FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDUARDO ARAKAKI

PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH

MARÍLIA
2015

EDUARDO ARAKAKI

PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Ms. Ricardo José Sabatine.

MARÍLIA
2015

ARAKAKI, Eduardo

PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH / Eduardo Arakaki; orientadora: Profª. MS. Ricardo José Sabatine. Marília, SP: [s.n.], 2014.

Monografia (Bacharelado em Ciência da Computação): Centro Universitário Eurípides de Marília.

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Eduardo Arakaki

TÍTULO: PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH.

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 7,5 (sete e meio)

Orientador: Ricardo José Sabatine Ricardo Sabatine

1°.Examinador: Ildeberto de Gênova Bugatti

2°.Examinador: Fabio Lucio Meira

Marília, 30 de novembro de 2015.

AGRADECIMENTOS

Ao professor Ricardo Sabatine, que me orientou durante este trabalho.

À minha namorada Angélica que sempre me apoiou e me compreendeu nas diversas vezes em que deixei de estar com ela para terminar este trabalho.

ARAKAKI, Eduardo. **PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH**. 2015. 62f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2015.

RESUMO

Com o grande crescimento do acesso a internet, a segurança dos dados se tornou de extrema importância para todos. Nos dias atuais em que todos usam serviços como redes sociais, a informação restrita para devidos usuários mal-intencionados se tornou fundamental. Assim o presente trabalho teve como principal objetivo pesquisar e comparar mecanismos de autenticação e autorização com foco no protocolo OAuth, além da implementação de uma API OAuth Mobile open-source de integração com serviços baseado no protocolo para uso da comunidade. Para aplicar os conhecimentos obtidos, foi desenvolvido um ambiente para teste com uma aplicação Android para consumir o recurso protegido do serviço também implementado e por fim uma analise comparando as versões do protocolo.

Palavras-Chave: Protocolo OAuth, API Java, Mobile

ARAKAKI, Eduardo. **PESQUISA E COMPARAÇÃO DE MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO: ESTUDO DE CASO DO OAUTH**. 2015. 62f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2015.

ABSTRACT

With the huge growth of internet access, data security has become very important for everyone. Nowadays they all use services such as social networks, restricted information due to malicious users has become critical. Work gift has as director purpose Search and Compare Authentication and authorization mechanisms focusing on the OAuth protocol In addition to implementing a mobile API OAuth open-source and Integration Services based on the protocol for use community . For applying knowledge obtained, developed hum test environment with a paragraph Android Application paragraph consume the protected also implemented of service resource and finally an analysis comparing how versions of the protocol .

Keywords: OAuth protocol, Java API, Mobile

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

OAuth	*Open Authorization*
JSON	*Javascript Object Notation*
REST	*Representational State Transfer*
API	*Application Programming Interface*

SUMÁRIO

INTRODUÇÃO

Com o grande crescimento do uso da internet, a segurança da informação que navegam pela internet, se tornou um grande problema na vida de todos. O grande número de dados que podem ser acessadas utilizando a web, demanda por cada vez mais segurança de acesso a dados pessoais.

A informação é um ativo que, como qualquer outro ativo importante para os negócios, tem um valor para a organização e, consequentemente, necessita ser adequadamente protegida. A segurança da informação protege os dados de diversos tipos de ameaças para garantir a continuidade dos negócios, minimizar os danos aos negócios e maximizar o retorno dos investimentos e as oportunidades de negócio. (ISO/IEC NBR 17799, 2005).

Junto com o crescimento das redes abertas fez com que surgissem vários problemas de segurança, que vão desde o roubo de senhas e interrupção de serviços até problemas de fraudes de autenticação para ter acesso privilegiado. (FIORESE, 2000). Com esse problema, surgiu a necessidade de autenticação, que consiste na verificação e validação da identidade tanto dos usuários quanto dos sistemas e processos. Os mecanismos de autenticação de usuários dividem-se em três categorias: baseados no conhecimento baseados em propriedade e baseados em características. (YOUNG, 1996).

Para alcançar um nível de segurança maior, qualquer sistema *web* que possua dados importantes precisa assegurar a proteção dos dados e recursos contra a divulgação não autorizada e modificações impróprias ou proibidas, e ao mesmo tempo, garantir a utilização aos usuários que possuem permissão. A segurança da informação, portanto, deve assegurar que todos os acessos a um sistema e seus recursos sejam controlados e que apenas acessos autorizados sejam permitidos. Assim a autorização a recursos protegidos também faz parte nesse requisito de segurança junto com a autenticação.

O controle de acesso defini as atividades autorizadas a quem tem permissão, e controla todo o acesso aos recursos protegidos do sistema, definindo o que pode e o que não pode ser acessado. Atualmente, o principal protocolo utilizado pelos serviços web como Twitter, Facebook, Dropbox, entre outras, usam o protocolo de autenticação e autorização OAuth que faz esse controle de segurança.

Objetivos

Com o elevado número de pessoas acessando conteúdos web, a segurança da informação se tornou importante para todos. Por este motivo é importante um estudo dos principais métodos e conceitos do protocolo OAuth e suas versões para uma comparação dos mais seguros e viáveis com relação ao desenvolvimento relacionado ao protocolo.

Objetivos Específicos

Pesquisar o protocolo OAuth e nesse contexto desenvolver uma API OAuth mobile, um serviço e uma aplicação para testes.

Os objetivos específicos são basicamente:

- Pesquisar e testar métodos isolados de redes sociais utilizando o protocolo OAuth, para entender como funcionam esses métodos;
- Desenvolver uma API OAuth de integração com serviços;
- Testar a API em aplicações testes para ver as possíveis falhas;
- Produzir documentação para pesquisas e desenvolvimentos de melhorias;
- Implementar as funções para testes;
- Desenvolver um serviço e aplicação real para demonstrar os resultados obtidos.

1 SEGURANÇA

Com o grande crescimento do acesso às aplicações *web*, como redes sociais, reunindo os mais diferentes tipos de usuários, objetivos e dispositivos, tornou a segurança da informação ainda mais importante. Um usuário pode ter varias contas ou identidades na internet e em muitas situações, é fundamental garantir que a credencial apresentada seja válida.

A necessidades de segurança dos ambientes tecnológicos sofrem alterações e avanços constantes, assim os provedores de conteúdos evoluíram a forma de segurança para solucionar as falhas em sistemas computacionais. (MARTINS, 2007).

Na *Internet*, a segurança é fundamental, pois várias tarefas que exigem segurança são utilizadas, como por exemplo, acesso às contas bancárias, compras *online*, entre outras, várias informações importantes e confidenciais são trocadas a todo momento na internet.

É comum prover segurança em ambientes distribuídos, atribuindo a execução de funções de segurança aos serviços distribuídos nos ambientes que o integram. O uso mais comum do uso da segurança é a imposição de um conjunto de regras implementadas através de controles criptográficos e de acesso.

A segurança garante o fornecimento do serviço e também garante que não ocorram violações de segurança. Segundo Russel e Gangeni (1991), Amoroso (1994), a segurança está fundamentada em quatro propriedades que devem ser garantidas:

- Disponibilidade: Garantir que as informações e recursos num sistema estarão bloqueados e prontos para serem usados quando requisitados por usuários autorizados;
- Autenticidade: Acesso ao sistema só deverá ser feito por usuários autênticos.
- Confidencialidade/Autorização: A informação só deve ser revelada para usuários autorizados a acessá-la;
- Integridade: A informação não pode ser modificada, intencionalmente ou acidentalmente, por usuários que não possuam acesso para isso;

A quebra de segurança pode ocorrer devido à exploração das vulnerabilidades de código existentes. Uma falha no código, um erro de configuração ou mesmo um erro de operação, pode permitir que pessoas mal-intencionadas entrem no sistema, podendo assim comprometer o funcionamento correto do sistema. (BISHOP, BAILEY, 1996).

Uma ameaça consiste em uma possível ação que, se concretizada, poderá produzir efeitos indesejados ao sistema, comprometendo a confidencialidade, a integridade, a disponibilidade e/ou a autenticidade. Já o ataque é a concretização de uma ameaça, através da exploração de alguma vulnerabilidade do sistema, executado por algum intruso de forma maliciosa ou não.

As quatro categorias de ataques, normalmente, identificados em sistemas distribuídos são:

- Interceptação: Uma parte não autorizada obtêm acesso à informação (revelação não autorizada de informação);
- Interrupção: O fluxo normal da mensagem é interrompida, impossibilitando que a informação chegue ao destino (negação de serviço);
- Modificação: Uma parte não autorizada modifica a informação recebida da origem e a transmite para o verdadeiro destino (modificação não autorizada da informação);
- Personificação: Entidade não autorizada transmite uma mensagem maliciosa pela rede, se passando por uma parte autêntica.

Em sistemas mais complexos, a tendência é ter mais falhas de programação ocasionando mais brechas de segurança, mesmo que nesses sistemas a segurança é mais focada.

A política de segurança de um sistema consiste em normas e procedimentos e é liberada para um sistema específico. As diretrizes ditadas em uma política de segurança, indicam o que cada componente do sistema (usuários, máquinas, etc.) pode ou não pode fazer. As normas indicam o que cada componente está habilitado a fazer e como deverá ser feito.

Segundo Melo (2002) as políticas de segurança de sistemas podem ser divididas em três partes:

- Segurança lógica: Direitos de acesso ao sistema e quais os direitos que cada usuário possuirá.
- Segurança física: Protege o meio físico em que opera o sistema;
- Segurança gerencial: Defini processos para criação e manutenção das próprias políticas de segurança;

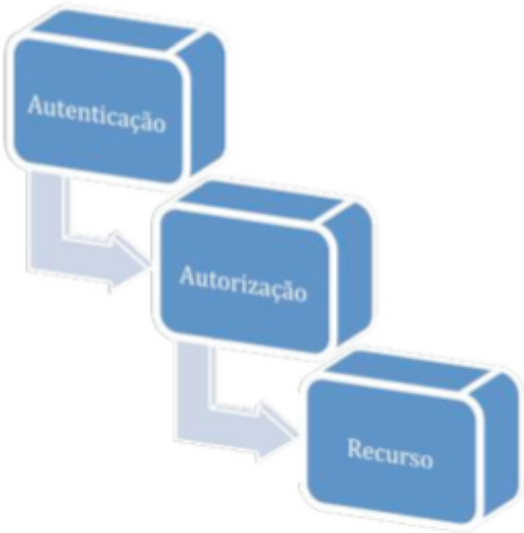
1.1 Autenticação e Autorização

A segurança da identificação única é normalmente a base para o controle de acesso a recursos protegidos de serviços usando autenticação e autorização. Os serviços e os acessos estão atribuídos ao tipo de política imposta nos controles de autorização. De acordo com as necessidades dos sistemas, são adotadas políticas para autenticação e autorização, dispostas em serviços distribuídos.

