Estudo Comparativo de Mecanismos de Segurança Aplicados à Autenticação e Autorização em Sistemas Web

Rafael Strack[1], **Adriano Ferrasa**[1]

[1]Departamento de Informática – Universidade Estadual de Ponta Grossa (UEPG)
84.030-900 – Ponta Grossa – PR – Brasil

rafa_strack@hotmail.com, ferrasa@uepg.br

Abstract. *This article presents a comparative study of authentication and authorization mechanisms in web systems, aiming to provide an in-depth analysis of the available options and their characteristics. The study describes the most commonly used methods such as passwords, tokens, OAuth and OpenID Connect, analyzing their advantages and disadvantages. Additionally, sequence diagrams are provided to illustrate the usage flow of each method. Finally, a comparison of the studied methods is conducted, evaluating their effectiveness in terms of security. Proper understanding of these mechanisms is crucial to ensure the security of web systems and guide the correct choice in future projects.*

Resumo. *Este artigo apresenta um estudo comparativo dos mecanismos de autenticação e autorização em sistemas web, visando fornecer uma análise aprofundada das opções disponíveis e suas características. O estudo descreve os métodos mais utilizados, como senhas, tokens, OAuth e OpenID Connect, analisando suas vantagens e desvantagens. Além disso, são apresentados diagramas de sequência para ilustrar o fluxo de utilização de cada método. Ao final, é realizada uma comparação dos métodos estudados, avaliando sua eficácia em termos de segurança. A compreensão adequada desses mecanismos é fundamental para garantir a segurança dos sistemas web e orientar a escolha correta em projetos futuros.*

1. Introdução

Com a expansão da internet, os sistemas web assumiram um papel crucial no cotidiano de bilhões de pessoas em todo o mundo. Desde o uso de aparelhos domésticos até o gerenciamento de negócios online, esses sistemas se tornaram indispensáveis para diversas atividades [Greengard 2015]. No entanto, a segurança desses sistemas é uma preocupação constante para desenvolvedores e usuários, pois há uma série de ameaças e vulnerabilidades que podem comprometer sua integridade.

A fundação OWASP (*Open Worldwide Application Security Project*) atualiza regularmente um relatório chamado OWASP Top 10, onde são descritos os 10 riscos de segurança mais críticos em sistemas web. Na última edição, realizada em 2021, a categoria que ficou em primeira colocação foi a quebra de controle de acesso. Em sétima colocação, ficou a categoria de falhas de identificação e autenticação [OWASP 2021]. Esses problemas são diretamente relacionados aos processos de autenticação e autorização de usuários, os quais são essenciais para garantir a proteção adequada dos sistemas.

De modo geral, a autenticação é o processo de validação de usuários, enquanto a autorização é o método que fornece as permissões de acesso corretas aos recursos para usuários previamente autenticados [Tumin e Encheva 2012]. Atualmente, existem diversos mecanismos de autenticação e autorização de usuários disponíveis, como senhas, *tokens*, autenticação multifator, OAuth, OpenID, entre outros. Cada um desses mecanismos apresenta características distintas, pontos positivos e negativos, sendo fundamental garantir a correta implementação dos mecanismos escolhidos, de forma a assegurar a efetividade da segurança dos sistemas web.

Diante desse contexto, o presente trabalho tem como objetivo realizar um estudo comparativo dos diferentes mecanismos de autenticação e autorização, com o propósito de fornecer uma análise aprofundada que auxilie na escolha adequada desses mecanismos em projetos de sistemas web. O estudo visa oferecer uma compreensão ampla das características, pontos fortes e limitações de cada mecanismo, permitindo a seleção correta e a implementação eficiente das medidas de segurança necessárias.

2. Autenticação e Autorização

Na maioria dos sistemas web, é necessário realizar um controle de acesso para que somente certos usuários possam acessar recursos protegidos. Para isso, o mecanismo de controle de acesso depende de dois processos relacionados: a autenticação e a autorização [Sullivan e Liu 2011].

A autenticação pode ser definida como o processo de confirmação de identidade. Em sistemas web, devido a falta de conhecimento do mundo real, este processo pode não ser simples [Chapman e Chapman 2012]. Existem três grupos de fatores amplamente utilizados para confirmar a identidade de um usuário: algo que o usuário sabe, algo que o usuário é e algo que o usuário possui. No primeiro grupo, inclui-se as senhas, PINs (*Personal Identification Number*) e frases secretas. No segundo grupo, inclui-se certificados digitais, *smart cards* e *tokens* de segurança. O terceiro grupo inclui técnicas biométricas, como como impressões digitais, reconhecimento facial ou de voz, entre outras [Sullivan e Liu 2011].

De forma complementar, a autorização é o processo pelo qual o sistema verifica se um usuário previamente autenticado possui permissão para acessar um recurso ou executar uma determinada ação [Spilca 2020]. Ela pode ser realizada de várias formas, porém as mais comuns são as baseadas em usuários, perfis (*roles*) e com o protocolo OAuth [Chapman e Chapman 2012]

2.1. Autenticação Básica HTTP

A autenticação básica HTTP (*Hypertext Transfer Protocol*) foi definida na especificação HTTP/1.0 [Nielsen et al. 1996], porém tornou-se um padrão na RFC 2617 [Franks et al. 1999]. Neste tipo de autenticação, o servidor web recusa uma transação caso o cliente não esteja autenticado, desafiando-o para obter um nome de usuário e senha válidos. Este desafio de autenticação é iniciado retornando o status HTTP 401 (não autorizado) e especificando o domínio de segurança (*security realm*) a ser acessado, com o cabeçalho `WWW-Authenticate`. Ao receber o desafio, o cliente abre uma caixa de diálogo para que o usuário insira as credenciais para acesso ao domínio. O cliente então junta as informações de usuário e senha, colocando dois pontos entre

eles, e os codifica usando o método de codificação base-64. Estas credenciais codificadas são colocadas no cabeçalho `Authorization`, e então a requisição é enviada para o servidor, que fará a validação das credenciais e, caso validadas, retorna-se o status HTTP 200 (OK) [Gourley e Totty 2002] (Figura 1).

