



LOG8100 — DEVSECOPS

AUTOMNE 2024

**Laboratoire 1: Analyse Statique de
Vulnérabilité**

Alek Delisle - 2024651
Bryan Junior Ngatshou – 1956611
Nina Cussac - 2311536
Martin Hervé Mouya - 1891541

Soumis à : Antoine Boucher

26 septembre 2024

Introduction

Avec la multiplication des menaces numériques, assurer la sécurité des applications est devenu une priorité essentielle non seulement pour les organisations, mais aussi pour les développeurs. C'est la raison pour laquelle il est important d'effectuer une analyse statique des vulnérabilités en examinant le code source afin d'y identifier des failles potentielles.

Dans le cadre de ce travail, l'analyse est effectuée sur le code source d'une application sélectionnée du site de OWASP (Open Web Application Security Project), une organisation internationale à but non lucratif qui se consacre à la sécurité des applications web.

L'application sélectionnée est PyGoat : il s'agit d'une application web intentionnellement vulnérable à la sécurité dans Django. Les vulnérabilités peuvent être basées sur le Top 10 de l'OWASP. Elle est à la fois "Online" et "Offline".

Pour nous aider dans cette analyse, nous avons utilisé SonarLint, une extension IDE qui aide à détecter et à corriger les problèmes de qualité lorsque l'on écrit du code et SonarCloud, un service cloud de code propre (en termes de qualité et de sécurité du code).

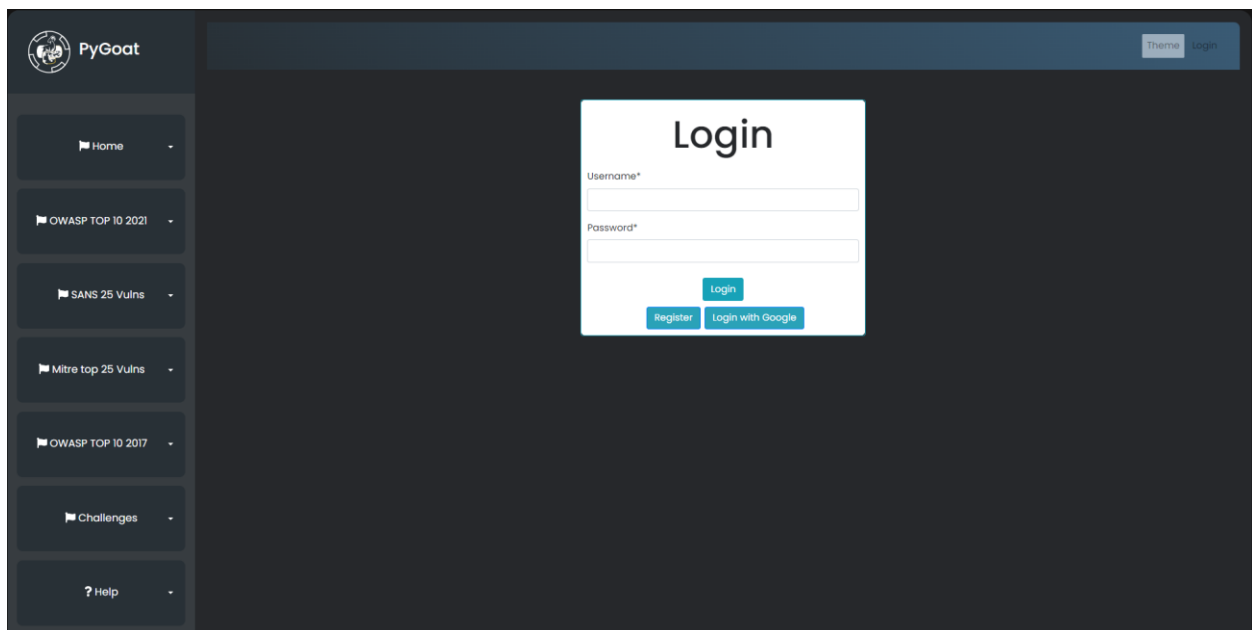


Figure 1. Page principale de l'application "PyGoat"

