Rest API com JWT

Lucas Gabriel de Oliveira Lima - 231003406
Vinicius da Silva Araujo - 221001981

26 de maio de 2025

Resumo

Este projeto apresenta a implementação de uma API REST segura em Python, desenvolvida para fins educacionais e demonstração prática de técnicas modernas de autenticação e criptografia. A API suporta autenticação de usuários utilizando JSON Web Tokens (JWT) com algoritmos HMAC e RSA, hash de senhas com bcrypt e armazenamento seguro de dados em SQLite. Todas as comunicações são protegidas com HTTPS utilizando um certificado SSL autoassinado. Também é fornecido um cliente de linha de comando para interação com a API, permitindo autenticação e acesso a recursos protegidos. O projeto enfatiza boas práticas de segurança, como gestão de segredos, validação de tokens e comunicação criptografada, servindo como base para o entendimento do desenvolvimento de APIs seguras.

1 Introdução

Com a crescente demanda por aplicações web seguras, compreender os princípios e a implementação de APIs seguras tornou-se essencial para desenvolvedores.

Este trabalho tem como objetivo demonstrar, de forma prática e acessível, como construir uma API REST segura e padrões de segurança. O projeto integra diversos mecanismos de segurança fundamentais: autenticação de usuários via JWT (suportando assinaturas simétricas HMAC e assimétricas RSA), armazenamento de senhas com hash bcrypt e comunicação criptografada via HTTPS.

O servidor é implementado utilizando apenas módulos nativos do servidor HTTP do Python, evitando frameworks externos para manter a arquitetura transparente e didática. Além disso, um cliente de linha de comando é fornecido para facilitar testes e ilustrar o fluxo de autenticação. Este relatório detalha a arquitetura, as escolhas de implementação e as considerações de segurança do projeto.

Em nossa análise de segurança, utilizaremos também como base o sniffing de pacotes trocandos por requisições e respostas trocados entre cliente e servidor. Para isso, fizemos isso nos pacotes trocados sob o protocolo HTTP apenas para fazer análise, para que pudessemos visualizar o conteúdo original dos pacotes. No entanto, com a aplicação funcionando normalmente, utilizamos o HTTPS.

Por motivos de simplificação, trabalharemos com um usuário já existente por padrão, com nome 'admin' e senha 'admin', de forma que será guardado no banco seu nome e o hash da sua senha, e este será usado para os métodos de autenticação.

O código-fonte do projeto está disponível no seguinte repossitório no GitHub: https://github.com/Vini-ara/Rest_api_tac. Nele, também está disponivel um README que contém os comandos de como executar o cliente e o servidor da aplicação, e como ocorre as interações para fazer requisições ao servidor.

2 Análise de segurança

2.1 JWT - Json Web Token

O padrão **JWT** (Json Web Token) é utilizado para gerar tokens de autenticação para usuários de uma aplicação. O JWT trabalha com dois conceitos importantes. Um deles é o algoritmo que será utilizado na geração desse token, que em nossa aplicação foram utilizados **HMAC** e **RSA PKCS**, que será guardado no *header* do token. O outro conceito é o *payload*, que é onde será guardado informações como tempo de expiração e de quem é aquele token.

Definidos o algoritmo e quais dados será armazenado no payload, nosso servidor cria um token para o usuário, com tempo de expiração de uma hora, e então ele é enviado de volta para o cliente, que o armazenará nos *cookies*. A partir disso, toda requisição a rotas protegidas passará por uma validação do token, a partir do algoritmo escolhido para se trabalhar com o token, para permitir ou não que o cliente tenha o acesso às informações que ele solicitou, obtendo sucesso em sua requisição, com status 200 (OK), ou não, obtendo um erro 401 de acesso não autorizado.

Os erros na validação de token podem ter diersos motivos, mas as principais são a de tempo expirado, ou de token inválido, que pode ser um token gerado a partir de uma chave diferente da usada pela aplicação, ou por outras modificações maliciosas.

2.2 HMAC - Hash-based Message Authentication Code

O **HMAC** é um médoto criptográfico que busca assegurar integridade e autenticidade utilizando uma função de hash e uma chave secreta.