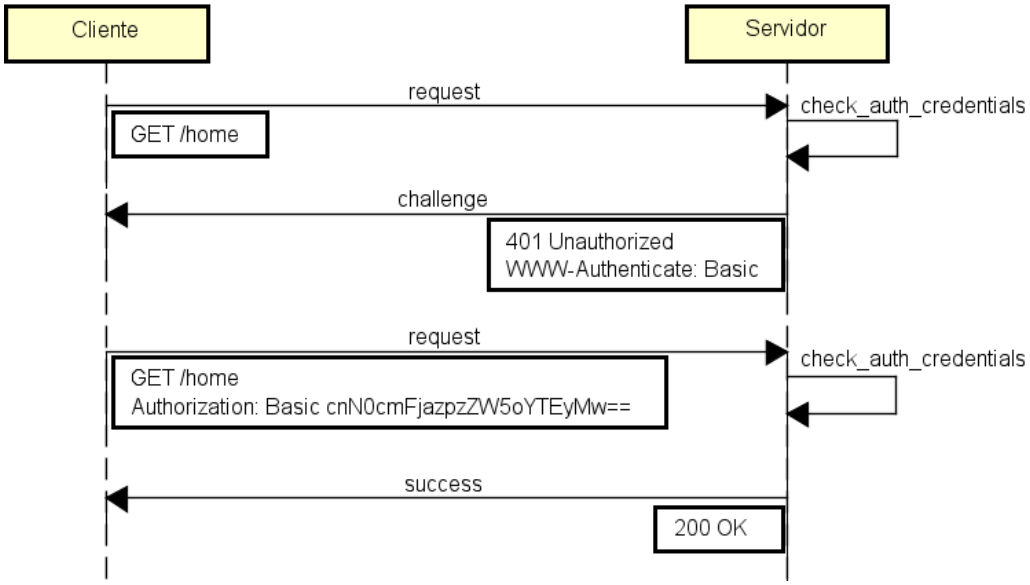

Figura 1. Exemplo de autenticação básica HTTP

A diretiva de domínio (*realm*) utilizada nas autenticações HTTP define os espaços de proteção do sistema web. Esses domínios permitem que os recursos protegidos sejam particionados, cada um com seu próprio esquema de autenticação e/ou autorização [Franks et al. 1999].

2.2. Autenticação *Digest* HTTP

A autenticação *Digest* foi proposta na RFC 2019 [Hallam-Baker et al. 1997], porém também tornou-se um padrão na RFC 2617. Foi desenvolvida para ser uma alternativa mais compatível e segura para a autenticação básica, corrigindo as falhas mais graves da mesma, como a falta de criptografia de senhas, vulnerabilidade a captura e repetição de pacotes e proteção contra vários outros tipos comuns de ataques [Gourley e Totty 2002].

Assim como a autenticação básica HTTP, a *Digest* é baseada no paradigma desafio-resposta [Shekh-Yusef et al. 2015]. A diferença é que foram adicionados diversos parâmetros nos cabeçalhos, para identificação única de desafios, nível de qualidade de proteção, especificação de algoritmo de hashing utilizado entre outros recursos [Chapman e Chapman 2012]. Por padrão, o algoritmo utilizado é o MD5, porém na RFC 7616 foram adicionados e recomendados os algoritmos SHA-256 e SHA-512/256 [Shekh-Yusef et al. 2015].

O parâmetro `response` é a principal parte do cabeçalho `Authorization`: ele contém uma concatenação criptografada de dados da requisição, como nome do usuário, `realm`, senha, método HTTP, URL, entre outros parâmetros, todos separados por dois pontos. [Chapman e Chapman 2012]. O cliente realiza o cálculo resultante no valor de `response`, assim como o servidor, que compara o valor calculado com o valor recebido. Caso as credenciais sejam válidas, o servidor retorna o status HTTP 200 (OK)

e o cabeçalho `Authentication-Info`, que contém parâmetros utilizados para uma futura autenticação, autenticação mútua e reenvio de parâmetros para confirmação de legitimidade (Figura 2).

