TD ANALYSE STATIQUE CODE

Léo Le Bihan - Emile Derain

24 janvier 2024

1 BANDIT

Bandit est un outil conçu pour détecter les problèmes de sécurité courants dans le code Python. Pour ce faire, Bandit traite chaque fichier, crée un AST à partir de celui-ci et exécute les plugins appropriés sur les nœuds AST. Une fois que Bandit a fini d'analyser tous les fichiers, il génère un rapport.

Github : https://github.com/PyCQA/bandit

1.1 Exercice 1

Commencez par faire un scan Bandit sans préciser de fichier de configuration et sauvegardez ce premier rapport aux formats html et text.


Metrics:
Total lines of code: 105
Total lines skipped (#nosec): 0

django_extra_used: Use of extra potential SQL attack vector.
Test ID: B610
Severity: MEDIUM
Confidence: MEDIUM
CWE: CWE-89
File: ./1.py
Line number: 12
More info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html

```
11
12      User.objects.filter(username='admin').extra(dict(could_be='insecure'))
13      User.objects.filter(username='admin').extra(select=dict(could_be='insecure'))
```

django_extra_used: Use of extra potential SQL attack vector.
Test ID: B610
Severity: MEDIUM
Confidence: MEDIUM
CWE: CWE-89
File: ./1.py
Line number: 13
More info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html

```
12      User.objects.filter(username='admin').extra(dict(could_be='insecure'))
13      User.objects.filter(username='admin').extra(select=dict(could_be='insecure'))
14      query = '"username") AS "username", * FROM "auth_user" WHERE 1=1 OR "username"=? --'
```

django_extra_used: Use of extra potential SQL attack vector.
Test ID: B610
Severity: MEDIUM
Confidence: MEDIUM
CWE: CWE-89
File: ./1.py
Line number: 15
More info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html

```
14      query = '"username") AS "username", * FROM "auth_user" WHERE 1=1 OR "username"=? --'
15      User.objects.filter(username='admin').extra(select={'test': query})
16      User.objects.filter(username='admin').extra(select={'test': '%secure' % 'nos'})
```

FIGURE 1 – Rapport html - Exercice 1

Etudiez ce rapport et lorsque vous pensez avoir trouvé quelle vulnérabilité est contenue dans chacun des 5 scripts, générez un nouveau fichier de configuration ne contenant que le module associé à cette vulnérabilité. Sauvez votre rapport aux formats html et text

```
Run started:2024-01-23 14:04:18.864460

Test results:
>> Issue: [B610:django_extra_used] Use of extra potential SQL attack vector.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html
   Location: ./1.py:12:0
11
12    User.objects.filter(username='admin').extra(dict(could_be='insecure'))
13    User.objects.filter(username='admin').extra(select=dict(could_be='insecure'))

--------------------------------------------------
>> Issue: [B610:django_extra_used] Use of extra potential SQL attack vector.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html
   Location: ./1.py:13:0
12    User.objects.filter(username='admin').extra(dict(could_be='insecure'))
13    User.objects.filter(username='admin').extra(select=dict(could_be='insecure'))
14    query = '"username") AS "username", * FROM "auth_user" WHERE 1=1 OR "username"=? --'

--------------------------------------------------
>> Issue: [B610:django_extra_used] Use of extra potential SQL attack vector.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html
   Location: ./1.py:15:0
14    query = '"username") AS "username", * FROM "auth_user" WHERE 1=1 OR "username"=? --'
15    User.objects.filter(username='admin').extra(select={'test': query})
16    User.objects.filter(username='admin').extra(select={'test': '%secure' % 'nos'})

--------------------------------------------------
>> Issue: [B610:django_extra_used] Use of extra potential SQL attack vector.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html
   Location: ./1.py:16:0
15    User.objects.filter(username='admin').extra(select={'test': query})
16    User.objects.filter(username='admin').extra(select={'test': '%secure' % 'nos'})
17    User.objects.filter(username='admin').extra(select={'test': '{}secure'.format('nos')})

--------------------------------------------------
>> Issue: [B610:django_extra_used] Use of extra potential SQL attack vector.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b610_django_extra_used.html
   Location: ./1.py:17:0
16    User.objects.filter(username='admin').extra(select={'test': '%secure' % 'nos'})
17    User.objects.filter(username='admin').extra(select={'test': '{}secure'.format('nos')})
18
```

FIGURE 2 – Rapport txt - Exercice 1

Les fichiers des différents rapports (config.yml, text et html) sont sur ce repository : https://github.com/EmileDerain/Analyse_statique_code

2 SEMGREP

Semgrep accélère votre parcours de sécurité en analysant rapidement les dépendances du code et des packages à la recherche de problèmes connus, de vulnérabilités logicielles et de secrets détectés avec une efficacité inégalée.

