|  |
| --- |
|  |
| Rapport d’Audit de la sécurité du système d’information |
| De VAmPI |

|  |
| --- |
| Serena KOUAM  20/01/2025 |



Table des matières

[1. Introduction 2](#_Toc191211159)

[1.1 Contexte 2](#_Toc191211160)

[1.2 Périmètre 2](#_Toc191211161)

[1.3 Méthodologie 2](#_Toc191211162)

[2 Résultats de l’audit 3](#_Toc191211163)

[2.1 Vulnérabilités détectées 3](#_Toc191211164)

[2.2 Exploitation détaillée 4](#_Toc191211165)

[2.2.1 Vulnerabilité : Broken Object Level Authorization 4](#_Toc191211166)

[2.2.2 Vulnerabilité : Broken authentification 7](#_Toc191211167)

[*2.2.3 Object Property level Authorization* 11](#_Toc191211168)

[2.2.4 Unrestricted Resource Consumption 13](#_Toc191211169)

[2.2.5 Vulnerabilité : Broken function level authorization 15](#_Toc191211170)

[2.2.6 Unrestricted Access to Sensitive Business Flows 17](#_Toc191211171)

[Description 17](#_Toc191211172)

[2.2.7 Server-Side Request Forgery(SSRF) 19](#_Toc191211173)

[2.1.8 Security Misconfiguration 21](#_Toc191211174)

[2.2.9 Improper Inventory management 23](#_Toc191211175)

[2.2.10 Unsafe Consumption of API 26](#_Toc191211176)

[3. Recommandations générales 27](#_Toc191211177)

[3.1 Mesures correctives prioritaires 27](#_Toc191211178)

[Conclusion 28](#_Toc191211179)

# Introduction

## Contexte

Dans un environnement numérique en constante évolution, la sécurisation des API REST est une priorité pour les entreprises soucieuses de protéger leurs données et infrastructures. Ce rapport d’audit a pour objectif d’évaluer le niveau de sécurité de l’API VAmPI, en appliquant les recommandations du OWASP API Security Top 10 - 2023.

L’audit vise à identifier et analyser les vulnérabilités potentielles afin de mesurer les risques associés et d’émettre des recommandations concrètes pour renforcer la sécurité de l’application. Ce processus s’inscrit dans une démarche proactive de cybersécurité et de conformité réglementaire afin de prévenir toute exploitation malveillante et d’assurer une protection optimale des actifs numériques de l’organisation.

## Périmètre

L’audit de sécurité a été réalisé sur l’API REST VAmPI, en prenant en compte les éléments suivants :

* API testée : VAmPI
* Environnement cible : API exposée localement sur un serveur Flask
* URL de l’instance analysée : <http://127.0.0.1:5000/>
* Méthodes HTTP testées : GET, POST, PUT, DELETE
* Environnement cible : API exposée localement sur un serveur Flask\*
* Mécanisme d’authentification : Jetons JWT (JSON Web Token)
* Infrastructure de test : Conteneurisation via Docker
* Plateforme de déploiement : Instance exécutée en environnement Docker

L’analyse a été menée dans un cadre contrôlé, en simulant des scénarios d’attaque réalistes afin d’évaluer les failles de sécurité exploitables.

## Méthodologie

L’évaluation de la sécurité de l’API VAmPI s’est déroulée en plusieurs étapes, combinant à la fois des tests manuels et automatisés. L’environnement de test a été mis en place sous Docker, permettant de simuler un déploiement réaliste et d’évaluer l’impact des vulnérabilités découvertes.

Dans un premier temps, l’analyse a porté sur la structure des requêtes et réponses HTTP en utilisant Burp Suite et Postman, afin de mieux comprendre le fonctionnement de l’API et de détecter d’éventuels comportements anormaux.

L’évaluation des mécanismes d’authentification et d’autorisation a été approfondie par des tests ciblés sur la gestion des jetons JWT. L’outil jwt\_tool a été utilisé pour examiner d’éventuelles faiblesses cryptographiques et tenter de falsifier des tokens d’authentification. En parallèle, des attaques de force brute ont été effectuées à l’aide de Burp Suite Intruder, visant les endpoints de connexion pour évaluer leur résilience face à des tentatives d’authentification non autorisée.

Des techniques de fuzzing ont été mises en œuvre avec ffuf et Burp Suite Intruder afin de repérer d’éventuelles erreurs d’implémentation et des fuites d’informations sensibles.

Enfin, des tests de robustesse ont été réalisés pour évaluer la capacité de l’API à gérer des charges anormales et prévenir les attaques par déni de service (DoS). Plusieurs scénarios ont été exécutés afin d’identifier d’éventuelles failles liées à l’absence de limitations sur les ressources consommées.

# Résultats de l’audit

## 2.1 Vulnérabilités détectées

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Catégorie | Description | Impact | Exploitation | Criticité |
| API1 :2023 | BOLA | Mauvaise gestion des autorisations des objets (ID modifiable) | Accès non autorisé aux données d'autres utilisateurs | Modification d'un ID dans l'URL | Critique |
| API2 :2023 | Broken authentification | Failles dans l’authentification permettant une prise de contrôle de compte | Compromission de comptes, usurpation d’identité, accès non autorisé | Attaque par force brute sur les mots de passe, manipulation du JWT |  |
| API3 :2023 | Broken Object Property Level Authorization | Absence de restrictions sur la modification des propriétés d’un objet | Un utilisateur peut modifier son rôle en admin ou accéder à des champs sensibles | Modification directe des attributs via les requêtes PUT/PATCH | Élevée |
| API4 :2023 | Unrestricted Resource Consumption | Absence de limitation des ressources API, exposant le serveur à un déni de service (DoS) | Saturation des ressources, ralentissement ou crash du service | Envoi massif de requêtes, absence de rate limiting, absence de pagination | Critique |
| API5 :2023 | Broken Function Level AUthorization | Absence de contrôle sur certaines fonctionnalités sensibles | Un utilisateur peut exécuter des actions réservées aux administrateurs | Suppression de comptes, modification de rôles via des requêtes non sécurisées | Élevée |
| API6 :2023 | Unrestricted Access to Sensitive Business Flows | Possibilité d’exploiter des flux critiques de l’API sans restriction | Exécution illimitée de certaines opérations, automatisation d’attaques | Création en masse de ressources, modification répétée d’objets sensibles | Critique |
| API7 :2023 | Server Side Request Forgery | Possibilité de forcer le serveur à envoyer des requêtes HTTP vers des ressources non prévues | Accès à des services internes, exfiltration de données sensibles, pivot réseau | Tentative d’accès à des URL internes, exploitation des métadonnées cloud | Modérée |
| API8 :2023 | Security Misconfiguration | Mauvaise configuration de l’API exposant des informations sensibles | Fuite d’informations système, accès à des interfaces d’administration ou debug | Headers HTTP révélant la version du serveur, accès à des endpoints non sécurisé | Élevée |
| API9 :2023 | Improper Inventory Management | Manque de gestion des endpoints exposant des fonctionnalités non documentées | Un attaquant peut découvrir et exploiter des fonctionnalités cachées | Scan des endpoints avec ffuf, accès à des interfaces internes comme /console | Élevée |
| API10 :2023 | Unsafe Consumption of API | Absence de validation des données provenant de sources externe | Exécution de code malveillant, ingestion de données non sécurisées | Mauvaise validation des entrées API, injection de données tierces non contrôlées | Modérée |

## 2.2 Exploitation détaillée

### 2.2.1 Vulnerabilité : Broken Object Level Authorization

#### Description

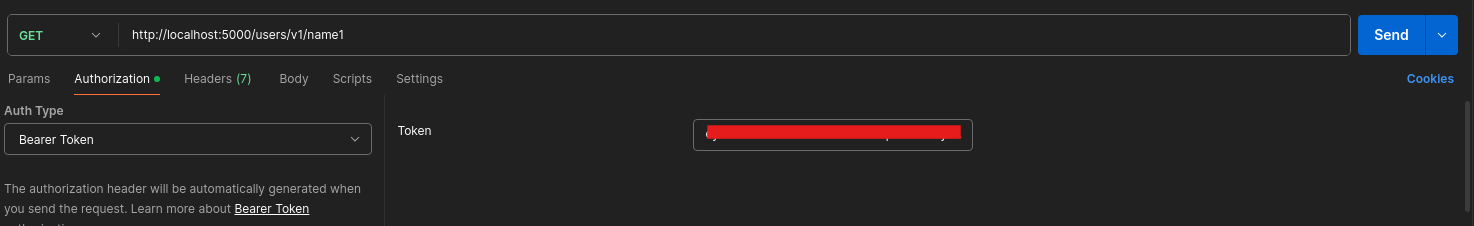
La vulnérabilité BOLA (Broken Object Level Authorization) survient lorsqu’un utilisateur peut accéder, modifier ou supprimer des ressources qui ne lui appartiennent pas en manipulant les paramètres d’une requête API, notamment via des identifiants d’objets prévisibles. L’absence de vérification des autorisations permet à un attaquant d’accéder à des informations sensibles appartenant à d’autres utilisateurs ou d’effectuer des actions non autorisées.

