Módulo 7

Segurança

Lição 2

Sandbox

Versão 1.0 - Jan/2008

Autor
Aldwin Lee
Cheryl Lorica

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Angelo de Oliveira
Bruno da Silva Bonfim
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Guilherme da Silveira Elias
Leandro Souza de Jesus
Lucas Vinícius Bibiano Thomé
Luiz Fernandes de Oliveira Junior
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach e Vinícius G. Ribeiro (Especialista em Segurança)
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Sempre que é discutida a segurança Java, normalmente ela é focada no modelo de *Sandbox* (caixa de areia). Este modelo permite ao usuário final configurar restrições e permissões para um programa.

Tradicionalmente, executamos segurança nos computadores baseados em um sistema de confiança. Antes de executar um sistema, deveríamos ter a confiança no código-fonte deste. Uma vez que a aplicação tenha passados por esta checagem, possui mandato no sistema em que está sendo executada. Se por qualquer motivo o programa possui uma intenção maliciosa, poderá destruir o sistema sem problemas, visto que não existem restrições ao programa. Expondo isto de forma simples, para proteger o computador devemos restringir o que fazer em primeiro lugar.

Ao final desta lição, o estudante será capaz de:

- Identificar o modelo de segurança padrão empregado – *Sandbox*
- Conhecer os componentes da *Sandbox*
- Realizar as configurações dos componentes da *Sandbox*
- Definir os domínios de proteção de sua aplicação
- Aplicar a política de segurança, por intermédio das permissões
- Entender como as classes podem ser assinadas (certificação digital)

2. Modelo Sandbox

O modelo *Sandbox* torna fácil proteger o computador de programas carregados a partir de fontes não confiáveis. Em lugar de obter a confiança no recurso, a segurança é alcançada permitindo que qualquer programa seja executado entretanto, restringindo suas ações que poderia danificar seu computador. A vantagem é que invés de restringir ou checar qualquer coisa que entra em seu computador, a *Sandbox* permite que qualquer programa seja executado e previne que este realizar qualquer ação que poderia danificar o sistema.

O modelo de segurança Java gira ao redor desta idéia de uma *Sandbox*. Gira ao redor da idéia de que quando é permitido que um programa seja executado no computador, deveríamos ser capazes de fornecer um ambiente onde o programa pode fazer o que foi projetado para fazer e ao mesmo tempo permite a habilidade de restringir acesso aos recursos que ele não deveria acessar.

Há diferentes tamanhos de *Sandbox* tipicamente configuradas para um programa Java. Estes são:

- **Mínima** – Onde um programa tem acesso à CPU, possui sua própria memória, bem como acesso para dispositivos de interface humana (isto é, Monitor, Teclado, Mouse)
- **Padrão** – Similar a **mínima**, exceto que também fornece acesso ao servidor WEB da qual o programa Java foi carregado
- **Restrita** – Similar a **padrão**, exceto que também fornece acesso para certos, mas não todos, recursos do sistema operacional
- **Aberta** – Esta permite ao programa acessar qualquer recurso do sistema anfitrião

O modelo de segurança Java envolve todo aspecto de sua arquitetura. Para ser capaz de assegurar que este está corretamente no lugar em que necessitamos olhar para diferentes partes da arquitetura, bem como entender como estas tecnologias funcionam juntas.

Os seguintes componentes são os fundamentos do Modelo de Segurança Java:

- Características de segurança construídas na JVM (Máquina virtual Java)
- A arquitetura do carregador de classes
- O verificador de arquivos de classe
- O gerenciador de segurança e a API Java

Uma força chave do modelo da *Sandbox* Java é sua customização. Da lista acima, o carregador de classes e o gerenciador de segurança são completamente customizáveis. Para definir sua própria segurança, você define sua própria implementação ou subclasses de java.lang.SecurityManager. Com sua implementação do gerenciador de segurança, você define seus próprios métodos para controlar o acesso a um determinado recurso - como escrever em um arquivo, por exemplo. Adicionalmente, é recomendado criar um gerenciador de segurança quando criamos um carregador para instanciar classes a partir de recursos não confiáveis.

3. Componentes Principais de uma Sandbox

Três componentes da *Sandbox* funcionam juntos para assegurar a segurança Java quando o código é carregado. O gerenciador de segurança reforça os limites de uma *Sandbox* e restringe as ações que são válidas.

3.1. Carregador de Classes

Em Java, o carregador de classes carrega os *bytecodes* a partir das classes compiladas, força limites de espaços de nome. Isto controla partes da JVM que o código pode acessar.