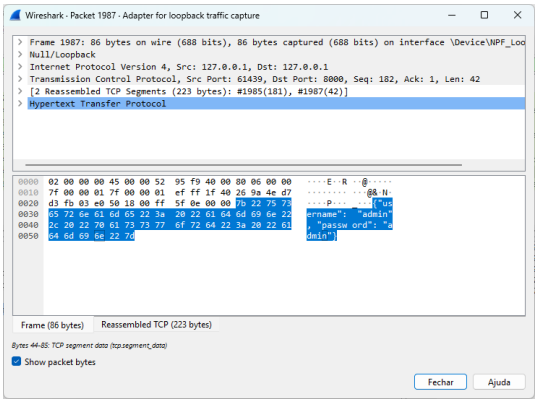
No contexto deste projeto, o HMAC é utilizado para assinar e validar tokens JWT quando o algoritmo simétrico **HS256 (HMAC com SHA-256)** é selecionado. O servidor gera um token JWT contendo informações do usuário e um campo de expiração, e cria esse token usando uma chave secreta de 32 bytes armazenada de forma segura. Ao receber o token, o servidor pode verificar sua autenticidade recalculando o HMAC e comparando com a assinatura recebida, garantindo que o token não foi alterado e que foi realmente emitido pelo servidor.

Por ser um algoritmo simétrico, requer a gestão robusta da chave secreta, que trabalha não só na geração, como também na validação do token. Isso faz que ele seja mais aconselhável a aplicações monolíticas e serviços internos confiáveis, de preferência que não haja muitos compartilhamentos da chave ssecreta.

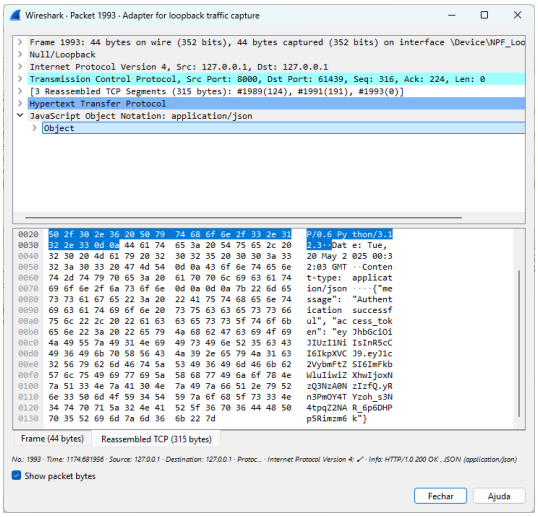
A seguir está um pouco da dinâmica da autenticação por HMAC em nossa aplicação:

Primeiramente, ocorre uma solicitação de autenticação do usuário, passando seu nome de usuário e sua senha.

Caso o usuário existir e sua estiver correta, o servidor responder a requisição com status 200 (OK) e devolve um token JWT gerado utiliando HMAC.



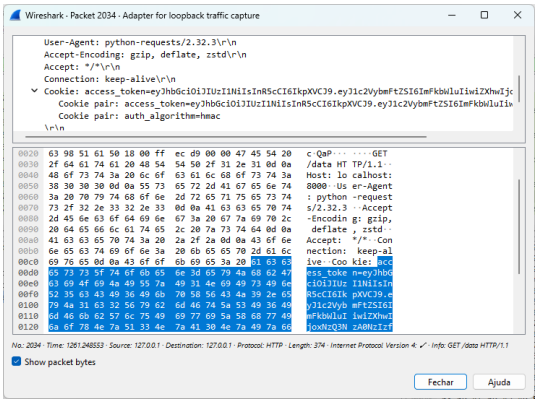
(a) Requisição de autenticação por HMAC



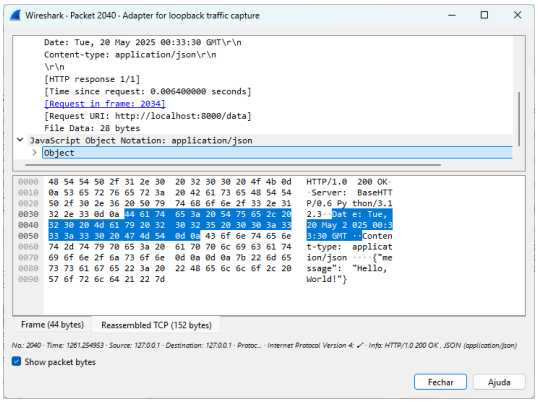
(b) Resposta de autenticação por HMAC

Figura 1: Autenticação por HMAC

Ao receber esse token, ele é inserido nos cookies da sessão do cliente, no campo *access_token*, e no campo *algorithm* é guardado que foi utilizado HMAC. A partir disso, toda requisição que ele fizer a rotas protegidas passará por uma validação desse *access_token* por meio do algoritmo definido, e somente será possível ter acesso ao dado protegido se o token estiver realmente válido.



