Sistemas Web Seguros

UNIDADE 05

Gestão de identidade e acesso

Nesta unidade, conheceremos: os fundamentos da gestão de identidade e acesso, um campo de estudo da segurança da informação; a gestão de perfis de negócio por meio do padrão *Role Based Access Control* (RBAC); o padrão de autorização de usuários com JSON *Web Token* (JWT); a biblioteca Java JSONWebToken, que nos auxiliará na criação e validação de JWT; e a construção de um sistema de gestão de identidade e acesso RBAC com JWT.

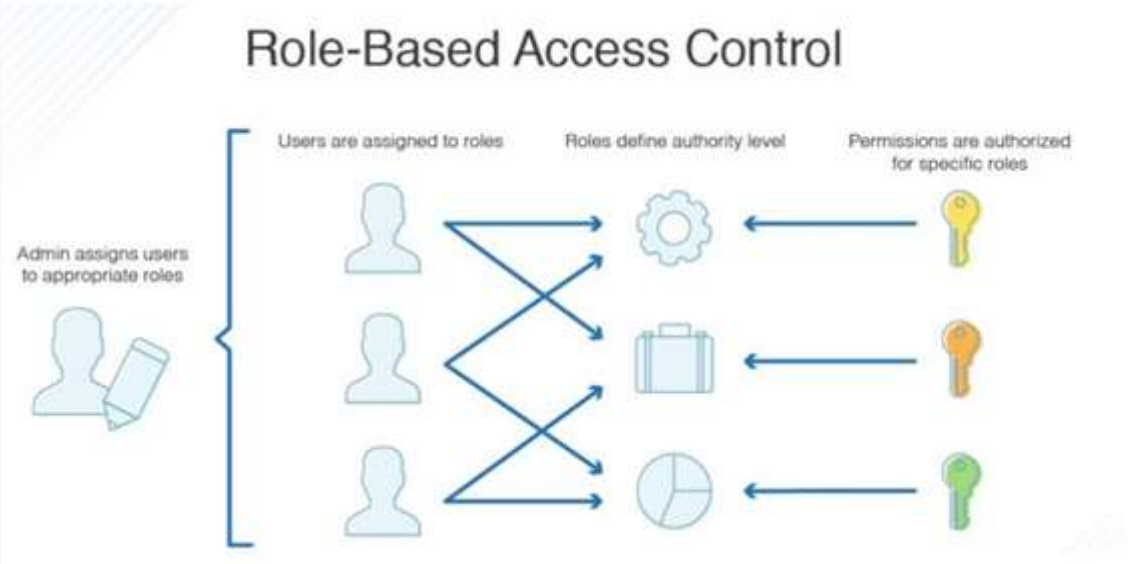
GESTÃO DE IDENTIDADE E ACESSO

Um serviço de gestão de identidade e acesso tem por missão gerenciar com segurança o acesso aos serviços e recursos de uma solução de TI. O gerenciamento de acesso a recursos é uma função crítica para qualquer organização, sendo os sistemas de controle de acesso essenciais para operacionalizar a gestão de quem deve ter acesso aos recursos de determinada solução de TI, o que eles podem fazer com esses recursos e qual é o escopo desse acesso. Nesta unidade, apresentaremos um dos modelos mais utilizados em sistemas corporativos, o padrão RBAC.

Controle de acesso RBAC

RBAC significa controle de acesso baseado em função (do inglês *Role Based Access Control*) e descreve uma metodologia para restringir o acesso a recurso por meio da atribuição de funções e autorizações na infraestrutura de TI de uma organização (FERRAIOLO; CUGINI; KUHN, 1995). Os direitos de acesso não são atribuídos de acordo com usuários, mas com base em um modelo de função definido, como a estrutura organizacional, funções, cargos, localização, entre outros.

Figura 1. RBAC



Fonte: Microsoft Azure (2020).

As autorizações do RBAC especificam quais direitos há sobre as informações, ou seja, leitura, gravação ou controle total. Também definem direitos de acesso a aplicativos da organização, além de autorizações específicas desses aplicativos, tudo vinculado à respectiva função do usuário. Cada usuário recebe uma ou mais funções, dependendo de sua posição na empresa. Com base nessa atribuição de

função, ele recebe os direitos de acesso apropriados a dados e aplicativos na infraestrutura de TI, o que idealmente habilita todas as suas atividades, sem ajustes adicionais.

