CESAR SCHOOL- CIÊNCIA DA
COMPUTAÇÃO - 6º PERÍODO

Maria Luísa Vieira Arruda

Profº: Henrique Arcoverde

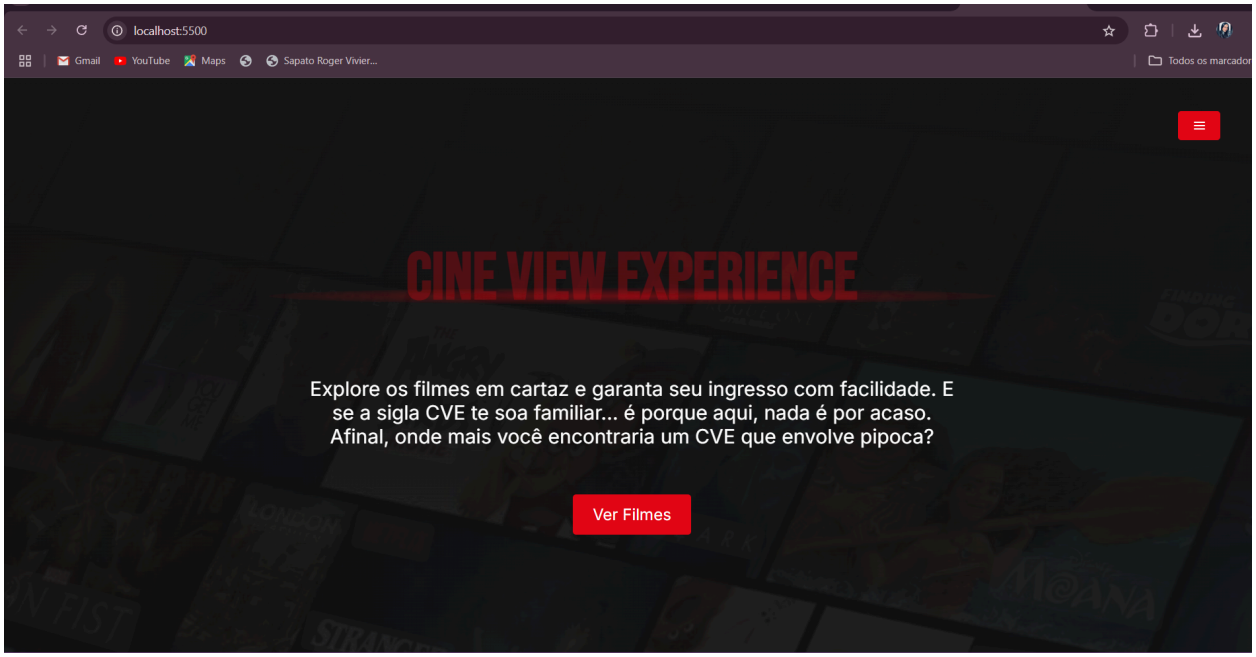
Análise de segurança para o Cine View Experience

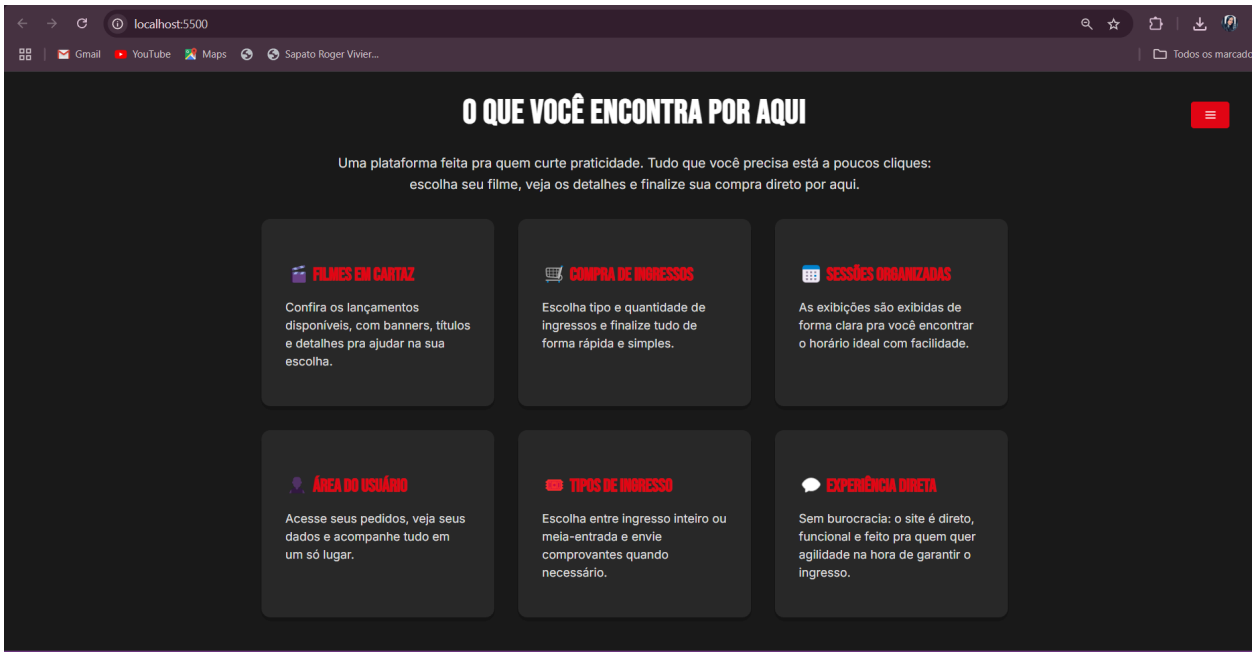
Recife,
2025

De início, após baixar e extrair o documento da aplicação do google Classroom fui executá-lo usando o DOCKER, seguindo estes passos:

