Análise das vulnerabilidades OWASP API Security TOP 10 2019 no ambiente virtual crAPI

Analysis of OWASP API Security TOP 10 2019 vulnerabilities in the crAPI virutal environment

Guilherme Augusto Rodrigues da Silva Mota
Análise e Desenvolvimento de Sistemas
Fatec Baixada Santista – Rubens Lara

Resumo

Com a evolução das arquiteturas web e o aumento da capacidade de processamento do lado cliente, tornou-se evidente a necessidade de modularização das aplicações e uso de APIs como solução para problemas de acoplamento, sobrecarga e manutenibilidade. A integração entre back-end e front-end em um único repositório pode comprometer a disponibilidade de sistemas, tornando a arquitetura baseada em APIs uma alternativa mais segura e escalável. Entretanto, essa abordagem expõe a lógica de negócio e dados sensíveis, tornando as APIs um dos principais alvos de ataques. Este trabalho tem como objetivo analisar e explorar vulnerabilidades descritas no OWASP API Security Top 10 2019, por meio de testes controlados utilizando o ambiente crAPI em uma máquina virtual com Kali Linux. A metodologia adotada baseia-se na Avaliação de Risco do OWASP, e os testes práticos visam a identificação e exploração da falhas. Durante os experimentos, foram utilizadas ferramentas como Burp Suite para interceptação de requisições e análise de endpoints. A análise permitiu compreender os riscos associados e propor medidas mitigatórias para fortalecer a segurança de aplicações modernas baseadas em APIs.

Palavras-chave: Segurança da Informação. API. OWASP. Pentest. Aplicações Web.

Abstract

With the evolution of web architectures and increased client side processing power, the need for modularization and the use of APIs has become evident as a solution to issues such as tight coupling, overload, and poor maintainability. The integration of back-end and front-end in single repository may compromise system availability, making API-based architectures a more secure and scalable alternative. However, this approach exposes business logic and sensitive data, making APIs a primary target for attackers. This study aims to analyze and exploit vulnerabilities described in the OWASP API Security Top 10 through controlled tests using the crAPI environment deployed on a Kali Linux virtual machine. The adopted methodology is based on OWASP's Risk Assessment Model, and the practical tests focus on the identification and exploitation of the vulnerability. During the experiments, tools such as Burp Suite were used to intercept requests and analyze endpoints. The analysis enabled a better understanding of associated risks and the proposal of mitigation measures to enhance the security of modern API-based applications.

Keywords: Information Security. API. OWASP. Pentest. Web Applications.

1 INTRODUÇÃO

O crescimento exponencial no uso de APIs (Application Programming Interfaces) na arquitetura de software moderno tem redefinido o paradigma de desenvolvimento de aplicações web. Esta transição para arquiteturas baseadas em microsserviços trouxe benefícios significativos em termos de escalabilidade, manutenibilidade e desempenho. No entanto, esta mesma evolução expôs novas superfícies de ataque, transformando as APIs em alvos privilegiados para agentes maliciosos.

Este trabalho tem como objetivo principal analisar e explorar de forma prática quatro vulnerabilidades do OWASP API Security Top 10 2019, utilizando para isso o ambiente controlado crAPI (Completely Ridiculous API). Através da metodologia baseada no OWASP Risk Assessment Framework, buscou-se compreender os mecanismos de exploração, identificar vetores de ataque e propor medidas mitigatórias que possam fortalecer a segurança de aplicações baseadas em APIs.

A relevância deste estudo reside na necessidade crescente de compreensão prática das vulnerabilidades que afetam APIs, especialmente considerando o aumento na adoção desta arquitetura por organizações de diversos portes. A abordagem experimental adotada permite não

apenas a verificação teórica, mas a demonstração tangível de como falhas de implementação podem comprometer seriamente a confidencialidades, integridade e disponibilidade de sistemas.

2 FUNDAMENTAÇÃO TEÓRICA

Devido o advento da modularização de aplicações web, com a iniciativa de propor soluções para problemas de acoplamento relativos à abordagem tradicional na arquitetura de software, conveniente quando o poder de processamento no lado cliente era consideravelmente inferior quando comparado com o cenário atual (OLIVEIRA;RODRIGUES, 2021), tornou-se evidente que o uso das APIs em sistemas web é uma solução para problemas de acoplamento, sobrecarga, baixa disponibilidade de recursos e manutenibilidade comprometida.

Segundo Oliveira e Rodrigues (2021, p.3) “Com os códigos do back-end e front-end em um repositório único, tornam-se recorrentes as situações em que todo um sistema é retirado do ar por conta de um erro em somente uma dessas partes.”

As APIs (Application Programming Interface) são mecanismos que permitem a integração de dois ou mais softwares.

“Uma API pode ser entendida como um sistema que permite a comunicação entre dois ou mais sistemas, em que essa comunicação é conduzida por um protocolo.” (OLIVEIRA;RODRIGUES, 2021. p.9).

Por meio da implementação de APIs os programadores tem a possibilidade de comunicação entre diferentes sistemas, organizando o alto volume de dados que trafega entre eles.

No caso de aplicações web, as APIs utilizam o protocolo HTTP (Hyper Text Transfer Protocol) para a troca de dados em rede. A API Web interage com os dados através do fluxo das requisições HTTP. O fluxo de uma requisição HTTP passa por uma série de etapas antes de finalmente chegar ao servidor web de hospedagem da API.

No caso das APIs Web a resposta das requisições é comumente retornada em um formato representacional chamado JSON (JavaScript Object Notation).

Uma API Web é estruturada através de rotas, que são associadas com as tarefas, regras de negócio que um sistema possui, dentre outras funcionalidades.

Rotas em API Web utilizam métodos estabelecidos pelo protocolo HTTP, sendo os mais utilizados:

GET: Requisitar informações sobre um recurso específico da API. Chamadas feitas com o método GET apenas consultam e retornam dados.

POST: Enviar informações para um recurso da API. Nesse tipo de requisição, os dados são transmitidos no corpo da mensagem – geralmente no formato JSON. É comum em operações como cadastro e login, nas quais são enviados dados como e-mail e senha para validação.

PUT: Utilizado para atualizar completamente um recurso existente. Assim como o POST, o método PUT envia as informações no corpo da requisição, substituindo todos os dados antigos pelos novos.

PATCH: Atualização parcial de um recurso, sem a necessidade de modificar seus atributos. Os dados a serem alterados também são enviados no corpo da requisição.

DELETE: Responsável por excluir um recurso específico da API. Removendo permanentemente os dados associados a ele.

2.1 Open Worldwide Application Security Project (OWASP)

"O Open Worldwide Application Security Project (OWASP) é uma fundação sem fins lucrativos que trabalha para melhorar a segurança do software". (OWASP, 2025).

Em 2019 a organização publicou um trabalho sobre a importância da segurança de APIs, com foco em estratégias e soluções para a compreensão e mitigação de vulnerabilidades únicas associadas às APIs. (OWASP FOUNDATION, 2019)

"Por natureza, as APIs expõem a lógica dos aplicativos e dados sensíveis, inclusive, dados pessoais sensíveis, e por esta razão, as APIs vem se tornando cada vez mais alvo de atacantes. Sem APIs seguras, inovações podem se tornar impossíveis." (OWASP FOUNDATION, 2019).

"É importante realizar que durante os últimos anos a arquitetura das aplicações foram significativamente modificadas. Atualmente, as APIs representam um papel muito importante nessa nova arquitetura de microsserviços, *single page applications*, aplicativos móveis, IoT e etc. " (OWASP FOUNDATION, 2019).

De acordo com o WSTG da OWASP Foundation (2023), a adoção tecnologias como computação em nuvem, aplicativos de página única, contribuíram para a adoção de APIS, tornando necessária a realização de testes em API, podendo, caso contrário, fornecer um caminho para dados confidenciais.

