THE VIBE CODEUR

LE GUIDE OFFERT

70 FAILLES À VÉRIFIER AVANT DE METTRE EN LIGNE

La checklist complète que je génère dans la masterclass, mise au propre. Sept familles d'attaques, ce qu'il faut vérifier pour chacune, et l'ordre dans lequel procéder.

CAMEL DAGBEGNON

The Vibe Codeur · Dagbegnon Agency · Bénin

masterclass.
dagbegnonagency.com

AVANT DE COMMENCER

Dans la masterclass, je te montre comment obtenir cette liste en deux minutes : tu demandes à ChatGPT de lister toutes les formes d'attaque possibles, tu colles sa réponse dans Claude, et tu lui demandes de vérifier ton projet. La voici au propre, pour que tu n'aies pas à la régénérer.

> **UN ÉCRAN DE CONNEXION NE SUFFIT PAS**
>
> Sécuriser une application, c'est vérifier les comptes, les permissions, les API, les fichiers, le serveur **et** les règles métier. Aucune liste ne couvre absolument tout — celle-ci couvre les familles qui font tomber les applications réelles.

COMMENT S'EN SERVIR

Coche au fur et à mesure. Chaque ligne se vérifie soit en testant toi-même avec deux comptes, soit en demandant à Claude de contrôler ton code sur ce point précis.

1 · Comptes et sessions	**10 points**
2 · Accès aux données d'autrui	**8 points**
3 · Injections et contenu malveillant	**10 points**
4 · Fichiers, serveur, infrastructure	**12 points**
5 · Navigateur, réseau, disponibilité	**11 points**
6 · Fraude et règles métier	**9 points**
7 · Messages, profils, fil d'actualité	**10 points**

> **LES SIX À TRAITER EN PREMIER**
>
> Si tu n'as le temps que pour une chose : **les permissions entre utilisateurs, l'accès administrateur, les secrets exposés, les sessions, les fichiers privés et les paiements.** Ce sont ceux qui coûtent le plus cher quand ils tombent.

1 COMPTES ET SESSIONS

10 POINTS À VÉRIFIER

	LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐	**Force brute** Essais répétés de mots de passe	Limiter les tentatives par compte et par origine, avec des délais progressifs.
☐	**Identifiants volés ailleurs** Credential stuffing	Détecter les connexions automatisées, proposer la double authentification, surveiller les connexions inhabituelles.
☐	**Même mot de passe sur beaucoup de comptes** Password spraying	Détecter les échecs répartis sur plusieurs comptes.
☐	**Énumération des utilisateurs**	Connexion, inscription et récupération ne doivent jamais révéler si une adresse existe.
☐	**Détournement de récupération de compte**	Jetons courts, à usage unique, nombre d'essais limité. Aucune modification sans preuve valide.
☐	**Contournement du code à usage unique**	Expiration, usage unique, limite de tentatives. Protéger aussi les procédures de secours.
☐	**Vol ou fixation de session**	Cookies *HttpOnly*, *Secure*, *SameSite*. Renouveler la session après connexion.
☐	**Session encore active après révocation**	Tester déconnexion, changement de mot de passe et suppression de compte. Permettre de révoquer les appareils.
☐	**Jetons mal vérifiés** JWT	Vérifier signature, algorithme autorisé, expiration, émetteur et destinataire.
☐	**Connexion Google ou Apple détournée**	Vérifier les redirections et la liaison entre comptes.

2 ACCÈS AUX DONNÉES D'AUTRUI

8 POINTS À VÉRIFIER

> **LA PRIORITÉ ABSOLUE**
>
> Un utilisateur connecté ne doit **jamais** pouvoir accéder aux données privées d'un autre en changeant simplement un identifiant dans l'adresse.

LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐ **Accès direct par identifiant** IDOR · BOLA	Avec deux comptes de test, vérifier que A ne peut ni lire ni modifier les objets privés de B.
☐ **Élévation de privilèges**	Un utilisateur normal ne doit jamais devenir administrateur ni appeler ses fonctions.
☐ **Droits vérifiés seulement à l'écran**	Contrôler les droits côté serveur à chaque requête, même quand le bouton est masqué.
☐ **Champs modifiables non filtrés** Mass assignment	Autoriser explicitement les champs modifiables. Rejeter les changements de rôle, de propriétaire ou d'abonnement.
☐ **Fuite entre organisations**	Si plusieurs entreprises utilisent l'application, vérifier leur isolation à chaque requête.
☐ **API trop bavarde**	Ne retourner que les champs nécessaires. Exclure données privées, secrets et attributs internes.
☐ **Stockage de fichiers public par erreur**	Vérifier les droits de lecture et d'écriture, contrôler les accès aux fichiers privés.
☐ **Règles de confidentialité contournées**	Appliquer les restrictions aux recherches, exports, notifications, médias et accès directs.