1. Acessei via terminal o caminho da aplicação no meu computador
2. Ainda no terminal, utilizei o “ls” para listar e verificar todos os arquivos presentes
3. Subi o container da aplicação utilizando docker compose, comando utilizado: docker compose up --build

Depois de tudo subir corretamente, abri no meu navegador o localhost. O qual me devolveu está renderização da aplicação:





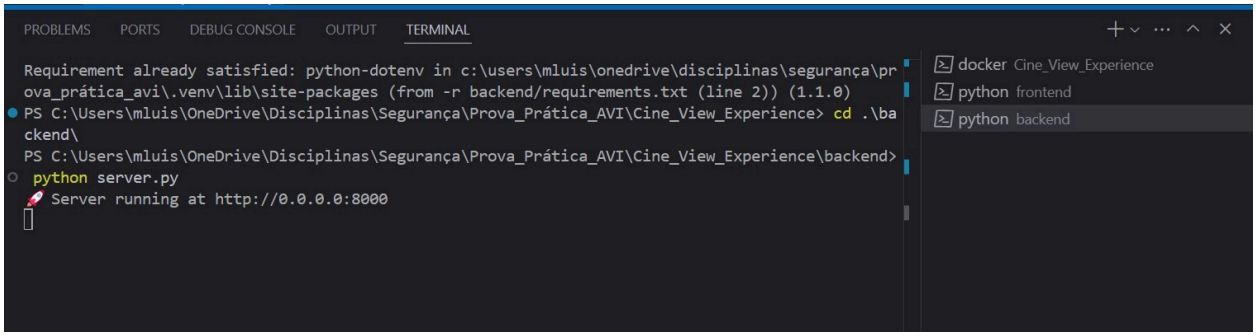
Resumo:

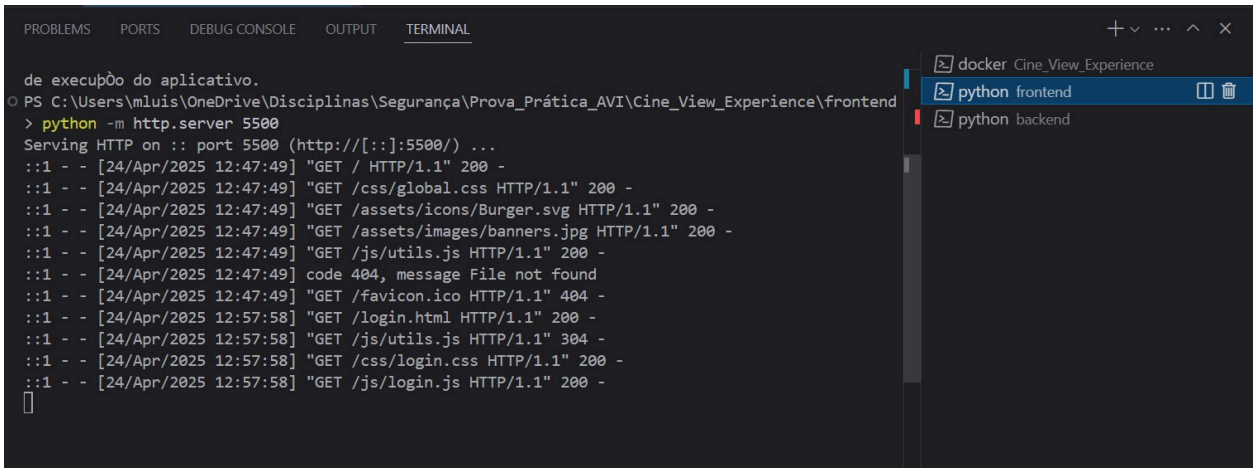
Durante o período de análise de segurança da aplicação Cine View Experience (CVE), foram identificadas 8 vulnerabilidades principais. O projeto teve início no dia 17/03/2025 e término no dia 25/03/2025. As falhas encontradas comprometem a integridade e a confidencialidade dos dados dos usuários, além de possibilitar acesso não autorizado a funcionalidades restritas. Agora, vamos começar a mexer e descobrir as vulnerabilidades de segurança desta aplicação. A princípio, irei utilizar o navegador do Burp para melhores interceptações.

