FIAP ON

WEBSERVICES &
RESTFUL TECHNOLOGIES

SEGURANÇA EM WS

EDUARDO GALEGO

5

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

5 SEGURANÇA EM WEB SERVICES

O assunto Segurança Tecnológica está muito aquecido no mercado nos últimos anos, haja vista a presença, nos noticiários, de diversas empresas que sofreram ataques cibernéticos que ora inoperaram os sistemas, ora sequestraram seus dados. Neste capítulo, vamos abordar o tema Segurança no contexto de Web Services.

Merece atenção um aspecto técnico já conhecido: por ter sido todo baseado em tecnologias e protocolos utilizados pela Web, grande parte do que aplicamos em Segurança para Web também se aplica para Web Services.

Um grande exemplo disto é o protocolo TLS (acrônimo de *Transport Layer Security)* e HTTPS (acrônimo de *Hyper Text Transfer Protocol Secure*). Em aplicações Web, utilizamos este protocolo para garantir ao usuário um canal seguro de transmissão de dados, uma vez que o conteúdo dos pacotes HTTP é criptografado antes do envio e descriptografado no destino.

O mesmo se aplica aos Web Services: utilizamos TLS e HTTPS para garantir que, caso um agente suspeito consiga interceptar os pacotes, ele não tenha acesso aos dados enviados no *payload*. Se você tem dúvidas sobre as diferenças entre SSL, TLS e HTTPS, não deixe de conferir a página (https://howhttps.works/https-ssl-tls-differences/).

A organização OWASP elaborou uma pesquisa e apresentou um relatório contendo as dez maiores brechas de seguranças em APIs. Seguem alguns dos exemplos obtidos na lista:

- **Broken Object Level Authorization:** um e-Commerce expõe o seguinte método da API: `GET /api/shops/{shopId}/orders`, na qual o parâmetro `shopId` é o identificador da loja.
  Cada parceiro varejista possui um conjunto de lojas que opera, não permitindo ver os pedidos de lojas que não o pertencem. Mas, por se tratar de um *Path Param*, qualquer consumidor pode informar qualquer valor no parâmetro `shopId`.
  Se o *software* que implementa o serviço não realizar o devido tratamento, poderá expor dados de pedidos para outros varejistas. Um agente suspeito que descobrir esta vulnerabilidade pode criar um robô consumidor que tenta

diversos valores para o parâmetro e obtém acesso a listas de pedidos de diversas lojas.

- **Excessive Data Exposure:** uma API de uma rede social possui o seguinte método: `GET /api/articles/{articleId}/comments/{commentId}`, que retorna dados dos comentários feitos pelos usuários em um determinado artigo. Mas, por se tratar de um recurso aninhado, dados do usuário também são retornados junto ao comentário.
  Se um agente suspeito tiver acesso aos dados de comentários, pode capturar junto os dados dos usuários.
  Neste sentido, a implementação da API deve filtrar os dados antes do envio, evitando expor dados de forma desnecessária.

- **Lack of Resources & Rate Limiting**: um determinado método da API retorna uma lista com todos as instâncias de um determinado recurso. Caso seja uma lista extensa, isto pode custar processamento e tráfego aos servidores, abrindo uma brecha para alguém que esteja explorando uma vulnerabilidade capaz de inutilizar um serviço (conhecido como DDoS - *Distributed Denial of Service* ou Negação de Serviço Distribuída). Mesmo aplicando paginação (quando o consumidor informa a quantidade máxima de registros que gostaria de receber), se não houver um limite superior na lógica da implementação, um agente suspeito pode informar um valor exorbitante, contribuindo para a degradação do serviço.

- **Mass Assignment**: uma aplicação que lida com saldo expõe o método `GET /api/v1/users/me` e obtém como resultado o seguinte JSON: `{"user_name":"inons","age":24,"credit_balance":10}`.
  O método PUT está disponível para o mesmo recurso, mas a documentação descreve o envio apenas para os parâmetros `user_name` e `age`, deixando de fora o `credit_balance`, uma vez que se trata de um campo calculado. Entretanto, a implementação do serviço utiliza os mesmos dados enviados pelo consumidor para persistir no banco de dados. Desta forma, ao informar o parâmetro `credit_balance` no método PUT, ele é persistido de maneira literal, não por meio de cálculo.

Para ter acesso ao relatório completo, acesse a página oficial do grupo, disponível na seção ‘Referências’.

5.1 Autenticação e Autorização

Na base de toda segurança em WS estão os processos de Autenticação e Autorização.