Segundo o OWASP Web Security Testing Guide (2023) é possível encontrar Endpoints da aplicação através da navegação do aplicativo, expecionalmente quando a documentação está incompleta, utilizando a ferramenta escolhida nos testes como um proxie de interceptação. É possível identificar links com esquemas de nomenclaturas de URL de API facilmente identificáveis, tais como:

https://example.com/api/v1

Ou até mesmo subdomínios dos quais o aplicativo pode depender em:

https://api.example.com/api/v1

(WSTG, 2023)

3 METODOLOGIA

3.1 Ambiente de testes – crAPI – Completely ridiculous API (crAPI)

O ambiente de estudo utilizado nesta pesquisa foi o crAPI, uma aplicação web intencionalmente vulnerável, modelada como um sistema de serviços automotivos. Em termos funcionais, o WebApp permite que usuários registrem contas, gerenciem veículos, busquem e contactem mecânicos, enviem solicitações de revisão, participem de seções comunitárias com publicações e comentários (OWASP Foundation, 2024).

Por projeto, o crAPI não implementa práticas de segurança robustas: diversas funcionalidades estão propositalmente inseguras para viabilizar o aprendizado e experimentação em um contexto controlado. Essa característica torna o crAPI adequado para investigações acadêmicas e testes de segurança reproduzíveis, em ambiente isolado, sem risco a sistemas produtivos de terceiros.

Arquiteturalmente, o crAPI foi concebido como um conjunto de microsserviços (exemplo: serviço web, serviço de identidade, serviço de comunidade, serviço de oficina, mailhog), o que permite replicar cenários reais em que falhas de autenticação, exposição excessiva de dados (entre outras), consumo de recursos e limite de taxa, podem ocorrer em pontos distintos da malha de serviços. (OWASP Foundation, 2024).

Os testes foram realizados em ambiente controlado utilizando a API vulnerável **crAPI** (instância local em Docker) e o Burp Suite Community Edition.O Burp Suite foi usado como proxy

HTTP(S) para interceptação e modificação de requisições, e como ferramenta de repetição e automação (Repeater e Intruder) para validar falhas.

Para cada caso de teste foi capturada a requisição raw (HTTP request) e a resposta raw (HTTP Response) via Burp; Os artefatos foram salvos e anexados como evidência (prints das telas do Burp). Os procedimentos seguem as diretrizes do OWASP API Security Top-10 (2019) e do OWASP Web Security Testing Guide - seção de API Testing.

Para a análise de riscos e vulnerabilidades em APIs, optou-se por utilizar como base o OWASP API Security Top 10 – 2019. Apesar da existência de versões mais recentes (como a de 2023), a escolha fora alinhada com o ambiente de testes utilizado neste trabalho – o crAPI(Complete Ridiculous API) -- que fora desenvolvido com base na classificação de 2019. Esta decisão visa manter consistência entre os conceitos teóricos e práticas experimentais aplicadas.

Figura 1 – Classificação de Risco do OWASP API 2019

Agentes de Ameaça	Explorabilidade	Prevalência da Fraqueza	Detecção da Fraqueza	Impacto Técnico	Impacto ao Negócio
Específico da API	Fácil: **3**	Difundida **3**	Fácil **3**	Severo **3**	Específico do negócio
	Médio: **2**	Comum **2**	Média **2**	Moderado **2**	
	Difícil: **1**	Difícil **1**	Difícil **1**	Menor **1**	

Fonte: OWASP, 2019.

Este trabalho baseia-se na metodologia do OWASP Risk Assessment Framework, previamente utilizada para classificação e avaliação das vulnerabilidades do OWASP API Security Top 10.

O OWASP classifica os riscos presentes nas APIs, considerando principalmente os aspectos: Explorabilidade, Prevalência, Detecção, Impacto Técnico e Impacto ao Negócio. Cada critério recebe uma pontuação de 1 (baixo) a 3 (alto)

3.2 API1-2019: BOLA - Broken Object Level Authorization

Desafio 1 – Acessar detalhes do veículo de outro usuário:

- Como os IDS de veículos não são números sequenciais, mas GUIDs, você precisa encontrar uma maneira de expor o ID do veículo de outro usuário.

- Entre um ponto de extremidade de API que receba um ID de veículo e retorne informações sobre ele.
-

Desafio 2 – Acessar relatórios mecânicos de outros usuários

- Analisar o processo de envio do relatório.
- Encontre um ponto de extremidade de API oculto que exponha detalhes de um relatório mecânico.
- Alterar o ID do relatório para acessar outros relatórios.

Fonte: (OWASP FOUNDATION,2024)

Execução dos testes:

1**.** Identificação do Endpoint Refresh Location para a obtenção de UID/GUID de usuário;

2.Interceptação da requisição do EndPoint “Refresh Location” para verificar retorno de informações de veículo.

3.Interceptação da Aba Comunnity que expõe UID/GUID de um veículo.

4.Copiar vehicleId da aba Community e substituir no Repeater.

5.Verificar se a response revela dados do veículo / geolocalização de outro usuário.

6.Resultado esperado de vulnerabilidade: retorno de dados sensíveis sem autorização. (substituição do UUID permitiu ver a geolocalização de “Pogba”.)

API2-2019: Broken Authentication

Desafio 3 – Redefinir a senha de um usuário diferente

- Encontre um endereço de e-mail de outro usuário no crAPI
- Força bruta pode ser a resposta. Se você enfrentar algum mecanismo de proteção, lembre-se de aproveitar a natureza previsível das APIs REST para encontrar Endpoints de API mais semelhantes.

Fonte: (OWASP FOUNDATION,2024)

Execução dos testes:

1. Mapear o fluxo de redefinição de senha: Localize a requisição que inicia o fluxo de redefinição e a requisição do OTP e observe headers, body (e-mail), versionamento v=2/v=3 e resposta de erro quando inválido.
2. Investigar controle de versão / limitação de taxa: requisições de navegação normalmente usam a versão 2, mas o endpoint de redefinição estava com versão 3.
3. Modificar o header/parametro de versão na requisição interceptada (alterar X-API-Version: 3 para X-API-Version: 2) no Repeater e reenvie para verificar comportamento e códigos de erro relacionados a rate-limit. (teste exemplificou sucesso em permitir múltiplas tentativas).
4. Navegar até a aba Community e encontrar requisições/responses que exponham e-mail ou credenciais parciais de outros usuários e copiar o e-mail alvo a ser testado.
5. Gerar/encaixar OTP para outro usuário: Usando a requisição de verificação de OTP (préviamente enviada para o Repeater), substitua o campo email pelo e-mail alvo. Se o enpoint aceitar e gerar/validar OTP para esse e-mail será possível automatizar tentativas.
6. Automatizar brute-force do OTP com Intruder: Envie a requisição de verificação OTP (com o e-mail alvo) para o Intruder.
7. Configure o payload: Payload type: Numbers, Range: 7000 -> 8000, Min/Max digits: 4. Em Settings: Grep-Match: Limpar a lista e adicionar “Invalid OTP!” (ou a mnesagem de erro observada) para filtrar respostas; quando a mensagem de erro não aparecer, a tentativa pode ter sido bem-sucedida.

3.3 API3-2019: Broken Object Level Authorization

Desafio 4 – Redefinir a senha de um usuário diferente

- Encontre um endpoint de API que vaze informações confidenciais de outros usuários

Desafio 5 – Propriedades internas de um vídeo:

- Localize a área de upload/download de vídeo

Fonte: (OWASP FOUNDATION,2024)

Execução dos testes:

1. Interceptar a requisição da página Community; Verificar se a aba expõe e-mail/credenciais de outros usuários.

2. Localizar a área de upload/download de vídeo, interceptar a requisição de download/upload.
3. Analisar a Response no Repeater: Campos internos do vídeo devem aparecer (metadados do back-end, caminhos internos, propriedades não destinadas ao cliente).
4. Resultado observado: download interceptado revela propriedades internas do vídeo.

API4-2019: Lack of Resources & Rate Limiting

Desafio 6 –Executar um DoS de camada 7 usando o recurso “contact_mechanic”
Fonte: (OWASP FOUNDATION,2024)

Execução dos testes:

1. Localizar a requisição “contact_mechanic” e interceptar com o Burp proxy observando campos numéricos de repetição (ex: number_of_repeats) e campos booleanos que controlam retentativas.
2. Enviar a requisição para manipulação dos parâmetros no Repeater, alterando “number_of_repeats” de “1” para “1000” repetições. Mudar o parâmetro booleano “repeat_request_is_failed” para “true”.
3. Verificar códigos de erro, atrasos de respostas, e tempo até apresentação de instabilidade.