Ces risques de permissions et d'exposition figurent parmi les risques centraux recensés par l'OWASP API Security Top 10.

3 INJECTIONS ET CONTENU MALVEILLANT

10 POINTS À VÉRIFIER

	LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐	**Injection SQL ou NoSQL**	Requêtes paramétrées, validation des types et des opérateurs autorisés.
☐	**Injection de commandes système**	Ne jamais construire une commande avec une saisie utilisateur. Isoler les traitements.
☐	**Script injecté dans la page** XSS	Encodage adapté au contexte, nettoyage du HTML autorisé, politique de sécurité du contenu.
☐	**Injection dans les modèles** SSTI	Ne jamais traiter une saisie utilisateur comme un modèle exécutable.
☐	**Injection LDAP, XML ou XPath**	Utiliser les interfaces sûres du moteur concerné, désactiver les fonctions inutiles.
☐	**Entités XML externes** XXE	Désactiver les entités externes et les accès réseau des analyseurs XML.
☐	**Désérialisation dangereuse**	Ne pas reconstruire d'objets exécutables à partir de données non fiables.
☐	**Pollution de prototype**	Contrôler les clés et les fusions d'objets, maintenir les bibliothèques à jour.
☐	**Formule injectée dans un export** CSV	Neutraliser les cellules interprétables comme des formules.
☐	**Injection dans les journaux**	Journaux structurés, validation stricte des valeurs et des caractères de contrôle.

4 FICHIERS, SERVEUR, INFRASTRUCTURE

12 POINTS À VÉRIFIER

	LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐	**Fichier malveillant envoyé**	Vérifier le contenu réel, le format et la taille. Empêcher toute exécution.
☐	**Remontée dans l'arborescence** Path traversal	Limiter les chemins aux emplacements autorisés, jamais de chemin libre fourni par l'utilisateur.
☐	**Serveur poussé à contacter l'interne** SSRF	Contrôler destinations, redirections et accès sortants. Bloquer les services internes sensibles.
☐	**Fichiers ou archives démesurés**	Limiter taille décompressée, mémoire, temps de traitement et profondeur d'archives.
☐	**Secrets exposés**	Vérifier code, historique Git, fichiers du frontend, journaux et configuration. Révoquer tout secret divulgué.
☐	**Base de données accessible depuis l'extérieur**	Restreindre le réseau, les comptes et leurs permissions. Supprimer les accès par défaut.
☐	**Mode debug laissé actif**	Désactiver le debug public. Garder les détails techniques dans des journaux protégés.
☐	**Dépendances vulnérables**	Inventorier les composants, analyser leurs vulnérabilités, contrôler les mises à jour.
☐	**Chaîne de déploiement compromise**	Restreindre les droits, protéger les secrets et les branches, examiner les actions de déploiement.
☐	**Sous-domaine abandonné détourné**	Protéger le compte du registrar, supprimer les configurations DNS abandonnées.
☐	**Compte administrateur compromis**	Double authentification, comptes individuels, droits minimaux, traces des opérations sensibles.
☐	**Sauvegardes volées ou détruites**	Sauvegardes isolées et protégées. Tester réellement leur restauration.

5 NAVIGATEUR, RÉSEAU, DISPONIBILITÉ

11 POINTS À VÉRIFIER

LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐ **Action déclenchée à l'insu de l'utilisateur** CSRF	Protection anti-CSRF adaptée, contrôle d'origine, cookies correctement configurés.
☐ **Partage de ressources mal configuré** CORS	N'autoriser que les origines nécessaires, vérifier les requêtes avec identifiants.
☐ **Application affichée dans un cadre piégé** Clickjacking	Restreindre l'intégration avec *frame-ancestors*.
☐ **Redirection ouverte**	Limiter les destinations de redirection autorisées.
☐ **Interception réseau**	HTTPS partout, configuration TLS correcte, aucun contenu sensible en clair.
☐ **Cache exposant des données privées**	Vérifier les règles du CDN et du cache, empêcher le partage de réponses personnalisées.
☐ **Désynchronisation des requêtes** Request smuggling	Maintenir et configurer correctement proxy et serveur, rejeter les requêtes ambiguës.
☐ **Saturation du service** DoS	Protection réseau, quotas, délais maximums, limites de concurrence.
☐ **Requêtes trop coûteuses**	Borner recherches, expressions régulières, tailles de réponse et traitements.
☐ **Abus GraphQL ou WebSocket**	Permissions sur chaque opération, limiter complexité, connexions et fréquence.
☐ **Explosion de facture SMS, e-mail ou IA**	Quotas par utilisateur, plafonds globaux, alertes de coût et mécanisme d'arrêt.

