UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Rodrigo José Brasil Costa e Leonardo Cesar Borges

GERENCIAMENTO DE IDENTIDADES, UM ESTUDO TEÓRICO-PRÁTICO

Florianópolis

2011

Rodrigo José Brasil Costa e Leonardo Cesar Borges

GERENCIAMENTO DE IDENTIDADES, UM ESTUDO TEÓRICO-PRÁTICO

Trabalho de Conclusão de Curso submetido ao Curso de Sistemas de Informação para a obtenção do Grau de Bacharel em Sistemas de Informação. Orientador: Profa. Dra. Carla Merkle Westphall

Florianópolis

2011

Rodrigo José Brasil Costa e Leonardo Cesar Borges

GERENCIAMENTO DE IDENTIDADES, UM ESTUDO TEÓRICO-PRÁTICO

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Sistemas de Informação", e aprovado em sua forma final pelo Curso de Sistemas de Informação.

Florianópolis, 23 de maio 2011.

Profa. Maria Marta Leite
Coordenador do Curso

Banca Examinadora:

Profa. Dra. Carla Merkle Westphall
Orientador

Prof. Dr. Carlos Becker Westphall

RESUMO

Com o crescimento exponencial da Internet e o surgimento constante e inovador das mais diversas aplicações web, vivemos um momento onde grande parte ou totalidade dos dados de uma pessoa se encontram nessa rede, em vários bancos de dados espalhados por servidores ao redor do mundo. Sendo assim, cresce conjuntamente a importância de manter esses dados em segurança, de modo que apenas essa pessoa (ou pessoas autorizadas por ela) tenham acesso aos mesmos. Um dos aspectos da segurança é o gerenciamento de identidades. Como garantir que o usuário é realmente quem ele diz ser? E mais: Como fazer essa autenticação de forma rápida, prática, segura e transparente? Várias pessoas já pensaram nisso e desenvolveram diversas ferramentas com diferentes potencialidades, no entanto qual a melhor delas? Em que situações cada uma delas melhor se aplica? Como as mesmas podem ser utilizadas na prática? Faz-se necessário, portanto, um estudo que exemplifique essas questões e permita que os desenvolvedores possam fazer a escolha certa em gerenciamento de identidades nos seus futuros empreendimentos tecnológicos.

Palavras-chave: Gerenciamento de Identidades. Segurança Computacional. OAuth. Microsoft InfoCards. OpenID. Shibolleth. SAML.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO, DEFINIÇÃO DO PROBLEMA E IDÉIA DO TRABALHO

Com o crescimento exponencial da Internet e o surgimento constante e inovador das mais diversas aplicações web, vivemos um momento onde grande parte ou totalidade dos dados de uma pessoa se encontram nessa rede, em vários bancos de dados espalhados por servidores ao redor do mundo. Sendo assim, cresce conjuntamente a importância de manter estes dados em segurança, de modo que apenas esta pessoa ou pessoas autorizadas por ela tenham acesso aos mesmos.

Um dos aspectos da segurança computacional é o gerenciamento de identidades. Esta área visa tratar diversas questões, entre elas: Como garantir que o usuário é realmente quem ele diz ser? Como gerenciar este processo de autenticação e autorização? Como compartilhar informações de usuários entre sistemas, sendo eles da mesma organização ou não? Que tipos e/ou combinações de credenciais funcionam melhor em cada um dos casos? Como tornar o processo de autenticação mais prático também para os usuários?

Nessa linha de pesquisa já existem diversas ferramentas desenvolvidas e em desenvolvimento que visam automatizar e/ou facilitar diversas das questões descritas acima. Salvo ferramentas que sejam diretamente concorrentes, não faz sentido uma comparação em termos de melhor e pior entre elas, no entanto cabe verificar qual atende melhor cada problema e tipo de sistema.

Portanto existe a necessidade de um estudo que possa fornecer uma visão geral e ampla de ferramentas já consolidadas no mercado dentro da área de Gerenciamento de Identidades, de modo que possam ser identificadas suas potencialidades e indicações de uso, servindo como referência futura aos desenvolvedores que desejem fazer a melhor escolha na área de Segurança em seus futuros empreendimentos tecnológicos.

1.2 OBJETIVOS

Este trabalho tem os seguintes objetivos:

1.2.1 Objetivo Geral

Realizar um estudo das principais ferramentas e conceitos em Gerenciamento de Identidades, descrevendo suas funcionalidades, indicações de uso e potencialidades. Onde couber, desenvolver aplicações práticas que exemplifiquem este embasamento teórico.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Revisão dos conceitos principais de segurança computacional
- Revisão dos conceitos principais de gerenciamento de identidades
- Estudo da ferramenta OpenID
- Estudo da ferramenta Shibolleth
- Estudo da ferramenta OAuth
- Estudo da ferramenta Microsoft Infocards e Windows CardSpace

O escopo desse projeto se limitará ao estudo das ferramentas descritas acima, tomando-se então como premissa que as mesmas são as mais relevantes para a proposta e objetivos do trabalho.

2 SEGURANÇA COMPUTACIONAL

2.1 INTRODUÇÃO E CONCEITO DE SEGURANÇA COMPUTACIONAL

A origem da palavra Segurança vem do Latim, com a palavra "securus"(LIST, 2011), que por sua vez tem origem na expressão 'SE' (sem) 'CURA' (cuidados), isso pode levar a uma interpretação equivocada de ser algo que não é cuidado. No entanto na realidade a expressão indica que algo seguro é algo que está garantido, e portanto não há real necessidade de se ter maiores cuidados. Sendo assim, aplicando este conceito à computação, podemos afirmar que um computador ou sistema está seguro quando não é preciso se preocupar com ele, pois o mesmo está livre de ou preparado para enfrentar qualquer ameaça. Segurança computacional, portanto, é a disciplina que estuda os meios para não precisarmos mais nos preocupar com os nossos computadores e sistemas.

De qualquer forma Segurança é uma propriedade de "elo-fraco"de um Sistema (MALCHO, 2010). Muitas vezes a forma mais simples de se obter informações (sem a devida autorização) é roubar um notebook, um HD, ou mesmo pegar papéis da lixeira, e essas possibilidades também não podem ser ignoradas. Inclusive existem meios de impedir ou dificultar essas violações, como a criptografia dos dados sensíveis nos HDs das máquinas, e não salvar as senhas nos computadores que acessam o Sistema.

Portanto a disciplina segurança computacional abrange muitos aspectos que vão desde software e hardware até o próprio comportamento das pessoas. Nesta seção serão descritos alguns conceitos básicos desta disciplina.

2.2 POLÍTICAS DE SEGURANÇA

Uma política de segurança é o conjunto das regras que definem o que deve e não deve ser feito em termos de segurança (LANDWEHR, 2001). Portanto a política irá traçar também as diretrizes e objetivos de segurança, e assim os desenvolvedores dos sistemas deverão pensar em como programar as questões de segurança de forma que os sistemas computacionais também não permitam a violação dessas regras.

A legislação de um país pode realmente servir como base e ponto

de partida para uma política de segurança em um sistema computacional. Uma nação ou um acordo entre nações podem impor políticas relativas à proteção e distribuição de informação que possam motivar as regras que compõe a política de segurança de um sistema. A necessidade de informação forense para fomentar processos judiciais sobre crimes digitais,as diretivas de privacidade da União Européia, as políticas reguladoras da proteção e distribuição de informações de Saúde nos Estados Unidos(HEALTH; SERVICES, 2000), são todos exemplos atuais de aplicação das leis em políticas de segurança computacional e da informação.

2.3 PROPRIEDADES DE SEGURANÇA

Por muitos anos, a convenção em segurança computacional definia que um sistema computacional deveria respeitar três propriedades básicas (LANDWEHR, 2001):

- Confidencialidade: Garantir que as informações no sistema não sejam divulgadas sem a devida autorização.
- Integridade: Garantir que a informação não foi modificada sem a devida autorização.
- Disponibilidade: Garantir que a informação esteja sempre disponível em todo momento que um usuário autorizado precise dela.

Mais recentemente, é comum adicionar ainda mais duas propriedades:

- Autenticação: Garantir que cada indivíduo seja quem ele diz ser.
- Não-repudiação: Garantir que uma terceira parte neutra possa ser convencida do acontecimento ou não de um evento ou transação.

Autenticação é realmente um pré-requisito para as outras três primeiras propriedades, pois sem uma autenticação confiável não é possível determinar se uma informação confidencial foi devidamente autorizada para visualização e/ou modificação.

Há um interesse maior relacionado a não-repudiação nas áreas financeira e em protocolos de comunicação. Já que traz a habilidade para o sistema de provar que um usuário e/ou parceiro realizou ou não determinada operação. Sendo assim o usuário não poderá clamar que fez o que não fez, ou vice-versa.

2.4 DELIMITAÇÃO DO SISTEMA

Uma política de segurança costuma ter uma abrangência que vai além de sistemas computacionais individuais. Ao cuidar da segurança de um sistema particular, o primeiro passo deve ser identificar o que deve estar e o que não deve estar contido no sistema. Esta atividade define o perímetro de segurança do sistema e os limites de escopo do que precisa ser protegido e deve ser controlado.

Atualmente os computadores individuais estão conectados de forma variável e imprevisível com outros, a conexão pode ser permanente ou intermitente, e a troca de informações está sempre acontecendo de alguma forma. Ou seja, aquilo que nós queremos proteger, as informações, estão em constante movimento.

Além disso, a configuração de máquinas clientes individuais, que são controladas diretamente por usuários, podem ter um impacto significativo na segurança do sistema como um todo. Por exemplo, o usuário pode instalar um modem em um notebook, de modo que ele possa acessar essa rede de qualquer lugar fora da empresa. Embora o usuário pense que seja por uma boa causa, essa atitude pode abrir o caminho para que um intruso, vindo de fora da intranet, acesse essa rede corporativa sem passar pelo firewall que a protegeria desses ataques externos.

Outro aspecto crítico é que os computadores constantemente trocam informações que representam programas que devem ser executados e muitas vezes até instalados permanentemente. As implicações de segurança com essa troca de programas acabam sendo complicadas de tratar, visto que até mesmo programas visando a própria segurança do computador, como anti-vírus, utilizam esse tipo de processo para instalar atualizações e melhorias. Isso dificulta a simples proibição de tais práticas, o recomendável então é que se verifique sempre a fonte e a integridade da informação recebida.

Um dos propósitos mais importantes de se definir o perímetro da segurança de um sistema é poder distinguir melhor os intrusos de usuários legítimos. Um intruso é um indivíduo que atravessa a barreira de segurança (o perímetro) de um sistema sem a devida autorização. Como um intruso pode obter esse acesso ao roubar e utilizar as credenciais de um usuário legítimo, dependendo do comportamento desse, distingui-lo do usuário pelo qual ele tenta se passar se torna também um problema complicado de resolver.

2.5 RISCO, VULNERABILIDADE, AMEAÇA E ATAQUE

Uma ameaça é uma intenção de provocar algum dano ao Sistema (CARROLL, 1996). Diferentes indivíduos e grupos possuem habilidades distintas para levar uma ameaça adiante, ou seja, fomentar um ataque. Por isso os desenvolvedores do sistema precisam conhecer a natureza das ameaças que o permeiam para então planejarem a sua arquitetura de segurança. Uma ameaça ligada a um usuário legítimo do Sistema é chamada de "ameaça interna", e é um tipo muito perigoso, pois este já possui certo nível de acesso ao Sistema, e consegue obter informações privilegiadas sobre seu funcionamento.

A vulnerabilidade é um ponto fraco em um sistema computacional. Ela pode ser uma falha em uma parte de um software que executa em modo privilegiado, uma senha mal elaborada, uma regra mal configurada no firewall. Pode ser até mesmo a dependência de uma informação externa ao sistema. Vulnerabilidades existem em todos os sistemas operacionais da atualidade. Quase todos os dias são disponibilizadas novas atualizações de segurança, com o intuito de desfazer vulnerabilidades.

Já o risco, que é assumido pelo dono e/ou administrador do sistema, é a probabilidade de o sistema não estar apto a controlar totalmente sua política de segurança (incluindo a continuidade de operações críticas) diante de um ataque. Risco também é uma função que envolve o nível de exposição das vulnerabilidades do sistema e a quantidade de ameaças manifestadas contra o mesmo num determinado espaço de tempo. Ou seja, o risco pode mudar ao longo do tempo mesmo sem que a empresa tome qualquer atitude positiva ou negativa em relação a ele, pois envolve também fatores externos e incontroláveis.

Organizações podem utilizar diferentes estratégias de como tratar os riscos, uma filosofia que visa evitar completamente os riscos irá focar em remover todas as vulnerabilidades e vencer qualquer ataque. Mas a adoção massiva de sistemas (inclusive sistemas operacionais) inerentemente vulneráveis tem tornado essa aproximação quase impraticável nos dias de hoje. Sendo assim as organizações tem adotado uma estratégia de gerenciamento de riscos, onde um certo grau de risco é aceito como normal e só há mais atenção e cuidado no momento de se lidar com ataques bem sucedidos.

Ainda assim muitas vezes é difícil distinguir uma estratégia de "gerenciamento de riscos"de uma simples "aceitação de riscos".

2.6 MECANISMOS DE SEGURANÇA

Para conseguir de fato colocar na prática a implementação das políticas de segurança no Sistema Computacional, são utilizados vários mecanismos, técnicas e ferramentas.

2.6.1 Definição de Domínios

Os mecanismos primários de segurança utilizados para construir sistemas computacionais, estes que conseguem agir em prol de políticas de segurança específicas, efetivamente criam dispositivos análogos a cercas e portões ao redor dos computadores e/ou sistemas. Uma área cercada em um sistema computacional corresponde a um domínio, que pode ser compreendido como uma coleção de dados e um conjunto de autorizações para manipular estes dados dentro o mesmo.

Quando um sistema inicializa, o estabelecimento de um domínio (que esteja de acordo com os requerimentos de segurança) é essencial. Basta recordar os vírus que tentam ganhar o controle sempre que o sistema é reiniciado. E não só os mecanismos para criação de domínios são importantes, como também as ferramentas para controlar a comunicação entre esses domínios são requeridas.

Os domínios podem ser construídos utilizando funções de hardware específicos do sistema computacional, ou utilizando apenas software ou também, como é mais comum, a combinação dos dois. Mecanismos para controlar o acesso a instruções privilegiadas de CPU, como funções de i/o, restrição de acesso à leitura da memória principal, são exemplos de controle de domínios no nível mais básico, de hardware.

2.6.2 Autenticação e Associação de Usuários com os Domínios

Segurança requer que ações e operações possam ser atribuídas ao usuário que as produziu. Esse princípio leva a necessidade de associar usuários com os domínios. Identificação (usuário diz quem ele é) e autenticação (ter a prova de que essa identificação condiz com a realidade) são os processos necessários para estabelecer esta associação.

Fornecer identificadores e senhas para os usuários continua sendo o método mais comum de identificação e autenticação, mas a utilização de tokens (utilizados em smart cards, em protocolos como OAuth e outras tecnologias atuais) e biometria vem crescendo cada vez mais.

Essas formas diferentes de autenticação dependem de três fatores:

1. Algo que o usuário conhece: uma senha.
2. Algo que o usuário possui: um token ou cartão de acesso.
3. Algo que o usuário é: uma impressão digital, a forma da sua íris, informações vitais.

De modo geral, quanto mais elementos são fornecidos em uma autenticação, mais forte ela será. No entanto não adianta levar em consideração somente o número de elementos de requeridos, mas também a força individual de cada um desses elementos.

Um elemento chave no processo de autenticação é a existência de um caminho confiável entre a fonte dos dados de autenticação e o componente que tomará a decisão de autenticação. Se houver um elemento não confiável nesse meio, este pode coletar e/ou modificar informações cruciais para este processo. Para dificultar a ação desse tipo de ouvinte, alguns protocolos de autenticação incluem alguns dispositivos de verificação a cada mensagem trocada, para assegurar que a mesma não foi modificada no caminho, ou enviada por uma entidade que não é a esperada.

Uma vez que a identidade de um indivíduo é associada a um domínio, as atividades dos programas executando naquele domínio vão herdar os privilégios daquele usuário. É importante perceber neste ponto que se um identificador de usuário é compartilhado por mais de uma pessoa, a associação entre o usuário e o domínio é severamente enfraquecida.

Por exemplo, pode parecer conveniente colocar simplesmente um identificador administrador que possua todos os privilégios máximos e distribui-lo entre os vários administradores deste sistema. Num caso como este, perde-se o poder de identificar especificamente o responsável no caso de algum dano ser causado pelo mau uso dos privilégios deste identificador.

2.6.3 Criptografia

Um algoritmo de criptografia tem a capacidade de transformar um texto legível em um texto cifrado que só poderá ser compreendido aplicando outra chave criptográfica e o mesmo algoritmo(ROTHMAN, 1999). Se as chaves e algoritmos são os mesmos na hora de cifrar e decifrar, então temos o que é chamado de sistema de chaves simétricas

ou chaves secretas. Se o algoritmo envolve duas chaves distintas, então este pode ser um algoritmo de chaves assimétricas ou chave pública.

