JSON Web Tokens: Um estudo sobre sua aplicação

Amanda de Oliveira Laurindo
Centro Universitário Dinâmica das Cataratas
Foz do Iguaçu, Brasil
amanda96laurindo@outlook.com

Ali Mohamad Termos
Centro Universitário Dinâmica das Cataratas
Foz do Iguaçu, Brasil
termosali963@gmail.com

Luciano Santos Cardoso
Centro Universitário Dinâmica das Cataratas
Foz do Iguaçu, Brasil
luciano.cardoso@udc.edu.br

***Abstract*— The JSON Web Token is a security standard for communicating data between clients and servers. It is structured by a header which has the cryptography norms used, the payload which has the data that is going to be cryptographed and the signature that has the key used to cryptograph the entire token using the cryptographic algorithm defined in the header. Its main advantage is not being dependable in databases to store the core information allowing different applications to share that data with each other, but as a disadvantage the token stays online till its due date which can cause problems in case of a security breach of confidential data of the users.**

***Keywords*—Security Breach; Algorithms; Data Sharing.**

***Resumo ou Resumen*— O JSON Web Token é um padrão para a segurança de comunicação de dados entre clientes e servidores. Ele é composto por um cabeçalho que tem as normas de criptografía usadas, a carga útil que possui os dados para criptografar e a assinatura que possui a chave para criptografar todo o token, usando o algoritmo de criptografia estabelecido no cabeçalho. Sua vantagem principal é não depender de banco de dados para salvar as informações principais permitindo a comunicação entre diferentes aplicações, e como desvantagem, o token fica no ar até a sua data de expiração o que pode causar problemas caso aconteça uma brecha de informações confidenciais dos usuários.**

***Palavras-chave*—Brechas de Segurança. Algoritmos. Compartilhamento de Dados.**

I. INTRODUÇÃO

O JSON (*JavaScript Object Notation*) Web Token (JWT) é um padrão aberto para requisições entre cliente e servidor, sua premissa é oferecer uma troca de dados de forma segura. Existem vários cenários para se usar o JWT, o mais comum é autenticação (processo para verificar a identidade do usuário), já que o JWT guarda informações como a identificação e suas permissões (*roles*).

A. *Anatomia de um JWT*

Um JWT tem 3 partes essenciais na sua composição, o cabeçalho (*header*), a carga útil (*payload*) e a assinatura (*signature*). As duas primeiras partes, sendo o cabeçalho e a carga útil são codificados usando base64, mais especificamente *base64urlEncode*, depois são separados por um ponto e a parte final de assinatura é codificando a mensagem composta pela chave secreta, o cabeçalho codificado e a carga útil codificada usando o algoritmo estabelecido no cabeçalho.

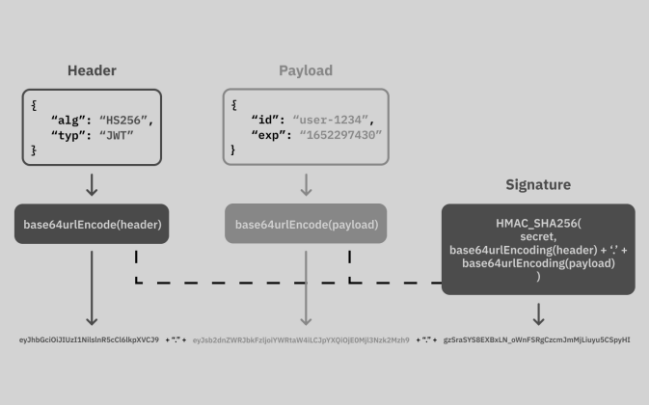

Figura 1 - Diagrama demonstrando a estrutura de um JWT. [1]

1) Cabeçalho: Tem a função de guardar os metadados do JWT, podendo ser composto de até três componentes: O algoritmo (alg), o tipo (*type*) e o identificador de chave (kid). O algoritmo mais comum para assinar JWTs é o algoritmo Código de Autenticação de Mensagem Baseado em Hash (HMAC), ele assina mensagens usando uma combinação de uma chave compartilhada e uma função de criptografia Hash. A criptografia hash funciona convertendo o bloco de dados em uma série de caracteres como mostra a figura 2. A função mais usada atualmente é a SHA-256 que é mais eficiente e segura comparada a SHA-1 da imagem 2, a diferença entre elas é que a SHA-256 possui 256-bit comprimento e a SHA-1 possui 160-bit.

