INTERFACE DE SEGURANÇA PARA SERVIDORES DE DADOS UNIVERSAIS

Flávia Linhalis
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - ICMC/USP
Centro Universitário de Araraquara – Uniara
flavialin@bol.com.br

Dilvan De Abreu Moreira
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - ICMC/USP
dilvan@icmc.sc.usp.br

RESUMO

Este artigo descreve a implementação de uma Interface de Segurança para Servidores de Dados Universais (SIUDS, em inglês). A SIUDS provê um ambiente para executar agentes de software e permitir uma interface segura entre estes e o banco de dados de um servidor de dados universal. Os agentes podem ser capazes de servir os mesmos dados nos mais diferentes formatos e protocolos, formando a base de um servidor de dados universal. As principais funções da SIUDS são receber os agentes, autenticá-los e prover acesso aos objetos do banco de dados e aos recursos do sistema de forma segura.

Os agentes são implementados em Java. Eles têm, nesse servidor de dados universal, o mesmo papel que as linguagens de consulta têm nos bancos de dados relacionais. Mas, como eles têm a vantagem de possuir todo o poder de computação do ambiente Java, eles podem realizar esse papel de forma mais eficiente.

ABSTRACT

This paper describes the implementation of a Secure Interface for Universal Data Servers (SIUDS). The SIUDS is to be a place to host software agents and securely interface them with the database of an universal data server. The agents can then serve the same data in all sorts of formats and protocols, this can be considered the basis of an universal data server. The SIUDS main functions are to receive the agents, authenticate them and provide access to database objects and systems resources securely.

The software agents are implemented in Java. They have, in the universal data server, the same role as query languages have in a relational database. However, as agents have the advantage of enjoying all the power provided by the Java environment, they can fulfill this role more efficiently.

1 INTRODUÇÃO

Este artigo tem como objetivo propor e implementar uma Interface de Segurança para Servidores de Dados Universais (SIUDS, em inglês).

O acesso ao banco de dados do servidor é feito por agentes de software móveis. Por esse motivo, há a necessidade de uma interface que garanta a proteção dos dados e do sistema hospedeiro, pois não é seguro que agentes desconhecidos entrem em execução e tenham liberdade irrestrita.

Este artigo é dividido em mais oito seções. A seção 2 apresenta o conceito e a importância dos Servidores de Dados Universais no cenário de banco de dados atual.

A seção 3 propõe a Interface de Segurança para Servidores de Dados Universais. São descritos os objetivos da interface e os módulos necessários para atingir os objetivos.

As seção 4 a 6 são dedicadas à revisão da literatura. A seção 4 apresenta conceitos relacionados aos agentes de software e as características que tornam Java a linguagem escolhida por vários sistemas que envolvem agentes móveis, inclusive a SIUDS, descrita neste artigo. A seção 5 descreve os mecanismos de criptografia, assinatura digital, certificação e autorização, que são necessários para garantir a segurança proposta pela SIUDS. A seção 6 apresenta os recursos de Java que são utilizados para implementar os mecanismos de segurança descritos na seção 5.

A seção 7 apresenta os detalhes de implementação da SIUDS, descrevendo os recursos de Java que foram utilizados para desenvolver cada módulo.

A seção 8 descreve o InfoAgent, um agente desenvolvido para ser executado pela SIUDS e testar sua funcionalidade. A seção 9 apresenta as conclusões deste trabalho.

2 CONCEITO DE SERVIDOR DE DADOS UNIVERSAL

Existem atualmente muitos tipos de programas servidores de informação disponíveis, tais como servidores de HTML, de SQL, etc. Todos eles possuem a mesma funcionalidade básica: fornecer informações a partir de requisições de clientes.

A existência desses diversos tipos de servidores e, conseqüentemente clientes, obriga os sites que fornecem informações a manterem vários tipos de servidores executando e, muitas vezes, armazenando a mesma informação em vários formatos diferentes. Seria útil ter toda a informação armazenada em apenas um formato em um banco de dados/servidor que pudesse fornecê-la em uma variedade de formatos e protocolos diferentes. Esse banco de dados/servidor poderia permitir que os usuários especificassem e recebessem a informação

procurada em uma variedade de formatos (HTML, SQL, etc.). Esta é a idéia de um servidor de dados universal.

Um servidor de dados universal (Krishnamurthy, 1999; Davis, 1997) deve ser capaz de responder a diferentes tipos de requisições utilizando o mesmo conjunto de dados. Para conseguir isso, todas as informações de interesse dos clientes devem ser mantidas em um (ou mais) banco de dados, que pode ser lido e atualizado por qualquer serviço.

O banco de dados de um servidor universal deve ser capaz de armazenar qualquer tipo de objeto. Para isso, a indústria de bancos de dados oferece os Sistemas Gerenciadores de Banco de Dados Orientados a Objeto (SGBDOO) e os Sistemas Gerenciadores de Banco de Dados Objeto-Relacionais (SGBDOR).

3 A INTERFACE

A Interface de Segurança para Servidores de Dados Universais, descrita neste artigo, enfoca a segurança para acesso aos objetos de um banco de dados que faz parte de um servidor de dados universal. O acesso a esses objetos e aos recursos do sistema será feita de forma autônoma por agentes de software (ver seção 4).

A figura 1 dá uma visão geral da SIUDS, que é composta de quatro componentes: a Interface com o Banco de Dados, o Gateway, o Pool de Agentes e o SecurityManager.

A função da SIUDS é, basicamente, receber os agentes, controlar o acesso dos agentes aos objetos do banco de dados e aos recursos do sistema.

A SIUDS fornece segurança em nível de autenticação e autorização. A autenticação diz respeito à validação da identidade do possuidor de um agente. A autorização define que privilégios um agente terá para realizar acesso aos objetos e aos recursos do sistema.

Para acessar os objetos, um agente deve utilizar-se da Interface com o Banco de Dados. Essa interface possui a definição de métodos que permitem ao agente criar grupos, criar objetos e associar permissões de acesso entre os objetos e os grupos. Assim, um agente só poderá acessar os objetos que lhe forem permitidos, o que dependerá das permissões de acesso concedidas ao grupo a que ele pertence.