Também conhecida como **AuthN**, a **Autenticação** pretende identificar quem é o consumidor para um conjunto de chamadas. Para tal, é necessário que previamente o desenvolvedor tenha cadastrado a aplicação e obtido como resposta a credencial de acesso. Tais credenciais são utilizadas como uma “prova” de que a aplicação é realmente quem diz ser.

Já a **Autorização**, também conhecida como **AuthZ**, vem logo após aAutenticação. Ela pretende responder o seguinte: “Já sei que você é quem diz que é, mas será que tem permissão para acessar este recurso?”. Portanto, ela está relacionada a questões de alçada e liberação/negação de recursos.

Durante muito tempo, a principal forma de Autenticação em Web Services era realizada utilizando **Basic Authentication**. Neste modelo, o consumidor envia no *header* da requisição os seguintes dados:

Key: `Authorization`

Value: "Basic " + base64(`<login>` + ":" + `<password>`)

sendo `<login>` e `<password>` as credenciais da aplicação e base64() uma função que encoda os dados utilizando o algoritmo Base 64.

Ao receber a requisição, o serviço obtém o valor do *header Authorization* e extrai os dados de login e senha. Isto ocorre porque o algoritmo Base 64 pode ser decodado, ou seja, obtém de volta os dados originais (não se trata de uma criptografia).

Em posse da credencial, é possível conferir se os dados estão armazenados em uma base de aplicações. Se estiverem, a chamada é processada e os recursos

solicitados são devolvidos. Caso contrário, o erro **401 Unauthorized** é retornado ao consumidor.

A grande desvantagem neste modelo é que a credencial é utilizada em toda e qualquer chamada. Se um agente suspeito tiver acesso ao *header*, poderá obter a credencial e fazer chamadas se passando pela aplicação. Normalmente, dados de credenciais não possuem tempo de expiração; assim, o agente pode utilizá-los por muito tempo até alguém identificar o ataque e inutilizar a credencial.

Para manter as aplicações seguras, surgiu um novo modelo que vem sendo aplicado em quase todos os Web Services, o qual abordaremos na próxima seção.

5.2 OAuth2

Trata-se da segunda versão da especificação oAuth (sendo esta primeira depreciada) e é definida pelo IETF OAuth Working Group. A página oficial da especificação pode ser encontrada na seção ‘Referências’.

A grande novidade neste modelo é a inclusão de um novo agente, conhecido como ***Authorization Server***. Ele é o responsável por manter os dados de aplicações e credenciais, emitir autorizações de acesso e validá-las quando solicitado.

Antes de explicar os conceitos técnicos, vamos a uma pequena história para clarificar o entendimento.

Imagine que você esteja muito ansioso para assistir um filme, um show ou uma partida de futebol. Então, você se desloca até o local do espetáculo e tenta entrar diretamente na arquibancada. Provavelmente, se deparará com algo semelhante à Figura “Imagem de uma catraca”:

Figura 5.1 – Imagem de uma catraca
Fonte: CEO Sistemas (2022)

O equipamento exigirá um *ticket* ou ingresso. Caso você não tenha este documento, a catraca não será liberada e você não terá acesso ao espetáculo.

Para a devida liberação, você precisa se dirigir até uma bilheteria e adquirir o *ticket*. É provável que exigirão a apresentação de algum documento. Estando tudo certo, você receberá o ingresso e poderá apresentá-lo na entrada.

Ao apresentá-lo, a catraca validará as informações contidas nele, como: validade do *ticket*, emissor, código de segurança etc. É importante citar que a base de dados desta validação provém da bilheteria. Após todas as validações, você será liberado para adentrar o local e desfrutar do espetáculo.

Observe nesta pequena história a ação de três agentes:

- **A pessoa que quer assistir ao espetáculo:** em se tratando de oAuth2, este agente é o consumidor da API. Em outras palavras, a aplicação consumidora interessada no resultado do serviço.
- **O espetáculo:** trata-se do agente que fornece o recurso que o consumidor espera obter. É o Web Service em si e, em se tratando de oAuth2, passamos a identificá-lo como **Resource Server**.

- **A bilheteria:** é o responsável pela emissão e validação dos *tickets*. Como comentado há pouco, em oAuth2 este agente é conhecido como **Authorization Server**. O *ticket* recebe o nome de **Access Token**.

As credenciais para identificar uma aplicação são conhecidas como **client_id** e **client_secret**. Eles fazem o mesmo papel que *login* e *password* no exemplo com Basic Authentication.

