent au jeu de caractères attendu.
Ceux qui ne le font pas sont supprimés ou transformés en espace blanc. Les caractères qui peuvent arriver codés comme **« 0xA0 »**, **« /xA0 »**, **« %A0 »**, etc. sont pris en compte pour être décodés avant d'être traités.

6.1.1.4. ASSAINISSEMENT DE LA CASSE : Tous les caractères sont convertis en majuscules ou en minuscules, en fonction du paramètre défini.

6.1.2. Type de données

Valide que les données reçues sont du type attendu avec le format approprié. Les types de données peuvent être simples ou complexes, selon les définitions mises en œuvre dans l'application. Les types de données simples seraient, par exemple : entier, décimal, booléen, booléen, chaîne de caractères, et les types de données complexes seraient : e-mail, numéro fiscal, adresse, nom, CP, date, heure, URL, etc.

6.1.3. Format



Il permet de valider des données aux formats plus ou moins complexes et variables mais qui peuvent parfaitement être définies au sein d'une expression régulière. Bon nombre des types de données complexes validés au point précédent sont susceptibles d'être réalisés à l'aide de cette technique.

Il faut toutefois veiller à ne pas créer une expression régulière si complexe qu'elle pourrait être vulnérable à une attaque ReDoS. Pour éviter que cela ne se produise, il est nécessaire de valider l'expression régulière à l'aide d'un outil qui garantit sa sécurité et son adéquation, tel que **RegEx 101** ou **RegEx Testing**.

6.1.4. Tailles minimales et maximales

Valide que la taille (longueur) des données dépasse un nombre minimum de caractères ou ne dépasse pas un nombre maximum de caractères. Cette vérification empêcherait l'envoi de données si volumineuses que l'application serait « bloquée » par le simple traitement de l'entrée.

6.1.5. Valeurs minimales et maximales

Contrairement au point précédent, cette validation s'applique aux valeurs numériques qui doivent se situer dans une fourchette de maximum et de minimum.

6.1.6. Liste blanche

Valide que les données se trouvent dans une liste de données préfixées. Cette validation est largement utilisée sur les données qui font partie des recensements.

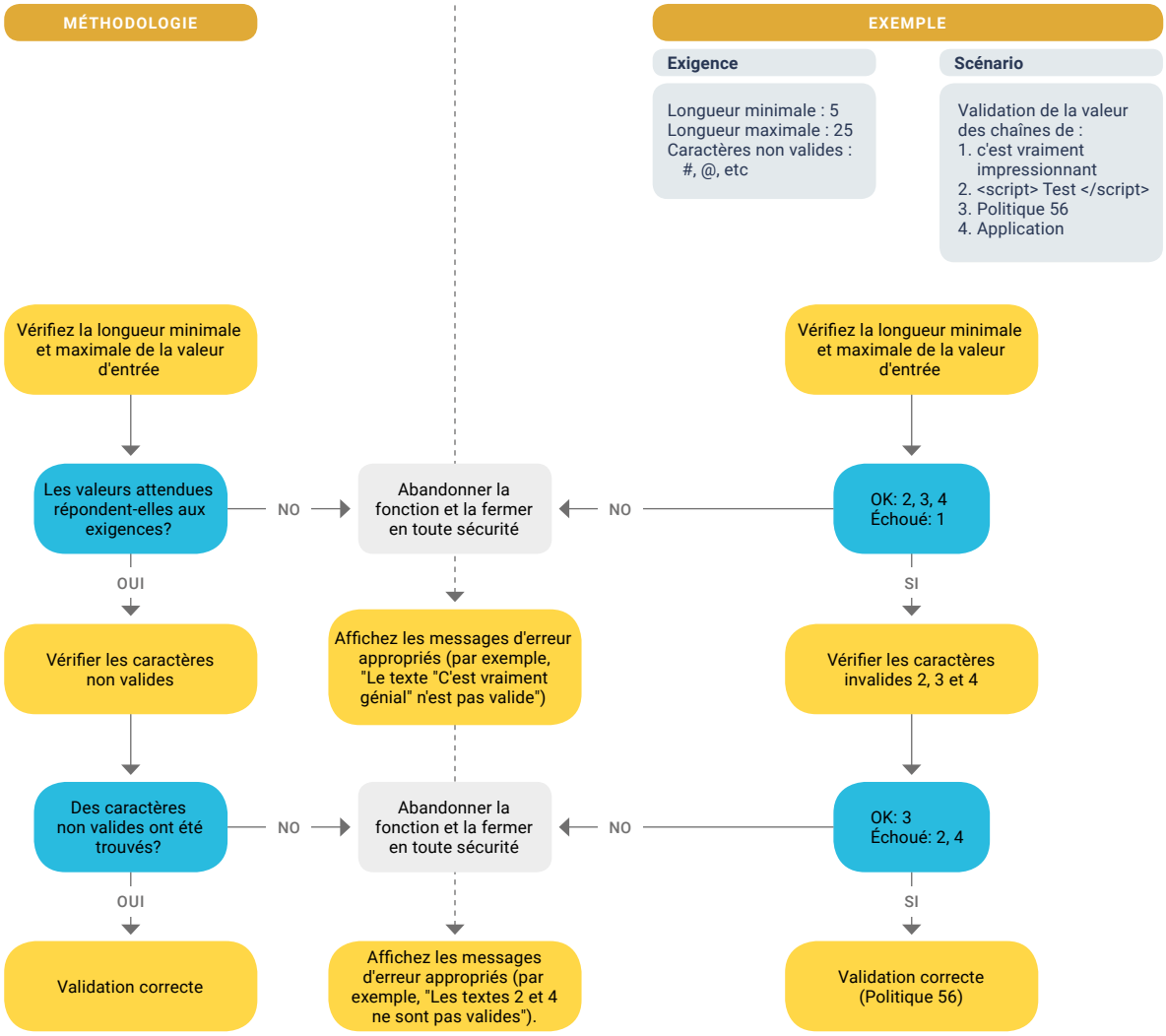
6.1.7. Liste noire

Valide que la donnée ne se trouve pas dans une liste de données préfixées. Ce type de validation est souvent insuffisant et dangereux car il peut difficilement couvrir toutes les possibilités de malveillance.

Toutefois, elle peut être utile lorsqu'il s'agit de décider de rejeter des textes contenant des mots offensants qui figurent sur une liste noire.

6.2. Organigramme

L'organigramme suivant est un exemple qui illustre le flux de travail pour valider les valeurs de texte libre. Il comprend quatre (4) valeurs d'entrée qui sont vérifiées par rapport à certaines exigences simples. Lorsqu'une valeur ne satisfait à aucune exigence, les données d'entrée doivent être rejetées.



6.3. Risques potentiels

La validation des données est à l'origine de nombreuses vulnérabilités qu'un attaquant peut exploiter. Voici quelques-unes des vulnérabilités liées à une validation insuffisante ou manquante des entrées :

- **Scripting inter site :** type d'attaque qui permet à un utilisateur malveillant d'injecter du code dans le navigateur web de la victime.
- **Injection SQL :** exploite les failles de la programmation applicative au niveau de la validation des entrées pour effectuer des opérations sur une base de données de manière illégitime.
- **Injection LDAP :** consiste à injecter des requêtes LDAP arbitraires pour accéder à des données interdites, voire obtenir des privilèges supplémentaires.
- **Injection de journaux :** consiste à injecter des commandes d'exécution dans le système ou tout autre type de problème en utilisant le système de journaux qui enregistre les données des informations des paramètres d'entrée.
- **Injection XEE :** une injection XPATH est l'injection d'un code XML arbitraire dans le but d'accéder à des données qui ne devraient pas être accessibles ou d'obtenir des informations sur l'arborescence XML.
- **Bombe XML :** cette attaque tente de surcharger le XML en dépassant les ressources mémoire d'une application pour provoquer un déni de service.
- **DoS, DDoS ou ReDoS :** attaques par déni de service dues à une erreur du système dans le traitement des entrées non validées, provoquant des défaillances du système.

La validation des données est à l'origine de nombreuses vulnérabilités qu'un attaquant peut exploiter

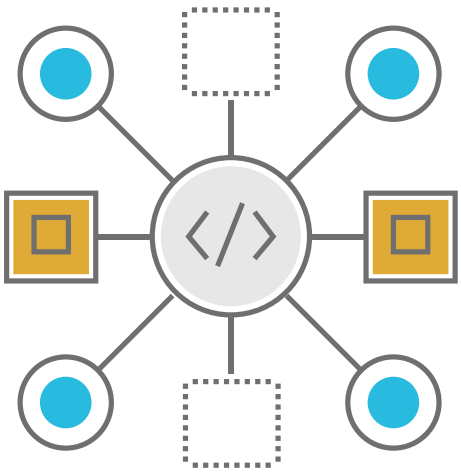
6.4. Recommandations en matière de sécurité

- Les validations doivent toujours être effectuées du côté du serveur. Les validations côté client peuvent également être utiles, mais ne sont que recommandées.
- À défaut, utiliser les mécanismes standard de validation des entrées fournis par des bibliothèques spécifiques à la technologie (Spring Validator, etc.)
- Couvrir entièrement la **validation des données au moyen de schémas de validation** ou de mécanismes standard qui garantissent l'entrée des données par : l'assainissement, le type de données, le format, les longueurs, les valeurs, les listes blanches, les listes noires, etc.
- Vérifier que les données structurées sont fortement typées et validées selon un schéma défini, y compris les caractères autorisés, la longueur et le modèle, par exemple les numéros de carte de crédit ou de téléphone, ou valider que deux champs connexes sont raisonnables, comme la validation des banlieues et des codes postaux.
- Vérifier que les données non structurées sont nettoyées pour imposer des mesures de sécurité génériques, telles que les caractères et la longueur autorisés, et éviter les caractères potentiellement dangereux.
- S'assurer que toutes les entrées non fiables sont correctement nettoyées à l'aide d'une bibliothèque d'assainissement.
- Éviter d'afficher des informations sensibles à la suite d'une erreur de validation d'un paramètre reçu.
- Accepter exclusivement les données attendues à chaque point d'entrée de l'application qui proviennent de l'utilisateur, du processus final de tous les champs de saisie, formulaires, URL, *cookies* d'application, etc. Toute donnée inattendue doit être rejetée.

Les validations doivent toujours être effectuées du côté du serveur

- Vérifier que les erreurs de validation des entrées côté serveur entraînent le rejet de la demande.
- Veiller à ce que toutes les requêtes de la **base de données** soient **protégées par des requêtes paramétrées** afin d'éviter toute injection SQL.
- Vérifier que l'application n'est pas susceptible de subir une injection de commande.
- Vérifier que toutes les variables de chaîne situées dans le code HTML ou dans tout autre code client Web sont correctement codées à la main dans leur contexte ou utilisez des modèles qui codent automatiquement le contexte afin de garantir que l'application n'est pas susceptible de subir des attaques de type cross-site scripting (XSS) reflétées, mises en cache ou DOM.
- Contrôler que l'application ne contient pas de vulnérabilités d'affectation de paramètres en masse (alias liaison automatique de variables). S'assurer que toutes les données d'entrée sont validées, pas seulement les champs de formulaire HTML, mais toutes les sources d'entrée telles que les requêtes REST, les paramètres de requête, les en-têtes HTTP, les *cookies*, les fichiers par lots, les flux RSS, etc., en utilisant des listes blanches, des formes de validation moins poussées telles que les listes grises (qui suppriment les mauvaises chaînes connues) ou les listes noires (qui rejettent les mauvaises entrées).
- Vérifier que l'application restreint les analyseurs XML pour qu'ils n'utilisent que les paramètres les plus restrictifs possibles et s'assurer que les fonctions dangereuses, telles que la résolution d'entités externes, sont désactivées.
- Vérifier que la désérialisation des données non fiables est empêchée ou largement protégée lorsque la désérialisation ne peut être évitée.

Vérifier que les erreurs de validation des entrées côté serveur entraînent le rejet de la demande



6.5. Exemple

Dans Java Spring, les annotations sont utilisées pour valider les paramètres d'entrée des méthodes et les valeurs autorisées dans les membres d'une classe :

```
@PostMapping("/users")
public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
  ...
  return ResponseEntity.ok(user);
}

public class User {
  @NotBlank
  privé String name ;

  @Email
  privé String email ;

  ...
}
```

L'annotation **@Valid** est utilisée en conjonction avec un objet de validation pour valider les paramètres d'entrée d'une méthode de contrôleur.

Et les annotations **@NotBlank** et **@Email** pour indiquer que le champ du nom ne doit pas être vide et que le champ email doit être dans un format d'adresse électronique valide.

6.6. Références

OWASP : validation des paramètres d'entrée
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html [10]

OWASP : validation des données - Librairie de l'ESAP
https://owasp.org/www-project-enterprise-security-api/ [11]

Validation de formulaires Web en Javascript
https://www.tutorialspoint.com/javascript_form_validation_web_application/index.asp [12]

Java : validation Spring Boot
https://www.baeldung.com/spring-boot-bean-validation [13]

Prévention XSS
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html [14]

Python : validation Colander https://pypi.org/project/colander/ [15]

Python : validation Cerberus https://docs.python-cerberus.org/en/stable/ [16]

PythonPython : validation Schematics
https://schematics.readthedocs.io/en/latest/ [17]

Python : validation Schema https://pypi.org/project/schema/ [18]

Python : validation jsonschema https://pypi.org/project/jsonschema/ [19]

7. Gestion des erreurs

La gestion des erreurs consiste à éviter d'afficher aux utilisateurs des informations pertinentes ou sensibles qui pourraient être utilisées pour lancer d'autres types d'attaques sophistiquées contre l'application, et à gérer les erreurs non contrôlées au sein de l'application afin de fournir des sorties sécurisées qui ne permettent pas l'exploitation de situations inattendues par des utilisateurs malveillants.