Se os métodos da Interface com o Banco de Dados fossem acessados exclusivamente por usuários locais, bastaria o controle de acesso para garantir a segurança dos objetos. Mas, como serão aplicações desenvolvidas como agentes de software móveis a entrar em contato com os métodos, torna-se necessário autenticar os agentes, pois não é seguro para o sistema permitir que agentes desconhecidos entrem em execução. A autenticação é feita pelo agente Gateway. Ele recebe agentes vindos de hosts desconhecidos, verifica sua assinatura digital e os certificados de segurança relacionados a eles. Caso a verificação seja feita com sucesso, significa que o responsável por aquele agente é confiável e, por isso, o agente poderá entrar em execução no Pool de Agentes. A função do Pool é colocar os agentes em execução e controlar o tempo de vida deles. O tempo de execução de cada agente depende do grupo ao qual ele pertence.

A SIUDS e os agentes que ganharão acesso ao Pool foram desenvolvidos em Java (ver seção 4). Com isso, os agentes têm a vantagem de poder utilizar todos os recursos que o ambiente Java oferece. Eles poderiam, dependendo das permissões associadas ao seu grupo, ler e escrever no sistema de arquivos, estabelecer conexões com hosts remotos, modificar propriedades do sistema, etc. Mas, por questão de segurança, nem todos os grupos devem ter acesso a todos os recursos do sistema. Por isso, a classe SecurityManager da máquina virtual Java é instalada. Essa classe deve controlar o acesso dos agentes aos recursos do sistema de acordo com as permissões que estiverem associadas ao grupo do

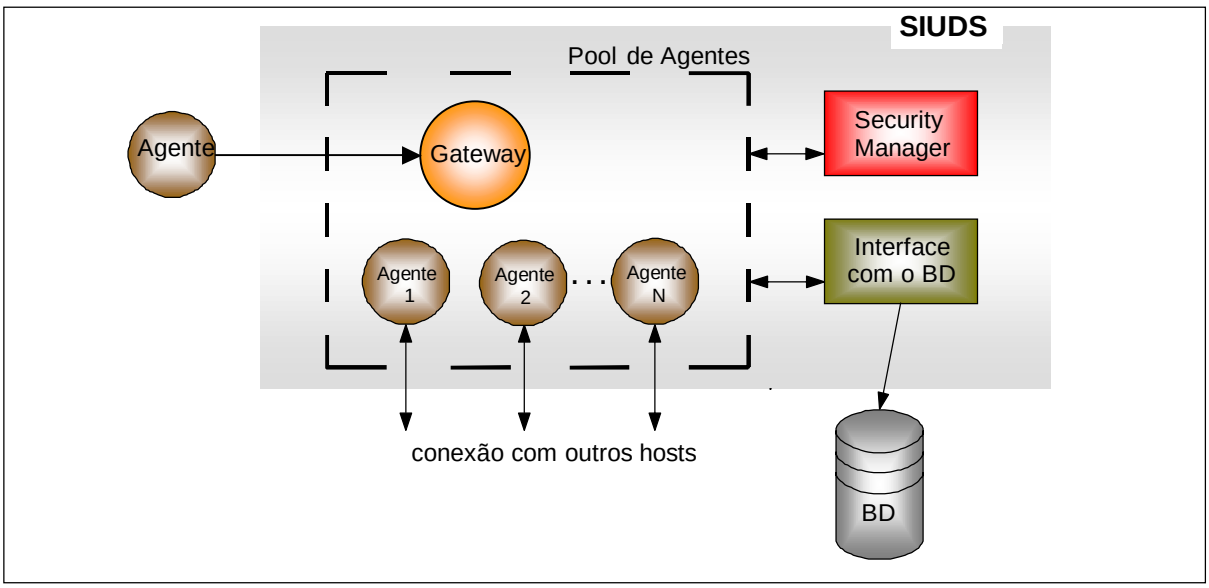

Figura 1 – Visão Geral da SIUDS

agente.

Como a SIUDS e os agentes foram desenvolvidos em Java, ela se torna, naturalmente, a linguagem de consulta ideal para o banco de dados. A substituição das linguagens de consulta por agentes de software pode tornar a aplicação mais flexível e poderosa. Uma aplicação pode criar sua própria linguagem para consulta e até mesmo mudar o comportamento do servidor (criar um novo servidor HTTP na porta 8080, por exemplo).

O conjunto da SIUDS e do banco de dados orientado a objetos pode ser considerado um servidor de dados universal porque os agentes que irão acessá-lo podem ser de qualquer tipo e, por isso, podem acessar os mais diversos tipos de objetos e servi-los a seus clientes das mais diferentes formas.

A autenticação realizada pelo Gateway, o controle de acesso aos objetos, proporcionado pela Interface com o Banco de Dados, e o controle de acesso aos recursos do sistema, feito pelo SecurityManager, possibilitam que o Pool seja um ambiente de execução de agentes aberto e ao mesmo tempo seguro. Os agentes poderão estabelecer conexões com hosts remotos, armazenar objetos no banco de dados e realizar uma série de outras operações de forma segura, ou seja, de acordo com os privilégios concedidos ao seu grupo.

4 A LINGUAGEM JAVA E OS AGENTES DE SOFTWARE

Os agentes de software (Maes, 1995; Franklin e Graesser, 1996) surgiram para facilitar a criação de softwares capazes de interoperar, ou seja, trocar informações e serviços com outros programas e, dessa forma, resolver problemas complexos. A metáfora utilizada pelos agentes de software é a de um assistente que colabora com o usuário e/ou outros agentes.