Benefícios do RBAC

O RBAC é geralmente considerado a melhor prática para gerenciamento de autorização, quando suas funções são definidas em toda a empresa. O conceito de função e autorização pode ser implementado com a ajuda de um sistema de gerenciamento de identidade e acesso. Seguem alguns benefícios, quando bem implantado:

- **Flexibilidade**: em contraste com a alocação rígida de autorizações individuais, que envolve um alto esforço administrativo e uma alta suscetibilidade a erros, a alocação de direitos com base em funções é muito mais flexível e menos complexa de gerenciar.
- **Eficiência:** a atribuição correta de direitos desde o início significa que os funcionários não precisarão esperar o suporte de TI, pois os acessos são aplicados automaticamente. Isso aumenta significativamente a eficiência do suporte de TI e dos funcionários, uma vez que raramente haverá solicitações de direitos adicionais.
- **Segurança:** “superautorizações” e acumulação de direitos para usuários individuais são efetivamente evitadas pela definição exclusiva de direitos de acesso usando o conceito de função.

Para construir um sistema de identidade e acesso utilizando o padrão RBAC, é necessário conhecer algumas estratégias tecnológicas. Por isso, vamos apresentar na próxima seção um mecanismo muito utilizado na internet para trafegar credenciais de acesso de forma segura.

JSON *Web Token* [/titulo]

Quando você utiliza um aplicativo de home banking para operações bancárias, há previamente uma identificação e autorização para realizar as funções do sistema, isto é, você possui uma credencial que permite o acesso a recursos da sua conta bancária. Suas credenciais certamente são protegidas, mas como isso pode ser feito?

O JWT é um padrão que estabelece uma forma para transmitir informações entre duas partes de modo seguro. Pelo próprio nome, veremos que as informações são encasuladas por meio de objetos JavaScript *Object Notation* (JSON).

As informações ganham uma camada de segurança adicional por meio de assinaturas digitais. Os JWTs podem ser assinados por par de chaves pública/privada, usando algoritmos, como Rivest-Shamir-Adleman (RSA) ou *Elliptic Curve Digital Signature Algorithm* (ECDSA).

Quando utilizar JWT

- **Autorização:** logo após um usuário ser autenticado, ele deve receber suas permissões de acesso, que serão utilizadas em todos os sistemas que aceitarem esse token. Note que o token pode ser válido em diversos sistemas, portanto podemos construir um login único, também conhecido pelo termo *single sign on*.
- **Troca de informações**: os tokens são uma boa maneira de transmitir informações entre as partes com segurança. Como são dados assinados, você pode garantir a autoria do remetente e garantir a integridade, ou seja, se o conteúdo não foi adulterado.

Estrutura do JWT

De modo geral, o JWT possui três partes:

- *header* (cabeçalho);
- *payload* (carga útil);
- *signature* (assinatura).