Github : https://github.com/semgrep/semgrep

2.1 Exercice 2

Pour chacun des dossiers, vous devez :
- *Réaliser un scan complet de l'application*
- *Identifier les vulnérabilités High/Medium*
- *Corriger ces vulnérabilités*
- *Réaliser à nouveau un scan afin de vérifier votre correction*

2.1.1 Dossier n°1

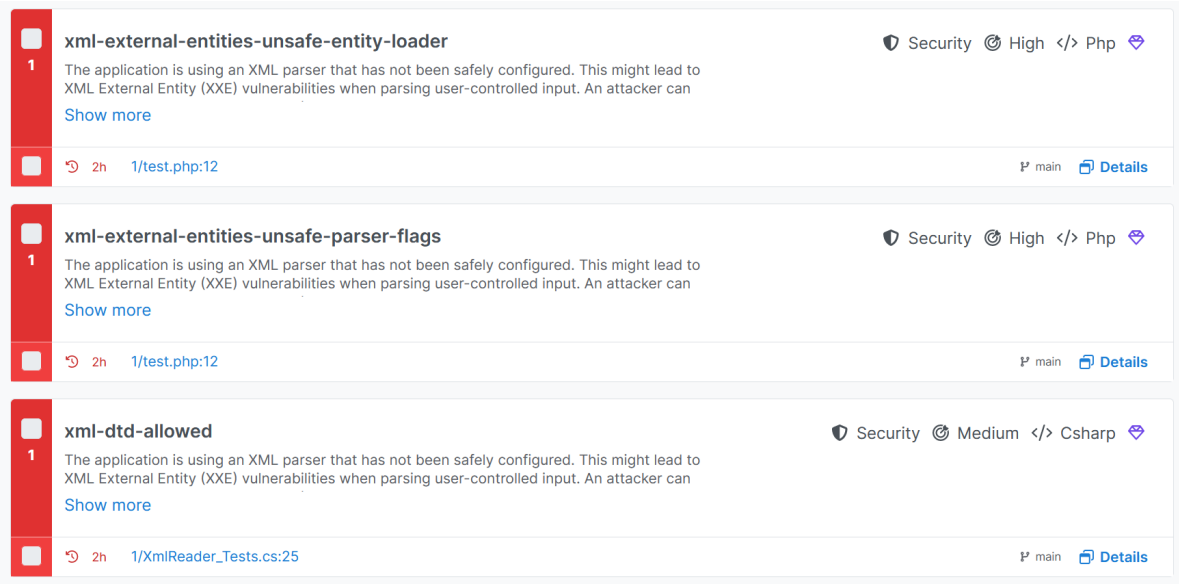

FIGURE 3 – Rapport - Exercice 2 - Dossier n°1.

Ressource pour corriger les vulnérabilités :

- Erreur *xml-external-entities-unsafe-entity-loader* : libxml_disable_entity_loader-deprecation
- Erreur *xml-dtd-allowed* : Avoiding XXE vulnerabilities in .NET