Para um utilizador acessar um determinado recurso protegido, é necessário em primeiro lugar que se autentique e em seguida seja autorizado a acessar o recurso pretendido. Um recurso é uma entidade que contém e/ou recebe informação. Cada um destes processos é explicado mais detalhadamente a seguir.

Na Figura 1 é ilustrada o fluxo até o acesos ao recurso.

Figura 1 – Passos para acesso a um recurso



1.1.1 Autenticação

A autenticação deve ser implementada para se obter aplicações seguras e lida com a identificação específica de um usuário do sistema, e com o mapeamento deste usuário para uma entidade identificável. Tipicamente, um *software* está dividido em dois domínios de alto nível como anônimo e autenticado. (MULARIEN, 2010).

Segundo Fiorese (2000), as soluções de autenticação baseadas na propriedade caracterizam-se por um objeto físico que a entidade possui. Este objeto pode ser um cartão inteligente (*smartcard*), uma chave ou um *token*.

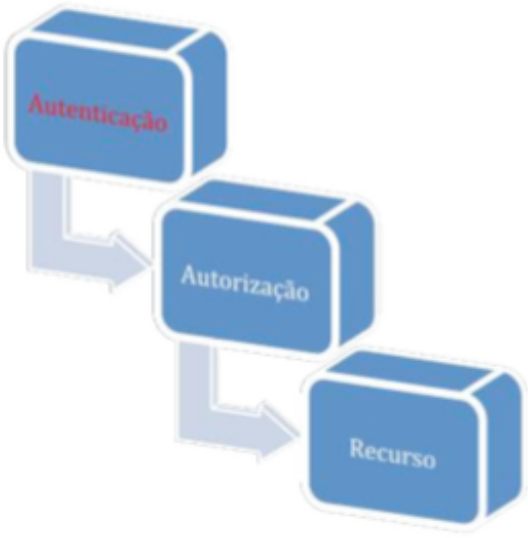
A autenticação é um processo que verifica a identidade de um usuário, confirmando se o mesmo é autêntico. Um exemplo no mundo real sobre autenticação é o processo de *check-in* nos aeroportos, onde é necessário se identificar mostrando um documento original com foto que comprove a identidade. (BOYD, 2012).

Independente do sistema ser *web*, *mobile* ou *desktop* o processo de autenticação se dá através do *login*, onde o usuário deve informar seu nome de usuário e senha para ter acesso ao sistema, dessa forma o sistema consegue identificar que o usuário é realmente autêntico.

Pode existir também funções públicas que não necessitam de uma identificação de usuário, como por exemplo, uma listagem de produtos de um *e-commerce*. As seções que permitem acesso anônimo não requerem a autenticação do usuário para sua utilização, não exibem informações confidenciais como nomes e cartões de créditos, e não fornecem meios para manipulação geral do sistema ou de seus dados. (MULARIEN, 2010).

Assim o primeiro passo é nada mais do que o processo necessário para que uma identidade seja provada. Este é um pré-requisito para o acesso a um determinado recurso (Figura 2).

Figura 2 – Passos para o acesso a um Recurso: Autenticação



As credenciais ou autenticação são criadas baseadas em algo que um indivíduo é, faz, sabe, tem, ou qualquer combinação destes fatores. O nível de segurança aumenta com o número destes fatores que são usados. Uma boa regra é quanto mais importante for o recurso a acessar, maior número de fatores deve ser utilizado.

1.1.2 Autorização

A autorização é um processo que verifica se o usuário tem a permissão para executar determinada ação, como por exemplo, visualizar determinada área restrita de um site ou banco de dados. Em um sistema, o processo de autorização geralmente ocorre após o processo de autenticação, pois primeiro é necessário verificar a identidade do usuário e após isso, verificar quais recursos do sistema o mesmo está autorizado a utilizar. (BOYD, 2012).

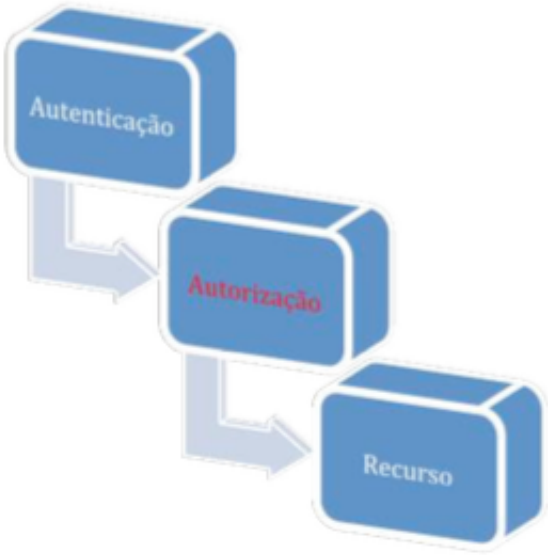
Autorização é o segundo dos dois principais conceitos de segurança cruciais na implementação e compreensão de segurança de aplicações. Autorização lida com a disponibilidade adequada de funcionalidades e dados para usuários que estão autorizados para acessar. Feito com base no modelo de autorização para a aplicação, a atividade de particionar as funcionalidades e os dados da aplicação, é feita de tal forma que, a disponibilidade destes itens possa ser controlada por uma combinação de privilégios, funcionalidades, dados e usuários. (MULARIEN, 2010)

O processo de autorização tipicamente envolve dois aspectos separados que se combinam para descrever a acessibilidade do sistema seguro. O primeiro é o mapeamento de um usuário autenticado para um ou mais papéis, por exemplo, o papel de administrador do sistema ou o papel de visitante. O segundo aspecto é a atribuição das checagens de autoridade para os recursos protegidos do sistema, o que tipicamente é feito durante o desenvolvimento do sistema, através de declaração explícita no código ou por meios de configuração. Um recurso protegido pode ser uma funcionalidade do sistema, como, por exemplo, gerenciamento de produtos. (MULARIEN, 2010).

A Autorização consiste em verificar quais são os direitos de acesso atribuídos a um utilizador em relação a um determinado recurso (Figura 3).

.

Figura 3 – Passos para o acesso a um Recurso: Autorização



O processo de Autorização tem que garantir que determinados usuários têm o acesso a determinado recurso e negar o mesmo a outros utilizadores não autorizados. A autorização implementa políticas de acesso. O objetivo principal é o de preservar e proteger a confidencialidade, integridade e disponibilidade de informação, sistemas e recursos.

1.2 Consideração Final

Para o desenvolvimento do ambiente de teste, o estudo sobre segurança, autenticação e autorização é de grande importância para entender os conceitos do protocolo OAuth que foca os princípios da segurança em aplicações.

A partir dos estudos foi possível dar andamento para o próximo passo sobre o protocolo OAuth e seus conceitos.

Baseado em autenticação e autorização foi possível verificar através de pesquisas que os principais serviços web estão migrando para o uso de compartilhamento de recursos protegidos, principalmente redes sociais.

2 PROTOCOLOS DE AUTENTICAÇÃO E AUTORIZAÇÃO

Como toda aplicação *Web*, sites de redes sociais são hospedados em servidores online. Para que outras aplicações consigam interagir com aplicações *Web*, são usados *Web Services*. Então, as API(s) de redes sociais online são *Web Services* destinados a oferecer recursos e funcionalidades presentes na rede social para aplicações externas. Essas API(s) transformam os sites de rede social em plataformas para aplicações sociais. Nessa seção serão descritas e exemplificadas as API(s) das principais redes sociais online. (BOYD. ELLISON, 2007).

As API(s) Web de Redes Sociais Online são Web Services que permitem acesso e manipulação de informações sociais. A maioria das API(s) são implementadas através dos princípios REST. O termo introduzido descreve um estilo de arquitetura para sistemas hipermídia derivado de vários estilos baseados em redes como a Web. Nessa técnica as mensagens trocadas são encapsuladas diretamente no protocolo HTTP. (ROY FIELDING, 2012)

Há um foco nos recursos e não nas chamadas aos procedimentos/serviços, como em outras abordagens. No REST são feitas requisições para recursos e não serviços. Essa abordagem pode ser interessante para aplicações em que é mais importante a interoperabilidade do que um contrato formal entre as partes.

Os recursos são representados por URI(s) e também podem ser chamados de métodos, caso representem alguma ação.

2.1 OpenAuth, da América Online (AOL)

O Flickr trabalha com um protocolo de autenticação/autorização proprietário. O protocolo foi um dos utilizados como inspiração para o desenvolvimento do OAuth. Como nas demais, na API do Flickr é necessário cadastrar uma aplicação. Com o registro da aplicação, são criados uma chave e um segredo. Caso a aplicação seja Web, é necessário configurar um URL de retorno (callback). (FLICKR, 2015)

2.2 AuthSub, do Google

O AuthSub trabalha com aplicações web que necessitam acessar recursos protegidos através de uma conta Google ou Google Apps usando uma interface proxy, que obtém o acesso ao recurso sem manipular as informações do usuário. A interface AuthSub trabalha com vários métodos para gerenciar os tokens de autorização. (GOOGLE, 2012)

O AuthSub foi removido oficialmente em 2012 e o próprio Google aconselha a migrar para o OAuth 2.0.

2.3 Browser-Based Authentication (BBAuth), do Yahoo!

Assim como o AuthSub e outros, o BBAuth foi criado para usuários acessar recursos protegidos do Yahoo!. A Autenticação é baseada em navegador como o próprio nome diz, basicamente o usuário registra a aplicação para que possa ter acesso aos dados com a permissão. O BBAuth também oferece um Sign-On (SSO), que possibilita o usuário Yahoo! usar os serviços sem ter que fazer mais uma etapa de cadastros. (YAHOO, 2015).

Assim como os demais, o Yahoo! está em etapa final de abandonar seu projeto.

2.4 ClientLogin, do Google

Até meados de 2011 o protocolo de autenticação ClientLogin foi utilizado pelo Google na autenticação e autorização do usuário no sistema Android e que durante pesquisas feitas pela Universidade de Ulm(Alemanha) foi descoberta uma vulnerabilidade detectada na transmissões de *tokens* através do uso do protocolo HTTP sem criptografia, possibilitando a captura desses *tokens* em redes abertas (como as redes wireless) durante o processo de autenticação podendo, a partir de então, ser utilizado por terceiros para o acesso aos dados do usuário. (GOOGLE, 2015).