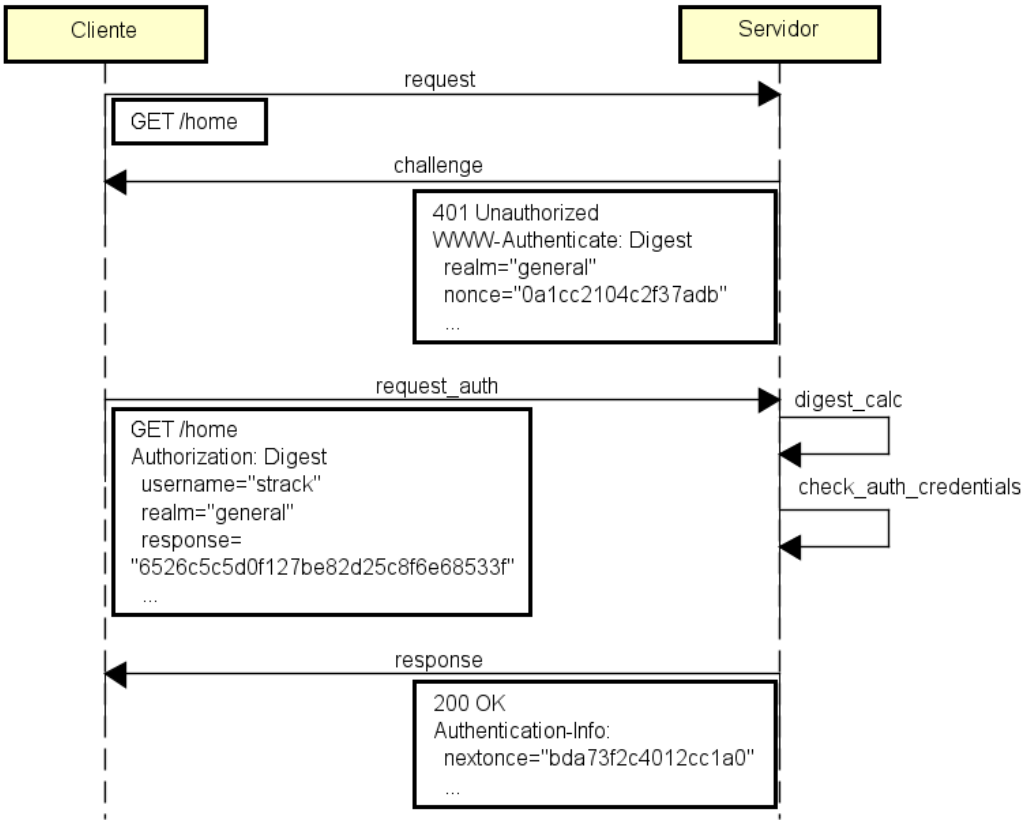

Figura 2. Exemplo de autenticação *Digest*

2.3. Autenticação Baseada em Sessão

Uma sessão web é uma troca de informações semipermanente entre um cliente e um servidor web [Calzavara et al. 2018]. O mecanismo de gerenciamento de estados para o HTTP, baseado em sessões, foi especificado na RFC 2109 [Montulli e Kristol 1997] com sua versão mais atual especificada na RFC 6265 [Barth 2011]. Este mecanismo utiliza o termo *cookie* para se referir às informações de estados que são passadas entre o servidor e o cliente, e salvas no cliente [Montulli e Kristol 1997].

A autenticação baseada em sessão é um dos métodos mais comuns de autenticação em sistemas web. Neste método, após o envio de credenciais de acesso e validação do usuário por meio de uma requisição HTTP, o servidor gera um *cookie*, armazena-o e envia-o pelo cabeçalho `Set-Cookie` da resposta para o cliente. O cliente salva o valor, que é enviado no cabeçalho `Cookie` em toda requisição para o mesmo servidor de origem [Papathanasaki et al. 2022] (Figura 3). Uma prática comum é utilizar *strings* aleatórias, chamadas *session identifiers*, para identificar a sessão. Elas precisam ser longas o suficiente para ser inviável adivinhá-las [Drhová 2018].

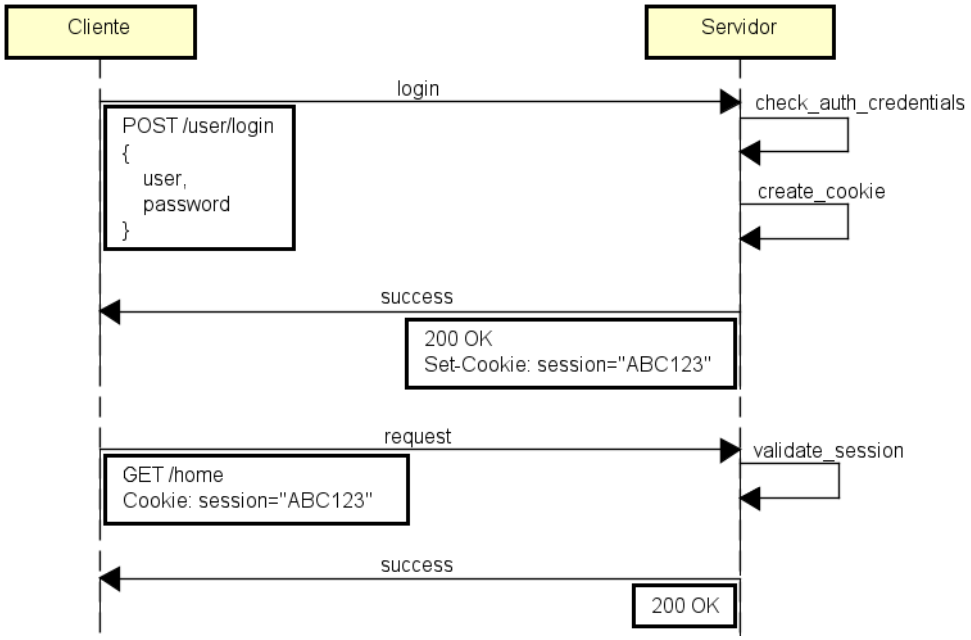

Figura 3. Exemplo de Autenticação baseada em sessão

2.4. Autenticação Baseada em Token

Tokens são itens utilizados para identificar e autenticar usuários. São palavras assinadas e não criptografadas, que carregam informações sobre um usuário autenticado [Balaj 2017]. O padrão mais utilizado deste tipo de autenticação é o JSON *Web Token* (JWT), que foi proposto na RFC 7519. O JWT é um formato compacto de representação de reinvindicações (*claims*), destinado a ambientes com restrição de espaço, como cabeçalhos HTTP e parâmetros de consulta de URI [Jones et al. 2015b].

Um JSON Web Token é composto por cadeias de caracteres codificadas em base64url, separadas por ponto. Geralmente possuem 3 cadeias: o cabeçalho, que contém informações a respeito do tipo de mídia do JWT e da criptografia usada para assinar o *token*; a carga útil (*payload*), que contém as informações sobre as *claims*, que são conjuntos de declarações sobre uma entidade (geralmente um usuário); e a assinatura, que é a concatenação dos *hashes* gerados a partir das outras duas cadeias com uma chave secreta ou certificado, utilizada para verificação de integridade do *token* [Montanheiro et al. 2017].

