Um Serviço de Autorização Java EE Baseado em Certificados de Atributos X.509

Stefan Neusatz Guilhen[*], **Francisco Reverbel**

Departamento de Ciência da Computação
Instituto de Matemática e Estatística da Universidade de São Paulo

{sneusatz,reverbel}@ime.usp.br

Abstract. *This paper presents the rationale for using X.509 attribute certificates to store subject-role associations in an RBAC environment and describes the implementation of an authorization service that follows this approach. The authorization service supports both pull and push models for credential propagation and is integrated with a Java EE application server.*

Resumo. *Este artigo apresenta a motivação para o uso de certificados de atributos X.509 para armazenamento das associações entre sujeitos e papéis num ambiente RBAC e descreve a implementação de um serviço de autorização que usa essa abordagem. O serviço implementado dá suporte tanto ao modelo "pull" como ao modelo "push" de propagação de credenciais e é integrado a um servidor de aplicações Java EE.*

1. Introdução

A plataforma *Java Enterprise Edition* (Java EE) [Sun Microsystems, Inc. 2006] compreende uma série de tecnologias e APIs voltadas para o desenvolvimento de aplicações escaláveis de grande porte. As especificações dessas tecnologias estabelecem que o mecanismo de controle de acesso aos componentes de uma aplicação Java EE segue a abordagem *Role-Based Access Control* (RBAC) [Ferraiolo and Kuhn 1992]. Nessa abordagem, dentro de uma organização são definidos papéis que geralmente representam posições profissionais, tais como contador, gerente e diretor. Cada usuário de uma aplicação Java EE é então associado a um conjunto de papéis que define o escopo de seus privilégios. Além disso, a configuração de cada componente da aplicação Java EE define quais papéis têm acesso a quais serviços do componente.

Embora a padronização da plataforma estabeleça que o controle de acesso deve ser baseado nos papéis do usuário, ela não define como deve ser feito o mapeamento entre os usuários do ambiente operacional e os papéis definidos pela aplicação. Dessa forma, os servidores de aplicações Java EE (tais como o JBoss Application Server, o BEA Weblogic e o IBM Websphere) são livres para implementar este mapeamento da maneira julgada mais conveniente por seus projetistas. Esses servidores em geral oferecem diversas alternativas, que vão desde o uso de arquivos de propriedades até o uso de serviços de diretório ou bancos de dados para armazenar as identidades dos usuários e seus respectivos papéis.

Entretanto, todas as alternativas atualmente oferecidas pelos servidores Java EE baseiam-se no conceito de que cabe ao servidor a tarefa de localizar as credenciais dos

[*]Stefan Neusatz Guilhen desenvolveu este trabalho com o apoio financeiro da JBoss, Inc.

usuários. Esse modelo obriga as entidades responsáveis pelo gerenciamento das credenciais dos usuários a manterem um repositório que possa ser consultado pelo servidor. Tal exigência limita muito o processo de emissão de credenciais. Um serviço de autorização mais completo deveria também contemplar o caso em que o cliente é o portador de suas próprias credenciais e deve apresentá-las aos servidores Java EE que ele utilizar. Com tal serviço de autorização, o servidor de aplicações daria às entidades emissoras a liberdade de escolher a forma de emissão de credenciais mais adequada às necessidades de uma organização. Assim, a emissão poderia ser feita tanto através da publicação de credenciais em um repositório disponível para consulta, como através da entrega de credenciais diretamente aos usuários.

Para tanto, um mecanismo mais forte de autorização se faz necessário. É preciso definir claramente quem são as entidades responsáveis pela emissão de credenciais, a fim de estabelecer uma relação de confiança entre o serviço de autorização e as diferentes entidades com as quais este atua. Além disso, é preciso garantir que as credenciais portadas pelos clientes possuam garantias de integridade e origem.

O padrão X.509 [ITUT 2001] define a *infra-estrutura de chaves públicas*, ou *PKI* [Housley et al. 2002], que tem como objetivo possibilitar a autenticação dos sujeitos envolvidos em uma interação. O elemento central da PKI é o *certificado de chave pública*. A partir da sua quarta edição, o padrão passou a definir também um mecanismo forte de autorização, a *infra-estrutura de gerenciamento de privilégios*, ou *PMI* [Farrel and Housley 2002]. A PMI dá suporte à autorização depois que a autenticação foi realizada, isto é, provê meios de autorização que pressupõem a autenticação prévia dos sujeitos. O elemento central da PMI é o *certificado de atributos*, um documento criptograficamente seguro que associa um conjunto de atributos a um sujeito (tipicamente o portador do certificado) e usa esses atributos para descrever os privilégios do sujeito. A PMI também introduz o conceito de *autoridade de atributos*: essa é a entidade que tem a responsabilidade de gerar certificados de atributos e de assiná-los digitalmente.

Como os atributos presentes em um certificado de atributos X.509 podem ser usados para armazenar os papéis do seu portador, esse tipo de certificado é um meio especialmente apropriado para se realizar a associação de um usuário a seus papéis. Além disso, por serem digitalmente assinados por uma autoridade confiável, esses certificados garantem tanto a integridade do seu conteúdo como a sua origem. Tais fatos fazem dos certificados de atributos X.509 uma escolha ideal para a construção de ambientes RBAC. O presente trabalho descreve como aplicamos essa idéia ao projeto e à implementação de um serviço forte de autorização para um servidor de aplicações Java EE de código aberto. O serviço atualmente implementado é voltado para o controle de acesso a componentes Enterprise JavaBean (EJB), mas pode ser estendido para controlar também o acesso a componentes web, tais como servlets.

