INSTRUCTION AU RESEAU

Type d'instruction : ☐ C ☐ LR ☒ IT	**Date de publication :** 2024
Numéro de l'instruction : IT-2024-101	
Processus S1 : Politique appliquée de sécurité dans les développements	
Résumé : Identification des bonnes pratiques de sécurité à appliquer dans les développements	

Emetteur : Direction : DSI/DCISN Département / pôle : Cyberdéfense	**A l'attention de :** Madame, Monsieur le Directeur, Madame, Monsieur le Directeur comptable et financier des Caisses d'Allocations Familiales et Centres de ressources
Référents à contacter :	**Informé(s) :** [Informé(s)]

Organismes destinataires : ☒ Caf ☐ Caisses multibranches ☒ Centre de Ressources
☒-Autres : -Cnaf
☐ Caf pivots ☐ Caf adhérentes

Champ d'application : ☒ Métropole ☒ DOM ☒ Mayotte

Processus de rattachement : S1 - Délivrer et garantir l'accès sécurisé au système d'information

Diffusion : ☒ Diffusion réseau ☐ Diffusion caf.fr ☐ Communicable loi CADA

Texte(s) de référence : ○ LR 2017-068 Politique de Sécurité des Systèmes d'Information (PSSI).	**Documents abrogés ou modifiés :** ○ [Liste des documents]

Action(s) à réaliser & échéances :

☒ Pour application ☐ Pour recommandation ☐ Pour information

Mots-clés : PROCESSUS S1, SECURITE INFORMATIQUE, DEVELOPPEMENT,OWASP	**Nombre de page(s) : 25** **Nombre et liste des annexes : 0**

Applicable à compter du : 2024

Applicable jusqu'au : Sans limitation de durée

Sécurité du système d'information de la branche Famille

Documentation de référence

Politique appliquée de sécurité dans les développements

ALLOCATIONS FAMILIALES
Caisse nationale

- ☐ public
- ☒ interne
- ☐ diffusion restreinte
- ☐ confidentiel

Janvier 2024

REVISIONS DU DOCUMENT

Date	Objet
Janvier 2024	Version initiale DSI/MCIS/S&P

Sommaire

1. Objectif

Ce document identifie les bonnes pratiques de sécurité à appliquer dans les développements, il est à destination de la communauté des développeurs et opérationnels.

2. Références

Ce document se base principalement sur le travail effectué par l'OWASP **Open Web Application Security Project.** Une communauté en ligne reconnue dans le monde de la sécurité des systèmes d'information pour ses travaux et recommandations liées aux applications Web.

Parmi les projets gérés par la communauté OWASP, les plus connus sont :

- Top Ten OWASP : le but de ce projet est de fournir une liste des dix risques de sécurité applicatifs Web les plus critiques. Ce classement fait référence aujourd'hui dans le domaine de la sécurité : il est cité par de nombreux organismes d'audits et de sécurisation des systèmes d'information (DoD, PCI Security Standard).
- OWASP Testing Guide : il s'agit d'un document d'aide pour évaluer le niveau de sécurité d'une application Web.
- OWASP Code Review Guide : il s'agit d'un document présentant une méthode de revue de code sécurité.

Dans la suite du document les références au Top Ten de l'OWASP se basent sur **la version 2021.**

3. Le Top Ten de l'OWASP

Les attaquants peuvent potentiellement utiliser différents chemins à travers une application pour porter atteinte au métier ou à l'entreprise.

Le Top Ten de l'OWASP se concentre sur l'identification des risques les plus graves ayant une probabilité de survenance et un impact technique suivant un schéma d'évaluation éprouvé. Il fournit également des techniques de base pour se protéger et fournit des conseils sur la direction à suivre.

Source : https://owasp.org/Top10/fr/

3.1. Table des matières

Le Top Ten de l'OWASP est découpé par catégorie de la manière suivante :

- Facteurs :

Ils correspondent aux éléments qui ont permis de noter la catégorie.

- Aperçu

Il s'agit principalement de l'historique et du positionnement de la catégorie dans le Top Ten.

- Description

Les détails sur les vulnérabilités associées à la catégorie sont expliqués.

- Comment s'en prémunir

Des recommandations sont fournies pour les vulnérabilités listées dans le descriptif. Celles-ci ne sont pas détaillés et restent « seulement » des bonnes pratiques.

- Exemple de scénarios d'attaque

Des scénarios sont décrits pour contextualiser l'exploitation des vulnérabilités si elles sont présentes dans le code développé.

- Références

L'OWASP fournit des aides plus précises et des documentations plus techniques pour chaque recommandation.

- Liste des CWEs associées

Il est possible de retrouver toutes les références à la Common Weakness Enumeration du MITRE.