Neste método, após o envio de credenciais de acesso e validação do usuário por meio de uma requisição HTTP, o servidor gera um *token*, que é enviado para o cliente e pode ser salvo em *cookies* ou no armazenamento local [Montanheiro et al. 2017]. Também pode ser enviado um *token* de atualização (*refresh token*), para a obtenção de um novo *token* de acesso assim que o atual expirar. Ao possuir o *token*, quando uma requisição é feita ao servidor o mesmo deve ser enviado no cabeçalho `Authorization`, precedido pela palavra *Bearer*. [Hardt 2012]. (Figura 4)

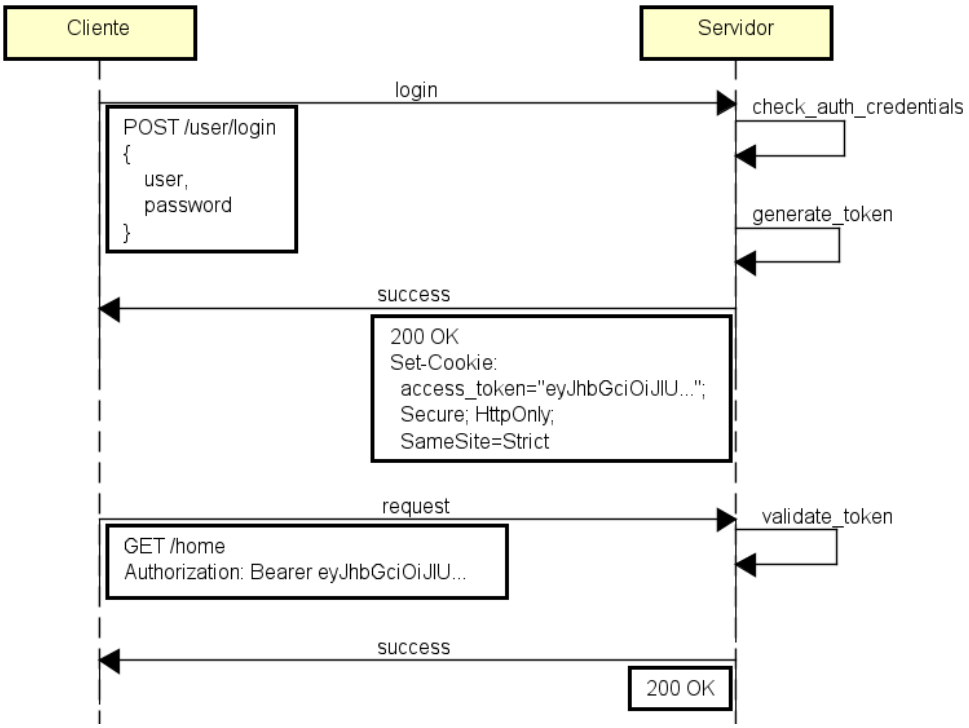

Figura 4. Exemplo de Autenticação baseada em *token*

2.5. OAuth e OAuth 2.0

O protocolo OAuth foi especificado na RFC 5849, fornecendo um método para clientes acessarem recursos de um servidor em nome de um proprietário de recurso e também um processo para que os usuários finais autorizem o acesso de terceiros aos seus recursos de servidor sem compartilhar suas credenciais [Hammer-Lahav 2010]. Este método de autenticação funciona seguindo as seguintes etapas:

- O usuário solicita o serviço de um sistema, chamado de cliente;
- O cliente, tendo previamente configurado acesso ao servidor de recursos, possuindo um identificador e um segredo compartilhado, envia uma solicitação a este servidor para receber um *token* de solicitação;
- O servidor valida a solicitação, enviando o *token* de solicitação não-autorizado no corpo da resposta HTTP;
- O cliente redireciona o agente do usuário para o servidor, que solicita o *login* do usuário e depois a autorização para o cliente acessar o recurso;
- O cliente recebe um *token* de solicitação, que utiliza em uma nova solicitação por *token* de acesso. Estas requisições são realizadas por meio de um canal TLS;
- O servidor valida o *token* de solicitação e envia o *token* de acesso ao cliente e opcionalmente um *token* de atualização;
- O cliente utiliza o *token* de acesso para solicitar os recursos do servidor.

Na RFC 6749 foi proposto o protocolo OAuth 2.0, tornando obsoleta sua versão anterior. Esta versão possui poucas semelhanças com a versão anterior, utilizando os mesmos princípios mas com fluxo diferente e abrangendo mais casos de uso, além de estabelecer o uso do protocolo HTTPS, possuindo todas as mensagens criptografadas. Por este motivo os dois protocolos não são compatíveis, coexistindo e podendo ambos

serem suportados pelos sistemas [Hardt 2012]. Diferente da versão 1.0, que depende de assinaturas criptografadas para cada requisição, a versão 2.0 utiliza *tokens* portadores (*bearer tokens*) que são protegidos devido ao uso do TLS [Siriwardena 2014].