O restante deste artigo está organizado da seguinte forma: a seção 2 discute os modelos de propagação de credenciais. A seção 3 apresenta a infra-estrutura de segurança do servidor de aplicações sobre o qual desenvolvemos este trabalho. Já a seção 4 descreve em detalhe o serviço de autorização implementado. A seção 5 apresenta trabalhos relacionados, a seção 6 discute nossos planos para trabalhos futuros e, finalmente, a seção 7 contém nossas conclusões.

2. Modelos de propagação de credenciais

Dois modelos podem ser utilizados pelas autoridades emissoras de credenciais: o modelo "*pull*", no qual as credenciais são publicadas em repositórios acessíveis para consulta por parte de serviços de autorização, e o modelo "*push*", no qual as credenciais são distribuídas diretamente a seus portadores, que assumem a responsabilidade de apresentá-las no momento da utilização de recursos de acesso controlado.

Supondo que as credenciais estão contidas em certificados de atributos X.509, o primeiro modelo transfere para o serviço de autorização a responsabilidade pela busca desses certificados em repositórios, o que aumenta a complexidade do serviço e também implica que a autoridade emissora deve manter um repositório disponível para consulta. Essas características fazem com que esse modelo seja mais adequado para ambientes nos quais os privilégios dos clientes são definidos e atribuídos no próprio domínio do servidor, em geral sendo emitidos por autoridades localizadas também no mesmo domínio.

Já o segundo modelo transfere para o cliente a responsabilidade de fornecer os seus certificados de atributos X.509 para o servidor, reduzindo a complexidade do serviço de autorização. Esse modelo é mais adequado quando os privilégios dos clientes são definidos e atribuídos em um domínio diferente do servidor, sendo geralmente emitidos por autoridades totalmente independentes do ambiente servidor. Embora o modelo "push" remova do servidor a responsabilidade de encontrar os certificados dos clientes, tal modelo exige maiores cuidados com a validação dos certificados.

Vários fatores podem fazer com que um certificado seja considerado inválido, mas os dois principais são: o certificado expirou (ou seja, seu prazo de validade venceu), ou foi revogado, seja porque o portador não é mais considerado um usuário aprovado do sistema ou porque seu conjunto de privilégios foi alterado. O primeiro caso é fácil de se verificar, pois o certificado contém o prazo de validade como um de seus atributos. Já com respeito à revogação, a solução usualmente adotada é fazer com que a autoridade emissora do certificado publique listas de revogação de certificados (*Certificate Revocation Lists* ou *CRLs*) contendo os números de série dos certificados que não devem mais ser considerados válidos. Entretanto, existem problemas conhecidos relacionados às CRLs [Rivest 1998]. O principal deles é que as listas de revogação são publicadas periodicamente e, portanto, a revogação de privilégios de um certo usuário não é imediata. Por conta disto, outros mecanismos [Iliadis et al. 2003, Micali 2002] também foram propostos, como, por exemplo, o *Online Certificate Status Prococol — OCSP* [Myers et al. 1999], um protocolo que permite a verificação em tempo real da validade de um certificado junto à entidade certificadora.

O modelo "pull", por outro lado, pode lidar com a revogação de certificados de uma maneira muito mais simples. Como os certificados são mantidos em repositórios gerenciados pelas autoridades de atributos, a revogação de um certificado pode ser diretamente tratada através da simples remoção do certificado do repositório. Isso implica que somente certificados válidos ficam armazenados e, portanto, o serviço de autorização não é obrigado a verificar a validade dos certificados obtidos. Em [Crampton and Khambhammettu 2003] discute-se com mais detalhes os prós e contras desses dois modelos e argumenta-se que, embora o modelo "pull" precise fazer uma busca para obter as informações de autorização, tal modelo é tão eficiente quanto o modelo "push", pois o último também precisa fazer uma consulta para estabelecer a validade do certificado por conta da possibilidade de revogação. A necessidade de tal consulta está

também na base da argumentação de [van de Graaf and Carvalho 2004], que propõe um modelo "pull" baseado em certificados de chave pública e no serviço de diretórios LDAP.

3. Infra-estrutura de segurança do servidor JBoss

O servidor de aplicações escolhido para o desenvolvimento deste trabalho é o JBoss Application Server [Fleury and Reverbel 2003], versão 4.0.2, por ser um servidor de código aberto e por já possuirmos experiência com sua arquitetura, o que facilitou o processo de implementação e integração do mecanismo de autorização desenvolvido.

A *JBoss Security Extension*, conhecida também como JBossSX, é o conjunto de módulos responsável pela implementação da infra-estrutura de segurança do JBoss e por aplicar as regras de autorização exigidas pela especificação de segurança de EJB. Esse conjunto de módulos estabelece um arcabouço de segurança definido em termos de algumas interfaces. A implementação padrão dessas interfaces utiliza a API do *Java Authentication and Authorization Service* (JAAS) [Sun Microsystems, Inc. 2001] para fornecer as funcionalidades requeridas. A figura 1 exibe as principais classes e interfaces envolvidas nos processos de autenticação e autorização.

