

## Faibles de sécurité potentiels sur notre projet Geneweb

L'OWASP a développé un certain nombre de ressources qui décrivent les **vulnérabilités les plus courantes** qui existent dans divers systèmes, notamment les **applications Web**, les API, les appareils mobiles, etc. Le plus célèbre d'entre eux est le **Top Ten** de l'OWASP, qui décrit les dix vulnérabilités les plus courantes et les plus impactantes qui apparaissent dans les applications Web de production. Cette liste est mise à jour tous les deux ans sur la base d'une combinaison de données de tests de sécurité et d'enquêtes menées auprès de professionnels du secteur.



**OWASP**

Open Web Application  
Security Project

- #1. **Contrôle d'accès défaillant** => Pas de système d'authentification donc pas concernés ?
- #2. **Défaillances cryptographiques** <=> Protection et sécurité des données, non utilisation du chiffrement lors de l'envoi et du stockage À Vérifier
- #3. **L'injection** => Accès à d'autres données que celles déjà mise par le client ?
- #4. **Une conception peu sûre** => Combien de dépendances ?
- #5. **Mauvaise configuration de la sécurité** => Pas notre cas ici
- #6 **Composants vulnérables et obsolètes** => Lié aux dépendances
- #7. **Défauts d'identification et d'authentification** => Pas notre cas ici
- #8. **Défauts d'intégrité des logiciels et des données** => Faiblesse pipeline devops / process de sécurité => malveillance possible par les mises à jour ou une dépendance altérée
- #9. **Défaillances dans l'enregistrement et la surveillance de la sécurité** => Pas de log de sécurité ? A VÉRIFIER POUR ÊTRE SUR
- #10. **Falsification de requête côté serveur** => Très peu probable en local ?

Cryptographique Failures + Security misconfigurations + Software and Data Integrity Failures

Vulnerable and Outdated Components => Vérifier si c'est le cas

### 1) Cryptographique Failures

Problème: Pas de chiffrement des données, les données personnelles des milliers d'utilisateurs sont stockées et transférées en clair. Stockage + Envoie

Conséquence: Divulcation de données personnelle (contraire aux **RGPD**)

Solution: Norme RGPD, moins de stockage et d'échange possible, meilleur protocole/cryptage, ne pas mettre en cache les informations

## Réflexions

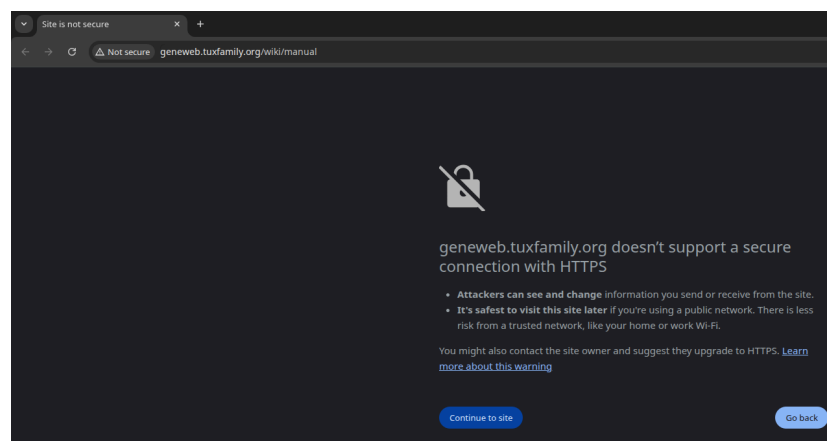
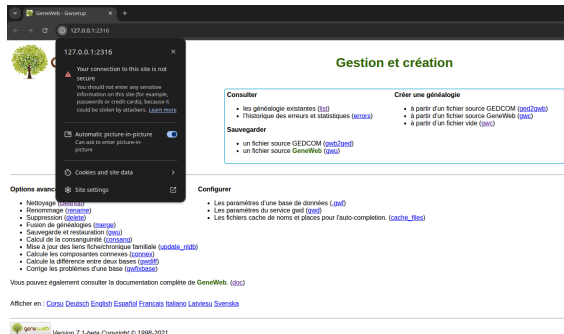
- => Risque de failles de sécurité lors de la **communication** entre le **client** et le **serveur** en local, de plus le site n'est pas en **https**
- => Manipulation de **données personnelles et sensibles** de milliers d'utilisateurs, soumis aux lois **RGPD**
- => Risque **d'injection** ou de mise en place **d'altération** malveillante au moment d'exporter les données ?
- => *Projet Open Source et supervisé mais ce qui n'empêche pas totalement les actes de malveillances (contributions au projet Git déguisé) => Jia Tan*
- => Absence d'authentification ⇔ Il y en a t-il obligatoirement besoin ?
- => Pas de cryptage ⇔ **À Vérifier**
- => Pas de présence de log de sécurité ? ⇔ **A Vérifier**
- => Nécessité d'avoir une authentification si plusieurs utilisateurs utilise la même machine => Donc implique une politique de mots de passe fiables