Scénario d’attaque et exploitation

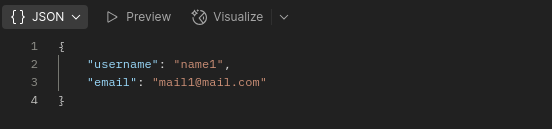
1. Accès non autorisé aux données d’un autre utilisateur

L’objectif est de vérifier si un utilisateur peut voir les informations d’un autre utilisateur sans permission.

Test réalisé :  
 L’utilisateur authentifié name1 effectue une requête pour récuperer ses propres informations :

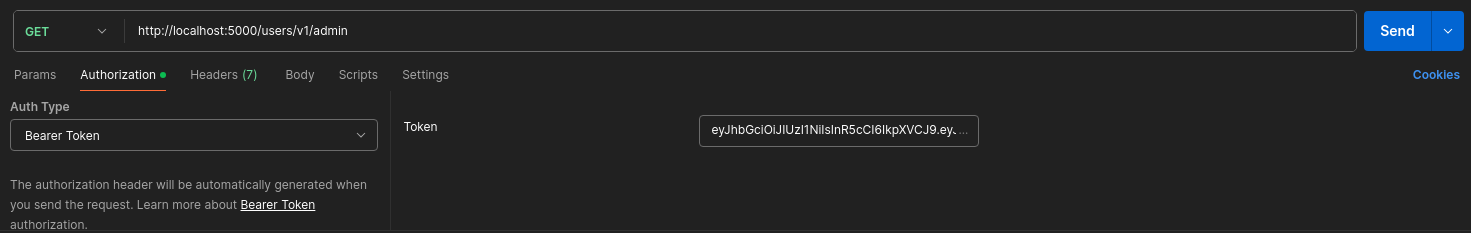


Reponse de l’API :



Les informations de l’utilisateur name1 sont correctement retournées.

L’utilisateur « name1 » modifie ensuite l’URL pour récuperer les données d’un autre utilisateur (admin),sans avoir les autorisations nécessaires :

  
  
Réponse de l’API :  


Les informations de l’utilisateur (admin) sont retorunées.  
  
Problème détecté :

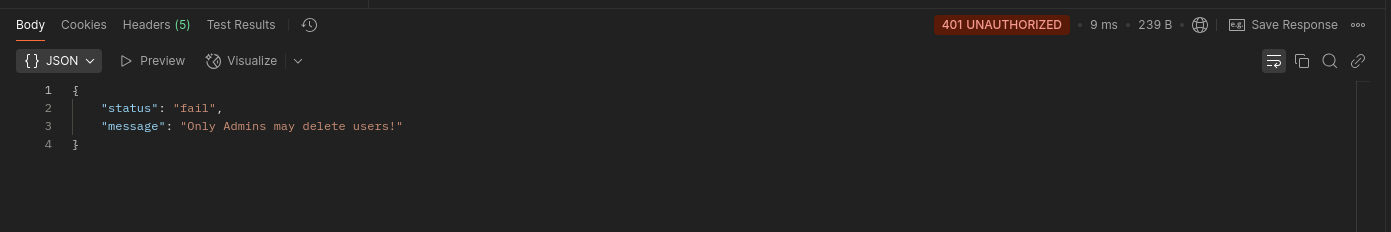
L’API ne vérifie pas correctement les droits d’accès et retourne des informations sensibles à un utilisateur non autorisé. Cette faille constitue une exposition de données critiques.

1. Suppression non autorisée d’un utilisateur

L’objectif est de verifier si un utilisateur standard peut supprimer un autre compte utilisateur.   
  
Test realisé :  
 L’utilisateur « name1 » tente de supprimer « name2 » en effectuant la requête suivante :

#### 

#### Réponse de l’API :



L’API retourne un message d’erreur indiquant que l’action n’est pas autorisée.

Problème détecté :

Dans ce cas précis, la suppression est bien sécurisée : un utilisateur ne peut pas supprimer un autre utilisateur sans autorisation suffisante.

#### Impact de la vulnérabilité :

Fuite de données sensibles :

Un attaquant peut consulter des informations confidentielles appartenant à d’autres utilisateurs, comme des adresses e-mail ou des données personnelles augmentant le risque de violation de la vie privée.

Usurpation d’identité et attaques de phishing

L’accès aux informations utilisateur peut permettre de mener des campagnes de phishing ciblé ou de tenter une prise de contrôle de compte via des attaques de type ingénierie sociale.

Non-conformité réglementaire :

Cette vulnérabilité enfreint les principes de confidentialité et de protection des données exigés par des réglementations comme le RGPD ou le HIPAA dans le secteur de la santé.

2.2.2 Vulnerabilité : Broken authentification

#### Description

#### La vulnérabilité Broken Authentication se produit lorsqu’un attaquant parvient à contourner les mécanismes d’authentification pour accéder à des comptes utilisateurs sans autorisation. Cette faille peut être exploitée via des attaques par force brute, une mauvaise gestion des sessions ou des failles dans la validation des tokens JWT.

#### L’absence de limitations sur les tentatives de connexion ou une mauvaise gestion des signatures de jetons peut permettre à un attaquant de prendre le contrôle de comptes sensibles et d’élever ses privilèges sans authentification légitime

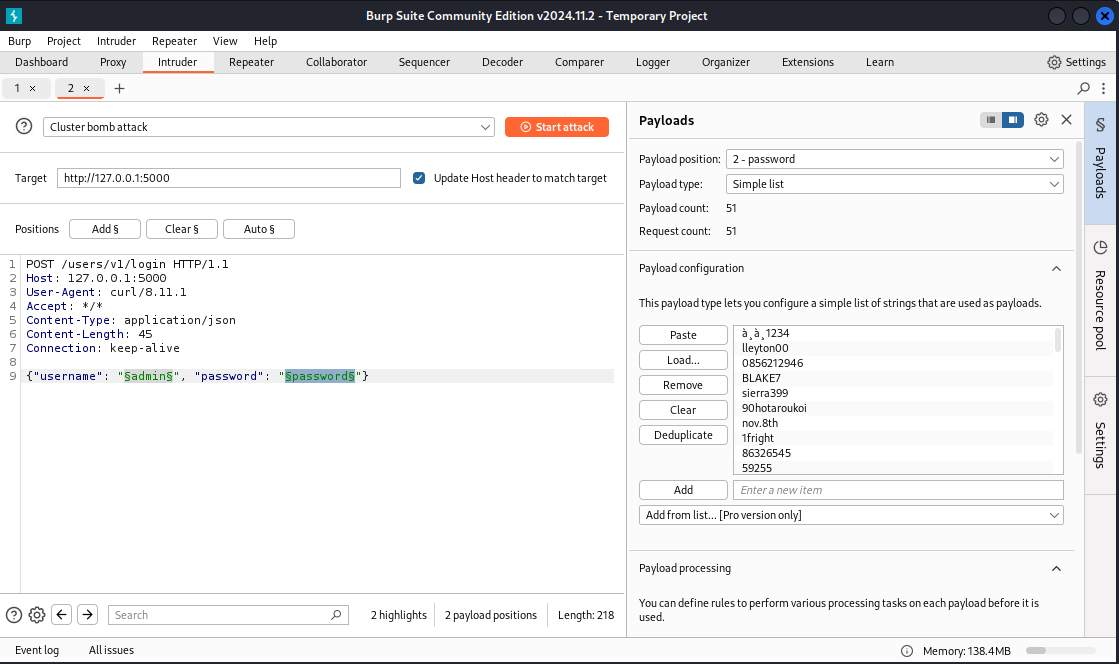
#### Scénario d’attaque et exploitation

1. Force brute sur le login

L’objectif est de vérifier si l’API permet d’effectuer un grand nombre de tentatives de connexion sans restriction.