Algoritmos de chave pública tem a vantagem de que uma das chaves pode ser divulgada publicamente, enquanto a outra precisa ser mantida em segredo. Sendo assim a informação cifrada com a chave pública só pode ser decifrada com a chave privada. Entretanto o aspecto interessante deste processo é que uma mensagem cifrada com a chave privada só pode ter sido gerada pela entidade que possui esta chave secreta, portanto este procedimento serve como assinatura digital, garantindo então a procedência da informação.

3 GERENCIAMENTO DE IDENTIDADES

3.1 CONCEITOS BÁSICOS SOBRE GERENCIAMENTO DE IDENTIDADES

3.1.1 Visão Geral

Uma maneira de pensar sobre o gerenciamento de identidade é imaginando um projeto enorme de um prédio de escritórios. Ele mostra as salas em que cada pessoa que trabalha no edifício pode entrar. O modelo também mostra que tipo de pessoa precisa de chave para abrir a porta para entrar em uma sala e o que essa pessoa pode fazer uma vez estando lá.

Uma rede de computadores é como o prédio, e cada quarto representam um arquivo de banco de dados ou a aplicação na rede. Os empregados que trabalham no edifício são os usuários. As chaves são os privilégios que o administrador do sistema distribui a cada pessoa que trabalha na rede, fornecendo acesso a um arquivo de banco de dados ou aplicativo (AITORO, 2008). As chaves também determinam o que podem fazer ao acessar um arquivo ou aplicativo específico.

O Gerenciamento de identidades define uma identidade digital para cada entidade (pessoas, hardware ou processo), associa atributos à identidade e reforça os meios através dos quais a identidade pode ser verificada. Normalmente distribuída como um programa ou um conjunto de programas e por vezes acompanhada por um equipamento dedicado.

Desde a criação até a dissolução, contemplando todas as alterações neste processo, um sistema completo de gestão de identidades inclui autenticação e autorização dos utilizadores, direitos de acesso, privilégios e autorizações. A autenticação dos usuários por sign-on , insto é, a capacidade de um utilizador acessar a todos os recursos de rede após uma única autenticação e adicionalmente gerir identidades dos indivíduos. Ferramentas de gestão de identidade permitem aos administradores controlar a infra-estrutura de identidades, eliminando os privilégios de acesso inativos.

3.1.2 Definição

Segundo o autor Sean Daily (DAILY, 2004), uma definição geral seria: "A essência da gerencia de identidade como uma solução é fornecer uma combinação de processos e tecnologias para gerenciar o acesso seguro às informações e aos recursos de uma organização, protegendo simultaneamente os seus usuários. Gerencia de identidade pode prover os recursos para gerir eficazmente esses processos internos e externos de uma organização para funcionários, clientes, parceiros e até mesmo aplicações e conseqüentemente alguém, ou qualquer coisa que precise interagir com a organização."

Junto com as considerações citadas acima, esta definição, nos faz entender que a gerencia de identidades tem seu foco em prover segurança de acesso e facilidades de acesso aos seus usuários. Sendo estes usuários os utilizadores do sistema ou os fornecedores de conteúdo.

Devido ao crescente interesse em Gerenciamento de Identidades nos últimos anos. Alguns analistas ofereceram seus pontos de vista e definições de mercado, como pode-se observar nas citações abaixo:

- "Identidade digital compreende os registros eletrônicos de identidade, incluindo nomes ou identificadores únicos, credenciais, endereços, direitos, e outros dados de posse de domínios sobre a identidade na rede"(GROUP, 2002).

- "A noção de Gerenciamento de Identidades como uma questão de negócio é: Gerenciamento de Identidade é muitas vezes vendido puramente como uma solução de segurança, mas as organizações estão percebendo que também abrange a experiência do usuário, eficiência e agilidade nos negócios. As organizações estão começando a perceber isso, e com isto desenvolvendo Gerenciamento de Identidade como estratégias de gestão e incorporando nos seus planos de arquitetura corporativa(INFORMATION, 2002).

- "Gerenciamento de Identidades engloba a integração de produtos, tais como diretórios, single sign-on e aplicativos em uma estrutura unificada de administração de usuários e direitos de acesso através de múltiplos sistemas e contextos de negócio. O interesse empresarial em gerenciamento de identidades está aumentando não apenas porque melhora a segurança. Gerenciamento de Identidades também aborda problemas críticos de negócios e proporcionam um quantificável retorno do investimento em quatro áreas principais: produtividade do usuário, eficiência da gestão, redução dos

custos e agilidade no desenvolvimento."(INFORMATION, 2002)

3.1.3 Desafios em Gerenciamento de Identidades

Gerenciamento de Identidade irá criar alguns problemas de segurança, no entanto. Depois de criar uma central de soluções de identidade controlada, você também cria um foco para todos os ataques de segurança. Outra questão surge quando uma identidade incorreta é capaz de ser usada. Em outras palavras, a informação acessada ou fornecida é imprecisa. Talvez um usuário está usando a identidade certa no contexto errado: você está tentando se autenticar na intranet da sua empresa usando o seu nome pessoal e senha para o provedor de serviços Internet (ISP). Identidade também pode ser perigosa quando a identidade esta correta, porem sendo usada por outra pessoa indevidamente. Um dos objetivos de um programa que abrangente soluções em Gerenciamento de Identidade é garantir que o contexto certo será utilizado no momento oportuno.

Hoje, os funcionários que estão com uma empresa por mais de três anos são considerados geralmente empregados a longo prazo, e os consumidores que fazem compras na Web têm a capacidade de passar de um local para outro até encontrar as melhores ofertas. Esta taxa de troca natural de volume de negócios cria um problema significativo para as organizações em termos de saber com quem estão lidando, gerando assim o desafio de disponibilizar e essas pessoas o acesso certo ao que eles precisam, quando precisam e bloquear o acesso, se não são permitidos. No caso dos funcionários, que por vezes podem levar muitos meses para o direito de acesso aos sistemas de direito, sendo assim outro desafio do Gerenciamento de Identidades.

É provável que você possua um cartão de identificação de funcionário para o trabalho, outro para fazer compras na Internet em diversas lojas usando identificações de login diferentes.Isto nos mostram inúmeras formas de "identificação"como a carteira de motorista, cartões de identidade ou certidão de nascimento,cartões de crédito e débito e assim por diante. O número de ferramentas que você tem, e deve utilizar, para demonstrar sua identidade para outros pode ser significativo, e parece estar aumentando rapidamente. Eu poderia utilizar uma combinação destes, dependendo da circunstâncias ou contexto.

Viajei para vários lugares ao redor do mundo, aumentando o número de identidades nacionais. Esta figura também ilustra que, como um indivíduo com muitas identidades, também tenho muitas relações

Figura 1 – Várias Identificações

diferentes e variadas com as organizações, governos, e as empresas: como funcionário, como cliente ou consumidor, como cidadão, como um estrangeiro, e assim por diante. Essas relações são conhecidas como contexto de identidade, que é um conceito importante.

Como os problemas anteriormente mencionados ilustram, há muitos fatores que estimulam a adoção de soluções em Gerenciamento de Identidade, esses desafios são enfrentados não só pela empresa, mas também pelos consumidores e governos. Tal como acontece com muitas organizações, os governos têm os suas próprias políticas sobre o que os dados de identidade exigem dos indivíduos dentro de suas fronteiras. Governos possuem políticas sobre como essas informação são compartilhada entre órgãos do governo. Da mesma forma, eles poderiam ter políticas sobre se a informação podem ser compartilhada fora das agências. O mesmo vale para os dados de funcionários e clientes das empresas e outras organizações. Assim, podemos ver que pelo menos alguns componentes de Gerenciamento de Identidades são vitais em todo o vasto conjunto de entidades empresariais e governamentais em todo o mundo.

Os desafios são muitos, não só porque este conjunto de requisitos e disciplinas são novidades para aqueles que estão tentando implementar soluções de Gerenciamento de Identidade, mas também porque a abrangencia das soluções estão evoluindo rapidamente em termos de alcance e capacidades. As soluções de Gerenciamento de Identidade têm historicamente, tomado diversas formas e são comumente aceitas como uma parte específica da segurança da empresa ou como um conjunto de componentes principalmente na infra-estrutura de segurança. A realidade é que o componente de segurança é apenas uma pequena área dentro das fronteiras de Gerenciamento de Identidade.

3.1.4 Benefícios do Gerenciamento de Identidades

A lista a seguir apresenta os principais objetivos e vantagens da implementação de uma solução com o Gerenciamento de Identidades na gestão de uma organização:

- Reduzir o custo total de propriedade (TCO) para todos os sistemas (reduzir a administração e os custos de suporte técnico)
- Reduzir a sobrecarga de gerenciamento
- Proporcionar uma vantagem competitiva por meio da automação que permite o racionamento e otimização de processos de negócios
- Melhorar o serviço do empregador ao cliente e manter o controlo e confidencialidade dos clientes, fornecedores e funcionários
- Reduzir o tempo levado para que os novos empregados obtenham acesso aos recursos necessários dentro da organização
- Reduzir o risco de informações incorretas sendo usadas para processos de negócios
- Reduzir o risco de manter de ex-empregados com o acesso aos recursos organizacionais
-

Feito corretamente, as soluções de Gerenciamento de Identidade apoiam as diversas iniciativas de segurança , bem como:

- VPNs
- Infra-estruturas Públicas chave pública (PKI)
- Os serviços de pesquisa, como Páginas Brancas e Domain Name Service (DNS)
- Acesso controlado a dados corporativos.

Soluções de Gerenciamento de Identidade pode também suporta muitos requisitos de gerenciamento de perfil, incluindo:

- A satisfação do cliente, garantindo a consistência das informações
- O gerenciamento de perfis que permite a personalização de sites e aplicativos

- Gerenciamento de rede
- Diretório Enabled (DEN) Networking

Para ilustrar esses benefícios, considere um exemplo do setor de saúde. A maioria dos hospitais ou centros de saúde têm as seguintes questões: Vários locais, parceiros e fornecedores. Sistemas ambulatorial díspares e deconectados. Interfaces diferentes entre os sistemas internos, tais como aqueles usados para clínicos, funções financeiras e administrativas. O resultado deste ambiente é que as informações de tratamentos anteriores não são encontradas facilemte quando um paciente é admitido; históricos de pagamento não são mantidas, e informações demográficas pode não ser consistente. As implicações de tais deficiências poderiam ser, na pior das hipóteses, fatais. Prontuários eletrônicos, que permitem o acesso às históricos dos pacientes on-line, são uma ferramenta essencial para os médicos. Sem um registro permanente do paciente, prontuários eletrônicos não são viáveis. No entanto, essa ferramenta é extremamente difícil de implementar quando há muitos sistemas distintos que não podem manter uma identidade do paciente entre eles. Nos Estados Unidos, quando você adiciona prontuáeios para este cenário, a situação torna-se uma crise de identidades. Prontuários exigem a criação e a manutenção de um registro permanente do paciente, com disponibilidade de informações aos cuidados de doadores e com restrições de segurança para preservar a confidencialidade. Hospitais devem cumprir estes requisitos para disponibilizarem um bom serviço para os pacientes. Como você pode ver, uma solução de gerenciamento de identidade é necessária para superar essa crise de identidade.

3.2 IDENTIDADES FEDERADAS

3.2.1 Visão Geral

O conceito de federação é usada quando os sistemas desligados ou empresas precisam interoperar uns com os. Mas o que é a federação? Foi sugerida a seguinte definição (GROUP, 2002) "Identidades Federadas são a utilização de acordos, normas e de tecnologias para tornar identidades e seus direitos portáveis através de domínios autonomos de identidades."O objetivo da federação é o de permitir o intercâmbio transparente e seguro de identificações e informações que permitam aos sistemas distintos interoperar com um nível de segurança.

Federação cria um risco. Federação exige pausas nas fronteiras

organizacionais e pode diminuir qualquer controle de conteúdo que você possa ter, a fim de facilitar a circulação de dados de identidade entre os limites. Isto exige que as políticas sejam tenazmente definidas fora da tradicional técnica de definições de limites. As questões legais e de conformidade devem ser cuidadosamente examinada. A realidade é que o conceito federações ainda está evoluindo.

Os acontecimentos estão nos mostrando que os padrões estão evoluindo. E que ainda há uma quantidade considerável de trabalho que precisa acontecer.Um dos requisitos essenciais são os acordos que são estabelecidos antes do tempo em torno de um objetivo específico e fisicamente assinado e executado para garantir que existam processos de resolução em vigor. Caso algo inaceitável acontece durante as interações na federação. Por exemplo, se os dados da conta, como limite de gastos, não é atualizado por um parceiro no processo, o que provoca uma perda financeira, a outra parte, que permite uma compra a ser feita ou um serviço a ser utilizado com base no que fora de informações desatualizadas. Quem é responsável? Isso não é algo tratado com os padrões atuais, como SAML.

3.2.2 Projetos

Identidades Federadas aparecem como uma solução real no mercado e foram incorporadas por três projetos principais: Shibboleth, WS-Federation e Liberty Alliance. Essas iniciativas promovem especificações, alternativas e motivações sobre Identidades Federadas. São formados por vários grupos de diversos segmentos organizacionais: instituições de ensino, indústria, fornecedores de tecnologia, consumidores, varejistas, etc.

WS-Federation foi inteiramente formada por fornecedores de tecnologia, Shibboleth por comunidades academicas e Liberty Alliance é dirigido principalmente por empresas orientadas ao consumidor (United Airlines, American Express, Hertz e outros), ao invés de orientado a tecnologias.

O Shibboleth foi criado pela entidade que também é responsável pela Internet2 privada, a sua especificação, sem dúvida, será utilizada neste ambiente. Para acesso à Internet a tendência é que tanto Liberty Alliance ou o WS-Fed sobrevivam por atender os interesses comerciais.

Identidades Federadas têm três características importantes:

- Single Sign-on: o usuário se autentica uma vez e, depois disso, pode navegar para sites de diversas empresas que prestam serviços

na Internet, uma vez que são parte de um mesmo círculo de confiança.

- Círculo de acordo de Confiança: para unificar as identificações diversas na internet a apenas um, é necessário as partes envolvidas (entidades) criar laços de confiança. O usuário não só tem a confiança nas entidades envolvidas, mas também sabe quais as entidades envolvidas e se estas têm o compromisso suficiente para elas gerirem o controle de sua identificação.

- Controle de usuários e intercâmbio: esta é a atividade de "federar"a identidade do usuário, onde a identificação e outros dados de uma entidade (site) estão associadas à identificação do mesmo usuário em outra entidade (site). Esta atividade é realizada apenas somente após o reconhecimento prévio do usuário.

3.2.3 Exemplos de Identidades Federadas

3.2.3.1 Exemplo 1

O exemplo a seguir mostra a experiência de um usuário em um ambiente onde a empresa que detém a identidade do usuário (Identity Provider) oferece a possibilidade de "federar"a identidade do usuário com outra empresa como provedor de serviços.

Personagens usados nos exemplos:

- Usuário: João

- Provedor de Identidade: VoeJá Aerolíneas

- Provedor de Serviço: BRCar Aluguél de Carros

No Exemplo 1 temos a demonstração do processo de federação de uma novo usuário dentro do cliclo de confiança. Todo o processo é compost por 6 etapas. Na primeira o usuário se loga na no site da companhia área. Segunda passo o usuário é solicitado se quer ser informado da possibilidade de federar sua identidade. Terceiro passo, o usuário se loga no site de alugueis de carros, que também pertence ao circulode confiança. No quarto passo, o usuário é avisado da possibilidade de federar sua identidade. No quinto passo a identidade é federada. No último passo o usuário tem acesso aos recursos federados.

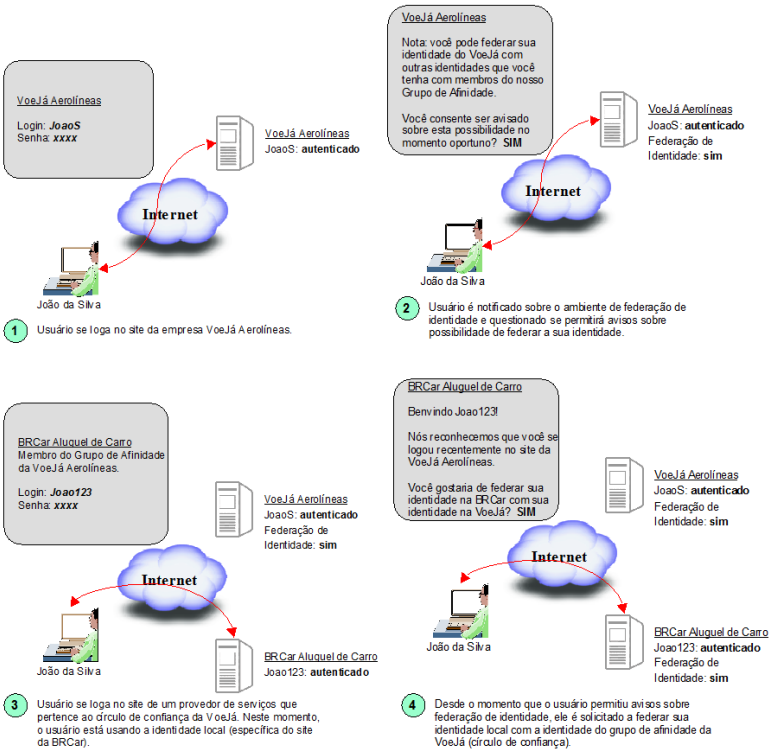

Figura 2 – Exemplo 1

3.2.3.2 Exemplo 2

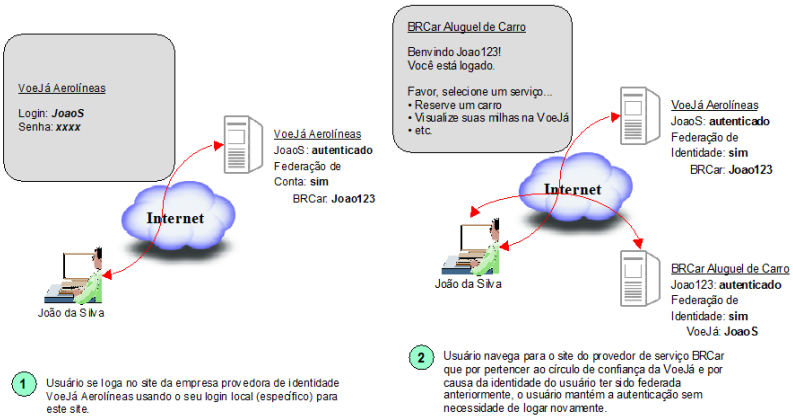

Figura 3 – Exemplo 2

No Exemplo 2, o usuário já possui uma identidade federada, então o usuário apenas acessa um dos sites que pertencem ao circulo de confiança e já consegue ter acesso aos recursos federados

4 MICROSOFT INFOCARDS E WINDOWS CARDSPACE

4.1 WINDOWS CARDSPACE

Cartões de informação são o principal componente da identidade CardSpace do sistema de gestão e de autorização da Microsoft. Cartões de informação são uma representação digital on-line de uma identidade pessoal. Cartões de informações têm algumas características excelentes tanto em termos de usabilidade e segurança. Do ponto de vista da usabilidade, o exemplo que podemos usar são os cartões de informações de plástico que todos estão familiarizados (DAVID,).

