Relatório de Bugs de Segurança e Validação da API - 04/09/2025

Este documento detalha os bugs encontrados durante a sessão de testes realizada em 04 de setembro de 2025. Para cada bug, são fornecidos os passos necessários para reproduzi-lo, o resultado esperado e o resultado observado.

🐞 BUG 1: API Fornece Dicas de Autenticação em Mensagens de Erro

A API expõe informações parciais do e-mail do usuário na mensagem de erro de autenticação, o que pode ser explorado por invasores para adivinhar credenciais de acesso.

Passo a passo para reproduzir:

1. Realize uma requisição `POST` para o endpoint de autenticação da aplicação (ex: `/login`).
2. No corpo da requisição, forneça um e-mail de usuário válido, mas com uma senha incorreta.
3. Envie a requisição e analise a resposta de erro retornada pela API.

Resultado Esperado:

A API deve retornar uma mensagem de erro genérica, que não revele nenhuma informação sobre o usuário ou o motivo exato da falha. Por exemplo:

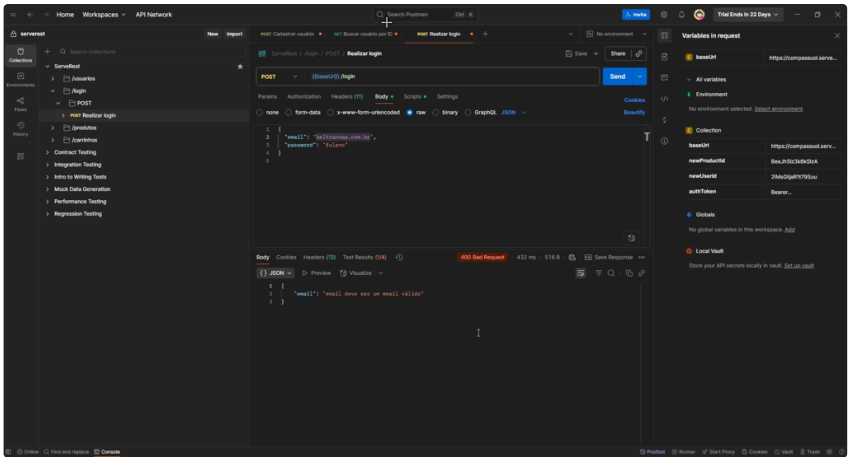
```
JSON
```

```
{
  "erro": "Credenciais inválidas"
}

```

Resultado Atual:

A API retorna uma mensagem de erro que inclui parte do e-mail do usuário, confirmando que o e-mail existe na base de dados e fornecendo uma dica indevida. (Conforme evidência: `Screenshot-2-nao-deve-fornecer-dicas-de-email-na-mensagem-de-autenticação.png`)



🐞 BUG 2: Credenciais de Usuário Aceitas como Parâmetros na URL (Path Params)

A rota `GET /produtos` permite que credenciais de autenticação sejam passadas diretamente na URL como parâmetros de caminho. Isso representa uma falha de segurança grave, pois as URLs são frequentemente armazenadas em logs de servidores e no histórico do navegador.

Passo a passo para reproduzir:

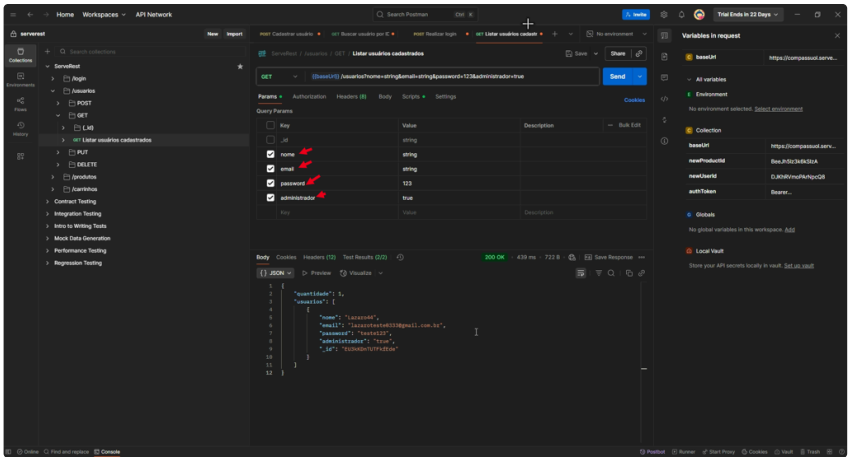
1. Construa uma requisição do tipo `GET` para a rota `/produtos`.
2. Anexe credenciais de usuário (login e senha) diretamente na URL como se fossem parte do caminho. Exemplo: `GET /produtos/nome_do_usuario/senha_do_usuario`
3. Execute a requisição em um cliente de API (como Postman, Insomnia, etc.).