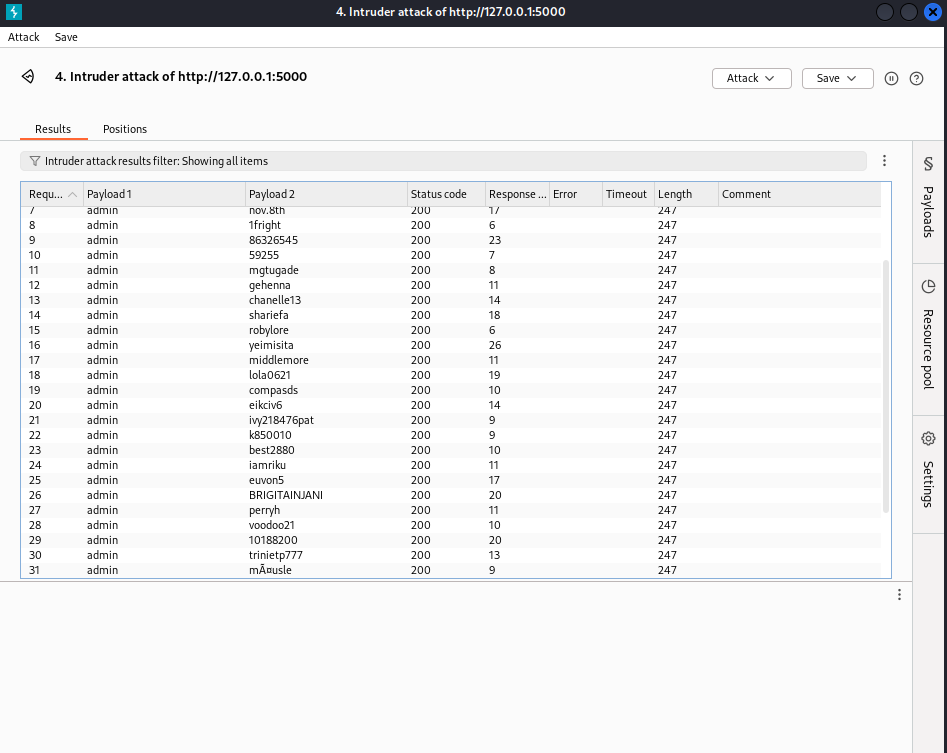
Test réalisé avec Burp Suite Intruder :

Un attaquant tente de bruteforcer l’authentification en envoyant plusieurs requêtes avec des combinaisons de noms d’utilisateur et mots de passe :

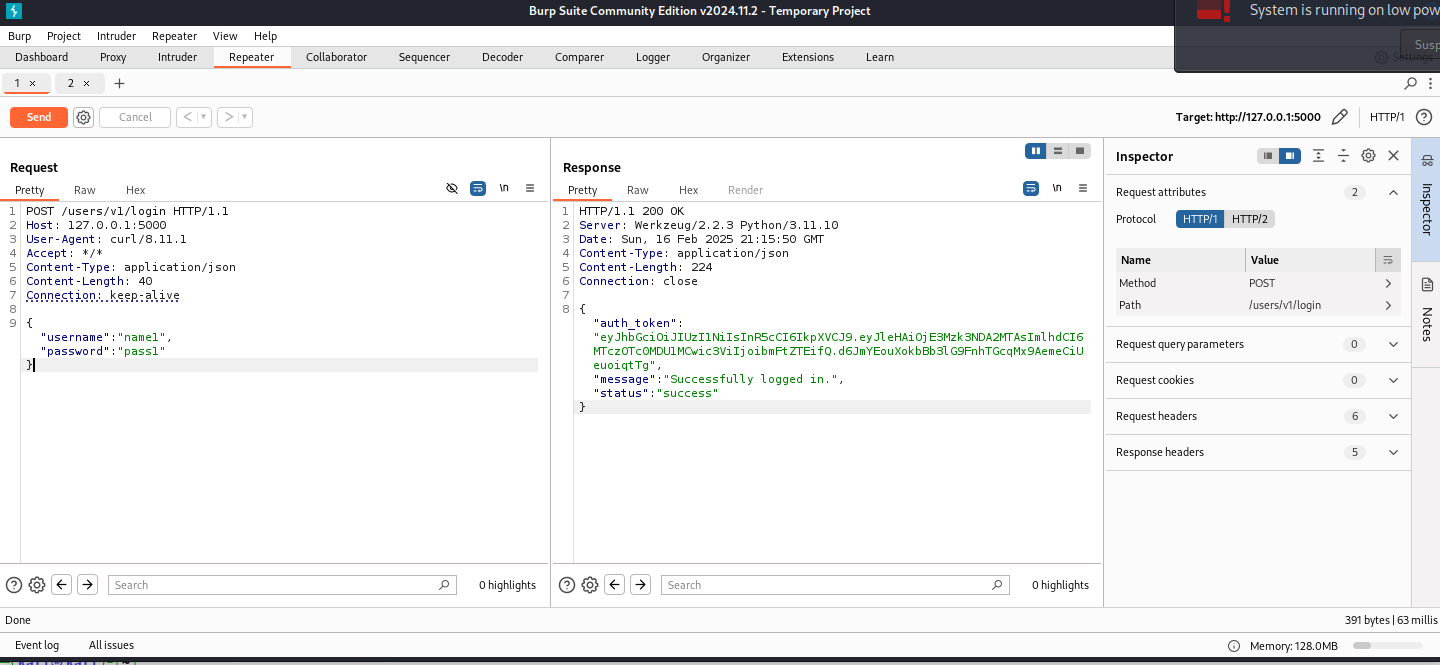


Les mots de passe sont testés automatiquement à l’aide d’un dictionnaire de valeurs courantes.

Réponse de l’API :



L’API ne met en place aucune limitation du nombre d’essais, ce qui signifie qu’un attaquant peut tester un grand nombre de mots de passe jusqu’à trouver la bonne combinaison.



Problème détecté :

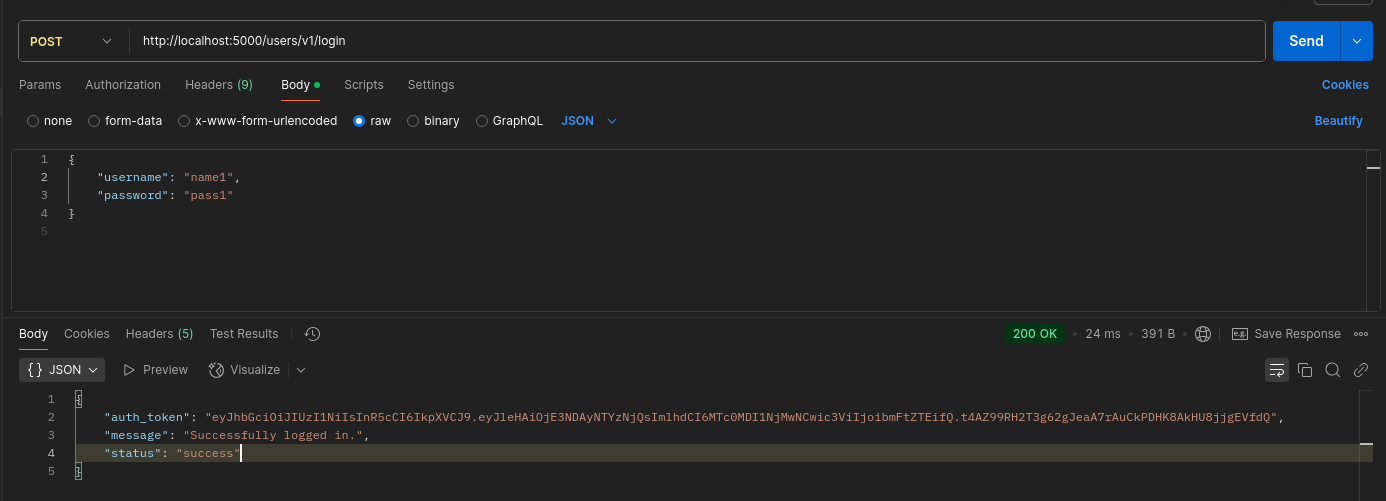
L’absence de rate-limiting et de mesures anti-bruteforce permet à un attaquant d’essayer un nombre illimité de mots de passe, rendant l’API vulnérable aux attaques par force brute.

1. Manipulation d’un token JWT

L’objectif est de vérifier si l’API accepte un JWT mal formé ou modifié, permettant ainsi une élévation de privilèges.

Test réalisé :

On commence par récupérer un token JWT valide en effectuant une authentification classique



L’API retourne un token JWT valide.

