Rapport de l’expertise en sécurité et qualité de code Python Flask et Angular 4 sur le projet GeoNature

Kevin Samuel 15/01/18

Résumé

En étudiant le système Geonature 2 dans son état au commit 1e8ed7… du 8 janvier 2018, nous n'avons pas trouvé de sensibilité aux failles courantes impliquant de l'injection de code JavaScript dans le front end, de l'injection SQL dans la base de données ou d'exécution de code Python uploadé sur le backend.

Parmi les attaques courantes, Geonature reste sensible au vol de session, notamment à travers une CSRF. Pour cette raison nous recommandons le passage a un mode d'authentification purement par token, de préférence en utilisant un standard tel que JWT, puisque le projet utilise déjà une API Web.

Néanmoins la recommandation la plus importante d'un point de vue de la sécurité est la mise en œuvre d'une stratégie de déploiement officielle et robuste, contenant notamment :

- La recommandation systématique de la mise en place d'une connexion HTTPS en lieu et place de la connexion en HTTP qui est le standard à l'heure actuelle. Ceci afin d'éviter notamment les vols de session et de données.
- La description d'une configuration d'Apache, Postgres et UsersHub recommandée par l'équipe, particulièrement leurs places dans l'architecture et les réglages importants pour la sécurité de la communication entre ces systèmes (ex: tout mettre sur un serveur, utiliser un VPN, modules à charger, etc.). Prévoir également un listing des dossiers et fichiers, leurs emplacements et permissions recommandés.

Du point de vue de la modularité en revanche, le projet avait besoin d'une restructuration de son fonctionnement. Les composants dans leur état au début de l'audit étaient fortement couplés, la configuration ainsi que les différents points d'entrée partagés en plusieurs points du code et la manière de créer son propre module se faisait à l'intérieur du code source même de Geonature. Nous recommandons plusieurs méthodes pour y remédier dans ce rapport.

À la suite de cet audit, une semaine a été consacrée à la mise en œuvre d'une partie de ces recommandations, avec notamment :

- Une restructuration de l'architecture des modules.
- La centralisation de la configuration.
- La création d'un point d'entrée unique pour le projet.
- L'explication de la mise en place d'un certificat Let's Encrypt pour profiter d'une connexion en HTTPS.
- La revue point par point de la qualité du code Python.
- La mise en place d'outils d'aide à la qualité de code : mypy, pylint, flake8, tslint et pytest.

Ces travaux ont été conduits avec succès, et bien assimilés par l'équipe qui sera en mesure de pousser plus loin, et de manière autonome, ces améliorations ainsi que celles restantes nécessaires.

Index

Introduction

Méthodologie

Une première passe d'analyse a été dédiée à la compréhension de la structure générale du projet, en suivant la hiérarchie des dossiers, le graphe des imports et les listings de dépendances.

À la suite de cela, le processus d'installation a été étudié pas à pas, et lancé à la main plutôt qu'en automatique afin de gagner une meilleure compréhension du système.

Enfin l'analyse détaillée du code a été effectuée depuis les points d'entrée principaux du backend et du front end, puis en suivant tous les appels jusqu'à arriver aux dépendances externes.

En complément, les outils flake8, mccabe, pylint et mypy ont été appliqué aux sources afin de bénéficier de leurs rapports synthétiques.

Points étudiés

Sécurité

Vérification des possibilités d'injection de code JavaScript dans les pages HTML, de bash dans les appels de commande, de Python suite à l'upload d'un fichier et de SQL dans les requêtes à la base de données. Test de la possibilité de procéder à une CRSF.

Évaluation des chances de vol de mot de passe ou d'identifiant de session.

Modularité

Analyse du système de configuration. Étude de l'indépendance et la ré-utilisabilité des composants et backend, s'ils existent, ainsi que de l'extensibilité générale. Recherche systématique des points d'entrée du programme.

Qualité du code

Le code python est-il idiomatique ? Est-il facile à lire ? Suit-il le PEP8 ? Suit-il les bonnes pratiques stylistiques (DRY, YAGNI, EAFP, etc) et architecturales (composition, isolation des responsabilités…) ?

Autre

Bien que ce ne soit pas le sujet principal de l'audit, quelques remarques concerneront l'ergonomie de l'UX et l'aspect légal.

Points forts

Potentiel limité d’injection de code JavaScript, SQL et Python

De par les outils utilisés, Geonature est peu sensible aux injections de code. En effet, l’utilisation du templating côté client fourni par Angular2 avec l’auto-escape activé désamorce la majorité des attaques d’injection JS connues. Nous n’avons pas vu de code où ces mécanismes d’échappement sont pour le moment désactivés.