(a) Requisição com token JWT autenticado por HMAC



(b) Resposta com token JWT autenticado por HMAC

Figura 2: Acesso a dados protegidos por com um token válido gerado por HMAC

2.3 RSA PKCS

O RSA é um algoritmo de criptografia assimétrica, ou seja, trabalha com chave pública e privada.

No projeto, o algoritmo RSA é utilizado para assinar e validar tokens JWT quando o modo de autenticação assimétrica é selecionado. O servidor gera um par de chaves RSA (privada e pública) de 2048 bits seguindo o padrão PKCS#1. A chave privada é usada para assinar o token JWT, garantindo que apenas o servidor pode emitir tokens válidos. A chave pública correspondente é utilizada para verificar a assinatura do token, permitindo que qualquer parte com acesso à chave pública valide a autenticidade do token sem comprometer a chave privada.

O uso do padrão PKCS garante interoperabilidade e segurança, pois define o padding (preenchimento) e o formato dos dados assinados, protegendo contra ataques criptográficos conhecidos. No projeto, a assinatura dos tokens JWT é feita utilizando o algoritmo **RS256, que combina RSA com SHA-256**, conforme especificado pelo PKCS#1.

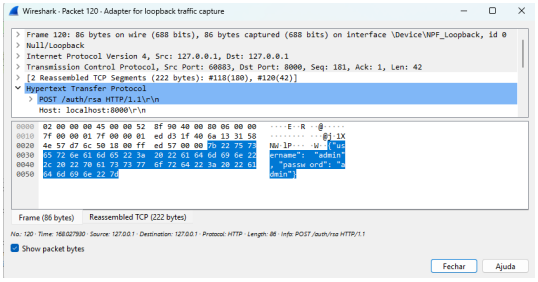
A segurança do uso do RSA reside no fato de que apenas o detentor da chave privada pode criar assinaturas válidas. A chave pública permite a verificação, mas não a criação de novas assinaturas.

Isso faz com que ele seja mais aconselhável que o método por HMAC para ser usado em sistemas distribuídos e microsserviços, pois quando diferentes serviços precisam validar tokens, eles não devem ter a capacidade de gerá-los. O serviço de autenticação mantém a chave privada, enquanto os serviços de recursos usam a chave pública para validação.

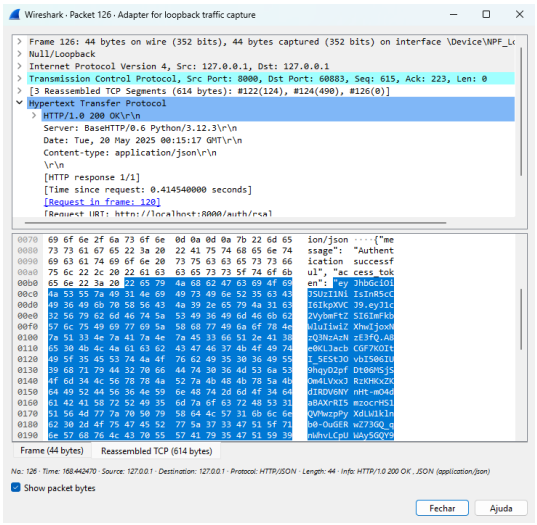
A seguir está um pouco da dinâmica da autenticação por RSA em nossa aplicação:

Primeiramente, ocorre uma solicitação de autenticação do usuário, passando seu nome de usuário e sua senha.

Caso o usuário existir e sua estiver correta, o servidor responder a requisição com status 200 (OK) e devolve um token JWT gerado utiliando RSA.