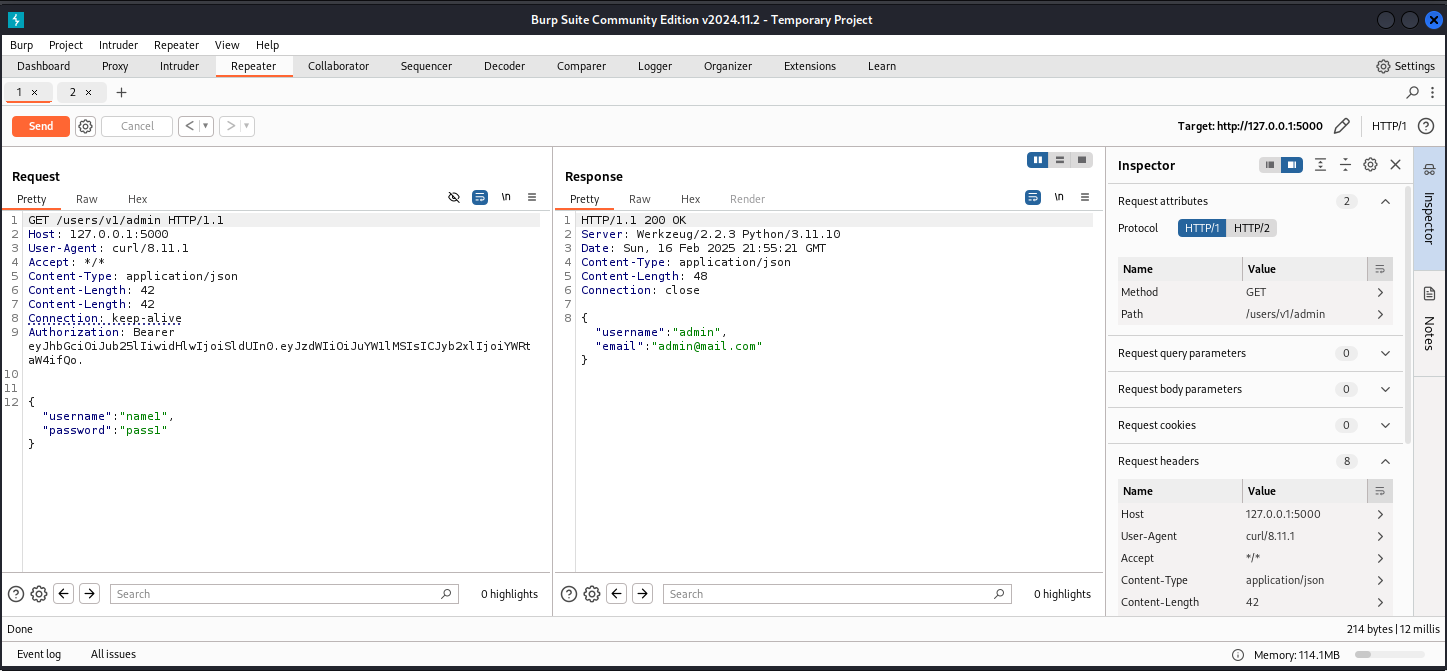
On décode et modifions le payload jwt pour changer de rôle en tant qu’(admin) :



L’exploitation de cette vulnérabilité s’est déroulée en plusieurs étapes. Tout d’abord, un utilisateur standard, authentifié sous l'identité "name1", a récupéré son token JWT légitime en effectuant une requête d’authentification. Ce token, structuré en trois parties distinctes (header, payload et signature), a ensuite été décodé afin d’analyser son contenu. Il est alors apparu que l’attribut "role", censé définir les permissions de l’utilisateur, était stocké en clair au sein du payload.

Après avoir isolé cette partie du token, nous avons procédé à une modification manuelle du champ "role" en remplaçant la valeur "user" par "admin". Une fois le token altéré, une tentative de requête vers une route réservée aux administrateurs a été réalisée en incluant le JWT falsifié dans l’en-tête de la requête HTTP.

Résultat de l’API :



L’API a répondu positivement, confirmant que le serveur ne procédait à aucune validation de l’intégrité du JWT, ni à une vérification rigoureuse de la signature permettant à un attaquant d’élever ses privilèges .

#### Impact de la vulnérabilité :

Accès non autorisé :

Un attaquant peut falsifier un JWT et accéder aux ressources administratives sans authentification valide.

Élévation de privilèges :

En modifiant le JWT, un utilisateur standard peut obtenir un accès administrateur.

Atteinte à la confidentialité :

Des informations sensibles peuvent être consultées ou modifiées.

#### 2.2.3 Object Property level Authorization

#### Description

La vulnérabilité Broken Object Property Level Authorization (BOPLA) se produit lorsque l’API ne met pas en place des contrôles d’accès stricts sur les propriétés sensibles des objets. Cela signifie qu’un utilisateur malveillant peut modifier des champs critiques comme le rôle d’un compte utilisateur ("role": "admin"), l’email, ou d’autres attributs sensibles sans disposer des permissions adéquates.

Dans le cadre de cet audit, cette vulnérabilité a été confirmée à travers l’exploitation des failles API1:2023 (BOLA - Broken Object Level Authorization) et API2:2023 (Broken Authentication), démontrant une gestion insuffisante des permissions sur les objets et leurs propriétés.

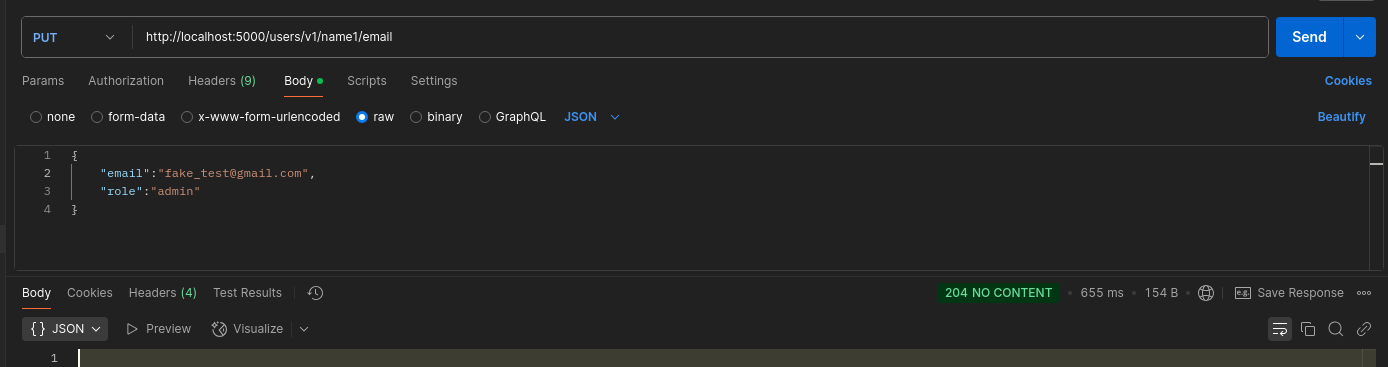
#### Résumé des test réalisés :

Les différentes tentatives menées ont mis en évidence plusieurs failles critiques dans la gestion des permissions :

Un utilisateur standard peut accéder aux informations d’un autre en modifiant simplement son identifiant.  
 Il est possible de modifier le rôle d’un compte en manipulant un JWT (JSON Web Token) mal protégé.  
L’API n’empêche pas la modification de champs sensibles lors des requêtes PUT ou PATCH.  
L’absence de validation stricte sur la signature du JWT permet une élévation de privilèges.

#### Exemple d’exploitation : modification des rôles utilisateurs :

Un utilisateur lambda peut tenter de modifier son propre rôle en envoyant une requête PUT non autorisée :



#### Impact de la vulnérabilité :

Escalade de privilèges :

Un utilisateur non autorisé peut obtenir des droits administrateurs et prendre le contrôle total de l’API.

Modification illégitime des données :

Un attaquant peut modifier ses propres privilèges ou ceux d’autres utilisateurs, compromettant l’intégrité du système.

Violation des principes de sécurité :

L’absence de contrôle strict sur les propriétés modifiables va à l’encontre des bonnes pratiques de gestion des accès.

Risque de compromission totale :

Si un attaquant obtient un accès administrateur, il peut supprimer, modifier ou exfiltrer des données sensibles et compromettre l’ensemble de l’application.

### 2.2.4 Unrestricted Resource Consumption

#### Description

L’Unrestricted Resource Consumption est une vulnérabilité qui se produit lorsque l’API n’impose aucune restriction sur l’utilisation des ressources serveur. Une API mal configurée permet aux utilisateurs d’envoyer un grand nombre de requêtes sans limitation, entraînant un risque de saturation du service.

#### Scénario d’attaque et exploitation

1. Test de Rate Limiting(Flooding)

L’objectif est de vérifier si l’API accepte un nombre illimité de requêtes en un laps de temps réduit.

Test réalisé :

Nous avons envoyé avec ffuf un volume important de requêtes en très peu de temps pour évaluer la réaction de l’API face à une tentative de déni de service.



Résultat de l’API :

L’API a retourné un statut 200 OK après des milliers de requêtes successives et admin est apparu dans la wordlist.

4614 requêtes envoyées en moins d'une minute, soit 48 requêtes par seconde.

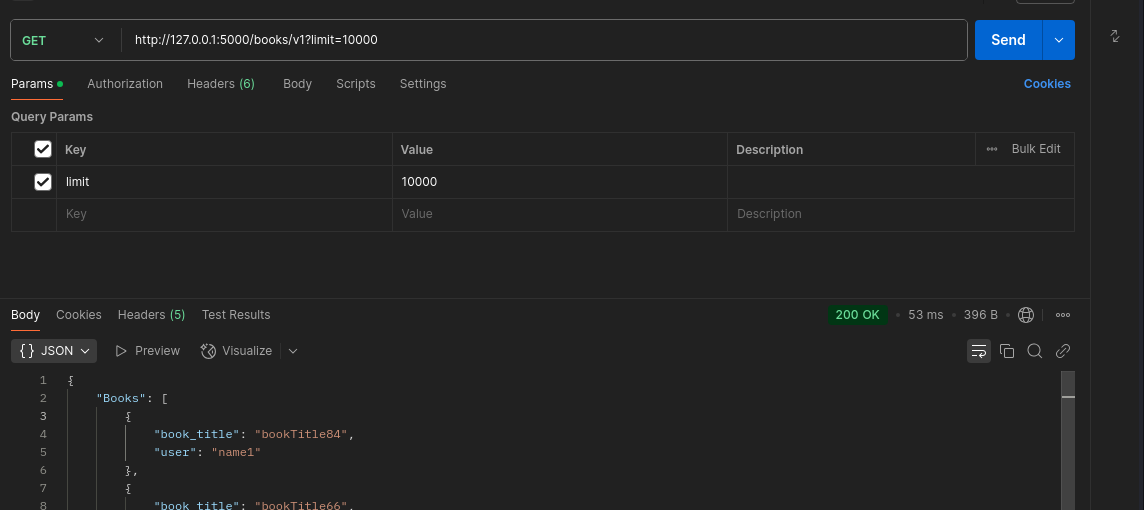
Aucune restriction appliquée : L’API n’a mis en place aucune limitation pour freiner cette attaque.  
Si une attaque DoS était en cours,l’Api pourrait être surchargée et ralentir/crasher.