6 FRAUDE ET RÈGLES MÉTIER

9 POINTS À VÉRIFIER

> **CELLES QU'ON MANQUE LE PLUS SOUVENT**
>
> Ces failles échappent presque toujours aux tests automatiques. Elles se trouvent à la main, en essayant de tricher avec sa propre application.

LA FAILLE	CE QU'IL FAUT VÉRIFIER
☐ **Prix ou statut premium modifié**	Calculer prix et droits côté serveur, à partir de données fiables.
☐ **Faux paiement, faux webhook**	Vérifier l'authenticité de l'événement, le montant, la devise et la commande concernée.
☐ **Opération rejouée deux fois**	Une même opération ne doit pas créditer deux fois. Prévoir l'idempotence.
☐ **Actions simultanées** Race condition	Transactions et contraintes en base pour les soldes, quotas et ressources limitées.
☐ **Étapes obligatoires contournées**	Vérifier côté serveur les transitions : validation, paiement, activation, annulation.
☐ **Abus de promotions ou de parrainage**	Limites d'utilisation et détection des comptes coordonnés.
☐ **Création massive de comptes**	Limites d'accès, détection comportementale, protections progressives contre les robots.
☐ **Support manipulé**	Procédure de vérification avant tout changement d'adresse, de numéro ou récupération de compte.
☐ **Abus interne**	Limiter et journaliser l'accès des employés aux données privées.

7 MESSAGES, PROFILS, FIL D'ACTUALITÉ

10 POINTS À VÉRIFIER

À ajouter si ton application contient des conversations, des profils ou un fil — c'est-à-dire la plupart des SaaS grand public.

	LE SCÉNARIO À TESTER
☐	Lire une conversation privée en changeant l'identifiant, ou en rejoignant un canal temps réel.
☐	Envoyer un message après blocage, ou après suppression de la relation.
☐	Consulter une photo privée par une ancienne adresse encore valide.
☐	Déduire une localisation précise à partir des distances affichées.
☐	Récupérer les coordonnées GPS laissées dans les métadonnées des photos.
☐	Collecter massivement les profils par la recherche.
☐	Créer de faux profils, usurper une identité, envoyer du spam.
☐	Manipuler les likes, abonnements, compteurs ou classements.
☐	Lire un message privé dans une notification affichée sur l'écran verrouillé.
☐	Retrouver des données après suppression du compte : stockage, recherche, caches, liens partagés.

SI TU AJOUTES DE L'IA

Teste aussi les injections de consignes, les fuites de données entre utilisateurs, et les actions exécutées par ses outils. **Un modèle ne doit jamais décider seul des permissions d'accès.**

DANS QUEL ORDRE PROCÉDER

Sept étapes, avant la mise en ligne.

1 **Inventorier la surface exposée**

Pages, API, administration, stockage, paiements, tâches automatiques, services externes.

2 **Créer trois comptes de test**

Deux utilisateurs ordinaires et un administrateur. Vérifier chaque permission en lecture, création, modification et suppression.

3 **Analyser code, dépendances et secrets**

Sans oublier la configuration de production.

4 **Tester sur un environnement de préproduction**

Avec des données fictives, et des limites pour les tests de charge.

5 **Tester les règles métier à la main**

Blocages, paiements, opérations simultanées. C'est là que les tests automatiques ne voient rien.

6 **Corriger, puis rejouer chaque test**

Et garder un test de non-régression pour les failles importantes.

7 **Installer les alertes**

Échecs de connexion, refus d'accès répétés, changements de rôle, téléchargements massifs, erreurs serveur, dépenses anormales. Jamais de mot de passe ni de conversation privée dans les journaux.

> **CE QU'UN SCANNER NE FERA PAS**
>
> Aucun outil automatique ne confirmera que ton application est sûre. Les failles qui coûtent le plus cher — permissions entre utilisateurs, règles métier, paiements — se trouvent en essayant de tricher soi-même.

Références : OWASP Top 10, OWASP API Security Top 10, OWASP ASVS et les fiches pratiques OWASP Cheat Sheet Series.

TU SAIS CONSTRUIRE. RESTE À SAVOIR QUOI.

La masterclass te prouve que tu peux sortir une application sans écrire de code. Elle ne te dit pas laquelle mérite ton temps, ni à qui la vendre. C'est ce qu'on travaille ensemble dans le programme.

Voir le programme →