3.2. Top Ten 2017 et Top Ten 2021

Le Top Ten de l'OWASP évolue régulièrement et les changements n'ont pas d'impact vis-à-vis de ce qui est déjà engagé en terme de processus de détection et de correction.

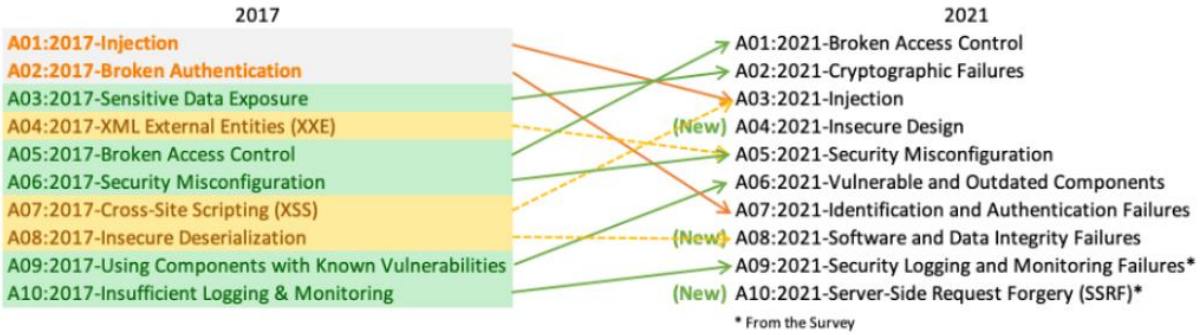

Figure 1 : changement de 2017 à 2021

Les changements notables entre ces deux versions :

- Changement des noms des catégories pour mettre en avant les causes
- **Exposition de données sensibles** devient **Défaillances cryptographiques**
- Les vulnérabilités de type **XSS** sont intégrées dans la catégorie **Injection**
- Les vulnérabilités de type **XML Entités externes (XXE)** sont intégrées dans la catégorie **Mauvaise configuration de sécurité**
- Les vulnérabilités de type **Désérialisation non sécurisée** sont intégrées dans la catégorie **Manque d'intégrité des données et du logiciel**
- **Supervision et Journalisation Insuffisantes** devient **Carence des systèmes de contrôle et de journalisation**

4. Présentation du SAST Sonar

Sonar Qube est un SAST : Static Application Security Testing. Les technologies SAST sont conçues pour analyser le code source d'une application. Ces solutions analysent une application « de l'intérieur » dans un état non exécuté. On parle généralement d'audit de code statique.

Le schéma ci-dessous décrit l'emplacement de SONAR dans le workflow de déploiement de la CNAF.

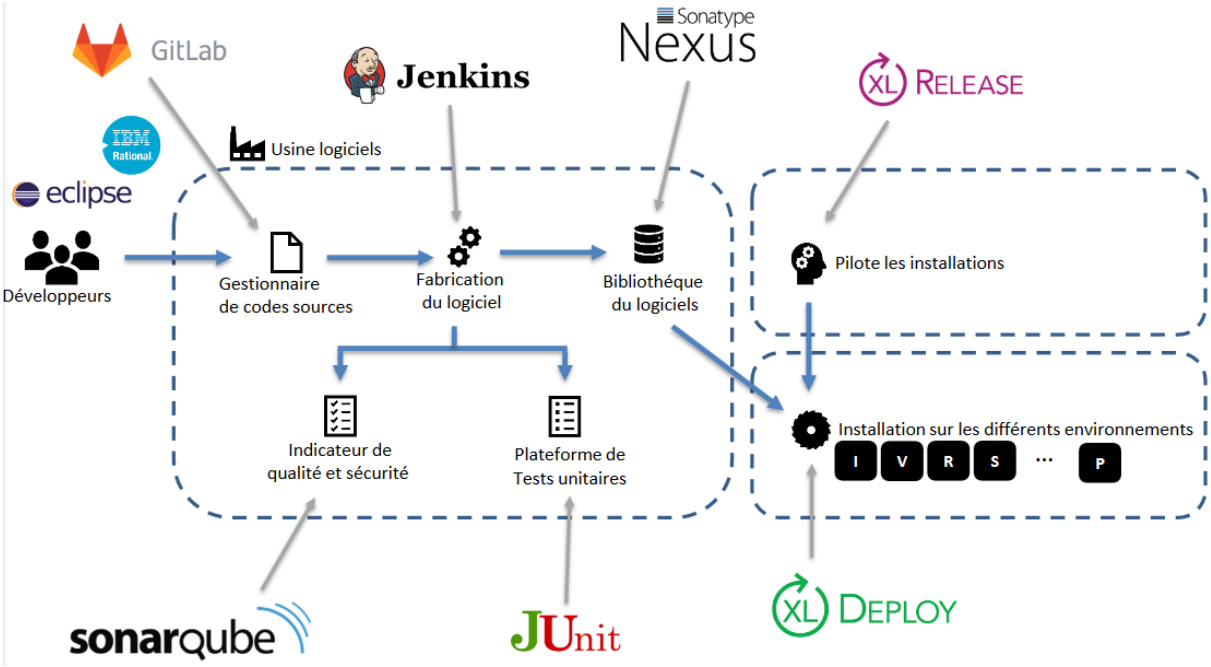

Figure 2 : Workflow de déploiement (src OCDA)

Tous les projets qui poussent leurs codes sur GitLab passent obligatoirement par Sonar Sécurité.