1. Test d’absence de pagination (Data Overload)

L’objectif est d’évaluer si l’API retourne un volume excessif de données en une seule reqête,ce qui peut entraîner un déni de service (DoS) et exposer une grande quantité d’informations sensibles.

Test réalisé :

Nous avons envoyé une requête demandant 10 000 utilisateurs via un paramètre limit non restreint.



Résultat de l’API :

L’API a repondu avec un statut 200 OK,confirmant l’absence de restructions, de plus une grande quantité de données a « été retournée d’un coup.

L’application a montré des signes de ralentissement après plusieurs requêtes successives.

#### Impact de la vulnérabilité :

Risque de DoS (Déni de Service) :

Un attaquant peut envoyer plusieurs requêtes volumineuses pour saturer le serveur et rendre l’API indisponible.

Épuisement des ressources serveur :

Une charge excessive peut consommer la RAM et le CPU du backend, affectant la stabilité du système.

Perte de performance :

L’API peut ralentir significativement, dégradant l’expérience utilisateur.

Exposition massive de données sensibles :

Sans mécanisme de limitation, un attaquant peut extraire la totalité de la base de données d’un seul coup.

2.2.5 Vulnerabilité : Broken function level authorization

#### Description

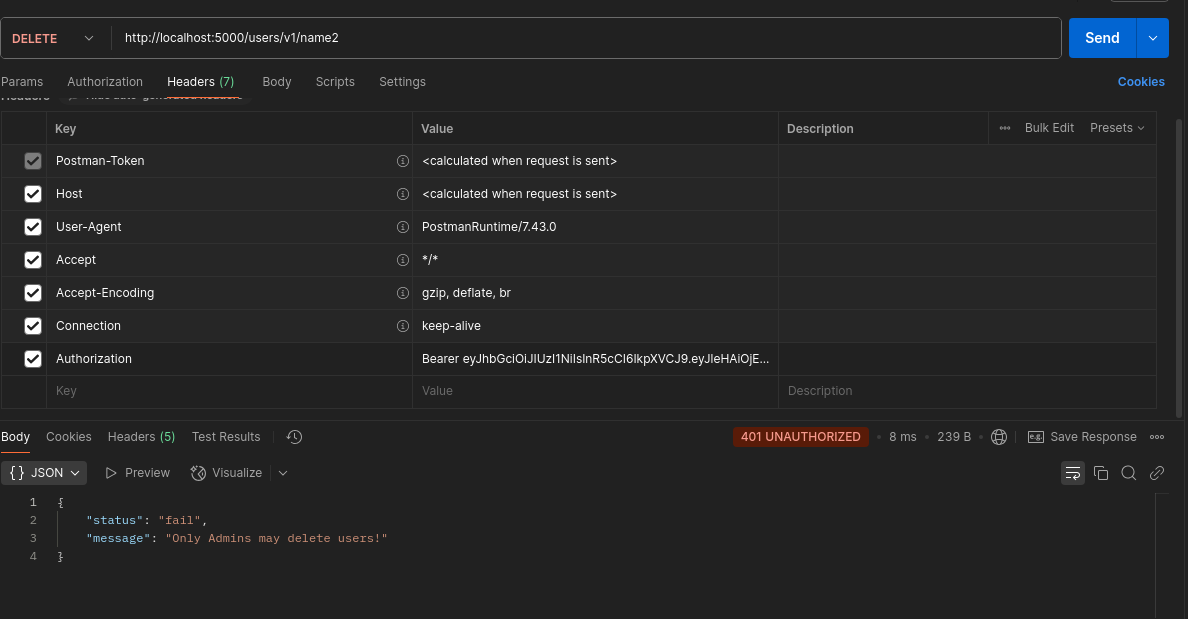
#### La vulnérabilité Broken Function Level Authorization se produit lorsque l'API ne vérifie pas correctement les permissions des utilisateurs avant d’autoriser l'exécution de certaines actions. Un attaquant peut ainsi appeler des endpoints réservés aux administrateurs ou modifier des ressources sensibles sans disposer des droits nécessaires.

#### Scénario d’attaque et exploitation

1. Accès non autorisé aux fonctions d’administration

L’objectif est de vérifier si un utilisateur classique peut exécuter des actions normalement réservées aux administrateurs.

Test réalisé : Supression d’un utilisateur



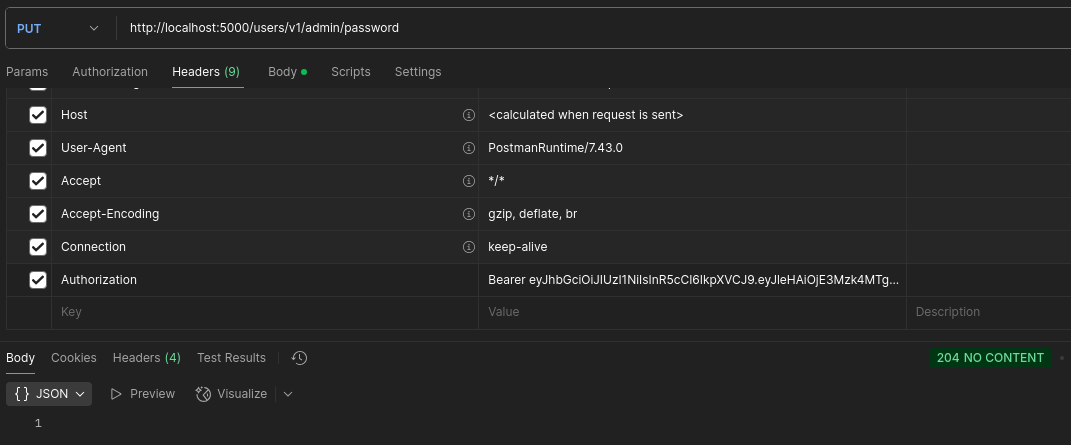
Réponse de l’API :

La supression n’a pas eu lieu,l’API a verifié cette autorisation.

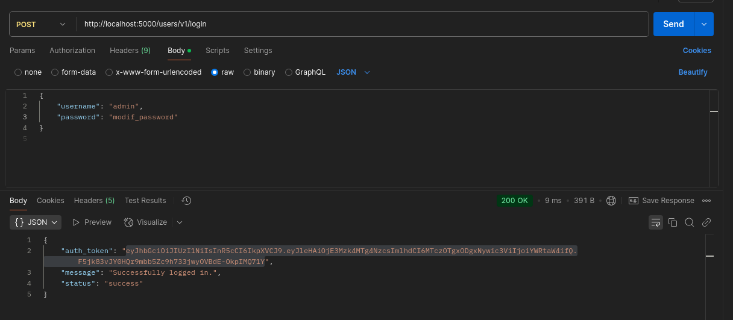
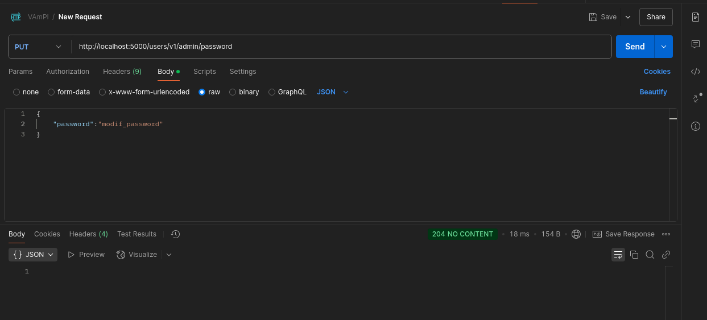
1. Exécution de commandes en tant qu’admin

L’objectif est de vérifier si un utilisateur non privilégié peut modifier le mot de passe d’un administrateur sans autorisation adéquate.

Test réalisé : Modifier le mot de passe d’un administrateur



Un utilisateur lambda tente de changer le mot de passe d’un administrateur en envoyant une requête de modification à l’endpoint concerné.



La modification a réussie, l’API accepte la requête et met à jour le mot de passe de l’administrateur sans vérifier si l’utilisateur a les privilèges nécessaires.