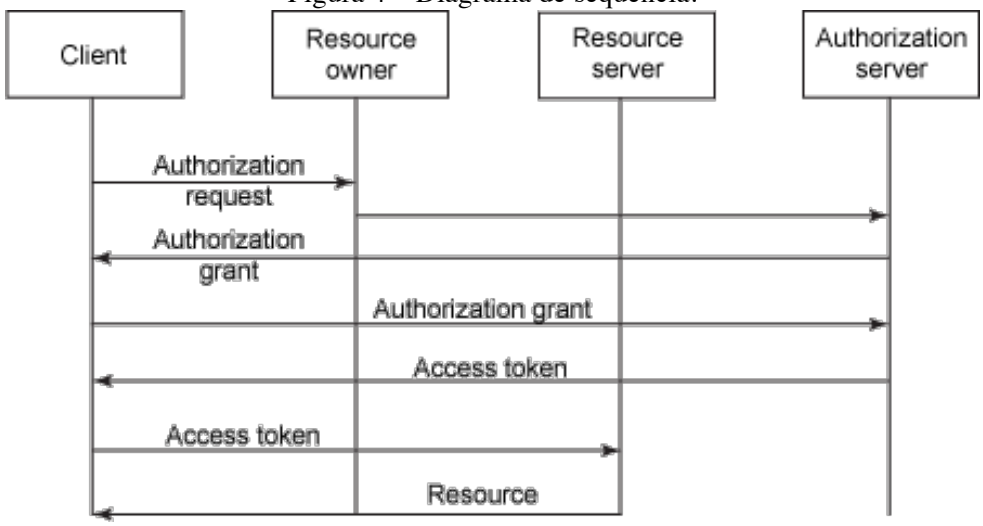
2.5 OAUTH

OAuth ou Open Authorization é um padrão aberto que permite aos usuários de um site garantirem acesso, por uma aplicação externa, aos seus recursos privados sem ter que compartilhar suas senhas e usuários. O OAuth foi desenvolvido em 2006 e o foco principal do OAuth está na autorização e não na autenticação. Sendo assim, ele prevê níveis de autorização, que o usuário pode aceitar ou não. (HAMMER-LAHAV, 2009).

O OAuth foi construído baseado nos padrões proprietários Google AuthSub, Yahoo BBAuth e Flickr API Auth. Também pode ser caracterizado como um protocolo. Sua especificação consiste em duas partes. A primeira parte define um processo no navegador para redirecionamento do usuário final para autorização aos seus recursos pelo cliente. A segunda parte define um método para realização de requisições HTTP autenticadas usando dois conjuntos de credenciais. Um conjunto destinado à identificação do cliente e outro à identificação do dono do recurso a ser requisitado. (HAMMER-LAHAV, 2009).

O protocolo segue o seguinte fluxo, como ilustrado na Figura 4.

Figura 4 – Diagrama de sequencia.



Fonte, IBM, Demystifying Oauth, 2012.

1. Token de Requisição: Quando o usuário entra na aplicação consumidora, esta requisita ao servidor um *token* de requisição. Quando a aplicação consumidora recebe o *token*, ela redireciona o usuário para a tela de autenticação do servidor. O *token* de requisição é enviado, bem como um link para redirecionamento, assim

que o usuário se autenticar;

2. Autenticação e Autorização: Ao ser redirecionado para a tela de autenticação do servidor, o usuário é requisitado se autenticar. Uma vez autenticado, o usuário recebe um questionamento acerca da autorização para a aplicação consumidora;

3. Redirecionamento à aplicação consumidora: Caso o usuário autorize o acesso, o servidor marca o *token* de requisição, enviado no passo 1, como autorizado. O usuário então é redirecionado para o URI previamente informado pela aplicação consumidora.

No modelo de autenticação cliente/servidor, o cliente usa as suas credenciais para liberar os seus recursos privados no servidor. O OAuth 1.0 tem uma terceira funcionalidade no modelo: o proprietário do recurso.

No modelo OAuth, o cliente (sem acesso ao recurso) faz pedidos de acesso aos recursos privados pelo proprietário do recurso, mas armazenado pelo servidor. Para que o cliente possa acessar os recursos privados, ele precisa obter permissão do proprietário do recurso. Essa permissão é expressa na forma de um Token e o correspondente com o "segredo partilhado" (shared-secret). O objetivo do *token* é não compartilhar o acesso (usuário e senha) do proprietário do recurso com o cliente. Ao contrario das credenciais do proprietário do recurso, os *tokens* podem ser emitidos com uma chave restrita e com expiração, e revogados de forma independente. (IBM, 2012).

- Desenhado para HTTP
- Definido no RFC 5849
- Baseado em alguns protocolos proprietários (Google AuthSub, API Flickr e Yahoo BBAuth).

Ao fornecer um *token* e o responder com o segredo partilhado (shared-secret) é possível para o proprietário do recurso dar acesso a um recurso protegido sem revelar as suas credenciais (utilizador/senha) ao serviço final que vai acessar os dados (cliente). Para isto é necessário que o servidor de recursos protegidos e o cliente suportem o protocolo OAuth. O *token* adiciona uma vantagem adicional para o acesso pretendido, por ser capaz de definir o acesso e o período de tempo válido. (IBM, 2012).

Há três entidades principais envolvidas numa conversação OAuth:

1. Proprietário do recurso: o utilizador final;
2. Cliente: aquele que vai acessar os recursos, geralmente um servidor;
3. Servidor de recursos protegidos: o servidor que guarda os recursos do proprietário.

Estes princípios são usados em qualquer transação OAuth. Por vezes, o proprietário do recurso e o cliente são o mesmo.

O OAuth usa três tipos de credenciais:

1. Credenciais do Cliente: As credenciais do cliente são usadas para autenticar o cliente.
2. Credenciais de Token: As credenciais de *token* são usadas para substituir o nome do proprietário dos recursos e da senha.
3. Credenciais temporárias: O processo de autorização OAuth também usa um conjunto de credenciais temporárias que são usadas para identificar o pedido de autorização.

Todos os pedidos baseados em OAuth são muitos parecidos. Para identificar um recurso de um pedido, é construída uma *string* que descreve o pedido e as suas credenciais necessárias, e então a *string* é assinada usando um conjunto de segredos. É como endereçar uma carta onde o endereço e o carimbo não descrevem apenas o destino, mas também o conteúdo. (ALVES, 2011).

Fluxo do pedido OAuth 1.0:

- É pedido um Token de pedido e é especificado o seu retorno;
- O utilizador é redirecionado para a tela de autorização (no servidor de recursos protegidos);
- Neste momento recebe uma chamada de retorno numa URL que foi especificada em 1, ou o membro digita um código PIN(autenticação *out-of-band*);

- Pedir Token de acesso;
- Fazer chamadas à API do servidor de recursos protegidos.

2.5.1 OAUTH 2.0

Em 2009 a comunidade começou a refazer o protocolo. A versão 2.0 do OAuth é um protocolo completamente novo. Sendo assim, ela não garante compatibilidade com as versões anteriores. Essa versão ganhou melhorias em 3 grandes áreas em que a versão 1.0 era limitada ou confusa. (HAMMER-LAHAV, 2010).

1. Autenticação e Assinaturas. A principal falha das tentativas de implementação do OAuth 1.0 foi causada pela complexidade da criptografia requerida na especificação. Mesmo depois de uma revisão, para a versão 1.0, OAuth continuou não necessário para uso no lado do cliente. Para usar a especificação, os desenvolvedores eram obrigados a buscar, instalar e configurar bibliotecas externas;

2. Experiência do Usuário e Opções Alternativas de Emissão de Tokens. OAuth inclui duas partes principais: obtenção do *token*, através da autorização de acesso pelo usuário e uso do *token* para obtenção de recursos protegidos. O método para obtenção do *token* é chamado *flow*. A versão 1.0 do OAuth possuía 3 *flows* (*web*, desktop e mobile). Entretanto, a especificação posteriormente para um único flow, que atendia os três tipos de clientes. Porém, com o aumento do número de sites que usam o OAuth, o *flow* disponível acabou se tornando cada vez mais limitado;

3. Performance em Escala. Com grandes empresas aderindo ao OAuth, a comunidade chegou à conclusão de que o protocolo não possuía um bom escalonamento. Ele necessitava de gerenciamento de estados através de diferentes etapas e requeria a emissão de credenciais de longa duração, que são menos seguras e mais difíceis de gerir. Com a versão 2.0 do OAuth foram criados 6 novos *flows*:

    - User-Agent Flow. Para clientes em navegadores Web;

- WebServer Flow. Para clientes que são parte de uma aplicação em servidor Web;
- Device Flow. Adequados para clientes de execução em dispositivos limitados;
- Username and Password Flow. Usado quando há um elevado grau de confiança entre o usuário e o cliente, para guardar suas credenciais;
- Client Credentials Flow. O Cliente usa suas credenciais para obter o token de acesso;
- Assertion Flow. O cliente apresenta uma afirmação como uma SAML4 para o servidor de autorização, em troca de um token de acesso.

A versão 2.0 do OAuth ainda provê uma opção para autenticação sem criptografia utilizando HTTPS.

As assinaturas foram simplificadas, não sendo mais necessária a análise, codificação e a ordenação de parâmetros. Nesta versão, é necessário apenas uma chave secreta. Também são implementados tokens de acesso de curta duração, que são renovados a partir de tokens de atualização. Isso permite ao cliente requisitar um novo token de acesso de forma transparente ao usuário. O OAuth 2.0 separa o papel de autenticação e autorização pelo servidor da requisição de recursos através da API. Isso facilita o uso de servidores distribuídos. (HAMMER-LAHAV, 2010)

2.5.2 Potenciais problemas

Uma vez que a delegação de acesso trata principalmente sobre manter a segurança dos usuários de provedores de serviços, nada mais natural que se ter uma atenção especial aos testes de segurança em um servidor que se comprometa em delegar acesso.

Este item trata exclusivamente dos potenciais problemas de segurança que podem surgir no uso do OAuth.

Phishing

Um aspecto de segurança muito comentado atualmente, principalmente quando trata-se de portais bancários, é o risco de *Phishing*. Segundo Symantec (2009), Phishing é basicamente um golpe online de falsificação. Seus criadores são falsários e ladrões de identidade especializados em tecnologia. Eles fazem com que os usuários entrem em sites que parecem ser autênticos, mas são cópias modificadas. Para tal finalidade, eles usam spams, websites maliciosos e e-mails para fazer com que as pessoas revelem informações sigilosas, como números de contas bancárias e de cartões de crédito. (SYMANTEC, 2009).

Quando se trata de delegação de acesso, deve-se dar uma atenção especial ao risco que *Phishing* pode causar. OAuth define que o usuário deve se autenticar, de algum modo, no *service provider*, para que não forneça suas credenciais ao *consumer*. Desta forma, os usuários não estariam a um passo de serem vítimas de *service provider* falsos, apenas esperando que os usuários digitem suas credenciais. (HUENIVERSE, 2007).

A melhor solução para ataques de Phishing é nunca digitar suas credenciais em páginas da Internet que não são acessadas manualmente no navegador (ou que não foi a partir dos seus favoritos).

Para que o Oauth funcione, o *consumer* solicita que o usuário conceda acesso para poder pegar os seus recursos protegidos. Note que a especificação OAuth não diz como o *service provider* deve fazer isso. Se o desenvolvedor assim desejar, pode obrigar o usuário a abrir um novo navegador no site do *service provider*, ir até a tela de *login* e autenticar-se manualmente. Desta forma, somente seria permitida a concessão de acesso via redirecionamento se o usuário já estivesse autenticado (HUENIVERSE, 2007).