Tous les développeurs ont accès à SONAR en consultation pour leurs projets. Ils ne voient pas les projets « voisins ».

L'accès se fait via le portail OCDA : http://ocda.intra.cnaf/portail

4.1. Profils de sécurité

Plusieurs profils de sécurité sont définis dans Sonar et appliqués à l'ensemble des projets. Un certain nombre de règles de sécurité sont associées à ces profils, créés en fonction du Top Ten de l'OWASP.

Les références autres que celles du Top Ten de l'OWASP subsistent dans l'outil et sont accessibles. Il s'agit des références suivantes : le TOP25 du SANS, le CWE du MITRE et le WASC du WebAppSec.

Quelques exemples de profils :

- CSS avec 23 règles
- Java avec 158 règles
- JavaScript avec 140 règles

Quelques règles :

HTTP responses should not be vulnerable to session fixation	Java	Vulnerability	cwe, owasp-a1
I/O function calls should not be vulnerable to path injection attacks	Java	Vulnerability	cwe, owasp-a1, …
JSON operations should not be vulnerable to injection attacks	Java	Vulnerability	cwe, owasp-a1
LDAP queries should not be vulnerable to injection attacks	Java	Vulnerability	cert, cwe, owas…

Figure 3 : Exemple de règles Java

4.1. Etat des projets

Tous les projets sont présentés avec différents indicateurs dont le nombre de vulnérabilité(s).

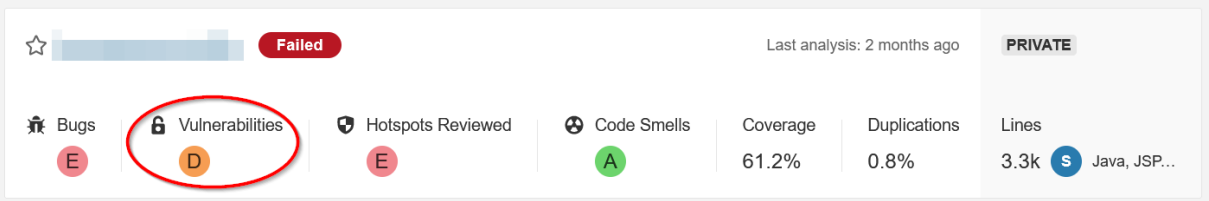

Figure 4 : Exemple de Dashboard résumé d'un projet

Chaque projet sous Sonar est soumis à une « barrière » de qualité suivant un ratio défini en amont. Celui-ci est calculé en fonction du nombre de vulnérabilité(s) et de sa gravité.

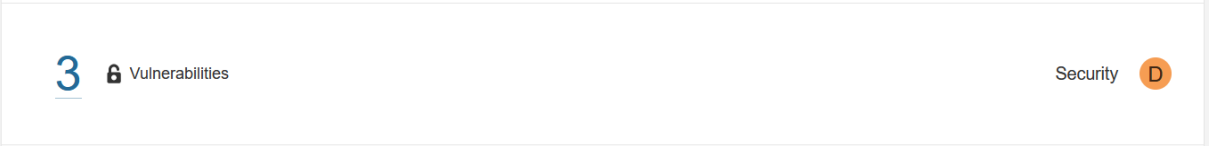

Figure 5 : Exemple de ratio sécurité d'un projet (ici D)

4.2. Vulnérabilité par projet

La fenêtre de détail sur les problèmes relevés par Sonar est composé de deux volets :

- Le premier, à gauche, permet de filtrer automatiquement les vulnérabilités non résolues sur les critères les plus importants (« Bloquants », « Critiques » et « Majeurs »)
- Le second, à droite, résume chaque vulnérablité filtrée

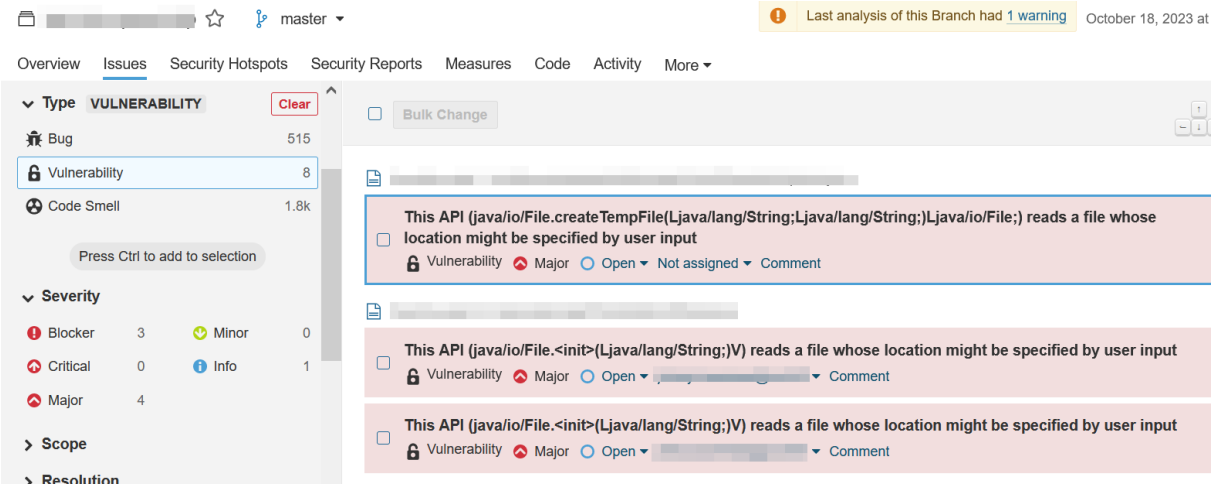

Figure 6 : exemple du détail des vulnérabilités

Chaque vulnérabilité renvoi vers le code incriminé par Sonar.

Figure 7 : exemple détaillé d'une vulnérabilité

5. Corriger avec l'aide de Sonar

5.1. Identifier la vulnérabilité

Sonar taggue systématiquement la vulnérabilité trouvée par un ou plusieurs mots clef. Ces mots clef sont associés à la règle de sécurité qui a trouvé le problème. Il est ainsi assez simple de retrouver la référence présente dans le Top Ten.

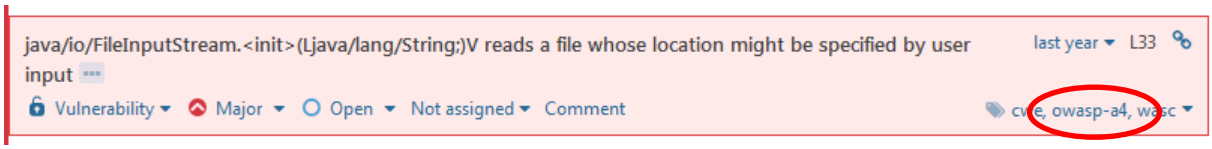

Figure 8 : tag Owasp A4

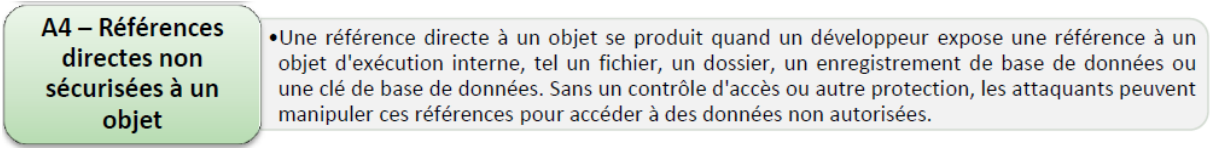

Figure 9 : la référence du Top Ten

5.2. Trouver de l'aide pour corriger

Sonar propose une aide qui regroupe :

- Une explication sur la vulnérabilité
- Un exemple de code vulnérable
- Un exemple de solution type
- Des références en lignes (Owasp, etc…)

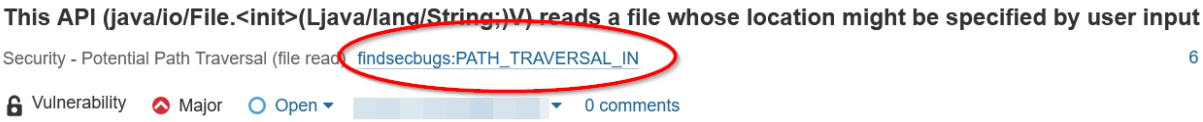

Figure 10 : lien vers l'aide de Sonar

Security - Potential Path Traversal (file read)

findsecbugs:PATH_TRAVERSAL_IN

Vulnerability Major cwe, owasp-a4, wasc Available Since Mar 22, 2019 Find Security Bugs (Java)

A file is opened to read its content. The filename comes from an **input** parameter. If an unfiltered parameter is passed to this file API, files from an arbitrary filesystem location could be read.

This rule identifies **potential** path traversal vulnerabilities. In many cases, the constructed file path cannot be controlled by the user. If that is the case, the reported instance is a false positive.