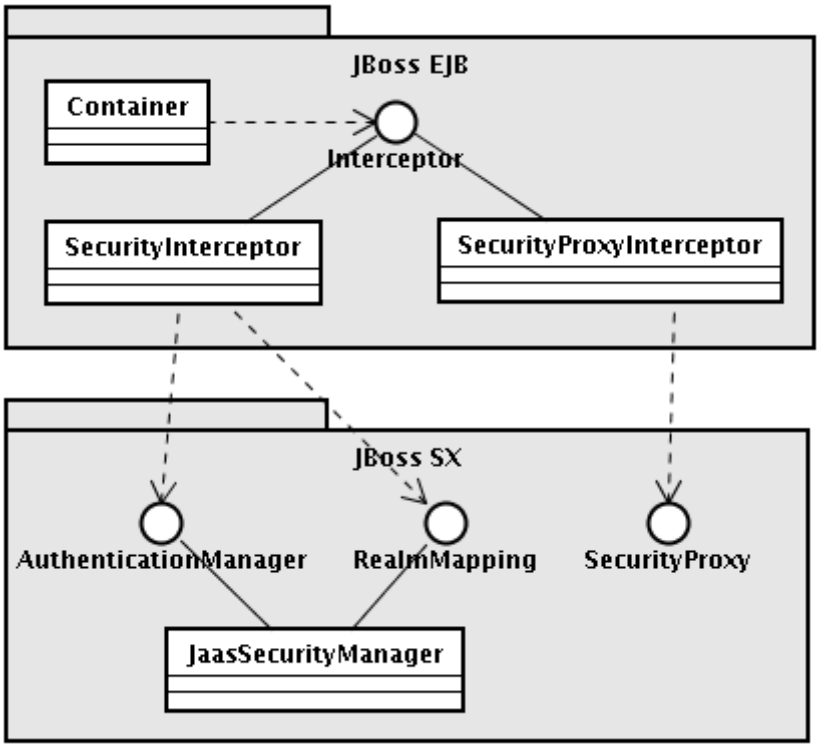

Figura 1. Arquitetura de segurança do JBoss

O `Container` é o componente do servidor responsável pelo gerenciamento do ciclo de vida dos EJBs implantados. No JBoss, existe um `Container` para cada EJB de uma aplicação. Quando um cliente invoca um método de um EJB, um objeto do tipo `Invocation` é criado, contendo o nome do EJB e o método a ser executado, bem como contextos adicionais de transação e segurança, esse último contendo a identidade do cliente e respectivas credenciais. Esse objeto é então enviado para o servidor. Antes de chegar ao `Container` responsável pelo EJB, o ob-

jeto passa por uma série de interceptadores que irão aplicar diversos serviços. Entre os interceptadores, encontramos o `SecurityInterceptor`, que é responsável pela autenticação e autorização do cliente. O outro interceptador envolvido com segurança é o `SecurityProxyInterceptor`, que é executado quando uma ou mais implementações da interface `SecurityProxy` foram configuradas.

As duas interfaces requeridas pelo JBossSX para implementação do modelo de segurança de EJB são a `AuthenticationManager` e a `RealmMapping`. A primeira é responsável pela validação das credenciais associadas com as identidades dos clientes. Ela define o método `isValid`, utilizado para verificar se as credenciais comprovam ou não a identidade do cliente. A segunda é responsável pelo mapeamento dos papéis da aplicação para os clientes do mundo real. Essa interface define o método `doesUserHaveRole`, utilizado para verificar se o cliente possui os papéis necessários para que o seu acesso seja autorizado.

Já a interface `SecurityProxy` permite que aplicações implementem verificações de segurança adicionais quando a segurança baseada nos papéis dos clientes não é suficiente. Por exemplo, pode ser que uma aplicação precise garantir que um argumento `String` fornecido a um certo método não ultrapasse um limite de caracteres. Esse tipo de verificação não é possível de ser realizada levando-se em conta apenas os papéis dos usuários.

É importante ressaltar que nenhuma dessas três interfaces tem relação alguma com a JAAS. Embora a implementação padrão do JBossSX seja fortemente dependente da JAAS, as interfaces que definem o modelo de segurança de EJB não são. Isso leva a uma arquitetura de segurança flexível, pois é possível substituir a implementação baseada na JAAS por uma outra implementação diferente se for necessário e apropriado.

A classe `JaasSecurityManager` fornece uma implementação baseada na JAAS para as interfaces `RealmMapping` e `AuthenticationManager`. Quando o `SecurityInterceptor` intercepta uma invocação, ele primeiro tenta autenticar o cliente usando as informações de segurança extraídas do objeto `Invocation`. A autenticação é delegada para o `JaasSecurityManager` através do método `isValid`, que é implementado de forma a utilizar a JAAS para executar o processo de autenticação. Para isso, o administrador do sistema deve ter configurado a aplicação implantada no servidor para utilizar um ou mais módulos de autenticação JAAS e atribuído a essa configuração um nome único. Feito isso, o processo de autenticação se desenrola da seguinte forma:

1. O método `isValid` cria uma instância de `LoginContext`, fornecendo o nome da configuração JAAS a ser utilizada e um objeto `CallbackHandler`. Esse último é o responsável pela interação com a aplicação (o servidor) para obter as informações requeridas pelos módulos para autenticar o cliente. No caso do JBoss, o `CallbackHandler` é instanciado e preenchido pelo método `isValid` com as informações de segurança fornecidas pelo `SecurityInterceptor`.
2. Ao ser criado, o `LoginContext` usa o nome da configuração para determinar todos os módulos a serem utilizados para autenticar o cliente. Os módulos, que são implementações da interface `LoginModule`, são então instanciados e inicializados pelo `LoginContext`. Durante a inicialização, cada módulo recebe, entre outras coisas, uma referência para o `CallbackHandler` e uma referência para um objeto `Subject` que representa o cliente sendo autenticado.

3. O método `isValid` executa o método `login` do objeto `LoginContext`, fazendo com que cada um dos módulos carregados execute o seu próprio método `login` para autenticar o cliente. Para obter as informações do cliente que foram extraídas da invocação, um módulo deve fornecer os objetos `Callback` apropriados para o método `handle` do `CallbackHandler`. Por exemplo, para obter a identidade do cliente, um `Callback` do tipo `NameCallback` deve ser passado para o `CallbackHandler`. Já para obter as credenciais do mesmo, um `ObjectCallback` deve ser utilizado. Cada `Callback` é preenchido pelo método `handle` de acordo com o seu tipo.
4. Uma vez que o módulo obtém os objetos `Callback` preenchidos do `CallbackHandler`, as informações extraídas são utilizadas para realizar a autenticação. Quando todos os módulos terminam de executar o método `login`, o `LoginContext` avalia o resultado do processo de autenticação. Em caso de sucesso, o método `commit` de cada `LoginModule` é invocado para que o módulo associe a identidade autenticada e outras informações importantes ao `Subject` recebido na inicialização. Em caso de fracasso, o método `abort` de cada módulo é executado, dando uma oportunidade para os módulos limparem qualquer estado salvo anteriormente.