Classes carregadas a partir do sistema de arquivos local tem seu próprio nome de espaço. Adicionalmente, um nome de espaço para cada fonte de rede também é definido. Isto assegura e protege a JVM do conflito provocado por classes com o mesmo nome.

3.2. Verificador de Bytecodes

Uma vez que o carregador de classes carregou uma classe, chama o verificador de *bytecodes*. A tarefa principal deste é conferir o código para ver se o mesmo está de acordo com a especificação da linguagem de programação Java e procurar por quaisquer das seguintes violações:

- Regras da linguagem de Programação Java
- Restrições de nome de espaço
- Estouros de pilha (*overflows* e *underflows*)
- Conversão ilegal de tipos de dados

O verificador de bytecodes pode provar o seguinte:

- O arquivo de classe tem o formato correto
- Classes finais não possuem subclasses e métodos finais não estão realizando polimorfismo por override
- Cada classe possui uma única superclasse
- Não existe conversão ilegal de atributos para tipos primitivos
- Não existe conversão ilegal em objetos

3.3. Gerenciador de Segurança

O propósito do gerenciador de segurança é determinar quais operações uma classe tem permissão para executar. De forma simples, o gerenciador de segurança é responsável por determinar a maioria dos parâmetros de uma *Sandbox* Java. Se um sistema em Java tenta escrever em um arquivo ou conectar-se a um recurso de rede, primeiro deve obter permissão do gerenciador de segurança. Além disso, quando um aplicativo deseja modificar o estado dos serviços, o gerenciador de segurança controla tais operações se as julgar perigosas.

Por padrão, um gerenciador de segurança não é usado quando um programa está sendo executado. Para habilitar uma aplicação Java a usá-lo, deve ser especificado:

```
-D java.security.manager
```

Para applets e plug-ins Java, o gerenciador de segurança é carregado automaticamente. Para reforçar: aplicações Java não tem um gerenciador de segurança instalado por padrão, enquanto *Applets* Java possuem um gerenciador de segurança muito estrito para proteger recursos locais.

4. Elementos da Sandbox Java

Em termos de administração de segurança, Java possui os seguintes elementos:

4.1. Permissões

Cada classe tem um conjunto de permissões que define o que está autorizada a executar. A *Sandbox* Java é então definida com base nestas permissões. Quando uma ação é executada por uma classe, as permissões são checadas pela máquina virtual. Se determinada classe não deveria ter a permissão para executar uma ação em particular, uma exceção é lançada e a operação é bloqueada.

Classes do núcleo da API Java sempre possuem permissão para executar qualquer ação. Quaisquer outras classes, mesmo que definidas na variável *classpath*, devem ter permissão para executar qualquer ação sensível. Estas permissões são definidas em arquivos de política de segurança que são administrados pelos usuários finais e são usados pela *Sandbox* para gerenciar os arquivos de política de segurança.

Permissões empregadas pela máquina virtual são baseadas em um sistema de classes. Isto significa que qualquer aplicação ou API pode definir suas próprias permissões e assegurar que os usuários ou administradores possuem as permissões das APIs antes de executar. A próxima lista descreve as permissões padrões usadas pelo núcleo de classes Java:

- **java.io.FilePermission**
  - Ações: ler, escrever, apagar e executar
- **java.net.SocketPermission**
  - Ações: aceitar, escutar, conectar e resolver
- **java.util.PropertyPermission**
  - Ações: ler e escrever
- **java.lang.RuntimePermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.awt.AWTPermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.net.NetPermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.security.SecurityPermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.io.SerializablePermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.lang.reflect.ReflectPermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução
- **java.security.AllPermission**
  - Ações: nenhuma, as classes tem ou não permissão para executar uma operação em tempo execução

4.2. Code Sources

São localizações que indicam de onde as classes serão carregadas. Pode incluir a URL das classes bem como quem as assinou. É importante notar que ambos são opcionais. A URL da classe pode ser o um arquivo de URL do sistema ou da rede.

Pode-se associar permissões baseadas na URL a partir da qual foi carregada ou baseada somente em quem a assinou. Também é possível assinalar permissões de uma combinação de URL e do assinador. Esta URL dentro do código fonte também é conhecida como *codebase*.