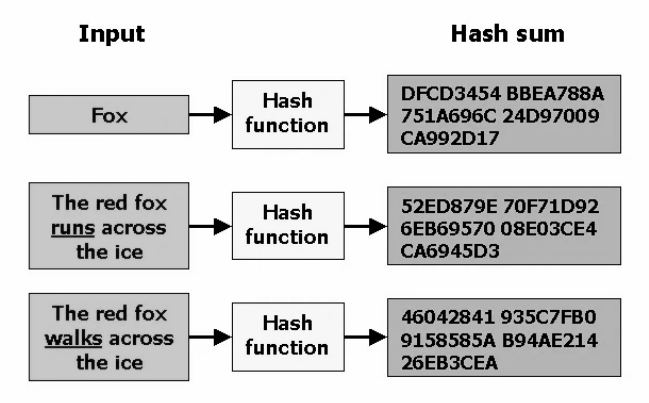

Figura 2 - Diagrama demonstrando como é a saída depois de um bloco de dados passar pela função de criptografia hash SHA-1.[2]

Mas o JWT pode aceitar outros algoritmos como RSASSA e ECDSA usando diferentes combinações de funções de criptografia hash como outras versões de SHA ou as curvas elípticas usadas nos algoritmos ECDSA.

O tipo é opcional e serve para indicar o tipo do JWT que normalmente vai estar definido simplesmente como JWT, mas pode ser definido como o valor at+JWT indicando que é um token de acesso.

O identificador de chave também é opcional, sendo útil quando você tem várias chaves para assinar os tokens e precisa ver a certa para verificar a assinatura. De acordo com Dan Moore[3], "Para uma chave simétrica, o kid pode ser usado para procurar o valor em um cofre secreto. Para um algoritmo de assinatura assimétrico, esse valor permite ao cliente de um JWT procurar a chave pública correta correspondendo à chave privada que assinou o JWT".

2) Carga útil: Nesta sessão vão estar armazenadas as claims (informações) da entidade foco do JWT (geralmente o usuário). Existem três tipos de claims: Registradas, públicas e privadas.

Claims registradas: São claims opcionais mas é fortemente recomendado o uso, ela possui informações extras. Por exemplo: sub (subject) que é a identidade do usuário podendo ser tanto um nome de domínio quanto URL, iss (issuer) que é quem emitiu o token, exp (expiration) e nbf (not before) ambos ditam até quando o token será válido , iat (issued at) que é quando o token foi criado e aud (audience) que é o destinatário do token, quem vai consumi-lo.

Claims públicas: Representam os atributos que usados durante a aplicação ou serviço disponível, são definidos pelo desenvolvedor geralmente incluindo coisas como nome ou usuário, cargos e permissões.

Claims privadas: São atributos que possuem dados que serão compartilhados definidos por um acordo entre as pessoas envolvidas.

3) Assinatura: A assinatura é a combinação de todos os setores que vimos acima mas codificados em base 64 com uma chave secreta ou certificado RSA, ela é essencial para garantir a integridade do token, permitindo garantir que a mensagem não foi alterada e que foi mandada pelo remetente certo prevenindo ataques como *man-in-the-middle* (homem no meio) na qual um hacker malicioso envia um token alterado para o usuário, um exemplo seria enviar um boleto ou fatura falso ao fim de roubar dinheiro da vítima.

B. Vantagens do JWT

O JWT é uma escolha popular de autenticação e método de autorização em SaaS (Softwares como Serviço) e aplicações que funcionam por nuvem. O motivo disso são as características que diferenciam o JWT.

1) Stateless (Sem estado): A autenticação é feita de modo que o próprio token possui todos os dados relevantes, então o servidor não precisa registrar todas as instâncias de sessões dos usuários, por isso é mais fácil para aplicações Saas serem distribuídas entre servidores e microsserviços.

2) *Logins únicos (SSO):* Os JWTs podem oferecer uma ferramenta de SSO (Single Sign-On) onde o usuário pode usar o mesmo token gerado pelo primeiro login para acessar diferentes serviços da aplicações sem precisar ficar entrando na sua conta repetidas vezes.