Description du projet

Pour ce travail, nous avons utilisé un répertoire git privé dans lequel nous avons cloné le code source de l'application étudiée. Par la suite nous avons trouvé nécessaire de n'avoir qu'une branche « main » pour répertorier une version complète provisoire du code et des branches « features/* », dans lesquelles nous sauvegardons les différentes solutions pour les vulnérabilités corrigées. Pour pouvoir organiser de manière optimale et efficace la distribution du travail pour ce TP1 et les prochains TP et projet, nous avons également créé un tableau Kanban composé de 5 colonnes :

- La colonne *Backlog* pour numériser les tâches(solutions)
- La colonne *À faire* effectuer juste celles qui sont le plus urgentes
- La colonne *En cours* dans laquelle se trouve les tâches en cours d'implémentation
- La colonne *Revue* pour les tâches dont l'implémentation est terminée mais doivent être vérifiées par les pairs
- La colonne *Terminée* pour les tâches vérifiés et approuvés par les pairs



Figure 2. Kanban utilisé par l'équipe pour diviser les tâches du tp

De plus, deux “workflows” sont utilisés pour automatiser le processus. Le workflow *Item added to project* sert à ajouter les tickets créés directement dans la colonne *Backlog*, et le workflow *Item closed* permet d'ajouter automatiquement les tickets terminés dans la colonne *Terminée*.

Résultats de l'analyse de l'application « PyGoat »

L'analyse de l'application nous a permis d'identifier de nombreux hotspots et vulnérabilités avec les outils SonarLint et SonarCloud. SonarCloud nous a révélé que l'application a de majeures vulnérabilités, ce qui lui attribue une note de E pour la sécurité et la fiabilité de l'application.

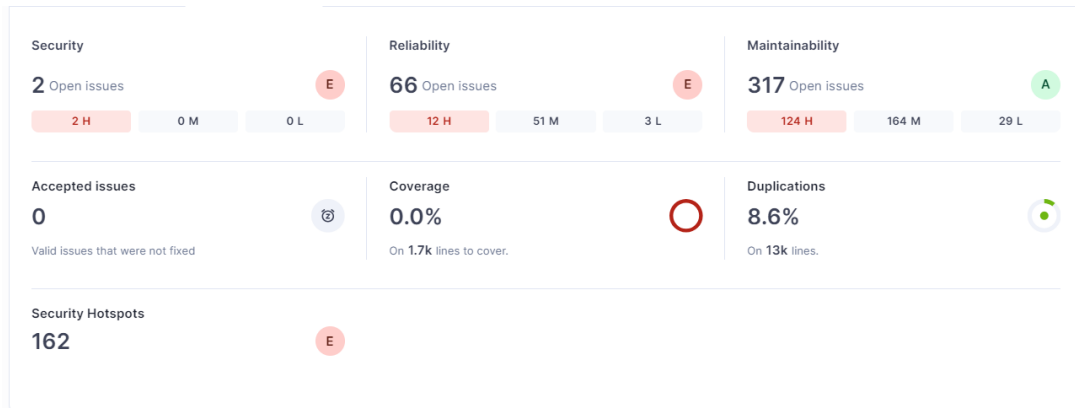


Figure 3. Résultats de l'analyse de sécurité de l'application PyGoat par SonarQube

Nous allons donc identifier et proposer des solutions pour les vulnérabilités les plus problématiques de l'application.

L'application comporte trois types de zones à risque plus importantes dans le code (hotspots):

1. Désactivation de la protection CSRF

La protection CSRF (Cross-Site Request Forgery) empêche les attaquants d'envoyer des requêtes malveillantes à un serveur en utilisant les privilèges d'un utilisateur authentifié sans son consentement. Désactiver cette protection rend les utilisateurs de l'application « PyGoat » vulnérables, car des requêtes peuvent être envoyées en leur nom, avec leurs identifiants, sans qu'ils ne les aient initiées eux-mêmes.