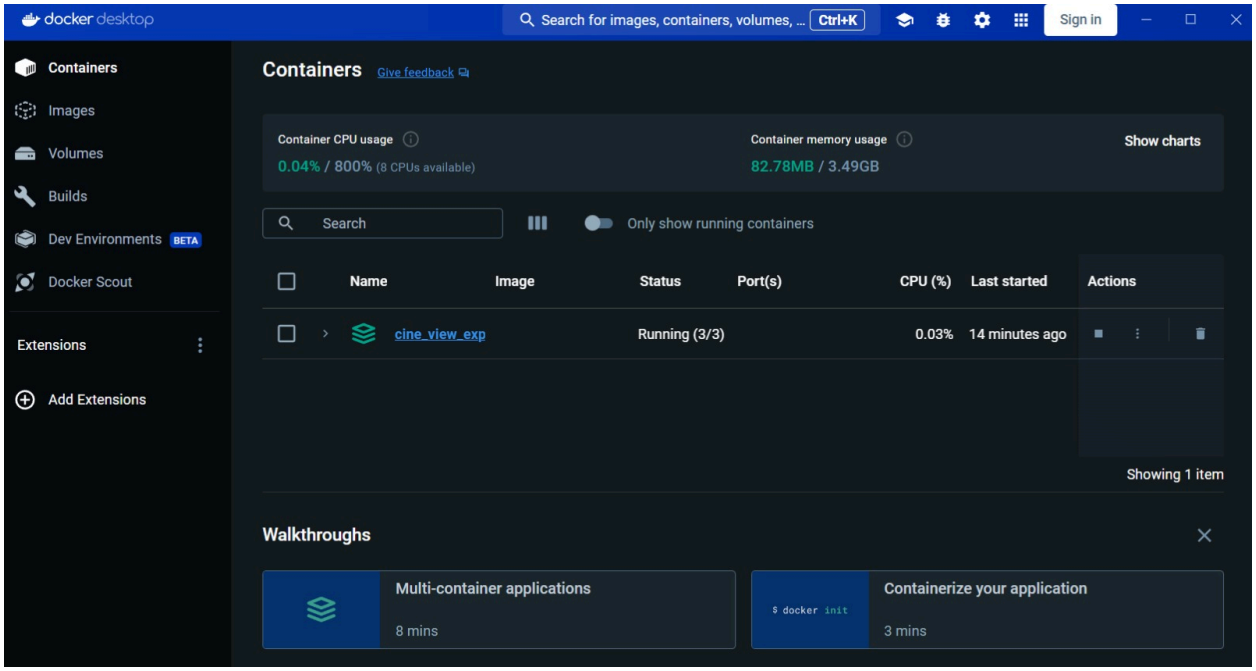
Observações Importantes:

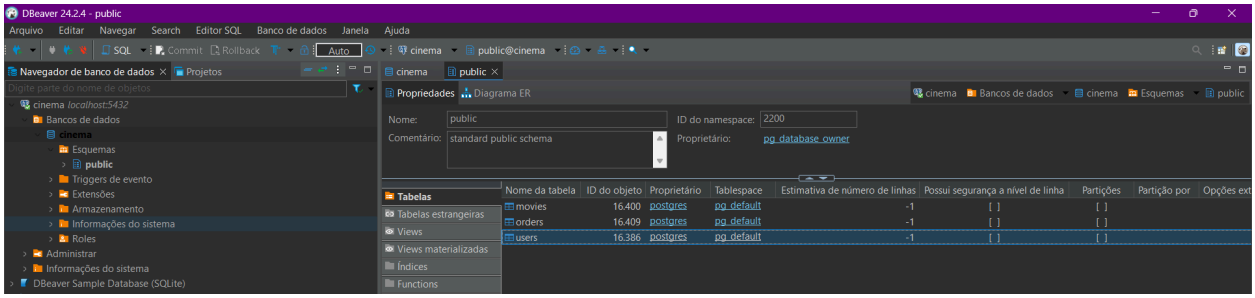
Estou tentando realizar esta prática de segurança, ao tentar conectar o frontend, backend e banco de dados encontrei um erro relacionado às portas. Para tentar resolver, troquei a porta de 5500, sendo utilizada pelo docker, para 5003, já que esta estava disponível. Mesmo assim, a aplicação não respondeu aos comandos dos botões, por exemplo, na aba de login, ao inserir um usuário e senha aleatórios (não cadastrados), o comportamento esperado seria retornar "invalid_answers" para a senha incorreta ou "user_not_found" para o usuário incorreto, conforme definido no código-fonte. No entanto, ao testar, não obtivemos nenhuma resposta.

Como a troca de portas não resolveu o problema, refiz todo o processo do zero, mais de uma vez e seguindo o readme passo a passo (desta vez, sem mudar nenhuma porta), após isso, obtive imagens/comandos de "sucesso". Seguem as imagens anexadas:


PROBLEMS
PORTS
DEBUG CONSOLE
OUTPUT
TERMINAL
Requirement already satisfied: python-dotenv in c:\users\mluis\onedrive\disciplinas\segurança\prova_prática_avi\.venv\lib\site-packages (from -r backend/requirements.txt (line 2)) (1.1.0)
PS C:\Users\mluis\OneDrive\Disciplinas\Segurança\Prova_Prática_AVI\Cine_View_Experience> cd .\backend\
PS C:\Users\mluis\OneDrive\Disciplinas\Segurança\Prova_Prática_AVI\Cine_View_Experience\backend>
python server.py
Server running at http://0.0.0.0:8000
docker Cine_View_Experience
python frontend
python backend


PROBLEMS
PORTS
DEBUG CONSOLE
OUTPUT
TERMINAL
de execuÞo do aplicativo.
PS C:\Users\mluis\OneDrive\Disciplinas\Segurança\Prova_Prática_AVI\Cine_View_Experience\frontend> python -m http.server 5500
Serving HTTP on :: port 5500 (http://[::]:5500/) ...
::1 - - [24/Apr/2025 12:47:49] "GET / HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:47:49] "GET /css/global.css HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:47:49] "GET /assets/icons/Burger.svg HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:47:49] "GET /assets/images/banners.jpg HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:47:49] "GET /js/utils.js HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:47:49] code 404, message File not found
::1 - - [24/Apr/2025 12:47:49] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [24/Apr/2025 12:57:58] "GET /login.html HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:57:58] "GET /js/utils.js HTTP/1.1" 304 -
::1 - - [24/Apr/2025 12:57:58] "GET /css/login.css HTTP/1.1" 200 -
::1 - - [24/Apr/2025 12:57:58] "GET /js/login.js HTTP/1.1" 200 -
docker Cine_View_Experience
python frontend
python backend


docker desktop
Search for images, containers, volumes, ... Ctrl+K
Sign in
Containers
Images
Volumes
Builds
Dev Environments BETA
Docker Scout
Extensions
Add Extensions
Containers
Give feedback
Container CPU usage
0.04% / 800% (8 CPUs available)
Container memory usage
82.78MB / 3.49GB
Show charts
Search
Only show running containers
Name
Image
Status
Port(s)
CPU (%)
Last started
Actions
cine_view_exp
Running (3/3)
0.03%
14 minutes ago
Showing 1 item
Walkthroughs
Multi-container applications
8 mins
$ docker init
Containerize your application
3 mins



Contudo, o problema persiste, com a aplicação ainda sem responder corretamente aos botões e funcionalidades. Por esta razão, para não ficar sem realizar a tarefa, analisei o código fonte e suas falhas de segurança. Seguirei o modelo exposto pelo professor e as partes referentes à “evidência na aplicação” e “passos para reproduzir” explicarei como eu faria se estivesse tudo funcionando perfeitamente.

Ademais, falei com alguns amigos e com a monitora, Luiza Omena, a qual guiou-me para verificar alguns arquivos, incluindo o routs.py e , caso necessário, ajustar as portas definidas neles. Sendo assim, testei com outras portas, ajustei o necessário, derrubei o container e o subi novamente, entretanto obtive o mesmo resultado, sem sucesso ao pressionar os botões e acionar funcionalidades.

Falhas descobertas

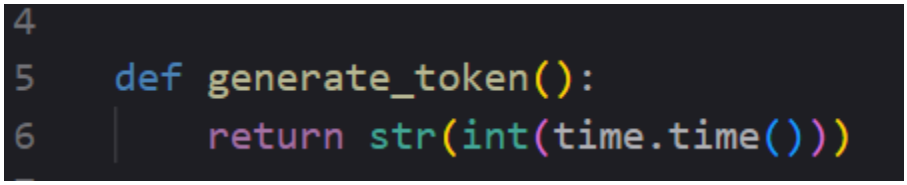
1. Sessão Fraca (Token Manipulável)

- **Ponto Afetado:** cookie de sessão, presente em todas as funcionalidades que dependem da autenticação.

- **Descrição:** a aplicação utiliza um mecanismo fraco para geração de tokens de sessão, baseado apenas no timestamp atual em segundos. Isso torna os tokens altamente previsíveis. Um atacante pode utilizar força bruta para gerar um token válido com base no tempo estimado da criação de sessões legítimas.

- **Evidência Aplicação:** interceptando o cookie com o Burp, deve ser possível gerar manualmente um token com um timestamp futuro e ainda assim obter acesso como se fosse uma sessão válida.

- **Evidência Código**: Trecho do código presente em session.py na função generate_token()



- **Passos para Reproduzir Utilizando o Burp:**
    1. Crie uma conta na aplicação.
    2. Capture o token gerado no cookie usando o Burp Suite.
    3. Acesse o Burp Intruder e configure um ataque de força bruta variando o token com valores de timestamp em torno do original.
    4. Envie as requisições para endpoints autenticados.
    5. Observe se algum dos tokens permite acesso não autorizado.

- **Impacto:** permite que um invasor acesse qualquer conta ativa no sistema, inclusive contas de administradores, comprometendo totalmente a confidencialidade e integridade dos dados.