Vulnerable Code:

```
@GET
@Path("/images/{image}")
@Produces("images/*")
public Response getImage(@javax.ws.rs.PathParam("image") String image) {
    File file = new File("resources/images/", image); //Weak point

    if (!file.exists()) {
        return Response.status(Status.NOT_FOUND).build();
```

Figure 11 : Le descriptif de la vulnérabilité et un exemple de code

Solution:

```
import org.apache.commons.io.FilenameUtils;

@GET
@Path("/images/{image}")
@Produces("images/*")
public Response getImage(@javax.ws.rs.PathParam("image") String image) {
    File file = new File("resources/images/", FilenameUtils.getName(image)); //Fix

    if (!file.exists()) {
        return Response.status(Status.NOT_FOUND).build();
    }

    return Response.ok().entity(new FileInputStream(file)).build();
}
```

References

WASC: Path Traversal

OWASP: Path Traversal

CAPEC-126: Path Traversal

Figure 12 : une solution type et des références en ligne

5.3. Les exigences de sécurité

La CNAF héberge, manipule et restitue une très grande quantité de données et notamment des données à caractère personnel. Elle utilise également des ressources machines importantes en termes de CPU, stockage, etc… Toutes ces différentes ressources attirent un panel d'acteurs malveillants de la sphère « cyber criminelle ». C'est pourquoi les développements doivent être exempts de toute vulnérabilité importante afin de limiter au maximum notre exposition au risque.

Pour limiter au maximum ces risques, des principes obligatoires sont définis, à savoir :

- **Toutes les applications** ne doivent pas passer en production si elles ont des vulnérabilités bloquantes d'après SONAR.
- Les applications de **niveau 0** ne doivent pas passer en production si elles ont des vulnérabilités bloquantes ou critiques ou majeures ou mineures d'après SONAR.
- Les applications de **niveau 1** ne doivent pas passer en production si elles ont des vulnérabilités bloquantes ou critiques ou majeures d'après SONAR.
- Les applications de **niveau 2** ne doivent pas passer en production si elles ont des vulnérabilités bloquantes ou critiques d'après SONAR.

Le niveau est défini en fonction de la criticité de l'application (exposition, données hébergées, etc…).

Niveau de l'application	Type de projet
0	- Projets relevant du Système d'Information Essentiel ou Réglementé (directive NIS v1/v2) - Projets relevant du Référentiel Général de Sécurité (RGS) - Projets traitant des Données à Caractère Personnel (DCP) avec des composants exposés sur Internet
1	- Projets traitant des Données à Caractère Personnel (DCP) sans composants exposés sur Internet - Projets traitant des Données à Caractère Personnel (DCP) avec un hébergement externe (SaaS, PaaS, IaaS)
2	- Projets ne traitant pas de données à Caractère Personnel (DCP)

5.4. Les règles dans SONAR

Ces niveaux sont définis par les **Quality Gates** sous Sonar (« barrière » de qualité).

Metric	Operator	Value
Blocker Issues	is greater than	0

Figure 13 : Quality Gate actuelle (niveau 0)

Les projets qui ne sont pas exempts de vulnérabilité bloquante ne passent pas cette barrière.

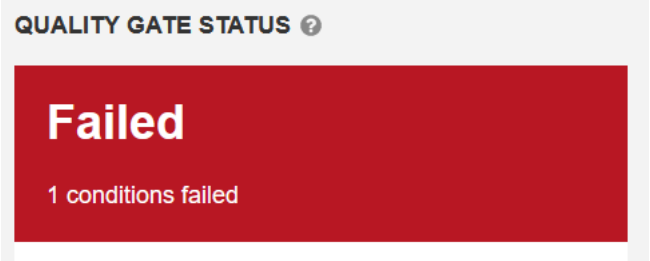

Figure 14 : Projet non conforme

6. Les règles pour couvrir les risques

6.1. Couvrir le risque de manque de contrôle d'accès

Source : https://owasp.org/Top10/fr/A01_2021-Broken_Access_Control/

6.1.1. Restreindre l'accès aux seuls utilisateurs authentifiés

Le développeur (DEV) **DOIT** restreindre l'accès aux seuls utilisateurs authentifiés. Dans le cadre des APIs, il est nécessaire de vérifier la signature de jeton d'accès (JWT) transmis par l'application Front.

1 - Cette règle ne s'applique pas si la page est publique.

2 - Une solution de vérification du jeton JWT est disponible pour les développeurs d'API (se reporter à l'annexe pour plus de détail). Il est recommandé de s'appuyer sur cette offre pour effectuer ce contrôle.

6.1.2. Renforcer les permissions basées sur les rôles

Pour éviter les accès à des fonctionnalités sans autorisation, le développeur (DEV) **DOIT protéger** ces fonctionnalités en vérifiant que l'utilisateur a le droit de les utiliser à travers ses rôles.