Si les applications n'ont pas mis en œuvre un traitement correct des erreurs, elles peuvent divulguer par inadvertance des informations sur leur configuration, leur état interne, des messages de débogage, des données sensibles, voire porter atteinte à la vie privée

7.1. Confidentialité des messages

Si les applications n'ont pas mis en œuvre un traitement correct des erreurs, elles peuvent divulguer par inadvertance des informations sur leur configuration, leur état interne, des messages de débogage, des données sensibles, voire porter atteinte à la vie privée.

D'autres façons de divulguer des informations consistent, par exemple, à indiquer le temps nécessaire au traitement de certaines transactions, ou à proposer différents codes pour différentes entrées.

Toutes ces informations peuvent être exploitées pour lancer, voire automatiser, des attaques très puissantes, ce qui rend essentiel un bon traitement des erreurs.

RISQUES POTENTIELS

Voici les risques potentiels les plus courants si le traitement des erreurs n'est pas correctement mis en œuvre dans l'application.

- **Fuite d'informations sensibles :** version du serveur, moteur de la base de données, documents sensibles, structure des fichiers, etc.
- **Déni de service :** lorsque des erreurs forcées par des abus peuvent provoquer l'arrêt du système.
- **Cross-site scripting :** lorsque les messages d'erreur présentent des paramètres d'entrée qui n'ont pas été correctement échappés.

7.2. Erreurs non contrôlées

Lorsque des situations où des exceptions ou des erreurs en certains points de l'application pourraient se produire n'ont pas été envisagées, et qu'elles se produisent, l'application provoquerait une sortie inattendue de la logique d'entreprise qui pourrait la laisser dans un état vulnérable aux activités ultérieures de l'utilisateur.

Il est donc impératif que toutes les transactions soient entièrement analysées en fonction des diverses exceptions qui peuvent se produire et que ces sorties exceptionnelles soient traitées de manière appropriée.

Une autre façon d'exploiter les erreurs non vérifiées est lorsque les ressources ne sont pas correctement fermées, ce qui peut éventuellement provoquer un déni de service en raison de la consommation excessive des ressources qui ne sont pas fermées à la suite du forçage des erreurs dans l'application. Pour éviter cette vulnérabilité, il est essentiel de s'assurer que toutes les ressources sont fermées lorsqu'elles ne sont plus utilisées, quelles que soient les erreurs qui peuvent apparaître pendant les opérations. Il est recommandé d'utiliser les instructions **try / finally** pour assurer la fermeture des ressources.

RISQUES POTENTIELS

Voici les risques potentiels les plus courants si le traitement des erreurs n'est pas correctement mis en œuvre dans l'application.

- **Déni de service :** lorsque des erreurs forcées par des abus peuvent provoquer l'arrêt du système.
- **Contournement de la logique métier :** lorsque des sorties inattendues de la logique métier se produisent en forçant des exceptions ou des erreurs incontrôlées.

7.3. Recommandations en matière de sécurité

- Utiliser des messages d'erreur génériques qui ne donnent pas d'indices aux utilisateurs finaux sur les aspects sensibles de l'application.
- Utiliser un traitement centralisé des exceptions.
- L'application doit gérer les erreurs sans s'appuyer sur les messages d'erreur du serveur affichés aux utilisateurs.
- Toute logique de contrôle d'accès qui conduit à une erreur doit refuser l'accès par défaut.
- Analyser en détail toutes les exceptions qui peuvent se produire en raison de l'utilisation de bibliothèques système ou de bibliothèques tierces dans l'application, les traiter de manière appropriée et fournir une sortie sûre à l'application.
- Utiliser **try/catch/finally** pour garantir l'arrêt de toutes les ressources en cas d'erreur.
- Consigner toutes les exceptions inattendues dans un journal spécifique indiquant, entre autres, la date et l'heure de la défaillance, l'utilisateur qui l'a provoquée et la méthode où la défaillance s'est produite, ainsi que des informations sur l'exception. Ce journal doit se trouver dans un environnement sécurisé accessible uniquement aux utilisateurs autorisés.

7.4. Exemple

L'exemple suivant est le bloc de code typique que l'on devrait trouver dans toute méthode où l'on veut contrôler les erreurs qui peuvent apparaître dans un bloc de code.

Toute exception non gérée dans le bloc try sera récupérée par **catch**, d'où une exception gérée sera lancée. Et, dans les deux cas, le bloc finally sera exécuté pour fermer les ressources ouvertes avant le **try** ou après le **try**.

```
try {
   ...
   // code qui peut lancer une exception
   ...
} catch (Exception e) {
   // gestion des exceptions
   log.error("ERROR", e);
   lancer une nouvelle CustomException("ERROR", e);
} finally {
   // fermeture ou libération de ressources ouvertes
   ...
}
```

7.5. Références

Java : Try - Catch - Finally
https://www.w3schools.com/java/java_try_catch.asp [20]

Java : Traitement des exceptions
https://www.baeldung.com/java-exceptions [21]

Stratégies de traitement des erreurs
https://dzone.com/articles/error-handling-strategies [22]

Python : Erreurs et exceptions
https://docs.python.org/es/3/tutorial/errors.html#errors-and-exceptions [23]

8. Journal sécurisé

Il s'agit d'enregistrer l'activité des applications et les événements du système liés à la sécurité, en termes d'authentification, d'autorisation, d'intégrité ou de confidentialité, tels que les tentatives de connexion échouées, l'autorisation acquise par un utilisateur, les accès à des données sensibles, etc., ainsi que les éventuelles menaces détectées qui peuvent être contrôlées depuis l'application : injections, énumération d'utilisateurs, attaques par force brute, etc. Ce dernier point est particulièrement important lorsque vous souhaitez effectuer une analyse approfondie sur des incidents de sécurité qui se sont produits.

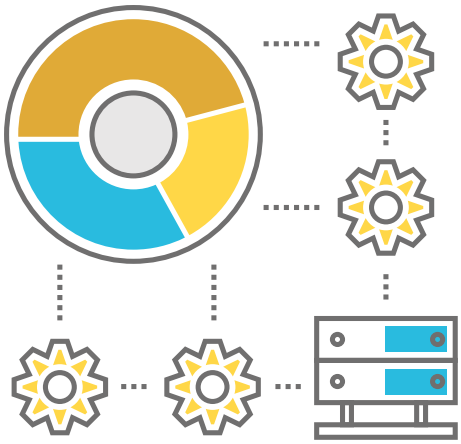
Les journaux de sécurité doivent être spécialement protégés au niveau de l'authentification, de l'intégrité, de la confidentialité, de la disponibilité et de la traçabilité :

- **Authentification/Autorisation :** seules les personnes identifiées et autorisées peuvent avoir accès au journal.
- **Intégrité :** le journal conserve une signature d'intégrité qui garantit qu'il n'a pas été altéré au niveau du journal ou de l'entrée. Cette signature est mise à jour à chaque nouvelle entrée dans le journal.
- **Confidentialité :** les données sensibles du journal sont mises sous forme de jetons, anonymisées ou cryptées pour empêcher l'accès au journal par des méthodes autres que celles mises en œuvre pour l'accès autorisé.
- **Disponibilité :** les documents sont stockés avec une redondance et des copies de sauvegarde.
- **Traçabilité/auditabilité :** ils doivent être stockés en toute sécurité pendant une durée de conservation à des fins d'audit.

Il s'agit d'enregistrer l'activité des applications et les événements du système liés à la sécurité, en termes d'authentification, d'autorisation, d'intégrité ou de confidentialité

8.1. Risques potentiels

- **Fuite d'informations :** les journaux peuvent contenir des informations sensibles sur l'application ou le système et peuvent ne pas avoir été protégés de manière adéquate contre les failles dans l'autorisation d'accès.
- **Falsification des journaux :** un utilisateur non autorisé pourrait modifier les fichiers journaux, ce qui peut entraîner une perte de traçabilité dans l'application ou même permettre à l'utilisateur malveillant d'exécuter du code sur le système.
- **Suppression des journaux :** un utilisateur malveillant pourrait supprimer les journaux pour éviter de laisser des traces de ses méfaits.



8.2. Recommandations en matière de sécurité

- Ne pas enregistrer d'informations sensibles telles que des mots de passe, des informations financières, des cartes de crédit, des données personnelles, etc. Dans ces cas, utiliser des jetons, l'anonymisation des informations ou le cryptage.
- Valider les paramètres des composants variables qui composeront l'entrée du journal pour éviter les injections et les comportements inattendus.
- Prendre des notes suffisamment précises pour la sécurité :
  - Tentatives d'authentification, en particulier les échecs.
  - Accès accordés avec les rôles associés à l'utilisateur.
  - Accès aux données sensibles et actions entreprises sur celles-ci, quel rôle a été utilisé et par quel utilisateur.
  - Erreurs de validation des entrées
  - Exceptions au système.

- Menaces ou tentatives de menaces détectées pouvant être contrôlées par l'application : injections, attaques par force brute, énumération d'utilisateurs, défaillances de la logique métier, tentatives d'opérations non privilégiées, tentatives de traversée de répertoire, etc.

- Utilisez les fonctions de hachage pour garantir l'intégrité des données des journaux.

- Le journal de sécurité doit être indépendant des autres journaux et disposer de ses propres protections de sécurité dans le système.

8.3. Exemple

La journalisation sécurisée en Java peut être réalisée à l'aide de la bibliothèque java.util.logging.

```
import java.util.logging.Logger ;

public class MyClass {
  private static final Logger log = Logger.getLogger(MyClass.class.getName()) ;

  public void myMethod() {
    // certaines opérations pouvant entraîner une exception
   essayez {
      // code qui peut lancer une exception
    } catch (Exception e) {
      log.severe(log.severe("Une erreur grave s'est produite : " + e.getMessage()) ;
    }
  }
}
```

8.4. Références

Journal de sécurité : Bonnes pratiques en matière de journalisation et de gestion
https://www.dnsstuff.com/security-log-best-practices [24]

java : journalisation
https://docs.oracle.com/javase/7/docs/technotes/guides/logging/index.html [25]

9. Cryptographie

La cryptographie est une technique utilisée pour protéger les informations et les communications par l'utilisation de codes secrets ou de clés, et constitue un outil essentiel pour la protection des informations et des communications.

Elle est utilisée pour garantir la confidentialité, l'intégrité et l'authenticité des informations et des communications : la confidentialité en protégeant les informations afin que seules les personnes autorisées puissent y accéder, l'intégrité en protégeant les informations contre toute modification non autorisée et l'authentification en vérifiant l'identité des personnes ou des systèmes accédant aux informations.

La cryptographie peut être utilisée de différentes manières : comme cryptage de données sensibles pour protéger les informations ou comme signature numérique pour garantir l'intégrité des données.

La cryptographie est une technique utilisée pour protéger les informations et les communications par l'utilisation de codes secrets ou de clés

9.1. Utilisation du cryptage

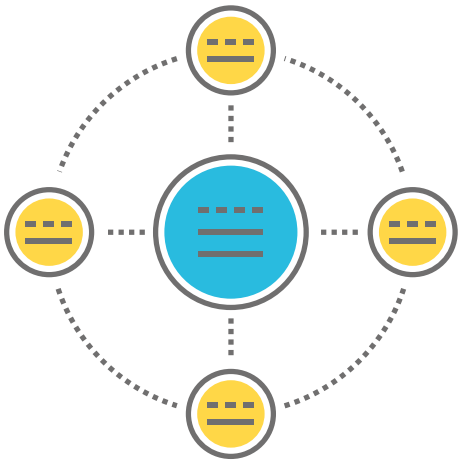
9.1.1. Fonctions de hachage

Un hachage cryptographique est une fonction mathématique qui, à partir d'un ensemble de données, produit une quantité fixe de données appelée « condensé » ou « hachage ». Selon la fonction utilisée, le nombre de données obtenues peut varier, mais il générera toujours le même nombre de données pour la même fonction, quel que soit le nombre de données utilisées en entrée.

La puissance du hachage cryptographique réside dans le fait que la probabilité de générer les mêmes données de sortie, dans le même ordre, à partir d'une entrée différente est très faible, très improbable. Si cela se produisait, une « collision » se produirait, et cet effet, découvert dans l'une de ces fonctions mathématiques, pourrait être exploité par des utilisateurs malveillants pour certaines attaques de sécurité.

Une fonction de hachage est couramment utilisée pour vérifier l'intégrité des données, car toute modification de l'entrée originale se traduira par un changement significatif dans le hachage résultant. Lorsque la fonction de hachage est alimentée avec les données que nous voulons sécuriser, les données de sortie de la fonction nous fournissent le hachage de vérification. Ce hachage ne pouvait être récupéré qu'avec la même fonction, avec les mêmes données d'entrée et dans le même ordre.

Les fonctions de hachage sont utilisées pour le stockage des mots de passe ou pour vérifier l'intégrité des données afin de s'assurer qu'elles n'ont pas été modifiées.



9.1.2. Cryptage symétrique

Le cryptage symétrique est un type de cryptage dans lequel la même clé est utilisée pour le cryptage et le décryptage des informations. Il est utilisé pour protéger la confidentialité des informations lors de leur transmission ou de leur stockage sur un appareil.

Le principal avantage du cryptage symétrique est qu'il est rapide et facile à mettre en œuvre. Son principal inconvénient est que les deux parties doivent partager la clé secrète afin de communiquer en toute sécurité. Cela peut poser un problème dans les environnements distribués, car il faut un mécanisme sûr pour partager la clé de manière fiable.

