Autenticação REST OAuth2 em Java com Apache Oltu

Postado dia 10/01/2017 por João Paulo Sossoloti em Arquitetura, Programação 21

Atualmente existem diversos projetos que estão sendo desenhados no formato SinglePage ou Microservices, com uma camada REST no servidor respondendo algum formato, geralmente JSON. Quando falamos em aplicações web, necessitamos de uma forma segura e usual de autenticar os usuários e atrelá-los a perfis (autorização). Normalmente pensaríamos em um objeto de sessão gerenciado por algum framework (SpringSecurity ou JAAS), mas isso quebra os conceitos de Restful (servidor sem estado).

Então, como implementar segurança sem ferir os conceitos e design da arquitetura?

A ideia central é que a identificação do usuário seja trafegado durante as requests sem mantê-las no servidor, e dessa maneira, podemos recuperá-la para tratar a segurança. A equipe do projeto pode solucionar isso desenvolvendo seu próprio código e formato de autenticação, trafegando um usuário e senha encriptada ou algo do tipo. No entanto, podemos usar um padrão já estabelecido chamado OAuth. Ele não é um protocolo de segurança, e sim um padrão de código em cima do HTTP, sendo uma arquitetura de referência desse tipo de solução. No entanto, existem outros conceitos bacanas como o Shared Key Authentication abordado nesse artigo do Rodrigo.

O padrão OAuth 1 foi criado em 2006 por Blaine Cook em uma solução open source ao Twitter, quando em 2007 foi utilizada pela Google e expandida ao mercado. Muitos acham que o OAuth 1 é muito burocrático de implementar, com regras fixas e sem muitos conceitos para web. Por isso em 2012 criou-se uma atualização chamada OAuth 2, a que iremos abordar nesse artigo. Na referência RFC 6749 temos todos os conceitos do OAuth2 e vamos resumi-los em dois tipos:

- **Tipo de Acesso por Password**: O servidor espera que o aplicativo (ou navegador) que necessite de acesso passe no corpo da request um usuario e senha previamente cadastrado no servidor. Com posse dessas credenciais, gera-se um número aleatório que identifica-o nas próximas requisições (o Token). Alguns sistemas não aproveitam o login e senha do usuário para gerar o token, e assim, entra-se no site para solicitar uma outra credencial, aumentando a sensação de segurança. Esse tipo de acesso é bacana para quando temos uma aplicação Rest distribuída ou Microserviços sem acessos a outros sistemas externos.

- **Tipo de Acesso por Código de Autorização**: Neste caso, o aplicativo (ou navegador) ao tentar acessar o recurso no servidor, recebe como response um redirect (HTTP 302) para uma URL previamente cadastrada. O redirect contem dados na request para poder voltar ao servidor após concluído o login com um código de autorização. A partir dos demais acessos gera-se um número aleatório que identifica-o nas próximas requisições (o Token). Esse tipo de acesso é o mais comum entre aplicativos distintos, por exemplo, quando usamos login pelo Facebook, Google ou Twitter.

Como visto, o OAuth2 é mais um conceito do que um formato de arquitetura, e dessa forma, não diz sobre como será a implementação. Um código OAuth2 pode ser implementado diferente se comparado com outro código OAuth2, mesmo que os dois sigam todos os preceitos.

Vamos simular aqui um servidor que recebe dados de uma mesma fonte usando o navegador, mas poderíamos fazer o mesmo para um outro aplicativo do mesmo site (como uma versão mobile). Sendo assim, exemplificaremos o tipo de acesso *password*, e para isso, iremos usar um framework Java chamado Apache Oltu. Já usei-o em duas aplicações que estão em produção, e funciona bem para o que ele foi proposto.

A ideia do Apache Oltu é manipular a requisição e a geração do Token. O restante, sobre como armazenar esse Token, gerenciar o tempo de expiração ou bloqueio de requisições a recursos restritos, ficam a cargo do programador. Olhando apenas para a referência sobre o OAuth2, ele realmente não diz nada sobre essas outras partes. Cada aplicação pode fazê-las da forma que achar melhor. Por exemplo, onde armazenar o Token para recuperação entre requisições e identificação do usuário logado, você poderia optar por:

1. Armazená-lo em uma tabela no banco de dados.
2. Guardá-lo em uma persistência NoSQL.
3. Adicioná-lo em um mapa em memória.