Os cartões são exibidos no desktop do usuário para que o usuário possa selecionar o cartão ele quer usar na sua federação. Cartões que são aceitáveis para o prestador de serviços ,e , portanto, selecionáveis, aparecem em cores, enquanto os cartões que são incompatíveis com os requisitos são acinzentados e, portanto, não selecionáveis. Os cartões podem ser auto-gerado ou gerenciado por um gerenciador. Cartões gerados automaticamente contêm informações (atributos) afirmados pelo próprio usuário, enquanto que os cartões gerenciados conter atributos que são afirmado pelo IDP (DAVID,).

Microsoft chama os atributos de afirmação de "reivindicações". O fato de que o atributo de afirmação (ou reclamações) dos cartões gerenciados na verdade, não residem no desktop do usuário, mas são retirados do IdP sobre demanda, é em grande parte escondida do usuário. A única característica que o usuário tem que informar são suas credenciais de login com o IdP. Isso poderia ser visto como uma desvantagem de usabilidade ou inconveniencia para os utilizadores, uma vez que o usuário é distraído de sua tarefa principal, que é está acessando um provedor de serviços, em fornecer credenciais de autenticação para um partido de alternativo, o provedor de identidade (DAVID,).

4.2 REPRESENTANDO IDENTIDADE DIGITAL: TOKENS DE SEGURANÇA

Apesar da sua diversidade, as identidades digitais têm uma coisa importante em comum: quando transmitidas na rede, cada identidade digital é representada por algum tipo de token de segurança.Um token de segurança é apenas um conjunto de bytes que expressa informações sobre identidade digital. Essa informação consiste em um ou mais

reivindicações, cada qual contém uma parte do total das informações transmitidas sobre esta identidade. Um token de segurança simples podem incluir apenas um pedido contendo um nome de usuário, enquanto um mais complexo pode incluir os créditos com nome de usuário, sobrenome, endereço residencial, e muito mais. Tokens de segurança para algumas identidades digitais pode incluir também informações sigilosas, como números de cartão de crédito.

Como na maioria dos tokens de segurança, algumas informações são fornecidas para provar que estas reivindicações realmente pertencem ao usuário que está apresentando. Uma simples (e atualmente muito comum) maneira de fazer isso é enviar uma senha junto com as reivindicações. Uma abordagem mais poderosa é a assinatura digital de todos ou parte das reivindicações, usando uma chave privada, e então fornecer a chave pública correspondente, talvez envolto em um certificado. No entanto, é feito, os tokens de segurança que representam as identidades digitais geralmente oferecem algum tipo de prova que permite que um receptor do sinal verifique se esse símbolo realmente representa a pessoa ou organização com essa identidade.

A idéia básica de cada token de segurança é a mesma: é uma coleção de declarações em uma variedade de formatos diferente. O exemplo mais simples é apenas um nome de usuário representado como uma seqüência de texto, mas os formatos mais complexos, tais como certificados X.509 e tickets Kerberos também são comuns. Nenhum destes formatos foram concebidos para permitir o envio de um conjunto arbitrário de reivindicações, no entanto, algo que é bastante útil para algumas identidades digitais. Tokens criados usando o Security Assertion Markup Language (SAML), um padrão criado pela OASIS, permite isso. Tokens podem ser baseados em XML, SAML, para definir os conjunto necessário de reivindicações.

Tradicionalmente, as identidades digitais têm sido utilizadas principalmente para autenticação.Atualmente é mais comum tokens de segurança formatos de nome de usuário, certificados X.509. Os tickets Kerberos refletem isso. Porque a informação carregada é amplamente focada em autenticação de uma identidade. Mas por que as identidades digitais não são tão amplamente úteis como identidades reais? Cada cartão em sua carteira reflete algum tipo de identidade, e cada um carrega também informações úteis que pode ser aprocado por alguma autoridade. Por exemplo, sua carteira de motorista inclui o seu nome, sua idade, talvez a sua imagem, e outras informações, todas validadas por uma organização governamental. A identidade digital que expressa essa informação poderia ser útil para várias coisas, como pro-

var que você tem 21 anos ou mais, ou que você realmente usa óculos. Da mesma forma, cada um dos seus cartões de crédito traz o número do cartão e a data de validade, juntamente com o seu nome. Assim como esses cartões são úteis no mundo físico, também seria útil criar uma identidade digital para cada cartão que pode ser usado para gerar um token de segurança carregando os créditos apropriados.

4.3 ESTRUTURA DE PAPÉIS

As múltiplas identidades digitais que usamos vêm de várias fontes diferentes, e são representadas de varias maneiras. Em outras palavras, que normalmente contam com um número de diferentes sistemas de identidade digital, cada um dos quais pode usar uma tecnologia subjacente diferente. Para pensar sobre esta diversidade de uma maneira geral, é útil definir três funções distintas:

- Usuário: Conhecido como o sujeito, o utilizador é a entidade que está associado com uma identidade digital. Os usuários são muitas vezes as pessoas, mas as organizações, aplicações e máquinas.

- Prestador de Identidades: Um provedor de identidade é apenas aquilo que o nome sugere: algo que proporciona uma identidade digital para um usuário. A identidade digital atribuída a você pelo seu empregador, por exemplo, o provedor de identidade é normalmente um sistema como o Active Directory. Para a identidade digital que você usa com a Amazon, o provedor de identidade é eficaz, desde que você defina seu nome de usuário e senha. Identidades digitais criadas por diferentes provedores de identidade pode trazer informações diferentes e oferecem diferentes níveis de garantia de que o usuário é realmente quem diz ser.

- Parte Confiável: É uma aplicação que, de alguma forma se baseia em uma identidade digital. A Parte confiável freqüentemente usará uma identidade (ou seja, as informações contidas nas reivindicações que compõem essa identidade do token de segurança) para autenticar um usuário, e depois tomar uma decisão de autorização, como permitir que que o usuário acesse algumas informações. Outra Parte Confiante também pode usar a identidade para obter um número de cartão de crédito, para verificar se é o mesmo usuário que está acessando em momentos diferentes uma aplicação ou para outros fins. Exemplos típicos incluem si-

tes na internet, como livraria on-line e sites de leilão, e qualquer aplicação que aceita solicitações através de Web Services.

Tendo em conta estes três papéis, não é difícil entender como o Windows CardSpace e o meta-sistema de identidade pode apoiar qualquer identidade digital. A Figura 4 mostra as interações fundamentais entre as três funções.

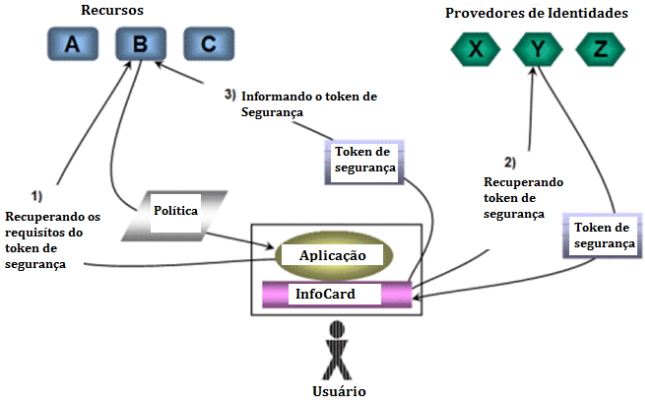

Figura 4 – Interação entre as funções

Um usuário pode contar com um aplicativo que oferece suporte CardSpace, com um navegador da Web, para acessar qualquer um dos vários partidos de confiança. Ela também pode ser capaz de escolher entre um grupo de provedores de identidade como a fonte da identidade digital. Independentemente da escolha que ela faz, a troca básica entre esses partidos tem três etapas:

1. Primeiro, o aplicativo recebe os requisitos do token de segurança da Parte Confiável que o usuário deseja acessar. Esta informação está disposta em confiar na política das Partes Confiantes, isso inclui que a Parte Confiável aceitará e confirmará os símbolos contidos.

2. Uma vez que os detalhes da Parte Confiável deste token de segurança for confiado, o aplicativo passa essa informação para o CardSpace, pedindo-lhe para solicitar um token de um provedor de identidade adequado.

3. Uma vez que este símbolo de segurança tenha sido recebido, o CardSpace dá para o aplicativo, que passa para a Parte Confiável.

A Parte confiável pode usar esse token para autenticar o usuário ou para outros fins.

5 SHIBBOLETH

5.1 O QUE É O SHIBBOLETH

O Sistema Shibboleth é um pacote de software de código aberto, baseado em padrões, que permite o single sign-on dentro e fora das fronteiras de uma organização. Com o Shibboleth os sites podem tomar decisões (baseadas em informações) precisas em relação a autorização de acesso de indivíduos à recursos protegidos, de maneira que preserve a privacidade dos mesmos (INICIATIVE, 2011).

Ele foi desenvolvido especificamente para tratar os seguintes problemas:

- Múltiplas senhas requeridas por múltiplas aplicações
- Tratar o gerenciamento de contas (registros de usuários) entre múltiplas aplicações.
- Problemas de segurança associados com o acesso a serviços de terceiros.
- Privacidade.
- Interoperablidade dentro e fora das fronteiras de uma organização.
- Permitir que instituições escolham sua tecnologia de autenticação.
- Permitir que provedores de serviços controlem o acesso aos seus recursos.
- Facilitar a integração entre vários serviços de terceiros.

Para transpor esses desafios, o Shibbolleth conta com dois componentes principais, que trabalham em conjunto:

- O Provedor de Identidades (Identity Provider, ou simplesmente IdP): este se encontra e executa nas organizações que possuem usuários que desejam acessar algum serviço restrito.
- O Provedor do Serviço: este se encontra e executa nas organizações que dispõem de serviços restritos.

5.2 FUNCIONAMENTO DO SHIBBOLETH, EXEMPLO NA VISÃO DO USUÁRIO

Imaginemos uma situação: tomando como premissa que a UFSC, UDESC e UNISUL fazem parte de uma federação de Universidades que compartilham recursos e possuem um sistema de identidades federadas baseado na ferramenta Shibbolleth.

Sendo assim um usuário, aluno da UFSC, deseja acessar um artigo acadêmico que se encontra nos servidores da UNISUL. A Figura 5 ilustra as três entidades principais envolvidas no processo: Usuário, Serviço de Descoberta, a organização da qual ele faz parte (no caso a UFSC), e por fim o Recurso (nesse caso o Artigo).

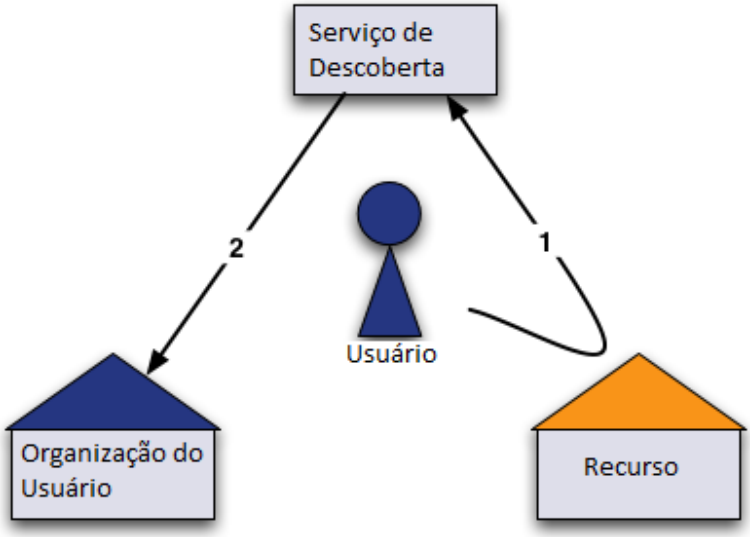

Figura 5 – Visão geral das entidades envolvidas

O primeiro passo é o usuário tentar acessar o artigo, hospedado no site da UNISUL, em http://www.unisul.br/artigo1 . Como ele não havia acessado recentemente nenhum recurso pelo Shibbolleth (se o tivesse feito, já estaria autenticado e provavelmente o recurso já poderia ser acessado diretamente), ele primeiro vai ter que se autenticar junto à sua organização de origem (UFSC). Pelo fato do Recurso (página do artigo) ainda não ter conhecimento de qual Universidade da Federação

este usuário faz parte, ele primeiro será redirecionado para o Serviço de Descoberta, também conhecido como WAYF (Where Are You From), que irá obter essa informação.

Nesse cenário imaginário, serviço de Descoberta está hospedado em http://www.ufederadasdosul.com.br/wayf. Ao ser redirecionado para esta página, o usuário visualizará uma lista de organizações, e deverá selecionar aquela da qual faz parte, como mostra a figura 6.

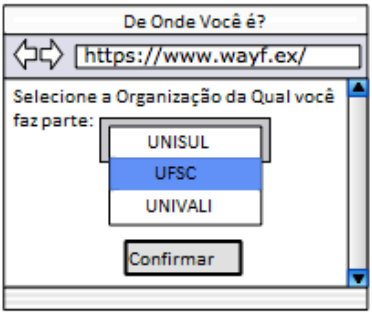

Figura 6 – Tela principal do Serviço de Descoberta

Ao selecionar sua organização, ele será encaminhado para a página de login da mesma (Figura 7), neste exemplo alguma página de autenticação dentro do domínio da UFSC.

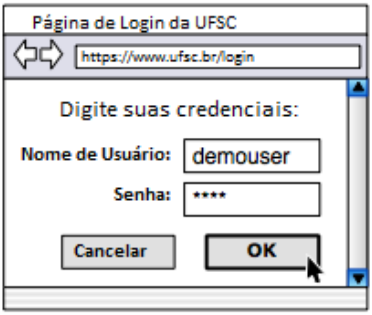

Figura 7 – Autenticação feita pela Organização de Origem do Usuário

Se o usuário inserir corretamente seu "nome de usuário"e "senha", ele será então redirecionado de volta para o Recurso que ele

queria acessar inicialmente, a página do artigo no site da UNISUL. No entanto, mesmo estando autenticado, isso não garante que este determinado usuário tenha acesso ao artigo, essa decisão cabe ao dono deste recurso, que receberá somente as informações estritamente necessárias para tomar esta decisão. Sendo assim a privacidade dos dados do usuário é preservada. A figura 8 resume todo esse procedimento de login do Shibboleth.

O procedimento de login federado pelo Shibboleth ocorre de forma transparente, na visão do usuário ele acaba acontecendo da mesma forma que um login comum em uma única instituição. A diferença, neste caso, é que estando autenticado, o usuário tem acesso à recursos em todas as instituições da federação, utilizando uma única credencial válida.

5.3 UTILIZAÇÃO DO SAML NA PLATAFORMA

Não há como tratar de Shibolleth sem tratar também de SAML, então nesta seção será explanado o funcionamento básico, recursos e funções principais deste padrão.

O SAML é amplamente utilizado dentro do Sistema Shibboleth, ele padroniza toda a gama de funções associadas com a recepção, transmissão e compartilhamento de informações de segurança, e mais especificamente (COMMUNITY, 2011):

- Provê modelos em XML com a finalidade de codificar as informações de segurança dos usuários. Utilizando esquemas que sejam compreensíveis tanto por humanos como por máquinas.

- Define como essas mensagens serão inseridas nos protocolos de comunicação.

- Especifica precisamente certos processos de troca de mensagem que fazem parte dos casos de uso mais comuns, como o Web SSO.

- Dá suporte a vários mecanismos de proteção de privacidade, incluindo a habilidade de obter determinados atributos do usuário sem revelar suas identidades.