A portabilidade, combinada com a facilidade de programação, tornam Java a linguagem escolhida por vários sistemas de agentes, em especial os agentes móveis, os quais devem ser capazes de migrar de uma plataforma a outra. Além da portabilidade, a linguagem Java possui várias outras características adequadas para a implementação de agentes, entre elas (Wong et al, 1999):

- Java Beans: Um bean é um componente Java. Ele pode trabalhar em conjunto com outros beans para resolver problemas complexos.
- Serialização: É a capacidade de armazenar e recuperar objetos Java. Entre as vantagens na serialização pode-se citar a portabilidade necessária para objetos remotos e o suporte à persistência.
- Introspecção: É a capacidade de detectar os métodos e atributos disponíveis em uma classe e produzir seus nomes sem ter acesso ao seu código fonte. Com introspecção, um agente pode descobrir os métodos de outros agentes e estabelecer uma comunicação.
- Class Loading: Permite que as classes que compõem uma aplicação possam ser carregadas em tempo de execução.
- Segurança: O sistema de segurança da máquina virtual Java impõe restrições para o acesso aos recursos da plataforma Java. A segurança para a execução dos agentes é o ponto chave da SIUDS. Por esse motivo, as seções 4 e 5 se dedicam à questão da segurança.

5 MECANISMOS DE SEGURANÇA

Um sistema seguro deve levar em consideração as seguintes propriedades (Chin, 1999):

- Confidencialidade: apenas as entidades envolvidas podem ter acesso ao conteúdo dos dados que estão trafegando na rede.
- Integridade: a informação transmitida em um ponto é a mesma recebida em outro.
- Autenticação: as entidades envolvidas em uma comunicação devem ter meios de confirmarem mutuamente suas identidades.
- Controle de Acesso: serve para restringir o acesso aos recursos. Assim, apenas entidades autorizadas poderão acessá-los.
- Não Repudiação: previne que entidades neguem suas ações. Assim, se uma entidade enviou uma mensagem, ela não poderá negar que o fez.
- Disponibilidade de Serviços: garante que entidades autorizadas acessem determinados serviços.

Os mecanismos de segurança usados para implementar essas propriedades de segurança são: criptografia, assinatura digital, certificação e autorização.

5.1 Criptografia

A criptografia (Coulouris et. al., 1994) consiste em modificar a mensagem a ser transmitida, gerando uma mensagem criptografada na origem, através de um processo de codificação definido por um método de criptografia. A mensagem criptografada é então transmitida e, no destino, o método de criptografia é aplicado novamente para decodificar a mensagem.

Para tornar o mecanismo de criptografia mais seguro utiliza-se uma chave, onde a mensagem criptografada varia de acordo com a chave de codificação utilizada para o mesmo método de criptografia. Isto é, para a mesma mensagem e um mesmo método de criptografia, chaves diferentes produzem mensagens criptografadas diferentes. Assim, o fato de um intruso conhecer o método de criptografia não é suficiente para que ele possa recuperar a mensagem original, pois é necessário fornecer ao procedimento responsável pela decodificação tanto a mensagem criptografada quanto a chave de decodificação.

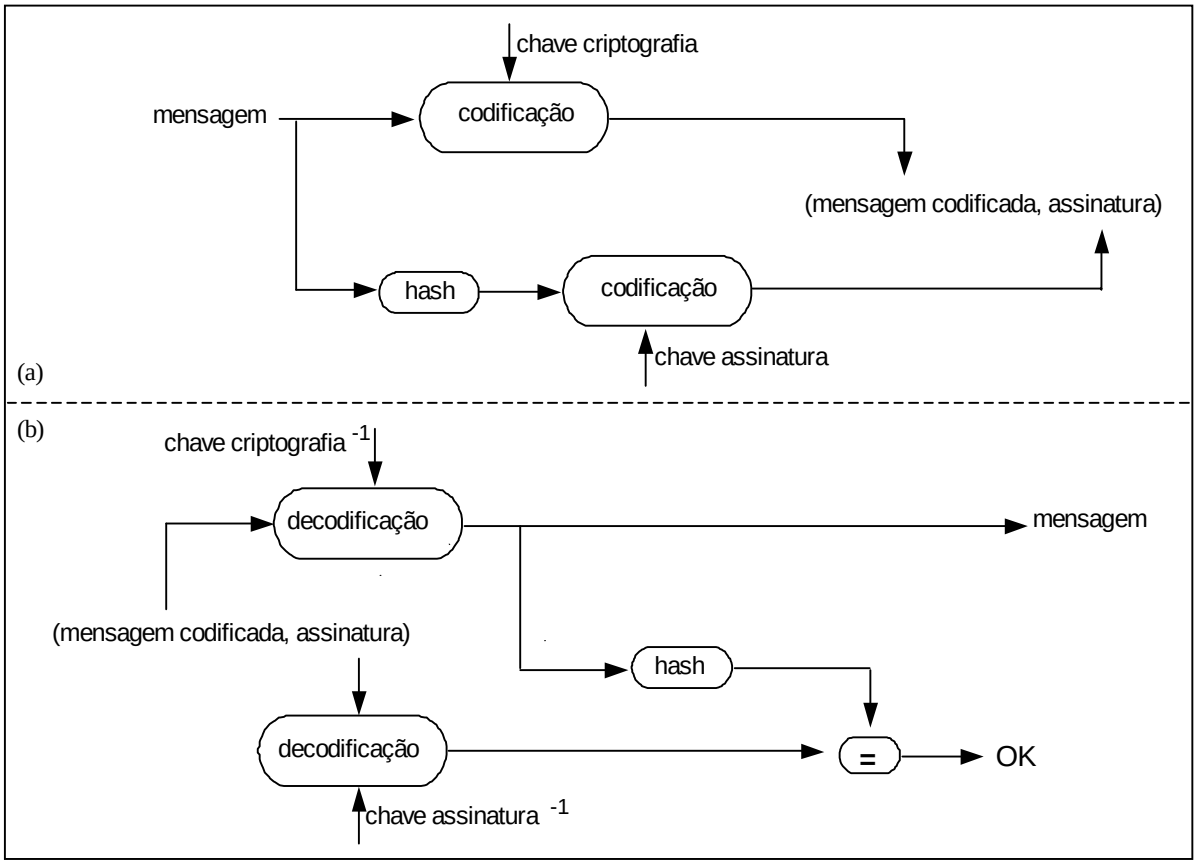

Figura 2 - Geração (a) e Verificação (b) de Assinatura Digital

A criptografia é utilizada para garantir a propriedade de confidencialidade, mas não garante a autenticidade do emissor da mensagem. Para isso, deve-se utilizar a criptografia juntamente com as assinaturas digitais.