O OAuth2 não vai impedir ou indicar a melhor solução. Em nosso projeto de exemplo, optamos por seguir alguns dos caminhos seguros, como armazenar o Token em uma tabela no banco de dados relacional e criar um Interceptador do Spring para gerenciar as requisições. Mas, novamente, escolha o que acredita ser o ideal para a sua aplicação. O exemplo pode ser conferido no github.

- Clone o projeto: git clone https://github.com/jopss/exemplo-oauth2.git
- Execute o maven: mvn clean install.
- Configure a base de dados PostgreSQL ("bd_exemplo_oauth" com o schema "main").
- Importe na sua IDE e configure no Tomcat.
- Acesse pelo navegador: http://localhost:8080/exemplo-oauth2/

Para um cliente efetuar o login de uma aplicação que utiliza OAuth2, deve criar um POST contendo dentro do formulário alguns dados padrões. Se o nome das propriedades do form não for o esperado na especificação, o Apache Oltu retornará erros.

- client_id: indica o id do APLICATIVO cliente.
- client_secret: indica a senha doAPLICATIVO cliente.
- grant_type: indica qual o tipo de autenticação. No exempo iremos de "password".
- username: login do USUÁRIO.
- password: senha do USUÁRIO.

Veja que além do tipo de autenticação e dados do usuário, temos a identificação do aplicativo cliente. Isso é bacana para gerenciar quem está acessando nosso servidor, podendo criar um formulário para tal, criando, inativando ou excluindo acessos.

Código Java

Em código no servidor a recuperação dos dados do login é automatica dentro do framework, onde pedimos as propriedades sem mexermos na Request:

```
OAuthTokenRequest oauthRequest = new OAuthTokenRequest(request);
String appClientId = oauthRequest.getClientId();
String appClientSecret = oauthRequest.getClientSecret();
String senha = oauthRequest.getPassword();
```

```
String login = oauthRequest.getUsername();
```

Após verificado em regra dados de usuários e permissão do aplicativo, se tudo ok, podemos gerar o Token:

```
String accessToken = new OAuthIssuerImpl(new MD5Generator()).accessTo
```

Por fim retornamos esse Token e qualquer outro dado por uma resposta JSON.

```
return OAuthASResponse.tokenResponse(HttpServletResponse.SC_OK).setAc
.setParam("nome", usuario.getNome())
.setParam("login", usuario.getLogin())
.setParam("perfil", usuario.getPerfil().getNome())
.buildJSONMessage();
```

Isso é o suficiente para identificar usuários, aplicativos e retornar o Token para quem chamou o login com OAuth2. Como mencionado anteriormente, esse Token deve ser armazenado em algum local e faremos isso em um banco de dados relacional.

Para os clientes, nas próximas requisições, é necessário colocar como parâmetro do Header um atributo de autenticação:

- Authorization: Bearer TOKEN

Onde o TOKEN é o valor retornado pelo login. Agora, novamente quanto ao servidor, como verificar nessas próximas requisições se o Token passado está valido e a qual usuário está associado? Usamos o framework de novo, sem acessarmos diretamente o Request:

```
OAuthAccessResourceRequest oauthRequest = new OAuthAccessResourceRequ
String token = oauthRequest.getAccessToken();
```

Se não houver os parâmetros corretos, retornará um erro. Com o Token em mãos, validamos se existe o mesmo no banco de dados e demais regras. Por fim, basta colocarmos esse código em um filtro HTTP ou Incerceptor do Spring e teremos uma verificação a cada requisição ao recursos.

Um recurso importante, que não iremos abordar aqui, é a revalidação de uma requisição. O Token possui um tempo de expiração e o cliente pode, por uma URL, solicitar a recriação da chave para ele continuar usando nas suas requisições. O Apache Oltu tem métodos para recriar o Token.

Em nosso exemplo demos uma inteligência um pouco maior associando também os perfis com uma anotação @Publico ou @Privado nos métodos do controle, indicando qual perfil acessá-lo. Isso está além do escopo do OAuth2 e fica a cargo da aplicação.

Veja ainda no Github do projeto de exemplo como testar os acessos simulando requisições cliente usando cURL, de forma simples e rápida.

Segurança dos Dados

Muita gente que entende e estuda a solução acaba perguntando duas coisas:

1. Mas não é inseguro trafegar a cada requisição um Token que identifica um usuário?
2. Com posse de um Token gerado previamente, eu não poderia entrar na aplicação simulando outro usuário?