- Detalha como trabalhar com as informações de identidade em formatos e plataformas amplamente utilizados, como Unix, Windows, X.509, LDAP, DCE e XACML.

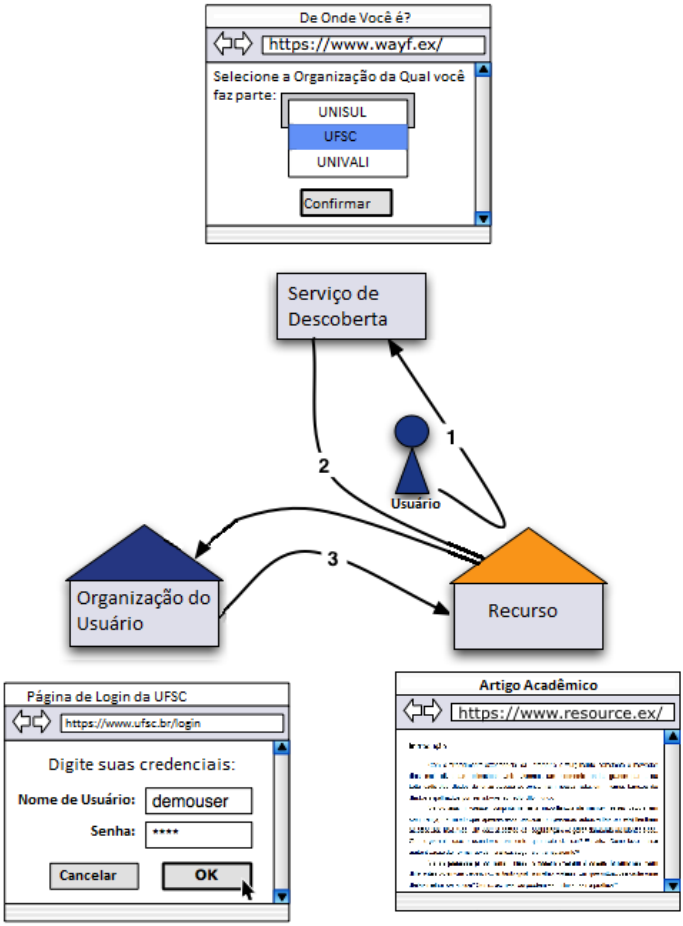

Figura 8 – Procedimento de login do Shibolleth

- Define também um esquema de metadados que informe com quais opções e funções de SAML cada sistema trabalha.

- Tem flexibilidade e uma grande extensibilidade, de modo que possa cumprir também requisitos que ainda não foram estabelecidos pelo padrão.

A utilização dos elementos provedor de serviço, provedor de identidade e sujeito no Shibboleth provém da SAML. É importante constar que em uma federação é bem provável a existência de mais de um Provedor de Serviço e de Identidade, e muitas organizações podem desempenhar os dois papéis simultaneamente, dependendo do caso de uso (COMMUNITY, 2011).

Na SAML, um elemento denominado Asserção (Assertion) é responsável por transportar as informações. Essa Asserção contém um Cabeçalho, o nome do Sujeito e uma ou mais Declarações (Statements). O Cabeçalho contém o nome do Provedor de Identidade e outras informações como o tempo em que foi gerado e o tempo de expiração. Para garantir a procedência da Asserção, esta pode ser assinada digitalmente pelo Provedor de Identidade, que pode escolher diferentes métodos para fazê-lo (COMMUNITY, 2011). Embora se possa definir outros, os dois tipos mais importantes de Declarações são:

- Declarações de Autenticação: Declaram que o Sujeito foi autenticado utilizando um determinado método num tempo e local específicos. A SAML tem expressividade para definir mais de 20 métodos de autenticação.

- Declarações de Atributo: Contém propriedades associadas ao Sujeito. A qual Grupo este sujeito pertence, ou qual (is) papel (éis) ele exerce são exemplos típicos desses atributos, no entanto qualquer outro tipo de propriedade, como dados financeiros, pode ser representada em uma Declaração de Atributo.

As Asserções costumam seguir o caminho do Provedor de Identidade para o Provedor do Serviço, entretanto há vários meios para a informação "viajar". O Provedor do Serviço pode se comunicar com o Provedor de Identidade diretamente através de um canal dedicado, ou então o próprio Sujeito pode levar as Asserções de um para o outro, ou também utilizar um terceiro elemento para intermediar este transporte. Ainda, em ambientes de Web Services, pode-se utilizar o SOAP para este fim, existem diretrizes específicas para isso (COMMUNITY, 2011).

A SAML define um conjunto de mensagens de requisições e respostas em XML que podem ser utilizadas pelo Provedor de Serviço para obter as Asserções diretamente. As requisições especificam o que o Provedor do Serviço quer, por exemplo "todos os atributos do João da Silva". As respostas irão retornar uma ou mais Asserções que estejam de acordo com a requisição. Para permitir que produtos diferentes trabalhem em conjunto, é também necessário especificar como os protocolos de rede irão carregar as requisições e respostas.

O padrão provê vários mecanismos úteis para dar suporte a um ambiente de identidades federadas. Um protocolo permite que o Provedor do Serviço determine para onde dirigir especificamente uma requisição particular de um cliente dentro dos possíveis Provedores de Identidade. Outro protocolo permite que dois provedores de identidade associem os dados que cada um possui para o mesmo usuário. Por exemplo, em um provedor de identidades o usuário pode ser "Zé da Silva"e no outro ele é "José Barros da Silva". No entanto essa associação, por razões de privacidade, requer autorização do usuário.

É possível também usar a proteção de privacidade, um identificador temporário para evitar a revelação da identidade dos Sujeitos. Para exemplificar: Um provedor de Identidade sabe que Asserções contendo o SujeitoXYZ se referem a Zé da Silva, o outro associaria esse XYZ com José Barros da Silva, mas nenhum deles conheceria o verdadeiro nome da conta que o outro utiliza. No dia seguinte um outro identificador temporário do Sujeito (diferente de XYZ) será utilizado para evitar que as outras entidades possam identificar um padrão de uso.

Além disso a SAML especifica um protocolo de logout bem rápido e leve que consegue informar todos os Provedores de Serviço e de Identidades que um usuário se desconectou. A intenção principal deste protocolo é liberar rapidamente os recursos que não serão mais utilizados, visto que o usuário utilizador dos mesmos já fez o logout. Outras funções do SAML incluem:

- Criptografia total ou parcial das Asserções.
- Especificação do consumidor de uma Asserção.

O padrão SAML ainda tem documentado vários critérios de conformidade relacionando combinações possíveis de funcionalidades. Há também um documento que discute e exemplifica o que foi levado em consideração no desenvolvimento, principalmente as questões de Segurança e Privacidade.

Em virtude dos fatos mencionados, pode-se afirmar que o SAML contém um excelente conjunto de mecanismos para auxiliar na imple-

mentação de um sistema de Gerenciamento de Identidades Federadas. A maneira como foi construído permite que a interoperabilidade dos sistemas aconteça de maneira clara, rápida, segura e bem padronizada. Sendo então uma ferramenta útil e poderosa na construção do Sistema Shibolleth como um todo.

5.4 ABORDAGENS DA FERRAMENTA

Nesta seção serão abordados os pontos principais da ferramenta Shibolleth, com objetivo de explanar quais técnicas e conceitos foram utilizados na sua implementação, dentro de cada ponto específico.

5.4.1 Identidades Federadas, Troca de Informação e Comunicação

Toda a operação do Sistema Shibolleth é baseada no padrão SAML (INITIATIVE, 2011), já discutido na seção anterior, que por sua vez é baseado no conceito de Identidades Federadas, este padrão (SAML) define o formato de cada mensagem trocada entre os elementos da Federação. Essas mensagens, por sua vez, são trocadas pela Internet e utilizam então basicamente protocolo IP, mas a já mencionada flexibilidade do SAML permite várias formas de comunicação de modo que os mais variados modelos de aplicações sejam atendidos (CAJU, 2005).

5.4.2 Autorização Baseada em Atributos

Os métodos básicos de autenticação de usuário apenas fornecem à aplicação uma identificação de usuário permanente da pessoa que recém se autenticou. Essa abordagem acaba sendo insuficiente em sistemas modernos, pois as aplicações precisam de mais informação sobre os usuários, os seja, seus atributos, para então poderem tomar decisões acertadas relativas à autorização. Prover essas informações como parte do processo de sign-on é muito útil, especialmente quando estamos trabalhando num ambiente misto, entre organizações distintas, onde as informações sobre o usuário se encontram na sua organização de origem e não estão acessíveis de fora dela (LAWRENCE, 2009).

O Shibolleth foi projetado especificamente para prover esses atributos do usuário para as aplicações com toda flexibilidade, extensibilidade, segurança e privacidade que são requeridas nesse cenário. As

organizações podem utilizar um esquema de atributos já incluso no Shibolleth, ou criar novos atributos de acordo com suas necessidades e aplicações (LAWRENCE, 2009). O Software do Provedor de Identidades (IdP) contido no Shibolleth foi implementado de forma que permite o encaixe com os serviços de gerenciamento de identidades e informações de usuários já existentes na organização, estendendo-os para que consigam também trabalhar em conjunto com elementos externos, ou seja, os sistemas das outras organizações.

5.4.3 Federações

Considerando a natureza flexível de padrões e softwares modernos, organizações interessadas em utilizar o Sistema Shibolleth devem fazer várias escolhas. Para conseguirem interoperar utilizando identidades federadas elas devem concordar em muitas questões técnicas, como (INICIATIVE, 2011):

- Mecanismos de segurança utilizados entre os servidores Shibolleth (normalmente a Infra-Estrutura de Chaves Públicas baseada no X.509, PKI).
- Definição dos atributos.
- O meio de localizar os servidores dos outros participantes da Fedaração.

Também devem concordar em políticas de alto-nível, como:

- A eficiência e nível de confiança nas suas próprias práticas de gerenciamento de usuários.
- Quais procedimentos utilizar para lidarem com informações pessoais sensíveis.
- Que tipos de organizações podem participar da Federação.

Apesar de não ser uma tarefa fácil conseguir a aprovação de todos esses aspectos entre um grande número de organizações e comunidades distintas. De forma comparativa, acaba sendo mais prático fazê-lo apenas uma vez e trabalhar com todos os parceiros da mesma forma, é melhor do que ter acordos distintos "dois-a-dois"com cada entidade.

O Sistema Shibolleth dá suporte a essas federações definindo formatos para gerenciar as configurações dos sites e providenciando os

procedimentos de criação, distribuição e importação dessa informação. Além disso, os Provedores de Serviço e Identidades implementados pelo Shibolleth ainda podem ser configurados para trabalharem com Federações distintas simultaneamente assim como acordos entre duas entidades. É possível, portanto, utilizar configurações avançadas, onde políticas distintas são ativadas em cada situação específica.

5.4.4 Desenvolvimento e Manutenção do Código Fonte

O Sistema Shibolleth, parte do Projeto Shibolleth, utiliza um processo aberto de projeto e desenvolvimento de software. Já recebeu e continua recebendo a contribuição de várias organizações acadêmicas ao redor do mundo, além de vários parceiros na Indústria. Ou seja, o Sistema Shibolleth é um software de código aberto. Seus termos de uso são definidos pela Licença de Software Apache, o que permite e promove uma ampla utilização e adoção tanto de outros softwares tanto livres como também proprietários (INICIATIVE, 2011).

5.5 FEDERAÇÕES UTILIZANDO SHIBBOLETH

5.5.1 CAFe Comunidade Acadêmica Federada

Figura 9 – Comunidade Academica Federada

A Comunidade Acadêmica Federada (CAFe) é uma federação que reúne instituições de ensino e pesquisa brasileiras. Através da CAFe, um usuário mantém todas as suas informações na instituição de origem e pode acessar serviços web oferecidos pelas instituições que participam da federação.

As instituições pertencentes à CAFe podem atuar como provedoras de identidade e como provedoras de serviço. A RNP é responsável pela gestão do serviço e por manter o repositório centralizado com dados sobre integrantes da federação.

A CAFe possibilita que cada usuário tenha uma conta única em

sua instituição de origem, válida para todos os serviços oferecidos à federação, eliminando a necessidade de múltiplas senhas de acesso e processos de cadastramento.

A relação de confiança entre instituições participantes da Federação permite que o usuário se autentique unicamente em sua instituição de origem, que fornece as garantias de autenticidade e credibilidade necessárias às demais.

Diversos países já têm federações em funcionamento ou em implantação. Dentro das redes de instituições de ensino, serviços de ensino a distância e atividades de colaboração estão entre os maiores beneficiários das infraestruturas oferecidas por federações.

As seguintes instituições já estão conectadas através do CAFe:

- Universidade Federal do Vale do São Francisco (UNIVASF)
- Universidade Federal do Rio Grande do Sul (UFRGS);
- Universidade Federal de Viçosa (UFV);
- Universidade Federal de Santa Catarina (UFSC);
- Universidade Federal do Pará (UFPA);
- Universidade Federal do Pernambuco (UFPE);
- Universidade de São Paulo (USP);
- Instituto Nacional de Matemática Pura e Aplicada (IMPA);
- Universidade Federal de Mato Grosso do Sul (UFMS);
- Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES);
- Universidade Federal de Minas Gerais (UFMG).

O exemplo do CAFe demonstra na prática como o Shibboleth é uma ferramenta robusta e poderosa para suprir as necessidades de uma grande federação.

6 OPENID

6.1 O QUE É OPENID

OpenID é um protocolo de autenticação descentralizado que possibilita que as pessoas inscrevam-se em várias contas pela internet. Elimina a necessidade de múltiplas autenticações em websites diferentes, simplificando a experiência online (FOUNDATION, 2004). OpenID não é uma tecnologia proprietária e é totalmente gratuíta.

OpenID possibilida um single sign-on dentro da Internet. OpenID é uma tecnologia centrada no usuário que permite uma pessoa ter controle e gerenciamento sobre sua identidade on-line. Por ser descentralizada não há um único servidor para serviços OpenID. Pelo contrário, as pessoas fazem sua escolha do provedor OpenID.

Uma função-chave suportada pela OpenID é a capacidade de uma pessoa ter "single sign-on"através de múltiplos serviços OpenID habilitados. A parte de autenticação fica ligada diretamente com o provedor OpenID escolhido. Isso significa que sites que implementam OpenID nunca sabem a real senha do usuário (ou outras credenciais). O benefício para os usuários é o aumento da segurança, nomeadamente através do emprego de uma abordagem forte como uma senha, utillizada apenas uma vez para acessar seu provedor, tendo assim uma experiência muito mais simples na web.

OpenID cresceu principalmente na comunidade de blogs onde havia uma exigência. Utilizar sua própria identificação do seu blog para comentar em blogs de terceiro.. OpenID foi originalmente desenvolvida para 9 milhões de usuários do site LiveJournal.com, Que acompanha desde entao a crescente ascensão deste protocolo na internet.

Embora tenha começado com a comunidade dos blogs, OpenID está começando a ser adotado em tecnologias como o Ruby on Rails, o framework Zend PHP, o framework Django e Python. Além disso, serviços como o Technorati, WordPress.com, 37signals, e Digg.com (em andamento) adicionou suporte para OpenID. Prestadores de serviços e grandes empresas como a AOL, Symantec, a VeriSign, Mozilla, Novell, Microsoft e Reebok também começaram a trabalhar com OpenID em vários produtos e serviços.

Na maioria dos casos, OpenID é uma URL, o que significa que a terceira parte pode facilmente usá-la para determinar a localização do provedor de OpenID, sem recorrer a algum tipo de serviço de diretório (como o Where Are You From (WAYF) serviço utilizado no

Shibboleth). Além disso, a formulação do OpenID como um URL, gera a vantagem de disponibilizar para a terceira parte informações sobre o proprietário da identidade. As URLs também são fáceis de lembrar e podem ser marcadas posteriormente, utilizando como exemplo o serviço de armazenamento de sites da internet com marcações por palavras-chaves, como o del.icio.us. Um exemplo disto pode ser visto no site Doxory.com , um serviço que tenta descobrir todos os amigos que você tenha listado via FOAF (Friend of a Friend) em sua URL OpenID. Por exemplo, se você entrar com um OpenID Doxory LiveJournal.com, Doxory mostra automaticamente qual dos seus amigos LiveJournal já estão usando o serviço. A combinação de OpenID e Microformats, como hCard também torna possível descobrir informações de contato público sobre o proprietário de uma OpenID (POWELL; RECORDON, 2011). Assim o OpenID começa a fornecer as peças do núcleo de infra-estrutura necessárias para integrar verdadeiramente as redes sociais.

OpenID é um padrão relativamente simples, que faz um trabalho relativamente simples - um ambiente single sign-on distribuído e livre. Ele não tenta fornecer soluções completas em autorização distribuída, mas é uma tecnologia que permite que estes tipos de serviços sejam construídos. Por exemplo, suporta a transferência de atributos (informações sobre o dono do OpenID) em que as decisões sobre a confiança e autorização podem se basear. Isto significa também que os utilizadores podem facilmente compartilhar informações tais como seu endereço de e-mail, nickname, ou código postal, quando existem logins OpenID em sites habilitados, se optarem por fazê-lo.