Em sua forma compactada, ele separa as três partes com o caractere “ponto”. Veja um exemplo de JWT na versão em Base64. Cada parte possui uma cor diferente.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey
J0YWdfcHJvamV0byI6IkF1ZGluMjM2NSIsInRhZ
19mdW5jYW8iOiJhdWRpdG9yIiwibWF0cmljdWxh
IjoiMjE1NTk4Iiwibm9tZSI6IkFuYSBLYXJsYSI
sInN1YiI6IlNJU1JIIiwiaWF0IjoxNjM0MDQxNz
kzLCJleHAiOjE2MzQwNDI2OTN9.B3zdDly8EIW8
aKamXAJXSKAKLtZ28ZQv8KRw7eV4jBc
```


***Header* JWT**

O cabeçalho consiste em duas partes: o tipo de *token*, que é JWT, e o algoritmo de assinatura utilizado, como HMAC SHA256 ou RSA.

Exemplo:

***Payload* JWT**

A segunda parte do *token* contém as declarações, ou seja, informações sobre a entidade, que geralmente é um usuário. Veja o exemplo com a continuação da leitura do *token.*

Temos um conjunto de declarações predefinidas que não são obrigatórias, mas recomendadas. Algumas delas são: **iss** (emissor), **iat** (tempo de criação), **exp** (tempo de expiração), **sub** (assunto), **aud** (público) e outras.

É importante destacar que *tokens* assinados, embora protegidos contra adulteração, podem ser lidos por qualquer pessoa. Por isso, não coloque informações secretas no *payload* ou nos cabeçalhos de um JWT, a menos que esteja criptografado.

Assinatura *JWT*

Para criar a assinatura, precisamos das seguintes partes: os cabeçalhos, o *payload*, um segredo e o algoritmo especificado no cabeçalho.

Por exemplo, se você deseja usar o algoritmo HMAC SHA256, a assinatura será criada da seguinte maneira:

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), segredo)

A assinatura é usada para verificar se a mensagem não foi alterada ao longo do caminho e, no caso de *tokens* assinados com uma chave privada, se o remetente do JWT é quem diz ser.

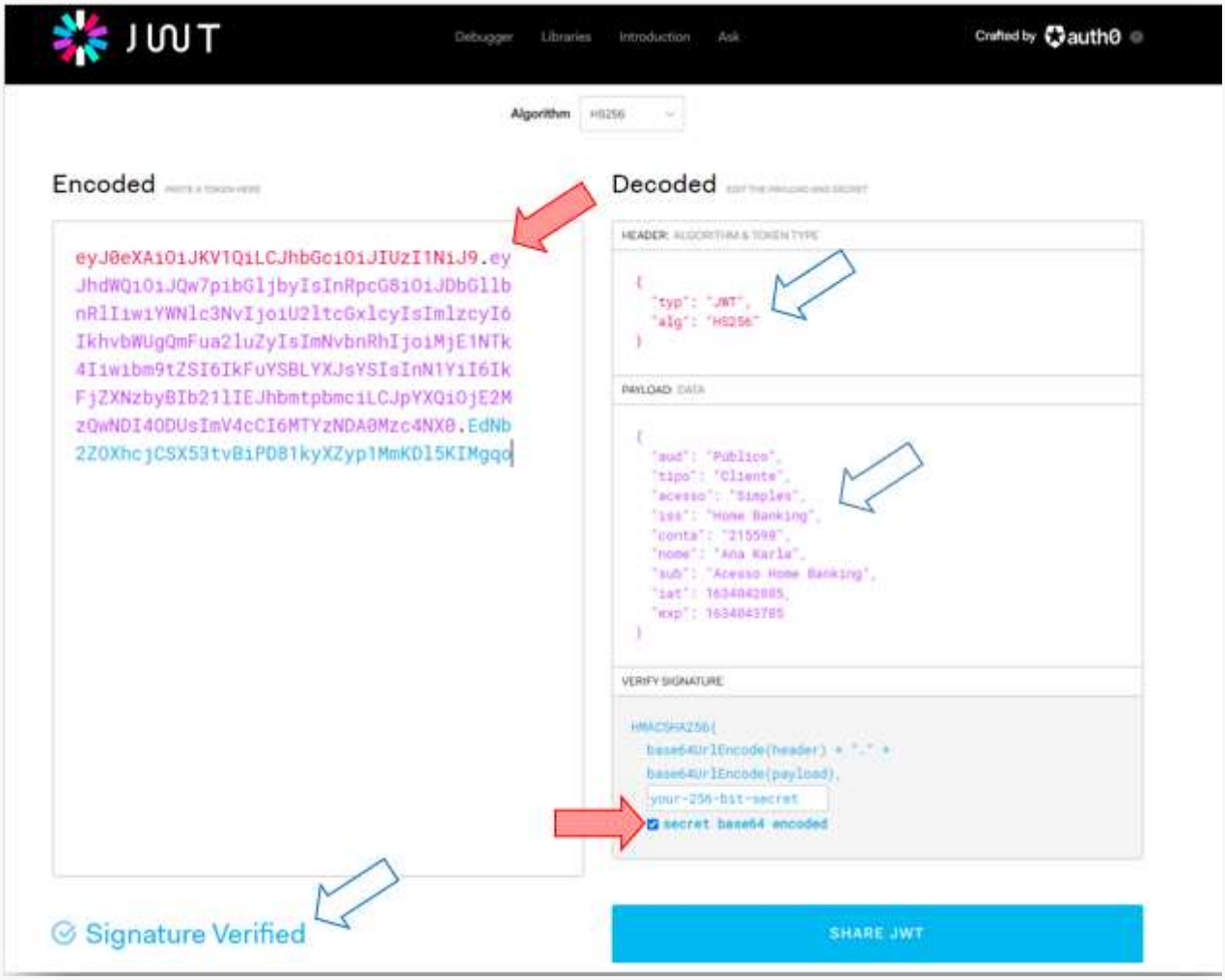
Veja a seção de assinatura do exemplo:

JWT.io *debugger*

Você pode validar JWTs no *site* https://jwt.io. Experimente copiar o *token* de exemplo a seguir e o informe no campo **Encoded**. Para validar a assinatura, marque a opção **secret base64 encoded.**

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJQw7pibGljbyIsInRpcG8iOiJDbGllbnRlIiwiYWNlc3NvIjoiU2ltcGxlcyIsImlzcyI6IkhvbWUgQmFua2luZyIsImNvbnRhIjoiMjE1NTk4Iiwibm9tZSI6IkFuYSBLYXJsYSIsInN1YiI6IkFjZXNzbyBIb21lIEJhbmtpbmciLCJpYXQiOj