Para as duas respostas, é sim. Só que estranhamente as pessoas esquecem que a tão segura Sessão Web funciona exatamente dessa mesma forma, mas a geração do token (SessionID) e adição nos cookies é automática entre navegador e servidor. Com qualquer navegador podemos mexer nos cookies e pegar os dados de sessão simulando logins com outros usuários. Então, quanto a essas duas perguntas, funciona exatamente como é hoje.

E não esqueçam de configurar o HTTPS nos servidores. Isso sim faz muita diferença em relação a segurança 🙂

E você, como você pretege seus recursos Restful?

Compartilhe isso:

446

Relacionado

Morte à sessão! Entenda esse tal de stateless session com tokens
03/04/2017
Em "Inovação"

Protegendo sua API Rest via Shared Key Authentication
12/08/2015
Em "Programação"

Receba notificações da api de Servlet via Listeners
18/12/2012
Em "Programação"

Tags: Arquitetura, autenticação, escalabilidade, Java, javaee, oauth, segurança

João Paulo Sossoloti

21 COMENTÁRIOS

Fernando Almeida

13/01/2017 at 14:31 #

Muito bom o artigo, simples e objetivo. Tirou um pouco do receio que tinha dessa implementação(vi uns tutoriais assustadores hehe). Hoje utilizo Spring Security com oAuth2 e já me atende, mas é sempre bom uma oura visão de como pode ser feito também essa implementação. Apenas uma dúvida, o status resposta da Url cadastrada não seria o HTTP302(found)? visto que o recurso foi encontrado com sucesso?

João Paulo Sossoloti

13/01/2017 at 15:11 #

Obrigado Fernando. E é verdade! Valeu pela correção.

Eduardo Ribeiro

20/01/2017 at 09:33 #

Parabéns pelo post. Exemplo simples, objetivo e esclarecedor.

João Gabriel

27/01/2017 at 02:30 #

Excelente artigo, João Paulo.

Tenho poucos conhecimentos em OAuth e me surgiu uma dúvida sobre aplicações de terceiros.

Para configurar a autenticação entre duas aplicações, sem a interação do usuário, qual solução você indica? Exemplo: queremos criar uma relação de confiança entre Twitter e Facebook para que a primeira publique conteúdo na segunda. Neste caso, o usuário autorizaria o uso do Facebook pelo Twitter uma única vez.

Obrigado!

Plínio Mos

01/02/2017 at 04:23 #

Exemplo rápido e claro. Muito bom

João Paulo Sossoloti

01/02/2017 at 08:59 #

“Para configurar a autenticação entre duas aplicações, sem a interação do usuário, qual solução você indica?”

Olá, um dos pontos do OAuth é justamente para esse tipo de Single Sign-on. Através da sua conta no Facebook, vc cria uma chave (e/ou senha) que irá servir de registro do Single Sign-on no Twitter (e ainda dar as permissões necessárias). Em código, o Apache Oltu dá suporte a este tipo de fluxo também.

Valdomiro

01/02/2017 at 13:18 #

Muito bacana seu post, parabéns simples e direto, vou deixar algumas dicas aqui para os leitores para complementar seu post, se eu estiver errado favor pode corrigir.

Lembrando que mesmo utilizando HTTPS podemos ter falhas pois a SSL 1.0 já foi quebrada em 2016, talvez vale a pena utilizar versões mais recentes.

Quando trabalhamos com autorização com tokens seja oauth ou qualquer outro, o importante é que esse token não seja exposto para a camada de front ou seja manter o fluxo de transição do token apenas em backend utilizando uma API por exemplo.

Para deixar um pouco mais seguro utilizamos as vezes o conceito de clientid e secret nos requets para dar um duplo check junto ao token do oauth as vezes utilizamos uma camada acima do token abstraindo o token utilizando um jwt token e dentro utilizando o clientid e secret de chave assim quem conseguir interceptar o token ainda precisará abrir o jwt token pra depois obter o token de autorização.

Existem várias boas práticas inclusive a especificação do oauth ajuda muito.

Espero ter ajudado.

ABS.

Tiago Wanke Marques

08/02/2017 at 13:54 #

Muito bom o post.