9.1.3. Cryptage asymétrique

Le cryptage asymétrique est un type de cryptage dans lequel deux (2) clés différentes, appelées clé publique et clé privée, sont utilisées pour crypter et décrypter des informations. La clé publique est utilisée pour chiffrer les informations et peut être partagée sans problème, tandis que la clé privée est utilisée pour déchiffrer les informations et doit rester secrète.

L'avantage du cryptage asymétrique est que la clé privée ne doit pas être partagée pour établir une communication sécurisée. La clé publique peut être partagée de manière transparente et est utilisée pour crypter les informations, tandis que la clé privée est utilisée pour les décrypter. Toutefois, le chiffrement asymétrique est plus lent que le chiffrement symétrique et peut être plus difficile à mettre en œuvre.

9.2. Risques potentiels

Les informations qui ne sont pas cryptées ou qui ne sont pas correctement cryptées sont exposées aux risques suivants :

- **Exposition d'informations sensibles :** les données sensibles seraient en clair pour toute personne non autorisée.
- **Vol de justificatifs et usurpation d'identité :** les informations sensibles du mot de passe volé pourraient être utilisées pour se faire passer pour un autre utilisateur et pour mener d'autres attaques telles que le spoofing.
- **Fuite de données personnelles :** il s'agit de données protégées par la réglementation du pays dont la violation de la confidentialité pourrait entraîner des sanctions financières.
- **Perte de la réputation de l'entreprise :** conséquence de tout ce qui précède.
- **Attaques de l'intercepteur (MitM) :** si les communications ne sont pas bien sécurisées, elles peuvent être interceptées et l'ensemble du flux de données décrypté pour obtenir des informations précieuses pouvant être utilisées pour d'autres attaques.

9.3. Recommandations en matière de sécurité

- Toutes les informations sensibles d'une organisation telles que les mots de passe, les données personnelles, les référentiels de journaux ou toute autre information étiquetée comme confidentielle ou supérieure à l'entreprise doivent être stockées sous une forme illisible (cryptée) afin de garantir la confidentialité de l'entreprise.
- Vérifier que tous les nombres aléatoires, noms de fichiers aléatoires, GUID aléatoires et chaînes aléatoires sont générés par un générateur de nombres aléatoires approuvé par le module cryptographique.
- Vérifier qu'il existe une politique explicite sur le traitement des clés cryptographiques (telles que la génération, la distribution, la révocation et l'obsolescence).
- Vérifier que le cycle de vie des clés cryptographiques est correctement mis en œuvre.
- S'assurer que les mots de passe sensibles ou les informations critiques stockés en mémoire soient remplacés par des zéros dès qu'ils ne sont plus utilisés afin d'atténuer les attaques par vidage de mémoire.
- Vérifier que les nombres aléatoires sont créés avec un niveau d'entropie approprié, même lorsque l'application est soumise à une charge élevée.
- Vérifier que des algorithmes cryptographiques obsolètes ou faibles, tels que l'algorithme DES à clé symétrique ou des fonctions de hachage telles que MD5 ou SHA-1, ne sont pas utilisés en raison de l'impossibilité de garantir la confidentialité.
- Utiliser les bibliothèques cryptographiques existantes et, dans tous les cas, utiliser des algorithmes ou des implémentations cryptographiques personnalisés ou créés par l'utilisateur.

Toutes les informations sensibles d'une organisation telles que les mots de passe, les données personnelles, les référentiels de journaux ou toute autre information étiquetée comme confidentielle ou supérieure à l'entreprise doivent être stockées sous une forme illisible (cryptée) afin de garantir la confidentialité de l'entreprise

9.4. Exemple

Cet exemple utilise la cryptographie symétrique pour chiffrer un message à l'aide d'une clé secrète et d'un vecteur d'initialisation (IV) afin de se protéger contre les attaques par « réutilisation de clé ».

```
import java.security.SecureRandom ;
import javax.crypto.Cipher ;
import javax.crypto.KeyGenerator ;
Importation de javax.crypto.SecretKey ;
import javax.crypto.spec.IvParameterSpec ;

public class AdvancedCryptographyExample {
   public static void main(String[] args) throws Exception {
      // Nous générons une clé secrète pour la cryptographie symétrique.
      KeyGenerator keyGenerator = KeyGenerator.getInstance("AES") ;
      keyGenerator.init(256) ; // nous utilisons une clé de 256 bits
      SecretKey secretKey = keyGenerator.generateKey() ;

      // Nous générons un vecteur d'initialisation (IV) pour la cryptographie symétrique.
      byte[] iv = new byte[16] ; // Les IV ont généralement une taille de 16 octets.
      SecureRandom secureRandom = nouveau SecureRandom() ;
      secureRandom.nextBytes(iv) ;
      IvParameterSpec ivParameterSpec = new IvParameterSpec(iv) ;

      // Nous cryptons un message en utilisant la clé secrète et le IV.
      String message = "Ceci est un message secret" ;
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding") ;
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec) ;
      byte[] message crypté = cipher.doFinal(message.getBytes()) ;

      // Nous stockons la clé secrète de manière sécurisée
         (par exemple, en utilisant une clé de cryptage).
      saveSecretKey(secretKey) ;

      // Nous enregistrons le IV et le message crypté d'une manière ou d'une autre
         (par exemple dans une base de données).
      saveIvAndEncryptedMessage(iv, encryptedMessage) ;
   }

   private static void saveSecretKey(SecretKey secretKey secretKey) {
      // Nous stockons la clé secrète d'une manière ou d'une autre
         (par exemple, en utilisant une clé de cryptage).
   }

   private static void saveIvAndEncryptedMessage(byte[] iv, byte[] encryptedMessage) {
      // Nous enregistrons le IV et le message crypté d'une manière ou d'une autre
         exemple dans une base de données).
   }
}
```

9.5. Références

OWASP : Cryptographie sécurisée en Java
https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions [26]

Fonction de dérivation des clés Scrypt
https://www.tarsnap.com/scrypt.html [27]

Python : bcrypt fonctions cryptographiques
https://pypi.org/project/bcrypt/ [28]

10. Gestion sécurisée des fichiers

Il s'agit d'un processus qui implique la protection des données stockées dans des fichiers afin de garantir leur intégrité, leur confidentialité et leur disponibilité, y compris des mesures telles que la cryptographie, l'authentification, l'autorisation, le contrôle d'accès et la sauvegarde.

Ce processus doit être envisagé pendant la phase de conception d'une application, puis mis en œuvre pendant le développement. La plupart des applications reposent sur des fichiers internes pour fonctionner et, en outre, si le téléchargement de fichiers par les utilisateurs est autorisé, des contrôles de sécurité appropriés doivent être mis en place.

10.1. Risques potentiels

Sans une bonne gestion des fichiers, les risques suivants peuvent exister :

- **L'accès non autorisé** à des fichiers, ce qui aurait pour conséquence :
  - Divulgation de données.
  - Perte d'informations sensibles.
  - Manipulation de données.
  - Suppression de données.
- **Chargement de fichiers malveillants,** ce qui aurait pour conséquence :
  - Exécution de fichiers à distance.
  - Attaque par déni de service.
  - Infection par un logiciel malveillant.
- **Manque de sauvegardes**
  - Indisponibilité du service (DoS).
  - Perte de données de l'utilisateur.
  - Perte de données ayant une valeur pour l'entreprise.

10.2. Recommandations en matière de sécurité

- Authentifier et autorisez l'utilisateur avant de charger ou de télécharger des fichiers, en particulier si les données sont sensibles.
- Ne pas utiliser les entrées fournies par l'utilisateur pour nommer les fichiers ou les répertoires.
- Valider les types de contenu non seulement par extension, mais aussi vérifier les types MIME pour vérifier les fichiers.
- Ne pas autoriser le téléchargement de fichiers exécutables vers l'application.
- Limiter la taille des fichiers au minimum que le serveur peut traiter sans causer de problèmes de disponibilité et sans affecter la fonctionnalité des applications.
- Avant de traiter les fichiers vers le serveur, un scanner antivirus vérifie l'absence de logiciels malveillants ou de virus dans les fichiers.
- Désactiver les privilèges d'exécution sur les répertoires où les utilisateurs peuvent télécharger des fichiers.
- Ne pas utiliser de chemins absolus lorsque vous fournissez un lien de téléchargement à l'utilisateur.
- Ne pas stocker les fichiers avec leurs noms de manière séquentielle.
- N'utilisez pas d'informations sensibles pour nommer les fichiers.
- S'assurer que le contrôle d'accès est défini comme étant en lecture seule.
- Limiter le nombre de fichiers téléchargés par l'utilisateur.
- Stocker les hachages des fichiers téléchargés pour garantir leur intégrité.

10.3. Exemple

L'exemple suivant valide la taille du fichier et vérifie que le type MIME correspond à ce qui est attendu avant de traiter le fichier.

```
import java.io.File ;
import java.io.IOException ;
import java.nio.file.Files ;
import java.nio.file.Path ;
import java.nio.file.Paths ;
import java.util.List ;

public class FileManager {
   private static final long MAX_FILE_SIZE = 10485760 ; // 10MB
   private static final List<String> ALLOWED_MIME_TYPES = List.of
      ("text/plain", "image/jpeg", "image/png") ;

   public static void processFile(String filePath) throws IOException {
      // Vérifier que le fichier existe et qu'il est lisible.
      File file = new File(filePath);
      if (!file.exists() || !file.canRead()) {
         lancer une nouvelle IOException("Le fichier n'existe pas ou ne peut être lu")
      }
      // Vérifier la taille du fichier
      long fileSize = file.length();
      if (fileSize > MAX_FILE_SIZE) {
         lancer une nouvelle IOException("Le fichier est trop grand") ;
      }
      // Vérifier le type MIME du fichier
      Path path = Paths.get(filePath);
      String mimeType = Files.probeContentType(path);
      if (!ALLOWED_MIME_TYPES.contains(mimeType)) {
         lancer une nouvelle IOException("Le type de fichier n'est pas autorisé") ;
      }
                  // Traiter le fichier
      // ...
   }
}
```

Une autre solution consiste à configurer le serveur Apache pour limiter ce type de requête. Cette méthode est optimale en termes de performances et plus sûre car la demande reste sur le serveur web et n'entre même pas dans l'application.

```
LimitRequestBody 10485760
SetEnvIf Request_URI "^.*$" ALLOWED_MIME_TYPE=1
SetEnvIf Request_URI "^.*\.(txt|jpe?g|png)$" ALLOWED_MIME_TYPE=1
```

10.4. Références

Serveur Apache : Directives de configuration
https://httpd.apache.org/docs/2.4/mod/core.html#limitrequestbody [29]

Java : Canonisation du chemin
https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/File.html#getCanonicalPath() [30]

PHP : Canonisation des chemins
https://www.php.net/manual/es/function.realpath.php [31]

Python : mappage des fichiers avec leurs MimeTypes
https://docs.python.org/3/library/mimetypes.html [32]

11. Sécurité des transactions

Il s'agit d'un ensemble de mesures destinées à protéger les transactions financières et de paiement contre d'éventuelles fraudes ou attaques. Ces mesures comprennent :

- **Authentification des utilisateurs :** seules les personnes autorisées ont accès aux transactions financières et aux comptes. L'authentification des utilisateurs comprend généralement des mots de passe forts et, idéalement, une authentification à deux facteurs.
- **Cryptographie :** permet de protéger les informations confidentielles lors des transactions.
- **Validation des transactions :** pour s'assurer qu'elles sont légitimes et ne font pas partie d'une fraude ou d'une cyberattaque.
- **Surveillance des transactions :** pour aider à détecter et à prévenir les fraudes ou les attaques potentielles en temps réel.
- **Sauvegarde :** un plan de récupération des données en cas d'attaque ou de perte accidentelle des données garantit la disponibilité des données.

Toute interaction avec une structure de données complexe composée de plusieurs processus appliqués de manière séquentielle doit être effectuée en une seule fois et de manière sécurisée.

Elle doit répondre aux propriétés suivantes :

Atomicité	Cohérence	Isolement	Durabilité
Les transactions doivent toujours avoir un point de départ et un point d'arrivée. Elles doivent être identifiées de manière unique et toutes les opérations effectuées par la transaction doivent s'exécuter avec succès ou, en cas d'erreur, être ramenées à l'état initial (rollback).	Les données de transaction doivent être validées et cohérentes en termes d'intégrité. En cas de changement d'état, ceci doit également être valide et attendu.	Les opérations de transaction peuvent être effectuées indépendamment des opérations d'autres transactions sans affecter les données des autres ou leurs états particuliers.	Les transactions ont une durée de vie maximale et, une fois terminées, leurs informations sont persistantes.

11.1. Risques potentiels

Il existe plusieurs risques potentiels liés aux transactions :

- **Exploitation de la vulnérabilité *Bypass* de paiement :** en raison d'une configuration inadéquate du système de paiement. Il permet à un attaquant de manipuler les paramètres échangés entre le client et le serveur en traitant la réponse avant qu'elle ne soit envoyée à la passerelle de paiement et en contournant le système de paiement en général.
- **Fraude :** utilisation de fausses informations ou manipulation de transactions pour obtenir des avantages illicites.
- **Vol d'informations confidentielles :** utilisation à des fins de préjudice commercial ou pour de futures attaques.
- **Perturbation du traitement des transactions :** par des attaques DoS/DDoS.
- **Perte de données :** en raison de défaillances du système, d'attaques ou de catastrophes naturelles ayant de graves conséquences sur les transactions, qui peuvent affecter la confidentialité et l'intégrité des informations.