#### Impact des vulnerabilités :

Atteinte à l’intégrité des comptes :

Un attaquant peut modifier le mot de passe d’un administrateur, bloquant son accès légitime.

Prise de contrôle des comptes privilégiés :

En changeant le mot de passe, un attaquant peut s’octroyer un accès total aux fonctions administratives.

Risque de compromission totale :

Une fois en possession d’un compte admin, l’attaquant peut modifier des droits, supprimer des utilisateurs ou altérer la base de données.

Violation des bonnes pratiques de sécurité :

L’absence de vérification des privilèges dans cette action critique expose l’API à des abus.

### **2.2.6 Unrestricted Access to Sensitive Business Flows**

### Description

L'API VAmPI ne limite pas l'accès ou l'utilisation de certaines fonctionnalités critiques, permettant aux utilisateurs non autorisés d'exploiter ces fonctionnalités en masse. Cette absence de restriction peut entraîner une consommation excessive des ressources, une indisponibilité du service, ainsi que des abus sur des fonctions sensibles.

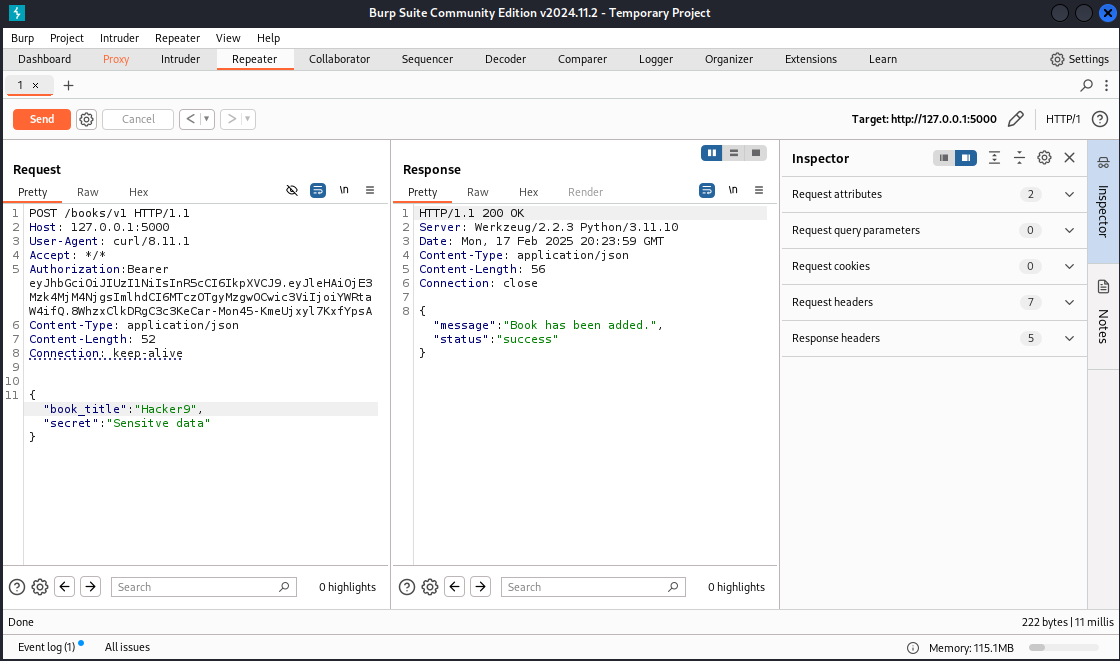
#### Scénario d’attaque et exploitation

1. Ajout illimité de ressources (Spam API)

L’objectif est de vérifier si un utilisateur peut ajouter un nombre illimité de livres sans restriction.

Test réalisé :

Un attaquant tente d’ajouter un grand nombre de ressources en automatisant des requêtes répétitives vers l’API.

  
  
Résultat de l’API :

L’API accepte un nombre illimité de requêtes sans restriction, alors la vulnérabilité est confirmée,aucune restriction n’empêche l’ajout massif de ressources, ce qui ouvre la porte à du spam d’API et peut surcharger la base de données.

#### Impact de la vulnérabilité :

Spam d’API :

Un attaquant peut remplir la base de données avec des millions d’entrées inutiles, compliquant la gestion et l'exploitation des données.

Épuisement des ressources :

Un trop grand nombre de requêtes simultanées peut surcharger le serveur et dégrader les performances de l’application.

Déni de service (DoS) :

L’accumulation excessive de requêtes peut rendre l’API indisponible pour les utilisateurs légitimes.

Risque de coût élevé :

Dans un environnement cloud, une surconsommation des ressources peut entraîner des coûts importants pour l’entreprise.

### 

### 2.2.7 Server-Side Request Forgery(SSRF)

#### Description

La vulnérabilité Server-Side Request Forgery (SSRF) permet à un attaquant d’exploiter une application web pour forcer le serveur à envoyer des requêtes HTTP non autorisées vers des ressources internes ou externes.

Dans le cas de l’API VAmPI, si des URL externes sont acceptées comme entrée pour récupérer ou traiter des données, un attaquant pourrait potentiellement les manipuler afin de :

Scanner le réseau interne et identifier des services accessibles.

Accéder à des ressources protégées, telles que des bases de données ou des interfaces d’administration.

Exfiltrer des informations sensibles stockées sur des services internes.

Contourner des restrictions d’accès en exploitant la confiance du serveur.

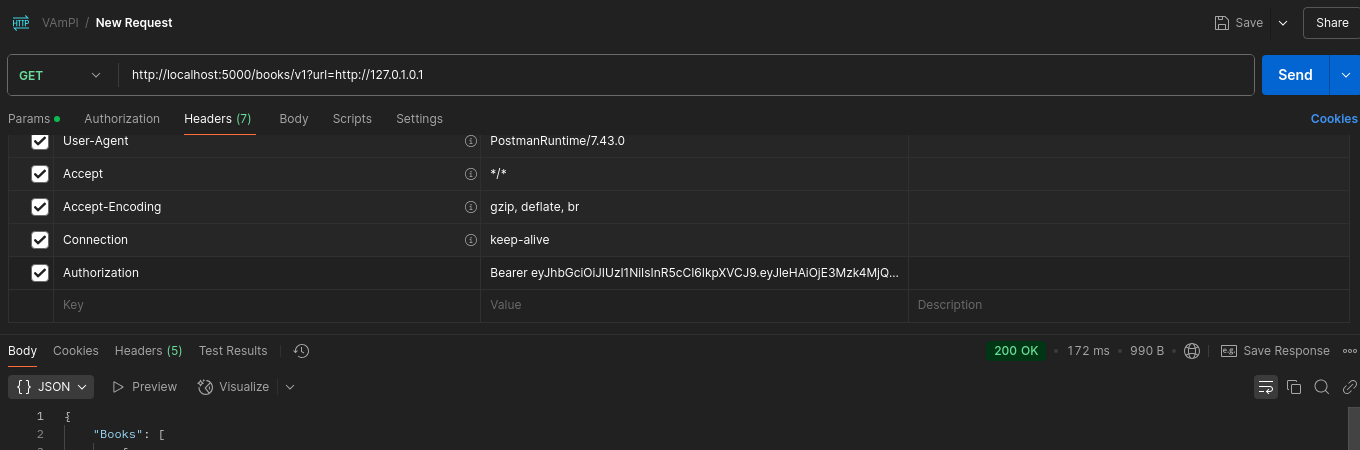
Utiliser l’API comme proxy pour mener des attaques contre d’autres systèmes.

Bien que nos tests n’aient pas permis d’exploiter cette vulnérabilité sur VAmPI, il est important d’expliquer pourquoi cette attaque peut constituer une menace et d’analyser les raisons pour lesquelles elle ne fonctionne pas dans ce cas précis.

#### Scénario d’attaque et exploitation

Pour évaluer la présence d’une vulnérabilité SSRF sur l’API VAmPI, plusieurs scénarios ont été testés en injectant des URL locales et externes dans les paramètres d’entrée.

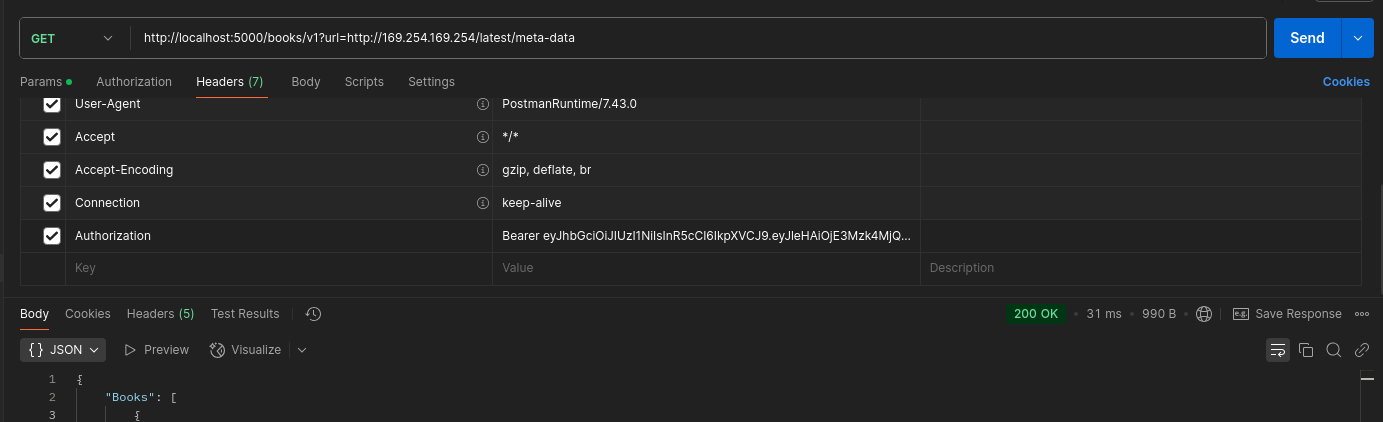
1. Accès à des ressources locales



Résultat de l’API :

L’application ignore la requête et retourne les books.