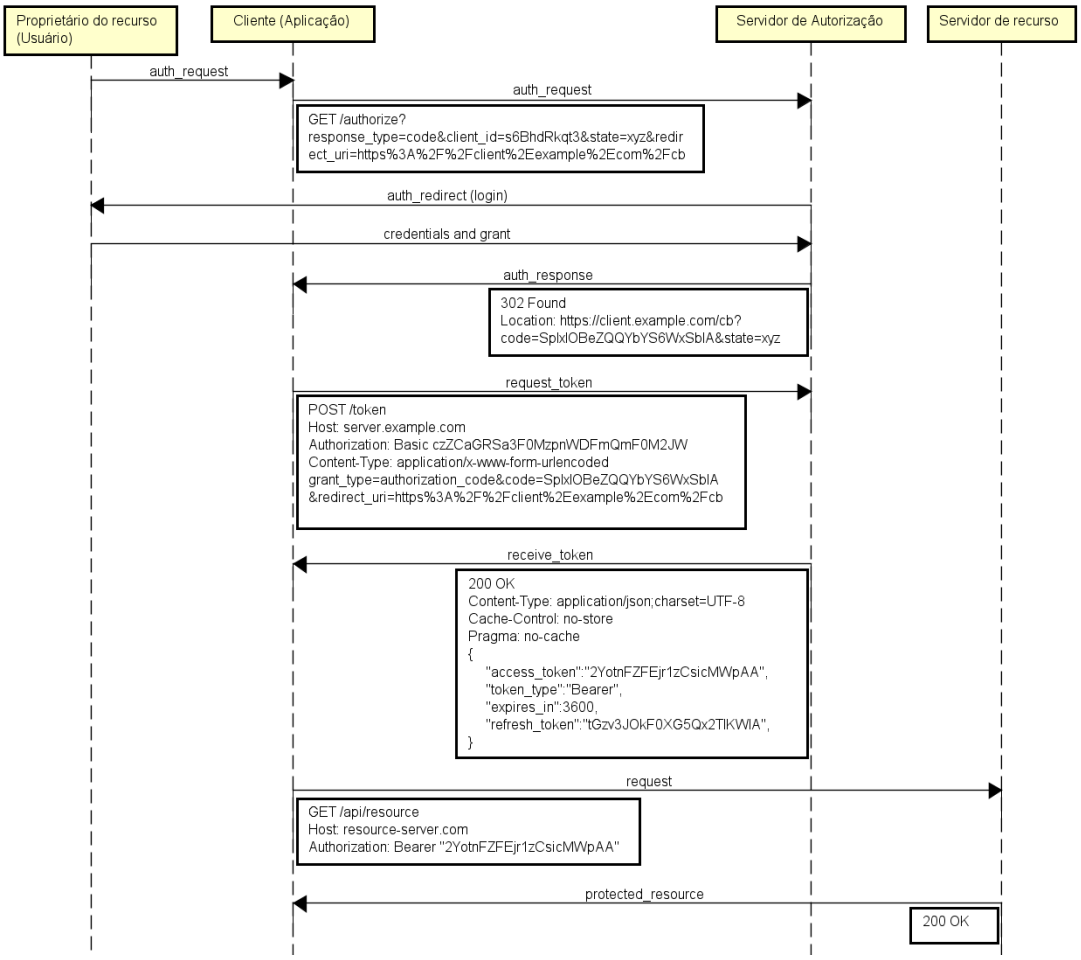

Figura 5. Exemplo de autenticação utilizando OAuth 2.0

Um dos fluxos de funcionamento disponíveis, por código de autorização, é descrito pelas seguintes etapas (Figura 5):

- O cliente (sistema solicitando acesso) direciona o proprietário do recurso para o *endpoint* de autorização, no servidor de autorização, passando o identificador do cliente, o estado local, a URI de identificação e o escopo solicitado;
- O servidor de autorização autentica o proprietário do recurso e espera a concessão de acesso aos recursos para o cliente;
- Após o proprietário do recurso garantir o acesso ao cliente, o servidor de autorização redireciona para o cliente utilizando a URI fornecida pelo cliente anteriormente, juntamente com o código de autorização;
- O cliente solicita um *token* de acesso ao servidor de autorização, utilizando o código de autorização recebido anteriormente;
- O servidor de autorização autentica o cliente, valida o código de autorização e garante que o URI de redirecionamento recebido corresponde ao utilizado anteriormente. Se válido, responde com um *token* de acesso e opcionalmente com um *token* de atualização;

- O cliente utiliza o *token* de acesso para solicitar os recursos do servidor [Hardt 2012].

2.6. OpenID Connect

O OpenID Connect (OIDC) é um protocolo para fornecimento de identificação que permite que dados de usuários sejam transmitidos de forma segura de um provedor para um cliente, com o consentimento do usuário. É uma camada sobre o protocolo OAuth 2.0, fazendo uso de todos seus conceitos, *tokens* e fluxos, com a adição do fornecimento de atributos do usuário. Esses atributos podem ser fornecidos para um sistema por meio de uma API RESTful `userinfo` ou por meio de um *token* de identificação [Biehl 2019].

O fluxo de código de autorização do OIDC é similar ao fluxo do OAuth 2.0, com a adição do *token* de identificação, fornecido ao cliente pelo servidor de autorização (chamado de provedor OIDC). Este *token* é no formato JWT e possui declarações sobre a autenticação de um usuário (Figura 6). Estas declarações também podem ser acessadas realizando uma requisição ao *endpoint* `userinfo` [Sakimura et al. 2014].