Os objetivos do projeto OAuth é se manter o mais simples possível e fácil de implementar. Se estiver muito difícil, aplicações de terceiros usarão o método tradicional com nome de usuário e a senha, para em seguida, enviar os dados pessoais. A decisão de quão seguro serão os recursos, cabe ao desenvolvedor do *Service Provider*. (HUENIVERSE, 2007).

Falsas Requisições

Além do risco de *Phishing*, outra grande preocupação dos desenvolvedores é quanto ao envio de falsas requisições para o servidor OAuth. Um atacante que consiga utilizar um *sniffer* em requisições autênticas de *Consumers*, estaria apto a forjar requisições falsas e

enviá-las ao *Service Provider*, se não fosse pelo uso de *nonces* e *timestamps*. (HUENIVERSE, 2008).

Dessa forma, o reenvio de requisições, ou mesmo o envio de requisições falsas feitas com base em requisições autênticas, não será aprovado pelo servidor OAuth, uma vez que um mesmo *nonce* não será usado em duas requisições diferentes e o *timestamp* muda constantemente. A biblioteca desenvolvida já prove este nível básico de segurança aos desenvolvedores, desde que sejam utilizadas as páginas-base contidas no *namespace*. O OAuth bem desenvolvido em seus eventos de carregamento, já verificam estas e outras brechas de segurança. (HUENIVERSE, 2008).

2.6 Estudo de Caso

Para exemplificar o uso do OAuth, será utilizado o caso do Facebook para exemplificar a questão que envolve a autenticação e autorização.

O Facebook (FACEBOOK A, 2015) é a plataforma de rede social mais popular em 2015 com mais de 500 milhões de usuários. É frequentemente utilizado por sites de terceiros como um provedor de identidade além de fornecer conteúdo de informações a terceiros.

Ao vincular seu cadastro ou identidade no Facebook, o usuário não precisa registar em cada site individualmente e evita a necessidade de criar mais um identificador de usuário e senha. O aplicativo de um terceiro terá que obter um *token* de acesso, que fornece acesso temporário, seguro aos recursos da API do Facebook, com esse *token* é possível obter as informações do usuário sem precisar preencher um cadastro por exemplo.

Para a autenticação e autorização de sites de terceiros e aplicações, o Facebook tem suporte a versão mais recente o OAuth 2.0. Com o protocolo, além de simplificar a autenticação para os desenvolvedores, ele inclui suporte para vários fluxos para servidores *web* e aplicativos móveis com uma interface mais simples do que a do OAuth 1.0. Além disso, permite autenticação em dispositivos não muito práticos ou possíveis para os usuários, como por exemplo, equipamentos de automação residencial, sensores e atuadores.

Para acessar as informações, o Facebook disponibiliza a API Graph. A API Graph é a principal forma de obter dados dentro e fora da plataforma do Facebook. É uma API baseada em HTTP (REST) de baixo nível que permite consultar dados, gerenciar anúncios, envio de fotos e uma variedade de outras tarefas que um aplicativo pode precisar fazer. (FACEBOOK B, 2015).

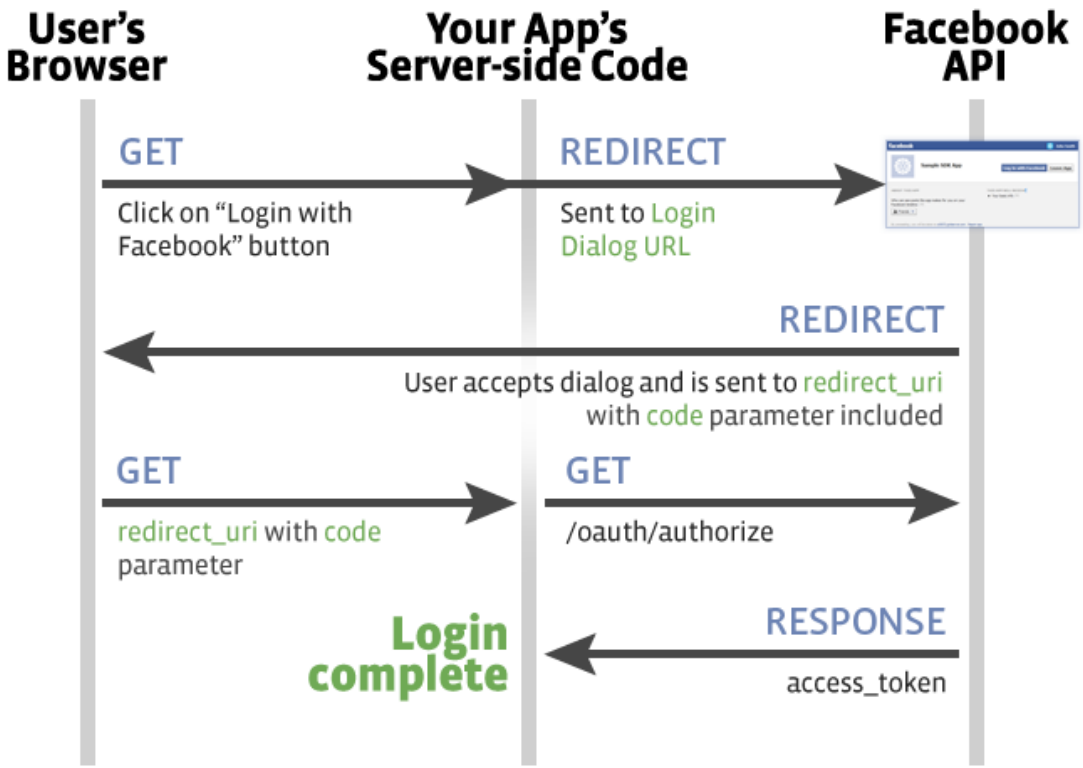
Todos os objetos como usuários, fotos, eventos e páginas do Facebook são um identificador único pelo sistema, o nome do usuário ou um ID único.

Por exemplo, para obter informação sobre Eduardo Arakaki, deve utilizar a seguinte URL https://graph.facebook.com/arakaki?access_token=xxxxxxxxxxx.

Um *token* de acesso é uma string que identifica um usuário, aplicativo ou página. O *token* inclui informações sobre quando o *token* expirará e qual aplicativo gerou o *token*. Por causa de verificações de privacidade, a maioria das chamadas de API no Facebook precisa incluir um *token* de acesso caso o recurso seja privado.

Para exemplificar melhor, a figura 5 demonstra o fluxo do aplicativo junto com o serviço.

Figura 5. Fluxo cliente/serviço.



Fonte. FacileLogin, Why OAuth it self is not an authentication framework?, 2013.

1. O usuário acessa aplicativo cliente.

2. O aplicativo cliente fornece lista de vários provedores de identidade com suporte ao OAuth.

3. Usuário escolhe o provedor de identidade e p aplicativo cliente redirecionar a solicitação de autenticação para provedor de identidade escolhido fornecendo sua credencial junto.

4. Provedor de identidade valida a credencial do usuário, e redireciona a solicitação para o cliente incluindo o código de autenticação.

5. Usuário acessa a URL do cliente redirecionado com código de autenticação

6. Cliente envia este código de autenticação junto com seu ID do cliente e cliente secreto que ele obteve durante o registro com o provedor de identidade inicialmente.

7. Provedor de identidade valida código de autenticação, ID Cliente e cliente secreto e retorna um *token* de acesso (chave de *valet*), que é um símbolo de vida longa.

8. Isso conclui o processo de autenticação e os usuários do aplicativo está conectado.

2.7 Considerações Finais

Para o desenvolvimento da API OAuth, foi necessário o estudo para o desenvolvimento da ferramenta para uso mobile para integração com serviços *web*.

Foi possível notar que a grande parte dos serviços que disponibilizam recursos de informações em 2015, utilizam a versão mais recente como ilustra a Tabela 1.

Tabela 1 - Tabela de serviços com a versão OAuth

Serviços 2015	Versão OAuth
DropBox	2.0
Facebook	2.0
Google	2.0
Twitter	1.0a e 2.0
Linkedin	2.0
Vimeo	2.0
Microsoft	2.0

3 IMPLEMENTAÇÃO E COMPARACAO ENTRE OS PROTOCOLOS, OAUTH 1 E OAUTH 2

Muitas aplicações na Internet vêm se tornando uma grande fonte de recursos como fotos, textos, dados, entre outras.

Com a tão esperada *web* 2.0, muitos sites passaram a fornecer APIs que podem ser usadas por terceiros para agregar valor ao sistema. Serviços como Twitter e Facebook devem grande parte de seu sucesso pelo fato de ter essa possibilidade de extensão, a disseminação de acessos a conteúdos como redes sociais por algum dispositivo móvel, o uso de API de integração será essencial.

Atualmente, essas APIs são fornecidas principalmente por serviços que fornecem seu próprio código. O programador que busca alternativas para o desenvolvimento de aplicações mobile com a integração de serviços, está praticamente preso a elas. Considerando que o conceito de OAuth tem pouco tempo de vida, existem pouca documentação e boas práticas a serem utilizadas e boa parte das existentes são em inglês.

Nesses sites de relacionamento, normalmente a API é acessada "em nome" de um usuário, ou seja, é necessário ter um usuário autenticado para obter os dados. Agora vem a questão: E para quem gosta de compartilhar sua senha com os outros, mesmo que o outro seja uma aplicação? Ao alterar sua senha, terá que ir em cada aplicativo e reconfigurar, o que não é uma boa prática para o usuário. (MAFRA, 2010).

O OAuth trabalha para tratar desses problemas fornecendo um serviço de autorização separado do proprietário do recurso. Serviços de autenticação confiáveis podem ser usados para fornecer "*tokens*", concedendo acesso aos recursos. Um proprietário do recurso do OAuth concede a um cliente, o acesso aos seus recursos por meio da autenticação e o serviço de autorização que emite *tokens* de acesso específicos de serviço para terceiros.

Inspirados em protocolos de delegação de acesso proprietários e com a necessidade de um protocolo que fosse extensível, simples e seguro para uso em suas aplicações, um grupo de desenvolvedores norte-americanos (incluindo funcionários do Google, Yahoo! e AOL) iniciou, em novembro de 2006, o rascunho do que viria a se tornar o protocolo OAuth. O protocolo OAuth propõe uma forma padronizada para desenvolvedores oferecerem seus serviços na web, sem forçar seus usuários a exporem suas credenciais. (HUENIVERSE, 2007).

Com a especificação do protocolo em mãos, o desenvolvedor monta a arquitetura que

julgar conveniente, bastando conhecimento de chamadas e cabeçalhos HTTP, assim como o domínio de alguma linguagem de programação. É possível, por exemplo, definir o tipo de criptografia que será utilizada nos *tokens* e URLs, parâmetros adicionais, escopo de acesso que será concedido a aplicação, etc. Um desenvolvedor que esteja com a especificação em mãos, está apenas limitado a sua criatividade e ao seu conhecimento.