5.2 Assinatura Digital

A função das assinaturas digitais (Chin, 1999; Coulouris et. al., 1994) no mundo eletrônico é o mesmo das assinaturas no papel do mundo real. Desde que uma chave privada é conhecida apenas pelo seu possuidor, a utilização dessa chave é vista como uma evidência de identidade.

O mecanismo de assinatura digital envolve dois procedimentos: assinatura de uma mensagem e verificação dessa assinatura. Uma mensagem M pode ser assinada por uma entidade P através da codificação de uma cópia de M com uma chave C_a (única e secreta) pertencente a P, acoplando-se isso ao texto original de M e ao identificador de P. Assim, um documento assinado consiste de <M, P, $\{M\}_{Ca}$>. O propósito de se acoplar uma assinatura a um documento é de permitir que qualquer um que receba o documento possa verificar que ele se originou de P e que o conteúdo de M não foi modificado.

A figura 2 ilustra o processo de geração e verificação de uma assinatura digital juntamente com criptografia.

De acordo com a figura 2, primeiramente é gerado o valor hash da mensagem. Os algoritmos de hash são utilizados para produzir a "impressão digital" dos dados, ou seja, identificadores de dados únicos e confiáveis. Esses algoritmos obtêm uma entrada de tamanho arbitrário e geram uma saída de tamanho fixo.

Ao valor do hash é aplicado um método de criptografia com uma chave privada pertencente ao assinante, o que é uma evidência de identidade. Assim é gerada a mensagem assinada ou simplesmente assinatura. Deve-se enviar uma cópia da mensagem original juntamente com a assinatura, para que elas possam ser comparadas no destino. Para garantir a confidencialidade da mensagem, deve-se codificá-la com um método de criptografia.

A assinatura é enviada pela rede juntamente com a mensagem codificada. No destino, acontece o processo inverso. A mensagem assinada é decodificada utilizando-se o mesmo método de criptografia da origem e a chave de decodificação, que pode ter sido enviada juntamente com a mensagem ou obtida através de um serviço de distribuição de chaves públicas. A mensagem codificada também sofrerá o processo inverso para obter-se a mensagem original. Um novo valor hash é gerado para a mensagem original e comparado ao que foi decodificado a partir da assinatura. Se os hashes forem iguais significa que o conteúdo da mensagem original não foi alterado e que o

possuidor da chave privada é o único que pode ter enviado a mensagem.

5.3 Certificação

Considerando a figura 2, caso os valores hashes sejam iguais, surge a seguinte uma questão: quem é o possuidor da chave privada? Uma mensagem só pode ser autenticada caso exista a certeza de que a chave pública recebida veio da entidade que assinou a mensagem. Tal garantia é obtida através de certificados e autoridades certificadoras.

Os certificados servem para documentar a associação de chaves públicas com entidades. São declarações digitalmente assinadas por um possuidor de chave privada autorizado, dizendo que a chave pública de uma determinada entidade é autêntica.

Para garantir a integridade do certificado, ele é assinado por uma autoridade certificadora, ou seja, uma entidade confiável, cuja chave pública é amplamente divulgada.

Quando implementadas juntamente com certificados, as assinaturas digitais garantem as propriedades de integridade, autenticação e não repudiação (Chin, 1999). Para garantir também a confidencialidade, a mensagem deve estar criptografada, como ilustrado na figura 2.

5.4 Autorização

Os mecanismos de autorização (Coulouris et. al, 1999) estão diretamente relacionados às propriedades de controle de acesso e disponibilidade de serviços, pois servem para garantir que o acesso a recursos só seja permitido ao conjunto de usuários autorizado a acessá-los.

É de responsabilidade da aplicação implementar mecanismos de autorização para acesso aos seus recursos. A SUIDS promove o controle de acesso através da Interface com o Banco de Dados e do Security Manager, melhor descritos na seção 7.

6 JAVA E OS MECANISMOS DE SEGURANÇA

Além de suas características serem favoráveis para o desenvolvimento de agentes de software (ver seção 4), a linguagem Java fornece APIs e ferramentas que facilitam a implementação dos mecanismos de segurança descritos na seção anterior. Os recursos que Java oferece para tornar aplicações seguras são apresentados a seguir.

6.1 Definição de uma Política de Segurança

A máquina virtual Java possibilita o desenvolvimento de programas seguros. Ela possui endereçamento automático de ponteiros, liberação automática de memória (Garbage Collection), controle de casting, verificação de byte codes e class loading. Entretanto, o aspecto de segurança de Java mais importante é de responsabilidade da classe SecurityManager, a qual promove controle de acesso aos recursos da plataforma Java (Zukowsky e Rohaly, 2000).

Sempre que uma aplicação tentar realizar uma operação que pode ser prejudicial ao sistema (escrever no sistema de arquivos, por exemplo), o SecurityManager irá verificar se aquela operação é permitida para a aplicação em questão. No modelo de segurança adotado em Java 2, o SecurityManager permite o estabelecimento de uma política de segurança, podendo restringir ou liberar recursos para uma determinada aplicação Java (Gong, 1998; Meloan, 1999).

Uma aplicação pode possuir um domínio de proteção, que é formado por um conjunto de permissões. Como ilustra a figura 3, é possível estabelecer uma política de segurança para a aplicação a ser executada. A política de segurança irá definir o domínio de proteção para uma aplicação. Assim, uma aplicação pode ter, por exemplo, acesso ao sistema de arquivos, mas não ter acesso à rede. Isso significa que seu domínio de proteção é composto apenas pelo sistema de arquivos.