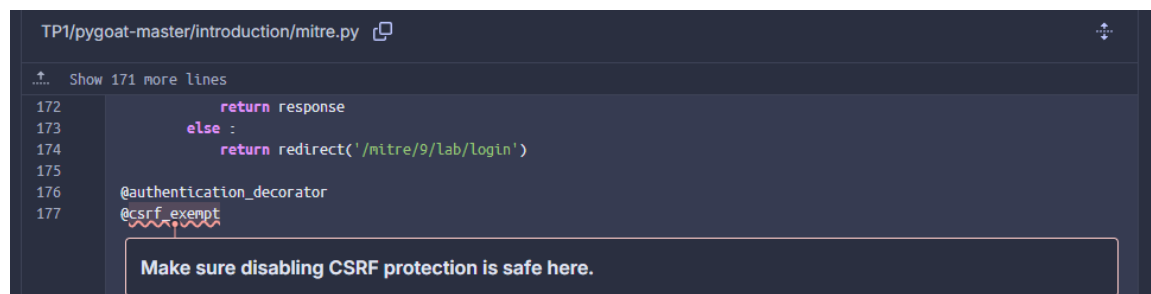


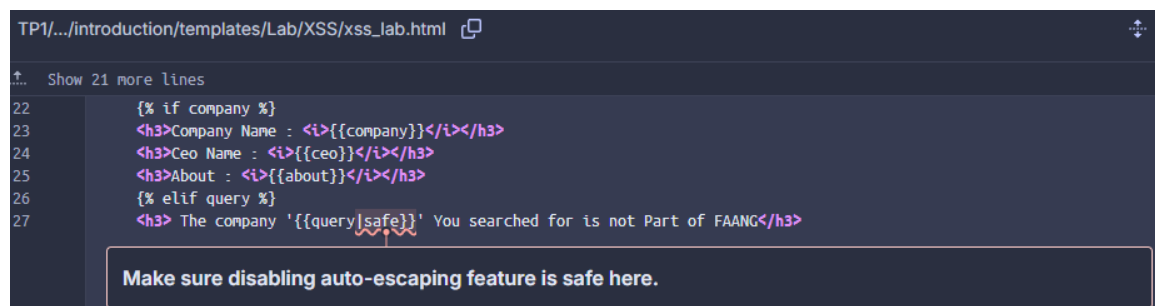
Figure 4. Partie du code comportent le hotspot relié au CSRF

Cette zone du code à risque correspond à un CWE-352. Ce type de vulnérabilité est décrite (en anglais) comme suit: "The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request."¹

2. Désactivation de l'*auto-escaping*

Cette zone du code à risque correspond à une vulnérabilité de type CWE-79. Elle est décrite (en anglais) comme suit: "The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users."² Cette vulnérabilité fait également partie du top 10 de l'OWASP.³

La fonctionnalité *auto-escaping*, lorsqu'elle est activée, permet de prévenir les attaques de type XSS (Cross-Site Scripting). En effet, l'*auto-escaping* sécurise le HTML en convertissant automatiquement les caractères spéciaux en entités HTML, ce qui empêche l'interprétation du code comme HTML ou JavaScript. Sans cette protection, les attaquants pourraient injecter du code HTML ou JavaScript malveillant dans l'application, modifiant ainsi le contenu affiché à l'utilisateur. Par exemple, ils pourraient créer des liens malveillants ou modifier le contenu de la page pour exposer des informations sensibles sur les données de l'utilisateur.



```
TP1/.../introduction/templates/Lab/XSS/xss_lab.html
+ Show 21 more lines
22     {% if company %}
23     <h3>Company Name : <i>{{company}}</i></h3>
24     <h3>CEO Name : <i>{{ceo}}</i></h3>
25     <h3>About : <i>{{about}}</i></h3>
26     {% elif query %}
27     <h3> The company '{{query|safe}}' You searched for is not Part of FAANG</h3>

Make sure disabling auto-escaping feature is safe here.
```

Figure 5. Partie du code où l'*auto-escaping* est désactivé

Dans les deux cas, on retrouve ces zones à risque à plusieurs endroits dans le code. La solution pour éviter ces risques serait simplement de réactiver les protections désactivées, ce qui rendrait l'application moins sujette aux attaques.