6.2. Couvrir le risque de défaillance cryptographique

Source : https://owasp.org/Top10/fr/A02_2021-Cryptographic_Failures/

Source : « Politique Appliquée de chiffrement » pour obtenir les recommandations de choix et de version d'algorithmes. ».

6.2.1. Chiffrer les données en transit

Toutes les données en transits doivent être chiffrées afin de garantir leur confidentialité.
Exemple : HTTPS, JDBC over TLS, etc…

6.2.2. Stocker les données persistantes dans des bases chiffrées

Toutes les données sensibles ou à caractère personnel persistantes sur le système d'information doivent être stockées dans des bases chiffrées (SQL ou NOSQL). **Il est strictement interdit de faire persister des données sensibles ou à caractère personnel sur une VM Hôte.**

6.2.3. Interdire le stockage de données de production sur les environnements de test

Les jeux de données présents dans les bases (SQL ou NOSQL) de test **DOIVENT** être des données fictives ou des données anonymisées (si ces dernières proviennent d'une copie de la base de production).

6.2.4. Interdire le stockage en clair des données de configuration sensibles

Les développeurs **NE DOIVENT PAS** stocker en clair des données de configuration sensibles (par exemple, mot de passe d'un compte technique). Toutes les données de configuration sensibles doivent être chiffrées afin de garantir leur confidentialité. Les secrets devront être variabilisés et externalisés dans un fichier de configuration. Lors du processus de déploiement les variables seront valorisées en fonction de l'environnement ciblé.

6.3. Couvrir le risque d'injection

Source : https://owasp.org/Top10/fr/A03_2021-Injection/

6.3.1. Contrôler les paramètres en entrée

Le composant **DOIT** contrôler et valider tous les paramètres présentés en entrée du composant (format attendu et longueur). En aucun cas ce contrôle est réalisé par le client.

L'application de ces règles limite le risque d'attaque par Injection.

6.3.2. Limiter l'usage des caractères spéciaux

Les attaques par injection reposent principalement sur l'utilisation de caractères spécifiques qui permettent de mettre en commentaire des portions de code et d'insérer du code frauduleux.

Dans la mesure du possible, éviter l'usage des caractères spéciaux ci-dessous :

```
&~"#'{}([]()-|`_\^@\*/.<>,;:!$
```

6.3.3. Couvrir le risque de Cross-Site Scripting (XSS)

6.3.3.1. Vérifier les données saisies par les utilisateurs

Le développeur (DEV) FrontEnd **DOIT** prévoir, avec JavaScript, de vérifier les données saisies par les utilisateurs.

6.3.3.2. Vérifier les données récupérées côté serveur

Le développeur (DEV) **DOIT** prévoir, côté serveur, de vérifier les données récupérées en paramètre par son composant. Il faut rejeter toutes les données qui ne sont pas conformes à ce qui est attendu.

6.3.3.3. Positionner l'attribut de cookie HTTPOnly

Pour éviter le vol de cookie par du code Javascript, l'exploitant (OPS) **DOIT** configurer la plateforme de manière à positionner l'attribut de cookie « HTTPOnly » sur le jeton de session. S'il est présent, le navigateur interdit au moteur JavaScript de lire et d'écrire dans les cookies.

Le développeur (DEV) PEUT également, de manière programmatique, positionner cet attribut, si le cookie applicatif contient des données sensibles et que cela est compatible avec la technologie utilisée.

6.3.3.4. Positionner l'attribut de cookie « Secure »

Pour éviter la transmission du cookie sur un canal non chiffré, l'exploitant (OPS) **DOIT** configurer la plateforme de manière à positionner l'attribut de cookie « Secure » sur le jeton de session. Cet attribut interdit l'envoi du cookie sur un canal non chiffré.

6.3.3.5. Eviter l'utilisation de l'entête « hostname »

L'entête http « hostname » est contrôlée par le client. Aussi, le développeur (DEV) **NE DOIT PAS** se baser sur cet élément dans le cas de traitement critique ou sensible. Les méthodes *ServletRequest.getServerName()* et *HttpServletRequest.getHeader("Host")* sont donc à utiliser uniquement sur des traitements non critiques.

6.3.3.6. Eviter l'utilisation de l'entête « Referer »

L'entête http « Referer » est contrôlée par le client. De plus, cet élément ne sera pas transmis si la requête émane d'une autre source sécurisée.
Aussi, le développeur **NE DOIT PAS** :

- Se baser sur cet élément pour effectuer un contrôle d'accès ;
- Faire de protection CSRF (Cross-Site Request Forgery***)*** sur ce seul élément.

6.4. Couvrir le risque de conception non sécurisé

Source : https://owasp.org/Top10/fr/A04_2021-Insecure_Design/

6.4.1. Utilisez un cycle de vie de développement sécurisé

Tous les projets doivent passer les contrôles SONAR avant une mise en production.