O JBossSX oferece implementações de diversos `LoginModules`. A peculiaridade é que todos os módulos do JBossSX, além de realizar autenticação, também são responsáveis por encontrar os papéis do cliente. A classe `AbstractServerLoginModule`, superclasse de todos os módulos, fornece implementações para os métodos da interface `LoginModule` e define os métodos abstratos `getIdentity` e `getRoleSets`, que são invocados pelo método `commit` seguindo o padrão *Template* [Gamma et al. 1995]. Cada módulo deve implementar o método `getIdentity` de forma a devolver a identidade autenticada do cliente, e o método `getRoleSets` de modo a devolver o conjunto de papéis do cliente que foram obtidos pelo módulo. O método `commit` da classe `AbstractServerLoginModule` associa tanto a identidade autenticada quanto os papéis do cliente ao objeto `Subject` que o representa.

Ao final do processo de autenticação, o `JaasSecurityManager` armazena o `Subject` resultante em um cache em caso de sucesso e devolve o resultado para o `SecurityInterceptor`. Se a autenticação tiver falhado, o interceptador lançará uma exceção e impedirá que a invocação chegue ao `Container`. Caso contrário, o interceptador irá extrair do descritor de implantação o conjunto de papéis que têm permissão para realizar a invocação e invocará o método `doesUserHaveRole` do `JaasSecurityManager` para verificar se o cliente autenticado possui algum desses papéis. Esse último irá extrair o conjunto de papéis do `Subject` armazenado e responderá ao interceptador se uma intersecção não nula entre os dois conjuntos existe ou não. Em caso afirmativo, a invocação é liberada para prosseguir para o próximo interceptador ou para o `Container`.

4. Implementação do serviço de autorização

A arquitetura do JBossSX, baseada nos módulos de autenticação do JAAS, nos levou a dividir a implementação do serviço de autorização em dois módulos JAAS, um imple-

mentando o modelo “pull” e outro o modelo “push” para obter os certificados de atributos dos clientes. Dois motivos principais nos levaram a essa decisão:

Separação de responsabilidades. Cada modelo exige configurações diferentes para encontrar os certificados do cliente e misturar os dois modelos em um só módulo acabaria por criar um módulo inchado, com muitas responsabilidades.

Flexibilidade. Aplicações são livres para decidir como combinar os módulos existentes e não precisam necessariamente oferecer suporte ao dois modelos.

Todos os demais módulos implementados pelo JBossSX seguem o modelo “pull” para obter as credenciais e papéis dos clientes. A arquitetura suporta a adição de novos módulos que seguem esse modelo sem que alterações em outras classes sejam necessárias. A seção 4.1 aborda com mais detalhes a implementação desse modelo. A seção 4.2 apresenta o trabalho realizado para acomodar a implementação do modelo “push”. Já a seção 4.3 apresenta os mecanismos de verificação de revogação que podem ser utilizados por ambos os modelos para checar se um certificado foi ou não revogado após sua emissão.

4.1. Suporte ao modelo “pull”

O suporte ao modelo “pull” é oferecido pela classe `X509ACPullLoginModule`, uma implementação de `LoginModule` que se conecta a um repositório configurável para obter os certificados de atributos de cada cliente. De posse dos certificados, o módulo invoca um verificador para testar a validade dos mesmos (checando a assinatura digital, data de validade, entre outras coisas). Se uma cadeia de verificadores de revogação tiver sido configurada, os certificados obtidos são submetidos à verificação de revogação. Os certificados válidos são então utilizados para extração dos papéis que o cliente desempenha. A figura 2 ilustra o processo de obtenção dos papéis.