Le code Python lui-même n’est pas chargé automatiquement à l’activation d’un chemin puisqu’il utilise la norme WSGI. Une tentative pourrait être faite à travers la fonctionnalité d’upload, mais la fonction qui gère ce travail sanitize le nom de fichier avant de l’utiliser.

Quant aux accès à la base de données, ils passent majoritairement par un ORM qui se charge de nettoyer les données entrantes automatiquement. Il existe plusieurs endroits où des requêtes SQL écrites à la main sont utilisées, et les paramètres sont correctement passés en évitant la concaténation de chaîne de caractères.

Droit de consultation

L’application utilise un système de droit d’accès calqué sur celui de leur service d’authentification, UsersHub, qui attribue à chaque couple utilisateur / application un niveau de 1 à 6 représentant ses privilèges d’accès. Les comptes utilisateurs sont également attachés aux données qu’ils produisent.

Ces privilèges et permissions sont vérifiés à la consultation des relevés.

Lazy loading du routing frontend

Angular fournit la possibilité de charger certaines parties de l’application uniquement lorsque l'on en a besoin. Le fait de ne pas utiliser cette fonctionnalité peut rendre le chargement initial du site inutilement lourd. Le fait d’utiliser trop cette fonctionnalité peut rendre le chargement de toute nouvelle page inutilement lent.

Dans Geonature, le choix des URLs à charger a été pensé pour un chargement rapide de la page d’accueil, et un lazy loading des modules du site.

Polyfills minimalistes

La liste des polyfills utilisés n’a pas été modifiée. Les paramètres par défaut visent les « ever green browsers », ce qui est la cible de Geonature, assurant ainsi de ne pas surcharger le bundle JavaScript produit par Webpack.

Utilisation des modules Angular les plus à jour

Bien qu’Angular 4 soit compatible avec Angular 2, Geonature exploite les fonctionnalités de la version 4 plutôt que de la 2. Ex : utilisation de HttpClient.

Gestion des utilisateurs et permissions standardisée en microservice

L'authentification et les permissions sont déléguées à UsersHub, un service séparé, et ne font pas partie du code du code de Geonature. Ce qui fournit des avantages en termes de découplage et de ré utilisabilité.

Indice de complexité cyclomatique faible

Le passage de l'outil mccabe sur le code relève que seules 14 fonctions ou méthodes dépassent l'indice 5 de complexité cyclomatique. Cela signifie qu'en moyenne, ces callables ont peu de chemins d'exécution linéairement indépendants et sont donc plus faciles à lire et à débugger.

Sécurité

Éviter le vol de session

Le vol d'une session utilisateur est actuellement possible sur Geonature. Les identifiants sont transmis en clair lors de la connexion, ce qui est le problème principal, étant donné que les utilisateurs ont tendance à réutiliser les mêmes mots de passe d'un service à un autre. Le token de session est également transmis à clair à chaque requête via un cookie.

Plusieurs mesures peuvent être prises pour mitiger ce risque.

D'abord, proposer par défaut l'utilisation du protocole HTTPS, de préférence avec une redirection automatique de toute URL HTTP vers son équivalent HTTPS. Le chiffrement des requêtes est un moyen simple et peu coûteux de grandement augmenter la sécurité du système. En développement, générer son certificat avec OpenSSL ne prend que quelques minutes. En production, l'existence du service Let's Encrypt permet aujourd'hui de profiter facilement d'un certificat fiable et gratuit.

Si pour quelques que raison que ce soit l'option d'utiliser HTTP doit être optionnellement conservée, il faut mettre en avant et décrire la mise en œuvre de HTTPS en priorité la dans configuration recommandée (voir plus bas).

Une mesure de sécurité additionnelle peut être de lier le token de session au certificat du client suite à la connexion HTTPS. Ainsi, même en cas de vol de token, il n'est pas possible d'utiliser celui-ci en dehors du navigateur. Néanmoins, les données de Geonature n'étant pas de nature critique, la priorité d'une telle implémentation est basse.

En revanche, Geonature utilise une Web API, mais continue de faire l'usage de cookies pour transmettre le token de session. Pour cette raison, le système est sensible aux attaques de Cross Site Request Forgery. Même si il apparaît peu probable qu'un attaquant soit intéressé par les données elles-mêmes du site, une CSRF est si peu coûteuse à mettre en place que s'en protéger n'est pas à négliger.