11.2. Recommandations en matière de sécurité

- **Pour éviter la vulnérabilité du contournement de paiement :**
  - L'autorisation et la confirmation d'un achat doivent être effectuées du côté du serveur.
  - Il est nécessaire de valider que les signatures utilisées sont correctes lors du processus de communication avec la passerelle de paiement.
  - Valider que le prix est correctement défini du côté du serveur.
  - Valider que les paiements ne sont pas réutilisés.
  - Le serveur de paiement doit vérifier à tout moment à quel stade de la transaction vous vous trouvez.

- **Prévoir un délai d'expiration de l'autorisation relativement court pour chaque transaction.**
- **Assurer la traçabilité des transactions.**
- **Chiffrer les communications avec des algorithmes asymétriques robustes.**
- **Enregistrement détaillé de toutes les transactions avec anonymisation des informations sensibles.**

11.3. Exemple

Cet exemple pourrait servir de guide pour éviter la vulnérabilité au *Bypass* de paiement :

```
import java.math.BigDecimal;

public class PaymentProcessor {

   private static final BigDecimal MIN_PAYMENT_AMOUNT = new BigDecimal("0.01");

   public void processPayment(String tId, BigDecimal amount, String paymentMethod) {
      // Vérifier si l'ID de la transaction est valide
      si (tId == null || !isValidTransactionId(tId)) {
         lancer une nouvelle exception IllegalArgumentException("ID de transaction non valide") ;
      }
      // Vérifier si le montant et le mode de paiement sont valides.
      si (montant == null || montant.compareTo(MIN_PAYMENT_AMOUNT) < 0) {
         lancer une nouvelle IllegalArgumentException("Montant du paiement non valide") ;
      }
      if (paymentMethod == null || !isValidPaymentMethod(paymentMethod)) {
         lancer une nouvelle IllegalArgumentException("Méthode de paiement non valide") ;
      }
      // Traiter le paiement
   }
   private boolean isValidTransactionId(String transactionId) {
      // Vérifier si l'ID de la transaction figure dans la liste des ID autorisés.
   }
   private boolean isValidPaymentMethod(String paymentMethod) {
      // Vérifiez si le mode de paiement figure dans la liste des modes de paiement autorisés.
   }
}
```

11.4. Références

WordPress : Plugin pour éviter la vulnérabilité du *Bypass* de paiement
https://www.acunetix.com/vulnerabilities/web/wordpress-plugin-nab-transact-security-bypass-2-1-0/

12. Sécurité des communications

La sécurité des communications entre applications est essentielle pour protéger la confidentialité, l'intégrité et la disponibilité des informations transmises par les applications. Cela est particulièrement important dans le contexte des applications mobiles, où les informations peuvent être transmises sur des réseaux non sécurisés ou publics.

La sécurité des communications entre applications est essentielle pour protéger la confidentialité, l'intégrité et la disponibilité des informations transmises par les applications

12.1. Risques potentiels

Une application sans mesures de sécurité en matière de communication est exposée à un certain nombre de menaces potentielles :

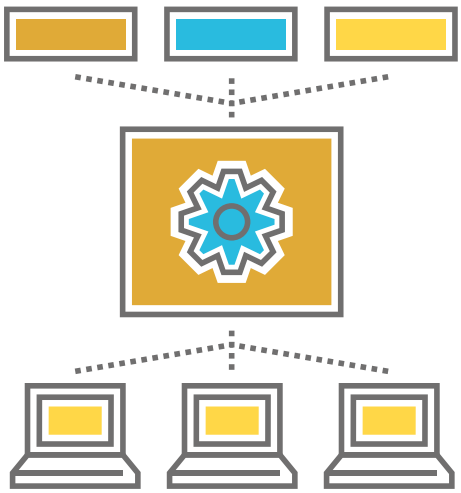
- **Le vol d'informations confidentielles.**
- **Interruption de service.**
- **Vol d'identité.**
- **Modification ou destruction des données.**
- **Propagation de logiciels malveillants.**

12.2. Recommandations en matière de sécurité

- **Toujours crypter les canaux de communication au moyen de :**
  - **TLS:** est un protocole cryptographique qui permet de crypter les canaux de communication. Ce protocole applique la confidentialité, l'authentification et l'intégrité des données comme propriétés du canal de communication.
  - **WebSocket:** est une technologie qui fournit un canal de communication bidirectionnel (bidirectionnel, envoi et réception) et full-duplex (simultané) sur le même socket TCP.
- **Il utilise une cryptographie puissante :** suffisamment forte pour rendre vaine toute tentative de décryptage.
- **Utiliser des protocoles de sécurité :** tels que HTTPS ou FTPS.

13. Protection des données

Les composants sur lesquels la sécurité se concentre pour protéger les données sont les suivants :



- **Authentification :** l'utilisateur qui accède aux données est bien celui qu'il prétend être et se voit attribuer des rôles ou des privilèges d'accès.
- **Autorisation :** les privilèges d'accès ou les rôles d'utilisateur n'autorisent que certains types d'opérations sur certaines données seulement.
- **Confidentialité :** les données doivent être protégées contre toute observation ou divulgation non autorisée pendant leur transit et leur stockage.
- **Intégrité :** les données doivent être protégées en cas de création, de modification ou de suppression malveillante par des attaquants non autorisés.
- **Disponibilité :** les données doivent être disponibles pour les utilisateurs autorisés chaque fois que cela est nécessaire (politiques de *sauvegarde*).

Cette norme suppose que la protection des données est mise en œuvre dans un système de confiance qui a été construit avec des garanties de sécurité suffisantes.

13.1. Risques potentiels

Toute vulnérabilité de sécurité exploitée dans les données d'une application peut conduire à :

- **Non-respect de la réglementation** et de la législation sur le traitement des données personnelles peut entraîner des sanctions financières ou l'interruption du service.
- Compromis ou **perte d'informations** sensibles de l'entreprise.
- Compromis ou **perte d'informations** sensibles de tiers, ce qui pourrait entraîner des litiges et des pertes financières.
- **Perte de l'image** de l'entreprise.
- **Perte de certifications** à la suite du non-respect de la protection des données.

13.2. Recommandations en matière de sécurité

- Vérifier que toutes les informations sensibles ou personnelles devant être traitées par l'application sont identifiées, et qu'il existe une politique explicite spécifiant comment l'accès à ces informations doit être contrôlé, traité et, lorsqu'elles sont stockées, correctement cryptées conformément aux directives correctes en matière de protection des données, dans le respect des lois et réglementations locales.
- Veiller à ce que toutes les données sensibles soient envoyées au serveur dans le corps du message HTTP ou dans les en-têtes, en évitant d'envoyer des données sensibles via des paramètres d'URL.

- Vérifier que les canaux de communication utilisés pour l'envoi de données confidentielles sont sécurisés à l'aide d'algorithmes de cryptage puissants.
- Vérifier que les données sensibles stockées sont cryptées à l'aide d'algorithmes de cryptage puissants.
- Vérifier que l'application définit suffisamment d'en-têtes contre la mise en cache, afin que toute information sensible ne soit pas stockée dans le cache des navigateurs modernes (par exemple, visiter le cache pour réviser le cache du disque).
- Veiller à ce que les informations sensibles stockées en mémoire soient remplacées par des zéros dès qu'elles ne sont plus nécessaires, afin de limiter les attaques par vidage de mémoire.
- Vérifier qu'une politique de suppression sécurisée des données est en place lorsque les actifs ont atteint la fin de leur cycle de vie.

13.3. Exemple

Un exemple de configuration du serveur Apache Tomcat pour sécuriser les communications avec TLS serait, dans le fichier server.xml :

```
<Connector
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   port="8443" maxThreads="200"
   scheme="https" secure="true" SSLEnabled="true"
   keystoreFile="mykeystore" keystorePass="<password>"
   clientAuth="false" sslProtocol="TLS"/>
```

Un exemple d'utilisation des sockets SSL en Java :

```
import javax.net.ssl.SSLSocket ;
import javax.net.ssl.SSLSocketFactory ;

// Créer une usine de socket SSL
SSLSocketFactory sslSocketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault() ;

// Créer un socket SSL et établir la connexion
SSLSocket sslSocket = (SSLSocket) sslSocketFactory.createSocket("www.example.com", 443) ;
```

Un exemple d'utilisation des connexions HTTPS en Java :

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.URL;
import java.net.URLConnection;

// Créer une connexion HTTPS
URL url = nouvelle URL("https://www.example.com") ;
URLConnection connection = url.openConnection() ;

// Envoyer une requête POST à la connexion HTTPS
connexion.setDoOutput(true) ;
OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream()) ;
out.write("param1=valeur1&param2=valeur2") ;
out.close() ;

// Lire la réponse de la connexion HTTPS
BufferedReader in = new BufferedReader(new InputStreamReader(connection.
getInputStream())) ;
String inputLine ;
while ((inputLine = in.readLine()) != null) {
   System.out.println(inputLine) ;
}
dans.close() ;
```

Un exemple d'utilisation de connexions sécurisées avec WebSockets en Javascript, côté client :

```
// Créer une connexion HTTPS avec WebSockets
const socket = new WebSocket("wss://www.example.com/ws") ;

// Envoyer un message via une connexion HTTPS
socket.send("Bonjour, monde !") ;

// Recevoir les messages de la connexion HTTPS
socket.onmessage = function(event) {
   console.log("Message reçu :", event.data) ;
};
```

13.4. Références

Algorithmes SSL/TLS
https://docs.oracle.com/en/java/javase/15/docs/specs/security/standard-names.html#sslcontext-algorithms [34]

Java : HttpClient avec SSL
https://www.baeldung.com/java-httpclient-ssl [35]

Python : SSL/TL
https://docs.python.org/3/library/ssl.html [36]

Python : WebSockets
https://pypi.org/project/websockets/ [37]

14. Python: indications complémentaires

14.1. Architecture

14.1.1. Environnement virtuel

Il est conseillé d'utiliser un environnement virtuel dans tout projet Python, qui est équipé pour séparer le développement d'applications dans des environnements virtuels.

Un environnement virtuel isole l'interpréteur Python, les bibliothèques et les scripts qui y sont installés. Cela signifie qu'au lieu d'utiliser une version globale de Python et des dépendances globales de Python pour tous les projets que vous voulez développer, vous pouvez avoir des environnements virtuels spécifiques pour chaque projet et dans chacun d'eux vous pouvez avoir vos propres versions de Python, ainsi que ses dépendances.

Les environnements virtuels facilitent le développement, le conditionnement et la livraison d'applications Python sécurisées. La plupart des IDE ont des fonctions intégrées pour passer d'un environnement virtuel à l'autre.

Un lien vers la bibliothèque Python permettant de créer des environnements virtuels **venv** se trouve ci-dessous.

https://docs.python.org/3/library/venv.html [38]

14.1.2. Importation de paquets

Lorsque vous travaillez avec des modules Python externes ou internes, assurez-vous toujours que vous les importez de la bonne manière et que vous utilisez les bons chemins. Il existe deux types de chemins d'importation en Python, absolu et relatif.

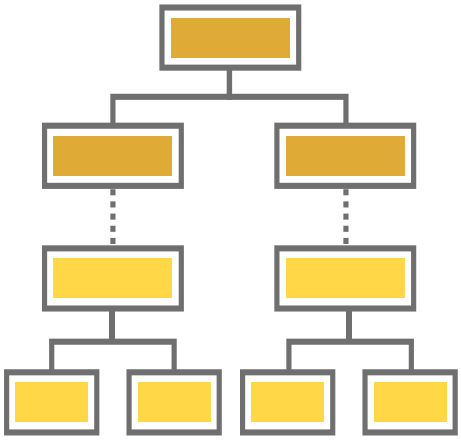
L'importation absolue spécifie le chemin de la ressource à importer en utilisant son chemin complet à partir du dossier racine du projet tandis que l'importation relative spécifie la ressource à importer par rapport à l'emplacement actuel dans le projet où se trouve la déclaration **import**.

```
/* Absolute Import */

from package1 import module1
from package1.module2 import function1


/* Relative Import */

from .some_module import some_class
from ..some_package import some_function
```



Il existe deux (2) types d'importations relatives :

- **Implicite :** les importations implicites ne précisent pas le chemin de la ressource par rapport au module actuel. L'importation implicite a été supprimée de Python 3, car si le module spécifié se trouve dans le chemin du système, il sera importé et cela pourrait être très dangereux. Il est possible qu'un module malveillant portant le même nom se trouve dans une bibliothèque open source populaire et se retrouve dans le chemin du système. Si le module malveillant est trouvé avant le module réel, il sera importé et pourra être utilisé pour exploiter les applications qui l'ont dans leur arbre de dépendances.

  Par conséquent, il convient d'utiliser l'importation absolue ou l'importation relative explicite, afin de garantir l'importation du module réel et du module prévu.