4 ANÁLISE E RESULTADOS

4.1 API1-2019: BOLA -Broken Object Level Authorization

O nível de autorização de objeto é um mecanismo implementado para determinar os níveis de permissão de acesso de um usuário. Os endpoints de API que recebem IDs de objetos devem validar se o usuário tem permissão para executar ações no objeto. (OWASP FOUNDATION, 2019)

No contexto dos testes realizados, essa vulnerabilidade consiste em mudar o ID do usuário para verificar informações sensíveis, ou que não possuam autorização, navegando por todo

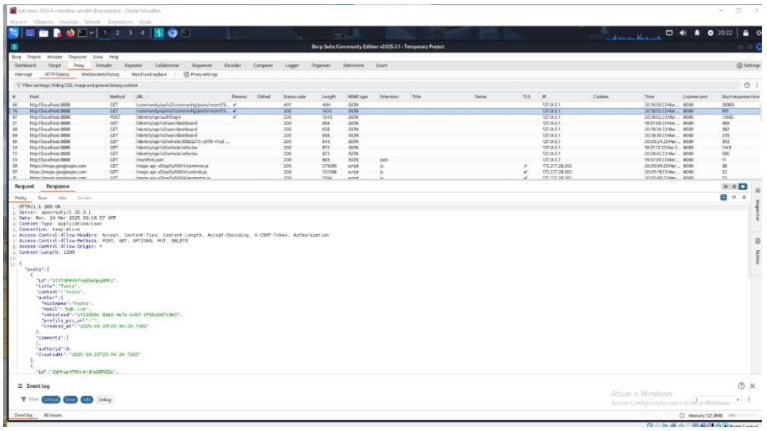
aplicativo verificando se algum endpoint está vazando algum tipo de informação substituindo o UUID/GUID pelo do próprio usuário criado na aplicação disponibilizada pelo crAPI.

Figura 2 – Response da requisição feita para o endpoint contendo informações do veículo.

Fonte: Elaborado pelo autor

Após isso fora obtido o ID de outro usuário, através do corpo de uma requisição da aba Community do aplicativo crAPI

Figura 3 – Identificando um EndPoint que revele o ID de outro usuário através de um UID ou GUID de veículo, na Aba Community.



Fonte: Elaborado pelo autor

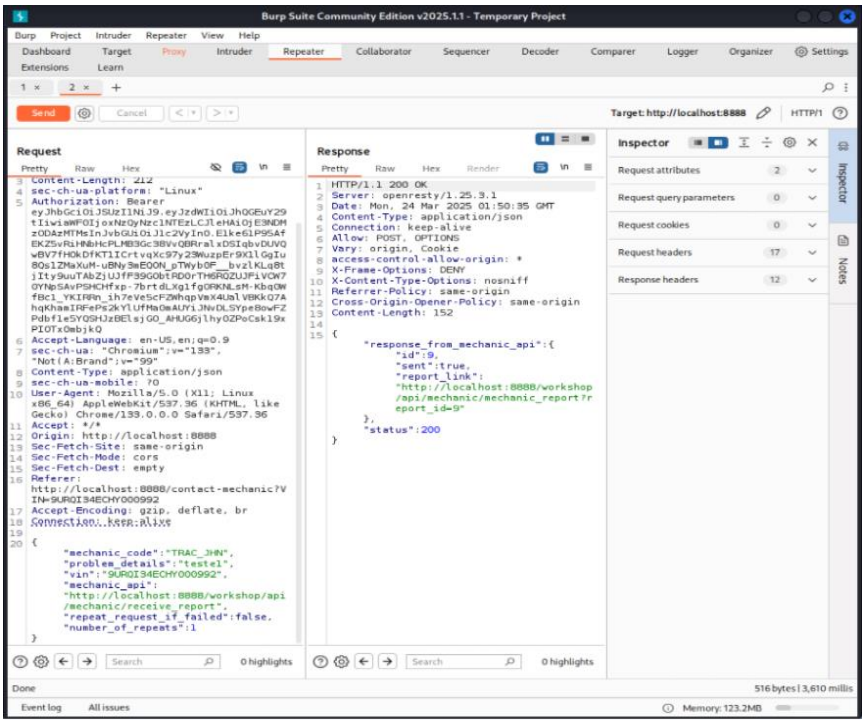
O ID obtido de outro usuário, no caso o usuário Pogba, fora copiado para substituir o ID do usuário criado no corpo da requisição do endpoint “Refresh Location”, tornando possível visualizar as informações, assim como a geolocalização do usuário através da modificação da requisição.

Após fora obtida a geolocalização e informações de outro usuário, através da modificação da requisição, completando o *Desafio 1*.

Desafio 2 – Acessar o relatório mecânico de outros usuários:

No Desafio 2 fora necessário analisar o processo de envio de envio de relatórios mecânicos para o encontrar o endpoint oculto da API.

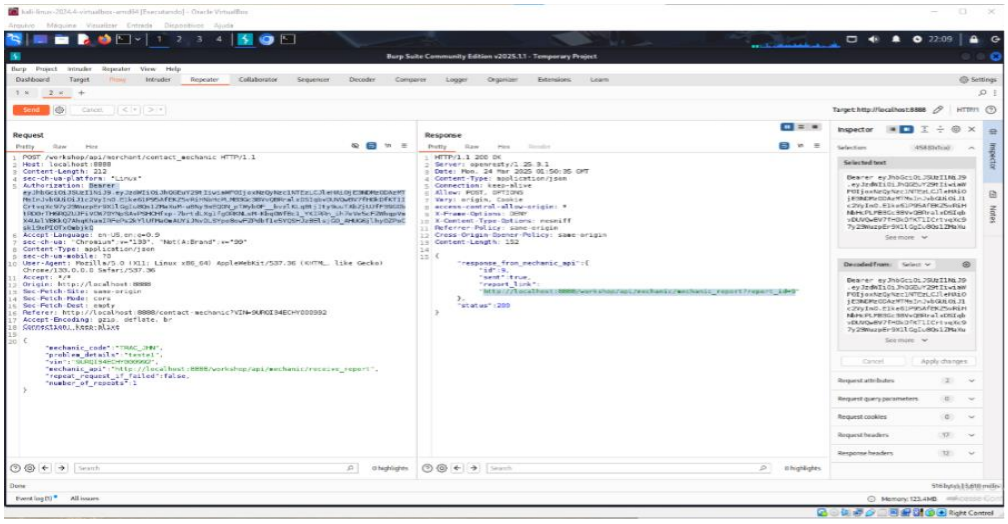
Figura 4 – Análise da Response da “Requisição de Serviço” no Repeater do Burp Suite.



Fonte: Elaborado pelo autor

Ao analisar a response https desse endpoint é possível obter os detalhes adicionais do enpoint que incluem o **report_id.** Ao acessar o report link no navegador fora possível descobrir que a aplicação utiliza **tokens JWT** para autenticação, o que tornou-se necessária a manipulação das requisições para burlar esse mecanismo de autenticação.

Figura 5 – Obtendo “Autorization bearer” na requisição do relatório mecânico.

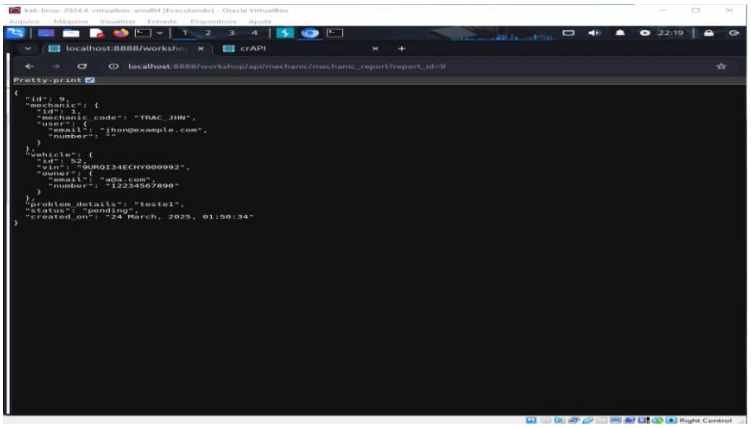


Fonte: elaborado pelo autor.

Figura 6 – Colando “Authorization” interceptada na página de relatório mecânico no header da Requisição de contatar serviço mecânico.