E2MzQwNDI4ODUsImV4cCI6MTYzNDA0Mzc4NX0.yyCGwMIcSoW8ECPj_CqylOCi8LcXKtX-wHOLSl3WG1o



Altere um único caractere do *token* e veja que a assinatura ficará inválida.

Agora que conhecermos a estrutura de um JWT, vamos ver uma biblioteca Java que permitirá construir *tokens* assinados e validá-los.

Biblioteca JSONWebToken

Esta biblioteca Java permite facilmente construir *tokens* assinados e validá-los. Ela é composta por três projetos:

- **jjwt-api**: contém as especificações da biblioteca.
- **jjwt-impl**: contém as implementações das respectivas especificações.
- **jjwt-jackson**: contém implementações para manipulação dos objetos JSON.

Na seção anterior, vimos que a assinatura depende de um conteúdo a ser criptografado, um algoritmo de criptografia e um segredo, que chamaremos chave privada. Veremos agora como definir cada uma dessas partes utilizando a biblioteca.

Chave privada

Há diversas formas de construir chaves privadas. Vamos explicar uma das mais simples e eficientes.

O primeiro passo é definir um dado que será guardado em segredo.

String segredo = "b8338e24f11f4692a95738fe2e893c2ab8338e24f11f46";

Por segurança, o conteúdo do segredo deve ser superior a 256 bits, portanto você não conseguirá utilizar segredos curtos.

O segredo pode ser armazenado em um banco de dados de forma criptografada ou em dispositivos de *hardware* específicos para o armazenamento seguro. Em nossos exemplos, por simplificação, o segredo será armazenado no próprio código-fonte.

Agora, precisamos converter a *string* **segredo** para o formado Base64 e convertê-la para um *array* de bytes. Aqui, utilizaremos a classe **Decoders** da biblioteca **jsonwebtoken**.

byte[] keyBytes = Decoders.***BASE64***.decode(segredo);

O último passo será a criação da chave privada (**java.security.Key**), por meio de outra classe da biblioteca. A classe **Keys** possui o método **hmacShaKeyFor** para criação da chave, que utilizará o algoritmo HMAC SHA256.

Key chavePrivada = Keys.*hmacShaKeyFor*(keyBytes);

***Header* do JWT**

Nossos *headers* podem ser criados por meio da classe **Map** do Java, ou seja, vamos utilizar apenas recursos básicos da linguagem. Neste exemplo, criaremos um *header* para definir o tipo do *token*.

Map<String, Object> headers = **new** HashMap<String, Object>();