Resultado Esperado:

A API não deve processar credenciais via parâmetros de caminho. O esperado seria uma resposta de erro, como `404 Not Found` (rota inexistente) ou `401 Unauthorized` (não autorizado).

Resultado Atual:

A API processa incorretamente as credenciais passadas na URL, autentica o usuário e retorna a lista de produtos com um status `200 OK`, validando a brecha de segurança. (Conforme evidência: `Screenshot-credenciais-sensíveis-sendo-passadas-como path-params.png`)



🐞 BUG 3: Rota PUT de Produtos Permite Inserir Preço como String

A rota para atualização de produtos (`PUT /produtos/{id}`) não valida corretamente o tipo de dado do campo `preço` , permitindo que valores do tipo *string* (texto) sejam inseridos em um campo que deveria aceitar apenas números.

Passo a passo para reproduzir:

1. Identifique o ID de um produto existente no sistema.
2. Construa uma requisição do tipo `PUT` para o endpoint de atualização do produto (ex: `/produtos/{id_do_produto}`).
3. No corpo (body) da requisição, informe um valor não numérico (string) para o campo `preco` . Por exemplo:

```
JSON
```

```
{
  "nome": "Produto de Teste",
  "preco": "10",
  "descricao": "Descricao de teste",
  "quantidade": 10
}

```

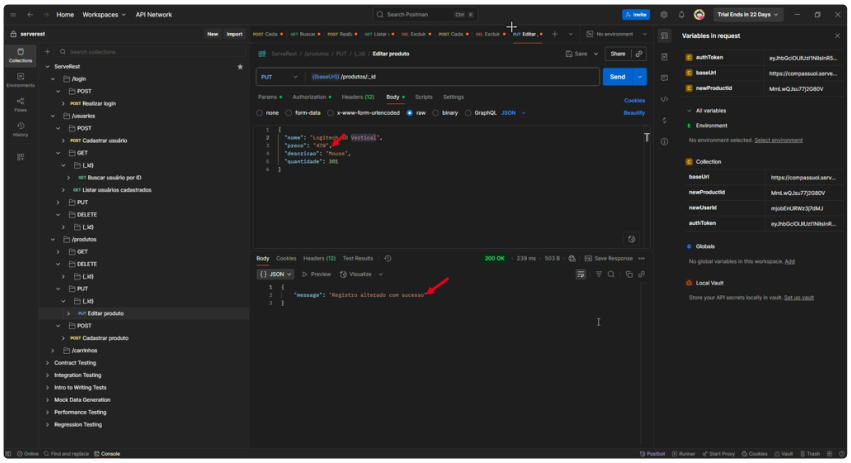
4. Envie a requisição para a API.

Resultado Esperado:

A API deveria validar o tipo de dado do campo `preco` . Ao receber uma string, ela deveria rejeitar a requisição com um status de erro `400 Bad Request` e uma mensagem indicando que o formato do preço é inválido.

Resultado Atual:

A API aceita a requisição com o valor de texto no campo `preco` e retorna um status `200 OK`, confirmando a atualização indevida do produto com dados corrompidos. (Conforme evidência: `Screenshot-rota-PUT-de-produtos-permite-preço-tipo-string.png`)



🐞 BUG 4: API Permite que um Usuário Comum Exclua um Usuário Admin

Foi identificada uma falha crítica de controle de acesso na API que permite que um usuário autenticado com permissões comuns (não-administrador) exclua o registro de um usuário com privilégios de administrador.