6.2 QUESTÕES EM ABERTO

Como qualquer padrão na área de acesso e gerenciamento de identidade, OpenID tem suas questões em aberto - e seus pontos críticos. Um potencial problema reside na área conhecida como phishing. Em um ataque de *phishing*, um prestador de serviços apresenta uma caixa de login em suas páginas Web que leva o usuário para uma página que se parece com a de seu provedor de OpenID, mas que na verdade é um serviço oferecido por outra pessoa. Esse serviço poderia recolher as senhas de usuários desavisados do OpenID (OLLMAN, 2006).

Houve algumas tentativas para contornar este problema, em parte, contando com a educação do usuário sobre tais perigos, o uso de HTTPS ao invés de HTTP simples para se comunicar com o provedor

e marcas d'água específicas do usuário ou outras imagens embutidas na página de provedor de identidade. Recentemente, alguns provedores de OpenID têm oferecido apoio a mais certificados de navegadorres para contornar o problema do *phishing*. Além disso, o recém-lançado MyVidoop.com é projetado para ajudar a resolver o problema do phishing com OpenID, mudando a forma de autenticar os usuários online, removendo senhas tradicionais. VeriSign também demonstrou uma extensão do Firefox, a ser lançado na Internet Identity Workshop em Maio de 2007 , que visa ajudar a resolver essas questões dentro do próprio navegador.

6.3 OPÇÕES DE EXECUÇÃO

Vários kits de ferramentas e bibliotecas estão disponíveis para ajudar com a implementação do OpenID, tanto para provedores de Identidade como para as Partes Confiantes, em uma enorme variedade de linguagens de programação.

Para o Servidor Apache 2, temos o modulo mod auth openid. Em ColdFusion temos as blibliotecas para o consumidores: OpenID CFC, CFOpenID, CFKit OpenID.

Para Java temos o JOpenID, este é uma ferramenta desevolvida sobre a versão 5 do Java e permite a implementação para assinar o contrato OpenID. Foi licenciado através da licença Apache.

Outra linguagem que está bastante avançada com o suporte ao desenvolvimento é o PHP. Um exemplo disso é o suporte do framework Zend ao OpenID. A estrutura do Zend OpenID se apresenta da seguinte forma: Dois pacotes, O primeiro é o Zend Openid Cosumidor, para os desenvolvedores de sites habilitados pelo OpenID. O segundo é o Zend Openid Provider, para os desenvolvedores da parte servidor do OpenID. Estes são completamente independentes um do outro, podendo assim podem ser utilizados de forma separada. Zend Openid utiliza a extersão GMP, para melhor desempenho.

6.4 EXEMPLO PRÁTICO OPENID UTILIZANDO A BIBLIOTECA OPENID4JAVA

Antes de descrevermos o exemplo propriamente dito, iremos descrever alguns conceitos envolvidos diretamente nesta aplicação exemplo.

6.4.1 OpenID Autenticação

O mecanismo de autenticação é a parte principal do OpenID e consiste em três conceitos principais:

- Identificador OpenID: Um texto que identifica unicamente um usuário. Não existe 2 usuários com a mesma String de identificação. Este identificador é normalizado pela OpenID Authentication Specification Version 2.0. Esta normalização ajuda o processo de autenticação a reconhecer um identificador válido. Dentro dos identificadores temos 2 sub-divisões:

  - o Identifcadores de usuários: São os identificadores fornecidos pelos usuários sem qualquer normalização ou formatação.
  - o Claimed Identificador: Este é o identificador já normalizado e formatado por alguma ferramenta. Este que será fornecido aos provedores no momento da "descoberta".

- OpenID Relying Party (RP): Um recurso online (geralmente um web site, mas também podendo ser um arquivo, uma imagem entre outros) que utiliza o identificador OpenID para definir quem pode acessá-lo. Um RP não sabe como e tão pouco se preocupa de como o Identificador Claimed foi autenticado. Este apenas quer saber se o provedor autenticou com sucesso ou não o usuário. Sendo assim, a função de um RP fica sendo especificamente de permitir ou não o acesso ao conteúdo protegido.

- Provedor OpenID: Um web site onde usuários podem acessá-lo e conseqüentemente fazer sua autenticação para se beneficiarem de qualquer RP. Também é papel de um Provedor disponibilizar e autenticar identificadores. Podendo também ser local de gerenciamento desses identificadores. Esta gerência inclui manipular atributos como: email, nome completo, data de aniversário e endereço. O mecanismo de autenticação funciona da seguinte maneira. Uma página de login e apresentada para o usuário, o usuário informa seu identificador e senha; O provedor faz a verificação e autenticação destas informações; Autenticando com sucesso o provedor redireciona o navegador para o RP, caso contrário, informa ao usuário uma mensagem de erro de autenticação.

6.4.2 Sobre a Aplicação

Esta aplicação exemplo consiste em uma construção do tipo WAR, podendo ser executada localmente em qualquer contêiner de aplicação Java como o Tomcat. O foco do exemplo pode ser desmembrado nos seguintes pontos:

- O usuário informa seu identificador OpenID em uma página de registro.
- A aplicação verifica o identificador informado pelo usuário, esta verificação é feita redirecionando o usuário para a página de login de seu provedor OpenID.
- Caso a verificação seja autenticada com sucesso, a aplicação faz a busca do perfil do usuário dentro do provedor OpenID e redireciona para a página de visualização de perfil dentro da aplicação.
- Após validados os dados do perfil o usuário finaliza o cadastro salvando os dados.

6.4.3 Obtendo o Recurso On-line (RP)

Um dos principais pontos da autenticação é provar que o usuário é verdadeiramente ele. Após esta autenticação, outro ponto importante é prover a este usuário os privilégios corretos. Esta aplicação exemplo executa uma função comum na maioria dos Web Sites; cadastro de usuários. Esta assume que se o usuário pode prover sua identificação, então o mesmo pode se registrar no sistema. Segue abaixo os passos básicos para executarmos esta premissa:

1. Obter o identificador do usuário: A aplicação solicita através da página de registro a identificação do usuário.
2. Descoberta: A aplicação normaliza o identificador do usuário para determinar como será feito o contato entre a aplicação e o provedor OpenID.
3. Associação: Este passo é opcional, porém altamente recomendado. Neste ponto, é estabelecido um canal de comunicação seguro entre a aplicação (RP) e o provedor OpenID.

4. Solicitação da Autenticação: A aplicação (RP) solicita ao provedor OpenID que autentique o identificador.

5. Verificação: A aplicação verifica o retorno do provedor OpenID e se a comunicação não foi adulterada.

6. Retorno para a aplicação: Após a autenticação, o usuário é direcionado para o recurso solicitado inicialmente.

6.4.3.1 Obter o identificador do usuário

Este passo é executado pela aplicação (RP). Basicamente, é exibido uma tela no qual o usuário poderá inserir seu identidicador OpenID, como podemos ver na Figura 10.

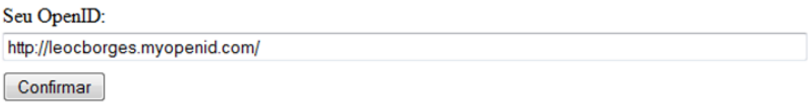

Figura 10 – Identificador OpenID

Nesta parte, a aplicação deve se preocupar com três coisas. Validar se o dado inserido é uma String e submeter à String para a camada de negócio para dar continuidade ao processo e após isto, redirecionar para a tela de login do provedor OpenID. O código abaixo mostra o ManagedBean executando estás ações:

Code 6.1 – ManagedBean.java

```
1  /**
2           * Inicia o processo de autenticação.
3           */
4          public String confirmaOpenID() {
5                  try {
6
7                          AuthRequest authRequest = this.
                               openIDBBusiness
8                                                      .
                                                           autenticaIdentificador
                                                           (openid);
9
10                         // Now take the AuthRequest and
                               forward it on to the OP
11                         FacesContext faces = FacesContext.
                               getCurrentInstance();
```

```
12                          ExternalContext context = faces.
                                getExternalContext();
13                          context.redirect(authRequest.
                                getDestinationUrl(true));
14
15                  } catch (Throwable e) {
16                          super.exibiMensagemErro(
                                MensagensConstants.ERROR001,
17                                      MensagensConstants.
                                            ERROR001, e);
18                  }
19
20                  // carrega a pagina de perfil
21                  return "identificadorok";
22          }
```

Abaixo temos o código da camada de negócio que chama os serviços da classe RegistrationService.java para efetuar a Descoberta do Identificador e criar o objeto AuthRequest. Que será utilizado para fazer a requisição junto ao provedor OpenID.

Code 6.2 – AutenticacaoBusiness.java

```
1           /**
2            * Descobre, coloca na sessão o objeto
                 DiscoveryInformation e cria o request
3            * para autenticaçao.
4            *
5            * @return
6            */
7           public AuthRequest autenticaIdentificador(String
                openid) {
8                   DiscoveryInformation discoveryInformation =
                        RegistrationService
9                                       .
                                            performDiscoveryOnUserSuppliedId
                                            (openid);
10
11                  // seta na sessao o objeto
                        discoveryInformation
12                  HttpSession session = (HttpSession)
                        FacesContext.getCurrentInstance()
13                                      .getExternalContext().
                                            getSession(false);
14                  session.setAttribute("discoveryInformation",
                         discoveryInformation);
15
16                  // Create the AuthRequest
17                  AuthRequest authRequest =
                        RegistrationService.
                        createOpenIdAuthRequest(
```

```
18                                      discoveryInformation,
                                            RegistrationService.
                                            getReturnToUrl());
19
20                  return authRequest;
21          }
```

6.4.3.2 Descoberta

Nesta etapa o identificador é convertido para um formato que pode ser usado para determinar duas coisas: quem é o provedor OpenID e como vai manter contato com este provedor OpenID.

O processo de descoberta é usado pelo RP para determinar como fazer solicitações ao provedor OpenID. Mas antes que o identificador do usuário fornecido possa ser utilizado para a descoberta, ele deve ser normalizado. No nosso caso, a biblioteca reconhece o myopenid.com como sendo o provedor OpenID automaticamente. A biblioteca openid4java realmente faz o trabalho pesado para normalizar o identificador fornecido pelo usuário, segue abaixo o código da Descoberta:

Code 6.3 – RegistrationService.java

```
1  public static DiscoveryInformation
       performDiscoveryOnUserSuppliedIdentifier(
2                       String userSuppliedIdentifier) {
3                   DiscoveryInformation ret = null;
4                   //
5                   ConsumerManager consumerManager =
                        getConsumerManager();
6                   try {
7                           // Perform discover on the User-
                                Supplied Identifier
8                           List<DiscoveryInformation>
                                discoveries = consumerManager
9                                       .discover(
                                            userSuppliedIdentifier
                                            );
10                          // Pass the discoveries to the
                                associate() method...
11                          ret = consumerManager.associate(
                                discoveries);
12
13                  } catch (DiscoveryException e) {
14                          String message = "Error occurred
                                during discovery!";
15                          log.error(message, e);
16                          throw new RuntimeException(message,
```

```
                                    e);
17                  }
18                  return ret;
19  }
```

A classe no centro desse processo é a ConsumerManager. A biblioteca openid4java faz severas orientações quanto ao uso desta classe. Em apenas uma linha de código, a biblioteca permite que seu código seja normalizado (código em negrito). Este método retorna um java.util.List de objetos DiscoveryInformation. Estes objetos serão utilizados posteriormente no processo de Associação.

6.4.3.3 Associação

A associação é feita quando a parte confiante e o provedor OpenID estabilizam um segredo compartilhado (através do Diffie-Hellman Key Exchange), tornando a conexão mais segura e confiável. Associação é requerida pela especificação OpenID. O código abaixo mostra a classe de serviço fazendo a associação chamando apenas um método, "associate()"

Code 6.4 – RegistrationService.java

```
1   public static
2      DiscoveryInformation
           performDiscoveryOnUserSuppliedIdentifier(
3          String userSuppliedIdentifier) {
4
5     DiscoveryInformation ret = null;
6     ConsumerManager consumerManager = getConsumerManager();
7     try {
8       // Perform discover on the User-Supplied Identifier
9       List<DiscoveryInformation> discoveries =
10        consumerManager.discover(userSuppliedIdentifier);
11      // Pass the discoveries to the associate() method...
12      ret = consumerManager.associate(discoveries);
13    } catch (DiscoveryException e) {
14      String message = "Error occurred during discovery!";
15      log.error(message, e);
16      throw new RuntimeException(message, e);
17    }
18    return ret;
19  }
```

Este método retorna um objeto DiscoveryInformation que descreve o resultado da descoberta. A aplicação guarda este objeto na sessão para utilizar postariormente na requisição da autenticação.

6.4.3.4 Autenticação

Após a parte confiante ter feito com sucesso a descoberta do identificador do usuário, é feita a autenticação do usuário. "ConsumerManager"é chamado para montar o objeto AuthResquest que será utilizado pelo provedor OpenID para proceder com a autenticação.

Dentro deste método, também é criada uma extensão OpenID, chamada "SimpleRegistration". Esta extensão permite que a parte confiante obtenha certos atributos do perfil do usuário junto ao provedor OpenID. O código abaixo mostra a instanciação dos objeto AuthRequest e o "SRegRequest"(uma extensão OpenID).

Code 6.5 – RegistrationService.java

```
1  public static AuthRequest
2  createOpenIdAuthRequest(DiscoveryInformation
3  discoveryInformation, String returnToUrl) {
4    AuthRequest ret = null;
5    //
6    try {
7      // Create the AuthRequest object
8     ret =
9  getConsumerManager().authenticate(discoveryInformation,
10                                                  returnToUrl);
11     // Create the Simple Registration Request
12    SRegRequest sRegRequest =
13 SRegRequest.createFetchRequest();
14     sRegRequest.addAttribute("email", false);
15     sRegRequest.addAttribute("fullname", false);
16     sRegRequest.addAttribute("dob", false);
17     sRegRequest.addAttribute("postcode", false);
18     ret.addExtension(sRegRequest);
19   } catch (Exception e) {
20     String message = "Exception occurred while building " +
21                      "AuthRequest object!";
22     log.error(message, e);
23     throw new RuntimeException(message, e);
24   }
25   return ret;
26 }
```

A primeira linha em negrito não faz propriamente a chamada da autenticação. Apenas pega o objeto Discoveryinformation retornado da descoberta e a url para onde o usuário deve ser redirecionado. A segunda linha em negrito cria o objeto Sreg através da chamado do método SregRequest.createFetchRequest(). Então os atributos que você quer recuperar devem ser adicionados através do método addAtribute().

Feito isto, o objeto extendido é adicionado ao AuthRequest pelo método addExtension();

Após estas ações, o usuário é redirecionado para o provedor OpenID que é responsável pela autenticação do usuário. A figura 11 mostra a tela de autenticação do provedor "www.myopenid.com"

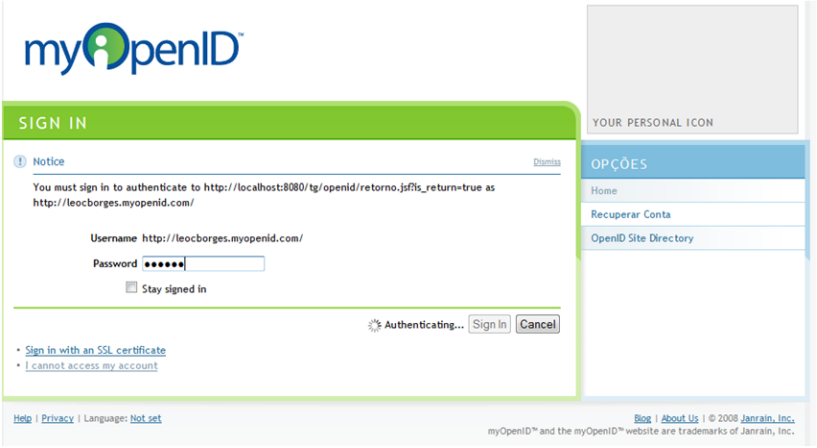

Figura 11 – MyOpenId

6.4.3.5 Verificação

O código abaixo mostra o tratamento do retorno à aplicação. O fato mais importante a ser analisado neste código, é a passagem dos parâmetros do request para o método que irá fazer o tratamento dos atributos.

Code 6.6 – RetornoBean1.java

```
1  public RetornoBean() {
2                  HttpSession session = (HttpSession)
                       FacesContext.getCurrentInstance()
3                                  .getExternalContext().
                                       getSession(false);
4
5                  Map<String, String> parametros =
                       FacesContext.getCurrentInstance()
6                                  .getExternalContext().
                                       getRequestParameterMap()
                                       ;
```

```
7
8                   DiscoveryInformation discoveryInformation =
                        (DiscoveryInformation) session
9                               .getAttribute("
                                    discoveryInformation");
10
11                  RegistrationService.processReturn(
                        discoveryInformation, parametros,
12                              RegistrationService.
                                    getReturnToUrl());
13
14  }
```

6.4.3.6 Retorno a aplicação

A resposta da autenticação é verificada, caso a autenticação tenha sido feita com sucesso. O recurso protegido é liberado para o usuário. No nosso caso, esse recurso é a tela de registro. O código abaixo exibe os processos de verificação, obtenção dos dados solicitados e o preenchimento do objeto que representa o usuário na aplicação.