- **Correção:** Substituir generate_token() por um gerador seguro; Definir tokens com expiração; Usar HttpOnly e SameSite=Strict para proteger o cookie

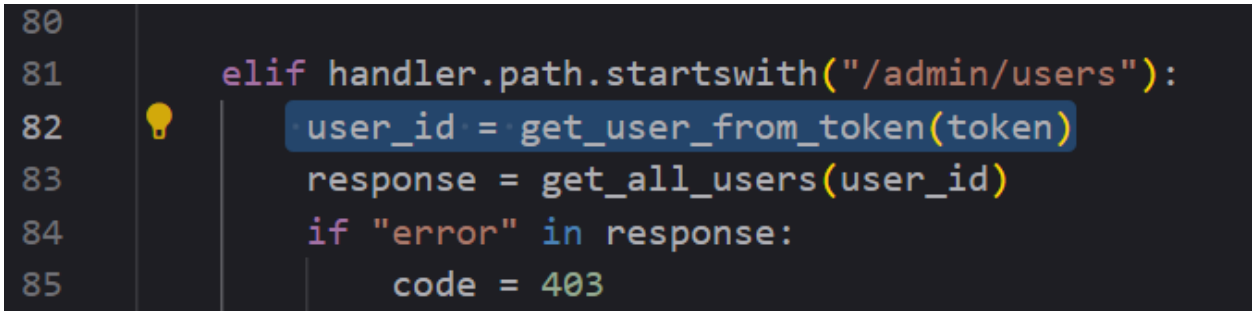
2. Bypass de Autenticação e Privilégios

- **Ponto Afetado:** Todos os endpoints que verificam autenticação e autorização de forma isolada: /admin/*, /order, /profile, etc.

- **Descrição:** a aplicação utiliza uma verificação de privilégio baseada exclusivamente no valor do token recebido do cliente. O backend apenas verifica se o token existe na memória e, em seguida, recupera o usuário associado para verificar se ele é administrador ou não. Como os tokens são manipuláveis (visto na vulnerabilidade anterior), um atacante pode gerar ou reutilizar tokens válidos de outros usuários, obtendo acesso a áreas restritas e privilegiadas.

- **Evidência Aplicação:** interceptando uma requisição feita por um usuário comum ao acessar uma funcionalidade normal, deve ser possível alterar manualmente o token para um token associado a um usuário com privilégios administrativos. Em seguida, reenviando a requisição, o sistema deve responder com dados administrativos, como a lista de usuários ou estatísticas do sistema.

- **Evidência Código**: trecho do código presente em routes.py, o qual permite a extração do token do usuário antes de verificar se o mesmo é válido ou se o usuário é autenticado



- **Passos para Reproduzir Utilizando o Burp:**

    1. Faça login como um usuário comum na aplicação.
    2. Intercepte a requisição para /profile ou outro endpoint autenticado.
    3. No Burp, localize o valor do cookie token.
    4. Substitua esse token por um token associado a um usuário com privilégios administrativos (gerado ou capturado previamente).
    5. Reenvie a requisição para o endpoint /admin/users ou /admin/stats.
    6. Verifique se os dados administrativos são retornados.

- **Impacto:** possibilita a um invasor o acesso à áreas administrativas, podendo realizar qualquer operação privilegiada, desta forma comprometendo completamente a integridade e confidencialidade do sistema.

- **Correção:** antes de confiar em um token recebido é preciso validar primeiro sua autenticidade com is_authenticated(token) e buscar o usuário com get_user_from_token(token), assim reorganize os blocos condicionais para que o acesso ao usuário só ocorra após a autenticação.; Usar a verificação de

assinatura digital; Utilizar tokens com expiração

3. IDOR (Insecure Direct Object Reference)

- **Ponto Afetado:** endpoint /profile

- **Descrição:** a aplicação permite que usuários atualizem seu próprio perfil enviando uma requisição PATCH para o endpoint /profile. No entanto, o backend aceita o campo user_id enviado pelo cliente, sem verificar se ele pertence ao usuário autenticado. Isso permite que um atacante modifique dados de outros usuários ao manipular o user_id na requisição.

- **Evidência Aplicação:** é possível interceptar a requisição patch com o Burp Suite e, assim , poder alterar o campo user_id no corpo da requisição para um ID de outro usuário. Desta maneira, após reenviar a requisição, o sistema deve responder com sucesso, indicando que os dados de outro usuário haviam sido modificados.

- **Evidência Código:** trecho do código presente em routes.py, o qual evidencia que a função update_user() utiliza o user_id vindo do post_data, ou seja, do corpo da requisiçã,enviada pelo cliente sem comparar se oid pertence ao usuário autenticado

```
141
142     elif handler.path.startswith("/edit-username"):
143         user_id = post_data["id"]
144         new_username = post_data["value"]
145         response = update_username_raw(user_id, new_username)
146         code = 200 if response.get("success") else 401
147         handler.send_response(code)
148
149     elif handler.path.startswith("/edit-email"):
150         response = update_profile_field(post_data["id"], "email", post_data["value"])
151         handler.send_response(200 if response.get("success") else 400)
152
153     elif handler.path.startswith("/edit-phone"):
154         response = update_profile_field(post_data["id"], "phone", post_data["value"])
155         handler.send_response(200 if response.get("success") else 400)
156
```

- **Passos para Reproduzir Utilizando o Burp:**

1. Faça login com um usuário comum.

2. Vá até a funcionalidade de edição de perfil e salve alguma alteração (por exemplo, mudar o nome).

3. Intercepte a requisição PATCH para /profile usando o Burp Proxy.

4. No corpo da requisição (JSON), localize o campo user_id e altere-o para o ID de outro usuário (ex: "user_id": 3).

5. Reenvie a requisição com o Burp Repeater.

6. Verifique se a resposta indica sucesso e se os dados foram realmente alterados.

- **Impacto:** permite alteração não autorizada de dados sensíveis de qualquer usuário, comprometendo a integridade do sistema e privacidade dos dados.