Passo a passo para reproduzir:

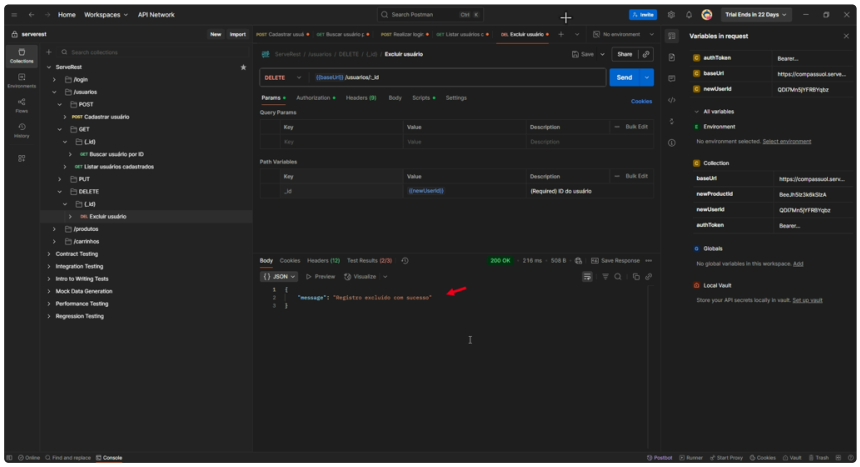
1. **Login como usuário comum:** Autentique-se na API utilizando as credenciais de um usuário que **não** possua permissões de administrador. Obtenha o token de autenticação gerado para esta sessão.
2. **Identificar um usuário admin:** Obtenha o ID (identificador único) de um usuário que possua perfil de administrador no sistema. Isso pode ser feito através de uma listagem de usuários (`GET /usuarios`), se disponível.
3. **Realizar a requisição de exclusão:** Utilizando o token de autenticação do usuário comum (do passo 1), construa e envie uma requisição do tipo `DELETE` para o endpoint de exclusão de usuários, utilizando o ID do usuário administrador (do passo 2).
   - **Exemplo da requisição:** `DELETE /usuarios/{id_do_usuario_admin}`
   - **Header de autorização:** `Authorization: Bearer {token_do_usuario_comum}`
4. **Verificar o resultado:** Analise o código de status da resposta e verifique se o usuário administrador foi removido da base de dados.

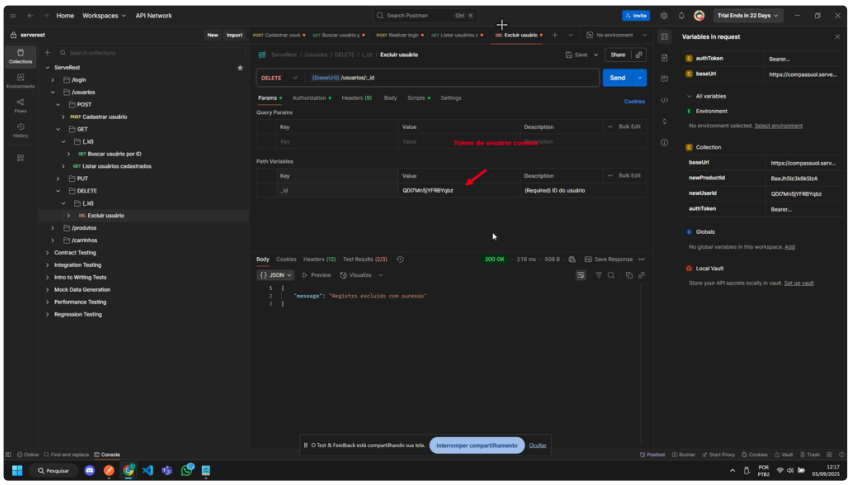
Resultado Esperado:

A API deveria intervir e bloquear a operação. A requisição deveria ser rejeitada com um código de status `403 Forbidden` (Proibido) ou `401 Unauthorized` (Não autorizado), acompanhado de uma mensagem de erro indicando que o usuário não tem permissão para realizar tal ação.

Resultado Atual:

A API processa a solicitação com sucesso e retorna um código de status `200 OK`, confirmando que o usuário administrador foi excluído, permitindo que um usuário de baixo privilégio realize uma ação destrutiva de alto privilégio. (Conforme evidências: `Screenshot-usuario-comum-pode-excluir-usuario-admin.png` e `Screenshot-usuario-comum-pode-excluir-usuario-admin-2.png`).





🐞 BUG 5: API Permite Cadastro de Usuário com Emojis no Nome

A rota `POST /usuarios` não possui uma validação ou sanitização adequada para o campo `nome`, permitindo que usuários sejam cadastrados utilizando caracteres especiais, incluindo emojis. Isso pode causar problemas de renderização em interfaces (front-end) e inconsistência nos dados.