2.1.2 Dossier n°2

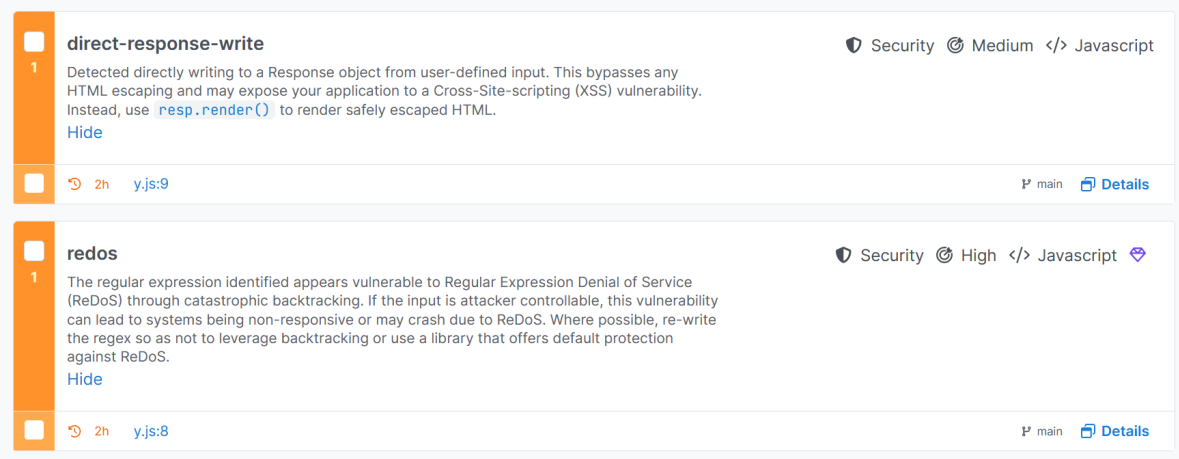

FIGURE 4 – Rapport - Exercice 2 - Dossier n°2

Ressource pour corriger les vulnérabilités :

- Erreur *direct-reponse-write* : utilisation de render()
- Erreur *redos* : utilisation de la bibliothèque super-regex
- Erreur *res-render-injection* : Understanding res.redirect and res.render in Express.js: Usage and Security Measures