Fonte: elaborado pelo autor.

Figura 7 – Obtenção do próprio relatório mecânico.

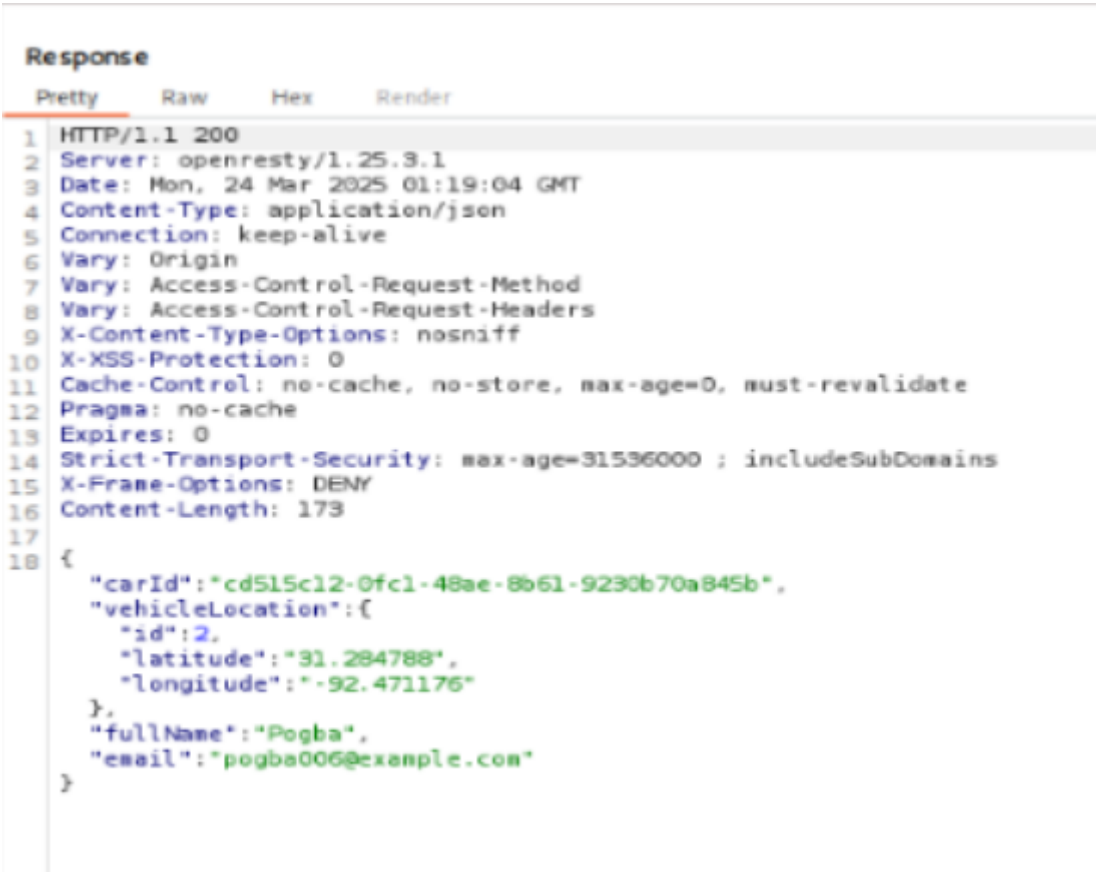


Fonte: Elaborado pelo autor.

Após isso, ao mudar o número do report_id na URL do navegador e repetir o processo de manipulação da “Authorization” será possível obter informações sensíveis de outro usuário através do relatório mecânico, tais como número de telefone, email, ordem de serviço e data de criação, ou seja, apenas alterando o ID de relatório obtemos informações confidenciais de outro usuário, completando assim, o Desafio 2.

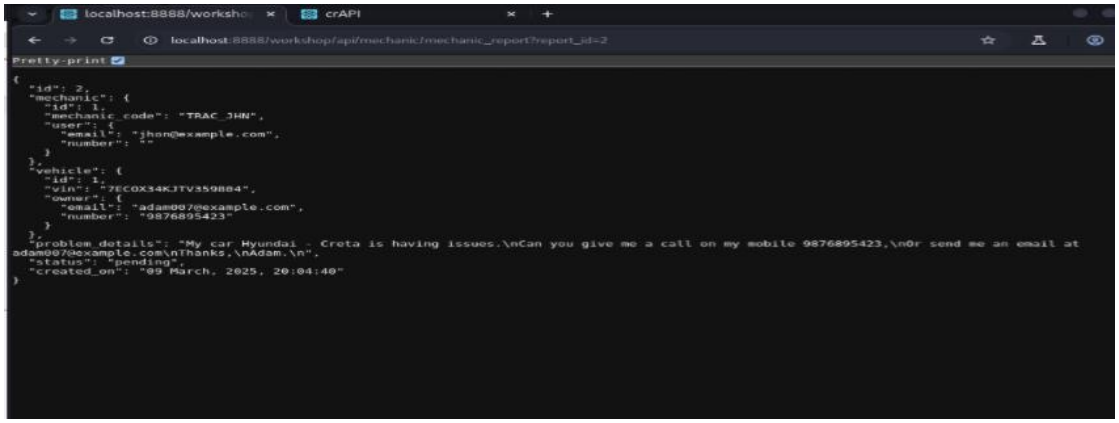
Através da execução do experimento fora possível concluir que aplicações que não diferenciam níveis de permissão podem expor dados sensíveis, pois através da substituição de id na em um endpoint que revela localização, obtivera-se a geolocalização de outro usuário de forma fácil. Porém, no desafio 2 após a identificação do mecanismo de autorização, exigiu-se um maior conhecimento técnico relacionados com tokens JWT para manipular a requisição e obter o relatório mecânico de outro usuário. Dessa forma, torna-se evidente a necessidade de verificar todos os endpoints da aplicação, analisando quais informações estão disponíveis para um possível atacante.

Figura 8 – Obtenção da geolocalização e infos de outro usuário através da modificação da requisição



Fonte: Elaborado pelo autor

Figura 9 – Obtenção de informações sensíveis de usuário através da manipulação de ID na url do relatório mecânico de outros usuários.



Fonte: Elaborado pelo autor

4.2 API2:2019 Broken User Authentication

Falhas de autorização e autenticação em aplicativos, podem permitir acesso não autorizado ou indesejado a determinados recursos, o que significa que um usuário comum pode abusar da funcionalidade de autenticação do usuário, sendo capaz de ignorar certas verificações de segurança e obtendo a oportunidade de acesso a conta de outro usuário.

É muito comum que engenheiros de software possuam diferentes concepções na implementação de mecanismo de autenticação, fazendo com que muitas vezes careçam da proteção adequada de acordo com diversos fatores que influenciam na mitigação de riscos, tais como:

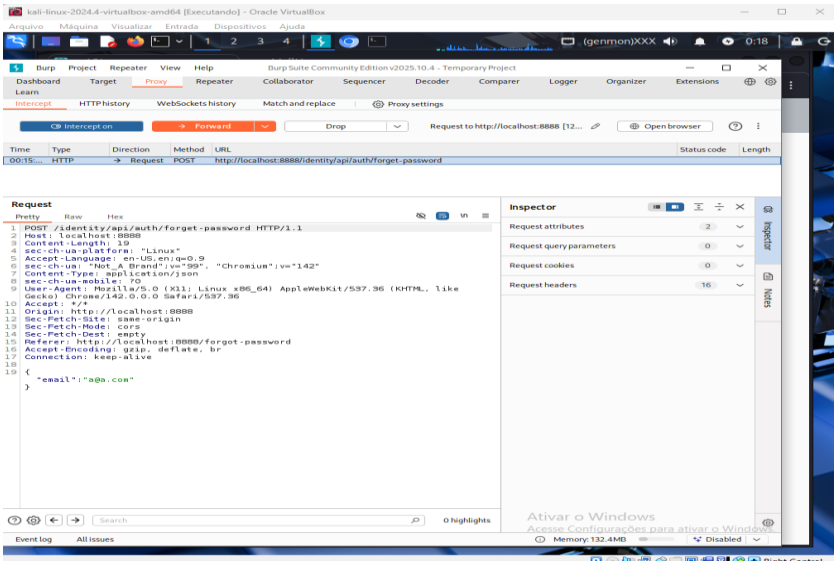
- Chance da realização de ataques de força-bruta para uma conta, onde não há bloqueio por excesso de tentativas de autenticação falhadas;
- Informações de autenticação como tokens e passwords presentes nas urls;
- Falta de validação de autenticidade de tokens de autenticação;

Desafio 3 - Redefinir a senha de um usuário diferente

O Endpoint explorado no desafio fora o de forget-password.