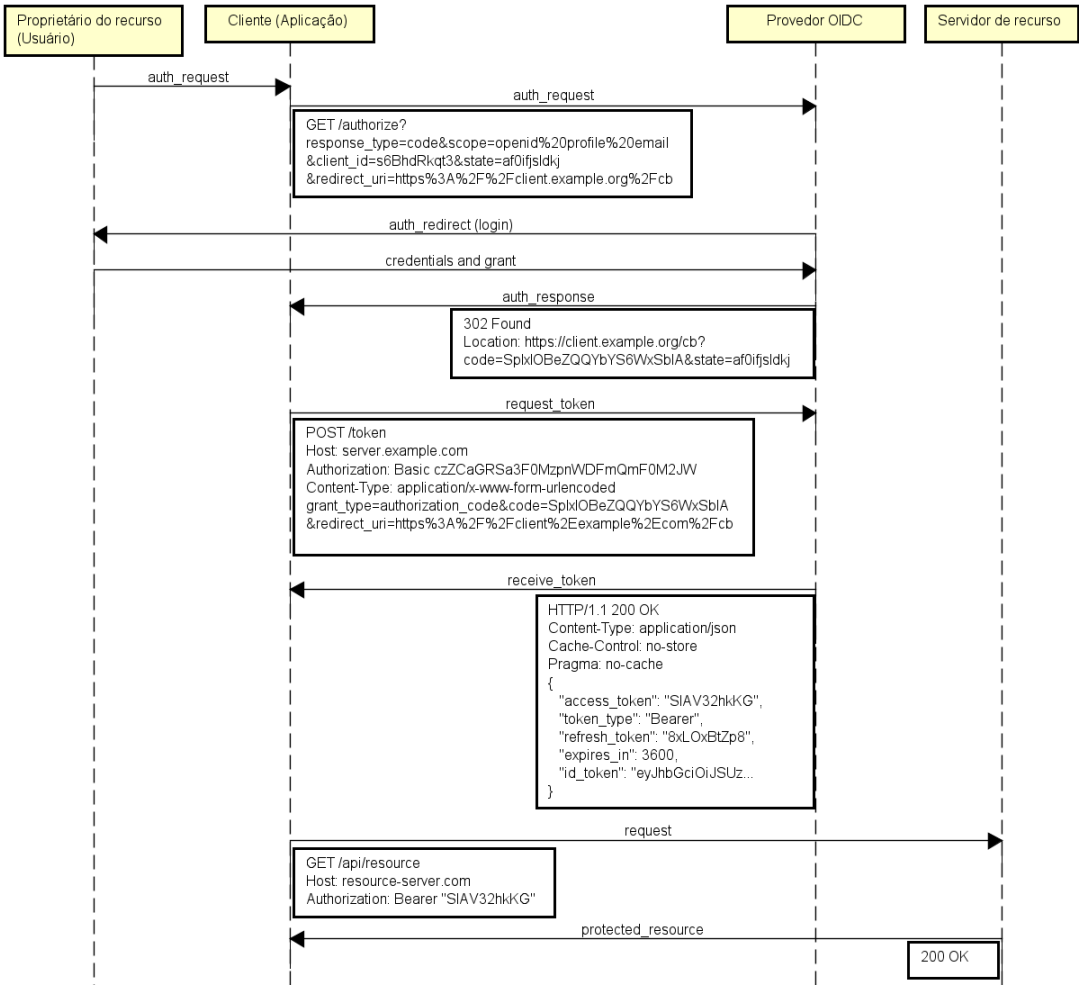

Figura 6. Exemplo de autenticação utilizando OpenID Connect

3. Materiais e Métodos

Para o desenvolvimento do presente estudo na área de segurança da informação, utilizou-se uma abordagem qualitativa. Os dados coletados relativos aos mecanismos de autenticação e autorização são descritivos em sua totalidade, fornecendo as características de cada um.

A coleta de dados foi realizada através de uma revisão sistemática da literatura, buscando artigos científicos publicados em revistas e conferências internacionais na área de segurança da informação. Além disso, foram consultados livros e documentos técnicos, como RFCs (*Research for Comments*), emitidos pela IETF (*Internet Engineering Task Force*).

Durante a coleta de dados, foi dada ênfase aos aspectos relacionados à usabilidade, nível de segurança, vulnerabilidades, implementação e obsolescência. Esses critérios foram escolhidos com o objetivo de fornecer uma análise comparativa dos métodos investigados. As informações coletadas foram registradas em uma tabela comparativa, utilizando como ponto principal as vulnerabilidades de cada método estudado, que acaba por implicar em outros pontos importantes a se considerar na escolha de um método para implementar em um sistema web. Assim, permite-se a análise das características de cada método, possibilitando ao desenvolvedor escolher de forma mais assertiva quais ferramentas são mais indicadas para cada caso de uso.

4. Resultados e Discussão

A autenticação básica HTTP é simples e de fácil implementação, porém não possui mecanismos de garantia efetiva de segurança. As credenciais do usuário podem ser facilmente decodificadas, visto que a codificação base64 é facilmente reversível, podendo ser realizada em poucos segundos. Também é possível realizar ataques de repetição, visto que terceiros podem capturar pacotes e replicá-los, mesmo que codificados, podendo obter acesso ao sistema. Este tipo de autenticação não possui proteção contra *proxies* ou *middlewares*, que podem facilmente modificar o corpo da mensagem, e também são vulneráveis a servidores falsificados, que se passam por outros para realizar o roubo de credenciais [Gourley e Totty 2002].

Além destes pontos negativos, a autenticação básica não possui desconexão, pois o navegador mantém os dados em *cache* por padrão, até ser fechado. A documentação deste método cita a sua falta de segurança, indicando seu uso somente para propósitos simples de identificação, sem risco de acesso a informações sensíveis ou valiosas [Reschke 2015].

Apesar da grande melhora de segurança em relação a autenticação básica, a autenticação *Digest* ainda possui diversos riscos de segurança. Os cabeçalhos `WWW-Authenticate` e `Authorization` possuem certo nível de proteção a manipulação, porém todos os outros cabeçalhos não possuem. Ataques de repetição podem ser realizados se a implementação de identificadores únicos por desafio não for realizada. Caso não seja estabelecida nenhuma política de força de senha, podem ser realizados ataques de dicionário, tentando adivinhar a senha e outros parâmetros, visto que o nome do usuário é obtido sem esforço. Se a requisição passar por *proxies* hostis ou comprometidos, o cliente pode ficar vulnerável a ataques *man-in-the-middle* [Gourley e Totty 2002].

Assim como a autenticação básica, a Digest não possui desconexão, possuindo o mesmo problema citado anteriormente. Sua documentação cita que, pelos padrões modernos de criptografia, este método é fraco, porém um substituto muito superior a autenticação básica [Shekh-Yusef et al. 2015].

Em relação à autenticação baseada em sessão, a grande vantagem é a diminuição do envio das credenciais do usuário nas requisições, diminuindo a janela de ataques, já que os *cookies* são utilizados para a validação das requisições. Por outro lado, os cookies podem ser lidos por outros aplicativos, tornando o sistema exposto a ataques *Cross Site Scripting* (XSS) e *Cross Site Request Forgery* (CSRF). Para evitar ataques XSS, pode-se definir no *cookie* a *flag* `http-only`, que faz com que o acesso por APIs do lado cliente seja negado [Papathanasaki et al. 2022].

Existe a possibilidade do ataque de fixação de sessão, onde o atacante insere seu próprio valor dentro do *cookie*, esperando o usuário entrar no sistema com esse identificador. Se o servidor aceitá-lo, o atacante obtém acesso ao servidor, podendo obter informações privadas sobre a vítima e realizar ações usando sua identidade [Drhová 2018].