  ```
  from safe_module import package, function, class
  o
  from  ..relative_module import package, function, class
  ```

- **Explicite :** Les importations explicites spécifient le chemin exact du module à importer par rapport au module actuel.

14.2. Authentification

Le cadre Django, basé sur le langage Python, possède, dans sa configuration par défaut, un **système d'authentification** et supporte une extension et une personnalisation de l'authentification. Il fournit à la fois l'authentification et l'autorisation. Il gère les comptes d'utilisateurs, les groupes, les autorisations et les sessions d'utilisateurs sur la base de cookies. Ce système se compose de :

- **Objets utilisateurs :** ils constituent le cœur du système d'authentification. Ils représentent généralement les personnes qui interagissent avec l'application et sont utilisés pour permettre par exemple la restriction de l'accès, l'enregistrement des profils d'utilisateurs, l'association du contenu aux créateurs, etc. Il n'y a qu'un seul type d'utilisateur dans le cadre d'authentification de Django, c'est-à-dire que les utilisateurs administrateurs « super utilisateurs » ou « employés » sont juste des objets utilisateurs avec des attributs spéciaux définis.

- **Permissions et autorisation :** Django est livré avec un système de permissions intégré. Il permet d'attribuer des autorisations à des utilisateurs et des groupes d'utilisateurs spécifiques.

- **Authentification sur les requêtes web :** Django utilise des sessions et des intergiciels pour accrocher le système d'authentification aux objets **request**. Ceux-ci fournissent un attribut **request.user** sur chaque demande qui représente l'utilisateur actuel. Si l'utilisateur actuel n'est pas connecté, cet attribut sera une instance de AnonymousUser, sinon ce sera une instance de *User*.

- **Gestion des utilisateurs dans le panneau d'administration :** lorsque **django.contrib.admin** et **django.contrib.auth** sont tous deux installés, le panneau d'administration propose un mode.

Le cadre Django, basé sur le langage Python, possède, dans sa configuration par défaut, un système d'authentification et supporte une extension et une personnalisation de l'authentification

14.3. Gestion des sessions

Il est recommandé d'utiliser les implémentations par défaut des protections CSRF qui existent dans la grande majorité des cadres, comme le **cadre Django**, dont l'intergiciel CSRF est activé par défaut, de même que la balise de modèle et le **cadre Flask**.

Exemple en Python avec le cadre Django.

- **fichier settings.py :** laisser le CSRF par défaut de l'intergiciel activé dans le fichier de paramètres.

```
…
MIDDLEWARE = [
…
'django.middleware.csrf.CsrfViewMiddleware',
…
]
```

- **Fichier template_example_csrf.html :** ajouter la balise **csrf_token** à l'intérieur de l'élément <form> avec la méthode POST pour une URL interne :

```
…
<form method="post">{% csrf_token %}
…
```

- **Fichier views.py :** **RequestContext** doit être utilisé dans le rendu de la réponse pour que la balise **{% csrf %}** fonctionne correctement. Noter qu'en cas d'utilisation de la fonction **render()**, les vues génériques des applications contrib, ceci est déjà couvert car elles utilisent toutes le **RequestContext**:

```
…
def ejemplo_uso_request_context(request):
    …
return render(request, 'plantilla_ejemplo_csrf.html', {form: form, 'username': username'}))
```

14.4. Validation des paramètres d'entrée

14.4.1. Requêtes de base de données paramétrées

L'utilisation de requêtes paramétrées est recommandée pour toute application accédant à la base de données.

Exemple de code Python sécurisé avec passage sécurisé des paramètres de requête. Le fragment de code suivant définit la fonction **is_admin** qui reçoit une chaîne de caractères comme paramètre d'entrée et renvoie un booléen. Si l'utilisateur n'existe pas, il retourne False et si l'utilisateur existe, il retourne la valeur de la colonne **admin** qui peut être True si c'est un administrateur ou False si ce n'est pas un administrateur.

En paramétrant la requête et en validant si le résultat de la requête est None, une attaque par vulnérabilité d'injection SQL est évitée :

```
def is_admin(username: str) -> bool:
    with connection.cursor() as cursor:
        cursor.execute("SELECT admin FROM users WHERE username = %(username)s",{'username': username});
        result = cursor.fetchone()

    if result is None:
        # User does not exist
        return False

    admin = result
    return admin
```

```
>>> is_admin('leo')
False
>>> is_admin('irati')
True
>>> is_admin('lemon')
False
>>> is_admin("'; select true; --")
False
```

14.4.2. Protection des formulaires

Il est recommandé de protéger les formulaires qui contiennent des paramètres dont la valeur peut être modifiée par les utilisateurs. Cet objectif est atteint grâce aux mesures suivantes :

- **Codage correct de la sortie des données :** il s'agit principalement d'appliquer le codage **HTML** à toute sortie qui reproduit des données saisies par l'utilisateur à l'entrée, afin qu'elle ne puisse pas être interprétée comme du code par le navigateur :
  - Indiquer le type du contenu de la réponse dans son en-tête Content-type (voir **IANA media-types**[1] [39] pour les types supportés : **application/json, text/html**...), afin que le côté client sache comment l'interpréter.
  - Inclure l'en-tête X-Content-Type-Options=nosniff pour éviter que certains navigateurs ne détectent les messages MIME et que le contenu soit rendu différemment de ce qui est déclaré dans le Content-type.

- **Filtrer les métacaractères potentiellement dangereux :** dans les entrées vulnérables. Par exemple, les caractères « < » « > » « ; » « / » « / » "et tous les caractères non imprimables doivent être correctement filtrés à l'entrée dans l'application.

- **Appliquer des politiques de filtrage des données de formulaire.**

  Exemple de code Python sécurisé utilisant les fonctions **html.escape** (convertit les caractères &, < et > de la chaîne de caractères reçue en paramètre d'entrée en séquences sûres pour le HTML) et **html.unescape** (convertit les caractères) de la bibliothèque **html**. :

```
import html

bp1 = html.unescape('<html><head></head><body><h1>Best Practices</h1></body></html>')
bp2 = html.unescape('x > 2 && x < 7 single quote: \' double quote: "')
bp3 = html.escape('<html><head></head><body><h1>Best Practices</h1></body></html>')
bp4 = html.escape('x > 2 && x < 7 single quote: \' double quote: "')
print("UNESCAPE")
print(bp1)
print(bp2)
print("ESCAPE")
print(bp3)
print(bp4)
```

```
UNESCAPE
<html><head></head><body><h1>Best Practices</h1></body></html>
x > 2 && x < 7 single quote: ' double quote: "
ESCAPE
&lt;html&gt;&lt;head&gt;&lt;/head&gt;&lt;body&gt;&lt;h1&gt;Best Practices&lt;/h1&gt;&lt;/body&gt;&lt;/html&gt;
x &gt; 2 &amp;&amp; x &lt; 7 single quote: &#x27; double quote: &quot;
```

1 Types de médias https://www.iana.org/assignments/media-types/media-types.xhtml

- *Cookies* de session marqués de l'indicateur **HttpOnly** pour empêcher le vol de session en cas d'exploitation d'une vulnérabilité XSS.

- **Toutes les recommandations du côté du serveur doivent être mises en œuvre.**

- **Validation des entrées.** Il est recommandé de toujours utiliser une approche de type liste blanche ou liste noire, la liste blanche étant l'approche la plus recommandée où tout ce qui ne correspond pas aux spécifications est rejeté.

  Ces vérifications doivent être effectuées à la fois du côté client et du côté serveur, au minimum du côté serveur pour garantir que la logique métier traite les données de manière sécurisée (CWE-602)[2] [40].

  D'autre part, il est également nécessaire dans tout langage typé de transformer les types de données dans le type attendu, par exemple, si un String est reçu et qu'un Int est attendu, d'effectuer un transtypage ou une transformation.

  Exemple de code Python sécurisé sur liste blanche :

```
if tipo_seguro == "opciónA" or tipo_seguro == "opciónB":
    print("Sí soportado")
else:
    print("No soportado")
```

  Exemple de vérification sûre des expressions régulières du code Python :

```
import re

pat = re.compile(r"[A-Za-z0-9]+")
test = input("Introduzca cadena: ")

if re.fullmatch(pat, test):
    print(f"'{test}' Sí es cadena alfanumérica")
else:
    print(f"'{test}' No es cadena alfanumérica")
```

```
>Introduzca cadena : Alfanumerica1
  ' Ejemplo cadena alfanumérica ' Sí es cadena alfanumérica.
>Introduzca cadena : Alfanumerica 2
  ' Ejemplo cadena alfanumérica ' No es cadena alfanumérica.
```

  Exemple de vérification de type et de casting de code sécurisé Python :

```
#Integer
x = int(1)   # x es 1
y = int(2.8) # y es 2


z = int("3") # z es 3
v = "6"
if not isinstance(v,int):
    print("No es tipo int")
    u = int(v)    # u es 6

#Float
x = float(1)     # x es 1.0
y = float(2.8)   # y es 2.8
z = float("3")   # z es 3.0
w = float("4.2") # w es 4.2
v = "7"
if not isinstance(v,float):
    print("No es tipo float")
    u = float(v)    # u es 7.0

#String
x = str("s1") # x es 's1'
y = str(2)    # y es '2'
z = str(3.0)  # z es '3.0'
v = 2
if not isinstance(v,str):
    print("No es tipo string")
    u = str(v)    # u es '2'
```

2 CWE-602 https://cwe.mitre.org/data/definitions/602.html

- **Éviter l'utilisation abusive de la saisie de l'utilisateur.** L'utilisation de l'entrée utilisateur pour les appels système ou pour fournir des parties de fichiers doit être évitée. Voici un exemple Python de la façon de restreindre l'accès aux fichiers d'un répertoire spécifique :

```
import os
import sys

def is_safe_path(basedir, path, follow_symlinks=True):
    # resolves symbolic links
    if follow_symlinks:
        matchpath = os.path.realpath(path)
    else:
        matchpath = os.path.abspath(path)
    return basedir == os.path.commonpath((basedir, matchpath))

def main(args):
    for arg in args:
        if is_safe_path(os.getcwd(), arg):
            print("safe: {}".format(arg))
        else:
            print("unsafe: {}".format(arg))

if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Prévenir le XEE.** Désactiver toujours la résolution des DTD externes, de sorte que seules les DTD locales définies statiquement soient utilisées. Il est également recommandé de toujours valider la structure du document XML fourni par l'utilisateur, en se basant sur le fichier DTD de définition du format défini statiquement par le serveur.

  Exemple d'une bibliothèque d'analyseurs XML Python sécurisée : **defusedxml**[3] [41] est un paquetage purement Python avec des sous-classes modifiées de tous les analyseurs XML stdlib qui empêchent toute opération potentiellement malveillante. L'utilisation de ce paquet est recommandée pour tout code serveur qui analyse des données XML non fiables. Le paquet comprend également des exemples d'attaques et une documentation étendue sur d'autres vulnérabilités XML, telles que l'injection XPath.

  La bibliothèque **defusedxml** empêche les attaques XEE car elle n'autorise pas l'utilisation de XML avec des déclarations **<!ENTITY>** à l'intérieur de la DTD et lance l'exception **EntitiesForbidden** lorsqu'une entité est déclarée. En revanche, elle n'autorise aucun accès à des ressources locales ou distantes dans les entités externes ou les DTD et soulève l'exception **External ReferenceForbidden** lorsqu'une DTD ou une entité fait référence à une ressource externe.

  Exemple en Python avec la bibliothèque **defusedxml :**

```
from defusedxml import pulldom
data = parse('ejemplo.xml')
```

- **Normalisation.** Utiliser les fonctions natives de Python pour la normalisation du charset de toutes les informations traitées par le système.

  Exemple : bibliothèque de codecs[4] [42]:

```
codecs.encode(obj, encoding='utf-8', errors='strict')
codecs.decode(obj, encoding='utf-8', errors='strict')
```

3 Analyseur XML https://pypi.org/project/defusedxml/

4 Normalisation Codecs https://docs.python.org/3/library/codecs.html

- **Assainissement des données.** Voici quelques-uns des jeux de caractères spéciaux les plus courants et les paquets recommandés pour l'assainissement
  - HTML/URL: **html-sanitizer**[5] [43]
  - XML: **EscapingXML**[6] [44]
  - JSON: **JsonSchema**[7] [45] implémentation de la spécification JSON Schema pour Python.

  La plupart des cadres ont des fonctions d'assainissement : **Flask**[8] [46] et **Django**[9] [47]

- **Formatage des chaînes de caractères.** Python dispose de l'une des méthodes les plus puissantes et les plus souples pour formater les chaînes de caractères. Si elle n'est pas utilisée correctement, elle peut ouvrir une faille de sécurité dans le code. Python3 a introduit **f-strings**[10] [48] et **str.format()**[11] comme un moyen flexible de formater les chaînes de caractères et c'est vraiment très intéressant.

  Cependant, cela ouvre une brèche pour l'exploitation des données lorsqu'il s'agit de la saisie de l'utilisateur. Si l'application construite en Python permet aux utilisateurs de contrôler la chaîne de format, elle peut être mal utilisée pour filtrer des données sensibles. Par exemple :

```
CONFIG = {
    "API_KEY": "secret_key"
}
class User:
    name = ""
    email = ""
    def __init__(self, name, email):
        self.name = name
        self.email = email
    def __str__(self):
        return self.name

name = "Nombre"
email = "micorreo@gmail.com"
user = User(name, email)
print(f"{user.__init__.__globals__['CONFIG']['API_KEY']}")
/* secret_key */
```

  Avec cela, les données globales sensibles d'un dictionnaire CONFIG peuvent être accessibles via l'argument.
  Cependant, Python dispose d'un module de chaîne intégré qui peut être utilisé pour corriger et prévenir ce problème. Utilisation de la classe Template du module chaîne :

```
from string import Template

nombre_plantilla = Plantilla("Hola, mi nombre es $nombre.")
greeting = name_template.substitute(name="Python")
/* Hola, me llamo Python */
```

5 Assainissement HTML https://pypi.org/project/html-sanitizer/
6 Sanitisation XML https://wiki.python.org/moin/EscapingXml
7 Assainissement des schémas JSON https://python-jsonschema.readthedocs.io/en/latest/
8 Assainissement des flacons https://flask.palletsprojects.com/en/2.0.x/api/#flask.escape
9 Assainissement de Django https://docs.djangoproject.com/en/4.0/_modules/django/utils/html/
10 Formatage de la chaîne II https://docs.python.org/3/library/stdtypes.html#str.format
11 Formatage de la chaîne II https://docs.python.org/3/library/stdtypes.html#str.format

15. *Checklist* des contrôles de sécurité

ARCHITECTURE	ID	CONTROLE DE SECURITE	DESCRIPTION
	ARQ-01	Identifier les composants	Les composants qui n'ont pas été correctement identifiés constituent des risques potentiels pour la sécurité.
	ARQ-02	Composants du bastion	Veiller à ce que les configurations soient aussi sécurisées que possible : pas d'options de débogage activées, pas d'utilisateurs et de mots de passe par défaut, etc. S'assurer que seuls les ports de communication strictement nécessaires sont ouverts. État de la dernière mise à jour du système. Identification de tous les composants du système : bibliothèques, modules, cadres, services, etc. Pour chaque composant du système, effectuer le même examen de base des configurations et mettez à jour l'état.
	ARQ-03	Analyse des risques	Obtenir un rapport des composants pour lesquels des vulnérabilités ont été détectées et pour lesquels il n'existe actuellement aucun correctif de sécurité et analyser leur niveau de risque au sein de l'application.
	ARQ-04	Mises à jour du suivi	Surveiller de près les composants vulnérables afin qu'ils soient mis à jour dès que possible.
	ARQ-05	Autres mesures d'atténuation	Réaliser une étude sur la manière dont les problèmes de sécurité créés par ces vulnérabilités aux risques pourraient être évités ou atténués par des systèmes de sécurité alternatifs.