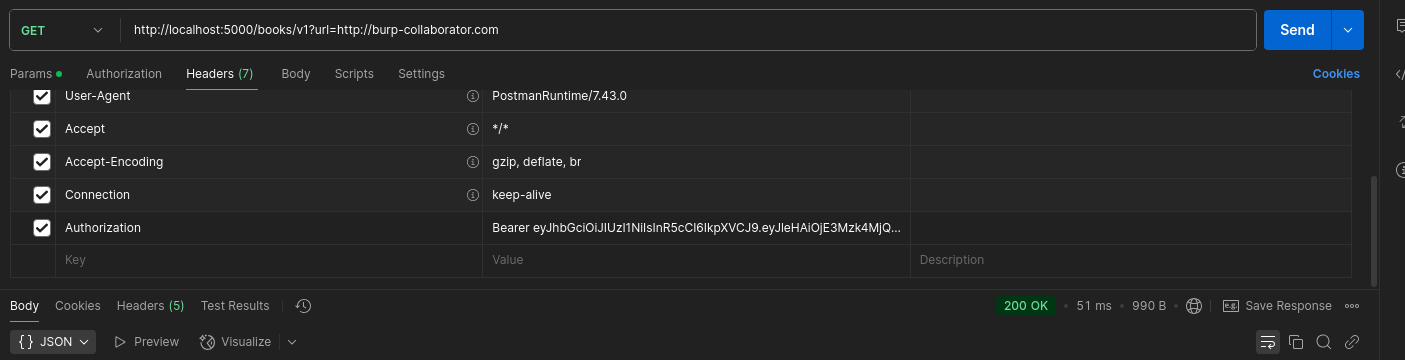
1. Tentative d’accès aux métadonnées cloud



Résultat de l’API :

L’application ignore la requête et retourne les books.

1. Tentative de communication externe avec Burp Collaborator



Résultat de l’API :

L’application ignore la requête et retourne les books.

#### Conclusion

Aucune preuve d’exploitation SSRF n’a été trouvée sur cette API, elle semble filtrer ou ignorer les URL externes et locales, empêchant ainsi les attaques SSRF classiques.

### 2.1.8 Security Misconfiguration

#### Description

La Security Misconfiguration regroupe un ensemble de mauvaises pratiques de configuration qui peuvent exposer l’API à des risques de compromission. Ces erreurs peuvent conduire à la divulgation d’informations sensibles, à une élévation de privilèges ou à une exploitation facilitée par un attaquant.Les exemples courants de Security Misconfiguration sont :

**- Fichiers et endpoints d'administration accessibles**

**- Messages d'erreur détaillés affichant des informations sensibles**

**- Clés API ou credentials exposés**

**- CORS mal configuré permettant l'accès depuis n'importe où**

**- Paramètres par défaut non sécurisés**

#### Scènes d’attaques et exploitation :

Nous allons chercher les mauvaises configurations possibles sur l'API VAmPI.

Test réalisés :

1. Vérification des headers http

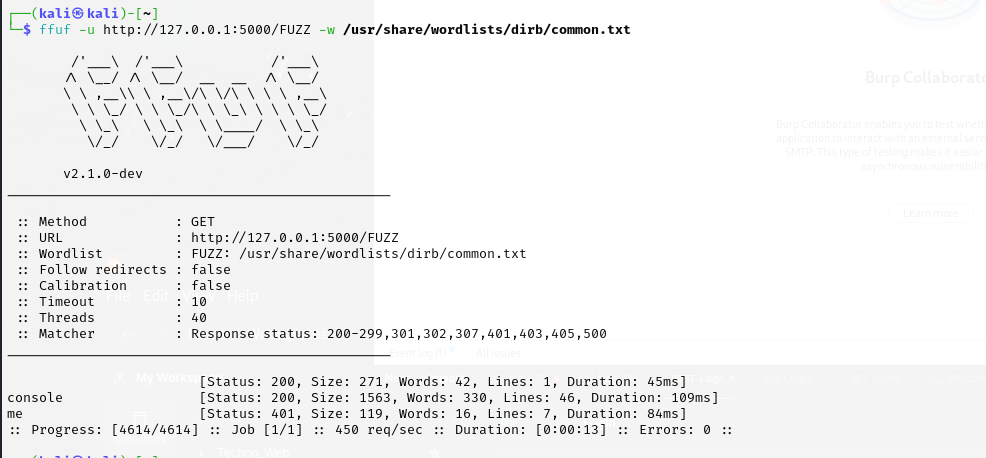
L’objectif de ce test est d’analyser les en-têtes HTTP de réponse pour identifier des configurations faibles qui pourraient révéler des informations sensibles.



Résultat de l’API :

Server → Révèle des informations sur le serveur web et sa version (Server: Werkzeug/2.2.3 Python/3.11.10).C’est une mauvaise configuration.

1. Fichiers ou endpoints exposés

Les fichiers de configuration exposés ou les endpoints cachés peuvent contenir des informations sensibles, comme des identifiants de base de données, des clés API ou des fichiers de sauvegarde.

/console :

Il est possible que ce soit un endpoint d'administration ou une interface de debug exposée.

/me :  
Cet endpoint est souvent utilisé pour récupérer les informations de l’utilisateur connecté.

#### Impact de la vulnérabilité :

Divulgation d’informations sensibles :  
L’exposition d’en-têtes HTTP détaillant la version du serveur ou du framework utilisé (ex: Werkzeug/2.2.3 Python/3.11.10) facilite l'identification des technologies en place. Un attaquant pourrait exploiter une faille connue associée à ces versions pour lancer une attaque ciblée (ex: exploitation de vulnérabilités CVE connues).

Exposition d’endpoints sensibles  
L’accès non restreint à /console peut permettre à un attaquant de manipuler le serveur si une console interactive est exposée. Si cet endpoint est un panneau d’administration, il pourrait offrir un accès total à l’API en cas de mauvaise gestion des permissions.  
L’endpoint /me, s'il est mal sécurisé, pourrait être exploité pour accéder aux données d’autres utilisateurs, révélant des informations personnelles ou permettant une élévation de privilèges.

Atteinte à la confidentialité et à l’intégrité des données  
L’accès non restreint à des endpoints sensibles peut permettre la modification ou la suppression de données critiques. Par exemple, si /console donne accès à une interface permettant d’exécuter du code, un attaquant pourrait exécuter des commandes arbitraires et compromettre l’ensemble de l’API.

### 2.2.9 Improper Inventory management

#### Description

Une gestion inadéquate des inventaires dans une API se manifeste par une mauvaise gestion des endpoints, des fonctionnalités cachées ou non documentées, et des données internes exposées. Cette vulnérabilité permet à un attaquant de découvrir et d'exploiter des fonctionnalités qui ne sont pas censées être accessibles publiquement.

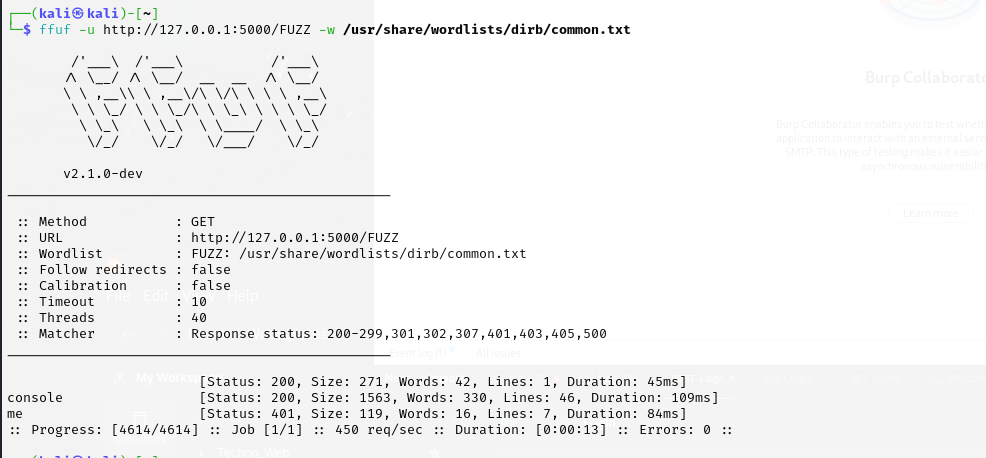
Dans le cadre de cet audit, nous avons cherché à identifier si des endpoints sensibles ou cachés étaient accessibles et exploités sans les contrôles de sécurité appropriés.