É importante citar que oAuth2 estipula dois escopos de credenciais: **Aplicação** (sendo identificado por *client_id* e *client_secret*) e **Usuário** (sendo identificado por *username* e *password*).

Atenção para não confundir client_id com username e client_secret com password!

Um exemplo para ilustrar: imagine que você crie uma aplicação *mobile* que consumirá um determinado serviço Web. Logo na primeira tela, sua aplicação exibe os campos “Nome do Usuário” e “Senha”, no qual o usuário deverá informar suas credenciais para acessar a área logada.

Quando a aplicação fizer as chamadas, é necessário informar tanto client_id e client_secret (que identificam a aplicação) quanto *username* e *password* (que identificam o usuário).

O mesmo client_id será utilizado para todos que fizerem a instalação da sua aplicação a partir de uma loja (ex. Play Store). Já o username será único por usuário.

OAuth2 estipula ainda quatro tipos de fluxos, conhecidos como **Grant Type**. Observe a Tabela “Tabela comparativa entre Grant Type”.

Tabela 5.1 – Tabela comparativa entre Grant Type

Grant Type	Descrição	Complexo de desenvolver?
Client Credentials	Interações entre sistemas de negócio (não possui contexto de usuário)	Não
Password	Recursos são obtidos por um usuário e a aplicação consumidora é confiável	Um pouco
Authorization Code	Recursos são obtidos por um usuário e a aplicação consumidora NÃO É confiável	Muito
Implicit	Recursos são obtidos por um usuário e a aplicação é baseada em um navegador (Javascript, por exemplo) e NÃO É confiável	Muito, porém é menos seguro que Authorization Code

Fonte: Elaborado pelo autor (2022)

O fluxo mais simples é o Client Credentials, uma vez que ele aceita apenas *tokens* com contexto de aplicação. A Figura “Fluxo Client Credentials” apresenta as interações entre os agentes.

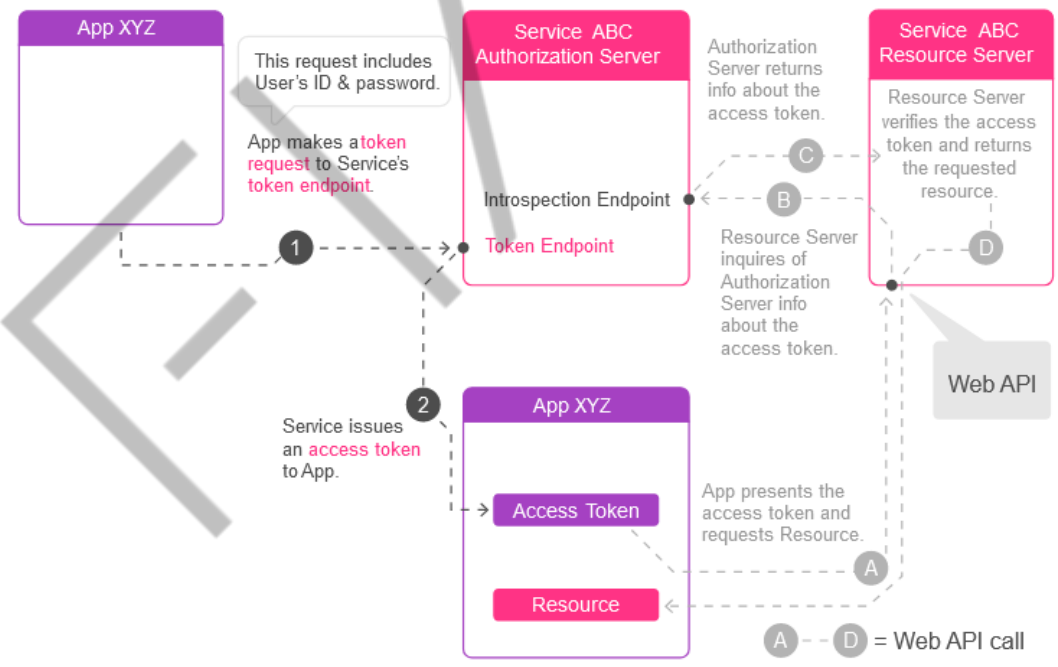

Figura 5.2 – Fluxo Client Credentials
Fonte: Kawasaki (2017)

Uma versão animada desta imagem está disponível no endereço: (https://www.youtube.com/watch?v=XJuFcXI2v2k).

Repare que, na Figura, os retângulos laranja representam a mesma aplicação consumidora, mas em dois momentos distintos: na primeira sem o *access token* e na segunda com. O retângulo verde ao centro representa o Authorization Server e o retângulo verde à direta o Resource Server.