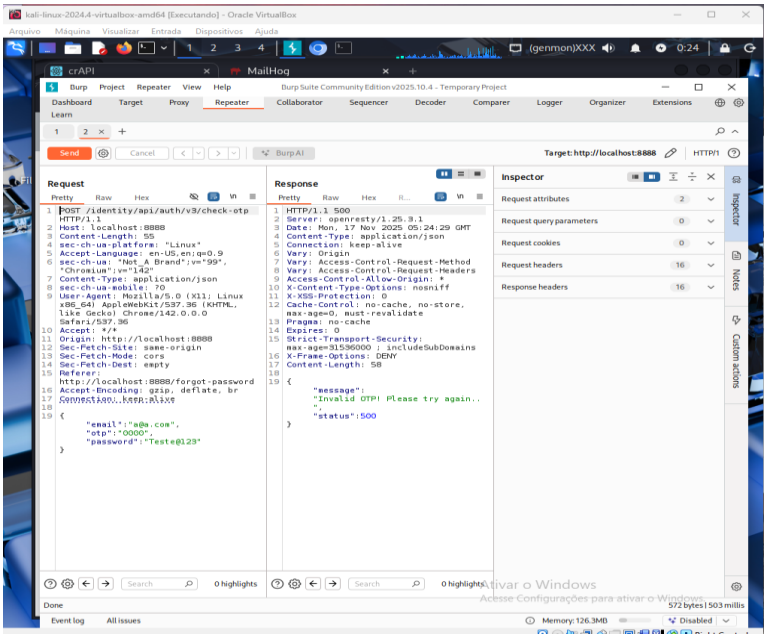
A Request interceptada deste endpoint fora enviada para o Repeater do Burp Suite para tratamento posterior, pois fora possível observar que por padrão o aplicativo inclui o endereço de email na solicitação de verificação OTP.

Figura 10 – Endpoint “forget-password”.



Fonte: Elaborado pelo autor

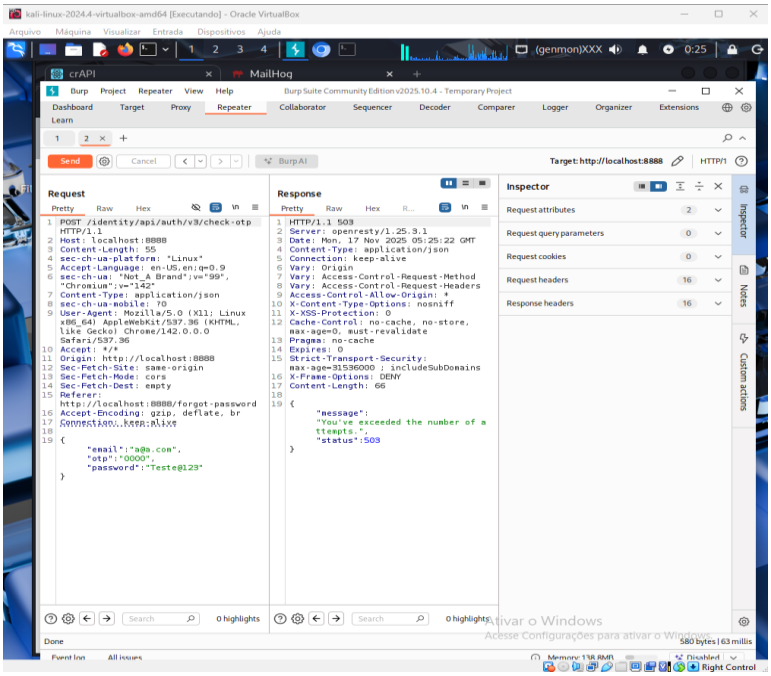
Figura 11 – Requisição de redefinição de senha com OTP gerado e enviado para email cadastrado e código de status 500.



Fonte: Elaborado pelo autor

Fora possível observar que ao enviar repetidas tentativas existe um mecanismo de limitação e taxa implementado na requisição de checagem de OTP.

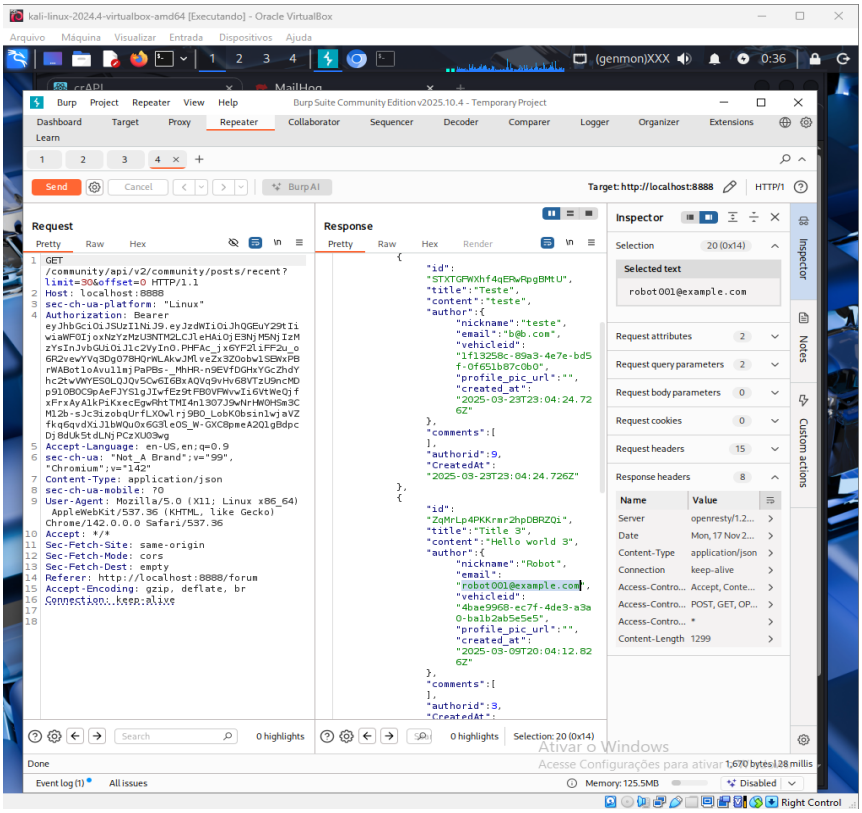
Figura 12 – Verificação de mecanismo de limitação de taxa com código de status 503.



Fonte: Elaborado pelo autor

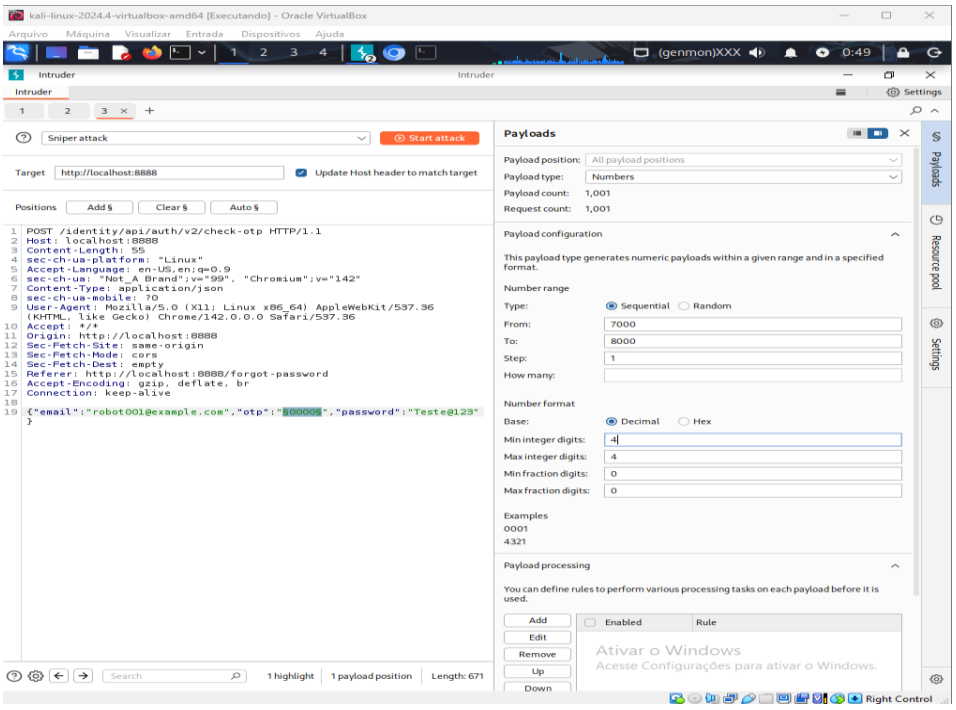
Obtendo o email de outro usuário na requisição da aba Community, é possível gerar também o OTP para este usuário em particular para que seja realizada a mudança de sua senha através de um brute-force attack que envolve a manipulação do versionamento para burlar o mecanismo de limitação de taxa e realizar diversos envios para o servidor, utilizando a requisição do endpoint de verificação de OTP armazenada no Repeater anteriormente.