- **Correção:** ignorar o campo user_id enviado pelo cliente, ou seja, validar no lado do cliente; Utilizar apenas o identificador derivado de get_user_from_token(token) ; Aplicar checagens de autorização rigorosas antes de permitir atualizações de perfil.

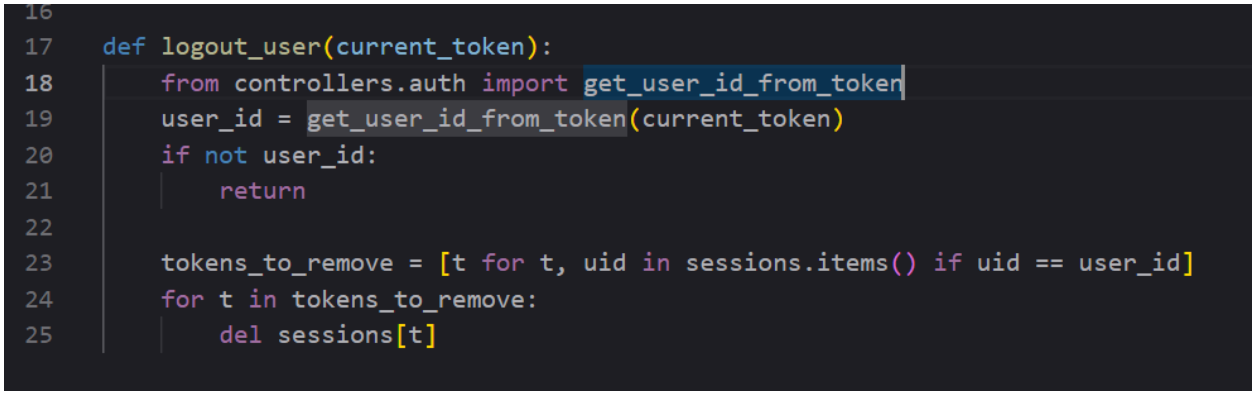
4. Ausência de Controle de Sessão no Logout

- **Ponto Afetado:** endpoint /logout e controle de sessão

- **Descrição:** após o logout, o backend apenas remove o token da memória (SESSIONS), mas não impede que o token continue sendo aceito se for reenviado manualmente enquanto ele ainda estiver ativo na memória. Como não há mecanismo de expiração, um atacante com acesso ao token pode continuar acessando a conta mesmo após o usuário legítimo ter feito logout.

- **Evidência Aplicação:** capturando o token de sessão com o Burp e armazenando-o, deve ser possível realizar logout normalmente pelo frontend. Em seguida, deve ser feita uma requisição autenticada (usando Burp Repeater) com o token antigo, e a aplicação ainda tende a permitir acesso como se a sessão fosse válida.

- **Evidência Código:** trecho do código presente em session.py, onde o token é removido da memória, mas não garante que ele não será aceito caso haja alguma falha de sincronização ou múltiplas instâncias. O método logout_user() remove todos os tokens associados ao usuário sem verificar se o token que solicitou o logout é válido ou pertence de fato ao usuário autenticado, o que pode permitir abusos como logout forçado de terceiros ou persistência de sessões em alguns contextos.



```
16
17  def logout_user(current_token):
18      from controllers.auth import get_user_id_from_token
19      user_id = get_user_id_from_token(current_token)
20      if not user_id:
21          return
22
23      tokens_to_remove = [t for t, uid in sessions.items() if uid == user_id]
24      for t in tokens_to_remove:
25          del sessions[t]
```

- **Passos para Reproduzir Utilizando o Burp:**

    1. Faça login e capture o cookie ou header de autenticação com o token (Authorization: Bearer <token>).

    2. Realize o logout normalmente pela aplicação.

    3. Volte ao Burp e use o Burp Repeater para enviar uma requisição autenticada com o token capturado.

    4. Observe que o backend ainda aceita o token e responde como se o usuário ainda estivesse autenticado.

- **Impacto:** permite que um invasor continue acessando a conta após o logout, comprometendo a confidencialidade e permitindo ataques persistentes com essas sessões “encerradas”.

- **Correção**: implementar tokens com expiração; Garantir a invalidação imediata e global do token após logout, inclusive em ambientes com múltiplas instâncias.

5. Enumeração de Usuário

- **Ponto Afetado:** endpoint /login

- **Descrição**: a aplicação permite identificar se um nome de usuário existe no sistema através das mensagens de erro retornadas durante o login. Quando um usuário inválido é informado, a resposta do servidor retorna "user_not_found". Essa resposta diferenciada permite a um atacante realizar um ataque de enumeração, descobrindo quais usuários estão cadastrados na aplicação.