Se não forem transportados por um canal seguro, como TLS, as informações nos cabeçalhos referentes aos *cookies* podem ser visualizadas. Por este motivo, além de enviados por um canal seguro, os *cookies* devem ser criptografados e assinados pelo servidor[Barth 2011].

A autenticação baseada em sessão é *stateful*, contendo todos os identificadores de sessão do lado do servidor, causando uma sobrecarga no mesmo e dificultando o gerenciamento dessas sessões em múltiplos servidores, diferentemente das outras técnicas abordadas, que são *stateless* [Balaj 2017].

Sobre a autenticação baseada em *tokens*, mais especificamente utilizando JWT, que é a mais comum, pode-se listar algumas vulnerabilidades, caso boas práticas não sejam aplicadas. Caso seja validado pelo servidor o uso de JWTs sem assinatura, pode-se sofrer ataque de remoção de assinatura, removendo a terceira parte do *token* e alterando seu cabeçalho. Caso os *tokens* sejam armazenados em *cookies*, é possível a realização de ataques CSRF, além de ataques XSS, caso a *flag* `http-only` não seja declarada [Peyrott 2018]. Caso a chave de assinatura do JWT possua baixa entropia, um ataque de força-bruta pode ser realizado para descobri-la [Jones et al. 2015a].

O protocolo OAuth quando foi publicado, em 2007, tornou-se rapidamente o padrão na indústria para delegação de acesso na web. Porém, teve problemas no domínio empresarial, devido ao seu desempenho. A comunidade percebeu que o protocolo não era escalável: exige gerenciamento de estado em diferentes etapas, gerenciamento de credenciais temporárias e não fornece isolamento do servidor de autorização do próprio servidor de recursos protegidos [Noureddine e Bashroush 2011]. O OAuth 2.0 resolveu estes problemas, facilitando o fluxo ao substituir as assinaturas por *bearer tokens*, utilizando TLS durante todo o fluxo, não somente no *handshake* inicial e definindo o servidor de autorização separadamente do servidor de recurso, que traz maior flexibilidade [Siriwardena 2014].

Em relação a ataques, ambos são suscetíveis a CSRF. A versão 1.0 é mais suscetível devido a falta de uso de TLS em todo seu fluxo, enquanto a versão 2.0 é suscetível

a este ataque caso as diretrizes de implementação não forem seguidas [Fett et al. 2016]. Também acaba se tornando suscetível a ataques de *phishing* e *spoofing*, caso não forem seguidas as diretrizes [Lodderstedt et al. 2013]

O OpenID Connect, por ser uma camada sobre o protocolo OAuth 2.0, possui muitas vulnerabilidades em comum com este. Porém, o OIDC possui algumas em particular que são aplicadas somente a ele: ataques de falsificação de solicitações no servidor, que são facilitados pela descoberta *ad hoc* e pelos recursos de registro dinâmico; ataques de injeção, permitidos devido aos mesmos recursos; ataques CSRF, devido ao recurso de inicialização de login de terceiros e ataques ao /emphtoken de identificação, que não existe no OAuth [Fett et al. 2017].

Da mesma forma, algumas das vulnerabilidades do OAuth 2.0 não são aplicadas ao OIDC: suas configurações são menos propensas a ataques de redirecionamento aberto, uma vez que *placeholders* não são permitidos em URIs de redirecionamento; o TLS é obrigatório para algumas mensagens no OIDC, enquanto é opcional no OAuth 2.0; o valor `nonce` pode evitar alguns ataques de repetição quando o valor do estado não é usado ou vaza para um invasor [Fett et al. 2017].

Tabela 1. Tabela comparativa entre aspectos de ferramentas de autenticação e autorização

Mecanismo	Nível de Segurança	Vulnerabilidades	Obsolescência
HTTP Basic Authentication	Muito baixo	Força-Bruta, Repetição, CSRF *Man-in-the-middle Sniffing*, *Spoofing*	Não recomendado para sistemas modernos
HTTP Digest Authentication	Baixo	Força-Bruta Repetição, CSRF *Man-in-the-middle Sniffing*, *Spoofing*	Não recomendado para sistemas modernos
Session-Based Authentication	Médio	Força-Bruta, CSRF, XSS (sem HTTPS), Fixação de sessão, *Sniffing*	Amplamente utilizado mas possui opções modernas disponíveis
Token-Based Authentication	Alto	Força-Bruta, CSRF XSS, Remoção de assinatura	Amplamente utilizado mas possui opções modernas disponíveis
OAuth e OAuth 2.0	Muito Alto	CSRF, Repetição, Redirecionamento, *spoofing*	OAuth: Obsoleto porém ainda aplicável OAuth 2.0: Amplamente utilizado e recomendado
OpenID Connect	Muito Alto	CSRF, Injeção, *spoofing*	Amplamente utilizado e recomendado

A Tabela 1 sumariza os dados levantados. O nível de segurança implica das vulnerabilidades apresentadas sobre cada método, assim como a obsolescência. Devido à baixa

segurança, as autenticações HTTP básica e Digest são recomendadas somente a propósito de identificação [Franks et al. 1999], assim como a autenticação baseada em sessão. Caso o uso seja para acesso a conteúdo privado, indica-se a utilização de um método mais recente e com maior segurança.

Entre os métodos abordados neste estudo, o que demonstra ser mais completo em relação a segurança e recursos é o OpenID Connect: utiliza como base o OAuth 2.0, um padrão seguro e amplamente utilizado de autorização, com adição de uma camada de autenticação, além de melhoras em relação a definições e segurança, dada a obrigatoriedade do uso de SSL. Apesar disso, deve-se levar em consideração a complexidade de implementação e a necessidade de cada caso de uso, tornando a escolha do mecanismo a ser utilizado uma decisão dos desenvolvedores. Independente de qual a escolha, deve-se seguir as diretrizes impostas na especificação de cada mecanismo, para uma menor vulnerabilidade.