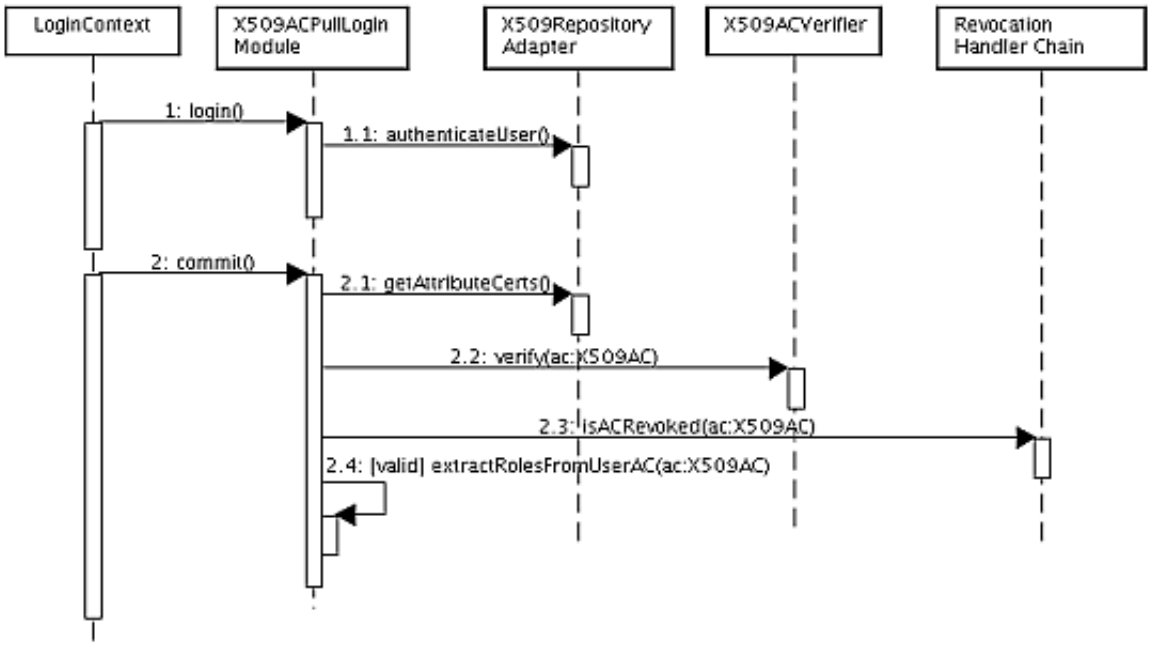

Figura 2. Diagrama de seqüência do modelo “pull”

A comunicação do módulo com o repositório contendo os certificados de atributos X.509 se dá através da interface `X509ACRepositoryAdapter`. Essa interface

permite que o módulo trabalhe com diversos tipos de repositório, bastando para isso que a implementação a ser utilizada seja configurada na opção apropriada do módulo. Uma implementação dessa interface, a `X509ACLdapAdapter`, que utiliza um serviço de diretórios LDAP como repositório de certificados, foi desenvolvida como parte do trabalho.

A escolha de um serviço de diretórios LDAP como repositório padrão foi motivada pelo fato de a especificação do padrão X.509 definir uma série de esquemas para habilitar o armazenamento desses certificados nos serviços de diretório de maneira padronizada. Isso simplifica a implementação do repositório porque o nome do atributo LDAP que deve ser utilizado para buscar o certificado de um cliente é padronizado e não precisa ser configurado externamente.

O `X509ACPullLoginModule` pode ser utilizado como um módulo autosuficiente, capaz de autenticar os usuários e buscar seus papéis no repositório, ou apenas como um módulo de extração dos papéis dos usuários. No primeiro caso, a autenticação é delegada para a implementação do repositório, que decide se as credenciais fornecidas por um cliente são válidas ou não. No segundo, o módulo deve ser configurado em conjunto com outros módulos que façam a autenticação. Isso é especialmente útil nos casos em que o repositório não é capaz de autenticar o cliente, ou está configurando para permitir acesso anônimo.

Além do repositório de certificados de atributos, o módulo permite a configuração do verificador a ser utilizado para determinar a validade dos certificados obtidos. O processo de validação é delegado para o verificador configurado, que deve implementar a interface `X509ACVerifier`. Isso permite que cada aplicação defina seu próprio verificador, tendo flexibilidade para decidir o grau de validação mais adequado para suas necessidades. Por exemplo, certas aplicações podem ser mais sensíveis aos conteúdos de algumas extensões do que outras, e podem exigir testes adicionais para validar essas extensões. A fim de facilitar a configuração do módulo, desenvolvemos o `X509ACDefaultVerifier`, um verificador padrão que é utilizado automaticamente quando nenhum outro verificador é fornecido.

O `X509ACDefaultVerifier` implementa o método `verify`, que recebe, entre outras coisas, o certificado de atributos a ser validado, uma `KeyStore` contendo os certificados de chave pública das autoridades de atributos de confiança e a identidade do cliente. Ele realiza um conjunto de validações comuns, como verificação da assinatura digital, do prazo de validade e da identidade do portador, que deve corresponder à identidade apresentada pelo cliente. Se todas as verificações forem bem sucedidas, o `DefaultACVerifier` considera o certificado de atributos válido. É preciso lembrar, porém, que este verificador apenas implementa um conjunto de testes simples e que aplicações podem exigir testes adicionais dependendo dos seus requisitos de segurança. Nesses casos, deve-se definir um verificador alternativo, que atenda às exigências impostas.

É possível aumentar o escopo de validação dos certificados através da configuração de um ou mais verificadores de revogação. A seção 4.3 descreve esses verificadores com mais detalhes.

Todas as características configuráveis do `X509ACPullLoginModule` são especificadas através de propriedades definidas no arquivo `login-config.xml`. Por

exemplo, a implementação de repositório a ser usada é especificada pela propriedade `repositoryAdapter`, a implementação de validador é especificada pela propriedade `verifier`, e assim por diante. Algumas propriedades não são diretamente usadas por esse `LoginModule`, mas são repassadas para a implementação do repositório (o endereço IP e a porta do servidor LDAP, o protocolo de comunicação a ser utilizado, etc).

4.2. Suporte ao modelo "push"

O desenvolvimento do módulo que permite ao cliente passar para o servidor os seus certificados de atributos exige, além da implementação de um `LoginModule`, alterações em outras classes para acomodar esses certificados. Em primeiro lugar, é preciso um mecanismo que permita ao cliente fornecer os seus certificados para que estes possam ser transferidos para o servidor nas invocações subseqüentes. Depois, é preciso fazer com que o servidor esteja atento à presença de certificados de atributos nas invocações dos clientes e os disponibilize para o módulo, ao qual atribuímos o nome de `X509ACPushLoginModule`. A figura 3 a seguir ilustra a propagação do contexto de segurança do cliente para o servidor.