3) *Queries reduzidas:* Por causa da sua característica de stateless, o servidor não precisa fazer requisições para o banco de dados já que os dados já estão dentro do token.

4) *Compartilhamento de Recursos de Origens Diferentes (CORS):* Permitem o compartilhamento incluindo o JWT nos cabeçalhos de requisições HTTP. Isso é relevante por causa dos ataques de Forjações de Requisições de Sites Diferentes (CSRF), segundo o site oficial da IBM[4] "CSRF utiliza a confiança que um site tenha no navegador de um usuário autenticado para ataques maliciosos. CSRF utiliza links ou scripts para enviar solicitações de HTTP involuntárias para um site de destino onde o usuário está autenticado.".

C. Desvantagens do JWT

A maior desvantagem do JWT é a sua falta de invalidação, o token vai continuar válido desde quando ele foi gerado até sua expiração, então caso precise tirar ele do ar é necessário aumentar a complexidade do seu código implementando técnicas de blacklisting. O que se torna um grande problema caso aconteça um risco de segurança que necessite que o acesso de um usuário precise ser revocado.

Outros problemas incluem o fato do JWT ser praticamente imutável depois de ser gerado, então caso algum dado do usuário mude ele irá ter que logar novamente para atualizar o token e o fato de como todos os dados relevantes são guardados dentro do token, caso o desenvolvedor não balanceie bem que dados inserir no token, ele pode gerar um tráfego de rede maior.

D. Problemas de segurança

Como falado acima, o cabeçalho do JWT aceita vários algoritmos como RSA e HMAC, mas um que não foi mencionado é None (Nada), ele afirma que não possui um algoritmo para realizar a assinatura e é usado para a fase de teste no desenvolvimento do programa. Então se None for um valor permitido para o algoritmo, uma pessoa maliciosa poderia trocar o algoritmo por None, deletando a assinatura e podendo ter acesso aos dados dentro da carga útil. Para evitar isso é essencial rejeitar tokens que possuem o cabeçalho com qualquer capitalização variante de None como none/nOnE/NONE e idem.

Outro problema é algo chamado de Confusão de Chave ou Substituição de Algoritmos, ela só pode acontecer

quando tanto as chaves simétricas e assimétricas tem suporte e a aplicação não verifica se o algoritmo do token recebido é condizente com o algoritmo esperado. No artigo "Sobre a segurança (interna) do Criptografia e Assinatura de Objetos JavaScript"[5] eles afirmam o seguinte: "A vulnerabilidade ocorre se o sistema estiver esperando um token assinado com um dos algoritmos assíncronos. Um attacker talvez abuse da estrutura de verificação API para construir uma assinatura HMAC usando a chave pública do servidor como um segredo compartilhado. No lado do servidor, o sistema passa o token e a chave pública RSA para a função verify() para checar sua veracidade. A biblioteca JOSE, porém, baseia sua decisão de verificação no cabeçalho alg - que neste caso é HMAC. Então ele gera um novo HMAC com a chave pública dada e compara com a assinatura provida". A forma de solucionar isso seria definindo o algoritmo direto no código ou mantendo uma lista com valores aceitáveis que de preferência tenham algoritmos HMAC ou algoritmos assimétricos.

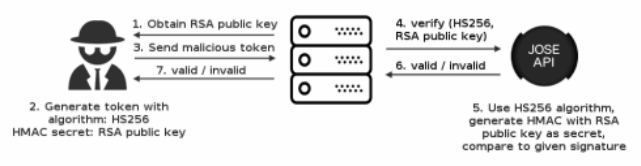

Figura 3 - Demonstração de como um ataque usando a confusão de chave funciona.[5]

Outra vulnerabilidade conhecida é injeção pelo parâmetro kid, na qual geralmente mandam o parâmetro kid e um diretório como por exemplo key/usr/bin/uname, o que força a aplicação a usar o arquivo como chave para verificação na qual o attacker pode adivinhar o valor dela. Na situação que o parâmetro kid seja realmente vulnerável então tem chance que o attacker consiga obter a chave do banco de dados pelo parâmetro kid então ele consegue realizar injeções de SQL, mandando queries SQL para o servidor.