Code 6.7 – RegistrationService.java

```
1   public static String processReturn(
2           DiscoveryInformation discoveryInformation,
3           Map<String, String> pageParameters, String
                returnToUrl) {
4
5           String nome = "1";
6           // Verify the Information returned from the OP
7           // / This is required according to the spec
8           ParameterList response = new ParameterList(
                pageParameters);
9           try {
10                  VerificationResult verificationResult =
                        getConsumerManager()
11                              .verify(returnToUrl,
                                    response,
                                    discoveryInformation);
12                  Identifier verifiedIdentifier =
                        verificationResult.getVerifiedId();
13                  if (verifiedIdentifier != null) {
14                          AuthSuccess authSuccess = (
                                AuthSuccess) verificationResult
15                                      .getAuthResponse();
16                          if (authSuccess.hasExtension(
                                SRegMessage.OPENID_NS_SREG)) {
17                                  MessageExtension extension =
                                        authSuccess
```

```
18                                                  .
                                                        getExtension
                                                        (
                                                        SRegMessage
                                                        .
                                                        OPENID_NS_SREG
                                                        );
19                                  if (extension instanceof
                                        SRegResponse) {
20                                                  SRegResponse
                                                        sRegResponse
                                                        = (
                                                        SRegResponse
                                                        )
                                                        extension
                                                        ;
21                                              nome = sRegResponse.
                                                    getAttributeValue
                                                    ("email");
22                                  }
23                          }
24                  }
25
26                  } catch (Exception e) {
27                          String message = "Exception occurred
                                while verifying response!";
28                          log.error(message, e);
29                          throw new RuntimeException(message,
                                e);
30                  }
31                  return nome;
32  }
```

7 OAUTH

7.1 O QUE É O OAUTH

Alguns carros de luxo possuem uma chave específica para terceiros, que permite dirigir o carro por pequenas distâncias, porém não dá acesso ao porta malas e nem outros recursos pessoais, como o telefone do veículo. Isso serve, por exemplo, para entregar o carro a um manobrista. Assim o dono do automóvel fornecerá a chave especial e terá toda segurança de que o funcionário do local não fará nada com o carro além de estacioná-lo. Os desenvolvedores que pensaram no OAuth quiseram trazer uma idéia parecida para a Web, visto que situações análogas ao exemplo citado acima acontecem também no ambiente virtual.

Atualmente os websites são ligados por muito mais do que hyperlinks, eles efetivamente compartilham dados, informações, preferências e até os próprios usuários. É comum você estar visitando um blog aleatório, depara-se com um artigo que o agrada, e tem o desejo de compartilhá-lo com os seus amigos. Pensando nisso o mantenedor do blog já disponibilizou um botão que permite o compartilhamento desse conteúdo diretamente com seus amigos de uma determinada rede social. É uma grande facilidade, mas para fazer isso por você, esse blog precisa ter acesso a sua conta na rede social, e consequentemente suas credenciais do serviço. Como garantir que esse Blog vai apenas compartilhar o conteúdo com seus amigos? Como garantir que ele não armazenará a sua senha para fins maliciosos?

Os desenvolvedores do OAuth pensam que a melhor forma de garantir a segurança num caso como esse é simplesmente não entregar a senha para o Blog (STREPPONE, 2010). O problema agora é resolver como o Blog ainda assim poderá agir no lugar do seu Leitor. Aí entra a analogia com a chave especial dos carros de luxo, o OAuth é algo como essa chave especial, que permitirá (no nosso exemplo) ao Blog utilizar a conta da rede social do seu leitor, mas com poderes limitados. Como implementar algo assim? Uma chave especial para um carro de luxo é fácil de se imaginar, ela deve possuir informações dentro dela que façam o carro reconhecer a situação e habilitar no seu computador interno somente as funções necessárias para o manobrista. Felizmente o OAuth de certa forma segue este princípio na sua implementação.

A conexão entre o Blog e o Leitor é feita no modelo Cliente-Servidor, o Leitor faz simples requisições HTTP do seu Browser e recebe

o HTML das páginas do Blog como resposta. A partir do momento em que este usuário deseja que seus amigos da Rede Social também tenham acesso a este conteúdo, entra um terceiro elemento no processo, que é o próprio servidor da Rede Social. Dentro da arquitetura do OAuth, cada um desses agentes terá um nome específico, que serão utilizados genericamente nas próximas seções:

- Cliente: O Cliente agora é o servidor do Blog, que precisa acessar um serviço externo, no caso a Rede Social.

- Servidor: É a Rede Social, possuidora dos recursos que o Blog agora precisa acessar, no entanto não tem a autorização para fornecer esses recursos.

- Dono do Recurso: É o leitor do Blog, que também possui uma conta na rede social e precisa autorizar a comunicação entre os dois a fim de que sua necessidade (compartilhar com os amigos) seja satisfeita. De forma que o cliente não tenha acesso as credenciais do dono do recurso naquele servidor, a autorização de acesso é feita pelo meio de tokens e chaves secretas compartilhadas. Assim a segurança é garantida, pois um token de acesso pode ter seu escopo e poderes reduzidos, coisa que não pode ser feita com as credenciais originais do usuário.

7.2 HISTÓRIA DO OAUTH

O OAuth começou a ser pensado a partir de um cuidadoso estudo sobre protocolos semelhantes já existentes como o Google Authsub, Yahoo BBAuth e Flicker API. Então podemos afirmar que o objetivo dos criadores do OAuth foi levantar todo o conhecimento combinado das ferramentas existentes, buscando trazer o que cada uma tem de melhor, integrando todas as boas práticas ao seu produto. Essa é a versão resumida, mas é importante conhecer os detalhes dessa história, para entender melhor o "como"e o "por que"do OAuth ter se tornado o que é hoje.

Tudo começou quando Blaine Cook (na época trabalhando como Arquiteto Chefe do Twitter) estudava possibilidades de melhorar a API do Twitter, de forma que seus usuários não necessitassem compartilhar seus "nomes de usuário"e senhas com aplicações externas. Iniciou sua busca estudando o OpenID, mas não encontrou o que necessitava, como ele próprio explica:

"Queremos algo como o Flicker Auth / Google AuthSub / Yahoo! BBAuth, mas publicado como um padrão aberto, com bibliotecas comuns para clientes e servidores e etc... O complicado do OpenId é que com ele os usuários não tem mais senhas, então você não pode usar a autenticação simples para chamadas de API sem precisar de senhas (confrontando um dos pontos principais do OpenId) ou entregando tokens copiados e colados."(COOK, 2010)

Ao mesmo tempo, Chris Messina estava trabalhando em conjunto com Larry Half e o time do Ma.gnolia para integrar o OpenID aos Ma.gnolia Dashboard Widgets. Eles acabaram entrando em contato com Blaine, o que acabou levando a organização de um encontro maior que trouxe também David Recordon (co-criador do OpenID) e outras pessoas envolvidas para discutir as soluções de autenticação existentes.

Fizeram então uma revisão das funcionalidades do OpenID, levantaram todas as limitações do seu design e também estudaram outros protocolos proprietários. Com isso o grupo chegou a conclusão de que realmente era necessária a criação de um novo padrão aberto de controle de acesso à APIs, que não compartilhasse nomes de usuário e senhas, e que fosse agnóstico em relação ao login, permitindo que tecnologias como o OpenID pudessem então serem usadas com chamadas à APIs. A iniciativa recebeu o nome de OpenAuth, pois não era um substituto ao OpenID, mas sim um protocolo que seria de fato utilizado internamente pelo OpenID, só que ainda poderia também ser utilizado por qualquer desenvolvedor que quisesse permitir acesso externo à suas aplicações.

Foi criado um Google Group para iniciar o desenvolvimento deste novo protocolo, no entanto, pelo fato da AOL já ter criado anteriormente um protocolo com o mesmo nome OpenAuth, seus criadores (naquele momento eram Blaine Cook, Kellen Elliott-McCrea, Larry Haiff, Tara Hunt, Ian McKeller, Chris Messina e alguns outros) tiveram que mudar o nome, reduzindo a palavra "Open"a somente "O", ficando então OAuth, o qual se pronuncia "Ou-Auff"(na verdade o "ff"é uma aproximação, pois o som do "th"em inglês não possui equivalência em língua portuguesa).

Ao verificar que o projeto estava pronto para receber o apoio do público geral, em 10 de Julho de 2007 o OAuth Google Group foi criado e aberto a contribuição de qualquer um que tivesse interesse (VARIOUS, 2011). Em 15 de Julho, David Recordon já havia juntado todo o material produzido até então, que se transformou no rascunho da primeira especificação oficial do protocolo. A partir daí, com a importante colaboração de vários grupos e companias ao longo do tempo,

o OAuth foi crescendo, ganhando forma e identidade. Hoje em dia é amplamente usado em várias soluções reais, como o Facebook, Twitter, Orkut e outros serviços do Google.

7.3 CONCEITOS IMPORTANTES NO CONTEXTO DO OAUTH

Antes de entrar em detalhes relativos ao funcionamento do OAuth, é importante introduzir mais alguns conceitos que fazem parte desta tecnologia. Além dos conceitos de Cliente, Servidor e Dono do Recurso (já introduzidos na seção inicial deste capítulo), temos então:

- Recursos Protegidos que são recursos que estão armazenados no Servidor, mas são de propriedade e/ou controlados pelo Dono do Recurso. Então, apesar dos dados estarem no Servidor, para acessá-los é necessário, além da conexão com o servidor, a autorização do Dono do Recurso. A implementação desse controle de acesso e autorização normalmente fica a cargo do Servidor. Um recurso protegido pode ser dados (fotos, documentos, contatos), serviços (postar um item num blog, transferir dinheiro). Apesar de existir a possibilidade de utilizar o OAuth com outros protocolos da camada de transporte, ele só é formalmente definido para o protocolo HTTP(S).

- 2-Legged, 3-Legged, n-Legged que são as várias formas de autenticação possíveis com OAuth, o número de pernas ("legs") se refere ao número de partes que participam ativamente do processo de autenticação. No exemplo descrito na seção 7.4 temos um Blog (Cliente), uma rede social (Servidor) e um leitor do blog (Dono do Recurso) que precisa autorizar ao Blog o acesso a recursos que ele controla dentro da rede social, então fica claro que nesse caso utilizaremos autorização 3-legged, por se tratar de 3 elementos. Mas em casos específicos (como o próprio exemplo de implementação do OAuth com Orkut que abordaremos mais adiante neste trabalho), o Cliente pode ser também o Dono do Recurso, então neste caso é utilizado o modelo de autenticação 2-Legged.

- Credenciais e Tokens são utilizados pelo OAuth em três tipos principais: credenciais do cliente, credenciais temporárias e credenciais de token. As credenciais do cliente são usadas para autenticar o cliente. Isso permite que o servidor possa coletar dados

> sobre quais aplicações estão acessando seus serviços, dar um tratamento especial para clientes específicos se for necessário, fornecer mais informações ao dono do recurso sobre quem está buscando acesso ao seus recursos protegidos (ZERVAAS, 2010).

Credenciais de token são usadas no lugar do nome de usuário e senha do dono do recurso (STREPPONE, 2010). Como já citado anteriormente, um dos grandes trunfos do OAuth é permitir que as credenciais originais dos usuários não precisem ser compartilhadas com mais ninguém alem do dono do recurso e o servidor onde esses recursos estão disponíveis. O token então autoriza o servidor a fornecer uma classe especial de credencial.

Essas credenciais de token são compostas primeiro pelo identificador, que geralmente é uma string única de caracteres, difícil de se compreender e muito menos adivinhar, que é pareada com uma chave secreta que o impede de ser usado por entidades não autorizadas. Por fim, os tokens são credenciais normalmente limitadas em tempo e escopo, e podem ser desfeitas a qualquer momento tanto pelo servidor como pelo dono do recurso (ZERVAAS, 2010).

O processo de autorização também utiliza credenciais temporárias que identificam uma única requisição de autorização. Esse tipo de credencial oferece uma segurança extra e flexibilidade ao se trabalhar com os mais variados tipos de clientes.

7.4 EXEMPLO DO FLUXO DE FUNCIONAMENTO DO OAUTH

Nesta seção serão abordados todos os aspectos técnicos do OAuth na forma de um exemplo prático, que apesar de fictício, representa muito bem várias das utilizações do "mundo real", utilizando a autenticação 3-Legged. Será tomada como base o exemplo da seção 7.1, estendendo-o com dinâmica e características dos seus agentes e processos. O objetivo dessa história é demonstrar como todos os envolvidos são beneficiados com a utilização de um protocolo aberto e seguro, bem como facilitar o entendimento do processo de autenticação do OAuth.

José é um rapaz jovem, na casa dos vinte anos, que é apaixonado por jogos eletrônicos. Esse hobby, ao contrário do que muitos erroneamente pensam, o levou a fazer vários amigos que conheceu tanto dentro dos jogos online como na rede social YouAndI, onde participa de várias comunidades e fóruns dedicados aos amantes deste tipo de entretenimento. Como todo bom "gamer"(como os jogadores gostam de serem chamados), José costuma navegar na Internet por blogs e sites especi-

alizados atrás de novidades que permeiam seu hobby.

Ao se deparar com a última notícia sobre um novo jogo de Playstation 3 no blog da Sony Computer Entertainment, ele imediatamente sente a necessidade de compartilhar esse conteúdo com todos seus amigos "gamers". Felizmente os desenvolvedores do Blog em questão já haviam percebido que muitos dos seus leitores também eram membros dessa e de outras redes sociais. Pensando no potencial de propaganda gratuita que esses poderiam fazer, o "Staff"do Blog desenvolveu uma integração do seu sistema com o da rede YouAndI que permite aos seus leitores o compartilhamento automático do conteúdo do blog com os amigos do usuário em questão na citada rede.

Para isso, antes de tudo, os desenvolvedores do blog entaram em contato com a área de desenvolvedores da rede, e fizeram um pedido para obter uma Credencial de Cliente, sendo assim a rede YouAndI, a cada requisição feita pelo Blog da Sony Computer Entertainment, já sabe com certeza que a requisição partiu deste Cliente, mas também precisará saber de qual usuário da rede (Dono do Recurso) os recursos serão utilizados.

Então prosseguindo com a estória, com toda a estrutura já montada, o leitor encontrará no Blog um ícone da rede social, cuja legenda é "Compartilhe com seus amigos", ao clicar ali, será solicitado o seu usuário e senha da rede social. No entanto, esta tela não foi gerada pelo Blog, e sim pela Rede Social, os dados digitados ali serão enviados para a rede social. Ao autenticar o Dono do Recurso, o Servidor (YouAndI) enviará para o Cliente (Blog) credenciais do Dono do Recurso na forma de um Token, com o qual o Blog poderá autenticar suas requisições à API da rede para realizar serviços no lugar do Dono do Recurso.

Então, com toda a facilidade e segurança, o leitor do Blog agora tem seu conteúdo automaticamente compartilhado com todos seus amigos da rede social. Os desenvolvedores da Rede Social podem estender as possibilidades de interação de seus usuários sem precisar eles mesmos desenvolverem. E os Blogs tem seu conteúdo mais divulgado e consequentemente acessado por mais pessoas. É um processo onde todos os envolvidos saem ganhando.

7.5 EXEMPLO DE IMPLEMENTAÇÃO, UMA APLICAÇÃO FICTÍCIA DE COMPRA COLETIVA PARA O ORKUT

Nesta seção será explorada a utilização prática do OAuth. O exemplo fará uma autenticação do tipo 2-Legged, partindo de um apli-

cativo executado dentro do Orkut (Orkut App), visando a recuperação de dados de um usuário do Orkut em um site externo onde também possui um registro. Os códigos-fontes completos encontram-se nos Apêndices deste trabalho, ao longo do texto encontram-se enfatizados os trechos explanados em cada momento.

Por ser um protocolo aberto e amplamente utilizado, já existem várias implementações do OAuth disponíveis em várias linguagens de programação. Muitas vezes as próprias APIs dos serviços compatíveis já trazem embutida a implementação do OAuth, sendo uma grande facilidade para o programador, que só precisará então se preocupar com os problemas referentes a lógica do seu próprio negócio.

A aplicação fictícia "Descontos no Orkut"ilustrará bem a flexibilidade do OAuth, utilizando duas aplicações fictícias que fazem parte da mesma organização cujo nome fantasia é "Descontos", ambas criadas apenas para o propósito deste estudo. A idéia da aplicação se baseia no conceito de compra coletiva, sendo um website onde o usuário pode adquirir diversos produtos e serviços com descontos especiais.

Para chamar mais usuários com um meio de propaganda gratuita, os supostos desenvolvedores deste website fictício resolveram desenvolver também uma aplicação para Orkut onde seus usuários (que também tiverem uma conta no Orkut) podem visualizar os produtos que desejam adquirir (ou favoritos) e também exibi-los para seus amigos em seus perfis da rede social.

Considerando que o usuário já está autenticado no Orkut, a própria rede social já tem a capacidade de garantir a identidade deste usuário. Então, para que o usuário não necessite digitar sua senha do "Descontos"cada vez que quiser utilizar este serviço de dentro do Orkut, a sessão logada do Orkut servirá de credencial também para o site externo do "Descontos". No entanto, para que isso funcione, é necessária então uma associação prévia entre um registro de usuário no "Descontos"e um registro de usuário no Orkut.

Iniciando a aplicação do Orkut, se faz necessário primeiramente uma breve explanação sobre o funcionamento de um Orkut App.