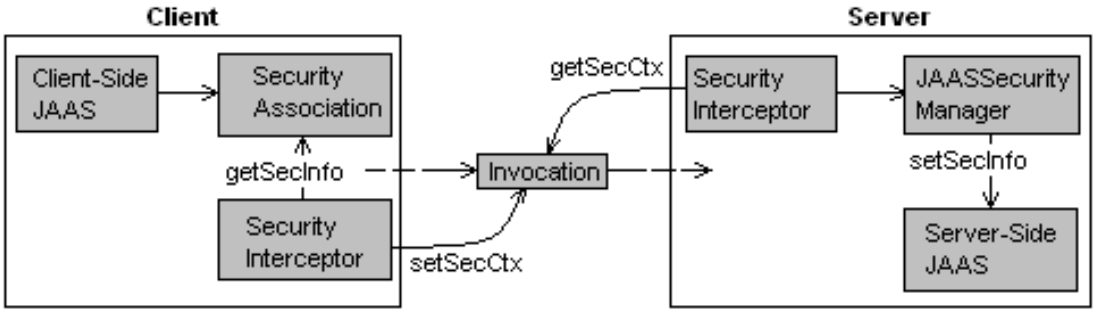

Figura 3. Propagação do contexto de segurança

No JBoss, o cliente também usa um `LoginModule` para fornecer as suas credenciais ao servidor. Porém, os módulos-cliente atualmente disponíveis não permitem que o cliente passe certificados de atributos como parte do contexto de segurança de suas invocações. Para contornar essa limitação, um novo módulo, o `X509ACClientLoginModule`, foi desenvolvido. Esse módulo obtém os certificados de atributos do cliente através do `X509ACCallback`. A aplicação cliente é responsável por fornecer um `CallbackHandler` que seja capaz de lidar com esse novo `Callback` e preenchê-lo com os certificados de atributos do cliente. A autenticação do cliente se dá então da seguinte forma (assumindo que o cliente configurou o `X509ACClientLoginModule` como módulo de autenticação JAAS):

1. O cliente cria uma instância de `LoginContext`, fornecendo o `CallbackHandler` definido pela aplicação. Em seguida, invoca o método `login` de `LoginContext`, fazendo com que o método de mesmo nome do `X509ACClientLoginModule` seja invocado.
2. O `X509ACClientLoginModule` cria instâncias de `NameCallback`, `ObjectCallback` e `X509ACCallback` e pede ao `CallbackHandler` fornecido para preencher esses objetos com a identidade do cliente, suas credenciais e seus certificados de atributos, respectivamente.

3. O módulo extrai as informações dos objetos `Callback` preenchidos e passa essas informações para a classe `SecurityAssociation`, que é responsável por armazenar esses dados localmente de forma que o cliente não precise realizar o processo de autenticação a cada invocação que fizer para o servidor. Se o módulo tiver a propriedade `multi-threaded` configurada com o valor `true`, a classe `SecurityAssociation` irá armazenar esses dados em variáveis `ThreadLocal`, implementado o padrão *Thread-Specific Storage* [Schmidt et al. 2000]. Com isso, cada *thread* da aplicação cliente terá seu contexto próprio de segurança. Se a mesma propriedade estiver ausente, ou configurada com valor `false`, os dados de segurança serão armazenados em variáveis comuns, compartilhadas por todas as *threads* da aplicação cliente.

Quando o cliente executa uma chamada a um método de um EJB, um objeto do tipo `Invocation` é criado. Esse objeto carrega informações a respeito da chamada, como o método a ser executado, seus parâmetros, contexto de transações e de segurança. Assim como ocorre no servidor, esse objeto passa por uma cadeia de interceptadores. A diferença é que no lado cliente os interceptadores são usados para adicionar informações ao objeto `Invocation` criado. Um desses interceptadores, o `SecurityInterceptor`, é responsável por colocar as informações de segurança do cliente na invocação. Para isso, ele consulta a classe `SecurityAssociation` e obtém identidade e as credenciais que o cliente forneceu no processo de autenticação.

Alterações foram necessárias nesses objetos para adicionar o suporte aos certificados de atributos X.509. Campos adicionais foram criados nas classes `SecurityAssociation` e `Invocation` para permitir que os certificados fornecidos pelo cliente sejam transferidos junto com o contexto de segurança da invocação para o servidor. Com isso, o `SecurityInterceptor` assume a responsabilidade adicional de preencher a invocação com os certificados de atributos que estiverem armazenados na classe `SecurityAssociation`.

No lado servidor, alterações nos objetos envolvidos com a segurança também foram efetuadas. Primeiro, o `SecurityInterceptor` do servidor deve extrair os certificados do objeto `Invocation` e transferí-los para o `JaasSecurityManager`. Este, ao criar o objeto `CallbackHandler`, que será utilizado pelos módulos para obter as informações de segurança, deve fornecer os certificados recebidos do interceptador juntamente com a identidade e as credenciais do cliente.

Para obter os certificados de atributos do cliente, o `X509ACPushLoginModule` deve criar uma instância de `X509ACCallback` e passá-la para o `CallbackHandler`, que irá preencher o conteúdo desse objeto com os certificados recebidos do `JassSecurityManager`. De posse do `X509ACCallback`, o módulo irá extrair os certificados do cliente para então validá-los e extrair os papéis encontrados.

O modelo "push" requer cuidados adicionais na validação dos certificados recebidos porque estes podem ter sido revogados. A forma mais comum usada pelas autoridades para divulgar as informações de revogação é através das CRLs. Normalmente essas listas são acessíveis publicamente, em um servidor web ou um serviço de diretório LDAP. O maior problema das CRLs é que elas são geradas periodicamente, o que impede que a revogação de um certificado seja imposta imediatamente. Por conta disso, soluções alternativas foram propostas e boa parte das autoridades também usa o OCSP, um protocolo

para consulta dinâmica da situação de um ou mais certificados. No OCSP, um verificador pode criar uma requisição contendo um ou mais números de série de certificados e enviar essa requisição para um serviço de consulta, conhecido por *OCSP Responder*, que criará uma resposta contendo o status de cada certificado identificado pelos números de série contidos na requisição.