Figura 13 – Obtendo email de outro usuário na aba Community



Fonte: Elaborado pelo autor

Figura 14 – Planejamento de ataque de força-bruta com o Burp Intruder

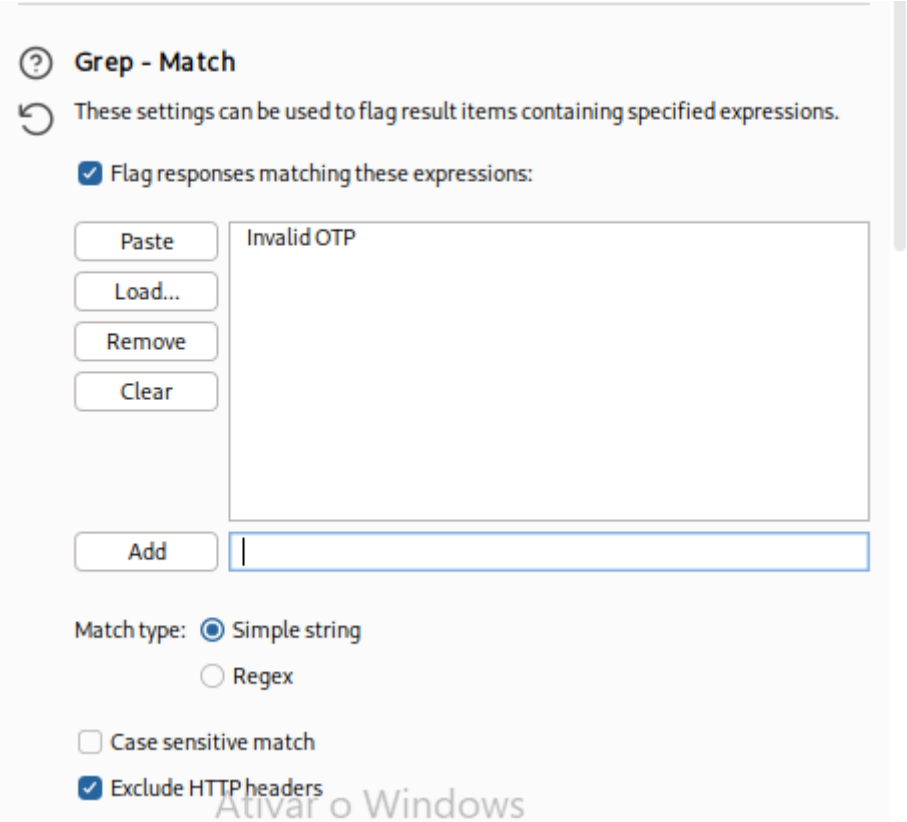


Fonte: Elaborado pelo autor

O versionamento fora alterado para v2. Nas opções do Payload será necessário o tipo de Payload type para Numbers, assim como ajustar o range de 7000 para 8000 pois será uma verificação em lotes de variação de dígitos, para não estender muito o tempo do teste de payload.

O número mínimo e máximo de dígitos fora colocado em 4 por conta do OTP possuir apenas 4 digitos. Para visualizar mensagens de erro, na aba settings/Grep -Match, fora limpada a lista e adicionada a mensagem de erro “Invalid OTP” para filtrar o erro.

Figura 15 – Filtragem de erro com o Grep – Match do Burp Intruder

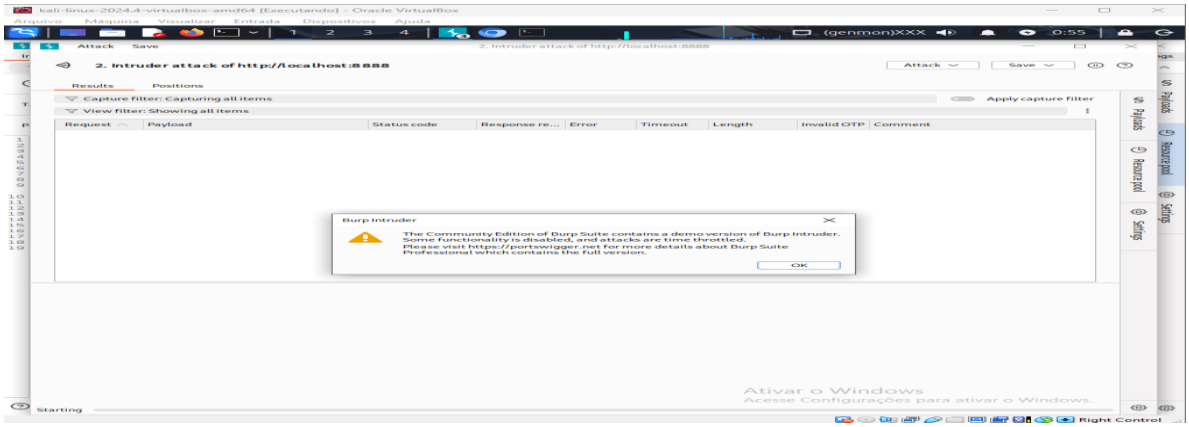


Fonte: Elaborado pelo autor

Se obter o OTP for bem sucedido a senha para o usuario em questão (robot001@example.com) irá ser automaticamente alterada como a opção que fora colocada (Teste@123).

O OTP com sucesso deve ter o código de status 200.

Figura 16 – Mensagem do Burp Suite sobre limitação da versão demo do Burp Intruder



Fonte: Elaborado pelo autor

4.3 API3:2019 – Excessive Data Exposure

A vulnerabilidade API3:2019 – Excessive Data Exposure ocorre quando a API retorna mais dados do que deveria, incluindo informações sensíveis ou pessoais (PII – Personally Identifiable Information), confiando que o cliente (como o app mobile ou frontend web) irá filtrar esses dados corretamente. Isso é um erro de segurança grave.

Ameaça: Qualquer pessoa que possa interceptar as respostas da API (ex: usando proxy, sniffers como Burp Suite ou mitmproxy).

Vetor: O atacante intercepta a comunicação e vê os dados que a API retorna, mesmo que o frontend oculte ou filtre o conteúdo

Os desenvolvedores criam endpoints genéricos, por exemplo, retornando objetos inteiros com toJSON(), sem considerar o que deve ou não ser exposto.

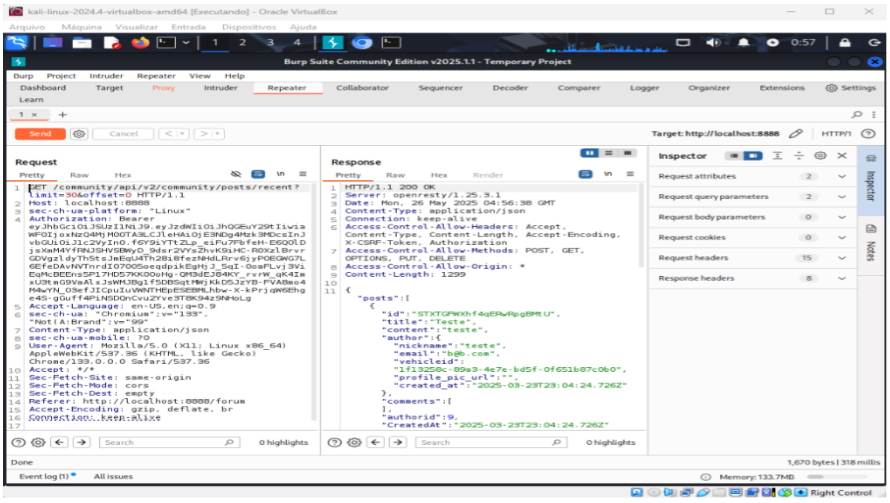
Muitas APIs foram feitas para servir como fonte de dados para múltiplos front-ends, e isso encoraja práticas inseguras como retornar tudo e deixar o cliente decidir o que exibir.