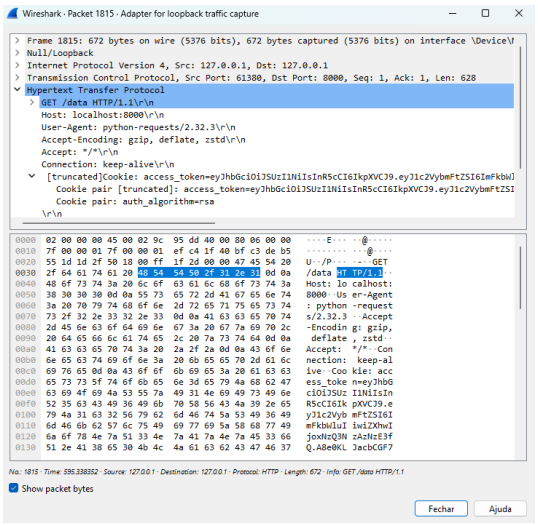
(a) Requisição de autenticação por RSA



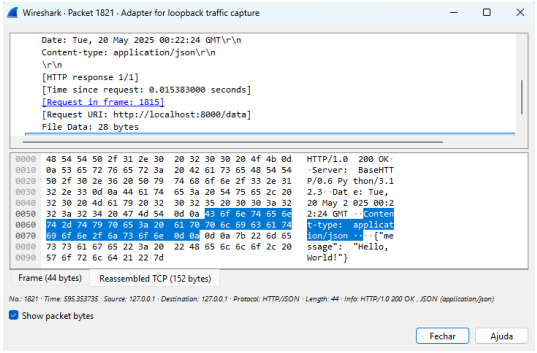
(b) Resposta de autenticação por RSA

Figura 3: Autenticação por RSA

Ao receber esse token, ele é inserido nos cookies da sessão do cliente, no campo *access_token*, e no campo *algorithm* é guardado que foi utilizado RSA. A partir disso, toda requisição que ele fizer a rotas protegidas passará por uma validação desse *access_token* por meio do algoritmo definido, e somente será possível ter acesso ao dado protegido se o token estiver realmente válido.



(a) Requisição com token JWT autenticado por RSA



(b) Resposta com token JWT autenticado por RSA

Figura 4: Acesso a dados protegidos por com um token válido gerado por RSA

2.4 Tentativas inadequadas à acesso de dados

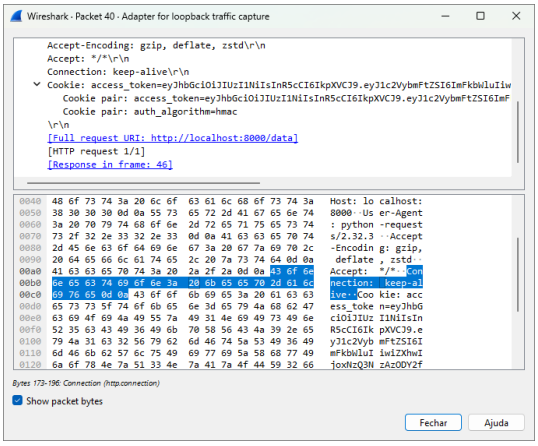
O servidor implementa mecanismos para lidar com tentativas inadequadas de acesso a dados protegidos. Sempre que um cliente tenta acessar o endpoint protegido (no nosso caso, "/data"), o servidor exige a apresentação de um token de acesso válido, que pode ser acessado nos *Cookies*. O token deve ser um JWT assinado corretamente e dentro do seu período de validade.

Quando o servidor recebe uma requisição para um recurso protegido, ele executa as seguintes verificações:

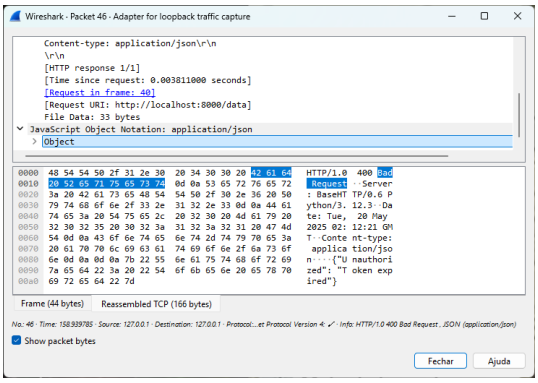
Ausência de Token: Se o token de acesso não estiver presente nos cookies da requisição, o servidor responde imediatamente com o código HTTP 401 (Unauthorized) e uma mensagem indicando que o

token não foi encontrado. Token Inválido: Caso o token esteja presente, mas seja inválido (por exemplo, adulterado, assinado com chave incorreta ou com algoritmo diferente do esperado), o servidor responde com o código HTTP 400 (Bad Request) e uma mensagem informando que o token é inválido.

Token Expirado: Se o token estiver expirado, o servidor também responde com HTTP 401 e uma mensagem específica indicando que o token expirou.