4.3. Mecanismos de verificação de revogação

Em ambos os modelos apresentados, os módulos de autenticação desenvolvidos permitem a configuração de um ou mais verificadores de revogação. Os verificadores configurados são invocados em cadeia, respeitando a ordem em que aparecem na configuração de cada módulo. Cada verificador deve tentar determinar se o certificado fornecido foi ou não revogado após sua emissão. Caso consiga, o status de revogação é devolvido e a chamada termina. Do contrário, o certificado é repassado para o próximo membro da cadeia de verificadores.

Por padrão, três implementações distintas de verificadores foram produzidas. Em todos os casos, informações sobre o mecanismo de revogação são obtidas do conjunto de extensões presente no próprio certificado a ser verificado.

X509NoRevAvailHandler verifica se a extensão `noRevAvail` está presente no certificado. Essa extensão é utilizada por autoridades emissoras para indicar aos mecanismos de verificação de revogação que nenhuma informação de revogação será publicada e que, portanto, todo certificado contendo a extensão deve ser considerado válido.

X509CRLRevocationHandler tenta localizar uma CRL para obter o status de revogação do certificado. Para isso, ela consulta a extensão `CRLDistribution Points`, que é utilizada pelas autoridades para indicar os locais de onde é possível obter uma CRL. Quando nenhuma CRL é obtida por esse verificador, a chamada prossegue para o próximo membro da cadeia.

X509OCSPRevocationHandler utiliza o protocolo OCSP para enviar requisições a um serviço online de consulta de revogação. A extensão `AuthorityInformation Access` é utilizada pelas autoridades para informar a localização dos serviços de consulta OCSP.

Aplicações podem utilizar qualquer combinação dos verificadores para formar sua cadeia de verificação. Além disso, novos mecanismos de consulta podem ser implementados, bastando para isso criar uma nova subclasse de `X509RevocationHandler` e configurar os módulos para utilizar a implementação desenvolvida.

5. Trabalhos relacionados

O *Akenti* é uma infra-estrutura de autorização desenvolvida no Lawrence Berkeley National Laboratory [Johnston et al. 1998, Akenti 2004]. Ele foi desenvolvido com o objetivo de promover o gerenciamento distribuído da política de acesso aos recursos, possibilitando que diversas partes descrevam seus próprios requisitos quanto ao uso de um mesmo recurso independentemente umas das outras. Tanto as políticas de uso quanto os atributos dos usuários são armazenados em certificados proprietários, que por sua vez podem ser distribuídos em diversos repositórios em uma rede. O mecanismo de controle de acesso obtém esses certificados dos repositórios (modelo “pull”) e utiliza as informações neles

contidas para decidir se o acesso a um recurso deve ou não ser garantido ao usuário em questão. A autenticação dos usuários é feita somente através de certificados de chaves públicas. O fato de o Akenti usar certificados proprietários e não aderir a nenhum padrão aberto como o X.509 compromete a interoperabilidade com outras autoridades. Além disso, pouca flexibilidade é oferecida na escolha do mecanismo de autenticação, ficando esse restrito à PKI.

O *PERMIS* [Chadwick and Otenko 2002b, Permis 2007] (PrivilEge and Role Management Infraestructure Standards) é uma infra-estrutura de autorização baseada nos certificados de atributos X.509. Todos os dados necessários para decisões de autorização, como a especificação dos papéis e os privilégios alocados a cada papel, são descritos por uma política de autorização [Chadwick and Otenko 2002a], que é por sua vez armazenada em um certificado de atributos X.509 para garantir sua integridade. Tanto o certificado que contém a política de acesso quanto aqueles que contém os papéis associados a cada usuário são armazenados em serviços de diretório LDAP distribuídos. O PERMIS oferece total flexibilidade na escolha do mecanismo de autenticação a ser utilizado, além de aderir ao padrão X.509 para a geração dos certificados de atributos. Porém, assim como o Akenti, o PERMIS não oferece suporte ao modelo "push" de propagação de credenciais.

O Serviço de Segurança CORBA [OMG 2002, OMG 2004] provê um mecanismo de autorização que oferece suporte ao modelo "push" e pode usar certificados de atributos X.509 como credenciais. As credenciais tanto podem ser propagadas a cada requisição IIOP como podem ser associadas a uma sessão. No segundo caso, a propagação das credenciais ocorre apenas uma vez, no início da sessão. Interceptadores são usados tanto no lado cliente, para incluir as credenciais nas requisições enviadas, como no lado servidor, para extrair essas credenciais das requisições recebidas. Há uma grande similaridade entre essa arquitetura e a que implementamos. Em nosso trabalho, entretanto, a especificação dos papéis autorizados a efetuar determinada operação é feita de modo declarativo, num descritor de implantação XML. Isso decorre do fato de nosso serviço de autorização estar integrado a um servidor de aplicações Java EE.

6. Trabalhos futuros

A implementação do modelo "push" permite que clientes EJB forneçam seus certificados de atributos ao servidor. Seria interessante que clientes web pudessem fazer o mesmo. Como o `X509ACPushLoginModule` é independente do tipo do componente (EJB ou componente web), basta fazer com que a propagação dos certificados de atributos ocorra também no caso de acessos via HTTP. Pretendemos estudar a viabilidade de implementar, no browser cliente, algum mecanismo de apresentação dos certificados de atributos do usuário.

Estamos conduzindo experimentos para avaliação do desempenho dos módulos que desenvolvemos. Em nossos resultados preliminares, obtidos com o cache de segurança habilitado e na ausência de verificadores de revogação, a substituição dos módulos normalmente presentes no JBoss pelos que implementam nosso serviço de autorização não causou diferenças significativas nos tempos médios para autenticação e autorização de um usuário. Tencionamos ainda realizar experimentos com o cache de segurança desabilitado e na presença de mecanismos de revogação.