La manière simple d'y remédier est d'une part de s'assurer que les requêtes suivent le standard HTTP et que GET ne déclenche pas d'effet de bord (ex : pas d'écriture en base de données, pas de log out, etc.). Ensuite, la mise en place d'un token à usage unique pour les requêtes POST permet de

mitiger le problème. Les autres verbes (PUT, DELETE, etc.), ne sont pas concernés puis qu'impossible à envoyer sans JavaScript.

Cependant, une meilleure pratique, plus générale, serait de remplacer l'usage du cookie par celui d'un autre header, avec un token d'identification standard tel que JWT. L'architecture gagnerait en généricité et la faille de sécurité disparaîtrait par nature. Pour se faire, créer un backend aiderait non seulement à la migration, mais également à l'amélioration de la modularité du projet (voir partie suivante).

Dans la continuation de ces pratiques, il faut rendre configurables les noms de domaines acceptés pour les requêtes CORS. Puisque Geonature, TaxHub et UsersHub sont organisés sous forme de services indépendants, il est bon de limiter les requêtes inter-sites aux noms de domaine de ces instances. L'utilisation de « * » permet à n'importe quel site avec JavaScript d'utiliser la session d'une personne qui chargerait une page malveillante.

Détecter les intrusions

Une instance Geonature doit permettre la détection des erreurs, qui peuvent être liées à un bug ou une tentative d'intrusion. Pour cette raison, il est important de logger les erreurs 404 et 500, et fournir un rapport de mail sur celles-ci.

Parallèlement, il est possible mettre un throttling sur l'API, particulièrement l'authentification, et la connexion SHT, accompagné d'une politique de bannissement et de logs adéquats est un plus. À défaut de le coder dans le cœur de Geonature, ou en complément, il est possible d'utiliser des outils tels que fail2ban.

Description d'une configuration recommandée

Même si nous n'avons pas trouvé de faille de sécurité majeure dans le code de Geonature, si le système est incorrectement mis en œuvre, il exposera par nature une surface d'attaque plus large. S'il est coûteux et difficile de mettre en place une instance sécurisée du produit, des raccourcis seront pris par ceux qui l'installeront.

Pour diminuer la probabilité d'une telle occurrence, il convient d'écrire une procédure complète d'installation qui contient des instructions claires sur les actions recommandées au déploiement de Geonature.

Cette procédure devra notamment contenir :

- L'utilisation de HTTPS.
- Les permissions à attribuer aux fichiers.
- La gestion des mots de passe, comptes utilisateurs et clés SSH.
- L'installation ou non de toutes les pièces amovibles (apache, usershub, taxhub, postgres, etc.) sur le même serveur.

- Dans le cas de l'installation sur plusieurs serveurs, comment sécuriser la communication entre ces pièces (ex : authentification, VPN, tunnel SHH…).
- La configuration à donner aux outils de protections généralistes (iptable, fail2ban, etc) pour éviter les scans, bruteforces et autres tentatives de pénétrer le serveur.
- Politique de backup et de restauration.

Modularité

Intégration d'un module

La manière de développer et intégrer un module pour étendre les fonctionnalités du système implique un couplage fort entre le nouveau code et celui de Geonature.

Dans le but de diminuer ce couplage, il est recommandé, sans ordre précis, de :

- Donner un nom au concept de modules. « module » est un terme qui est très chargé et ne permet pas de communiquer efficacement. Ex : appeler ces modules des « géomodules » ou « gn_module ». Utiliser ce terme pour namespacer toute référence à un module avec de clairement identifier tout package Python qui serait un module.
- Centraliser le système de configuration. Actuellement la configuration de Geonature est répartie dans de nombreux fichiers. Par ailleurs ces fichiers doivent résider à un endroit précis, dans l'arborescence du code de Geonature. Il faut grouper ces fichiers de configuration en un seul point d'entrée dans un format pratique à éditer pour l'utilisateur, et incluant un système de validation. On doit pouvoir écraser la configuration d'un module par ce point d'entrée. On doit pouvoir configurer ce point d'entrée pour le changer à la volée.
- Centraliser le lancement des actions liées à Geonature (lancer un serveur, builder, etc.) en un seul point d'entrée. Ex : une commande « geonature » acceptant des sous-commandes.
- Décrire un format pour les modules (arborescence, points d'entrée python, JS, installation, désinstallation, etc.). Versionner ce format, et permettre de donner un degré de compatibilité de version.
- Définir un mécanisme d'activation et de désactivation d'un module.
- Définir la partie du code de Geonature utilisable par un module (ex : API publique, widgets réutilisables) et les hooks disponibles (ex : routing URLS).

Extension des fonctionnalités existantes

Le cœur des fonctionnalités de Geonature est pour le moment immuable. Afin de gagner en modularité, il faut définir des parties de Geonature qui sont extensibles. Ex : comment changer le template de base sans toucher au code source ? Comment replacer la méthode d'authentification depuis un module externe ?

Cette extensibilité se fait à la fois en autorisant des possibilités d'héritage et de composition des fonctionnalités existantes, en définissant des backend remplaçables et leurs interfaces, et en mettant à disposition des hooks (ex : dans le fichier de configuration) pour les activer.

Qualité du code:

Utilisation d'outils d'aide à la qualité du code

Afin d'automatiser l'aspect stylistique et limiter l'introduction de bugs triviaux, il est intéressant que l'équipe du projet adopte des outils d'aide à la qualité du code tels que des linters (flake8, pylint, mypy, maccabe) et test runners (pytest). Pour ne pas lancer ces outils à la main, l'intégration dans les IDEs et via des hooks git (en duo avec tox) peut encore faire gagner en productivité.

Proposer un .gitignore plus complet aiderait également à éviter d'ajouter par erreur des fichiers inutiles dans le repository du projet.

Workflow git

La version 2 du projet Geonature est créée en utilisant une architecture et des technologies complètement différentes de la version 1, si bien qu'il n'y a aucun point commun entre les deux base de code. Pourtant le même repository Git a été utilisé, ainsi que le même gestionnaire de ticket.

De surcroît, tous les développeurs travaillent sur une même branche, perpétuellement instable.

À la suite de cet audit, il est donc recommandé de créer un nouveau projet dédié à Geonature et d'instaurer un système de collaboration basé sur l'usage de plusieurs branches Git ayant une convention officielle.

Gestion des erreurs

Après lecture du code, il s'avère que plusieurs extraits utilisent des gestions d'erreur avec « try / except » sans préciser le type d'exception à attraper, ce qui implique que toutes les exceptions sont capturées. Ce comportement peut facilement masquer un bug dans le futur, et il est donc recommandé de les transformer afin qu'ils précisent tous exactement l'exception qu'ils gèrent. Des fonctions sensibles comme « remove_file » sont la priorité pour ce changement.

Similairement, les appels AJAX retournent des promises auxquels aucun handler pour les erreurs n'est actuellement attaché, ce qui pourrait également master un bug dans le futur. Il faut donc passer un callback pour chaque appel AJAX pour gérer les erreurs, afin d'alerter l'utilisateur d'une erreur, mais aussi de permettre la remontée du problème.

Détails repérés dans le code source

Plusieurs détails non critiques ont été notés lors du parcours du code. Les corriger amènera a une amélioration générale de la qualité de la base de code.

- Mettre des fichiers « __init__.py » dans les dossiers pour les compter comme des packages et non des namespaces.

- Mettre des docstrings, au moins dans les fonctions à l'usage non évident. Exemple : « insert_in_cor_role(id_group, id_user) ».
- Utiliser Pipfile et un lock file pour locker les dépendances et inclure les dépendances de développement. La conservation d'un fichier « requirements.txt » est néanmoins recommandée pour des raisons de compatibilité.
- Retirer les imports inutilisés, les fichiers vides, et les restes de code Python 2 (ex : « # coding » et « import __future__ »).
- Créer un décorateur de rollback automatique des sessions SQLAlchemy pour gérer les exceptions dans les vues plutôt que de la faire manuellement à chaque fois.
- Créer un helper pour récupérer un objet ou renvoyer une réponse HTTP 404. C'est un cas suffisamment courant pour ne pas le faire à la main.
- Retirer les prints et les remplacer par l'usage du module logging.
- Gérer les valeurs aberrantes en entrée pour limiter le nombre de possibilité d'erreurs 500 et problèmes de performances. Ex : mettre un « try » et une limite de taille et interdire les valeurs négatives sur des codes tels que « int(parameters.get('limit')) if parameters.get('limit') else 100 » ou « q.limit(limit).offset(page*limit).all() ».
- Ne pas retourner de code ou de message d'erreur, mais plutôt lever une exception, ce qui est le style de gestion d'erreur idiomatique en Python. Par exemple dans la fonction « get_releve_if_allowed ».
- Utiliser les fonctions écrites en code natif de Cpython au lieu de procéder au traitement manuel. Ex : utiliser « list.extend » plutôt que « for o in observers ».
- Réécrire les fonctions de module « filemanager » afin d'améliorer leur qualité :
    - inclure des listes blanches de noms de dossier dans lesquelles elles ont le droit de s'exécuter, **même si ce travail est déjà fait en amont.**
    - Ajouter les tests unitaires.
    - Pour « removeDisallowedFilenameChars », retirer le code redondant que fait déjà « secure_filename », autoriser les caractères chinois (par exemple à l'aide de la lib « unidecode »), mettre un nombre de lettres minimales complétées par un hash du nom original si les lettres manquent car « secure_filename » peut retourner une chaîne vide.
    - Vérifier que le paramètre « filename » ne contient pas de « / » **même si ce travail est fait en amont** (ici par « removeDisallowedFilenameChars »).
- Restructurer petit à petit le code pour tendre vers plus de respect du PEP8. Ex : convertir les noms de camelCase à snake_case.
- Retravailler le code de « testDataType » pour utiliser un mapping plutôt qu'une série de « if ».

- Transformer « serializableModel » pour être appliquée via une métaclasse ou un décorateur afin de ne l'exécuter qu'une seule fois à l'initialisation du programme plutôt que pour chaque appel API.
- Échaper « "%s?ticket=%s&service=%s"%(configCas['URL_VALIDATION'], params['ticket'], base_url) » pour le cas où le service utilisé ne procède pas à l'échappement des données lui-même.
- Choisir un standard de notation pour les sélecteurs SCSS. Documenter les variables.

Aspect légal

L'audit révèle trois points de mise en conformité légale à adresser :

- Déclaration à la CNIL : si la base de données contient des données nominatives, cette déclaration est obligatoire.
- Fichier de LICENCE : Geonature doit déclarer sa licence et en fournir une copie.
- Dossier de licence des dépendances : puisque le logiciel est sous licence libre et distribué, il faut vérifier la liste des dépendances qui stipulent que la licence lors de la distribution doit être donnée et les auteurs mentionnées. Cela ne concerne pas les dépendances à installer, seulement celles distribuées avec le code source. Tous les fichiers du frontend sont notamment concernés puisqu'ils sont distribués automatiquement aux clients.

Ergonomie

Bien que ce ne soit pas l'objet de l'audit, des corrections de l'ergonomie générale de l'interface ont été notées, et sont donc portées à l'attention de l'équipe :

- Le menu pour changer la langue ne doit pas changer de langue lui-même, sans quoi il est difficile de retrouver une langue familière si on se retrouve par erreur avec l'interface dans une langue très différente.
- Les URLs utilisent un mode où la route est précisée après le '#', ce qui pose tout un tas de problèmes de navigation, de partage de liens et de logging. Il vaut mieux utiliser des URLs traditionnelles, ce que supporte Angular à condition d'ajouter un peu de code côté backend.
- L'ouverture dans un nouvel onglet des éléments du menu principal n'est actuellement pas possible, ce qui limite la productivité d'un outil destiné à la saisie. Il faut s'assurer notamment que le clic milieu, le clic droit + « ouvrir dans un nouvel onglet » et le Ctrl + clic soient disponibles.

Ces problèmes ont été détectés suite à un examen superficiel et il donc probable qu'une étude plus approfondie en mettrait en lumière d'autres. Une étude ergonomique impliquant l'observation d'utilisateurs utilisant le logiciel est donc recommandée pour aller dans ce sens.

Documentation

La documentation fait partie intégrante de ce qui fait la qualité d’un projet, et Geonature possède déjà de la littérature utile, particulièrement pour l’installation. Elle pourrait être étendue en rajoutant :

- Un graphique montrant le modèle général de données.
- Une liste des endpoints de l'API Web, incluant le versionnage, les verbes, paramètres et valeurs de retour attendues.

  Un schéma général de l'organisation des différents éléments de la stack déployée : Apache, Postgress, Angular, Flask, etc.
- Un schéma général de l'organisation des différents packages de la partie front end et backend.
- Une note précisant comment remonter les bugs et failles de sécurité.

Mise en œuvre durant la seconde partie de la prestation

À la suite de l’audit, l’équipe a commencé à mettre en place certaines mesures. Puis nous avons procédé à une semaine de sprint pour en implémenter d’autres. À la fin des deux semaines de prestations, les actions suivantes ont été mises en œuvres :

- Configuration HTTPS.
- Remplacement d’un certain nombre de try / except et des rollbacks.
- Centralisation des points d’entrée et de la configuration.
- Définition de la structure d’un module, de son mécanisme d’intégration et d’activation.
- Installation d’outils d’amélioration de la qualité du code.