6.5. Couvrir le risque de mauvaise configuration de sécurité

Source : https://owasp.org/Top10/fr/A05_2021-Security_Misconfiguration/

6.5.1. Désactiver les options inutiles

Les exploitants (OPS) **DOIVENT** désactiver les options inutiles des plateformes et infrastructures afin de diminuer le nombre de vulnérabilités potentielles, que ce soit au niveau du système d'exploitation, du système de gestion de bases de données ou du serveur HTTP.

6.5.2. Privilégier l'installation de plateforme en langue anglaise

Les exploitants (OPS) **DOIVENT** autant que possible privilégier l'installation de plateforme en langue anglaise. En effet, lors du développement de correctifs, la version anglaise est **TOUJOURS** privilégiée, les autres versions étant corrigées plus tardivement.

6.5.3. Désactiver/Supprimer les comptes inutiles

Après l'installation, les exploitants (OPS) **DOIVENT** désactiver, voire supprimer, tous les comptes inutiles. Le mot de passe des autres comptes doit être modifié dès que possible. Les comptes d'administration par défaut doivent être verrouillés. Il faut préférer l'utilisation de comptes d'administration créés manuellement.

6.5.4. Assurer le Maintien en Condition de Sécurité

Les exploitants (OPS) **DOIVENT** mettre à jour les différents composants de l'architecture en installant les correctifs dès qu'ils sont publiés.

6.5.5. Couvrir le risque de référence des entités externes (XXE) dans le XML

6.5.5.1. Désactiver les DTD

Le moyen le plus sûr d'empêcher une XXE (XML External Entity) consiste à désactiver complètement les DTD (entités externes).

6.5.5.2. Configurer l'analyseur

Les applications Java utilisant des bibliothèques XML sont particulièrement vulnérables à XXE. Pour utiliser ces analyseurs en toute sécurité, vous devez explicitement désactiver XXE dans l'analyseur que vous utilisez.

6.6. Couvrir le risque d'utilisation de dépendances avec des vulnérabilités connues

Source : https://owasp.org/Top10/fr/A06_2021-Vulnerable_and_Outdated_Components/

6.6.1. Utiliser des dépendances réputées sécurisées

Les développeurs (DEV) ont la responsabilité d'utiliser des dépendances (librairies, framework,…) n'embarquant pas de failles de sécurité majeures connues. A ce titre, les développeurs **DOIVENT** utiliser des dépendances (Framework) validées par la gouvernance de services.

6.6.2. Réaliser une veille régulière sur les dépendances

Il est essentiel d'assurer une veille sur ces dépendances et de procéder à des mises à jour régulières afin de limiter le risque de failles induites par ces librairies externes.

6.7. Couvrir le risque de violation de gestion d'authentification et de session

Source : https://owasp.org/Top10/fr/A07_2021-Identification_and_Authentication_Failures/

6.7.1. Utiliser l'offre sécurité de la branche Famille

Le développeur **DOIT** s'appuyer sur les offres de sécurité présentes dans le catalogue CNAF. Ces offres de sécurité sont homologuées et sont à l'état de l'art des bonnes pratiques « sécurité ». En s'appuyant dessus, le développeur se couvre par rapport à ces risques.
Exemple : utilisation de la librairie JAR ISU (jeton d'authentification).

6.7.2. Utiliser les standards de sécurité

Le développeur **DOIT** utiliser le mécanisme « Authorization Code Flow » pour toute application en frontal d'un utilisateur.

6.8. Couvrir le risque de manque d'intégrité des données et du logiciel

Source : https://owasp.org/Top10/fr/A08_2021-Software_and_Data_Integrity_Failures/

6.8.1. Utiliser des formats de données

Utiliser des formats de données tels que JSON ou XML.

6.8.1. Désérialiser uniquement des données signées

Si l'application sait, avant la désérialisation, quels messages doivent être traités, ceux-ci peuvent être signés, dans le cadre du processus de sérialisation. L'application pourrait alors choisir de ne pas désérialiser un message qui n'a pas de signature authentifiée.

6.9. Couvrir le risque de carence des systèmes de contrôle et de journalisation

Source : https://owasp.org/Top10/fr/A09_2021-Security_Logging_and_Monitoring_Failures/

6.9.1. Produire des évènements applicatifs

Les développeurs (DEV) **DOIVENT** produire des évènements applicatifs conformément aux besoins exprimés dans le document exploitation. Notamment les données accédées et les actions effectuées.

Les développeurs (DEV) **DOIVENT** utiliser les sources de temps fiable configurées par les OPS.

6.9.2. Collecter des évènements système

Les exploitants (OPS) **DOIVENT** collecter les évènements système produits par les plateformes conformément aux besoins exprimés dans le document exploitation.

Les exploitants (OPS) **DOIVENT** paramétrer les systèmes pour assurer un horodatage fiable des évènements.