¹ <https://cwe.mitre.org/data/definitions/352.html>

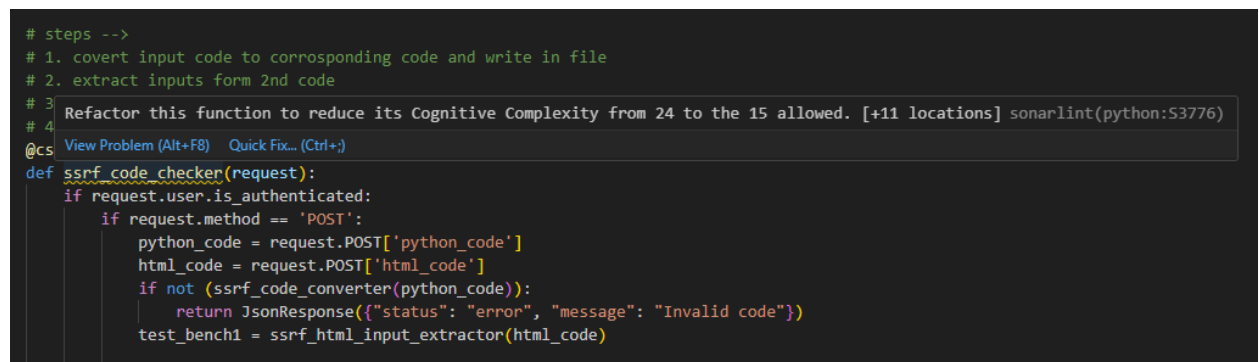
² <https://cwe.mitre.org/data/definitions/79.html>

³ <https://www.cloudflare.com/fr-fr/learning/security/threats/owasp-top-10/>

3. Refactorisation d'une fonction pour réduire sa complexité cognitive de 24 à 15, comme autorisé.

La complexité cognitive d'une fonction mesure sa difficulté à comprendre (nombre de branches conditionnelles, boucles, etc.). Une fonction trop complexe est plus difficile à tester, à maintenir, et à corriger. Cette zone de code à risque correspond à une vulnérabilité de type **CWE-710**, elle est décrite comme : The product does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

Une possibilité de correction serait de découper la fonction en plusieurs sous-fonctions plus petites et mieux définies.

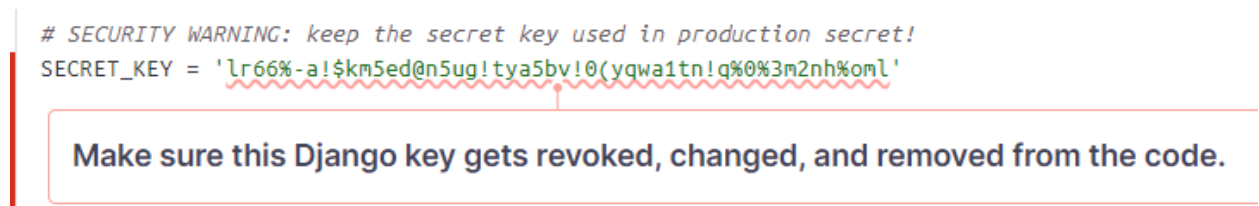


```
# steps -->
# 1. covert input code to corrsponding code and write in file
# 2. extract inputs form 2nd code
# 3
# 4
Refactor this function to reduce its Cognitive Complexity from 24 to the 15 allowed. [+11 locations] sonarlint(python:S3776)
View Problem (Alt+F8) Quick Fix... (Ctrl+;)
@cs
def ssrf_code_checker(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            python_code = request.POST['python_code']
            html_code = request.POST['html_code']
            if not (ssrf_code_converter(python_code)):
                return JsonResponse({"status": "error", "message": "Invalid code"})
            test_bench1 = ssrf_html_input_extractor(html_code)
```