[ARCHITECTURE]	ARQ-06	**Sécurité du périmètre logique**	En installant des pare-feu, des IDS ou des dispositifs similaires, ou en segmentant le réseau.
	ARQ-07	**Sécurisation des données sensibles**	Veiller à ce que les données soient protégées par des mécanismes d'autorisation entre les environnements, par une séparation physique ou logique, et par des sauvegardes pour garantir leur disponibilité.
	ARQ-08	**Sécurité des langages de programmation**	Utiliser la version la plus récente du langage de programmation. Utiliser un environnement virtuel comme espace de travail du projet, si cela est possible en fonction du langage de programmation. Importation correcte des paquets en fonction du langage de programmation, en vérifiant minutieusement la sécurité des paquets à installer.
	ARQ-09	**Désactiver les options de débogage**	En particulier dans la production pour éviter les fuites d'informations dans les messages d'erreur détaillés.
	ARQ-10	**Environnement de développement**	Utiliser les outils de l'IDE qui effectuent une analyse sémantique et de sécurité de base.

AUTHENTIFICATION	ID	CONTROLE DE SECURITE	DESCRIPTION
	AUT-01	***Hachage pour les mots de passe***	S'assurer que les mots de passe ne sont pas stockés dans un format lisible, de sorte que si le système ou la ressource contenant les mots de passe est compromis, l'utilisateur malveillant ne peut toujours pas les utiliser.
	AUT-02	**Bouton de déconnexion**	S'assurer que chaque page de l'application comporte un lien de déconnexion, que la session expire lorsque l'utilisateur se déconnecte, et que la session expire lorsqu'une période raisonnable de non-activité s'est écoulée.
	AUT-03	**Ne pas exposer les informations d'identification**	Ne jamais exposer les informations d'identification dans l'URL.
	AUT-04	**Utiliser POST**	Lors de l'utilisation des formulaires, les méthodes POST doivent être employées pour envoyer des informations entre le client et le serveur.

[AUTHENTIFICATION]	AUT-05	**Utiliser l'authentification multifactorielle**	Utiliser l'authentification multifactorielle pour mettre en place plusieurs niveaux de sécurité pour les applications sensibles. Utiliser une authentification à deux facteurs pour l'utilisateur pour les fonctions critiques de l'application, comme la modification des mots de passe ou l'accès à des ressources particulièrement sensibles.
	AUT-06	**Blocage des comptes**	Mettre en place un verrouillage du compte après 3 tentatives de connexion infructueuses et un moyen de contacter l'administrateur pour déverrouiller le compte.
	AUT-07	**Utilisation de CAPTCHA**	Mettre en œuvre CAPTCHA pour atténuer les attaques par force brute sur les applications exposées sur Internet.
	AUT-08	**Empêcher les énumérations d'utilisateurs**	En fournissant des messages d'erreur génériques en cas d'échec de l'authentification, tels que « L'utilisateur et/ou le mot de passe fournis ne sont pas corrects ». En fournissant des temps de réponse différents en cas d'utilisateur inexistant ou de mot de passe incorrect. Dans les procédures de récupération de mot de passe qui nécessitent la saisie du nom d'utilisateur.
	AUT-09	**Désactiver « autocomplétion ».**	Désactivez l'attribut « autocomplétion » du champ du mot de passe dans l'application, car il est activé par défaut.
	AUT-10	**Exigences en matière de complexité des mots de passe**	Il doit comporter un minimum de huit caractères et un maximum raisonnable. Il doit contenir au moins trois des caractères suivants : une lettre majuscule, une lettre minuscule, un chiffre, un caractère spécial.
	AUT-11	**Ne pas stocker les cookies**	Ne pas stocker de manière persistante le cookie d'authentification sur l'ordinateur du client, et ne pas l'utiliser à d'autres fins telles que la personnalisation.
	AUT-12	**Journaliser les accès**	Enregistrer tous les accès dans un journal de sécurité spécifique en indiquant le résultat de la tentative d'accès (qu'elle ait réussi ou échoué, qu'elle ait été annulée ou bloquée).

AUTORISATIONS	ID	CONTROLE DE SECURITE	DESCRIPTION
	ATZ-01	**Privilège minimal**	Assurer la mise en œuvre du principe du moindre privilège : les utilisateurs n'ont qu'un accès restreint aux fonctions et aux données dont ils ont réellement besoin pour effectuer leur travail normalement.
	ATZ-02	**Rôles et privilèges**	Attribuer des autorisations et des privilèges aux rôles d'application, jamais directement aux utilisateurs. Les utilisateurs doivent avoir des rôles et leurs privilèges sont tirés de ces rôles.
	ATZ-03	**Protection par autorisation**	Vérifier que l'accès aux données confidentielles est protégé, de sorte que seuls les objets ou les données autorisés et accessibles à chaque utilisateur puissent être atteints.
	ATZ-04	**Navigation dans les répertoires désactivée**	Vérifier que la navigation dans le répertoire est désactivée, sauf si elle est délibérément activée.
	ATZ-05	**Contrôle d'accès sur le serveur**	S'assurez les règles de contrôle d'accès sont appliquées du côté du serveur.
	ATZ-06	**Traitement sécurisé des informations relatives aux utilisateurs**	Vérifier que tous les attributs, données et informations de politique d'utilisateur utilisés par les contrôles d'accès ne peuvent pas être manipulés par les utilisateurs finaux, sauf autorisation spécifique.
	ATZ-07	**Manipulation sûre des ressources**	Vérifier qu'il existe un mécanisme centralisé (y compris les bibliothèques qui font appel à des services d'autorisation externes) pour protéger l'accès à chaque type de ressource protégée.
	ATZ-08	***Jetons anti-CSRF***	L'application utilise des jetons aléatoires forts anti-CSRF ou met en œuvre un autre mécanisme de protection des transactions.
	ATZ-09	**Contrôle d'accès au registre**	Enregistrez toutes les opérations sur les données sensibles dans un journal de sécurité spécifique, même si elles ont été refusées.

GESTION DES SESSIONS	ID	CONTROLE DE SECURITE	DESCRIPTION
	SES-01	**En-têtes sécurisés**	Implémentation d'en-têtes sécurisés avec des directives telles que **cache-control** ou **strict-transport security.**
	SES-02	**Invalidation de la session**	Vérifier que les sessions sont invalidées lorsque l'utilisateur se déconnecte.
	SES-03	**Expiration de la session**	Les sessions doivent expirer après une période d'inactivité déterminée.
	SES-04	**Déconnexion**	Veiller à ce que toutes les pages nécessitant une authentification disposent d'un accès facile et convivial à la fonctionnalité de déconnexion.
	SES-05	**Ne pas exposer l'ID de la session**	Vérifier que l'ID de session n'apparaît jamais dans les URL, les messages d'erreur ou les journaux.
	SES-06	**Nouveaux identifiants de session**	S'assurer que chaque authentification et réauthentification réussie génère une nouvelle session et un nouvel ID de session, détruisant l'ancien.
	SES-07	***Cookie* de session**	Vérifier que l'ID de session stocké dans les cookies est défini à l'aide des attributs **HttpOnly** et **Secure**. Configurer correctement l'attribut Path des *cookies* de session pour empêcher l'accès à d'autres domaines.
	SES-08	**Sessions de suivi**	Vérifier que l'application garde la trace de toutes les sessions actives et permet à l'utilisateur de mettre fin aux sessions de manière sélective ou globale à partir de son compte. Dans le cas d'applications à haute valeur ajoutée, veillez à ce que l'utilisateur soit tenu de fermer toutes les sessions actives si le mot de passe vient d'être modifié avec succès.
	SES-09	**Enregistrer l'activité de la session**	Enregistrer dans un journal de sécurité toutes les sessions qui sont créées, fermées manuellement ou expirées depuis le client ou le serveur.

VALIDATION DES DONNÉES D'ENTRÉE ET DE SORTIE	ID	CONTROLE DE SECURITE	DESCRIPTION
	VAL-01	Valider du côté du serveur	Les validations doivent toujours être effectuées du côté du serveur.
	VAL-02	Schémas de validation	Utilisez les schémas de validation des données comme la meilleure solution pour valider la saisie des données. À défaut, utilisez les mécanismes standard de validation des entrées fournis par les bibliothèques spécifiques à la technologie.
	VAL-03	Typification des données	Vérifier que les données structurées sont fortement typées et validées selon un schéma défini, y compris les caractères autorisés, la longueur et le modèle.
	VAL-04	Nettoyage des données	Vérifier que les données non structurées sont nettoyées pour imposer des mesures de sécurité génériques, telles que les caractères et la longueur autorisés, et éviter les caractères potentiellement dangereux. Assurez-vous que toutes les entrées non fiables sont correctement nettoyées à l'aide d'une bibliothèque de nettoyage.
	VAL-05	N'acceptez que les données attendues	N'accepter que les données attendues à chaque point d'entrée de l'application. Toute donnée inattendue doit être rejetée.
	VAL-06	Protection contre les injections SQL	Veiller à ce que toutes les requêtes de la base de données soient protégées par des requêtes paramétrées afin d'éviter toute injection SQL.
	VAL-07	Codage correct des variables HTML	Pour s'assurer que l'application n'est pas susceptible d'être victime de *Cross-Site Scripting* (XSS) reflété, stocké ou DOM
	VAL-08	Contraintes de l'analyseur XML	Vérifier que l'application restreint les analyseurs XML pour qu'ils n'utilisent que les paramètres les plus restrictifs possibles et s'assurer que les fonctions dangereuses, telles que la résolution d'entités externes, sont désactivées.
	VAL-09	Désérialisation des données	Vérifier que la désérialisation des données non fiables est empêchée ou largement protégée lorsque la désérialisation ne peut être évitée.

GESTION DES ERREURS	ID	CONTROLE DE SECURITE	DESCRIPTION
	ERR-01	**Exposition de l'information dans les erreurs**	Utiliser des messages d'erreur génériques qui ne donnent pas d'indices aux utilisateurs finaux sur les aspects sensibles de l'application.
	ERR-02	**Centralisation des erreurs**	Utiliser un traitement centralisé des exceptions. L'application doit gérer les erreurs sans s'appuyer sur les messages d'erreur du serveur affichés aux utilisateurs.
	ERR-03	**Refus par défaut en cas d'erreur**	Toute logique de contrôle d'accès qui conduit à une erreur doit refuser l'accès par défaut.
	ERR-04	**Erreurs contrôlées**	Analyser en détail toutes les exceptions qui peuvent se produire en raison de l'utilisation de bibliothèques système ou de bibliothèques tierces dans l'application, les traiter de manière appropriée et fournir une sortie sûre à l'application. Utiliser **try/catch/finally** pour garantir l'arrêt de toutes les ressources en cas d'erreur.
	ERR-05	**Erreurs d'enregistrement**	Consigner toutes les exceptions inattendues dans un journal de sécurité spécifique.

JOURNAL SÉCURISÉ	ID	CONTROLE DE SECURITE	DESCRIPTION
	LOG-01	**Ne pas enregistrer d'informations sensibles**	Comme les mots de passe, les informations financières, les cartes de crédit, les données personnelles, etc. Dans ces cas, utiliser des jetons, l'anonymisation des informations ou le cryptage.
	LOG-02	**Valider les variables d'enregistrement**	Valider les paramètres des composants variables qui composeront l'entrée du journal pour éviter les injections et les comportements inattendus.
	LOG-03	**Des notes précises**	Prendre des notes suffisamment précises qui permettent de connaître les opérations et les activités de l'utilisateur en toute sécurité.
	LOG-04	**Intégrité du journal**	Utiliser les fonctions de hachage pour garantir l'intégrité des données des enregistrements.
	LOG-05	**Isolement du journal de sécurité**	Le journal de sécurité doit être indépendant des autres journaux et disposer de ses propres protections de sécurité dans le système.