Exemple : serveur de temps qualifié ou service NTP surpervisé.

6.9.3. Procéder à une supervision continue de ces évènements

Les exploitants (OPS et SOC) **DOIVENT** exploiter les événements produits pour :

- Détecter une éventuelle tentative d'intrusion/attaque sur les plateformes
- Identifier une éventuelle dégradation de performance ou indisponibilité

6.10. Couvrir le risque de falsification des requêtes côté serveur (SSRF)

Source : https://owasp.org/Top10/fr/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

6.10.1. Assainir les données d'entrée fournies par le client

Le développeur (DEV) **DOIT** prévoir, côté serveur, de vérifier les données récupérées en paramètre par son composant. Il faut rejeter toutes les données qui ne sont pas conformes à ce qui est attendu.

6.10.2. Imposer le schéma d'URL, le port et la destination avec une liste positive d'autorisation

Le développeur (DEV) **DOIT** prévoir, côté serveur, une liste explicite de nom DNS ou adresse IP autorisées. Il en sera de même pour les protocoles.
N'atténuez pas les SSRF par l'utilisation d'une liste de refus ou d'une expression régulière.

6.10.3. L'accès illégitime au réseau doit être empêché

Les exploitants (OPS) **DOIVENT** appliquer des politiques de pare-feu ou des règles de contrôle d'accès au réseau "refusant par défaut" afin de bloquer tout le trafic intranet sauf celui qui est essentiel.

7. Matrice de responsabilité

Le tableau ci-dessous décrit les responsabilités de chacun sur les contributions « Sécurité ».

Risques	Règles	Responsabilités	
		DEV	OPS
Manque de contrôle d'accès	Restreindre l'accès aux seuls utilisateurs authentifiés	X	
	Renforcer les permissions basées sur les rôles	X	
Défaillance cryptographique	Chiffrer les données en transit		X
	Stocker les données persistantes dans des bases chiffrées		X
	Interdire le stockage de données de production sur des environnements « Hors production »		X
	Interdire le stockage en clair des données de configuration sensibles	X	
Attaque par injection	Contrôler les paramètres en entrée	X	
	Limiter l'usage des caractères spéciaux	X	
	XSS : Vérifier les données saisies par les utilisateurs	X	
	XSS : Vérifier les données récupérées coté serveur	X	
	XSS : Positionner l'attribut de cookie HTTPOnly	X	X
	XSS : Positionner l'attribut de cookie « Secure »		X
	XSS : Eviter l'utilisation de l'entête « hostname »	X	
	XSS : Eviter l'utilisation de l'entête « Referer »	X	
Conception non sécurisé	Contrôles SONAR	X	
Mauvaise configuration de sécurité	Désactiver les options inutiles		X
	Privilégier l'installation de plateforme en langue anglaise		X
	Désactiver/Supprimer les comptes inutiles		X
	Assurer le Maintien en Condition de Sécurité		X
	XXE : Désactiver les DTD	X	
	XXE : Configurer l'analyseur	X	
Utilisation de dépendances avec des vulnérabilités connues	Utiliser des dépendances réputées sécurisée	X	
	Réaliser une veille régulière sur les dépendances	X	

Violation de gestion d'authentification	Utiliser l'offre sécurité de la branche Famille	X	
	Utiliser les standards de sécurité	X	
Intégrité des données et du logiciel	Utiliser des formats de données	X	
	Désérialiser uniquement des données signées	X	
Carence des systèmes de contrôle et de journalisation	Produire des évènements applicatifs	X	
	Collecter des évènements système		X
	Procéder à une supervision continue de ces évènements		X
Falsification des requêtes côté serveur (SSRF)	Assainir les données d'entrée fournies par le client	X	
	Imposer le schéma d'URL, le port et la destination	X	
	Accès illégitime au réseau		X

8. Annexes

8.1. Glossaire

XSS : Le **Cross-Site Scripting** est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page.

CSRF : Le **Cross-Site Request Forgery** est une attaque dont le but est de transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits.

XXE : **External Entity** est une attaque contre les applications qui analysent et contrôlent les entrées XML.

SSRF : Les **Server Side Request Forgery** sont des vulnérabilités Web permettant de lire des fichiers sur le serveur local et de trouver des services actifs.

RCE : **Remote Code Execution** est la technique qui permet d'exécuter du code sur une machine distante.

DTD : La **Document Type Definition** est, soit un fichier, soit une partie d'un document SGML ou XML, qui décrit ce document ou une classe de documents.

JWT : **JSON Web Token** est un standard ouvert défini dans la RFC 7519.

8.2. URL

L'OWASP : https://www.owasp.org
Le Top Ten 2021 : https://owasp.org/Top10/
Le Top Ten 2017 : https://owasp.org/www-project-top-ten/2017/