Uma das grandes controvérsias do JWT são a um ataque na qual eles especificamente são os alvos conhecido como Cross-Site Scripting (XSS), na qual o attacker pode roubar o JWT do usuário para ganhar acesso a aplicação, podendo ver todos os dados sensíveis como dados bancários.

E. Medidas de segurança

Umas boas práticas de segurança são: Usar algoritmos fortemente codificados como HMAC-SHA256 ou RSA. Implementar um tempo de expiração curto para mesmo caso aconteça um ataque para o attacker não tiver acesso por muito tempo, com isso entra também a ferramenta de atualizar token gerando novamente. Evitar dados sensíveis nos tokens, no lugar usando um banco de dados para guardar essas informações. Implementar um mecanismo para revocar JWTs quando eles não são mais necessários. Usar uma assinatura digital para confirmar que não foi alterado. Usar HTTPS para garantir que o JWT será transmitido com segurança entre cliente e servidor, isso inclui o cookie ser setado para HttpOnly.

Os tokens geralmente são armazenados no LocalStorage, mas isso traz um enorme problema: ele é facilmente acessado via JavaScript. Para a próxima seção mostraremos algumas soluções usando de base o artigo "Usando tokens JWT de forma segura"[6] que está utilizando a arquitetura Node.js para os seus códigos:

1) *Armazenar o token em memória:* Neste método ao invés de armazenar o token no LocalStorage nós guardamos na memória, que deixa o token fora do alcance de scripts, mas a sua desvantagem é que se o usuário recarregar a página ou trocar de aba, vai ser preciso que o usuário logue novamente.

```
const _token = null

fetch('/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: 'usuario', password: 'senha' })
})
.then(data => data.json())
.then(token => { _token = token })
```

Figura 4 - Função para armazenar o token na memória.

Na primeira linha vemos a variável que será usada para armazenar o token sendo definida como _token e o valor null, ou seja, nada. Em seguida vemos declarado uma função chama fetch ou busca, que mandará uma requisição HTTP para a URL /login, dentro dessa função é definido o corpo da requisição HTTP, primeiro vemos o método da requisição que neste caso é POST, indicando que este token serve para mandar dados ao servidor, em seguida temos o cabeçalho indicando o tipo de dados que estão sendo mandados, neste caso JSON e por fim o corpo da mensagem HTTP, nesta linha ele está pegando o JWT que é um objeto JavaScript e transformando-o em uma string JSON com o comando JSON.stringify. Depois da requisição ser feita os métodos .then são chamados para lidar com a resposta do servidor, primeiro o data => data.JSON() que realiza um parse no corpo da resposta como um JSON e depois podemos ver os dados sendo armazenados na variável _token.

2) Cookies por HttpOnly: A definição de cookie é que ele é um pedaço de informação de até 4kb que ficam salvos no navegador entre abas desde que estejam no mesmo domínio. Ao armazenar o token com a flag httpOnly ativada, o cookies não são mais acessíveis por JavaScript, apenas pelo navegador e requisições. O maior problema com este método é que muitas vezes os JWT acabam passando do limite de 4kb, e apesar de deixarem de ser vítimas de ataque XSS, eles podem passar a ser vítimas de CSRF.

```
const express = require('express');
const app = express();

// preparação de middlewares e etc...

app.post('/login', (req, res) => {
  // realiza o login...
  res.cookie('token', '12345', { maxAge: 5*60*1000, httpOnly: true, sameSite: 'strict' });
  res.send('OK')
})
```

Figura 5 - Função para armazenar o token com ele sendo HttpOnly.

Na primeira linha está sendo importado o módulo express que é muito utilizado em arquiteturas nodes para controlar rotas de servidores, lidar com requisições e mandar respostas. A segunda linha inicializa uma aplicação Express e a armazena na variável app, que é um objeto usado para definir rotas. Em seguida temos a lista que define a rota para requisições HTTP Posts feitas para o endpoint /login. A próxima linha serve para mandar o cookie como resposta, nele é definido seu nome (token), o valor ’12345’ que está simulando o token gerado, o maxAge que é a data de validação do token, neste caso 5 minutos (5 * 60 * 1000 milisegundos), httpOnly: true que habilita a flag para que o cookie seja inacessível para JavaScripts rodando do lado do cliente e por último sameSite: strict que restrita o cookie de ser mandado com requisições cross-site, o que previne ataques CSRF. Por último, o servidor envia uma string ‘OK’ como indicação de que o login foi realizado com sucesso.