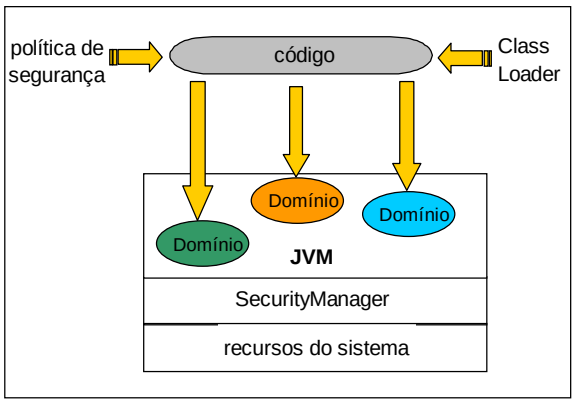

Figura 3 - Modelo de segurança de Java 2

Os domínios de proteção são definidos através de permissões. Uma permissão representa o acesso a um recurso do sistema. Então, se uma aplicação executando com SecurityManager quiser acessar um determinado recurso, a permissão correspondente deve ser explicitamente concedida a ele.

As permissões associadas a cada aplicação ou entidade são especificadas em arquivos especiais chamados policy files. Um policy file pode ser criado por um editor simples ou pela ferramenta gráfica policytool, incluída no JSDK 1.2 (Java Software Development Kit). Um policy file pode definir permissões associadas a uma entidade ou a uma determinada URL.

6.2 Recursos para Promover Assinaturas Digitais

Java 2 conta com uma API de segurança e com as ferramentas keytool e jarsigner (incluídas no JSDK 1.2) para promover assinaturas digitais. A API possibilita a assinatura de dados e sua verificação, enquanto as ferramentas possibilitam a assinatura e verificação de aplicações envolvendo certificados.

Utilizando-se a API em conjunto com as ferramentas, pode-se garantir a autenticidade do possuidor de uma aplicação (Dageforde, 2000).

A ferramenta keytool é utilizada para gerenciar chaves e certificados e armazená-los em keystores, que são arquivos onde a chave privada é protegida por senha.

Podem existir dois tipos de entrada em um keystore: para as chaves, o que inclui a chave privada e o certificado contendo a chave pública de uma determinada entidade, e para certificados confiáveis. Um certificado é dito confiável porque o possuidor do keystore confia na chave pública daquele certificado. Essa entrada é necessária caso o possuidor do keystore deseje receber dados/aplicações de outras entidades e autenticá-las. Os dois tipos de entrada do keystore estão associadas a um alias. É o alias que identifica o possuidor de chaves e certificados confiáveis no keystore.

Os certificados podem ser exportados e importados do keystore. Exportar um certificado significa extraí-lo do keystore para que ele possa ser enviado a uma entidade que precisará autenticar uma chave pública. A entidade que recebeu o certificado exportado deve, então, entrar em contato com o possuidor do certificado e verificar sua autenticidade para poder importá-lo, ou seja, adicioná-lo à lista de certificados confiáveis do keystore.

A ferramenta jarsigner gera assinaturas digitais para um arquivo JAR e as verifica.

O JAR é baseado no popular formato ZIP e é utilizado para compactar e agregar vários arquivos em um. Os arquivos que compõem um JAR podem ser assinados pelo autor de uma aplicação.

Ao criar-se um JAR é criado também um arquivo manifest. Esse arquivo consiste de várias seções, cada uma delas é uma entrada correspondente a um arquivo que compõe o JAR. É a partir dessas entradas que a assinatura digital será gerada.

Para que a assinatura possa ser gerada deve existir primeiramente uma chave privada e o certificado da chave pública. Jarsigner utiliza informações sobre chaves e certificados armazenados em um keystore para gerar a assinatura.

Um JAR assinado contém, entre outras coisas, o certificado extraído do keystore. Ele é exatamente igual ao original, a não ser por dois arquivos: o signature (.SF), e o signature block (.DSA).

O arquivo .SF é similar ao arquivo manifest. Ambos contêm uma seção para cada arquivo do JAR. Cada seção contém três linhas: o nome do arquivo, o nome do algoritmo utilizado para gerar o valor hash para aquele arquivo e o valor hash para o arquivo. A diferença é que, no manifest, o hash de cada arquivo é gerado a partir dos dados armazenados nele. No .SF o hash de cada arquivo é gerado a partir das três linhas do manifest. Esse esquema garante que o conteúdo dos arquivos do JAR não serão alterados.

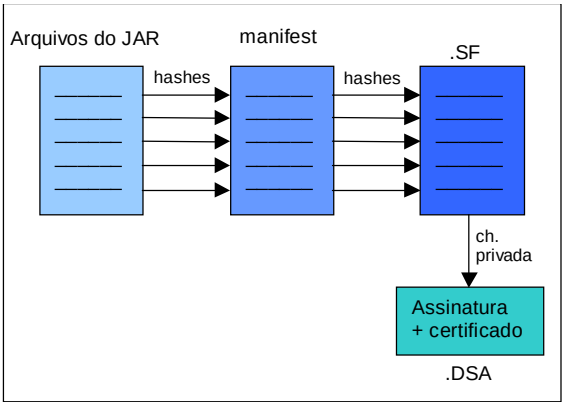

Figura 4 - Assinatura de um JAR

O arquivo .SF é assinado e a assinatura é armazenada no arquivo .DSA. Além disso, o .DSA contém o certificado codificado, para autenticar a chave pública no destino. A figura 4 ilustra o processo para gerar a assinatura digital em um JAR.

A verificação do JAR é feita com sucesso se a assinatura for válida e se nenhum dos arquivos que compõem o JAR foram alterados depois da assinatura ter sido gerada. A figura 5 ilustra o processo para verificação da assinatura digital de um JAR, que envolve os seguintes passos:

- Verificar a assinatura do arquivo .SF. Essa verificação assegura que a assinatura no arquivo .DSA foi gerada utilizando-se a chave privada correspondente à chave pública cujo certificado está armazenado no .DSA.
- Verificar se os hashes de cada entrada no .SF são iguais aos hashes da entrada correspondente no manifest.
- Ler cada arquivo do JAR e gerar novos hashes para eles. Se os hashes já existentes no arquivo manifest forem iguais aos novos valores, significa que os arquivos não foram modificados.