CRYPTOGRAPHIE	ID	CONTROLE DE SECURITE	DESCRIPTION
	CRT-01	Cryptage d'informations sensibles	Toutes les informations sensibles d'une organisation doivent être stockées sous une forme illisible (cryptée) afin de garantir leur confidentialité.
	CRT-02	Générateur aléatoire sécurisé	Vérifier que tous les nombres aléatoires, noms de fichiers aléatoires, GUID aléatoires et chaînes aléatoires sont générés par un générateur de nombres aléatoires approuvé par le module cryptographique. Vérifier que les nombres aléatoires sont créés avec un niveau d'entropie approprié, même lorsque l'application est soumise à une charge élevée.
	CRT-03	Politique de gestion des clés	Vérifier qu'il existe une politique explicite sur le traitement des clés cryptographiques (telles que la génération, la distribution, la révocation et la dépréciation). Vérifier que le cycle de vie des clés cryptographiques est correctement mis en œuvre.
	CRT-04	Divulgation d'informations sensibles	Vérifier à ce que les mots de passe confidentiels ou les informations critiques résidant en mémoire soient remplacés par des zéros dès qu'ils ne sont plus utilisés afin d'atténuer les attaques par vidage de mémoire.
	CRT-05	Utilisation d'algorithmes robustes	Vérifier que des algorithmes cryptographiques obsolètes ou faibles, tels que l'algorithme DES à clé symétrique ou des fonctions de hachage telles que MD5 ou SHA-1, ne sont pas utilisés en raison de l'impossibilité de garantir la confidentialité. Utiliser les bibliothèques cryptographiques existantes et, dans tous les cas, utiliser des algorithmes ou des implémentations cryptographiques personnalisés ou créés par l'utilisateur.

FICHIERS SÉCURISÉS	ID	CONTROLE DE SECURITE	DESCRIPTION
	FIL-01	Autorisation de téléchargement	Authentifier et autoriser l'utilisateur avant de charger ou de télécharger des fichiers, en particulier si les données sont sensibles.
	FIL-02	Ne pas autoriser le renommage	Ne pas utiliser les entrées fournies par l'utilisateur pour nommer les fichiers ou les répertoires.
	FIL-03	Valider les types MIME	Valider les types de contenu non seulement par extension, mais aussi vérifier les types MIME pour vérifier les fichiers. Ne pas autoriser le téléchargement de fichiers exécutables vers l'application.
	FIL-04	Limiter la taille des fichiers	Limiter la taille des fichiers au minimum que le serveur peut traiter sans causer de problèmes de disponibilité et sans affecter la fonctionnalité des applications.
	FIL-05	Analyse des fichiers	Avant de traiter les fichiers vers le serveur, un scanner antivirus vérifie l'absence de logiciels malveillants ou de virus dans les fichiers.
	FIL-06	Privilèges d'annuaire	Désactiver les privilèges d'exécution sur les répertoires où les utilisateurs peuvent télécharger des fichiers. Ne pas utiliser de chemins absolus lorsqu'un lien de téléchargement est fourni à l'utilisateur.
	FIL-07	Noms de fichiers sécurisés	Ne pas stocker les fichiers avec leurs noms de manière séquentielle. Ne pas utiliser d'informations sensibles pour nommer les fichiers.
	FIL-08	Lecture seule	S'assurer que le contrôle d'accès est défini comme étant en lecture seule.
	FIL-09	Limite du fichier	Limiter le nombre de fichiers téléchargés par l'utilisateur.
	FIL-10	Intégrité des fichiers	Stocker les hachages des fichiers téléchargés pour garantir leur intégrité.

SÉCURITÉ DES TRANSACTIONS	ID	CONTROLE DE SECURITE	DESCRIPTION
	TRN-01	**Protection contre le contournement des paiements**	L'autorisation et la confirmation d'un achat doivent être effectuées du côté du serveur. Il est nécessaire de valider que les signatures utilisées sont correctes lors du processus de communication avec la passerelle de paiement. Valider que le prix est correctement défini du côté du serveur. Valider que les paiements ne sont pas réutilisés. Le serveur de paiement doit vérifier à tout moment à quel stade de la transaction vous vous trouvez.
	TRN-02	**Délai d'expiration de la transaction**	Prévoir un délai d'expiration de l'autorisation relativement court pour chaque transaction.
	TRN-03	**Traçabilité des transactions**	Assurer la traçabilité des transactions.
	TRN-04	**Crypter la communication entre les transactions**	Chiffrer les communications avec des algorithmes asymétriques robustes.
	TRN-05	**Enregistrer l'activité de la transaction**	Enregistrement détaillé de toutes les transactions avec anonymisation des informations sensibles.

SÉCURITÉ DES COMMUNICATIONS	ID	CONTROLE DE SECURITE	DESCRIPTION
	COM-01	**Toujours crypter les canaux de communication**	TLS, un protocole cryptographique pour le cryptage des canaux de communication. WebSocket, une technologie qui fournit un canal de communication bidirectionnel sur le même socket TCP.
	COM-02	**Utiliser une cryptographie puissante**	Assez forte pour rendre vaine toute tentative de déchiffrage.
	COM-03	**Utiliser des protocoles sécurisés**	Utilise des protocoles de sécurité tels que HTTPS ou FTPS.

PROTECTION DES DONNÉES	ID	CONTROLE DE SECURITE	DESCRIPTION
	DAT-01	**Politique de protection des données**	Vérifier qu'une politique explicite de protection des données est en place.
	DAT-02	**Envoi de données sensibles**	S'assurer que toutes les données sensibles sont envoyées au serveur dans le corps du message HTTP ou dans les en-têtes. Vérifier que les canaux de communication utilisés pour l'envoi de données confidentielles sont sécurisés. Vérifier que l'application définit suffisamment d'en-têtes contre la mise en cache.
	DAT-03	**Données stockées en toute sécurité**	Vérifier que les données sensibles stockées sont cryptées à l'aide d'algorithmes de cryptage puissants. Vérifier que des politiques de sauvegarde sont en place pour assurer la disponibilité des données.
	DAT-04	**Divulgation de données sensibles**	Veiller à ce que les informations sensibles stockées en mémoire soient remplacées par des zéros dès qu'elles ne sont plus nécessaires, afin de limiter les attaques par vidage de mémoire.
	DAT-05	**Suppression des données**	Il existe une politique de suppression sécurisée des données lorsque les actifs ont atteint la fin de leur cycle de vie.

16. Vulnérabilité et contrôle de la sécurité

Il est important de connaître les vulnérabilités les plus courantes situées dans le code afin de savoir comment y remédier. Ces vulnérabilités ont été croisées avec les contrôles de sécurité dans les chapitres précédents.

VULNERABILITE	CONTROLE DE SECURITE
Communication non sécurisée	Authentification
Énumération des noms d'utilisateurs	
Mot de passe faible	
Cross-Site Request Forgery (CSRF) - Falsification de demande intersite	Autorisations
Défauts d'identification et d'authentification	
Accès direct aux objets	
Contrôle d'accès	
Cross-Site Scripting (XSS)	Validation des données
Injection SQL	
Dépassement de mémoire tampon	
Falsification de journaux	
SQL dynamique	
Vulnérabilité de la redirection ouverte	
Codage de sortie	
Divulgation d'informations	
Faible gestion des sessions	Gestion des sessions
Cache des formulaires	
Divulgation d'informations	Traitement des erreurs
Divulgation d'informations	Journal
Absence de journal pour les fonctions critiques	
Stockage d'informations sensibles dans un texte non crypté	Cryptographie
Cryptographie faible	
Téléchargement de fichiers non sécurisé	Gestion sécurisée des fichiers
Divulgation d'informations	

17. Mesures de sécurité et contrôles de sécurité

MESURES DE SÉCURITÉ		GUIDE DU DEVELOPPEMENT SECURISE
op.exp	**Exploitation**	**Architecture**
op.exp.4.1, op.exp.4.2, op.exp.7.r4.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.com	**Protection des communications**	
mp.com.1.1, mp.com.4		
op.acc	**Contrôle d'accès**	**Authentification**
op.acc.5.r3.2, op.acc.5.8, op.acc.5.r1.2, op.acc.5.r6.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.sw	**Protection des applications**	**Autorisations**
mp.sw.1.r1.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.s	**Protection des services**	
mp.s.2.1		
op.pl	**Planification**	
op.pl.2.4		
op.mon	**Surveillance du système**	
op.mon.1.r2.1		
op.acc	**Contrôle d'accès**	
op.acc.4.1, op.acc.6.r5.1		

mp.eq	**Protection de l'équipement**	**Gestion des sessions**
mp.eq.2.r1.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
op.pl	**Validation des données**	**Validation des paramètres**
op.pl.2.r3.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.s	**Protection des services**	
mp.s.2.3		
op.exp	**Exploitation**	**Journal**
op.exp.5.1, op.exp.7.r2.1, op.exp.8.2		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.s	**Protection des services**	
mp.s.3.r1.1		
op.acc	**Contrôle d'accès**	
op.acc.6.r9.2		
mp.si	**Protection des moyens d'information**	**Cryptographie**
mp.si.2.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
op.exp	**Exploitation**	
op.exp.10		
op.exp	**Exploitation**	**Gestion sécurisée des fichiers**
op.exp.6.3		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.com	**Protection des communications**	**Sécurité des communications**
mp.com.2.r5.1		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ
mp.info	**Protection de l'information**	**Protection des données**
mp.info.1.1, mp.info.2		RECOMMANDATIONS EN MATIÈRE DE SÉCURITÉ

18. Glossaire

OWASP (Open Web Application Security Project) : est un projet open source dédié à l'identification et à la lutte contre les causes qui rendent les logiciels non sécurisés. La Fondation OWASP est une organisation à but non lucratif qui soutient et gère les projets et l'infrastructure de l'OWASP.

La communauté OWASP est composée d'entreprises, d'organismes de formation et de particuliers du monde entier. Ensemble, ils forment une communauté de sécurité informatique qui s'efforce de créer des articles, des méthodologies, de la documentation, des outils et des technologies qui sont publiés et peuvent être utilisés gratuitement par tous.

***Cookie* :** un petit fichier envoyé par un site web et stocké dans le navigateur de l'utilisateur, afin que le site web puisse consulter l'activité précédente du navigateur. De cette manière, il est possible d'identifier l'utilisateur qui visite un site web et de conserver un enregistrement de son activité sur le site.

CSRF (Cross Site Request Forgery) : est une vulnérabilité de type « falsification de requête intersites ». Elle consiste à tromper un utilisateur légitime pour qu'il exécute des demandes ou des actions sans son consentement, sans savoir ce qu'il fait.

MFA (Multi-Factor de Authentication) : méthode de contrôle d'accès dans laquelle un utilisateur ne se voit accorder l'accès au système qu'après avoir fourni deux ou plusieurs preuves différentes qu'il est bien celui qu'il prétend être.

XEE/JEE (Xml/Json External Entity) : est une vulnérabilité d'injection de code dans une application qui analyse les données XML/Json.

DTD (Document Type Definition) : est une définition dans un document SGML ou XML, qui spécifie les restrictions sur la structure et la syntaxe du document.

CWE : est une liste de types de faiblesses logicielles et matérielles développée par la communauté. Il sert de langage commun, d'étalon pour les outils de sécurité et de référence pour l'identification des faiblesses, l'atténuation et les efforts de prévention.

19. Références

[1] « Design Patterns » , [En ligne]. Disponible : https://refactoring.guru/es/design-patterns

[2] « OWASP : Design Secure Web Applications », [En ligne]. Disponible : https://owasp.org/www-pdf-archive/APAC13_Ashish_Rao.pdf

[3] « Ámbitos de la Seguridad Nacional : Protección de Infraestructuras Críticas », [En ligne]. Disponible : file:///Users/lagor/Downloads/BOE-400_Ambitos_de_la_Seguridad_Nacional_Proteccion_de_Infraestructuras_Criticas.pdf

[4] « Java Secure Authentication », [En ligne]. Disponible : https://docs.oracle.com/javaee/5/tutorial/doc/bncbe.html#bncbn

[5] «Python: Librería para implementar OTP,» [en línea]. Available: https://pypi.org/project/pyotp/

[6] «Java: Autorisations Segura,» [en línea]. Available: https://docs.oracle.com/en/java/javase/19/security/java-authentication-and-authorization-service-jaas1.html

[7] «Python: Librería segura de autorización simple,» [en línea]. Available: https://pypi.org/project/python-authorization/

[8] « OWASP : Gestion des sessions sécurisées », [En ligne]. Disponible :: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[9] « Java : SpringSession Framework », [En ligne]. Disponible : https://www.baeldung.com/spring-session

[10] « OWASP : Validation des paramètres », [En ligne]. Disponible : https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[11] « OWASP : Validation des données - Bibliothèque ESAPI », [En ligne]. Disponible : https://owasp.org/www-project-enterprise-security-api/

[12] « Validating Forms in Javascript », [En ligne]. Disponible : https://www.tutorialspoint.com/javascript_form_validation_web_application/index.asp

[13] « Java : Spring Boot Validation », [En ligne]. Disponible : https://www.baeldung.com/spring-boot-bean-validation

[14] « Prévention XSS », [En ligne]. Disponible : https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[15] « Validation Colander », [En ligne]. Disponible : https://pypi.org/project/colander/

[16] « Validation Cerberus », [En ligne]. Disponible : https://docs.python-cerberus.org/en/stable/

[17] « Schémas de validation », [En ligne]. Disponible : https://schematics.readthedocs.io/en/latest/

[18] «« Schema Validation », [En ligne]. Disponible : https://pypi.org/project/schema/

[19] « JSON Schema Validation », [En ligne]. Disponible : https://pypi.org/project/jsonschema/

[20] « Java : Try - Catch - Finally », [En ligne]. Disponible : https://www.w3schools.com/java/java_try_catch.asp

[21] « Java : Exception Handling », [En ligne]. Disponible : https://www.baeldung.com/java-exceptions

[22] « Stratégies de gestion des erreurs », [En ligne]. Disponible : https://dzone.com/articles/error-handling-strategies