Metodologia

O projeto pode ser divido em cinco principais fases que abordam: pesquisa e análise do protocolo, pesquisa e desenvolvimento da API, pesquisa e desenvolvimento do serviço que a aplicação irá consumir para teste, desenvolvimento da aplicação mobile e testes finais.

A) Pesquisa e análise sobre segurança, autenticação e autorização

Pesquisar e documentar todas informações obtidas para uso futuro no desenvolvimento da API OAuth.

B) Pesquisa e desenvolvimento da API

O desenvolvimento da API baseado no protocolo OAuth é uma estrutura baseada em autenticação e autorização, permitindo que o proprietário de um recurso conceda permissão para o acesso de seus recursos sem compartilhar suas credenciais e forneça acesso limitado aos recursos armazenados por serviços baseados na web acessados por HTTP. Isso não deve ser confundido com serviços da web e arquitetura orientada a serviços (SOA).

Enquanto essas arquiteturas existem em uma ampla lista de implementações, o OAuth prioriza mais a infraestrutura da Web 2.0 emergente e a popularidade das APIs que existem para que possa fornecer acesso personalizado aos aplicativos.

Por exemplo, o Twitter e o Facebook fornecem APIs que ampliam seus aplicativos fornecendo recursos de compartilhamento de conteúdo. Cada uma dessas integrações, exige uma atenção focada em todos os aspectos da segurança, considerando que, todos os acessos não são confiáveis.

Em implementações estabelecidas, os recursos são protegidos e acessados ao se fornecer credenciais que podem ser autenticadas e, um servidor de recurso, pode usá-las para autorizar o acesso ao recurso.

Se o proprietário de um recurso quiser liberar acesso a um terceiro, será preciso fornecer credenciais, autenticá-las, e o acesso deverá ser autorizado pelo serviço de autorização que protege os recursos.

Essas arquiteturas estabelecidas apresentam diversos problemas, incluindo a fraqueza herdada da autenticação de senha. Senhas são vulneráveis. Uma pessoa que esteja na sala ou talvez olhando pelo canto do olho, poderia obter suas credenciais e ter acesso a seus recursos. Provavelmente, o proprietário do recurso precisa compartilhar sua senha com a entidade que precisa do acesso e quando uma senha é alterada, a entidade não tem mais acesso ao recurso.

Uma limitação mais fundamental é que a autenticação e a autorização são controladas pelo proprietário do recurso e, portanto, limitam o compartilhamento das informações de autenticação.

As organizações de TI desempenharam um papel fundamental nessas decisões de segurança que limitam a flexibilidade. O OAuth trabalha para tratar desses problemas, fornecendo um serviço de autorização separado do proprietário do recurso. Serviços de autenticação confiáveis podem ser usados para fornecer *tokens*, concedendo acesso aos recursos.

Um proprietário do recurso do OAuth concede a um cliente acesso a seus recursos por meio da autenticação e por meio da autorização, que emitem *tokens* de acesso específicos de serviço para terceiros.

Em um caso de uso típico do OAuth, você (proprietário do recurso) pode fornecer acesso a um serviço de impressão (terceiro, o cliente) às suas fotos (recurso), utilizando o serviço de autenticação e fornecendo acesso limitado (*token* de acesso);

C) Pesquisa e desenvolvimento do serviço que a aplicação irá consumir para teste

O Serviço *web* é um tipo de aplicação para a web, isto é, uma aplicação tipicamente oferecida através de HTTP (*Hyper Text Transport Protocol*). Também é uma aplicação distribuída, cujos componentes podem ser aplicados e executados em dispositivos distintos. Por exemplo, o Web Service de escolha de ações pode consistir em diversos componentes de código, cada um armazenado em um servidor de grau de negócio separado, podendo ser consumido em PCs, handhelds e outros dispositivos;

REST: Um estilo de arquitetura é um conjunto de restrições que podem ser aplicadas quando se cria algo. É algo que descreve os recursos, podendo ser usados para orientar a

implementação de um sistema de software e podendo ser utilizados para compilar softwares em que os clientes podem fazer solicitações de serviço. (INFOQ, 2008);

JSON: *JavaScript Object Notation* é um modelo para armazenamento e transmissão de informações no formato texto. Também tem sido bastante utilizado por aplicações Web, devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o *parsing* dessas informações. Isto explica o fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados. (DEVMEDIA, 2013);

D) Desenvolvimento da aplicação mobile.

Criação de uma aplicação Android para consumir o serviço criado na etapa anterior com o uso da API OAuth Mobile, desenvolvida para futuros testes e documentação;

E) Validação e testes.

Por fim, testar e analisar a API OAuth Mobile.

Implementação do serviço OAuth 1.0 e 2.0.

Foi desenvolvido um serviço OAuth básico que contempla as funcionalidades principais de qualquer sistema de delegação de acesso.

Contém todos os elementos necessários para efetuar a delegação de acesso segundo a especificação do protocolo. Basicamente, foram criadas as páginas essenciais ao *workflow* de delegação de acesso segundo a especificação e nada mais. Não foi desenvolvida interface alguma para fornecer autenticação ou não ao *consumer*.

Desenvolvimento do OAuth 1.0 e 2.0

Este protótipo de serviço foi desenvolvido na linguagem PHP junto com alguns módulos de terceiros, assim é completamente configurável e expansível, podendo inclusive ser utilizado como base para serviços reais.

O servidor atende a requisições HTTP provenientes de clientes escritos em qualquer linguagem. As etapas de desenvolvimento seguiram à risca a especificação do protocolo OAuth, uma vez que, como objetivo central do trabalho, o servidor deveria ser um exemplo 100% funcional de um serviço com o OAuth básico.

O desenvolvimento do servidor atendeu os princípios do *workflow* do protocolo OAuth, como a página de requisição de *RequestToken*, a página de autorização do usuário, etc. Todas devidamente estendidas a partir das páginas fornecidas pela biblioteca de classes.

Tecnologias utilizadas:

- Linguagem de programação PHP;
- Composer para controle de dependências.
- Bibliotecas open-source OAuth.

Testando o Serviço OAuth 1.0 e 2.0

O servidor é quem recebe e processa as requisições dos clientes segundo a especificação do protocolo OAuth, portanto, os testes realizados com o servidor mostraram que o uso de bibliotecas ajuda a controlar todas as requisições além de deixar o desenvolvimento mais rápido.

Nenhum outro teste específico do servidor foi realizado, como testes de invasão, muito embora existissem opções para tal na *Internet*, em sites que se propõe a testar seu servidor em todos os requisitos.

Apesar de não realizar testes específicos, foi possível constatar através dos testes básicos do protocolo, que o servidor atende as necessidades de um servidor de delegação de acesso, seguindo à risca a especificação 1.0 e 2.0 do protocolo OAuth.

Para o uso do servidor alguns modelos simples serão necessários para lidar com a autorização da solicitação de *tokens* e para o tratamento de pedidos de *tokens* de acesso.

Controladores:

1. oauth_register.php: Cada consumidor usa uma combinação de uma chave de consumidor com um segredo do consumidor e um *token* com um *token secret* para assinar os seus pedidos. Um usuário deve primeiro obter a chave de consumidor com um segredo do consumidor antes de pode começar o pedido de acesso ao servidor.
2. request_token.php: Depois do cliente obter a chave secreta do consumidor, ele pode solicitar um *token* para a obtenção de autorização do usuário.
3. authorize.php: Este controlador pergunta ao usuário se ele permite que o consumidor pode acessar sua conta. Quando permitido, então o consumidor pode trocar a sua marca pedido de um token de acesso.
4. access_token.php: Então a troca do *token* de solicitação de autorização por um *token* de acesso é feita. O *token* de acesso pode ser usado para assinar solicitações.

Cliente OAuth Móvel

A partir de um levantamento detalhado de requisitos e de uma análise da própria especificação do protocolo OAuth, foi desenvolvida uma API mobile de código aberto com uma documentação precisa, seguindo padrões de codificação que priorizam a legibilidade do código, eficiência e extensibilidade. A esta API foi dada o nome de API OAuth Mobile.

O Desenvolvimento ocorreu de forma iterativa e incremental, usando um pouco da metodologia conhecida como Processo Unificado (*Unified Process*, uma vertente do RUP) e experiências próprias com desenvolvimento JAVA. Assim, teve o início do desenvolvimento dos ambientes, em um primeiro momento, um cliente OAuth, e mais tarde o servidor. Durante pesquisas realizadas, foi utilizado a biblioteca scribe (indicada pelo próprio OAuth.org).

O scribe é uma biblioteca *open source* destinada ao uso de todas as versões da API OAuth. Ela trabalha com os serviços mais atuais como Google, Yahoo!, LinkdIn, Twitter, Foursquare e outras APIs.

O projeto foi refatorado diversas vezes antes de atingir a maturidade atual, sendo adicionado classes e métodos diversos (estudadas em pesquisas), sempre visando a eficiência sem perder a simplicidade. Foi dado uma atenção especial em torná-lo genérico e facilmente confiável como proposto no trabalho, para que fosse de serventia a outros projetos. Dessa

forma, arquivos de configuração, facilmente editáveis, armazenam informações como URLs, credenciais, métodos de assinatura e métodos HTTP, tanto dos clientes quanto dos servidores.

Na parte de recursos protegidos dos usuários, não foi importante implementar muita coisa, pois o foco não foi informações do usuário. A resposta retornada pelo servidor após um acesso, retorna as informações contidas no banco de dados, uma vez que tal resposta pode conter dados, texto, imagens, etc., variando da necessidade de cada servidor e/ou cliente. Dessa forma, enquanto a manipulação e interpretação de requisições HTTP durante as etapas de autorização e autenticação são transparentes ao usuário da API, o acesso aos recursos protegidos exige são mais complexos.

Ainda que a API tenha sido desenvolvida utilizando a linguagem JAVA junto com a biblioteca scribe, uma vez que seu código é aberto, padronizando e documentado, não é difícil que os desenvolvedores de outras linguagens a convertam para a linguagem de sua preferência ou mesmo customizem-na para se adequar às suas necessidades. Vale ressaltar que o objetivo central do trabalho era prover um mecanismo que além de funcionar por si próprio, servisse de exemplo para implementações futuras.

A decisão de desenvolver uma API foi puramente estratégica e intencional, uma vez que dessa forma, ela se tornou a mais genérica possível, permitindo seu uso e estudo independente do servidor e cliente utilizados. Para os desenvolvedores que desejem utilizar a API em suas aplicações mobile, basta adicionar o .JAR em sua aplicação mobile.