- **Evidência Aplicação:** utilizando o Burp, deve ser enviada uma requisição POST para o login com um nome de usuário inexistente e a resposta do servidor retornando uma mensagem específica: user_not_found. Enquanto, quando posto e enviado um nome de usuário válido, mas com senha incorreta, a resposta muda para: invalid_credentials. Logo, essas respostas distintas permitem inferir quais usuários estão registrados na aplicação.

- **Evidência Código:** trecho do código presente no arquivo auth.py

```
14      if not user:
15          cur.close()
16          conn.close()
17          return {"success": False, "error": "user_not_found"}
18
19      if (user[1] == data["favorite_color"] and
20          user[2] == data["birth_year"] and
21          user[3] == data["first_school"]):
22          cur.close()
23          conn.close()
24          return {"success": True}
25      else:
26          cur.close()
27          conn.close()
28          return {"success": False, "error": "invalid_answers"}
29
```

- **Passos para Reproduzir Utilizando o Burp:**

1. Abra o Burp Suite e intercepte a requisição de login.

2. Envie um POST para /login com um nome de usuário inexistente (ex: {"username": "naoexiste", "password": "qualquer"}).

3. Observe a resposta "user_not_found".

4. Tente novamente com um nome de usuário existente, mas senha incorreta, e note a diferença na mensagem de erro ("invalid_credentials").

5. Esse comportamento permite confirmar se um nome de usuário é válido.

- **Impacto**: permite a um atacante enumerar usuários válidos do sistema, o que pode ser um passo inicial para ataques mais sofisticados como brute force e engenharia social.

- **Correção:** padronizar as mensagens de erro para que não revelem se o nome de usuário existe. Por exemplo, sempre retornar: invalid_credentials

6. Ausência de Proteção Contra Força Bruta

- **Ponto Afetado:** endpoint /login e processo de verificação de identidade (/verify_user)

- **Descrição:** a aplicação permite tentativas ilimitadas de login e verificação de identidade (perguntas secretas), sem qualquer mecanismo de bloqueio, atraso progressivo ou captcha. Isso facilita ataques automatizados de força bruta para descobrir senhas ou respostas de recuperação de conta.

- **Evidência Aplicação:** utilizando o Burp intruder, deve ser possível enviar diversas requisições POST para o endpoint /login com o mesmo nome de usuário e variações na senha. A aplicação respondeu rapidamente a todas as tentativas sem aplicar qualquer tipo de limitação ou bloqueio, retornando consistentemente a mensagem de erro para senha incorreta: invalid_password. O mesmo comportamento deve ser observado no endpoint /verify_user, onde as perguntas secretas (cor favorita, ano de nascimento, escola) devem ser testadas repetidamente com respostas distintas, sem qualquer medida antifraude, permitindo identificar combinações corretas por tentativa e erro.

- **Evidência Código:** trecho presente em auth.py, login e verify_user:

```python
def login(data, token):
    conn = get_connection()
    cur = conn.cursor()

    cur.execute(
        "SELECT id FROM users WHERE username = %s AND password = %s",
        (data["username"], data["password"])
    )
    user = cur.fetchone()

    cur.close()
    conn.close()

    if user:
        user_id = user[0]
        set_session(token, user_id)
        return {"success": True}
    return {"success": False}
```

```python
def verify_user(data):
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("SELECT id, favorite_color, birth_year, first_school FROM users WHERE username = %s",
                (data["username"],))
    user = cur.fetchone()

    if not user:
        cur.close()
        conn.close()
        return {"success": False, "error": "user_not_found"}

    if (user[1] == data["favorite_color"] and
        user[2] == data["birth_year"] and
        user[3] == data["first_school"]):
        cur.close()
        conn.close()
        return {"success": True}
    else:
        cur.close()
        conn.close()
        return {"success": False, "error": "invalid_answers"}

```

- **Passos para Reproduzir Utilizando o Burp:**

    1. Capture a requisição de login via Burp Proxy.
    2. Envie a requisição ao Burp Intruder, configurando um payload de senhas (ex: rockyou.txt).
    3. Inicie o ataque.

4. Observe que a aplicação responde com o mesmo tempo e mensagem para todas as tentativas, permitindo automação contínua.

5. Repita o processo para /verify_user testando combinações de respostas.

- **Impacto:** permite que um atacante use ferramentas automatizadas para descobrir credenciais válidas ou respostas de recuperação de conta, comprometendo a segurança de múltiplos usuários e permitindo acesso não autorizado a contas legítimas.

- **Correção:** implementar delay progressivo a cada tentativa falha de login ou verificação; Adotar bloqueio temporário após X tentativas inválidas.; Utilizar MFA, como o captcha; Registrar tentativas e notificá-las ao usuário ou administrador.

7. Verificação de Identidade Fraca (Bypass de 2FA)

- **Ponto afetado:** endpoint /verify_user no arquivo auth.py

- **Descrição:** o processo de verificação de identidade (usado como uma forma de "segundo fator" para ações sensíveis como reset de senha) se baseia exclusivamente em perguntas estáticas definidas pelo próprio usuário, como cor favorita, ano de nascimento e nome da primeira escola. Essas informações são frequentemente previsíveis, podem ser coletadas em redes sociais ou obtidas por engenharia social, permitindo que um atacante burle essa etapa e tome controle da conta.