Aplicativos para o Orkut são adendos à rede social, uma forma de seus mantenedores estenderem suas funcionalidades, delegando este trabalho à criatividade e competência de outros desenvolvedores (GOOGLE, 2011). Tecnicamente são outras páginas embutidas no Orkut através de um Iframe, essa prática, pela política de "Mesmo Domínio"dos Browsers (W3C, 2011) impede que as páginas dentro do frame tenham acesso a elementos não autorizados no site principal. Só que essa prática acaba bloqueando o acesso a todos os dados da rede social, então para con-

Figura 12 – OpenSocial

tornar esse problema, e controlar o acesso das aplicações aos recursos da rede social, é fornecida uma API Javascript chamada OpenSocial (Figura 12), que é inserida ao código do Orkut App no momento da execução. Esta API traz várias funções que vão desde buscar o nome dos amigos do usuário no Orkut até a realização de requisições para domínios externos, tudo acessível pela linguagem Javascript nas páginas que compõe a aplicação.

Na implementação foi utilizada a IDE Eclipse (FOUNDATION, 2011) com o plugin do OpenSocialOpenSocialPlugin, onde foram criados dois projetos, um projeto web dinâmico (Java EE com Servlets e JSPs) e um projeto OpenSocial (XML, HTML e Javascript). A explanação se iniciará pelo lado da aplicação OpenSocial:

Code 7.1 – Header do XML da aplicação

```
1 <?xml version="1.0" encoding="UTF-8"?>
2  <Module>
3   <ModulePrefs title="Descontos no Orkut">
4     <Require feature="opensocial-0.7"></Require>
5     <Require feature="dynamic-height"></Require>
6   </ModulePrefs>
7   <Content type="html">
8    <![CDATA[
```

O elemento central de uma aplicação OpenSocial é um arquivo XML (Código 7.1), onde é definida a versão do OpenSocial e os recursos necessários para a aplicação. Depois deste header, é inserido um elemento ≪![CDATA[”onde se pode inserir código HTML para ser exibido como a tela inicial da aplicação. Os códigos-fonte completos encontram-se nos apêndices deste trabalho, no texto foram destacados apenas os trechos que são explanados.

Code 7.2 – Código HTML inserido no elemento CDATA

```
1     <div id="dom_handle">
2
```

```
3           <a href="http://www.tagsmaker.com/"><img alt="Tags
                Maker is a Text Image Generator to write
                Messages, Comments or Tags on Pictures" src="
                http://img101.imageshack.us/img101/840/335
                e116f71f93bb07d29b58.jpg" border="0"></a>
4
5
6                   <h2> Clique Abaixo para se Autenticar</h2>
7
8                   <input type="submit" value="Autenticar-se"
                        onclick="request()">
9
10
11      </div>
```

Este HTML (Código A.1 definiu o visual da página inicial da aplicação "Descontos no Orkut", esta página requisita a autenticação do usuário do Orkut no servidor externo da "Descontos". Percebe-se que, pela utilização da autenticação via OAuth, não é necessária a digitação de nenhuma credencial, o próprio Login no Orkut traz as informações necessárias para garantir a identidade do usuário também no servidor externo.

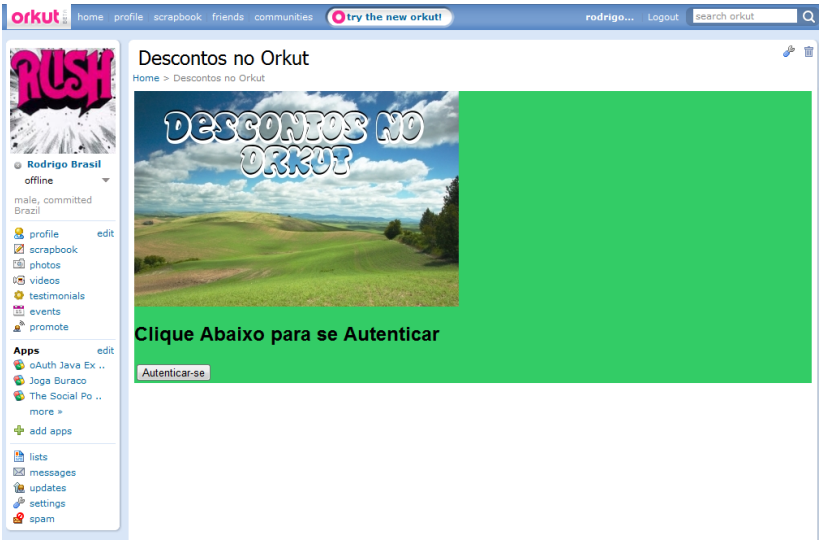

Figura 13 – Tela inicial do App "Descontos no Orkut"

O clique no botão "Autenticar-se"(Figura reffig:descontosinicial chama uma função Javascript que utiliza a API OpenSocial, cujo código

está descrito abaixo:

Code 7.3 – Função request(), utilizando a API OpenSocial

```
1        function request() {
2          var params={};
3
4
5                  params[gadgets.io.RequestParameters.
                       CONTENT_TYPE] = gadgets.io.ContentType.
                       TEXT;
6                  params[gadgets.io.RequestParameters.METHOD]
                       = gadgets.io.MethodType.POST;
7                  params[gadgets.io.RequestParameters.
                       AUTHORIZATION] = gadgets.io.
                       AuthorizationType.SIGNED;
8          gadgets.io.makeRequest(servletUrl,response,params);
9        };
```

Esta função (Código A.1) define uma série de parâmetros de configuração no array "params", estes estão previstos pela API do OpenSocial e irão caracterizar o tipo de requisição que o servidor do Orkut fará. "CONTENTTYPE"define o tipo de resposta que será esperado, no caso texto (HTML); "METHOD"define o método da requisição, no caso POST; "AUTHORIZATION", no contexto desse trabalho, é o mais importante, pois o valor "SIGNED"indica que o Orkut deve enviar juntamente com a requisição uma assinatura digital baseada em OAuth, para que o servidor de destino possa confiar na informação transmitida, garantir realmente a identidade do usuário, e por fim recuperar os seus dados.

Por fim chama-se o método "makeRequest()", também da API OpenSocial, este método, além dos parâmetros citados acima, recebe como parâmetro a URL destino para a qual a requisição deve ser feita, neste caso, um Servlet Java que executa no servidor principal da "Descontos". Sendo chamado, ele fará uma requisição primeiramente ao servidor do Orkut (para respeitar a política de Mesmo Domínio), que fará uma verificação dos parâmetros e assinará a requisição, para então repassá-la à URL destino. Mesma coisa acontece com a resposta, o servidor do Orkut a recebe, e a API OpenSocial chamará a função do aplicativo que havia sido designada (através do parâmetro "response"mostrado no código A.1) para recebê-la.

Feita a autenticação do lado do servidor da "Descontos", a função response (Código 7.4 receberá o HTML enviado como resposta, e o exibirá num elemento div (Código A.1 que já se encontrava na página.

Code 7.4 – Função que recebe a resposta da requisição OpenSocial

```
1        function response(data) {
2          document.getElementById('dom_handle').innerHTML=data
               .text;
3        };
```

Sendo assim temos os dados externos de "Descontos"do usuário logado no Orkut (Rodrigo Brasil) exibidos dentro do App "Descontos no Orkut"(Figura 14).

Figura 14 – Exibição das compras pendentes do usuário já autenticado

A requisição anterior chegou ao servidor da "Descontos"com três parâmetros importantes, que foram inseridos pelo servidor do Orkut:

- **opensocial viewer id**: O ID do usuário que está logado no Orkut no momento em que a requisição foi gerada.
- **xoauth signature publickey**: O nome do arquivo onde se encontra a última versão do certificado (chave pública) para decodificação da assinatura.
- **oauth signature**: Uma assinatura digital, gerada a partir de uma chave privada, que será decodificada com a chave pública descrita pelo parâmetro anterior.

Para facilitar a verificação desses dados no servidor da "Descontos", foi utilizada uma API OAuth escrita em Java (OAUTH, 2011), no Servlet SignedFetchVerifyServlet.java (Código 7.5).

Code 7.5 – Início do código do Servlet SignedFetchVerifyServlet.java

```
1  public class SignedFetchVerifyServlet extends HttpServlet {
2
3    private final static String CERTIFICATE =
4
5    @Override
6    protected void doPost(HttpServletRequest req,
         HttpServletResponse resp)
7        throws ServletException, IOException {
8      verifyFetch(req, resp);
9    }
10
11   private void verifyFetch(HttpServletRequest request,
         HttpServletResponse resp)
12       throws IOException, ServletException {
13
14
15     try {
16
17
18       //Define um objeto representando o provedor do Serviço
19       OAuthServiceProvider provider =
20           new OAuthServiceProvider(null, null, null);
21
22       //Define um objeto representando o consumidor do
             Serviço, no caso o Orkut.
23       OAuthConsumer consumer =
24           new OAuthConsumer(null, "orkut.com", null,
                 provider);
25
26       //Define o algoritmo de criptografia e o Certificado
             público para decriptar.
27       consumer.setProperty(RSA_SHA1.X509_CERTIFICATE,
             CERTIFICATE);
```

Para fins didáticos, o valor do certificado publickey foi inserido diretamente no código como atributo estático (CERTIFICATE). O método doPost() encaminha os parâmetros da requisição para o método verifyFetch(), onde são instanciados objetos da API OAuth que representam os elementos de autenticação OAuth. Como neste exemplo do Orkut, a única necessidade é a decriptação da assinatura, basta definir o algoritmo para decriptação e o valor da chave pública ao objeto do Consumidor OAuth.

Code 7.6 – Definição e decriptação da mensagem

```
1         OAuthMessage message =
2             new OAuthMessage(method, requestUrl,
                  requestParameters);
3
4         OAuthAccessor accessor = new OAuthAccessor(consumer);
5
6         //Aqui ocorre a validação da Assinatura
7         message.validateMessage( accessor, new
              SimpleOAuthValidator());
8
9         //Se a assinatura for validada, será mostrada a página
              personalizada para o usuário autenticado.
10        String page = "userMain.jsp";
11        RequestDispatcher rd = request.getRequestDispatcher(
              page);
12
13        // Encaminha a resposta do serlet para o jsp.
14        rd.forward(request, resp);
15
16        //Se houver algum problema de validação, cairá no
              tratamento de exceção e não redirecionará.
17    } catch (OAuthProblemException ope) {
```

Posteriormente é definida a mensagem a ser decodificada, de acordo com os parâmetros da requisição POST, também é definido um objeto com os dados do consumidor (certificado público e algoritmo de criptografia), e por fim é chamado o método validateMessage() do objeto message, onde a assinatura é verificada de acordo com o algoritmo (Código A.2). Este método gera uma exceção caso a assinatura não seja autêntica, portanto o código após a sua chamada só será executado no caso de autenticação válida. Sendo assim, o fluxo de execução é redirecionado para a página userMain.jsp (Código A.3, que verificará o usuário logado no Orkut e enviará seus respectivos dados do servidor da "Descontos".

Code 7.7 – Exibição dos Dados do Usuário

```
1  <%
2
3  //verifica o usuário autenticado.
4  String userId = request.getParameter("opensocial_viewer_id")
       ;
5
6
7
8  %>
9
10 <img src="http://img101.imageshack.us/img101/840/335
       e116f71f93bb07d29b58.jpg" border="2">
```

```
11
12 <% if(userId.trim().equals("01553276255527420356") )  {%>
13
14 <h2>Bem vindo, Rodrigo Brasil!</h2>
15
16 <h3>Suas compras pendentes são: </h3>
```

Como a aplicação é fictícia, não foi implementada uma real base de dados de usuários, portanto é feita para fins didáticos apenas uma verificação do ID do usuário (que já era previsto), e são exibidos os dados deste usuário caso a identificação seja positiva.

8 CONCLUSÃO

8.1 SOBRE AS FERRAMENTAS ESTUDADAS

8.1.1 InfoCards e Windows CardSpace

Os problemas abordados pelos meta-sistema de identidade são inegavelmente importantes. Proporcionar uma forma padrão de trabalho com diversas identidades digitais pode fazer o mundo em rede igualmente conveniente e seguro como o mundo físico. Dar aos usuários o controle sobre o como a identidade é usada em cada situação coloca as pessoas diretamente responsáveis pela forma como as suas identidades digitais são usadas.

O uso de identidades digitais, incluindo a identidade auto-emitida, pode reduzir a necessidade de senha baseada em logins da Web e juntamente melhorar a confiança do usuário em um site, ele também pode reduzir os ataques de phishing que atormentam muitos usuários.

O Windows CardSpace é uma parte fundamental dessas soluções. No entanto, até que as pessoas que criam software para outros sistemas operacionais, provedores de identidade, implementem o padrão de protocolos de serviços Web que permite que esse meta-sistema exista e interagir, o esforço ainda não terá atingido os seus objectivos. A Microsoft está fazendo a sua parte fornecendo o software para Windows, mas não pode, unilateralmente, tornar a visão meta-sistema de identidade um sucesso. Outros também devem compreender os benefícios que derivam da utilização mais eficaz das identidades digitais, e eles devem escolher para participar.

Ainda assim, o CardSpace é suscetível a ser amplamente disponível. Porque o meta-sistema de identidade subjacente CardSpace é baseado em protocolos abertos, software compatível com o CardSpace para provedores de identidade, confiando as partes, e outros seletores de identidade,podem ser construídos em qualquer plataforma ou dispositivo.

8.1.2 Shibolleth e SAML

O Sistema Shibolleth, baseado no padrão SAML, provê uma excelente solução na implementação de um sistema de Single Sign-On

baseado em Identidades Federadas. Seu design permite que a autenticações de usuários sejam realizadas de forma segura ao mesmo tempo que mantém totalmente a privacidade dos mesmos.

Muitos gerentes de TI imaginam que o Shibolleth seja complicado demais, no entanto a realidade é que as facilidades que ele provê superam os obstáculos enfrentados para sua implantação. Sua natureza open-source garante também um grande número de opções de suporte, além de permitir a flexibilidade e customização, de modo que o mesmo possa atender todas as necessidades de organizações em diferentes contextos.

8.1.3 OpenID

OpenID é uma tecnologia em torno do qual há um interesse crescente. Investimentos recentes para OpenID pela AOL, Microsoft e muitos outros serviços Web 2.0 colocaram ele firmemente no centro da identidade online e discussões sobre o gerenciamento de acesso. Parece altamente provável que tenha um impacto significativo sobre os serviços que os alunos e pesquisadores vão utilizar no futuro.

No momento da escrita, não há maior adoção do OpenID por alguns dos mais conhecidos serviços da Web 2.0. No entanto, com o passar do tempo , parece claro que estamos propensos a ver a sua adoção por mais e mais destes serviçoes da Web 2.0.

Para as instituições educacionais, OpenID está suscetível a se tornar um dos vários acesso e padrões de gerenciamento de identidade . Portanto, o planejamento estratégico na área de acesso e gerenciamento de identidade deve levar em conta na exigência de apoio a múltiplos padrões.

8.1.4 OAuth

OAuth pode ser resumido basicamente em um protocolo aberto, o qual descreve os processos que permitirão na prática às aplicações terem acesso a informações em domínios externos, passando-se pelos seus usuários. Nesse contexto abre-se então um leque de possibilidades, onde as implementações deste protocolo levam à grandes facilidades para os desenvolvedores destas aplicações e seus usuários.

Sua descrição é teórica, mas já existem várias implementações em diferentes linguagens de programação, na forma de bibliotecas. Há

também uma ampla adoção desta tecnologia por websites famosos, como o Orkut, que foi estudado neste trabalho, e também seu principal concorrente, o Facebook, que também utiliza o protocolo OAuth para dialogar questões de autorização com aplicações externas.

Em virtude dos fatos mencionados, o OAuth mostrou-se em teoria e prática ser uma ferramenta de grande eficiência e utilidade para o problema que se propõe a resolver.

8.2 QUADRO RESUMO

O quadro na figura 15 resume as principais características das ferramentas estudadas.

8.3 CONCLUSÃO GERAL DO TRABALHO

Foram levantados e estudados neste projeto vários problemas sobre a área de Gerenciamento de Identidades. Observou-se que a enorme variedade de situações e casos específicos levam ao desenvolvimento de iguamente variadas, amplas e flexíveis soluções. No meio digital, por exemplo, a quantidade de aplicações web levam os usuários à posse de muitas credenciais e senhas para sua identificação. No mundo físico, percebemos também a ocorrência deste problema, como a grande quantidade de documentos e seus contextos dentro de uma organização.

Devido a estes fatos, dentro da gama de ferramentas estudadas, não cabe afirmar simplesmente que uma é melhor do que a outra. A comparação seria injusta, pois ela não levaria em consideração a questão mais importante: "É melhor em que situação, para quais funções?". Além disso, as mesmas se encontram em domínios distintos de aplicação, onde cada uma se destaca, sendo feita exatamente para suprir as necessidades para as quais foi proposta.

Por este motivo pode-se afirmar que o estudo obteve sucesso, pois as ferramentas foram revisadas, suas potencialidades identificadas, e inclusive mostraram-se capazes de resolver os problemas que motivaram a realização deste trabalho.