F. Comparações com outras tecnologias

Uma das principais comparações é entre OAuth e JWT, com uma diferença crucial em seus propósitos: o JWT é voltado para autenticação e segurança na transferência de informações, enquanto o OAuth foca na autorização. Além disso, os tipos de token diferem: os tokens de acesso do OAuth são opacos e precisam de validação pelo servidor, enquanto os JWTs contêm todas as informações necessárias dentro do próprio token, permitindo verificação local. Outras comparações incluem OIDC (OpenID Connect) e SAML (Security Assertion Markup Language). O OIDC é construído sobre o OAuth 2.0 para autenticação de usuários, possibilitando o SSO em diferentes plataformas. O SAML, por sua vez, define um formato de protocolo e token para autenticação entre provedores de identidade e serviços, usando codificação em XML. Ao escolher entre essas tecnologias, as empresas consideram:

1) *Segurança:* OAuth é ideal para controle detalhado de acesso a dados por aplicações de terceiros, enquanto JWT é melhor para transferir informações de usuários sem manter dados de sessão por muito tempo.

2) Arquitetura: JWT é preferido em micro serviços e sistemas distribuídos devido à sua natureza sem estado. OAuth é mais adequado para autorizações entre diferentes serviços e plataformas. OIDC e SAML são mais indicados para logins únicos, com OIDC se destacando em dispositivos variados e SAML sendo mais adequado para a web.

3) Experiência do usuário (UX): JWT pode oferecer melhor desempenho com menos comunicação com o servidor, enquanto OAuth mantém usuários logados por mais tempo. O SAML pode exigir mais processamento devido ao uso de XML em vez de JSON.

II. Conclusão

Este estudo descreveu o funcionamento dos JSON Web Tokens, na qual eles podem ser versáteis mas também cheios de vulnerabilidades em sua segurança se o programador não levar em conta questões como essa e se preparar para todos os casos. Mas com todas as precauções de segurança e um bom planejamento de recursos, o JWT pode se tornar uma ferramenta essencial para o seu projeto.

Para um trabalho futuro envolvendo JWTs faríamos um sistema que precisasse de login para salvar as credenciais de cadastro do usuário e salvar a sua sessão de acesso, com isso o JWTs teria em sua carga útil dados como o nome do usuário, a sua senha criptografada e o papel do seu usuário, neste caso cliente ou administrador.

Agradecimentos

Oferecemos os nossos agradecimentos aos nossos professores, em especial ao nosso professor orientador que nos ajudou a redigir este artigo

Referências

[1] A. Rich, “JWT claims”, Stytch, https://stytch.com/blog/jwt-claims/.(Out. 3.2023)

[2] K. Nagaraj, “Security hash Algorithm 1(SHA-1): A Comprehensive Overview”, Medium, https://cyberw1ng.medium.com/secure-hash-algorithm-1-sha-1-a-comprehensive-overview-2023-7be2ca9a06eb. (Mar. 3. 2023)

[3] D. Moore, “Components of JWTs Explained”, Fusion Auth, https://fusionauth.io/articles/tokens/jwt-components-explained#:~:text=The%20payload%2C%20or%20body%2C%20is,roles%20or%20other%20authorization%20info.

[4] “Prevenção de ataques Cross-site Request Forgery (CSRF)”, IBM, https://www.ibm.com/docs/pt-br/sva/10.0.7?topic=configuration-prevention-cross-site-request-forgery-csrf-attacks (Jan. 30. 2024)

[5] D. Detering, J. Somorovsky, C. Mainka, V. Mladenov e J. Schwenk. “On The (In-)Security Of JavaScript Object Signing And Encryption”. Association for Computing Machinery. https://dl.acm.org/doi/abs/10.1145/3150376.3150379. (Nov. 16. 2017)

[6] L. Santos. “Usando tokens JWT de forma segura”. Lucas Santos.https://blog.lsantos.dev/jwt-seguro/. (Ago. 18. 2022.)

[7] “OAuth vs. JWT: Ultimate Comparison”, Permify, https://permify.co/post/oauth-jwt-comparison/. (Jun. 26. 2024)