Ferramentas automáticas raramente detectam isso, porque não conseguem saber se os dados são sensíveis sem conhecimento de negócio.

Desafio 4 – Encontrar um Endpoint de API que vaza informações confidenciais de outros usuários

O desafio 4 pode ser completo apenas interceptando a requisição da página Community.

Figura 17 – Requsição da página Community enviada para o Repeater vazando informações sensíveis de outros usuários.

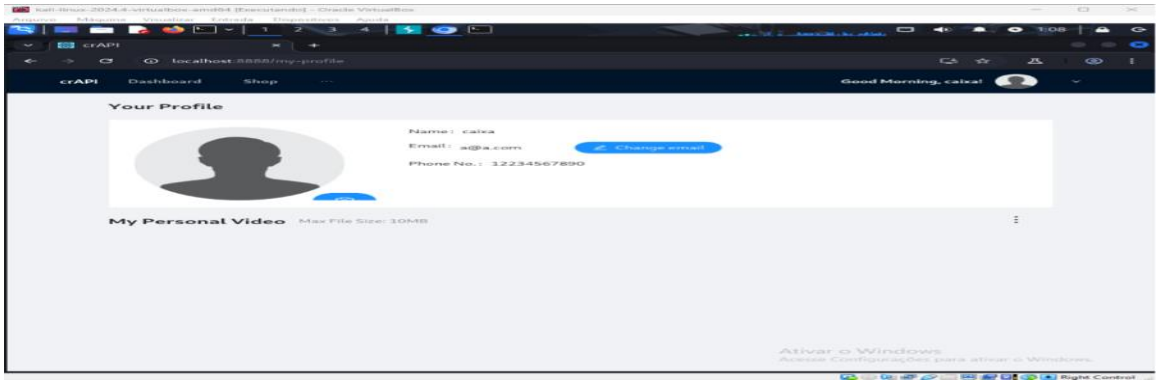


Fonte: Elaborado pelo autor

Desafio 5 – Encontrar um endpoint de API que vaza uma propriedade interna de um vídeo

Para a realização do Desafio 5 fora necessário identificar uma área de upload de vídeo na API.

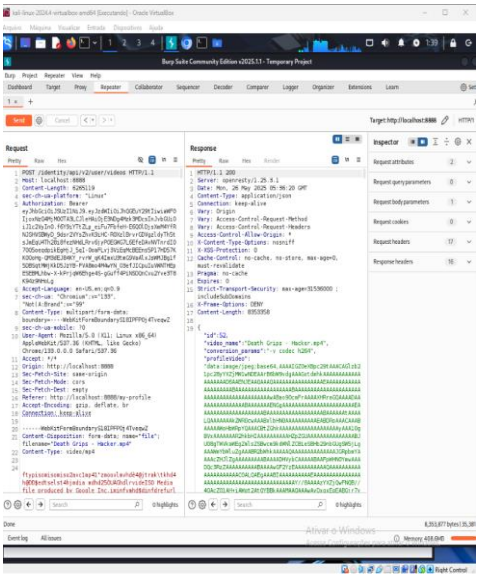
Figura 18 – Área de upload de vídeo na sessão MyProfile do crAPI.



Fonte: Elaborado pelo autor

Ao realizar o download de um vídeo da música “Hacker” do conjunto Death grips na área, fora interceptada a requisição com o Burp Suite e enviada para o Repeater, revelando assim propriedades internas de um vídeo.

Figura 19 – Visualizando propriedades internas de um vídeo através da Response da requisição de upload de vídeo.



Fonte: Elaborado pelo autor

Através da execução do experimento 3 e por conta da alta exposição de dados de usuário na aba Community, tornara-se evidentemente simples identificar a falha relacionada com o desafio, pois este resultado já havia sido identificado nos outros experimentos. Fora possível também verificar as propriedades internas de um arquivo .mp4 através da exploração de um Endpoint de upload presente na API.

4.4 API4:2019 – Lack of Resources & Rate Limiting

Quando uma API recebe requisições, as chamadas usam partes de recursos do servidor, tais como CPU, memória, armazenamento temporário e até mesmo número de conexões simultâneas disponíveis. Se muitas requisições chegam ao mesmo tempo, especialmente de vários clientes diferentes, todos começam a disputar esses recursos. Em cenários de alta demanda ou falta de configuração adequada, esse compartilhamento pode resultar em degradação de desempenho, instabilidade do serviço e até indisponibilidade.

A robustez de uma API está relacionada com o modo como seus parâmetros operacionais são definidos, como o tempo máximo de execução permitido por requisição (timeout), o limite de memória alocada, o **número máximo de requisições por cliente**. A ausência desses controles ou sua configuração inadequada abre margem para falhas e abusos.

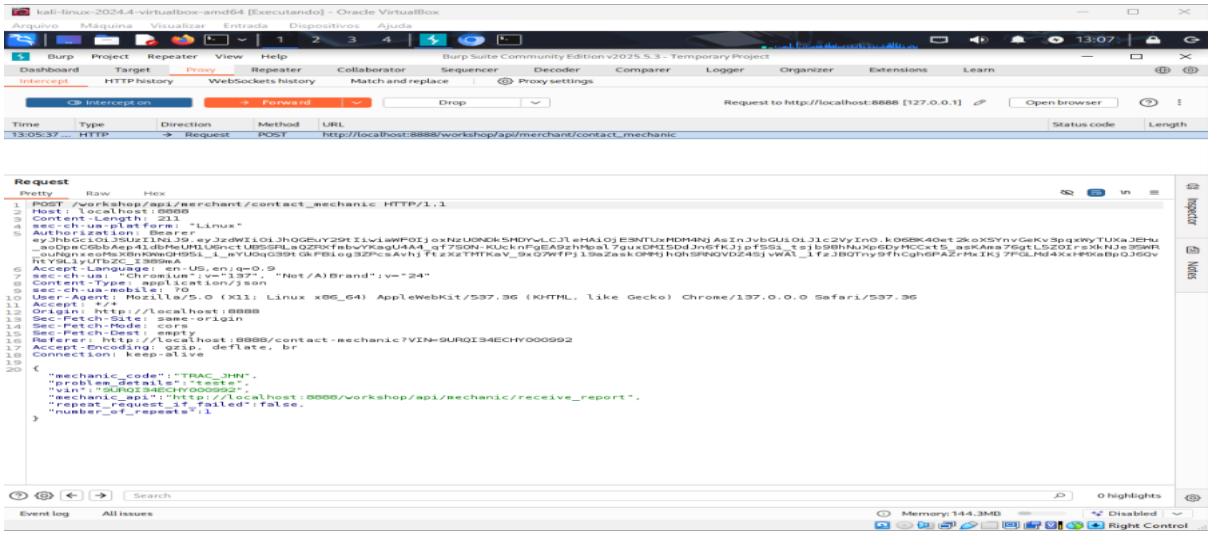
Nesse contexto, o **Rate Limiting** constitui um mecanismo essencial de proteção. Ele estabelece limites para o número de requisições de um usuário, cliente ou endereço IP pode realizar dentro de um intervalo de tempo específico. Essa técnica impede o consumo excessivo de recursos e reduz a superfície de ataque, garantindo maior estabilidade operacional.

Desafio 6 - Execute um DoS de camada 7 usando o recurso mecânica de contato

Basicamente a camada 7 de modelo OSI é a camada de aplicação.

Um Layer 7 DoS refere-se a um ataque na camada de aplicação do modelo OSI onde o usuario malicioso sobrecarrega o servidor da aplicação enviando uma grande quantidade de requisições HTTP ao mesmo tempo que acabam por consumir os recursos da aplicação.