8.4 TRABALHOS FUTUROS

O trabalho teve como objetivo trazer uma visão ampla da área de Gerenciamento de Identidades e suas ferramentas. Este tipo de visão

exige uma certa distância, onde detalhes mais específicos de cada uma das ferramentas precisam ser resumidos de forma que o trabalho mantenha sua uniformidade e não se estenda além do seu escopo. Portanto o estudo dessas ferramentas pode ainda ser mais aprofundado, a visão detalhada de apenas uma dessas ferramentas seria conteúdo suficiente para um Trabalho de Conclusão de Curso ou mesmo uma Dissertação.

Além disso o trabalho teve foco nas ferramentas que seus autores julgaram ser as mais importantes, no entanto a área ainda possui muitas outras ferramentas, cujo estudo também é importante e deve ser realizado para que sejam cobertas então todas as possibilidades de implementações e soluções na área de pesquisa proposta.

Ferramenta	O que é	Indicação de Uso	Plataforma	Quem Utiliza	Software Livre
OpenID	OpenID é um protocolo de autenticação descentralizado, gratuito.	Indicado para autenticações descentralizadas	Bibliotecas disponíveis para Java, Ruby and Rails, PHP, entre outras.	Google, Yahoo, WordPress, VeriSign entre outras organizações	Sim
Microsoft InfoCards e Microsoft CardSpace	Microsoft InfoCards e Microsoft CardSpace é um software cliente que faz a autenticação de uma identidade através de um cartão digital.	Indicado em casos onde uma autenticação física é necessária.	.NET	Instalado por padrão nos sistemas operacionais Windows Vista e Windows 7	Não
Shibboleth	O Shibboleth fornece os subsídios para a implantação de um sistema de identidades federadas a partir da implementação de uma base modificável de um provedor de serviço e um provedor de identidades.	Indicado para implementação de sistemas de identidades federadas.	Perl	Federação CAFe, Pennsylvania State University, JSTOR (http://www.jstor.org/)	Sim
OAuth	O OAuth é um protocolo que permite a comunicação entre aplicativos de modo que os mesmos possam trocar informações de identidade de seus usuários e compartilhar sessões autenticadas.	Indicado para integração de aplicativos web consagrados.	Bibliotecas disponíveis em PHP, Java, Python, entre outras.	Google, Facebook, Twitter	Sim

Figura 15 – Resumo das ferramentas estudadas

REFERÊNCIAS

AITORO, J. R. Identitiy management. *Next Gov*, p. 1–2, 2008.

CAJU, J. D. *Authentication and Authorization Infrastructure*. 2005. <http://shib.kuleuven.be/docs/AAI-Shibboleth-BelnetUserDay-27oct2005-v.1.1.pdf>. Acessado em 19/05/2011.

CARROLL, J. M. *Computer Security*. Boston, Estados Unidos: Butterworth-Heinemann, 1996. 3-22 p.

COMMUNITY, S. O. *SAML Wiki Knowledge Base*. 2011. <http://saml.xml.org/wiki/saml-wiki-knowledgebase>. Acessado em 19/05/2011.

COOK, B. *Liminal Existence and Projects*. 2010. <http://romeda.org/projects/>. Acessado em 19/05/2011.

DAILY, S. *Definitive Guide to Identity Management*. San Francisco, CA, USAl: RealTime Publishers, 2004. 3-8 p.

DAVID, W. C. Federated identity management. p. 20–21.

FOUNDATION, E. *Explore the Eclipse Universe*. 2011. <http://www.eclipse.org/>. Acessado em 19/05/2011.

FOUNDATION, O. *OpenID allows you to use an existing account*. 2004. <http://openid.net/get-an-openid/what-is-openid>. Acessado em 21/05/2011.

GOOGLE. *PÃ¡gina inicial de desenvolvedores para Orkut*. 2011. <http://code.google.com/intl/pt-BR/apis/orkut/>. Acessado em 19/05/2011.

GROUP, B. 2002. <http://www.burtongroup.com>. Acessado em 21/05/2011.

HEALTH, U. D. of; SERVICES, H. *Health insurance reform: standards for electronic transactions*. 2000. <http://aspe.hhs.gov/admnsimp/final/txfinal.pdf>. Acessado em 19/05/2011.

INFORMATION, G. 2002. <http://www.gigagroup.com/home.php>. Acessado em 21/05/2011.

INICIATIVE, I. M. *Shibboleth Project Homepage*. 2011. <http://http://shibboleth.internet2.edu/>. Acessado em 19/05/2011.

INITIATIVE, I. M. *FAQ on SAML and Shibboleth*. 2011. <http://shibboleth.internet2.edu/Shibboleth-SAML-FAQ.html>. Acessado em 19/05/2011.

LANDWEHR, C. E. Computer security. *IJIS*, p. 1–11, 2001.

LAWRENCE, G. *Shibboleth as Attribute Delivery for Authorization*. 2009. <http://net.educause.edu/ir/library/pdf/EAF652B.pdf>. Acessado em 19/05/2011.

LIST, L. W. *Latin Word for Unworried, Securus*. 2011. <http://www.latinwordlist.com/latin-word-for/latin-word-for-unworried-76152843.htm>. Acessado em 19/05/2011.

MALCHO, J. The weakest link of computer security. *Infosecurity Europe*, p. 1–3, 2010.

OAUTH. *OAuth Java API*. 2011. <http://oauth.googlecode.com/svn/code/java/core/>. Acessado em 19/05/2011.

OLLMAN, G. *The Phishing Guide: Understanding and Preventing Phishing Attacks*. 2006. <http://www.technicalinfo.net/papers/Phishing.html>. Acessado em 19/05/2011.

POWELL, A.; RECORDON, D. *OpenID: Decentralised Single Sign-on for the Web*. 2011. <http://www.ariadne.ac.uk/issue51/powell-recordon/>. Acessado em 19/05/2011.

ROTHMAN, M. *Public-key encryption*. 1999. <http://www.networkworld.com/news/$64452_0$5 − 17 − 1999.*html¿.Acessadoem*19/05/2011.

STREPPONE, C. *Gentle Introduction to OAuth*. 2010. <http://dev.opera.com/articles/view/gentle-introduction-to-oauth/>. Acessado em 19/05/2011.

VARIOUS. *OAuth Google Group*. 2011. <http://groups.google.com/group/oauth/?pli=1>. Acessado em 19/05/2011.

W3C. *Same Origin Policy*. 2011. <http://www.w3.org/Security/wiki/Same$_{O}rigin_{P}olicy$¿.$Acessadoem$19/05/201

ZERVAAS, Q. *How OAuth Works*. 2010. <http://www.phpriot.com/articles/twitter-authentication-zend-oauth/2>. Acessado em 19/05/2011.

APÊNDICE A – Códigos-Fontes do Capítulo sobre OAuth

A.1 OAUTHTEST.XML

Code A.1 – oauthtest.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2   <Module>
3    <ModulePrefs title="Descontos no Orkut">
4      <Require feature="opensocial-0.7"></Require>
5      <Require feature="dynamic-height"></Require>
6    </ModulePrefs>
7    <Content type="html">
8     <![CDATA[
9
10
11      <script type="text/javascript">
12        var servletUrl="http://187.65.212.119:8080/
              OAuthExampleOrkut/SignedFetchVerifyServlet";
13
14        function response(data) {
15          document.getElementById('dom_handle').innerHTML=data
                .text;
16        };
17
18        function request() {
19          var params={};
20
21
22                  params[gadgets.io.RequestParameters.
                        CONTENT_TYPE] = gadgets.io.ContentType.
                        TEXT;
23                  params[gadgets.io.RequestParameters.METHOD]
                        = gadgets.io.MethodType.POST;
24                  params[gadgets.io.RequestParameters.
                        AUTHORIZATION] = gadgets.io.
                        AuthorizationType.SIGNED;
25          gadgets.io.makeRequest(servletUrl,response,params);
26        };
27
28
29      </script>
30
31          <style type="text/css">
32
33          #dom_handle {
34
35                  background-color: #33CC66;
36                  font-family: sans-serif;
37                  font-weight: bold;
38
39
40
```

```
41
42          }
43
44
45
46          </style>
47
48
49      <div id="dom_handle">
50
51          <a href="http://www.tagsmaker.com/"><img alt="Tags
                Maker is a Text Image Generator to write
                Messages, Comments or Tags on Pictures" src="
                http://img101.imageshack.us/img101/840/335
                e116f71f93bb07d29b58.jpg" border="0"></a>
52
53
54                  <h2> Clique Abaixo para se Autenticar</h2>
55
56                  <input type="submit" value="Autenticar-se"
                        onclick="request()">
57
58
59      </div>
60     ]]>
61    </Content>
62  </Module>
```

A.2 SIGNEDFETCHVERIFYSERVLET.JAVA

Code A.2 – SignedFetchVerifyServlet.java

```
1  package rodrigojbc.inf.ufsc.oauthorkut;
2
3  import net.oauth.OAuth;
4  import net.oauth.OAuthAccessor;
5  import net.oauth.OAuthConsumer;
6  import net.oauth.OAuthMessage;
7  import net.oauth.OAuthProblemException;
8  import net.oauth.OAuthServiceProvider;
9  import net.oauth.SimpleOAuthValidator;
10
11 import net.oauth.signature.RSA_SHA1;
12
13 import java.util.ArrayList;
14 import java.io.IOException;
15 import java.util.Map;
16 import java.util.List;
17
```

```
18 import javax.servlet.RequestDispatcher;
19 import javax.servlet.ServletException;
20 import javax.servlet.http.HttpServlet;
21 import javax.servlet.http.HttpServletRequest;
22 import javax.servlet.http.HttpServletResponse;
23
24 public class SignedFetchVerifyServlet extends HttpServlet {
25
26   private final static String CERTIFICATE =
27       "-----BEGIN CERTIFICATE-----\n"
28           + "
                MIIDHDCCAoWgAwIBAgIJAMbTCksqLiWeMA0GCSqGSIb3DQEBBQUAMGg
                \n"
29           + "
                BAYTAlVTMQswCQYDVQQIEwJDQTEWMBQGA1UEBxMNTW91bnRhaW4gVml
                \n"
30           + "
                A1UEChMLR29vZ2xlIEluYy4xDjAMBgNVBAsTBU9ya3V0MQ4wDAYDVQQ
                \n"
31           + "
                bjAeFw0wODAxMDgxOTE1MjdaFw0wOTAxMDcxOTE1MjdaMGgxCzAJBgN
                \n"
32           + "
                MQswCQYDVQQIEwJDQTEWMBQGA1UEBxMNTW91bnRhaW4gVmlldzEUMBI
                \n"
33           + "
                R29vZ2xlIEluYy4xDjAMBgNVBAsTBU9ya3V0MQ4wDAYDVQQDEwVscnl
                \n"
34           + "
                BgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAseBXZ4NDhm24nX3sJRiZJhv
                \n"
35           + "j4HWAMmhAcnm2iBgYpAigwhVHtOs+
                ZIUIdzQHvHeNd0ydc1Jg8e+C+Mlzo38OvaG\n"
36           + "D3qwvzJ0LNn7L80c0XVrvEALdD9zrO+0
                XSZpTK9PJrl2W591Z1JFUk3pV+jFR8NY\n"
37           + "eB/
                fto7AVtECAwEAAaOBzTCByjAdBgNVHQ4EFgQUv7TZGZaI+
                FifzjpTVjtPHSvb\n"
38           + "XqUwgZoGA1UdIwSBkjCBj4AUv7TZGZaI+
                FifzjpTVjtPHSvbXqWhbKRqMGgxCzAJ\n"
39           + "
                BgNVBAYTAlVTMQswCQYDVQQIEwJDQTEWMBQGA1UEBxMNTW91bnRhaW4
                \n"
40           + "
                MBIGA1UEChMLR29vZ2xlIEluYy4xDjAMBgNVBAsTBU9ya3V0MQ4wDAY
                \n"
41           + "
                cnlhboIJAMbTCksqLiWeMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQE
                \n"
42           + "CETnhlEnCJVDXoEtSSwUBLP/147
                sqiu9a4TNqchTHJObwTwDPUMaU6XIs2OTMmFu\n"
```

```
43            + "
                  GeIYpkHXzTa9Q6IKlc7Bt2xkSeY3siRWCxvZekMxPvv7YTcnaVlZzHrVfAzql
                  \n"
44            + "P3J//C0j+8JWg6G+zuo5k7pNRKDY76GxxHPYamdLfwk=\n"
45            + "-----END CERTIFICATE-----";
46
47    @Override
48    protected void doGet(HttpServletRequest req,
          HttpServletResponse resp)
49        throws ServletException, IOException {
50
51      verifyFetch(req, resp);
52    }
53
54
55    @Override
56    protected void doPost(HttpServletRequest req,
          HttpServletResponse resp)
57        throws ServletException, IOException {
58      verifyFetch(req, resp);
59    }
60
61
62    private void verifyFetch(HttpServletRequest request,
          HttpServletResponse resp)
63        throws IOException, ServletException {
64
65
66      try {
67
68
69        //Define um objeto representando o provedor do Serviço
70        OAuthServiceProvider provider =
71            new OAuthServiceProvider(null, null, null);
72
73        //Define um objeto representando o consumidor do
              Serviço, no caso o Orkut.
74        OAuthConsumer consumer =
75            new OAuthConsumer(null, "orkut.com", null,
                  provider);
76
77        //Define o algoritmo de criptografia e o Certificado
              público para decriptar.
78        consumer.setProperty(RSA_SHA1.X509_CERTIFICATE,
              CERTIFICATE);
79
80        String method = request.getMethod();
81        String requestUrl = getRequestUrl(request);
82
83        List<OAuth.Parameter> requestParameters =
              getRequestParameters(request);
84
```

```
85          OAuthMessage message =
86              new OAuthMessage(method, requestUrl,
                    requestParameters);
87
88          OAuthAccessor accessor = new OAuthAccessor(consumer);
89
90          for (java.util.Map.Entry param : message.getParameters
                ()) {
91            String key = param.getKey().toString();
92            String value = param.getValue().toString();
93
94          }
95
96          //Aqui ocorre a validação da Assinatura
97          message.validateMessage( accessor, new
                SimpleOAuthValidator());
98
99          //Se a assinatura for validada, será mostrada a página
                personalizada para o usuário autenticado.
100         String page = "userMain.jsp";
101         RequestDispatcher rd = request.getRequestDispatcher(
                page);
102
103         // Encaminha a resposta do serlet para o jsp.
104         rd.forward(request, resp);
105
106         //Se houver algum problema de validação, cairá no
                tratamento de exceção e não redirecionará.
107       } catch (OAuthProblemException ope) {
108
109
110        } catch (Exception e) {
111
112            System.out.println(e);
113         throw new ServletException(e);
114       }
115     }
116
117
118     public static String getRequestUrl(HttpServletRequest
            request) {
119       StringBuilder requestUrl = new StringBuilder();
120       String scheme = request.getScheme();
121       int port = request.getLocalPort();
122
123       requestUrl.append(scheme);
124       requestUrl.append("://");
125       requestUrl.append(request.getServerName());
126
127       if ((scheme.equals("http") && port != 80)
128               || (scheme.equals("https") && port != 443)) {
129         requestUrl.append(":");
```

```
130         requestUrl.append(port);
131       }
132
133       requestUrl.append(request.getContextPath());
134       requestUrl.append(request.getServletPath());
135
136       return requestUrl.toString();
137     }
138
139
140     public static List<OAuth.Parameter> getRequestParameters(
141         HttpServletRequest request) {
142
143       List<OAuth.Parameter> parameters = new ArrayList<OAuth.
              Parameter>();
144
145       for (Object e : request.getParameterMap().entrySet()) {
146         Map.Entry<String, String[]> entry = (Map.Entry<String,
                String[]>) e;
147
148         for (String value : entry.getValue()) {
149           parameters.add(new OAuth.Parameter(entry.getKey(),
                  value));
150         }
151       }
152
153       return parameters;
154     }
155 }
```

A.3 USERMAIN.JSP

Code A.3 – Exibição dos Dados do Usuário

```
1  <%@ page language="java" contentType="text/html; charset=ISO
       -8859-1"
2       pageEncoding="ISO-8859-1"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
       EN" "http://www.w3.org/TR/html4/loose.dtd">
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=
       ISO-8859-1">
7  <title>Bem Vindo</title>
8  </head>
9  <body>
10
11 <%
12
```

```
13  //verifica o usuário autenticado.
14  String userId = request.getParameter("opensocial_viewer_id")
        ;
15
16
17
18  %>
19
20  <img src="http://img101.imageshack.us/img101/840/335
        e116f71f93bb07d29b58.jpg" border="2">
21
22  <% if(userId.trim().equals("01553276255527420356") )  {%>
23
24  <h2>Bem vindo, Rodrigo Brasil!</h2>
25
26  <h3>Suas compras pendentes são: </h3>
27
28
29  <p> 50% de Desconto no Sushi </p>
30
31  <p> <img src="http://boanoticia.com/wp-content/uploads
        /2010/04/sushi.jpg" />
32
33  <p> 62% de Desconto na Diária do Hotel City </p>
34
35  <p> <img src="http://3.bp.blogspot.com/_CcoVrXKWO3M/S4zqS-
        cNQ-I/AAAAAAAAGl0/5Ip7cfa2JcQ/s400/QUARTO+DE+HOTEL.jpg">
         </p>
36
37  <% }
38
39  else {
40
41          System.out.println("<h1> Usuario não registrado no
                Sistema </h2>");
42
43  }
44
45  %>
46
47
48
49  </body>
50  </html>
```