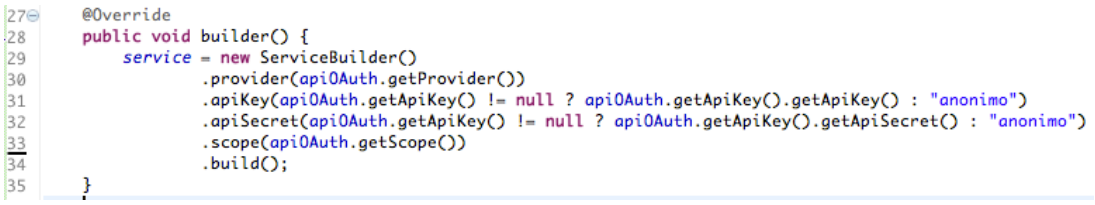
Teste cliente OAuth Móvel

Foi feito diversos testes de acesso a serviços *web* utilizando somente a API. Uma vez que somente o cliente utilize a API e seus componentes, todos os testes realizados com as aplicações de exemplo, acabaram consequentemente testando as capacidades da biblioteca em si. Ainda assim, os provedores de assinatura de requisições PLAINTEXT e HTMAC-SHA1 foram utilizados nos testes, embora o método RSA-SHA1 também tenha sido implementado.

Fluxo utilizando a API Mobile desenvolvida com o serviço do Twitter:

1. Na figura 6, demonstra a criação do objeto OAuthService passando os devidos valores para o fluxo do OAuth.

Figura 6. Criação do objeto OAuthService



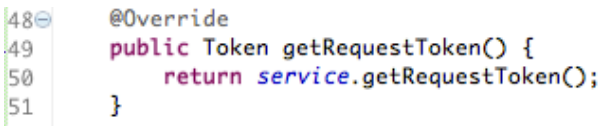
```
@Override
public void builder() {
    service = new ServiceBuilder()
            .provider(apiOAuth.getProvider())
            .apiKey(apiOAuth.getApiKey() != null ? apiOAuth.getApiKey().getApiKey() : "anonimo")
            .apiSecret(apiOAuth.getApiKey() != null ? apiOAuth.getApiKey().getApiSecret() : "anonimo")
            .scope(apiOAuth.getScope())
            .build();
}
```

Para a criação do objeto do serviço OAuth, foi necessário passar os valores para que o fluxo ocorra:

- Provider: Provedor dos recursos. Exemplo: Twitter.class.
- API Key: ID único gerado para a aplicação.
- API Secret: Identificador secreto gerado pela API.
- Scope: Equivalente ao parâmetro âmbito que é descrita na especificação OAuth 2.0.

2. Após a construção do objeto, o método getRequestToken() é chamado para obter o token de solicitação como mostra a figura 7.

Figura 7. Método para a solicitação do token.



```
@Override
public Token getRequestToken() {
    return service.getRequestToken();
}
```

Para exemplificar, a figura 8 mostra o cabeçalho de autorização do Twitter.

Figura 8. Cabeçalho de autorização.

```
POST /oauth/request_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization:
        OAuth oauth_callback="http%3A%2F%2Flocalhost%2Fsign-in-with
              oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
              oauth_nonce="ea9ec8429b68d6b77cd5600adbbb0456",
              oauth_signature="F1Li3tvehgcraF8DMJ7OyxO4w9Y%3D",
              oauth_signature_method="HMAC-SHA1",
              oauth_timestamp="1318467427",
              oauth_version="1.0"
```

Fonte. Twitter, Implementing Sign in with Twitter Overview, 2011.

Sua aplicação deve examinar o status HTTP de resposta. Qualquer valor diferente de 200 indica uma falha.

3. Assim que o pedido é envido, ocorre a validação do *token* de solicitação para verificar se esse aplicativo tem permissão como é mostrado na figura 9.

Figura 9. Método de autorização.

```
@Override
public Verifier authorizeApi() {
    String verification = null;
    if (apiOAuth.getAuthorizeUrl() != null && !apiOAuth.getAuthorizeUrl().trim().equals("")) {
        verification = getAuthorization(apiOAuth.getAuthorizeUrl(), getRequestToken());

    } else {
        verification = service.getAuthorizationUrl(getRequestToken());
    }
    return new Verifier(verification);
}
```

O corpo da resposta conterá o oauth_token, oauth_token_secret, e oauth_callback_confirmed como parâmetros como é apresentado na figura 10. Sua aplicação deve verificar se oauth_callback_confirmed é verdadeiro e armazenar os outros dois valores para as próximas etapas.

Figura 10. Resposta da solicitação do token.

```
HTTP/1.1 200 OK
Date: Thu, 13 Oct 2011 00:57:06 GMT
Status: 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 146
Pragma: no-cache
Expires: Tue, 31 Mar 1981 05:00:00 GMT
Cache-Control: no-cache, no-store, must-revalidate, pre-check=0, pc
Vary: Accept-Encoding
Server: tfe

oauth_token=NPcudxy0yU5T3tBzho7iCotZ3cnetKwcTIRlX0iwRl0&
oauth_token_secret=veNRnAWe6inFuo8o2u8SLLZLjolYDmDP7SzL0YfYI&
oauth_callback_confirmed=true
```

Fonte. Twitter, Implementing Sign in with Twitter Overview, 2011.

Após a autenticação o call_back é um redirecionamento para uma url como /oauth/authenticate?oauth_token=jJSAOjdOIJSdadUMAMdls.

4. Assim que a resposta da autorização é confirmada como positiva, o método é chamado para obter o *token* de acesso, como mostra a figura 11.

Figura 11. Método para obter o *token* de acesso.

```
54⊖    @Override
55     public Token getAcessToken() {
56         return service.getAccessToken(getRequestToken(), authorizeApi());
57     }
```

Para tornar o *token* de solicitação em um *token* de acesso utilizável, o aplicativo faz uma solicitação para a url do OAuth POST access_token, contendo o oauth_verifier valor obtido na etapa 2. O *token* também é transmitido no cabeçalho e também é acrescentado pelo processo de assinatura como mostra a figura 12.

Figura 12. Cabeçalho do token de acesso.

```
POST /oauth/access_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization: OAuth oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
                     oauth_nonce="a9900fe68e2573b27a37f10fbad6a755"
                     oauth_signature="39cipBtIOHEEnybAR4sATQTpl2I%3
                     oauth_signature_method="HMAC-SHA1",
                     oauth_timestamp="1318467427",
                     oauth_token="NPcudxy0yU5T3tBzho7iCotZ3cnetKwcT
                     oauth_version="1.0"
Content-Length: 57
Content-Type: application/x-www-form-urlencoded

oauth_verifier=uw7NjWHT6OJ1MpJOXsHfNxoAhPKpgI8BlYDhxEjIBY
```

Fonte. Twitter, Implementing Sign in with Twitter Overview, 2011.

E por fim, a resposta é retornada como mostra a figura 13.

Figura 13. Resposta do pedido de acesso.

```
HTTP/1.1 200 OK
Date: Thu, 13 Oct 2011 00:57:08 GMT
Status: 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 157
Pragma: no-cache
Expires: Tue, 31 Mar 1981 05:00:00 GMT
Cache-Control: no-cache, no-store, must-revalidate, pre-check=0, pc
Vary: Accept-Encoding
Server: tfe

oauth_token=7588892-kagSNqWge8gB1WwE3plnFsJHAZVfxWD7Vb57p0b4&
oauth_token_secret=PbKfYqSryyeKDWz4ebtY3o5ogNLG11WJuZBc9fQrQo
```

Fonte. Twitter, Implementing Sign in with Twitter Overview, 2011.

5. Finalmente a assinatura é montada e o pedido de acesso ao recurso protegido é chamado com a resposta do conteúdo como mostra a figura 14.

Figura 14. Método do pedido ao recurso protegido.

```
59    @Override
60    public Response requestOAuth() {
61        OAuthRequest request = new OAuthRequest(Verb.GET, apiOAuth.getProtectedResourceUrl());
62        service.signRequest(getAcessToken(), request);
63        return request.send();
64    }
```

Exemplo de Cliente Mobile

Para exemplificar o uso da API OAuth Mobile e do desenvolvimento de um cliente OAuth e até mesmo para testar o servidor OAuth criado, foi desenvolvido um cliente mobile básico, que assume o papel de Consumer, negociando *tokens* junto ao servidor e permissão junto ao usuário, com a finalidade de obter acesso aos recursos do usuário. O protótipo de cliente Mobile é 100% configurável, configuração esta que deve ser feita segundo a documentação do servidor OAuth a qual se deseja conectar, uma vez que ela inclui URL para obtenção de *Request Token*, *Access Token*, URL de autenticação de usuário, etc.

Contemplando as funcionalidades de um *Consumer* OAuth, a estrutura do cliente mobile *consumer mobile*, implementa uma interface presente na biblioteca API *Mobile*, que contém a assinatura de todos os métodos e propriedades obrigatórios a todos os clientes

mobile, tornando extremamente fácil sua implementação.

Enquanto a classe denominada *Consumer Mobile* cuida de toda a lógica essencial a um *Consumer*, a Classe MainActivity.java possui uma interface simples que permite realizar o gerenciamento de *tokens* e obtenção de palavras de respostas.

O cliente foi desenvolvido utilizando a mesma linguagem da API Mobile. Contempla todos os requisitos essenciais para negociar *tokens* com servidores OAuth e obtém acesso delegado por usuários a recursos protegidos. A construção de um *consumer* válido e operacional utilizando a API *Mobile*, exigiu boa parte do tempo de desenvolvimento e a releitura constante da especificação OAuth, embora isto pareça ilógico, foi desenvolvido antes mesmo do servidor OAuth.

Em linhas gerais, o cliente chama a API Mobile *Consumer* e invoca seus métodos que retornam o *Request Token*, retornam o *Acesses Token*, etc. Toda a lógica para os tramites necessários à delegação de acesso estão encapsulados na própria API, facilitando enormemente o trabalho do desenvolvedor que deseje utilizar OAuth.

Todo e qualquer tramite de delegação de acesso ficou sob responsabilidade da API, tornando a delegação de acesso algo transparente ao desenvolvedor, exceto durante a manipulação do *Protected Resources*. Alvo dos *Consumers* e protegido pelo *Service Provider*, os *Protected Resources* podem ser qualquer coisa de posse de um usuário dentro de um *Service Provider*. As tarefas dentro de uma agenda online são os *Protectes Resources* daqueles usuários, enquanto que as fotos de outro usuário em um serviço de álbuns online também são. Assim, o cliente retorna o HTTP Response do servidor dentro de uma propriedade da classe *Protected Resource*.

Tecnologias utilizadas:

- Sistema operacional mobile Android;
- Linguagem de programação Java;
- IDEs: Eclipse e Android Studio;
- Biblioteca open-source scribe;
- API Mobile desenvolvida para integrar o cliente com o serviço.

Considerações Finais

O cliente foi testado utilizando os métodos de assinatura PlainText e HTMAC-SHA1. Também foi satisfatório nos testes realizados, permitindo obter acesso delegado aos recursos protegidos em um servidor OAuth de terceiros. O servidor em questão foi totalmente importante para a compreensão do OAuth em servidores, e também negociava *tokens* para acesso à uma página que retorna parâmetros, como se fossem os recursos protegidos.