O ponto verde apontado pelo quadro de diálogo vermelho “Web API” é semelhante à catraca descrita em nossa história: é aqui que a validação do *access token* ocorre. Se o resultado obtido em (C) for sucesso, o Resource Server atende à solicitação do consumidor. Caso contrário, um erro **401 Unauthorized** é lançado.

Agora, observe a Figura “Fluxo Password”.

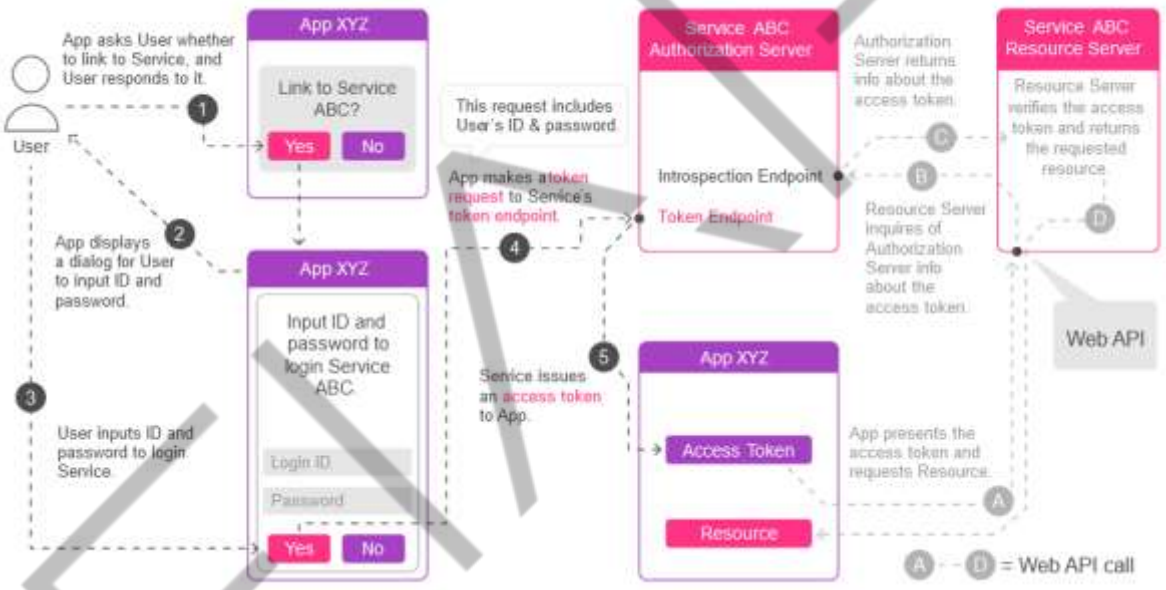

Figura 5.3 – Fluxo Password
Fonte: Kawasaki (2017)

Uma versão animada desta imagem está disponível no endereço (https://www.youtube.com/watch?v=CGMiOHrOAYQ).

A grande novidade deste fluxo é a inclusão do contexto do usuário. Ele passa a informar suas credenciais (Login ID e Password), os quais são validados pelo Authorization Server. Embora não esteja informado na Figura, alguns Authorization Server utilizam-se de aplicações terceiras para validação do usuário, como servidores de LDAP ou AD.

Uma vez validada as credenciais, o fluxo segue semelhante ao fluxo Client Credentials.

A principal desvantagem deste fluxo é que o usuário informa suas credenciais dentro de uma página exibida pela Aplicação Consumidora. Devemos confiar que a aplicação enviará os dados de credencial diretamente para o Authorization Server. Assim, um desenvolvedor com más intenções pode se aproveitar desta informação e criar sua base de credenciais válidas.

Em uma nova atualização do oAuth2 (conhecido como oAuth2.1), este fluxo está sendo depreciado devido à brecha de segurança discutida anteriormente.

Observe a Figura “Fluxo Authorization Code”.