Eu desconhecia o Apache Oltu, usei Apache Shiro para questão de autenticação e autorização em meus projetos. Fiquei na dúvida da diferença entre os dois. Um não acaba concorrendo com o outro?

João Paulo Sossoloti

08/02/2017 at 15:33 #

Olá. Nunca usei o Shiro, mas pelo o que há na documentação, ele é voltado para gerenciamento de sessões (com diversas integrações). Este aqui é voltado para “tokenização” de sistemas rest stateless (não há sessão usada no servidor).

Gabriel Segers

17/02/2017 at 15:39 #

João, boa tarde.

O protocolo OAuth2 sempre necessita de uma persistência em algum lugar(NoSQL, DB, Memória)? Estava discutindo com meu irmão esta necessidade, pois ele usa o IdentityServer do C# e diz que não precisa persistir o token em nenhum lugar.

João Paulo Sossoloti

17/02/2017 at 16:02 #

Olá.

Depende de como quer implementar. O OAuth não entra nesse mérito. Se não quiser persistir, pode deixá-lo em memória apenas, mas ao reiniciar o server todos os clientes precisam logar novamente. Se isso não for um problema, é até melhor, mais rápido e seguro.

Não sei como o IdentityServer funciona, só sei que ele deixa mais transparente e desacoplado o gerenciamento de token para uma app externa aos client/server. Por ser um componente robusto, ele deve ter várias opções de busca de chaves, sendo local (bd, nosql ou memoria), algum tipo de repo externo em rede, etc.

leandro prates

21/02/2017 at 11:54 #

No caso de voce usar Basic authentication junto com o Spring isso é pior que OAuth ja que em ambos os casos teriamos de qualquer jeito usar SSL ?
Qual o mais recomendado ?

João Paulo Sossoloti

21/02/2017 at 11:59 #

Olá Leandro.

O SSL proteje os dados da requisição entre o cliente e o servidor. Não tem relação com autenticação, sendo Basic, tokenizado, sessão ou outro meio...

Rafael Rossignol Felipe

13/03/2017 at 16:53 #

Presumindo que o padrão REST é sem estado e o Oltu/OAuth2 reiteram isso. Faz sentido eu diminuir drasticamente o tempo da sessão do servidor?

Gabriel

01/08/2017 at 14:36 #

Olá João, existe a possibilidade de obter este token de autenticação em um projeto web do eclipse? preciso desse token para utilizar o google cloud storage e gostaria de saber por onde começar, qual tipo de projeto criar, se é um servelet ou controller...

João Paulo Sossoloti

01/08/2017 at 14:53 #

Olá Gabriel. Aqui no post a gerência do Token fica a cargo da aplicação web que estamos criando, usando OAuth2 e Apache Oltu. O cliente, então, faz simples requisições HTTP, que pode ser outra aplicação web, desktop, mobile ou linha de comando.

No seu caso, a gerência do token é no Google Storage. Você precisa ir lá e pegar o token de acesso. E assim como no exemplo, o cliente, então, usando requisições HTTP, pode ser sim qualquer aplicação web, desktop ou linha de comando.

João Paulo Sossoloti

02/08/2017 at 09:29 #

Se é stateless mesmo, faz sentido você ter o mínimo de tempo de sessão possível, pois você não a usa. Só toma cuidado pra ver se a sua aplicação de alguma forma não tenta usar sessão para guardar o Token.

Danilo

02/08/2017 at 11:00 #

Olá Jõao, estou usando uma plataforma de desenvolvimento low code e para obter o token eu utilizo um REST, gostaria de saber se você possui a url do POST citado acima, e se os dados passados são colocados no header ou no corpo da request, obrigado!

João Paulo Sossoloti

02/08/2017 at 11:20 #

Olá. Os dados POST são passados no corpo da request normalmente, nada muda. Pensando em linha de comando, seria algo neste sentido aqui:

```
curl -X POST -H "Accept: application/json" -H "Authorization: Be
```

Danilo

02/08/2017 at 11:40 #

João, o meu problema é obter o “Authorization code”, pois a partir dai eu consigo gerar um “access token”, ainda estou meio confuso com o processo desde REST

DEIXE UMA RESPOSTA

Comentário

Nome (Obrigatório)

E-mail (Obrigatório)

Site

Enviar comentário

Notifique-me sobre novos comentários por e-mail.

Notifique-me sobre novas publicações por e-mail.