- **Evidência Aplicação:** usando o Burp repeater, deve ser possível simular múltiplas tentativas ao endpoint /verify_user com um nome de usuário válido e diferentes combinações de respostas para as perguntas secretas. Com tentativas sucessivas, é possível descobrir uma combinação correta e receber uma resposta positiva da aplicação. Isso demonstrou que a aplicação não possui proteção contra tentativas repetidas, nem exige múltiplos fatores reais de autenticação.

- **Evidência Código: t**recho presente em auth.py, na função verify_user

```python
def verify_user(data):
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("SELECT id, favorite_color, birth_year, first_school FROM users WHERE username = %s",
                (data["username"],))
    user = cur.fetchone()

    if not user:
        cur.close()
        conn.close()
        return {"success": False, "error": "user_not_found"}

    if (user[1] == data["favorite_color"] and
        user[2] == data["birth_year"] and
        user[3] == data["first_school"]):
        cur.close()
        conn.close()
        return {"success": True}
    else:
        cur.close()
        conn.close()
        return {"success": False, "error": "invalid_answers"}

```

- **Passos para Reproduzir Utilizando o Burp:**

1. Capture a requisição de login via Burp Proxy.
2. Envie a requisição ao Burp Intruder, configurando um payload de senhas (ex: rockyou.txt).
3. Inicie o ataque.
4. Observe que a aplicação responde com o mesmo tempo e mensagem para todas as tentativas, permitindo automação contínua.
5. Repita o processo para /verify_user testando combinações de respostas.
6. Obtenha um nome de usuário válido (por enumeração, engenharia social, etc.).
7. pita o processo para /verify_user testando combinações de respostas.
8. Intercepte a requisição ao endpoint /verify_user usando Burp Proxy.

9. Envie múltiplas variações de respostas via Burp Repeater até encontrar uma combinação válida.

10. Após acerto, observe que a aplicação retorna sucesso e permite continuar o fluxo de recuperação de conta ou reset de senha.

- **Impacto:** facilita que um atacante contorne etapas críticas de verificação de identidade e obtenha acesso total à conta de outro usuário. Como se trata de um método de "2FA" substituto ou etapa de recuperação, o impacto é elevado, pois expõe usuários a sequestro de conta, especialmente se combinada com enumeração de usuário e força bruta.

- **Correção:** substituir perguntas secretas por mecanismos de autenticação robustos; Adicionar limite de tentativas e bloqueio temporário por IP ou usuário.

8. Reset de Senha sem verificação forte

- **Ponto afetado:** endpoint /reset_password e função reset_password()

- **Descrição:** funcionalidade de reset de senha permite que qualquer pessoa com o nome de usuário redefina a senha diretamente, sem a exigência de um token seguro, prova de identidade válida ou verificação adicional (e-mail, SMS, etc.). Esse comportamento torna o processo vulnerável a ataques de sequestro de conta, especialmente se combinado com enumeração de usuário e verificação de identidade fraca.

- **Evidência Aplicação: c**om Burp Suite, deve ser possível capturar a requisição enviada ao endpoint /reset_password, mesmo sem autenticação ou token de recuperação, o backend tende a responder com sucesso. Dessa maneira, indicando que qualquer atacante que conheça o nome de usuário pode alterar a senha arbitrariamente.

- **Evidência Código: t**recho presente em auth.py, função reset_password, pois a função apenas recebe o nome de usuário e a nova senha, e executa diretamente a atualização no banco, sem a devida verificação do token e sem a devida validação e autenticação do usuário.

```python
def reset_password(data):
    if not is_strong_password(data["password"]):
        return {"success": False}

    conn = get_connection()
    cur = conn.cursor()

    try:
        cur.execute("UPDATE users SET password = %s WHERE username = %s", (data["password"], data["username"]))
        conn.commit()
        success = True
    except Exception:
        conn.rollback()
        success = False
    finally:
        cur.close()
        conn.close()

    return {"success": success}

```

- **Passos para Reproduzir Utilizando o Burp:**

1. Obtenha o nome de usuário de uma conta válida (pode ser por enumeração ou conhecimento prévio).

2. Intercepte a requisição com Burp e envie um POST /reset_password com o nome de usuário e nova senha.

3. Observe que a senha é alterada mesmo sem estar logado, sem token, sem verificação por e-mail ou outros fatores.

4. Use as novas credenciais para fazer login normalmente.

- **Impacto:** permite que qualquer pessoa redefina a senha de qualquer conta, contanto que conheça o nome de usuário.

- **Correção:** implementar fluxo seguro de recuperação de senha, o qual inclua: o envio de token único e temporário (com expiração) por e-mail ou SMS, a devida validação deste token antes de permitir o reset e garantir que o token esteja vinculado ao usuário correto; Bloquear os limites de tentativas de reset de senha para evitar automações maliciosas;