O processo de verificação da assinatura feito pela ferramenta jarsigner não inclui a verificação do certificado. Para isso, uma cópia do certificado já deve ter sido importada para ser comparada ao certificado que chegou com o JAR. A API de Java 2 permite que o certificado seja extraído do JAR e comparado com os certificados armazenados em um

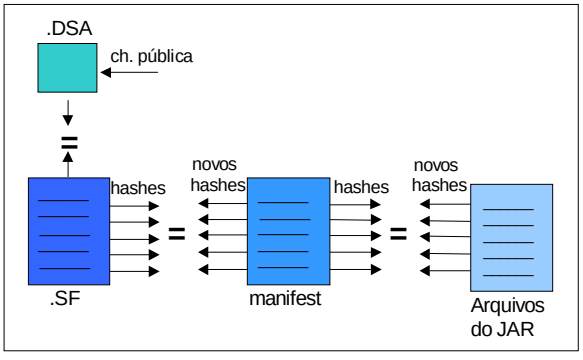

Figura 5 - Verificação da assinatura de um JAR

keystore, o que garante a autenticidade da chave pública.

É importante observar que a assinatura de um arquivo JAR não garante sua confidencialidade. Ele pode ser capturado em trânsito e o conteúdo de seus arquivos pode ser lido, porém não modificado. Uma solução é utilizar o Java Cryptography Extension (JCE), um pacote adicional que possui APIs para codificar e decodificar dados.

6.3 As Ferramentas Utilizadas em Conjunto

A figura 6 ilustra as ferramentas descritas sendo utilizadas para gerar uma aplicação Java assinada e autenticada por certificados.

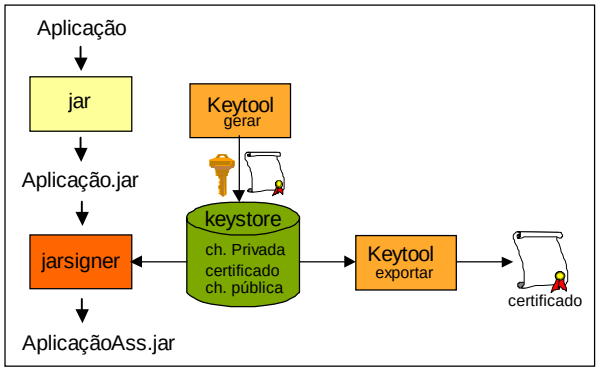

Figura 6 – Processo para Assinatura de Aplicações

De acordo com a figura 6, uma aplicação Java deve, primeiramente, ser embutida em um arquivo JAR. O par de chaves e o certificado são gerados com a utilização de keytool e armazenados em um keystore. Tendo as chaves e o JAR, utiliza-se jarsigner para assinar a aplicação. O certificado da chave pública pode ser exportado para qualquer entidade que desejar executar a aplicação.

A figura 7 ilustra o processo de autenticação da aplicação e sua execução de forma segura. Para isso, o certificado deve ter sido importado e aceito como válido. As permissões referentes ao assinante da aplicação devem estar definidas nos policy files. Ao executar a aplicação, as permissões nos policy files serão concedidas através da verificação do certificado já importado para o keystore.

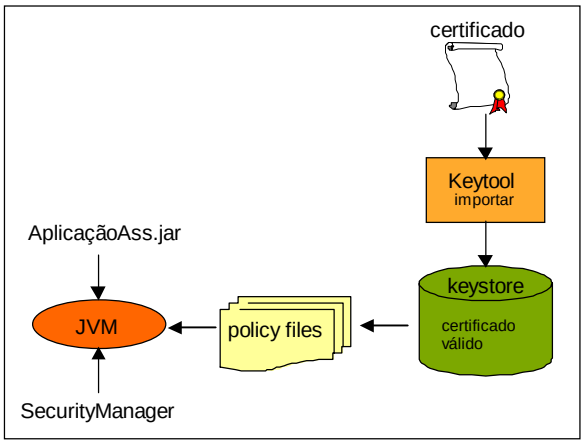

Figura 7 – Autenticação e Execução de Aplicações

As aplicações Java que desejarem utilizar a SIUDS deverão estar devidamente assinadas e seu certificado deve ter sido importado.

O agente Gateway da SIUDS irá verificar as assinaturas e os certificados através da API de segurança e das ferramentas. Se um agente for autenticado pelo Gateway, ele irá ser executado pelo Pool, que conta com uma política de segurança definida por policy files.

7 IMPLEMENTAÇÃO DA INTERFACE

A figura 1 apresenta uma visão geral da SIUDS. As seções seguintes têm por objetivo descrever com maiores detalhes cada um de seus módulos.

7.1 O Agente Gateway

Os agentes só poderão entrar no Pool se passarem pelo Gateway. Ele é devidamente assinado e, como todos os outros agentes, será controlado pelo Pool. Sua função é promover a entrada de outros agentes no Pool, verificar a assinatura digital desses agentes e se o certificado do possuidor do agente é confiável. Para cumprir a sua tarefa, o agente Gateway possui três classes principais: ServerFTP, JarVerifier e CertificateVerifier.

É através da classe ServerFPT que os agentes irão entrar em contato com o Gateway. A função dessa classe é transferir arquivos de extensão .jar (agentes) do cliente para o sistema de arquivos local.

Tendo recebido o agente, o controle passa para a classe JarVerifier. Sua função é verificar se o agente está devidamente assinado. A verificação é feita pela ferramenta Jarsigner, como descrito na seção 5.2. Caso a assinatura seja válida, resta verificar o certificado do assinante.

A classe CertificateVerifier irá extrair o certificado do agente. Tendo esse certificado, será extraído o alias dele, ou seja, a identificação de quem o assinou. Depois disso, o CertificateVerifier irá verificar se existe algum alias, igual ao extraído do certificado do agente, no keystore da SIUDS. Caso exista, o certificado do agente é comparado ao certificado armazenado no keystore para aquele alias. Se os dois forem iguais, significa que o certificado daquele assinante já foi importado para o keystore da SIUDS e aceito como válido. O Gateway pode então avisar ao Pool que há um agente pronto para entrar em execução.