```
headers.put("typ", "JWT");
```

***Payload* do JWT**

De forma semelhante aos *headers*, utilizaremos a classe **Map** para definir os dados que irão compor o *token*.

```
HashMap<String, String> claims = new HashMap<String, String>();

claims.put("iss", "Home Banking");

claims.put("aud", "Público");

claims.put("conta", "215598");

claims.put("nome", "Ana Karla");

claims.put("tipo", "Cliente");

claims.put("acesso", "Simples");
```

Não há limites para o número de declarações, mas atente-se para não informar dados sigilosos, pois o *token* pode ser lido.

Declaração de datas

Há duas datas importantes nos JWTs: o momento da criação do *token* e seu prazo de validade. Para informar a data e hora atual, podemos criar um objeto do tipo **Date**, com seu construtor-padrão.

```
final Date dtCriacao = new Date();
```

Imagine que o *token* terá um prazo de validade de 15 minutos; após esse período, o usuário precisará renová-lo. Podemos representar o momento de expiração a partir da data de criação acrescentada de 15 minutos.

```
final Date dtExpiracao = new Date(dtCriacao.getTime() + 1000 * 60 * 15);
```

Lembre-se de que no Java o tempo é representado em milissegundos, por isso multiplicamos 15 x 60 x 1000.

Construção do *token*

O *token* é uma *string* gerada pela classe **Jwts** da biblioteca. Observe que passaremos por parâmetros os *headers*, as declarações, o assunto, a data e hora de criação e expiração do *token* e a chave privada.

```
String jwtToken = Jwts
        .builder()
        .setHeader(headers)
        .setClaims(claims)
        .setSubject("Acesso Home Banking")
        .setIssuedAt(dtCriacao)
        .setExpiration(dtExpiracao)
        .signWith(chavePrivada).compact();
```

O resultado dessa operação é o JWT na versão compactada, separada por ponto.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 .
eyJhdWQiOiJQw7pibGljbyIsInRpcG8iOiJDbGllbnRlIiwiYWNlc3NvIjoiU2ltcGxlcyIsImlzcyI
6IkhvbWUgQmFua2luZyIsImNvbnRhIjoiMjE1NTk4Iiwibm9tZSI6IkFuYSBLYXJsYSIsInN1YiI
6IkFjZXNzbyBIb21lIEJhbmtpbmciLCJpYXQiOjE2MzQwNDk0MTYsImV4cCI6MTYzNDA1MD
MxNn0 . 3at_dSHZyQV-i_UxrC_ZEG75UbUhqj3D18i__g4BGMg

Como validar o JWT

A biblioteca permite validar o *token* por meio do método **parserBuilder**, que receberá por parâmetros o *token* (no formato *string*) e a chave privada, que validará se o *token* foi de fato criado por ela.

```
Jws<Claims> credencial = Jwts

        .parserBuilder()

        .setSigningKey(chavePrivada)

        .build().parseClaimsJws(jwtToken);
```

Caso o *token* não seja válido, exceções serão lançadas, ou porque a assinatura não foi validada, ou porque seu conteúdo foi adulterado, ou porque o *token* está expirado.

Se nenhuma exceção for lançada, significa que o *token* é válido e seus dados podem ser obtidos utilizando o objeto do tipo **Jws<Claims>**. Veja alguns exemplos:

- Ler o *subject* do *token:* credencial.getBody().**getSubject**()
- Ler a data de criação do *token:* credencial.getBody().**getIssuedAt**()
- Ler a data de expiração do *token:* credencial.getBody().**getExpiration**()
- Ler todas as declarações do *token:*

```
Iterator<String> it = credencial.getBody().keySet().iterator();

while (it.hasNext()) {

    String chave = it.next();

    String valor = credencial.getBody().get(chave).toString();

    System.out.println(chave + " = " + valor);

}
```

Próximos passos

Neste momento, temos condições de construir um gerenciador de identidade e acesso que utilizará o padrão RBAC para organizar os acessos aos recursos por meio de perfis. Além disso, utilizaremos o padrão JWT para trafegar com segurança as credenciais dos usuários, resultando em um sistema de *login* único.

RBAC e JWT

Nesta videoaula, vamos explicar como a união do padrão RBAC e JWT poderá ajudar na construção de um sistema de login único.

CONCLUSÃO

- A gestão de identidade e acesso tem por finalidade gerenciar os acessos aos serviços e recursos de uma solução de TI.
- O RBAC descreve uma metodologia para restringir o acesso a recurso por meio da atribuição de funções e autorizações.
- O JWT é um padrão que estabelece uma forma para transmissão segura de informações entre duas partes.
- A biblioteca JSONWebToken permite facialmente construir *tokens* assinados e validá-los na linguagem Java.

REFERÊNCIAS

FERRAIOLO, D.; CUGINI, J.; KUHN, D. R. Role-based access control (RBAC): features and motivations. *In*: ANNUAL COMPUTER SECURITY APPLICATION CONFERENCE, 11., 1995, New Orleans. **Proceedings** [...]. [S.l.: s.n.], 1995. p. 241-248.

MICROSOFT AZURE. **O que é o RBAC do Azure (controle de acesso baseado em função do Azure)?** 2020. Disponível em: https://docs.microsoft.com/pt-br/azure/role-based-access-control/overview. Acesso em: 19 ago. 2020.