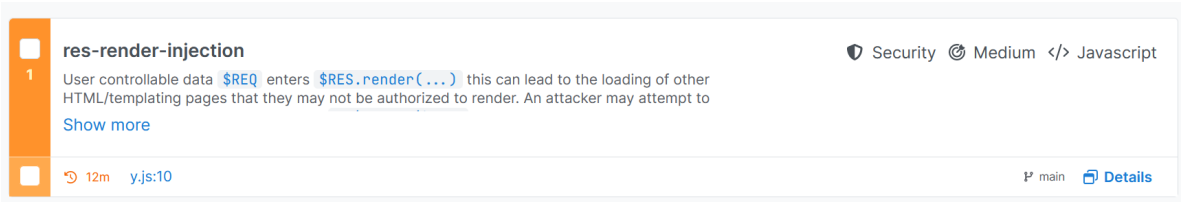

FIGURE 5 – Rapport - Exercice 2 - Dossier n°2 suite

2.1.3 Dossier n°3

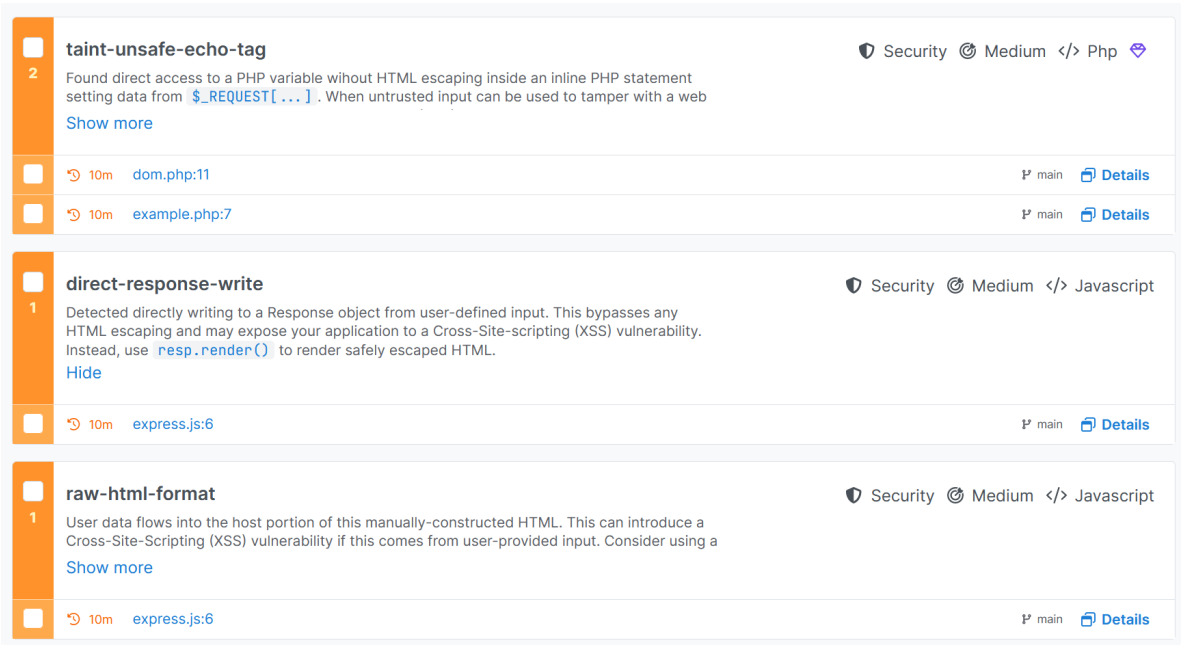

FIGURE 6 – Rapport - Exercice 2 - Dossier n°3

Ressource pour corriger les vulnérabilités :

- Erreur *taint-unsafe-echo-tag* : utilisation de htmlspecialchars()
- Erreur *raw-html-format* : utilisation de sanitize()

2.1.4 Dossier n°4

Ressource pour corriger les vulnérabilités :

- Erreur *open-redirect-deepsemgrep* : pour éviter cette vulnérabilité, effectuez une validation stricte des entrées du domaine par rapport à une liste autorisée de domaines approuvés.
- Erreur *non-literal-header* : pour résoudre ce problème, n'autorisez pas les espaces à l'intérieur de header().

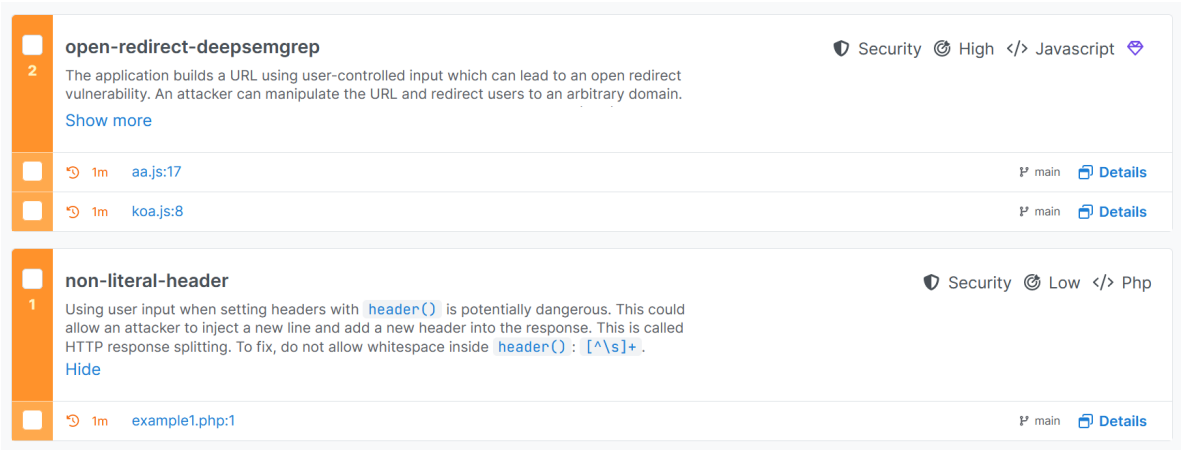

FIGURE 7 – Rapport - Exercice 2 - Dossier n°4

2.2 Exercice 3

Auditez l'application contenue dans le dossier 'ex_3', corrigez les vulnérabilités High détectées et vérifiez vos corrections avec un scan de contrôle.