Source:

<https://kincy.fr/faille-securite/>

<https://digicomp.ch/blognews/2022/11/29/owasp-top-10-failles-securitaires-applications-web>

<https://www.checkpoint.com/fr/cyber-hub/cloud-security/what-is-application-security-appsec/owasp-top-10-vulnerabilities/>



Problème: Injection

Solution: Bien Filtrer les champs que l'utilisateur peut remplir

Technique: Toujours vérifier ce qui vient de l'utilisateur, faire une white list des mots + caractères autorisés ?, des paramètres à ajouter en arrière fond et dans le code de la page web

Problème: Accès à la racine du projet pour le supprimer/modifier/récupérer les informations

Solution: `pathlib.resolve()` ?

Problème: Exécution de code arbitraire lors du chargement des données

Solution: Signer ou vérifié l'intégrité des fichiers importants

Problème: exposition involontaire sur le réseaux, le serveur écoute sur 0.0.0.0 et devient visible sur le LAN

Solution: Bind par défaut sur 127.0.0.1 ; documenter clairement l'option réseau.

### => Protection des données sensibles (PII/GDPR)

**Risque** : fuite d'arbres généalogiques, dates de naissance, adresses, notes privées.

**Chiffrement au repos** (base/fichiers via LUKS/dossier chiffré, ou champs sensibles chiffrés).

Exports **minimisés** (masking des personnes vivantes), **logs sans PII**.

Politique **RGPD** : consentement, droit à l'effacement, rétention limitée, registre de traitements.

### => Dépendances vulnérables / supply chain

**Risque** : libs Python/OCaml compromises ou obsolètes.

CI : **pip-audit**, **bandit**, SCA (Dependabot).

**Pinning** des versions, **hash pinning** (`pip --require-hashes`), revue des changements.

Releases signées; vérifier les checksums.

### => Mauvaises entêtes HTTP & clickjacking

**Risque** : framing malveillant, MIME sniffing, XSS reflet.

Entêtes : **X-Frame-Options: DENY** ou **frame-ancestors 'none'** (dans CSP), **Referrer-Policy: no-referrer**, **X-XSS-Protection: 0** (laisser au CSP), **Strict-Transport-Security** (si HTTPS).

## => DoS / ressources non bornées

**Risque** : fichiers GEDCOM immenses, ZIP bombs, regex catastrophiques.

**Limiter la taille** des uploads, temps CPU/mémoire, quotas par requête.

Parser robustes, **timeouts**, éviter regex  $O(n^3)$ .

Files d'attente/asynchrone pour tâches lourdes.

## => Secrets & clés en clair

**Risque** : clés API/SSH committées ou dans la config.

**GitHub Secrets**, variables d'environnement, **jamais** en dépôt.

Scans de secrets (trufflehog, Gitleaks) en CI.

## => Logs & traçabilité

**Risque** : logs contenant PII ou secrets.

**Rédaction** systématique, niveaux de logs adaptés, rotation/chiffrement.

Journaliser les **actions sensibles** (qui, quoi, quand) pour audit.

## => Packaging & exécution

**Risque** : exécution en root, permissions laxistes.

Paquet/containeur **rootless**, filesystem en **lecture seule**, **AppArmor/SELinux** si possible.

Principe du **moindre privilège** pour fichiers/dossiers.

## => Intégration Python ↔ cœur OCaml

**Risque** : interfaces FFI mal sécurisées, corruption mémoire.

Interfaces **strictement typées**, validation/normalisation des données aux frontières, fuzzing ciblé (AFL/LibFuzzer si applicable).

## => Petites “checklists” d'implémentation rapide

**CI sécurité** : **pytest** + **pytest-cov**, **pip-audit**, **bandit**, scan secrets, lint.

**Headers & CSP** : middleware unique qui fixe CSP, HSTS, nosniff, frame-ancestors.

**Inputs** : Pydantic pour tout ce qui entre, **escape par défaut** côté vues.

**Déploiement local sûr** : bind **127.0.0.1**, user non privilégié, dossiers à permissions minimales, backups chiffrés.

**RGPD** : mode “*anonymiser vivants*” par défaut dans exports/partages + documentation des droits.