[23] « Python : Erreurs et exceptions », [En ligne]. Disponible : https://docs.python.org/es/3/tutorial/errors.html#errors-and-exceptions

[24] « Journal de sécurité : Meilleures pratiques pour l'enregistrement et la gestion », [En ligne]. Disponible : https://www.dnsstuff.com/security-log-best-practices

[25] « java : Logging », [En ligne]. Disponible : https://docs.oracle.com/javase/7/docs/technotes/guides/logging/index.html

[26] « OWASP : Cryptographie sécurisée en Java », [En ligne]. Disponible : https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions

[27] « Fonction de dérivation de clé scrypt », [En ligne]. Disponible : https://www.tarsnap.com/scrypt.html

[28] « Python : Fonctions cryptographiques bcrypt », [En ligne]. Disponible : https://pypi.org/project/bcrypt/

[29] « Serveur Apache : directives de configuration », [En ligne]. Disponible : https://httpd.apache.org/docs/2.4/mod/core.html#limitrequestbody

[30] « Java : Canonisation du chemin », [En ligne]. Disponible : https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/File.html#getCanonicalPath()

[31] « PHP : Canonisation du chemin », [En ligne]. Disponible : https://www.php.net/manual/es/function.realpath.php

[32] « Python : Mappage de fichiers avec le type MIME », [En ligne]. Disponible : https://docs.python.org/3/library/mimetypes.html

[33] « WordPress : Plugin pour éviter la vulnérabilité du Bypass de paiement », [En ligne]. Disponible : https://www.acunetix.com/vulnerabilities/web/wordpress-plugin-nab-transact-security-bypass-2-1-0/

[34] « Algorithmes SSL/TLS », [En ligne]. Disponible : https://docs.oracle.com/en/java/javase/15/docs/specs/security/standard-names.html#sslcontext-algorithms

[35] « Java : httpClient avec SSL », [En ligne]. Disponible : https://www.baeldung.com/java-httpclient-ssl

[36] « Python : SSL/TLS », [En ligne]. Disponible : https://docs.python.org/3/library/ssl.html

[37] « Python : WebSockets », [En ligne]. Disponible : https://pypi.org/project/websockets/

[38] « Environnement virtuel », [En ligne]. Disponible : https://docs.python.org/3/library/venv.html

[39] « Types de médias », [En ligne]. Disponible : https://www.iana.org/assignments/media-types/media-types.xhtml

[40] « CWE-602 », [En ligne]. Disponible : https://cwe.mitre.org/data/definitions/602.html

[41] « Analyseur XML », [En ligne]. Disponible : https://pypi.org/project/defusedxml/

[42] « Codecs Standardisation », [En ligne]. Disponible : https://docs.python.org/3/library/codecs.html

[43] « Assainissement HTML », [En ligne]. Disponible : https://pypi.org/project/html-sanitizer/

[44] « Assainissement XML », [En ligne]. Disponible : https://wiki.python.org/moin/EscapingXml

[45] « Assainissement JSON Schema », [En ligne]. Disponible : https://python-jsonschema.readthedocs.io/en/latest/

[46] « Assainissement Flask », [En ligne]. Disponible : https://flask.palletsprojects.com/en/2.0.x/api/#flask.escape

[47] « Assainissement Django », [En ligne]. Disponible : https://docs.djangoproject.com/en/4.0/_modules/django/utils/html/

[48] « Formatage de chaînes I », [En ligne]. Disponible https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals

[49] « Formatage de chaînes II », [En ligne]. Disponible : https://docs.python.org/3/library/stdtypes.html#str.format

ANNEXE A. Tableau des contrôles de base

PRINCIPES DU DÉVELOPPEMENT SÉCURISÉ		*CHECKLIST* DES CONTRÔLES DE SÉCURITÉ	
Privilège minimum	Un acteur doit avoir le niveau minimum de permissions sur les ressources du système pendant le minimum de temps possible. L'accès aux ressources doit être refusé par défaut.	**Système d'autorisation**	Diviser une application en un système d'autorisation dès le départ, empêchant tout agent non autorisé d'accéder à l'ensemble de l'application. Par exemple, un système RBAC.
Séparation des responsabilités	Toute tâche complexe ou délicate devrait nécessiter l'implication de plus d'un acteur avec des niveaux de rôle différents. Cela permet d'éviter qu'un seul acteur ne compromette l'ensemble du système.	**Paramétré**	Utilisation de requêtes paramétrées pour l'accès de toute application à la base de données.
Défense en profondeur	Établir des mécanismes de sécurité à plusieurs niveaux, avec différents niveaux de complexité et de facteurs de contrôle. L'objectif est d'éviter le « Single Point of Failure» .	**Protection des formulaires**	Protection des formulaires où il y a des paramètres dont la valeur peut être modifiée par les utilisateurs. Cette protection consiste à chiffrer correctement la sortie des données, à filtrer les métacaractères potentiellement dangereux dans les entrées vulnérables et à appliquer des politiques de filtrage des données de formulaire.
Échec certain	En cas de défaillance, le système doit être remis dans un état sûr, en réduisant au minimum les risques d'atteinte à la confidentialité, à l'intégrité ou à la disponibilité du système.	**Validation des données d'entrée**	Validation de toute zone d'entrée et de manière efficace, y compris les données provenant de l'internet, des clients, des fournisseurs et des régulateurs.
Économie des mécanismes	La mise en œuvre des fonctionnalités et des contrôles de sécurité d'un système doit être aussi simple que possible. Plus simple, cela signifie que moins de choses peuvent mal tourner.	**Méthode de transaction sécurisée**	• L'autorisation et la confirmation d'un achat doivent être effectuées du côté du serveur. • Valider que les signatures utilisées sont correctes lors du processus de communication avec la passerelle de paiement. • Valider que le prix est correctement défini du côté du serveur. • Valider que les paiements ne sont pas réutilisés. • Le serveur de paiement doit vérifier à tout moment à quel stade de la transaction vous vous trouvez. • L'autorisation de chaque transaction doit avoir une période d'expiration relativement courte.
Médiation complète	Toute demande d'accès aux ressources du système doit être validée afin que les contrôles d'authentification et d'autorisation ne puissent être contournés.	**Utilisation de la MFA**	Mise en œuvre d'une authentification à facteurs multiples pour les applications de traitement de données sensibles exposées à l'internet et à l'accès d'utilisateurs privilégiés.

Conception ouverte	Si une conception est robuste, il n'est pas nécessaire de la garder secrète pour garantir sa sécurité. Les détails de mise en œuvre doivent être indépendants de la conception. Ne pas pratiquer pas la « sécurité par l'obscurité» .	**Protection des données sensibles**	Effectuer une série d'actions dès le début de la phase de développement d'une application, telles que : • Identifier les données à traiter qui sont sensibles au regard des exigences réglementaires en matière de confidentialité. • Appliquer des contrôles tels que le cryptage en transit ou au repos. • Ne pas stocker pas de données sensibles inutilement. • Désactiver la mise en cache des données sensibles. • Avoir des données cryptées au repos. • Tout le trafic de données doit être crypté avec des méthodes telles que TLS. • Stocker les mots de passe en utilisant des algorithmes de hachage puissants avec un facteur de service qui ralentit le craquage éventuel des mots de passe
Mécanisme le moins commun	Les ressources partagées entre les différents utilisateurs du système doivent être limitées au minimum, pour des raisons de confidentialité et de concurrence.	**Protections CSRF**	Inclure l'implémentation par défaut des protections CSRF dans le cadre utilisé.
Acceptabilité psychologique	L'impact des contrôles de sécurité sur la facilité d'utilisation du système doit être pris en compte. Idéalement, les contrôles de sécurité devraient être transparents pour l'utilisateur.	**Prévenir XEE/JEE**	Désactiver la résolution des DTD externes et valider la structure du document XML.
Maillon faible	Le principal point de compromission d'un système doit être identifié et les contrôles de sécurité nécessaires doivent être mis en œuvre pour le protéger. Un système n'est sécurisé que par rapport à son maillon le plus faible.	**Éviter l'utilisation de certains intrants**	Évitez d'utiliser l'entrée utilisateur pour les appels système ou pour fournir des parties de fichiers.
Tirer parti des composants existants	Il est conseillé d'encourager la réutilisation de composants éprouvés et de solutions établies dans l'architecture du système.	**Utiliser une version actualisée du langage**	Utilisez la version la plus récente du langage
Pour plus d'informations sur les principes de développement sécurisé, voir CCN-CERT - WorkShop - Practical Approach to Secure Application Development v 1.0		**Utiliser un environnement virtuel**	Utilisation d'un environnement virtuel comme espace de travail du projet, le cas échéant selon le langage de programmation.
		Importation correcte des paquets	Effectuer le chemin d'importation en fonction du langage de programmation.
		Utiliser le formatage Chaînes sécurisées	Formatage des chaînes de saisie de l'utilisateur d'une manière qui ne permet pas de contrôler la chaîne de format et empêche le filtrage des données sensibles.
		Utilisation sécurisée des requêtes HTTP	Traiter les requêtes HTTP de manière sécurisée en évitant les requêtes vers des sources exploitées qui peuvent renvoyer du code exploité dans les en-têtes ou dans le corps de la réponse.
		Paquets installés et importés Sécurisés	Contrôle approfondi de la sécurité des paquets à installer.
		Désérialisation sécurisée des données	Utiliser les fonctions de la bibliothèque de désérialisation qui empêchent les vecteurs d'attaque.
		Maintenir les vulnérabilités à jour	Maintenir à jour les vulnérabilités Open Source dans les paquets installés et importés.
		Désactiver le débogage en production	Définir le débogage de production sur False pour éviter les fuites d'informations dans les messages d'erreur détaillés.
		Balayage des codes	Utilisez les outils de l'IDE qui effectuent une analyse sémantique.

ANNEXE B. Tableau des contrôles avancés

PRINCIPES DU DÉVELOPPEMENT SÉCURISÉ	*CHECKLIST* DES CONTRÔLES DE SÉCURITÉ	ENTITÉS DE RÉFÉRENCE		CARTOGRAPHIE DES CONTRÔLES DE SÉCURITÉ	
Privilège minimum	Système d'autorisation	**ISO**	Promouvoir la sécurité, la clarté et la fiabilité des produits et pour diverses industries en publiant des normes applicables au niveau mondial.	**Communication non sécurisée**	Authentification
Séparation des responsabilités	Paramétré	**NIST**	Physical Sciences Laboratory et une agence non réglementaire du ministère américain du commerce.	**Énumération des noms d'utilisateurs**	
Défense en profondeur	Protection des formulaires	**OWASP**	Communauté en ligne qui fournit gratuitement des articles, des méthodologies, de la documentation, des outils et des technologies dans le domaine de la sécurité des applications web.	**Mot de passe faible**	
Échec certain	Validation des données d'entrée	**MITRE**	Organisation qui fournit des services d'ingénierie des systèmes, de recherche, de développement et de soutien informatique au gouvernement américain.	**Cross Site Request Forgery (CSRF) Falsification de demande intersite**	Autorisation
Économie des mécanismes	Méthode de transaction sécurisée	**CONTRÔLES PROACTIFS**	Pour plus d'informations sur les principes de développement sécurisé, voir CCN-CERT - WorkShop - Practical Approach to Secure Application Development v 1.0	**Échec de l'identification et de l'authentification**	
Médiation complète	Utilisation de la MFA	**Définir les exigences de sécurité**		**Accès direct aux objets**	
Conception ouverte	Protection des données sensibles	**Tirer parti des cadres et des bibliothèques de sécurité**		**Contrôle d'accès**	
Mécanisme le moins commun	Protections CSRF	**Accès sécurisé aux bases de données**		**Cross Site Scripting (XSS)**	Validation des données
Acceptabilité psychologique	Prévenir XEE/JEE	**Codage et échappement des données**		**Injection SQL**	
Maillon faible	Éviter l'utilisation de certains intrants	**Valider toutes les entrées**		**Dépassement de tampon**	
Tirer parti des composants existants	Utiliser une version actualisée de la langue	**Mise en œuvre de l'identité numérique**		**Falsification de dossiers**	

Utiliser un environnement virtuel	**Renforcer les contrôles d'accès**		**SQL dynamique**	
Importation correcte des paquets	**Protéger les données en permanence**		**Vulnérabilité de la redirection ouverte**	
Utiliser le formatage chaînes sécurisées	**Mettre en œuvre la surveillance de la sécurité**		**Codage de sortie**	
Utilisation sécurisée des requêtes HTTP	**Traiter toutes les erreurs et exceptions**		**Divulgation d'informations**	
Paquets installés et importés Sécurisés			**Faible gestion des sessions**	Gestion des sessions
Désérialisation sécurisée des données			**Cache des formulaires**	
Maintenir les vulnérabilités à jour			**Divulgation d'informations**	Traitement des erreurs
Désactiver le débogage en production			**Divulgation d'informations**	Registre
Balayage des codes			**L'enregistrement n'existe pas pour les fonctions critiques**	
			Stockage d'informations sensibles dans un texte non crypté	Cryptographie
			Cryptographie faible	
			Téléchargement de fichiers non sécurisé	Gestion sécurisée des fichiers
			Divulgation d'informations	

centro criptológico nacional

centro criptológico nacional

www.ccn.cni.es

www.ccn-cert.cni.es

oc.ccn.cni.es


ccn-cert
centro criptológico nacional
CCN
centro criptológico nacional


PUBLICACIONES
Pd
DE DEFENSA


GOBIERNO
DE ESPAÑA
MINISTERIO
DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA
SUBDIRECCIÓN GENERAL
DE PUBLICACIONES
Y PATRIMONIO CULTURAL