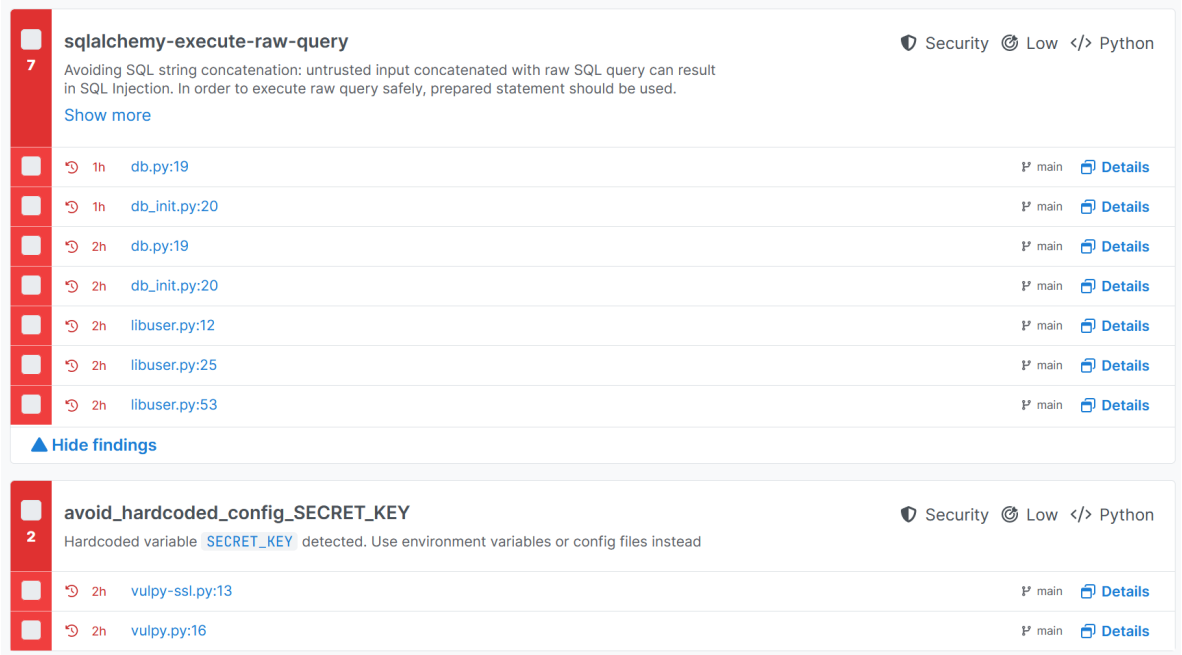

FIGURE 8 – Rapport - Exercice 3

Ressource pour corriger les vulnérabilités :

- Erreur *sqlalchemy-execute-raw-query* : Preventing SQL Injection Attacks With Python
- Erreur *avoid_hardcoded_config_SECRET_KEY* : How to use dotenv package to load environment variables in Python

2.3 Exercice 4

Pour ce dernier exercice, vous devez fournir un code contenant une vulnérabilité de votre choix qui n'est détectable qu'avec une analyse dynamique, réalisée avec un DAST.
Les documents attendus sont :

- *Le code vulnérable ainsi que tous les fichiers nécessaires pour lancer l'application.*

— *Le rapport de SemGrep*
— *des captures d'écran de la détection de la vulnérabilité par un DAST (Burp, Zap...)*
— *Une courte explication de la vulnérabilité ainsi que les raisons de sa non-détection par un SAST*

Scan de ex_4 avec le fichier qui contient une vulnérabilité non détectable par le SAST.
Nous avons écrit un petit programme en Python qui lance un serveur Flask et propose une réponse lors de l'exécution avec un paramètre d'entrée. Cependant, l'entrée utilisateur n'est pas sécurisée et est insérée dans un "os.popen" qui effectue un écho de l'entrée utilisateur. De notre côté, nous pouvons exploiter cette faille avec l'entrée utilisateur suivante : " ;ls", ce qui nous permet de lancer la commande "ls" sur la machine qui exécute le serveur.

```python
from flask import Flask, request
import os

app = Flask(__name__)

def vulnerable_function(input_data):
    # Vulnerabilit ~:~Injection de commande
    # Ce code vuln rable permet l'ex cution de commandes arbitraires.
    result = os.popen(f"echo {input_data}").read()
    return result

@app.route('/execute', methods=['GET'])
def execute_code():
    user_input = request.args.get('input', '')
    output = vulnerable_function(user_input)
    return output

if __name__ == '__main__':
    app.run(debug=True)
```