No momento o produto deste trabalho ainda é um protótipo. Estamos, entretanto,

interagindo com a comunidade de desenvolvedores do JBoss com vistas à inclusão de nosso serviço de autorização na distribuição oficial do servidor.

7. Considerações finais

A plataforma Java EE provê controle de acesso como um aspecto ortogonal à funcionalidade principal de cada aplicação. Como nosso serviço de autorização é integrado a um servidor de aplicações Java EE, o uso de certificados de atributos X.509 para armazenamento das associações entre sujeitos e papéis é totalmente transparente para os componentes EJB implantados no servidor. Isto ocorre tanto no caso em que as credenciais dos clientes são obtidas pelo servidor conforme o modelo "pull", como no caso em que são propagadas segundo o modelo "push". No lado da aplicação cliente, o uso de certificados de atributos não requer nenhuma alteração se o modelo "pull" for empregado e requer alterações mínimas no caso do modelo "push". Tais alterações são necessárias apenas para acomodar o fornecimento dos certificados de atributos do cliente.

Além de possibilitar suporte ao modelo "push", o uso de certificados de atributos traz também vantagens no caso do modelo "pull". O fato dos certificados serem digitalmente assinados oferece garantias de segurança fortes, que vão além do mero controle de acesso ao repositório onde ficam armazenadas as associações entre sujeitos e papéis. Em outras palavras, no caso do modelo "pull' a segurança das informações nos certificados não é limitada pela segurança do repositório no qual esses certificados são mantidos. Isso permite, inclusive, que o serviço de autorização consulte repositórios remotos, cujas políticas de segurança ele não necessariamente endossa.

Até onde pudemos averiguar, este projeto produziu o único servidor de aplicações Java EE que emprega certificados de atributos X.509 para implementar RBAC e que provê suporte aos dois modelos de propagação de credenciais. Sendo o único a implementar o modelo "push", nosso protótipo é também o único a contemplar cenários nos quais as credenciais dos clientes são definidas fora do domínio do servidor. Acreditamos serem essas as contribuições mais importantes deste trabalho. Outro aspecto que vale destacar é a implementação de um mecanismo extensível de verificação do status de revogação dos certificados. Tal mecanismo permite a utilização intercambiável de diversos métodos verificadores de revogação.

Referências

Akenti (2004). Akenti Project Home Page. `http://dsd.lbl.gov/Akenti/`. Acessado em 20 de Abril de 2007.

Chadwick, D. W. and Otenko, A. (2002a). RBAC Policies in XML for X.509 Based Privilege Management. In *Proceedings of 17th IFIP International Conference on Information Security - SEC2002*, pages 39–54.

Chadwick, D. W. and Otenko, A. (2002b). The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the 7th ACM Symposium on Access Control Models And Technologies - SACMAT*, pages 135–140.

Crampton, J. and Khambhammettu, H. (2003). Authorization and Certificates: Are We Pushing When We Should Be Pulling? In *Proceedings of IASTED International Conference on Communication, Network and Information Security*, pages 62–66.

Farrel, S. and Housley, R. (2002). An Internet Attribute Certificate Profile for Authorization, IETF Internet Draft RFC 3281.

Ferraiolo, D. and Kuhn, R. (1992). Role-Based Access Controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563.

Fleury, M. and Reverbel, F. (2003). The JBoss Extensible Server. In *Middleware 2003 — ACM/IFIP/USENIX International Middleware Conference*, volume 2672 of *LNCS*, pages 344–373. Springer-Verlag.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1st edition.

Housley, R., Polk, W., Ford, W., and Solo, D. (2002). An Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, IETF Internet Draft RFC 3280.

Iliadis, J., Gritzalis, S., Spinellis, D., Cock, D. D., Preneel, B., and Gritzalis, D. (2003). Towards a framework for evaluating certificate status information mechanisms. *Computer Communications*, 26(16):1839–1850.

ITUT (2001). ITU-T Recommendation X.509 ISO/IEC 9594-8. The Directory: Public Key and Attribute Certificate Frameworks.

Johnston, W., Mudumbai, S., and Thompson, M. (1998). Authorization and Attribute Certificates for Widely Distributed Access Control. In *Proceedings of 7th Workshop on Enabling Technologies, Infrastructure for Collaborative Enterprises - WETICE*, pages 340–345.

Micali, S. (2002). NOVOMODO: Scalable Certificate Validation And Simplified PKI Management. In *Proceedings of First Anual PKI Research Workshop*, pages 15–26.

Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. (1999). X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, IETF Internet Draft RFC 2560.

OMG (2002). *Security Service Specification, Version 1.8*. Object Management Group. OMG document formal/02-03-11.

OMG (2004). *Common Object Request Broker Architecture: Core Specification, Version 3.0.3*, chapter 24 (Secure Interoperability). Object Management Group. OMG document formal/04-03-01.

Permis (2007). Permis Project Home Page. `http://sec.cs.kent.ac.uk/permis/`. Acessado em 20 de Abril de 2007.

Rivest, R. L. (1998). Can We Eliminate Certificate Revocation Lists? In *Proceedings of Financial Cryptography*, pages 178–183.

Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture Volume 2 - Patterns for Concurrent and Networked Objects*. Wiley Series In Software Design Patterns. Wiley, 1st edition.

Sun Microsystems, Inc. (2001). Java Authentication and Authorization Service (JAAS) Reference Guide. `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html`. Acessado em 20 de Abril de 2007.

Sun Microsystems, Inc. (2006). Java Platform, Enterprise Edition (Java EE) Specification, version 5. Disponível em `http://jcp.org/en/jsr/detail?id=244`.

van de Graaf, J. and Carvalho, O. (2004). Reflecting on X.509 and LDAP, or How separating identity and attributes could simplify a PKI. In *Proceedings of IV Workshop em Segurança de Sistemas Computacionais - WSEC*, pages 37–48.