5. Conclusão

Este trabalho teve como objetivo realizar um estudo comparativo dos diferentes mecanismos de autenticação e autorização em sistemas web, visando fornecer uma análise aprofundada que auxilie na escolha adequada desses mecanismos em projetos. O estudo buscou oferecer uma compreensão ampla das características, pontos fortes e limitações de cada mecanismo, a fim de permitir a seleção correta e a implementação eficiente das medidas de segurança necessárias.

Ao longo do trabalho, foram apresentados mecanismos de autenticação e autorização, dentre eles a autenticação básica HTTP, autenticação *digest*, autenticações baseadas em sessão e em *tokens*, OAuth, OAuth 2.0 e OpenID Connect. Foram também apresentadas algumas características relacionadas a usabilidade, nível de segurança, vulnerabilidades, implementação e obsolescência destes mecanismos. Ao fim, apresentou-se uma tabela comparativa entre os métodos, para sumarizar as características de cada um, facilitando a análise dos dados.

A segurança dos sistemas web é um aspecto crucial que não pode ser negligenciado. A escolha e implementação correta dos mecanismos de autenticação e autorização são fundamentais para garantir a proteção dos dados e recursos dos usuários. Por meio deste estudo comparativo, espera-se que os desenvolvedores e profissionais da área possam tomar decisões mais assertivas ao selecionar os mecanismos de acordo com as necessidades de cada projeto, contribuindo assim para a construção de sistemas web mais seguros e confiáveis.

Referências

Balaj, Y. (2017). Token-based vs session-based authentication: A survey.

Barth, A. (2011). HTTP State Management Mechanism. RFC 6265.

Biehl, M. (2019). *OpenID Connect: End-user Identity for Apps and APIs: 6*. Api-University. Createspace Independent Publishing Platform, North Charleston, SC.

Calzavara, S., Focardi, R., Squarcina, M., e Tempesta, M. (2018). Surviving the web: A journey into web session security. *WWW '18: Companion Proceedings of the The Web Conference 2018*, páginas 451–455.

Chapman, N. e Chapman, J. (2012). *Authentication and Authorization on the Web*. MacAvon Media, Escócia, Reino Unido.

Drhová, K. (2018). Authentication, authorization, and session management in the http protocol. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology.

Fett, D., Küsters, R., e Schmitz, G. (2016). A comprehensive formal security analysis of OAuth 2.0. Em *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

Fett, D., Kusters, R., e Schmitz, G. (2017). The web SSO standard OpenID connect: In-depth formal security analysis and security guidelines. Em *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE.

Franks, P. J., Hallam-Baker, P., Stewart, L. C., Hostetler, J. L., Lawrence, S., Leach, P. J., e Luotonen, A. (1999). HTTP Authentication: Basic and Digest Access Authentication. RFC 2617.

Gourley, D. e Totty, B. (2002). *HTTP: The Definitive Guide*. O'Reilly Media, Califórnia, Estados Unidos.

Greengard, S. (2015). *The Internet of Things*. MIT Press, Massachusetts, Estados Unidos.

Hallam-Baker, P., Franks, P. J., Stewart, L. C., Sink, E. W., Hostetler, J. L., Leach, P. J., e Luotonen, A. (1997). An Extension to HTTP : Digest Access Authentication. RFC 2069.

Hammer-Lahav, E. (2010). The OAuth 1.0 Protocol. RFC 5849.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749.

Jones, M. B., Bradley, J., e Sakimura, N. (2015a). JSON Web Signature (JWS). RFC 7515.

Jones, M. B., Bradley, J., e Sakimura, N. (2015b). JSON Web Token (JWT). RFC 7519.

Lodderstedt, T., McGloin, M., e Hunt, P. (2013). OAuth 2.0 Threat Model and Security Considerations. RFC 6819.

Montanheiro, L., Carvalho, A., e Rodrigues, J. (2017). Utilização de json web token na autenticação de usuários em apis rest. Em *XIII Encontro Anual de Computação - EnAComp2017 - UFG*, páginas 186–193.

Montulli, L. e Kristol, D. M. (1997). HTTP State Management Mechanism. RFC 2109.

Nielsen, H., Fielding, R. T., e Berners-Lee, T. (1996). Hypertext Transfer Protocol – HTTP/1.0. RFC 1945.

Noureddine, M. e Bashroush, R. (2011). A provisioning model towards oauth 2.0 performance optimization. *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)*, páginas 76–80.

OWASP (2021). Owasp top 10:2021. `https://owasp.org/Top10/`. Acesso em: 26 abr. 2023.

Papathanasaki, M., Maglaras, L., e Ayres, N. (2022). Modern authentication methods: A comprehensive survey. *AI, ComputerScience and Robotics Technology*, páginas 1–24.

Peyrott, S. (2018). *The JWT Handbook*. Auth0.

Reschke, J. (2015). The 'Basic' HTTP Authentication Scheme. RFC 7617.

Sakimura, N., Bradley, J., Jonesm, M., de Medeiros, B., e Mortimore, C. (2014). Openid connect core 1.0. OpenID Foundation.

Shekh-Yusef, R., Ahrens, D., e Bremer, S. (2015). HTTP Digest Access Authentication. RFC 7616.

Siriwardena, P. (2014). *Advanced API Security: Securing APIs with Oauth 2.0, Openid Connect, Jws, and Jwe*. Apress, California, Estados Unidos.

Spilca, L. (2020). *Spring Security in Action*. Manning Publications, Nova Iorque, Estados Unidos.

Sullivan, B. e Liu, V. (2011). *Web Application Security, A Beginner's Guide*. McGraw-Hill, Estados Unidos.

Tumin, S. e Encheva, S. (2012). A closer look at authentication and authorization mechanisms for web-based applications. Em *Recent Researches in Applied Information Science*, páginas 100–105, Faro, Portugal.