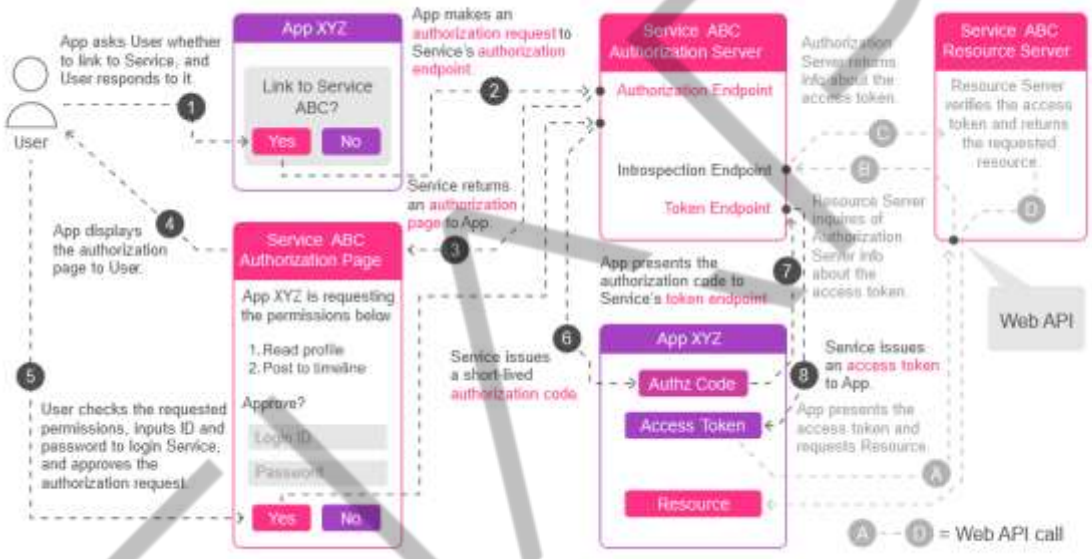

Figura 5.4 – Fluxo Authorization Code
Fonte: Kawasaki (2017)

Uma versão animada desta imagem está disponível no endereço (https://www.youtube.com/watch?v=7D-OU4hZW70).

Diferentemente do fluxo Password, desta vez o Authorization Server fornece uma página para o usuário informar as credenciais de forma segura, sem que a aplicação tenha conhecimento destes dados. O papel da aplicação é, portanto, redirecionar o usuário para a página do Authorization Server, que fará um *redirect* de volta para aplicação ao final do processo de Login.

Este processo de *login* é semelhante a uma Aplicação Web que permite logar utilizando as credenciais de uma rede social. Por exemplo: a aplicação exibe o botão “Logar com o Facebook”. Ao clicar, você é redirecionado para uma tela do Facebook

para informar seu usuário e senha. A seguir, uma página do Facebook exibe a seguinte mensagem: “A aplicação XPTO deseja acessar seus seguintes dados”, com um botão para permissão. Ao clicar, você é redirecionado de volta para a aplicação, desta vez logado.

Note também em (6) que, logo após o *login*, a primeira resposta do Authorization Server à Aplicação é um **Authz Code**. Trata-se de um *token* intermediário, utilizado apenas para autorizar a aplicação a obter um *access token*.

É papel, portanto, da aplicação efetuar uma chamada de volta ao Authorization Server, solicitando um *access token* a partir deste *authorization code*. Após isto, todo o fluxo segue como visto nos fluxos anteriores.

Para o oAuth2.1, este fluxo foi acrescido de um novo quesito de segurança denominado de PKCE (acrônimo de Proof Key for Code Exchange). Para saber mais, acesse (https://oauth.net/2/pkce/).

Observe a Figura “Fluxo Implicit”.