Passo a passo para reproduzir:

1. **Construir a requisição de cadastro:** Prepare uma requisição do tipo `POST` para o endpoint de criação de novos usuários (ex: `/usuarios`).
2. **Inserir emojis no payload:** No corpo (body) da requisição, preencha os campos obrigatórios. No campo `nome`, insira um texto que contenha um ou mais caracteres de emoji.
   - **Exemplo de payload:**

```
JSON
```

```
{
  "nome": "adminbetrano🤣🤣🤣💕💕🤣🤣",
  "email": "tememoji@qa.com.br",
  "password": "adminfulano",
  "administrador": "true"
}
```

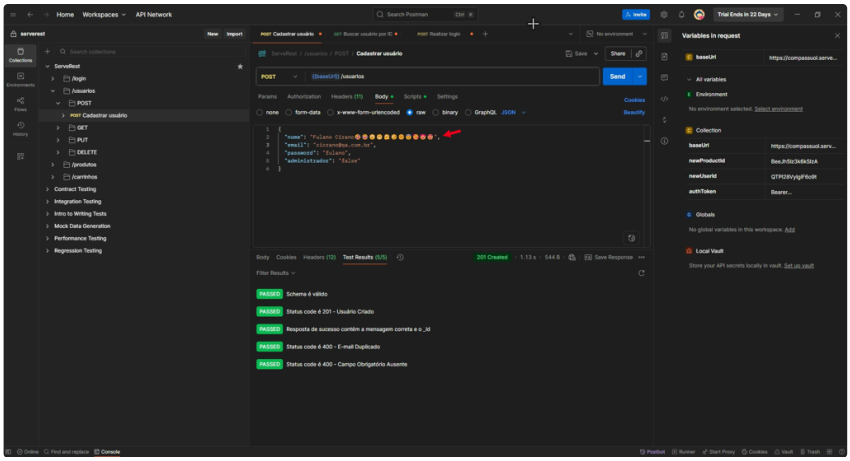
3. **Enviar a requisição:** Execute a requisição `POST` para a API.
4. **Verificar o resultado:** Analise a resposta da API e, se possível, consulte a base de dados ou a rota de listagem de usuários para confirmar que o novo registro foi criado com o emoji no nome.

Resultado Esperado:

A API deveria validar os campos de entrada, especialmente campos de nome de usuário, para permitir apenas um conjunto de caracteres válidos (ex: alfanuméricos e espaços). A requisição com emojis deveria ser rejeitada com um código de status `400 Bad Request` e uma mensagem clara, como `"O campo 'nome' contém caracteres inválidos."`.

Resultado Atual:

A API aceita o payload sem objeções e cria o novo usuário no sistema com o emoji em seu nome, retornando um código de status `201 Created`. Isso indica uma falha na validação dos dados de entrada, o que pode levar a comportamentos inesperados em outras partes do sistema. (Conforme evidência: `Screenshot-permite-que-insira-emojis-na-aplicacao.png`).


Home
Workspaces
API Network
Search Postman
Invite
Trial Ends in 22 Days
serverest
New
Import
POST Cadastrar usuário
GET Buscar usuário por ID
POST Realizar login
No environment
Collections
Environments
Flows
History
Search collections
ServeRest
/login
/usuarios
POST
POST Cadastrar usuário
GET
PUT
DELETE
/produtos
/carrinhos
Contract Testing
Integration Testing
Intro to Writing Tests
Mock Data Generation
Performance Testing
Regression Testing
ServeRest / Usuarios / POST Cadastrar usuário
Save
Share
POST
{{baseUrl}}/usuarios
Send
Params
Authorization
Headers (11)
Body
Scripts
Settings
Cookies
Beautify
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"nome": "Fulano Cicero
"email": "cicero@cs.com.br",
"password": "fulano",
"administrador": "false"
}
Body
Cookies
Headers (12)
Test Results (5/5)
201 Created
1.13 s
544 B
Save Response
Filter Results
PASSED
Schema é válido
PASSED
Status code é 201 - Usuário Criado
PASSED
Resposta de sucesso contém a mensagem correta e o _id
PASSED
Status code é 400 - E-mail Duplicado
PASSED
Status code é 400 - Campo Obrigatório Ausente
Variables in request
baseUrl
https://compassuol.serve...
All variables
Environment
No environment selected. Select environment
Collection
baseUrl
https://compassuol.serv...
newProductId
6ee.fr3z3ek6k3iz.A
newUserId
QTPI2dVyjiPEo0t
authToken
Bearer...
Globals
No global variables in this workspace. Add
Local Vault
Store your API secrets locally in vault. Set up vault
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash