

Rapport Summer Student - Liam Kirsh - SIS/OA

INSPIRE-HEP est une archive ouverte de la physique des particules chapeauté du CERN entre autres organisations. Cette archive est utilisée par plus de 50,000 chercheurs et étudiants à travers le monde. Elle comprend aussi des profils professionnels et académiques de la communauté et de l'information sur des conférences et des établissements.

Hyper Articles en Ligne (HAL) est une archive ouverte pluridisciplinaire française qui contient des articles scientifiques et des thèses émanant des établissements d'enseignement et de recherche et des laboratoires.

Mon premier projet au CERN a consisté à créer un module qui convertit les notices dans la base de données INSPIRE au format supporté par HAL. Les notices de INSPIRE sont entreposés dans PostgreSQL, le système de gestion de base de données et peuvent être extraites en format JSON. L'API SWORD de HAL accepte des dépôts dans un format basé sur le format TEI. Pour faire la conversion des métadonnées, j'ai utilisé deux modules : Jinja2, un moteur de templates pour Python et Name Parser, un module pour diviser des noms humains en pièces séparées.

En faisant ce projet primaire, outre le guide de style de Python PEP8, j'ai appris quelques principes généraux d'informatique. Par exemple, j'ai appris la différence entre les tests d'intégration et les tests unitaires et comment créer chacun. De plus, mon superviseur m'a enseigné le principe Python « Mieux vaut demander pardon que permission. » Afin d'éviter d'utiliser trop d'expressions conditionnelles, j'ai mis mes énoncés dans des blocs de traitement d'erreur *try-except* et j'ai écrit une fonction qui s'assure que les exceptions soient levées.

Après, j'ai commencé la deuxième partie du projet, la fonctionnalité qui permet la soumission des notices à HAL. J'ai testé et appris comment utiliser le module `python-client-sword2`. Ce module construit la requête qui communique à HAL les métadonnées et le document. Comme la première requête a été refusée, il me fallait inspecter le paquet utilisant le logiciel Wireshark et le comparer à l'exemple dans la documentation HAL. J'ai trouvé la différence et j'ai modifié les paramètres pour que la requête soit acceptée. J'ai découvert un bug dans le client SWORD2 aussi. Les spécifications SWORD2 disent que le serveur peut répondre avec le code 200 ou 202, mais le client interprète la réponse 202 comme une erreur. Ainsi, j'ai généré une demande de tirage (*pull request*) pour faire la modification.

Comme un projet secondaire, j'ai fait des tests d'intrusion sur le nouveau site web pour INSPIRE, INSPIRE-Nightly. Par conséquent j'ai trouvé plusieurs vulnérabilités concernant l'authentification, l'autorisation et les en-têtes HTTP de sécurité. J'ai soumis deux rapports de bug à *invenio-accounts*, le module utilisé pour identifier les utilisateurs de INSPIRE-Nightly, au sujet de la sécurité d'authentification. Le premier a permis une attaque par fixation de session.

Le module Flask-KVSession a été intégré incorrectement : la fonction *session.regenerate* n'était pas appelée quand l'utilisateur s'identifie. Cette fonction change l'identifiant de session qui est stocké dans un cookie suite à l'authentification. Si l'identifiant n'est pas changé, un adversaire pourrait envoyer à un utilisateur un lien qui met l'identifiant avant de l'authentification et quand l'utilisateur s'identifie, cet identifiant de session lui permettrait de détourner la session et s'identifier comme la victime.

L'autre vulnérabilité était un réglage manquant de configuration. Parce que la valeur par défaut du paramètre de Flask (le framework web) « *SESSION_COOKIE_SECURE* » est *Faux*, quand *invenio-accounts* est servi au moyen d'une connexion sécurisée, comme il se doit, l'indicateur *secure* n'est pas activé dans le cookie de session. C'est ainsi que le cookie puisse être divulgué et intercepté au moyen d'une connexion non sécurisée lors d'une *attaque de l'homme du milieu* (man-in-the-middle attack). La solution qu'on a trouvée a été de consigner un message d'avertissement dans le cas où l'application est exécutée dans un environnement de production sans ce paramètre activé. De cette façon, le site continuera à marcher lorsque le paramètre n'est pas activé. J'ai soumis une demande de tirage qui a été acceptée pour faire cette modification.

En analysant le site à la main, j'ai trouvé certaines parties du site auxquelles quelqu'un qui connaissait bien la structure du site pourrait accéder sans autorisation. On l'a noté pour que ce soit réparé avant la diffusion publique du site.

Enfin, j'ai utilisé un scanner de sécurité pour analyser plus le site web et j'ai découvert quelques en-têtes HTTP de sécurité manquants. J'ai soumis un rapport sur GitHub pour documenter chacun. Ce sont *X-Frame-Options*, *X-XSS-Protection*, et *X-Content-Type-Options*. *X-Frame-Options* est un en-tête qui empêche le détournement de clic (*clickjacking*). Sans cet en-tête, la page peut être rendue dans un cadre en ligne (*IFrame*). Cela peut permettre un adversaire à inciter la victime à exécuter une action à son insu. Ensuite, *X-XSS-Protection* active la protection contre le *cross-site scripting* dans plusieurs navigateurs. Dans ce cas, si quelqu'un met du code sur le site qui volait le cookie de session de l'utilisateur, le navigateur protégerait l'utilisateur. Troisièmement, l'en-tête *X-Content-Type-Options: no-sniff* s'assure que le navigateur n'interprète pas les fichiers autrement que comme le type déclaré (basé sur le contenu).