Figura 20 – Intercepção da requisição do Endpoint “contact_mechanic”.



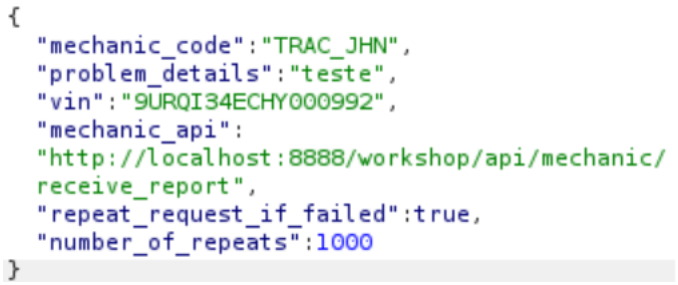
Fonte: Elaborado pelo autor

A requisição fora enviada posteriormente para o Repeater para a manipulação do parâmetro “number_of_repeats” e “repeat_request_if_failed” para “=true”.

Figura 21 – Response da requisição “contact_mechanic”.

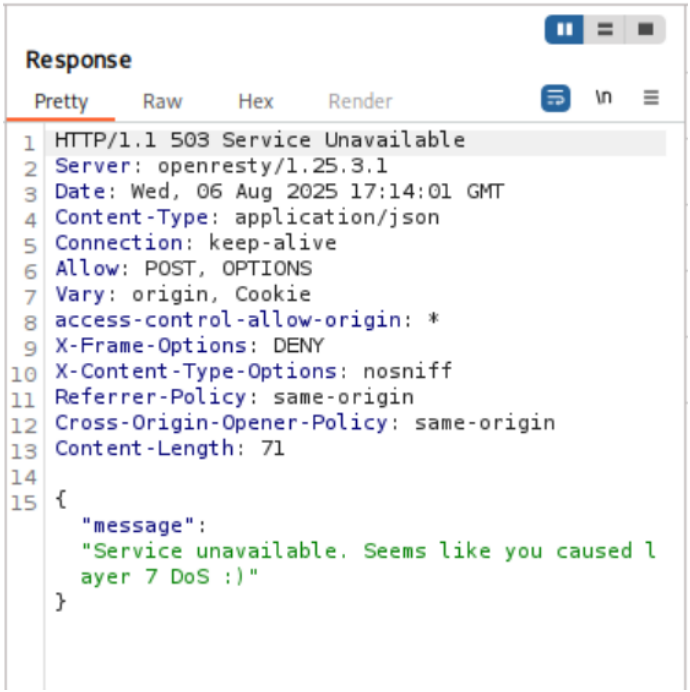
Fonte: Elaborado pelo autor

Figura 22 – Parâmetros de número de repetições e repetição de requisição alterados para sobrecarregar o sistema.



Fonte: Elaborado pelo autor

Figura 23 – DoS de camada 7 utilizando o recurso “contatc_mechanic”.



Fonte: Elaborado pelo autor

A partir da execução do experimento 6, tornou-se possível observar de forma clara como a ausência de limitação adequada de recursos impacta diretamente na estabilidade da aplicação. A manipulação de parâmetros do Endpoint “contact_mechanic” evidenciou que o servidor não possuía um controle efetivo de requisições por cliente, permitindo que chamadas consecutivas fossem processadas sem qualquer restrição.

Ao repetir as requisições de maneira contínua e automatizada, verificou-se uma sobrecarga imediata na camada de aplicação, caracterizando um DoS de camada 7. A indisponibilidade do serviço, demonstrou que o sistema não implementava mecanismos essenciais como *timeout*, limitação por IP ou quotas por usuário.

5 CONSIDERAÇÕES FINAIS

A análise experimental das vulnerabilidades OWASP API Security Top 10 2019 no ambiente crAPI permitiu constatar a materialização de falhas teóricas em cenários práticos de

exploração. Os resultados obtidos demonstram claramente como vulnerabilidades como Broken Object Level Authorization (API:2019), Excessive Data Exposure (API3:2019),

Lack of Resources & Rate Limiting (API:2019), podem ser exploradas para comprometer a segurança de aplicações baseadas em APIs.

Através da execução controlada dos testes, foi possível que a falha BOLA permitiu o acesso não autorizado a dados sensíveis de outros usuários mediante a simples manipulaçao de identificadores únicos (UUIDs). Medidas protetivas associadas, de acordo com o OWASP Foundation são: Implementar um sistema adequado de controle de acesso baseado em regras e níveis de proteção e permissão, garantir que toda funcionalidade que depende de dados fornecidos pelo usuário faça a checagem de autorização necessária para validar se o usuário pode visualizar ou manipular aqueles registros e priorizar o uso de identificadores (como GUIDs) aleatórios no lugar de IDs sequenciais para os registros.

A vulnerabilidade de autenticação teve o endpoint “forget-password”, analisado, cuja requisição interceptada no Burp Suite revelou que o aplicativo inclui o endereço de e-mail diretamente no processo de verificação do OTP. A análise demonstrou que, embora exista um mecanismo de limitação de taxa aplicado às tentativas de validação do OTP, esse controle pode ser parcialmente contornado. A partir da obtenção do e-mail de outro usuário na seção Community, tornou-se possível gerar OTPs para contas alheias e executar um atque de força-bruta utilizando manipulação de versionamento e envio automatizado de payloads numéricos via Burp Intruder. Com isso, confirmou-se a possiblidade de redefinir a senha de outro usuário caso o OTP correto fosse identificado, evidenciado por respostas com código HTTP 200. Medidas protetivas associadas, de acordo com o OWASP Foundation são: Garantir compreensão de todos caminhos possíveis de autenticação existestes na API, Aplicar proteções contra ataques de força bruta, e ataques por dicionário, nos enpoints da autenticação, tratar endpoints de recuperação de senha com o mesmo nível de atenção que os de login, incluindos medidas contra força, restrições de taxa e políticas de bloqueio.

A exposição excessiva de dados demonstrou como endpoints podem vazar informações sensíveis sem a devida filtragem no lado servidor. Medidas protetivas associadas, de acordo com o OWASP Foundation são: Analisar cuidadosamente as respostas da API para garantir que apenas os dados estritamente necessários estão sendo retornados, refletir quem realmente precisa do acesso a informação, antes de disponibilizar um novo endpoint.

A vulnerabilidade Lack of Resources & Rate Limiting no qual foi possível demonstrar um ataque de DoS camada 7 explorando a ausência de mecanismos adequados de limitação de requisições. Medidas protetivas associadas, de acordo com o OWASP Foundation são: Implementar mecanismos de rate limiting que controlem quantas vezes um cliente pode acessar a API dentro de um intervalo de tempo específico, incluir validações no servidor para checar parâmetros recebidos, especialmente no que diz respeito ao volume de registros solicitados, estabelecer limites para tamanho máximo de dados aceitos tanto em parâmetros individuais como em payloads completos.

Os resultados reforçam a importância de implementar controles de segurança robustos, incluindo: validação rigorosa de autorização em nível de objeto, implementação de mecanismos eficazes de rate limiting, utilização de tokens de autenticação com validação adequada, e o princípio do mínimo privilégio na exposição de dados através de endpoints.

REFERÊNCIAS

OLIVEIRA, Gabriel Bueno dos Santos Doria; RODRIGUES, Lucca Sabbatini Reid. Estudos e experimentos relativos ao uso de APIS Web para desenvolvimento Web. Julho, 2021

OWASP Foundation, OWASP API Security Top 10 - 2019. Disponível em: <https://owasp.org/API-Security/editions/2019/pt-BR/dist/owasp-api-security-top-10-pt-br.pdf>

OWASP FOUNDATION. *crAPI - Completely Ridiculous API*. 2024. Disponível em: <https://github.com/OWASP/crAPI.> Acesso em: out. 2025.

OWASP FOUNDATION. OWASP Web Security Testing Guide (WSTG). 2024. Disponível em: <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/12-API_Testing/00-API_Testing_Overview>. Acesso em: out. 2025.

OWASP Top 10 API 2019. [Recurso eletrônico]. Youtube, 2024. Disponível em: https://www.youtube.com/watch?v=YX-uivYFV2Q&t=8s