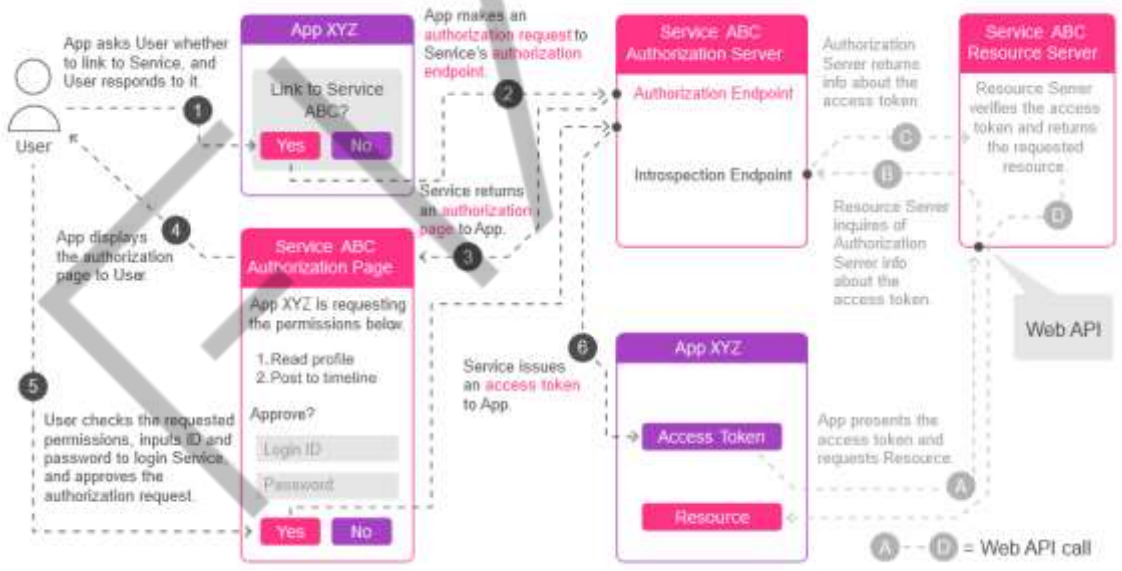

Figura 5.5 – Fluxo Implicit
Fonte: Kawasaki (2017)

Uma versão animada desta imagem está disponível no endereço (https://www.youtube.com/watch?v=KUruB156-h4).

A única diferença entre este fluxo e o anterior é a resposta do Authorization Server à aplicação logo após a etapa de *login* (em (6)): repare que o Access Token é retornado diretamente, sem o uso de um *token* intermediário.

Isto só é possível quando a aplicação consumidora é também responsável pelas chamadas à API, sem utilizar um servidor de *back-end* ou BFF (*back-end for front-end*). Assim, somente aplicações que rodam lógica no navegador (como *frameworks* que utilizam um motor Javascript) é que podem se utilizar deste fluxo.

Esta restrição limita muito sua utilização, sendo isto também depreciado na versão oAuth2.1.

Existe ainda um outro fluxo, denominado **Refresh Token**. Ele é diferente dos demais, visto que tem como objetivo renovar um *access token* assim que ele expirar. Um vídeo com a animação para este fluxo pode ser encontrado aqui (https://www.youtube.com/watch?v=jcKDsQfBgYY).

Os access token podem ser de dois tipos:

- **Opacos:** quando o valor não representa nenhum dado, apenas um conjunto aleatório de letras e números gerado pelo Authorization Server, de modo que apenas ele pode informar se um *access token* é válido e seus dados obtidos na chamada do método **/introspect**.
- **Não-opacos:** quando o valor pode ser interpretado e representa um conjunto de dados. Normalmente estes dados estão no formato JWT (acrônimo de *JSON Web Tokens*).

  A vantagem deste modelo é a validação do *token* sem a necessidade de invocar o método **/introspect** do Authorization Server, diminuindo assim o tempo de resposta das chamadas.

  Por ser assinado digitalmente, o JWT pode ser validado utilizando uma chave pública. Assim temos a comprovação que o *token* não foi violado durante o transporte.

  Para saber mais sobre o JWT, acesse o artigo *Introduction to JSON Web Tokens*, presente na seção ‘Referências’.

Para a etapa de Autorização, o oAuth2 possui dois atributos:

- **Scopes:** exibe uma lista de permissões para a aplicação;
- **Roles:** exibe uma lista de permissões para o usuário.

O *scope* pode ser informado no momento da criação do *access token*. Se o Authorization Server não tiver atribuído este valor à aplicação, a chamada devolve um erro.

Se o valor não for informado no momento de geração do *token*, o Authorization Server devolve, na resposta, o atributo com todos os *scopes* atribuídos à aplicação.

Depois de gerado, os dados de *scopes* e *roles* podem ser obtidos por meio da chamada **/introspect** do Authorization Server. Eles também podem ser obtidos dentro do JWT, como uma ***claim*** (quando se tratar de um *token* não opaco).

Aliás, ***claims*** são todos os atributos contidos dentro de um JWT. Eles podem ser do tipo *Custom*, *Public* ou *Private*. Para saber mais sobre claims, acesse o artigo *JSON Web Token Claims*, presente na seção 'Referências'.

Também é possível que os valores contidos nos *claims* sejam utilizados pelo Resource Server. Por exemplo: um método da API só é permitido se o *token* tiver um determinado valor dentro de *scopes*. Caso contrário, o erro **403 Forbidden** é retornado.

Para concluir esta seção, não deixe de conferir no artigo *What's new in OAuth 2.1?*, presente na seção 'Referências', as novidades que a nova versão do oAuth2.1 traz.

5.3 Ferramentas

Estão disponíveis diversas ferramentas no mercado (inclusive *open source*) que fazem o papel de Authorization Server.

Um estudo do Gartner apresenta os líderes de Mercado no segmento, conforme evidencia a Figura "Quadrante Mágico do Gartner sobre Access Management (2021)".