Terminada esta primeira etapa e após o desenvolvimento do servidor e API, o cliente foi novamente testado, dessa vez sobre o ambiente ao qual ele se destinava. Uma vez que funcionou sobre um servidor de terceiro, foi provado que ele realmente atendia às exigências da especificação do protocolo OAuth, portanto, estava apto a testar o servidor desenvolvido e afirmar se ele funciona ou não sobre OAuth. Mais uma vez os resultados obtidos foram satisfatórios em testes com credenciais válidas e inválidas, com requisições forjadas e autênticas, e assim por diante.

4 RESULTADOS

Este projeto teve como resultado a comparação entre as versões do protocolo, a pesquisa sobre o OAuth e o desenvolvimento nas camadas cliente, API e servidor para validar e testar o protocolo.

Cada ambiente foi desenvolvido utilizando diferentes linguagens de programação e frameworks que ajudaram no desenvolvimento dos ambientes deixando o código menos complexo. Por fim, foi feito vários testes do uso do protocolo para gerar possíveis comparações entre suas versões além do nível de dificuldade de implementar.

Com isso, foi possível elaborar um comparativo entre as versões do OAuth, além disso, foi feito alguns testes de cargas em alguns serviços OAuth conhecidos.

4.1 Comparativo OAuth 1.0 vs 2.0

Após as implementações e os testes podemos afirmar referente a OAuth 1.0.

Criptografia Complexa: A forma como o Auth 1.0 é implementado torna a criação de scripts complexa devido à conversação envolvida na negociação do *token*.

Suporte de Aplicações "não *web*": É necessário encaminhar o utilizador para abrir o browser para o serviço desejado, autenticar com o serviço, e copiar o *token* do serviço de volta para a aplicação.

Período de vida longa dos *tokens*: Os *tokens* gerados por esta versão têm um período de vida muito longa e poderão ser usados por um longo período de tempo, assim possibilitando falhas e ataques.

***Parsing* complicado:** Uma vez que esta versão do protocolo pode ser implementada usando HTTP não criptografado, é necessário que a aplicação use hash Token HMCA e cadeias de pedidos. Para o protocolo funcionar é igualmente necessário fazer um conjunto especial de ações (parsing) que apresentam uma complexidade elevada.

Além disso, o Auth 1.0 requer que os pontos terminais protegidos dos recursos tenham acesso às credenciais do cliente, a fim de validar o pedido. Isto impossibilita a

arquitetura típica da maioria dos grandes fornecedores em que um servidor centralizado de autorização é usado para a emissão de credenciais, e um servidor separado é usado para chamadas de API.

O OAuth 1.0 requer o uso dos dois conjuntos de credenciais: as credenciais do cliente e as credenciais *token* o que faz com que essa separação seja muito complexa.

Compatibilidade: A implementação foi incompatível com o 1.0;

Segurança e *Parsing* simplificado: O SSL passou a ser necessário em todas as comunicações quando é necessário para gerar o *token*. As assinaturas não são necessárias para a chamada à API a partir do momento que o *token* é gerado. Apenas um *token* de segurança, e a assinatura não é necessária. O suporte à assinatura foi significativamente simplificado para remover a necessidade de tratamento especial, codificação e classificação dos parâmetros.

Autorização: A separação de proprietário de recurso e servidor de autorização é clara.

Fluxo: Além, de implementar todas as características básicas da 1.0, existem seis novos fluxos

A. O cliente solicita autorização ao proprietário do recurso.
B. O cliente recebe uma concessão de autorização, que representa a autorização fornecida pelo proprietário do recurso.
C. O cliente solicita um *token* de acesso autenticando-se junto do servidor com a concessão de autorização.
D. O servidor de autorização autentica o cliente e valida a concessão de autorização, e se esta for válida emite o *token* de acesso.
E. O cliente pede o recurso protegido ao servidor de recursos protegidos e autentica-se com o Token de acesso.
F. O servidor de recursos valida o *token* de acesso e se for valido, serve o pedido.

Período de vida longa dos *tokens:* Tokens de curta duração com autorizações de vida longa. Em vez de emitir um *token* de longa duração (normalmente por um ano ou eterno), o servidor pode emitir um *token* de acesso de curta duração e uma vida longa de *token* de atualização.

Outros dois tipos de tokens:

1. MAC – é equivalente ao esquema de *token* no OAuth 1.0
2. SAML – usa afirmações SAML 2.0 em cada pedido como uma forma de estabelecer a identidade do cliente.

Separação de Papeis: O OAuth 2.0 separa o papel do servidor de autorização responsável pela obtenção da autorização do utilizador e a emissão de *tokens* do servidor de recursos que trata das chamadas à API. (HAMMER, 2010). Assim, e em contraponto com o OAuth 1.0 existe claramente quatro papéis:

1. Proprietário do recurso: utilizador final;
2. Cliente: aquele que precisa de acesso aos recursos, geralmente um servidor;
3. Servidor de recursos protegidos: o servidor que abriga os recursos do proprietário;
4. Servidor de Autorização: o servidor que emite os *tokens* para o Cliente.

4.2 Teste em outros serviços OAuth

Os testes foram realizados com a ferramenta APImetrics disponível online para testes de APIs e o serviço LinkedIn foi utilizado para os devidos testes de requisições para conhecimento da ferramenta, além dos testes já fornecidos pela ferramenta com outros serviços.

Como descrito nos capítulos anteriores, o processo do OAuth requer até 3 chamadas API separadas. As tabelas 2 e 3 mostram algumas das variações nos servidores OAuth utilizando dados reais do serviço LinkedIn e foi testado de forma padronizada, do mais rápido para o mais lento.

Para o teste de velocidade foram utilizadas as seguintes métricas:

- *Pass* %: Porcentagem de requisições que passaram nos testes.
- *Run Location*: Serviço onde os testes de requisições do OAuth foram feitas.
- *Average Pass Latency(ms):* Tempo médio em milissegundos das requisições

que foram aceitas.

- *Max Pass Latency(ms):* Tempo máximo das requisições aceitas realizadas.
- *Total of Calls:* Total de chamadas realizadas durante os testes.
- Avg Server Error Latency(ms): Tempo médio de erro na requisição ao servidor.

OAuth 1.0

Tabela 2 - Testes realizados utilizando OAuth 1.0

Name	Run Location	Top 5% Breakdown	Pass %		Average Pass Latency (ms)		Max Pass Latency (ms)	Avg Server Error Latency (ms)	Total # of Calls	Options
LinkedIn OAuth 1.0 User Profile Fetch TCC	AWS South-America East (Sao Paulo)		100%	⟷	425		600	-	112	☰

OAuth 2.0

Tabela 3 - Testes realizados utilizando OAuth 2.0

Name	Run Location	Top 5% Breakdown	Pass %		Average Pass Latency (ms)		Max Pass Latency (ms)	Avg Server Error Latency (ms)	Total # of Calls	Options
LinkedIn OAuth 2.0 User Profile Fetch TCC	AWS South-America East (Sao Paulo)		100%	⟷	467		740	-	109	☰

Na tabela 4, os testes foram todos realizados no período de uma semana sobre os mesmos servidores de teste que compartilham de uma ligação comum de internet. Os testes foram conduzidos concorrentemente.

Tabela 4 - Testes realizados com os serviços e suas versões Oauth

Name	# Tests	Max Latency	Avg Latency	Pass %	Breakdown
Microsoft Live OAuth 2.0 dialog	672	392	194	100.0%	
Tripit OAuth 1.0 request token	672	527	197	99.9%	
LinkedIn OAuth 2.0 dialog	672	544	264	99.9%	
Tumblr OAuth 1.0 request token	672	591	188	100.0%	
Facebook OAuth 2.0 dialog	672	1005	226	100.0%	
Tumblr OAuth 1.0 dialog	672	1089	609	99.9%	
Yahoo OAuth 1.0 request token	672	1195	207	99.9%	
Twitter OAuth 1.0 request token	672	1456	184	100.0%	
Salesforce OAuth 2.0 dialog	672	2823	390	100.0%	
LinkedIn OAuth 2.0 User Profile Fetch	672	3324	289	100.0%	
Instagram OAuth 2.0 dialog	672	9199	661	99.9%	
Intel Cloud Services OAuth 2.0 dialog	672	10335	1242	100.0%	
Foursquare OAuth 2.0 dialog	672	30164	780	98.2%	
Yahoo OAuth 1.0 dialog	671	41598	671	100.0%	
Google OAuth 2.0 dialog	672	59886	195	99.9%	
Github OAuth 2.0 dialog	672	385803	1478	99.1%	

Fonte. API Metrics, OAuth – The Standard That Isn't, 2014.

As requisições são fornecidas pela própria ferramenta para os mais diversos serviços que utilizam o Oauth. No testes feitos, o tempo de cada requisicão foi no intervalo de 10 minutos e durante três dias. A primeira informação que saiu da média das requisições é a variação na média latência, do mais rápido para o mais lento. Em segundo lugar, as latências máxima pode ter um impacto sobre os tempos de espera e confiabilidade geral para os usuários.

Os serviços com taxas de aprovação de requisições e a variação nas escalas de tempo, utilizando o teste no diálogo Yahoo OAuth, o serviço tinha uma taxa de aprovação de 100%(como mostra na coluna pass% da tabela) no fornecimento aos usuários a página de login, mas durante um teste levou mais de 45 segundos para efetuar a requisicão e a resposta.

4.3 Problemas encontrados e limitações

Apesar de todo o ambiente de testes estar completo, um fator que limitou muito o tempo de desenvolvimento foi a pouca documentação sobre métodos de implementar o OAuth tanto no cliente como no servidor. Basicamente todos os serviços que fornecem API para terceiros consumir, tem sua própria documentação e o que tem na comunidade são quase que cópias destas documentações.

Os serviços fornecem os conceitos básicos do protocolo em código sem nenhuma API completa para o programador usar os recursos desse serviço. Então, para terceiros a falta de bibliotecas com métodos já prontos para uso do protocolo com diferentes provedores interfere no tempo de desenvolvimento. Para qualquer comunicação diferente é necessário implementar a especificação fornecida pelo serviço, assim os aplicativos normalmente são fieis a um deles.

4.4 Lições aprendidas

A pesquisa junto com as comparações das versões do protocolo, junto com o desenvolvimento do ambiente de testes foram fundamentais para a compreensão do OAuth.

Pesquisa sobre métodos de autenticação e autorização:

Entender primeiramente o que é autenticação e autorização são os conceitos básicos para entender como funciona o OAuth, além de agregar conhecimento sobre segurança em sistemas.

Pesquisa do OAuth e suas versões:

Depois de entender os conceitos básicos, um estudo aprofundado sobre o protocolo é de total importância para entender suas variações de fluxos entre as versões 1.0 e 2.0.

Estudo de caso:

Para garantir o conhecimento adquirido anteriormente, um estudo de caso de um serviço atual ajudou a entender o funcionamento real do OAuth.