Code Sources são uma combinação de *codebase* e assinante. O campo assinante deve coincidir com o pseudônimo armazenado na chave *keystore*. *Codebases* podem ser quaisquer URL válidas, e como tais, usam barras "/" como separadores de níveis ou diretórios, mesmo que estejam no sistema de arquivos local.

O final da URL também tem um impacto na definição. Existem quatro casos:

- URL especifica um arquivo .jar – Somente classes dentro daquele arquivo jar são parte do *codebase*
- URL termina com uma barra – Somente arquivos .*class* no diretório são parte do *codebase*. Arquivos não estão inclusos
- URL termina com um asterisco – Todos os arquivos .*jar* e .*class* no diretório pertencem ao *codebase*
- URL termina com um hífen – Todos os arquivos .*jar* e .*class* pertencem ao diretório e todos os subdiretórios pertencem ao *codebase*

Note que a estrutura de um diretório não é afetada pelo nome do pacote. Por exemplo, para carregar a classe *up.jedi.Login*, não é necessário adicionar o diretório acima onde aparece. Por exemplo, *jedi.upd.edu.ph/*, e não *jedi.upd.edu.ph/up/jedi/*.

5. Domínios de Proteção

Um domínio de proteção é uma associação de permissões com um *Code Source* em particular. Domínios de proteção são o conceito básico para uma *Sandbox* padrão. Informam como o código carregado de um local em particular como por exemplo, *www.sun.com* tem a permissão de escrever em um arquivo e possui o código assinado por quem tem a permissão para iniciar os trabalhos de impressão.

5.1. KeyStores

Classes Java podem ser assinadas através do uso de certificados digitais juntamente com a ferramenta *jarsigner*. Deste modo, temos a possibilidade de conceder permissões para código assinado por uma determinada entidade.

O código assinado pode ser manipulado através do uso de uma *keystore*. A *keystore* é basicamente onde é armazenado os certificados, de chave pública, que são usados para assinar o código utilizado. Antes de executar qualquer código assinado, a chave pública é usada para assiná-la dever ser obtida e então instalada na *keystore* do sistema. Alguns sistemas, como os navegadores, aceitam certificados de chave pública quando os arquivos são carregados pela primeira vez, sendo então assinados mas, usualmente, estes certificados devem ser descarregados e instalados antes de se executar um aplicativo.

Keystores são administradas através do utilitário *keytool* fornecido na distribuição padrão Java. Por padrão, a *keystore* é localizada em um arquivo chamado *.keystore* encontrada no diretório *home* do usuário. Quando um certificado de chave pública é instalado em uma *keystore*, o administrador insere no certificado um nome ou um apelido que é usado para uma futura referência. Este apelido é usado na administração dos vários certificados e também é o nome usado no arquivo de política de segurança.

5.2. Arquivos de Política de Segurança

A administração da *Sandbox* Java é feita listando-se várias permissões no arquivo de política de segurança Java. A JVM pode manipular múltiplos arquivos de política de segurança em várias localizações. Apesar disso, por padrão, usa dois arquivos de política de segurança específicos. O arquivo de política de segurança global *java.policy*, encontrado em **$JREHOME/lib/security/**, é usado por todas as instâncias de uma JVM no sistema. É considerado um arquivo de política de segurança global porque é compartilhado por todas as instâncias da JVM a partir deste JRE particular.

Em adição ao arquivo de política de segurança global, um usuário específico chamado *.java.policy* também pode ser encontrado no diretório do usuário. As permissões definidas neste arquivo mais aquelas no arquivo de política de segurança global definem as permissões dadas para um programa.

Arquivos de política de segurança são apenas arquivos de texto simples. Ferramentas como *policytool* podem ser usadas para administrá-los. Estes arquivos de política de segurança também são usados com a JAAS (Java Authentication and Authorization System) e, devido às nuances de sintaxe, é recomendado editá-los manualmente.

Na maioria dos casos, o código carregado advém de uma única localização. Há também casos onde existem múltiplas *Code Sources* para uma simples localização, tal como, utilizar a RMI. Para casos como esse, as permissões dando acesso a um recurso em particular é sempre a interseção de permissões e de todas as permissões concedidas em todos os *codebases*.

Permissões concedidas para uma única *Code Source* são concedidas como a união de todas as permissões dos arquivos de política usados pela JVM para todas as *Code Sources* relacionadas, bem como os assinantes, enquanto que para uma aplicação remota as permissões são determinadas pela interseção de todas as permissões em todas as *Code Sources* ativas.