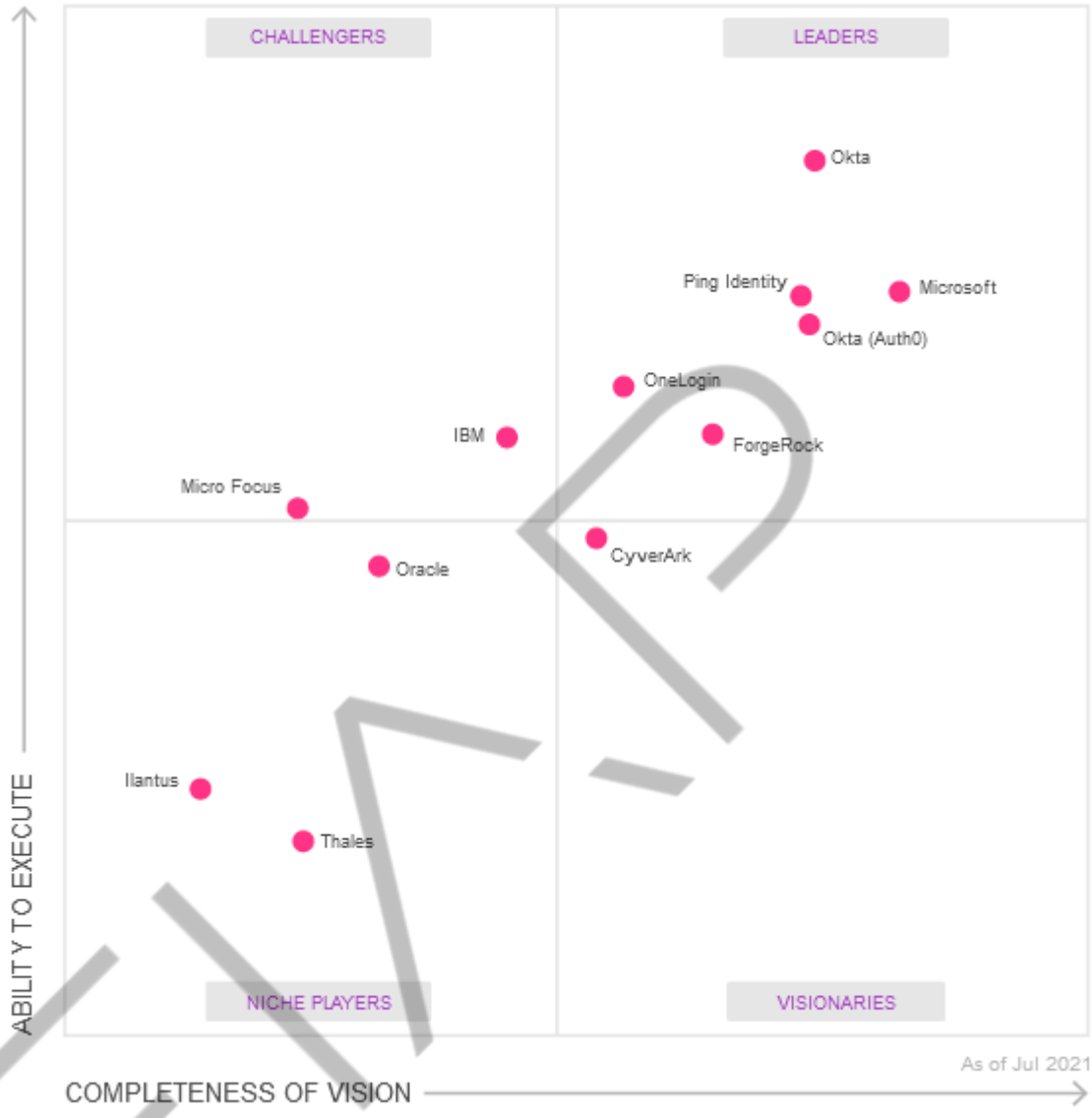

Figura 5.6 – Quadrante Mágico do Gartner sobre Access Management (2021)
Fonte: Blake (2021)

Ressalta-se que na seção ‘Referências’ é apresentada uma relação com 10 ferramentas *open source* para gestão de identidade.

É possível, ainda, desenvolver sua própria ferramenta de Autenticação e Autorização. Para saber mais, acesse o artigo “*Spring Security OAuth Authorization Server”*, também compartilhado na seção ‘Referências’ deste capítulo.

CONCLUSÃO

Neste capítulo abordamos o assunto da segurança em Web Services.