#### Scénario d’attaque et exploitation

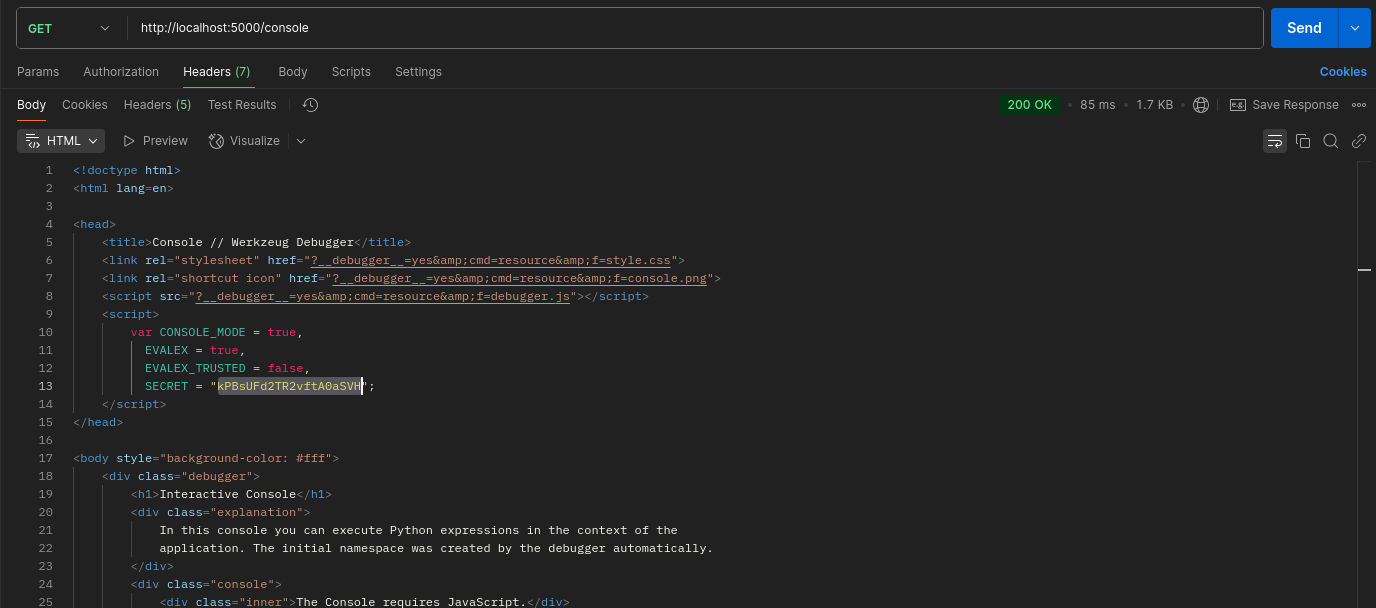
1. Recherche des endpoints non documentés

Test effecté :

En utilisant ffuf, on scanne l'API pour trouver des ressources cachées



Plusieurs endpoints ont été identifiés notamment console :

  
L’API expose une console de debug interactive, accessible publiquement via l’endpoint /console. Cet outil, généralement utilisé pour le développement et le débogage, permet d’exécuter du code Python à distance dans le contexte de l’application.  
Dans le cadre de notre audit, nous avons identifié que cette console est activée en production, ce qui représente une faille de sécurité critique.

En effectuant une requête GET vers http://localhost:5000/console, nous avons obtenu une page affichant l'interface du Werkzeug Debugger, accompagnée d’un mode évaluation active (EVALEX = true). De plus, une clé secrète (SECRET = "kPBsUFd2TR2vftA0aSVH") est exposée dans le code source.

1. Vérification des routes sensibles

Certains Endpoints permettent de récupérer des fichiers sensibles

Test effectué :



#### Impact de la vulnérabilité

Atteinte à la confidentialité des données  
Un attaquant pourrait exploiter ces endpoints cachés pour récupérer des informations sensibles telles que des comptes utilisateurs, des fichiers de configuration ou des clés API.

Compromission des systèmes internes  
La présence d’endpoints de debug ou d’administration accessibles en production peut exposer des fonctionnalités critiques permettant une prise de contrôle de l’API.

Élévation de privilèges  
Un attaquant pourrait exploiter ces routes pour bénéficier de droits plus élevés

### 2.2.10 Unsafe Consumption of API

#### Description

L’Unsafe Consumption of API (Consommation non sécurisée d’une API) désigne les failles résultant d’une **mauvaise validation des entrées** et d’une **absence de contrôle sur les données consommées par l’API**. Lorsqu’une API accepte des entrées non vérifiées provenant de sources externes, elle s’expose à divers types d’attaques, notamment l’SSRF :

#### Limitation de l’exploitation de l’attaque SSRF

L'attaque Server-Side Request Forgery (SSRF) permet à un attaquant d’exploiter une API pour lui faire envoyer des requêtes vers des ressources internes ou externes, souvent dans le but d’accéder à des informations sensibles ou d’exploiter des services non sécurisés.

Dans le cadre de cet audit, nous avons testé plusieurs scénarios d’exploitation de SSRF, mais aucune preuve d’exploitation réussie n’a été obtenue. L’API semble disposer de mécanismes de protection qui empêchent une exploitation directe.

**Conclusion** :  
L'attaque SSRF a été tentée, mais n’a pas abouti dans l’environnement testé. Cependant, cela ne garantit pas l'absence totale de vulnérabilité.

## 

## 

## 

## 3. Recommandations générales

Dans le cadre de l’amélioration de la sécurité de l’API VAmPI, il est impératif de mettre en place un ensemble de mesures correctives visant à renforcer la protection contre les vulnérabilités identifiées. Ces recommandations s’inscrivent dans une démarche de sécurité en conformité avec les meilleures pratiques du secteur et les standards de l’OWASP.

### 3.1 Mesures correctives prioritaires

Renforcement du contrôle d’accès et des permissions :  
Il est essentiel d’implémenter un contrôle strict des identifiants utilisateurs afin de prévenir les failles liées à Broken Object Level Authorization (BOLA). Pour cela, chaque requête doit être systématiquement validée en fonction du rôle et des autorisations de l’utilisateur authentifié.

Sécurisation de l’authentification :  
L’authentification doit être robuste et résiliente aux attaques en adoptant les pratiques suivantes :

Mise en place d’une authentification multifactorielle (MFA) pour limiter le risque de compromission des comptes.

Expiration et rotation des tokens JWT afin d’empêcher leur réutilisation indéfinie.

Utilisation de clés de signature JWT sécurisées et implémentation d’une vérification stricte des signatures.

Protection des données sensibles :  
Les données doivent être protégées à toutes les étapes de leur cycle de vie :

Chiffrement des données en transit avec TLS 1.2 ou supérieur.

Chiffrement des données stockées (au repos) pour éviter toute compromission en cas de fuite de base de données.

Utilisation de variables d’environnement pour stocker les secrets et credentials, plutôt que des fichiers statiques ou du code source.

Renforcement de la sécurité applicative :

Mise en place d’un pare-feu applicatif (WAF) pour filtrer et bloquer les requêtes malveillantes en amont de l’API.

Activation et surveillance des logs d’accès et d’audit pour détecter toute tentative d’exploitation.

Mise en place de mécanismes de rate limiting afin de limiter le nombre de requêtes par utilisateur et d’empêcher les attaques par bruteforce ou déni de service (DoS).

Désactivation du mode debug et des endpoints d’administration exposés en environnement de production.

Sécurisation de l’infrastructure et du déploiement :

Audit et durcissement des configurations de sécurité dans le fichier Docker Compose et les images Docker utilisées.

Application des principes du moindre privilège pour les containers et services associés.

Mise en place de scans de vulnérabilités réguliers sur les dépendances, les bibliothèques et les frameworks utilisés.

Renforcement des tests de sécurité :

Automatisation des tests de sécurité en intégrant des solutions comme OWASP ZAP, Burp Suite et SonarQube dans le pipeline CI/CD.

Audit de sécurité régulier basé sur l’OWASP API Security Top 10 - 2023 afin d’identifier et corriger les vulnérabilités émergentes.

Tests d’intrusion périodiques réalisés par des experts en cybersécurité pour évaluer la résistance de l’API face aux attaques avancées.

## Conclusion

En appliquant ces recommandations, l’API VAmPI pourra élever son niveau de sécurité et se conformer aux standards internationaux en matière de protection des données et des applications web. Une approche proactive de gestion des risques, combinée à une surveillance continue et des audits réguliers, garantira une protection efficace contre les menaces évolutives et les attaques ciblées.