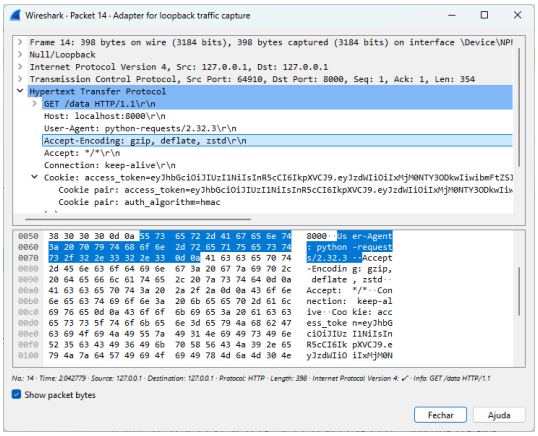
(a) Requisição a dados protegidos com token expirado



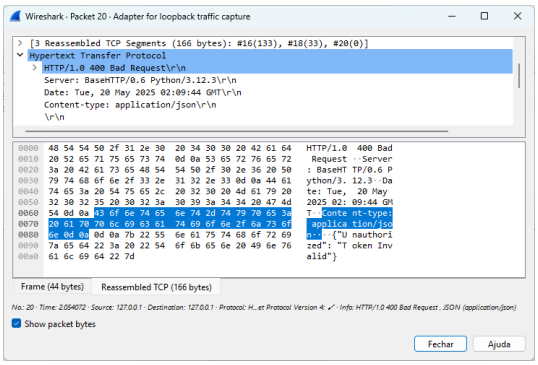
(b) Resposta a tentativa de acesso a dados protegidos com token expirado

Figura 5: Tentativa de acesso a dados protegidos com um token expirado

Token Válido: Apenas quando o token é válido e não expirou, o servidor permite o acesso ao recurso solicitado, respondendo com HTTP 200 e os dados protegidos.



(a) Requisição a dados protegidos com token inválido



(b) Resposta a tentativa de acesso a dados protegidos com token inválido

Figura 6: Tentativa de acesso a dados protegidos com um token inválido

Essas verificações são realizadas por funções específicas de validação de token, que utilizam a biblioteca PyJWT para decodificar e verificar a assinatura e o tempo de expiração do JWT. Caso qualquer uma das validações falhe, o servidor encerra imediatamente o processamento da requisição, garantindo que dados sensíveis não sejam expostos a usuários não autorizados.

Esse tratamento rigoroso de tentativas inadequadas de acesso reforça a segurança da API, prevenindo tanto ataques de força bruta quanto o uso de tokens comprometidos ou expirados.

2.5 Vulnerabilidades e procedimentos de mitigação

Tanto nos cenários de autenticação por meio de HMAC, quanto por RSA, há algumas vulnerabilidades em comum, tais quais:

Exposição da chave secreta: Se a chave for vazada, qualquer atacante pode gerar tokens válidos. Por isso é importante se gerenciado bem o acesso a essas chaves para que evite que as mesmas sejam vazadas ou expostas.

Chave fraca ou previsível: Chaves curtas ou geradas sem entropia suficiente podem ser quebradas por força bruta. Por isso a importância de ocorrer a geração de chaves aleatórias e com tamanhos satisfátórios. Em especial, é adequado para o HMAC chaves de no mínimo 256 bits, e para o RSA no mínimo 2048 bits, que são tamanhos considerados seguros às principais formas de ataque que se conhece atualmente a esses algoritmos.

Mensagens de erro detalhadas: Mensagens diferentes para token inválido e expirado podem ajudar um atacante a entender o sistema. Para propósitos eduacionais, demos um certo detalhamento dos tipos de erro que ocorrem ao tentar validar um token, para que pudesse ser identificado o que nossa aplicação faz em cenários diferentes, isso não é algo aconselhável para aplicativos em produção por ser algo que facilita o trabalho de atacantes.

3 Conclusão

O desenvolvimento deste projeto proporcionou uma compreensão prática e aprofundada dos principais conceitos de segurança aplicados a APIs REST modernas. Ao implementar autenticação baseada em JWT com algoritmos HMAC e RSA, armazenamento seguro de senhas com bcrypt, comunicação protegida por HTTPS e gerenciamento de usuários em banco de dados SQLite, foi possível demonstrar como diferentes camadas de segurança podem ser integradas de forma eficiente em aplicações reais.

Por meio deste trabalho, foi possível entender e aplicar técnicas de segurança na proteção de dados sensíveis em APIs, e permitindo que possamos amplificar os mecanismos de segurança de aplicações que construiremos futuramente.

Referências

[Doc24] Docs. Pyjwt — documentação da versão 2.10.1, 2024. Acessado em 26 maio 2025.