Exploramos inicialmente algumas das principais brechas de segurança em WS, resultado de um estudo da OWASP.

Logo após, discutimos os conceitos de Autenticação e Autorização e conhecemos uma maneira simples de implementar a Autenticação utilizando *Basic Authentication*.

Em seguida, discorremos com mais detalhes sobre o oAuth2: o papel do Authorization Server, os detalhes de cada fluxo, a diferença entre *token* opaco e não-opaco, os escopos de aplicação e usuário e como o oAuth2 lida com autorização.

Por fim, conhecemos algumas ferramentas para Gestão de Identidade, que faz o papel do Authorization Server.

REFERÊNCIAS

AUTH0 DOCS. **JSON Web Token Claims**. 2022. Disponível em: <https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-claims>. Acesso em: 04 ago. 2022.

AUTHLETE. **OAuth 2.0 Client Credentials Grant Flow**. 2017a. Disponível em: <https://www.youtube.com/watch?v=XJuFcXI2v2k>. Acesso em: 04 ago. 2022.

AUTHLETE. **OAuth 2.0 Código de autorização Fluxo de concessão**. 2017b. Disponível em: <https://www.youtube.com/watch?v=7D-OU4hZW70>. Acesso em: 04 ago. 2022.

AUTHLETE. **OAuth 2.0 Implicit Grant Flow**. 2017c. Disponível em: <https://www.youtube.com/watch?v=KUruB156-h4>. Acesso em: 04 ago. 2022.

AUTHLETE. **OAuth 2.0 Refresh Token Grant Flow**. 2017d. Disponível em: <https://www.youtube.com/watch?v=jcKDsQfBgYY>. Acesso em: 04 ago. 2022.

AUTHLETE. **OAuth 2.0 Resource Owner Password Credentials Grant Flow**. 2017e. Disponível em: <https://www.youtube.com/watch?v=CGMiOHrOAYQ>. Acesso em: 04 ago. 2022.

BAELDUNG. **Spring Security OAuth**: Authorization Server. 2022. Disponível em: <https://www.baeldung.com/spring-security-oauth-auth-server>. Acesso em: 04 ago. 2022.

BLAKE, N. **2021 Gartner Magic Quadrant for Access Management**. 2021. Disponível em: <https://www.pingidentity.com/en/resources/blog/post/gartner-iam-magic-quadrant.html>. Acesso em: 04 ago. 2022.

CEO SISTEMAS. **Controle de Acesso**. 2022. Disponível em: <https://ceosistema.com.br/controle-de-acesso/>. Acesso em: 04 ago. 2022.

HOW HTTPS. WORKS. **The differences between HTTPS, SSL, and TLS**. 2022. Disponível em: <https://howhttps.works/https-ssl-tls-differences/>. Acesso em: 03 ago. 2022.

JWT. **Introduction to JSON Web Tokens**. 2022. Disponível em: <https://jwt.io/introduction>. Acesso em: 04 ago. 2022.

KAWASAKI, T. **Diagrams And Movies Of All The OAuth 2.0 Flows**. 2017. Disponível em: <https://darutk.medium.com/diagrams-and-movies-of-all-the-oauth-2-0-flows-194f3c3ade85>. Acesso em: 04 ago. 2022.

MOORE, D. **What's new in OAuth 2.1?**. 2020. Disponível em: <https://fusionauth.io/blog/2020/04/15/whats-new-in-oauth-2-1>. Acesso em: 04 ago. 2022.

MOUSA, H. **10 Open-source Identity and Access Management IAM Systems for the Enterprise**. 2020. Disponível em: <https://medevel.com/iam-systems-10-identity/>. Acesso em: 04 ago. 2022.

OAUTH. **OAuth Working Group Specifications**. 2022. Disponível em: <https://oauth.net/specs/>. Acesso em: 04 ago. 2022.

OAUTH. **RFC 7636**: Proof Key for Code Exchange. 2018. Disponível em: <https://oauth.net/2/pkce/&as_qdr=y15>. Acesso em: 04 ago. 2022.

OWASP. **OWASP API Security Project**. 2022. Disponível em: <https://owasp.org/www-project-api-security/>. Acesso em: 04 ago. 2022.

GLOSSÁRIO

JWT	JSON Web Tokens
TLS	Transport Layer Security
HTTPS	Hyper Text Transfer Protocol Secure
SSL	Secure Sockets Layer
PKCE	Proof Key for Code Exchange
DDoS	Distributed Denial of Service
LDAP	Lightweight Directory Access Protocol
AD	Active Directory
BFF	Back-end For Front-end