5.2.1. A Ferramenta *Policytool*

É um software gráfico usado para gerenciar o arquivo *java.policy*. Possui um argumento na linha de comando

```
-file nomeDoArquivo
```

Que se informa o nome do arquivo de política de segurança que se deseja editar. Este argumento não define o carregamento do arquivo *$HOME/.java.policy* no diretório do usuário; se este arquivo não existir, por padrão, nenhum arquivo é carregado.

5.2.2. Permissões além dos Arquivos de Política de Segurança

A maioria das permissões dadas para um conjunto de código vem dos arquivos de política de segurança. Entretanto, algumas aplicações tem garantia de permissão para o código que elas carregam, e os carregadores de classe padrão Java concedem permissão adicional para classes que elas carregam.

Classes carregadas a partir do sistema de arquivos têm permissão para ler arquivos no diretório e subdiretórios a partir dos quais elas foram carregadas. Classes carregadas a partir de um recurso de rede através de HTTP têm, por padrão, concessão para conectar-se de volta com o *host* de onde elas foram carregadas; elas também têm permissão para aceitar conexões a partir daquele *host*.

5.2.3. O arquivo java.policy

Por padrão, usuários não têm acesso ao arquivo *java.policy* em seu diretório *home*, o que significa que o conjunto padrão de permissões para todos os programas Java sendo executados na JVM, são definidos no arquivo encontrado em *$JREHOME/lib/security/java.policy*.

Este é o conteúdo do arquivo java.policy para Java 5:

```
keystore "${user.home}${/}.keystore";

// As extensões padrões obtém as permissões por default
grant codeBase "file:${{java.ext.dirs}}/*" {
      permission java.security.AllPermission;
};

// Padrão de permissões garantidas para todos os domínios
grant {
  permission java.lang.RuntimePermission "stopThread";
  permission java.net.SocketPermission "localhost:1024-", "listen";
  permission java.util.PropertyPermission "java.version", "read";
  permission java.util.PropertyPermission "java.vendor", "read";
  permission java.util.PropertyPermission "java.vendor.url", "read";
  permission java.util.PropertyPermission "java.class.version", "read";
  permission java.util.PropertyPermission "os.name", "read";
  permission java.util.PropertyPermission "os.version", "read";
  permission java.util.PropertyPermission "os.arch", "read";
  permission java.util.PropertyPermission "file.separator", "read";
  permission java.util.PropertyPermission "path.separator", "read";
  permission java.util.PropertyPermission "line.separator", "read";
  permission java.util.PropertyPermission "java.specification.version",
    "read";
  permission java.util.PropertyPermission "java.specification.vendor",
    "read";
  permission java.util.PropertyPermission "java.specification.name", "read";
  permission java.util.PropertyPermission "java.vm.specification.version",
    "read";
  permission java.util.PropertyPermission "java.vm.specification.vendor",
    "read";
  permission java.util.PropertyPermission "java.vm.specification.name",
    "read";
  permission java.util.PropertyPermission "java.vm.version", "read";
  permission java.util.PropertyPermission "java.vm.vendor", "read";
```

```
  permission java.util.PropertyPermission "java.vm.name", "read";
};
```

A primeira linha define *keystores* inclusas no arquivo *$USER/.keystore*, onde verifica os certificados de entidades que possuem código assinado. As próximas seções definem permissões para certas propriedades Java.

Por padrão, as classes têm permissão para chamar *Thread.stop*(), escutar uma porta não privilegiada e umas poucas propriedades do sistema. É importante notar que, enquanto a aplicação tem a permissão para escutar as conexões de *socket*, não lhes é permitida as aceitas, por padrão.

5.2.4. O arquivo java.security

Por padrão, a *Sandbox* é controlada por meio dos dados do arquivo **java.security** encontrado em *$JREHOME/lib/security/java.security/*, que pode ser editado por administradores de sistema.

Para definir um novo arquivo de política de segurança, o definimos como:

```
policy.url.n = url
```

Onde *n* é um número. Por exemplo, os arquivos de política padrão são definidos como:

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

Outra entrada no arquivo *java.security* é *policy.expandProperties*, permite substituições para outras propriedades para o tornar mais portável.

Também é possível aos usuários definir seu próprio arquivo de política de segurança por ocasião da execução de uma aplicação através de linha de comando. Para restringir este comportamento, ajustamos a propriedade *policy.AllowSystemProperty* para *false*. Isso determina uma *Sandbox* com permissões e propriedades padrões que não podem ser modificadas pelos usuários finais.

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.