Figure 6. Fonction ayant une complexité cognitive trop forte

De plus, SonarQube a révélé 2 types de vulnérabilités importantes:

1. Révoquer, changer et supprimer une clé Django du code



```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'lr66%-a!$km5ed@n5ug!tya5bv!0(yqwa1tn!q%0%3m2nh%oml'
```

Make sure this Django key gets revoked, changed, and removed from the code.

Figure 7. Partie du code comportant la clé Django, sensé être secrète

Il est important de ne pas divulguer les clés secrètes Django, car elles sont utilisées pour signer les cookies de session et les tokens CSRF, et une clé secrète compromise peut permettre à un attaquant de falsifier ces cookies et tokens, ce qui peut entraîner des attaques de type CSRF (Cross-Site Request Forgery) ou de vol de session. Nous avons trouvé

pour cette vulnérabilité deux codes communs: CVE-2023-43791⁴ et CVE-2023-47117⁵. Nous avons également trouvé un code de faiblesse, le CWE-200: Exposure of Sensitive Information to an Unauthorized Actor⁶

Comme solution, nous pourrions sauvegarder la clé dans une variable d'environnement permettant ainsi de garder les informations sensibles hors du code source, ce qui est une bonne pratique en matière de sécurité.

```
import os
SECRET_KEY = os.environ["SECRET_KEY"]
```

Figure 8. Méthode pour éviter d'avoir la clé secrète directement dans le code

2. Désactiver l'accès aux entités externes lors de l'analyse XML

```
parser = make_parser()
parser.setFeature(feature_external_ges, True)

doc = parseString(request.body.decode('utf-8'), parser=parser)
```

Disable access to external entities in XML parsing.

Figure 9. Partie du code montrant l'activation de l'accès aux entités externes

Si une application permet le traitement des entités externes, un attaquant peut exploiter cette faiblesse pour effectuer des attaques de type XXE(XML External Entity), qui peuvent entraîner la divulgation de données sensibles, des attaques par déni de service ou l'exécution de code à distance. Ce problème est lié à la faiblesse CWE-611: "Improper Restriction of XML External Entity Reference"⁷. Des vulnérabilités communes en lien avec la nôtre sont la CVE-2021-41411⁸ et la CVE-2022-0221⁹. Une solution pour parier à ce problème

⁴ <https://www.cve.org/CVERecord?id=CVE-2023-43791>

⁵ <https://www.cve.org/CVERecord?id=CVE-2023-47117>

⁶ <https://cwe.mitre.org/data/definitions/200.html>

⁷ <https://cwe.mitre.org/data/definitions/611.html>

⁸ <https://www.cve.org/CVERecord?id=CVE-2021-41411>

⁹ <https://www.cve.org/CVERecord?id=CVE-2022-0221>

¹⁰ - <https://cwe.mitre.org/data/definitions/710.html>

serait de mettre à False le second attribut de `parser.setFeature` pour empêcher l'accès à des entités externes

```
parser = make_parser()  
parser.setFeature(feature_external_ges, False)
```

Figure 10. Solution facile à la vulnérabilité provenant de l'activation de l'accès aux entités externes

Conclusion

En conclusion, l'analyse statique de l'application PyGoat a révélé plusieurs vulnérabilités majeures liées à la désactivation de protections critiques telles que la protection CSRF et l'auto-escaping, ainsi que la divulgation de clés sensibles. Ces vulnérabilités exposent l'application à des attaques graves, notamment les attaques de type XSS, CSRF, et XML External Entity (XXE). Des solutions ont été proposées pour atténuer ces risques, notamment la réactivation des protections et la gestion sécurisée des clés sensibles. En corrigeant ces vulnérabilités, l'application gagnera en sécurité et fiabilité.