7.2 O SecurityManager

Quando um agente entra em execução no Pool, ele pode ter acesso a todos os recursos que a plataforma Java pode oferecer. Ele pode, por exemplo, escrever em todo o sistema de arquivos local, se conectar a qualquer host, etc. Mas, não é seguro que todos os agentes do Pool tenham liberdade irrestrita Um agente poderia modificar a

política de segurança ou até mesmo atacar e danificar o sistema. É para evitar ações que possam comprometer a segurança do sistema que existe o SecurityManager.

SecurityManager é uma classe Java que permite a implantação de uma política de segurança. Para definir uma política de segurança deve-se conceder permissões em um ou mais policy files ou então implementar uma subclasse do SecurityManager, o que é mais suscetível a erros.

O protótipo da SIUDS utilizou um policy file para promover a segurança. Esse policy file possui duas entradas, uma associada ao grupo superusers e a outra associada ao grupo commonusers. Para os superusers foram dadas todas as permissões possíveis. Já os commonusers possuem permissões para ler e escrever em alguns arquivos e estabelecer conexões com alguns hosts.

A partir do momento em que o SecurityManager for inicializado, as permissões concedidas nos policy files passam a vigorar. Então, se um agente deseja, por exemplo, deletar um arquivo local, a classe SecurityManager, irá verificar nos policy files se o agente que está tentando deletar o arquivo possui permissão para isso. Caso o agente pertença ao grupo superusers, ou a qualquer outro grupo que possua permissão nos policy files para apagar arquivos, o SecurityManager deixará que o arquivo seja apagado.

7.3 O Pool de Agentes

O Pool de Agentes é um ambiente onde todos os agentes irão estar executando. Ele é responsável por colocar os agentes em execução e controlar o tempo de vida de cada um deles. Para realizar essas tarefas, o Pool de Agentes é composto de basicamente três classes principais: Pool, JarRunner e TimeCounter.

A classe Pool deve ser a primeira a ser chamada. Quando ela é executada, a primeira ação tomada é colocar o agente Gateway em execução.

A execução do Gateway e de qualquer outro agente é feita pela classe JarRunner, através do disparo de um thread. É a JarRunner que instala o SecurityManager. A classe JarRunner irá começar a execução de um agente logo depois que o Gateway terminar sua verificação.

A classe TimeCounter irá controlar o tempo de vida de cada agente. Ela é chamada por JarRunner quando um agente entra em execução. O tempo de vida de um agente pode variar de segundos a anos.

7.4 A Interface com o Banco de Dados

A Interface com o Banco de Dados consiste na declaração dos métodos que irão possibilitar aos agentes manipularem os objetos, os grupos, fazer associações entre eles para leitura e/ou escrita e obter informações sobre eles. A Interface com o Banco de Dados, não possui a implementação de nenhum método. Ela define o cabeçalho dos métodos, especificando os parâmetros que devem ser passados e o tipo que deve ser retornado para cada método. Os métodos podem ser classificados em três grupos, como mostra a tabela 1.

Para armazenar objetos em um servidor universal, pode-se utilizar dois tipos de SGBD: os SGBDOOs, como ObjectStore ou Jasmine ou então um SGBDOR, como Oracle ou Informix. Para utilizar a SIUDS com um SGBD, deve-se criar uma classe Java que implemente os métodos definidos na Interface com o Banco de Dados para o SGBD escolhido, o que pode ser feito via JDBC (Java Database Connectivity).

Toda a informação que a SIUDS necessita para executar é armazenada no próprio SGBD com o qual ela faz a interface. Isso garante que ela seja tão robusta quanto o SGBD.

No protótipo da SIUDS, o banco de dados foi armazenado em memória e foi criado o InfoAgent (ver seção 8) para a realização de testes.

Tabela 1 - Métodos definidos na Interface com o Banco de Dados

	Métodos
Grupos	void initialize()
	void createGroup(String groupName, long time)
	void deleteGroup(String groupName)
	String[] getRootsToRead (String groupName)
	String[] getRootsToWrite (String groupName)
	long getTime(String groupName)
	boolean isGroup(String groupName)
	String[] getGroups()
Objetos (ou Roots)	void createRoot(String rootName)
	Object get(String rootName)
	void deleteRoot(String rootName)
	void set(String rootName, Object root)
	void setGroupToRead(String rootName, String groupToRead)
	void setGroupToWrite(String rootName, String groupToWrite)
	String[] getRoots()
	boolean isRoot(String rootName)
	String[] getGroupsToRead (String rootName)
	String[] getGroupsToWrite (String rootName)
	boolean isReadable(String rootName, String groupName)
	boolean isWriteable(String rootName, String groupName)
Threads	void associate (String threadName, String groupName, String agentName)
	String getGroup (String threadName)
	String getAgentName (String threadName)
	void removeAssociation (String threadName)

8 INFOAGENT: UM AGENTE PARA TESTES

O InfoAgent é um agente que foi desenvolvido para ser executado pela SIUDS. Sua função é dar informações a respeito do estado da SIUDS em um

determinado instante. Ele abre uma porta TCP e fica a espera de conexões vindas a partir de um browser. Ao receber uma requisição, ele envia uma página HTML contendo informações sobre a SIUDS, como: grupos cadastrados, agentes que estão executando e a que grupo pertence cada agente em execução.

Para ser executado pelo Pool, o InfoAgent deve satisfazer a todos os requisitos necessários. Ele está embutido em um JAR e sua assinatura pertence ao grupo de superusers, o qual já está cadastrado no banco de dados e possui um certificado confiável no keystore da SIUDS. Como descrito na seção 6.2, o protótipo da SIUDS possui os grupos superusers e commonusers cadastrados e com permissões em um policyfile. O banco de dados do protótipo SIUDS está armazenado em memória, como descrito na seção 6.4.