Code 1 – Code contenant une vulnérabilité

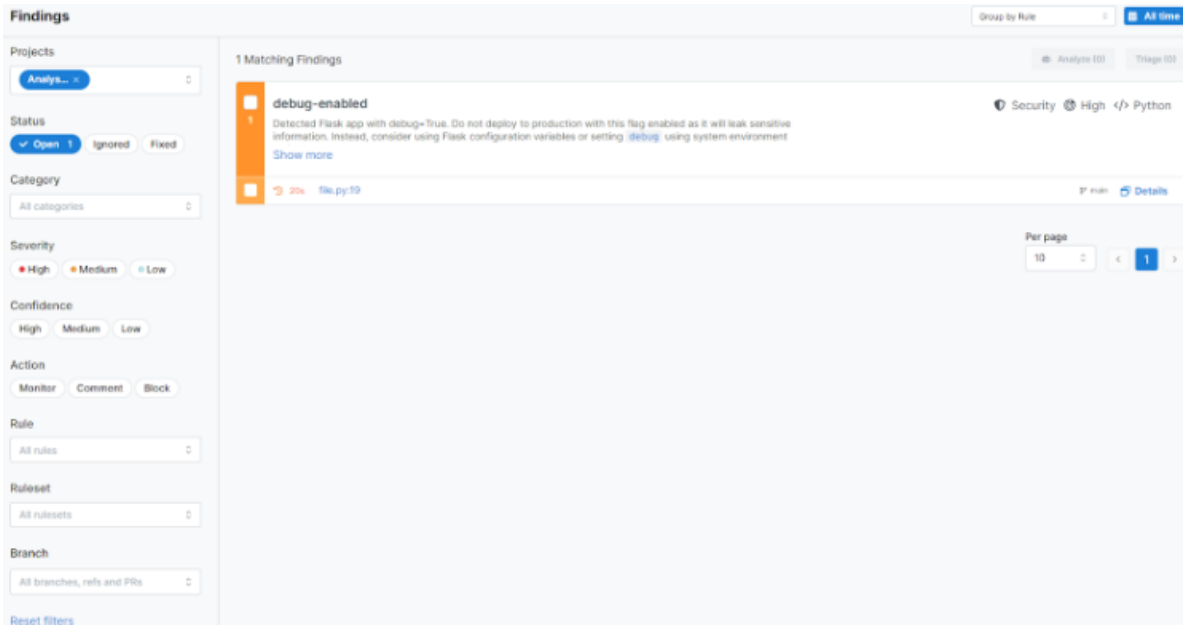

FIGURE 9 – Rapport - Exercice 4 - Semgrep ne détecte pas cette vulnérabilité

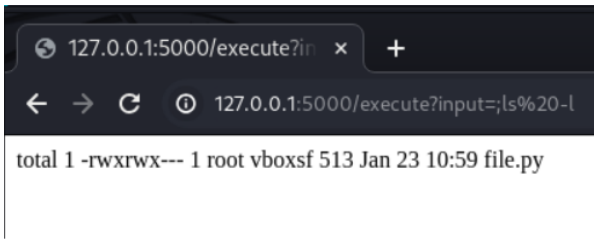

FIGURE 10 – Exercice 4 - Payload dans un navigateur

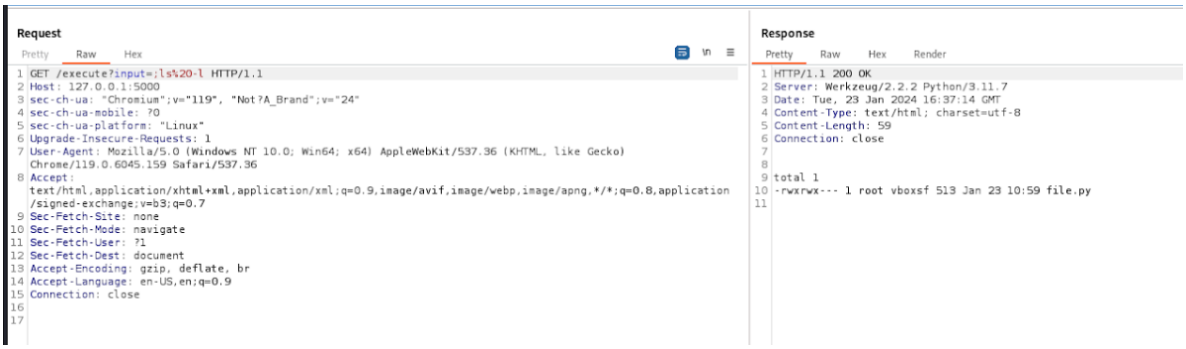

FIGURE 11 – Exercice 4 - Constatation de l'insertion du payload sur Burp