Desenvolvimento do ambiente de teste:

Para aplicar as comparações entre as versões e o conhecimento obtido sobre o protocolo, foi desenvolvido uma API Mobile contendo a possibilidade do programador utiliza-la em qualquer serviço, esse foi uma das dificuldades encontradas nas pesquisas realizadas e assim essa biblioteca serve de inicio para os problemas encontrados. O desenvolvimento de um cliente para usar a API desenvolvida foi importante para fazer a comunicação com o serviço também desenvolvido atendendo os conceitos do protocolo na camada de controle de acesso.

Enfim, todo esforço nesse presente trabalho foi de grande importância para a compreensão do OAuth possibilitou comparações entre suas versões para melhores praticas de desenvolvimento e uso do protocolo.

4.5 Resultados obtidos

Há uma grande variação no OAuth, não apenas em termos de implementação e documentação, mas também em desempenho dos serviços. É muito importante compreender este protocolo para desenvolver aplicações.

Finalmente, é necessário planejar e projetar a interface e a experiência do usuário de acordo com os atrasos de requisições, e também ter estratégias para assegurar que os usuários permaneçam conectados durante qualquer autenticação de aplicações de terceiros.

CONCLUSÃO

Com a grande necessidade de ter mais segurança na autenticação e autorização entre cliente e servidor, o OAuth se destaca sendo utilizado pelos principais serviços *web*. O pequeno número de informações contidas na internet, torna este projeto um fator positivo na comunidade do protocolo. Seguindo essa linha, pesquisar possíveis vulnerabilidades e desenvolver uma API OAuth Mobile exige um estudo aprofundado, teste e análise do protocolo para o desenvolvimento da API.

Esta monografia também apresenta os aspectos básicos do estudo sobre os protocolos de delegação de acesso, iniciando com uma rápida visão do cenário atual e enfatizando a necessidade de se controlar o acesso de terceiros sobre suas aplicações utilizando o OAuth, um protocolo de delegação de acesso seguro, resultando no desenvolvimento de um aplicativo, API OAuth mobile e um servidor para teste, esclarecendo o que foi estudado de forma prática.

O OAuth 2.0 ainda não está consolidado, as futuras implementações devem ficar preparadas para serem ajustadas à medida que os servidores de recursos protegidos e as APIs forem atualizando o protocolo de acordo com a evolução do draft. A título de exemplo o Facebook e o Google já disponibilizam serviços suportados somente em 2.0.

Além do conhecimento obtido, fica ainda para a comunidade de desenvolvedores, uma biblioteca *open source* contendo todas as classes necessárias para implementação de uma *Service Provider*, *Client*, API OAuth Mobile utilizando o protocolo OAuth. Disponível em, https://github.com/eduardoarakaki/tcc.

A documentação criada durante o desenvolvimento, possibilita trabalhos futuros utilizando delegação de acesso como tema principal ou como ferramenta de segurança para suas aplicações. Dentre possíveis trabalhos, pode-se utilizar a biblioteca para desenvolvimento de aplicativos de acesso a *Sources Providers*.

Com a chegada da versão 2.0 do protocolo OAuth e até mesmo o pouco suporte fornecido para criptografia nesta versão da biblioteca, podem ser usados como base para expandi-la e atualizá-la em trabalhos futuros.

TRABALHOS FUTUROS

A partir do projeto desenvolvido, pode-se destacar como possíveis trabalhos futuros:

- A customização da API Mobile;
- O desenvolvimento de clientes em outros sistemas operacionais mobile;
- Especificações das futuras versões OAuth;
- Testes de performance em mais serviços que utilizam o OAuth;

REFERÊNCIAS BIBLIOGRÁFICAS

APIMETRICS.IO. Intelligent API Monitoring. Disponível em: < https:// http://apimetrics.io/> Acessado em: 17 de Novembro de 2015.

ALVES, João. Principais diferenças e as razões para a criação de um novo protocolo de delegação de Credenciais. Disponível em: <http://pt.slideshare.net/jpralves/oauth-10-vs-oauth-20-principais-diferenas> Acessado em: 20 de Junho de 2015.

ANDROID DEVELOPER. Componentes da aplicação android, Disponível em: <http://developer.android.com/guide/components/index.html/> Acesso em : 17 de Abril de 2015.

BISHOP, BAILEY. A critical analysis of vulnera- bility taxonomies. Technical Report CSE-96-11, Department of Computer Science at University of California, Davis. 1996.

BOYD, Ryan, Getting Started with OAuth 2.0, 1Ed. Sebastopol: O'Reilly. 2012.

BOYD, D. M.; ELLISON, N. B. Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication, 13(11), 2007.

CERT.BR. Cartilha de sergurança para internet. Disponível em: < http://cartilha.cert.br/sobre/old/cartilha_seguranca_3.1.pdf> Acessado em: 12 de Junho de 2015.

CERT.BR. Mecanismos de segurança. Disponível em: < http://cartilha.cert.br/mecanismos/> Acessado em: 12 de Junho de 2015.

Charlene C. Q, Ann L. G.B, Michelle.S, Kelly.W, Suzanne S. C, Malinda P, Michael T ,Lauren B.H. Erik.B, Dan.L. Mobile diabetes intervention study .Disponível em:< http://www.sciencedirect.com/science/article/pii/S1551714409000342#fig1 >, Acesso em: 20 de Novembro de 2015.

CNET. Serius security flaw in OAuth and OpenID discovered. Disponível em: <http://www.cnet.com/uk/news/serious-security-flaw-in-oauth-andopenid-discovered/>. Acessado em: 11 de Junho de 2015.

DEVMEDIA, acesso em: < http://www.devmedia.com.br/introducao-ao-formato-json/25275>. Em 20 novembro 2015.

FLANDERS, Jon. Introdução aos serviços RESTful com o WCF, 2009. Disponível em: < https://msdn.microsoft.com/pt-br/magazine/dd315413.aspx> Acessado em: 12 de Junho de 2015.

F. D. JUNIOR, Luiz. Artigo Implementação de Servidor OAuth. 2010.

F. SILVA, Joni. Segurança em Serviços Web. 2006.

FACEBOOK B, acesso em: <https://developers.facebook.com/docs/graph-api>. Em 12 de Junho de 2015.

FLICKR. User authentication. Disponível em: <https://www.flickr.com/services/api/auth.oauth.html /> Acessado em: 12 de Junho de 2015.

FACEBOOK. Protocol OAuth. Disponível em: < https://developers.facebook.com/docs/reference/dialogs/oauth> Acessado em: 12 de Junho de 2015.

FIORESE, M.; Mecanismos de Autenticação de Usuários. Porto Alegre - Universidade Federal do Rio Grande do Sul - UFRGS, 1998.

FIORESE, M.; Uma Proposta de Autenticação de Usuários para Ensino a Distância, Universidade Federal do Rio Grande do Sul – UFRGS, 2000.

G. CRISTINI, Pablo. Uso de Redes Sociais para Colaboração em Pequenas Empresas e Médias Empresas. 2011.

GOOGLE. OAuth API for Java. Disponível em: < https://developers.google.com/appengine/docs/java/oauth/overview?hl=pt-br > Acessado em: 02 de Julho de 2015.

GOOGLE. AuthSub for web applications. Disponível em: < https://developers.google.com/accounts/docs/AuthSub/>. Acessado em: 11 de Junho de 2015.

HUENIVERSE. Would you like some quechup with your phish & chups?, 2007. Disponível em: < http://hueniverse.com/2007/09/10/would-you-like-some-quechup-with-your-phish-chips/> Acessado em: 11 de Novembro de 2015.

HUENIVERSE. Beginner guide to OAuth part 3: Security Architecture, 2008. Disponível em: <http://hueniverse.com/2008/10/beginners-guide-to-oauth-part-iii-security-architecture/> Acessado em: 12 de Novembro de 2015.

IBM. OAuth no IBM WebSphere DataPower Appliances, Disponível em: < http://www.ibm.com/developerworks/br/websphere/library/techarticles/1208_rasmussen/1208_rasmussen.html> Acessado em: 19 de Novembro de 2015.

IBM, RASMUSSEN. OAuth flow. Disponível em: <http://www.ibm.com/developerworks/br/websphere/library/techarticles/1208_rasmussen/1208_rasmussen.html> Acessado em: 13 de Junho de 2015.

ISO/IEC NBR 17799; Tecnologia da informação – Código de prática para a gestão da segurança da informação. Agosto, 2005.

JERSEY. Service web RESTfull in Java , Disponível em : <https://jersey.java.net/> Acesso em: 28 de Maio de 2015.

LANDWEHR, C. E. Computer Security. In International Journal of Information Security, volume 1, pages 3–13. Springer-Verlag Heidelberg. 2001.

LAHAV, Hammer. E. Introducing oauth 2.0. Hueniverse, 2010.

LAHAV, Hammer, E. The authoritative guide to oauth 1.0. Hueniverse, 2009.

LINKEDIN. Authenticating with OAuth 2.0. Disponível em: < https://developer.linkedin.com/documents/authentication> Acessado em: 11 de Junho de 2015.

MELO, Emerson Ribeiro. Redes de Confiança, 2002. Disponível em: <www.das.ufsc.br/seguranca/artigos/dissertacaomestrado.pdf> Acessado em: 10 de Novembro de 2015.

M. GERALDO, Fernando. Estudo de caso de uma estrutura de autenticação única utilizando o protocolo Oauth. 2011.

M. Eduardo. Gestão de Identidades e privacidade em redes de próxima geração. 2009.

MAFRA, Edegar.; Como funciona a autenticação OAuth, 2010.
SYMANTEC. Faude on-line: Phishin. Disponível em: < http://br.norton.com/cybercrime-phishing> Acessado em: 19 de Novembro de 2015.

MARTINS, P. R.; Relacionando o Conceito de Criptografia em Implantações de Firewalls, Senais – Florianópolis, SC, 2007.

OAUTH. OAuth Core 1.0. Disponível em: < http://oauth.net/core/1.0/ > Acessado em: 14 junho 2014.

S. OLIVO, Altair. Trabalho de Doutorado. Teias de Federações: Uma Abortagem baseada em cadeias de confiança para autenticação, autorização e navegação em sistemas de larga escala. 2004.

TWITTER. Send secure authorized requests to the Twitter API. Disponível em: < https://dev.twitter.com/docs/auth/oauth> Acessado em: 11 de Junho de 2015.

YAHOO. Authentication. Disponível em: < https://developer.yahoo.com/auth/> Acessado em: 12 Junho de 2015.

YAHOO. Browser-Based Authentication, 2015. Disponível em: <https://developer.yahoo.com/bbauth/> Acessado em: 19 de Novembro de 2015.

ZANTA, Adalberto. INFOQ. Novos rumos do OAuth 2.0: Criador deixa especificação e critica padrão. 2012. Disponível em: <http://www.infoq.com/br/news/2012/08/oauth-saida-criador> Acessado em: 14 de Junho de 2015.