Depois de devidamente assinado, o InfoAgent foi enviado para ser verificado pelo Gateway. A conexão com o Gateway foi estabelecida pela classe ClientFTP, descrita na seção 6.1. Depois de ter recebido o InfoAgent, o Gateway fez a verificação de sua assinatura digital utilizando a ferramenta jarsigner, como descrito na seção 6.1. O Gateway obteve sucesso nessa verificação, pois o conteúdo do InfoAgent não foi alterado em trânsito. Tendo terminado a verificação da assinatura, o Gateway extraiu o nome do assinante do InfoAgent e verificou no keystore da SIUDS se aquele possuía um certificado válido. O InfoAgent possui assinatura de superusers e o seu certificado já havia sido importado para o keystore da SIUDS. Assim, o Gateway terminou com sucesso a verificação do InfoAgent. Outros agentes foram enviados ao Geteway, apenas com o objetivo de testar a verificação de certificados. Notou-se que todos os agentes dos grupos superusers e commonusers foram verificados com sucesso, enquanto os agentes com assinantes não cadastrados no keystrore da SIUDS eram rejeitados pelo Gateway e, por conseqüência, não eram executados pelo Pool.

Depois de ter sido autenticado pelo Gateway, o InfoAgent foi executado pelo Pool. Como esperado, o InfoAgent ficou a espera de conexões na porta 82 e, ao receber requisições, ele enviava uma página HTML informando os grupos cadastrados, os agentes em execução naquele momento e seus respectivos grupos.

Os testes feitos com o Pool foram referentes a execução concorrente de agentes e controle de seus tempos de vida. O grupo de superusers tinha tempo ilimitado, mas os agentes dos communusers só poderiam estar em execução durante uma hora. O Pool obteve sucesso na execução concorrente de agentes e terminou o agente do grupo de communusers depois de uma hora de execução.

Os testes realizados com o SecurityManager também obtiveram sucesso. O SecurityManager agiu conforme as permissões concedidas para os superusers e commomusers no policyfile. De acordo com a seção 6.2, foram dadas todas as permissões para o grupo de superusers, enquanto os commonusers só poderiam ler e escrever em alguns arquivos e estabelecer conexões com os hosts especificados no policyfile. O InfoAgent pôde realizar qualquer ação, pois ele possui assinatura de superusers; mas o agente teste do grupo commonusers foi impedido, pelo SecurityManager, de escrever em arquivos para os quais ele não tinha permissão.

9 CONCLUSÕES

A SIUDS fornece um ambiente de execução aberto e ao mesmo tempo seguro para agentes de software. Eles podem armazenar e recuperar objetos de banco de dados e utilizar recursos do sistema hospedeiro para se conectarem a recursos externos, o que promove abertura no ambiente de execução. Tanto o acesso aos objetos do banco de dados quanto aos recursos do sistema hospedeiro são feitos de forma segura, ou seja, de acordo com as permissões associadas ao grupo ao qual o agente pertence.

A SIUDS foi desenvolvida em Java e possibilita que vários agentes possam estar executando, de forma concorrente, em um Pool de Agentes. Para garantir a segurança do Pool, três providências são tomadas:

- Verificação da autenticidade do agente, que é promovida pelo agente Gateway através de assinaturas digitais e certificados.
- Controle de acesso aos recursos do sistema, o que é feito pelo SecurityManager.
- Controle de acesso aos objetos do banco de dados, o que é garantido na implementação dos métodos definidos na Interface com o Banco de Dados.

A SIUDS, juntamente com um banco de dados orientado a objetos, provê um ambiente de execução onde agentes móveis podem ser carregados, executados e acessar dados, formando a base de um servidor de dados universal.

Nos servidores de dados universais, implementados utilizando a SIUDS, diferentes agentes de software podem servir o mesmo conjunto de dados nos mais variados formatos e protocolos de comunicação. Além disso, o servidor de dados poderá utilizar agentes desenvolvidos em Java como sua linguagem de consulta, uma forma mais completa e eficiente para realizar consultas em dados orientados a objetos do que as linguagens de consulta usuais.

AGRADECIMENTOS

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – pelo apoio financeiro a esse projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

CHIN, S.K. High-confidence design for security, *Communications of the ACM,* v. 42, n. 37, p. 33-37, 1999.

COULOURIS, G. et al. *Distributed systems: concepts and design.* Addison-Wesley, 1994.

DAGEFORDE, M. *The Java tutorial: trail - security in Java 2 SDK 1.2*. [online]. [18/03/2000]. Disponível na Internet: <http://web2.java.sun.com/docs/books/tutorial/security1.2/index.html>

DAVIS, J. R. Universal servers: the players, part 2, *DBMS*, v. 10, n. 8, p. 75-81, 1997.

FRANKLIN, S. and GRAESSER, A. Is it an agent or just a program?: a taxonomy for automonous agents, In: PROCEEDINGS OF THE THIRD INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURE AND LANGUAGES. Institute for Intelligent Systems, University of Memphis, 1996.

GONG, L. *Java security architecture*. October 1998. [online: 17/03/200]. Disponível na Internet: <http://www.java.sun.com/products/jdk/1.3/docs/guide/security/spec/security-spec.doc.html>

KRISHNAMURTHY, V. et al. Bringing Object-Relational Technology to the Mainstream. *ACM Sigmod Record*, v. 28, n. 2, p. 513, June 1999.

MAES, P. Artificial life meets entertainment: lifelike autonomous agents, *Communications of the ACM*. v. 38, n. 11, p. 108-114, 1995.

MELOAN, S. *Fine-grained security - The key to network safety: Strength in flexibility*. Nov. 1999. [online]. [19/11/1999]. Disponível na Internet: <www.java.sun>

WONG, D. et al. Java-based mobile agents, *Communications of the ACM*, v. 42, n. 3, p. 92-102, 1999.

ZUKOWSKY, J. and ROHALY, T. *Fundamentals of Java security*. [online]. [26/02/2000]. Disponível na Interrnet: <http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/abstract.html>